THESIS DISSERTATION

Cybersecurity Evaluation Methodology based on Metrics for Industrial Embedded Systems

Author:
Ángel LONGUEIRA-ROMERO

Advisors:
Dr. Iñaki GARITANO
Dr. Rosa IGLESIAS

Applied Engineering PhD Program
Electronics and Computing Department
Faculty of Engineering
Mondragon Unibertsitatea

November 22, 2022

Declaration of Authorship

I, Ángel LONGUEIRA-ROMERO, declare that this thesis titled, "Cybersecurity Evaluation Methodology based on Metrics for Industrial Embedded Systems" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Aurrera begiratzen ez duena, atzean dago."

Those who don't look forward, stay behind.

Traditional basque proverb

Abstract

MONDRAGON UNIBERTSITATEA

Faculty of Engineering

Electronics and Computer Department

Doctor of Philosophy

Cybersecurity Evaluation Methodology based on Metrics for Industrial Embedded Systems

by Ángel LONGUEIRA-ROMERO

During the last decades, Embedded Systems (ESs) have evolved from isolated systems into a fully connected devices. Today, due to the development of processing capabilities, ESs have the ability to control or manage a wide range of systems or applications, including the critical infrastructures.

Historically, the development of ESs has been focused on functionality rather than security, and today this still applies in many sectors and applications. Moreover, there is an increasing number of security threats over ESs, and a successful attack could have severe economical or environmental consequences, including the loss of human lives.

A standardized and general accepted security testing framework is needed to provide guidance, common reporting forms and the possibility to compare the results along the time. This can be achieved by introducing security metrics into the evaluation or assessment process. If carefully designed and chosen, metrics could provide a quantitative, repeatable and reproducible value that would reflect the security level of the ES.

This dissertation is divided into three main topics, each one specialized in a different task of security evaluation: (1) Analysis of existing security metrics, (2) Vulnerability analysis, and (3) Aggregation of security metrics.

In the first section of the thesis, more than 500 metrics were collected and analyzed. The current issues of existing metrics were also extracted. As expected, the 77.5% of them is related exclusively to software, and only the 0.6% of them addresses exclusively hardware security. We also surveyed the literature to find the which are the desirable properties of a comprehensive metric. Based on these data, we proposed a new taxonomy to classify security metrics based on the properties of ESs.

In the second section, we present a model to analyze the known vulnerabilities of industrial components over time. The proposed Extended Dependency Graph (EDG) model is based on two main elements: a directed graph-based representation of the internal structure of the component, and a set of quantitative metrics based on the Common Vulnerability Scoring System (CVSS). The EDG model can be applied

throughout the entire lifespan of a device to track vulnerabilities, identify new requirements, root causes, and test cases. It also helps prioritize patching activities. The model was validated by application to the OpenPLC project.

Finally, in the third section we propose a CVSS aggregation algorithm that integrates information about the functionality disruption of the SUT, exploitation difficulty, existence of exploits, and the context where the SUT operates. The aggregation algorithm was applied to OpenPLC V3, showing that it is capable of filtering out vulnerabilities that cannot be exploited in the real conditions of deployment of the particular system.

This work aims to lay the foundations for constructing a security evaluation methodology that uses standardized metrics to quantify the security level of an ES.

Laburpena

MONDRAGON UNIBERTSITATEA

Ingeniaritza Goi Eskola Politeknikoa

Elektronika eta Informatikako Saila

Doktorego Programa

Sistema Txertatu Industrialetarako Metriketan oinarritutako Zibersegurtasuna Ebaluatzeko Metodologia

by Ángel LONGUEIRA-ROMERO

Azken hamarkadetan, Sistema Txertatuak (ES) sistema isolatuak izatetik erabat konektatutako gailuak izatera igaro dira. Gainera, prozesatzeko gaitasuna handitzeari esker, ESek sistema eta aplikazio ugari kontrolatzen eta kudeatzen dituzte, kritikoenak barne.

Historikoki, ESen garapena funtzionaltasunean zentratu da segurtasunean baino gehiago, eta gaur egun ere horrela izaten jarraitzen du sektore eta aplikazio askotan. Horrek, ESei eragiten dieten mehatxuen kopurua handitzearekin batera, eraso arrakastatsu batek ondorio ekonomiko larriak izatea errazten du, baita giza bizitzak galtzea ere.

Orientazioa, txosten estandarizatuak eta emaitzak denboran zehar alderatzeko aukera emateko, beharrezkoa da alderdi horiek guztiak bateratuko dituen eta ebaluazio-prozesuan segurtasun-metrikak integratuko dituen metodologia bat. Kontu handiz diseinatzen eta aukeratzen badira, metrikek balio kuantitatiboa, errepikagarria eta erreproduzigarria eman lezakete, ESen segurtasun-maila islatuko lukeena.

Tesi hau hiru bloke nagusitan banatzen da, bakoitza zibersegurtasuna ebaluatzeko prozesuko zeregin batean espezializatua: (1) dauden segurtasun-metriken azterketa, (2) kalteberatasunen analisia, eta (3) Segurtasun-metriken agregazioa.

Tesi honen lehen atalean, 500 metrika baino gehiago bildu eta aztertu ziren, eta gaur egun metrikek dituzten arazoak atera ziren. Espero zen bezala, horien 77,5% softwarearekin soilik lotuta daude, eta 0,6% baino ez da hardware segurtasunari buruzkoa. Eskura dagoen literatura ere aztertu zen, metrika sakonean zer propietate nahi diren hautemateko. Datu horietatik guztietatik abiatuta, taxonomia berri bat garatu zen, segurtasun-metrikak ESen barne-egitura kontuan hartuta sailkatzen dituena.

Bigarren atalean, industria-osagaiek denboran zehar izan dituzten kalteberatasunak aztertzeko eredu bat aurkezten dugu. Proposatutako Extended Dependency Graph (EDG) eredua bi elementu nagusitan oinarritzen da: (2) System Under Test (SUT) delakoaren mendekotasunen barne-egituraren irudikapena, zuzendutako grafoetan oinarrituta; eta Common Vulnerability Scoring System (CVSS) delakoan oinarritutako metrika kuantitatiboen multzo bat. EDG eredua gailu baten balio-bizitza osoan aplika

daiteke, ahultasunen jarraipena egiteko, baldintza berriak identifikatzeko, urrakortasunen funtsezko kausak identifikatzeko eta test-kasu berriak sortzeko. Adabakiak aplikatzeko jarduerak lehenesten ere laguntzen du. Eredua OpenPLC proiektuan aplikatuta baliozkotu zen.

Azkenik, hirugarren atalean, GZEKren balioak gehitzeko algoritmo bat proposatzen da. Algoritmo horrek GLSren testuinguruko hainbat faktoreri buruzko informazioa biltzen du, hala nola funtzionaltasuna eteteari, ustiatzeko zailtasunari, esploiten existentziari eta SUTak jarduten duen testuinguruari buruzkoa. Agregazio-algoritmoa OpenPLC V3-ri aplikatu zitzaion, sistemaren hedapen-baldintza errealetan ustiatu ezin diren kalteberatasunak iragazteko gai dela frogatuz.

Lan honen bidez, IH baten segurtasun-maila kuantifikatzeko metrika estandarizatuak integratuko dituen zibersegurtasuna ebaluatzeko metodologia garatzeko oinarriak ezarri nahi dira.

Resumen

MONDRAGON UNIBERTSITATEA

Escuela Politécnica Superior de Ingeniería

Departamento de Electrónica e Informática

Programa de Doctorado

Metodología de Evaluación de la Ciberseguridad basada en Métricas para Sistemas Embebidos Industriales

by Ángel LONGUEIRA-ROMERO

En las últimas décadas, los Sistemas Empotrados (ES) han pasado de ser sistemas aislados a ser dispositivos totalmente conectados. Además, gracias al aumento en su capacidad de procesamiento, los ESs controlan y gestionan una amplia gama de sistemas y aplicaciones, incluidas aquellas más críticas.

Históricamente, el desarrollo de los ESs se ha centrado en la funcionalidad más que en la seguridad, y a día de hoy, esto sigue siendo así en muchos sectores y aplicaciones. Esto, sumado al aumento del número de amenazas que afectan a los ESs, facilita que un ataque exitoso pudiera tener graves consecuencias económicas, e incluso la pérdida de vidas humanas.

Para proporcionar orientación, informes estandarizados, y la posibilidad de comparar los resultados a lo largo del tiempo, se hace necesario una metodología que unifique todos estos aspectos, y que integre métricas de seguridad en el proceso de evaluación. Si se diseñan y eligen cuidadosamente, las métricas podrían proporcionar un valor cuantitativo, repetible y reproducible que reflejaría el nivel de seguridad de los ESs.

Esta tesis se divide en tres bloques principales, cada uno de ellos especializado en una tarea distinta del proceso de evaluación de la ciberseguridad: (1) Análisis de las métricas de seguridad existentes, (2) Análisis de vulnerabilidades, y (3) Agregación de las métricas de seguridad.

En la primera sección de esta tesis, se recopilaron y analizaron más de 500 métricas, y se extrajeron los problemas actuales a los que se enfrentan las métricas actualmente. Como era de esperar, el $77,5\%$ de ellas están relacionadas exclusivamente con el software, y sólo el $0,6\%$ de ellas aborda exclusivamente la seguridad hardware. También se analizó la literatura disponible para detectar cuáles son las propiedades deseables en métrica exhaustiva. A partir de todos estos datos, se desarrolló una nueva taxonomía que clasifica las métricas de seguridad teniendo en cuenta la estructura interna de los ESs.

En la segunda sección, presentamos un modelo para analizar las vulnerabilidades conocidas de los componentes industriales a lo largo del tiempo. El modelo Extended Dependency Graph (EDG) propuesto se basa en dos elementos principales: (2) La representación de la estructura interna de dependencias del System Under Test (SUT) basada en grafos dirigidos; y un conjunto de métricas cuantitativas basadas en el

Common Vulnerability Scoring System (CVSS). El modelo EDG puede aplicarse a lo largo de toda la vida útil de un dispositivo para hacer un seguimiento de las vulnerabilidades, identificar nuevos requisitos, identificar las causas fundamentales de las vulnerabilidades, y generar nuevos casos de test. También ayuda a priorizar las actividades de aplicación de parches. El modelo se validó mediante su aplicación al proyecto OpenPLC.

Por último, en la tercera sección se propone un algoritmo de agregación de valores CVSS que integra información sobre varios factores de contexto del SUT, como la interrupción de la funcionalidad, la dificultad de explotación, la existencia de exploits y el contexto en el que opera el SUT. El algoritmo de agregación se aplicó a OpenPLC V3, demostrando que es capaz de filtrar las vulnerabilidades que no pueden ser explotadas en las condiciones reales de despliegue del sistema.

Este trabajo pretende sentar las bases para el desarrollo de una metodología de evaluación de la ciberseguridad que integre métricas estandarizadas para cuantificar el nivel de seguridad de un ES.

Acknowledgements

This achievement would not be possible without the support, patience, and pieces of advice of many people. People that might not be directly involved in this work at first, but without them, all this work would not have turned into reality.

I want to start thanking my advisors, Iñaki Garitano and Rosa Iglesias. I am grateful for their support, encouragement, time, and knowledge. They managed to guide me during the difficult times, and celebrate each little achievement that was made. Moreover, they taught me interpersonal skills that I would not be able to learn in any other situation. A PhD. is not only about acquiring technical knowledge, but also about learning how to deal with people, frustration, and uncertainty. This is about knowing yourself, and improving every day.

As it could not be otherwise, I also want to thank Jose Luis Flores for giving me his most valuable resources: his time, and his experience. The quality of this work would not be the same without his advice.

I cannot forget my parents, that pushed me forward and always encouraged me to do my best. They mean a world to me. Thanks to them, I am able to stand where I am now. Leaving home to follow your own way is not an easy decision, but their support was and still is invaluable. Thank you both.

As important and my family, my chosen family is a core piece in my life. My friends, my colleagues at Ikerlan, my friends that were also working on their thesis. They were with me to help me to deal with every problem, failure, or issue. They were also always there to celebrate with me every achievement. And of course, I cannot forget my partner, the biggest serendipity I have ever lived. He had to see bits of me that I did not know even existed. Nevertheless, he managed to guide me, and turned bad times into good moments and memories. Every weekend was a new adventure to disconnect from the hard work of the week.

And finally, I want to thank Euskal Herria, because since I came here, everyone was really charming. I have never felt so accepted and integrated. I learned a new language, I was immersed in a new culture, and I also met new people. Mila esker bihotz bihotzez denagatik.

Contents

List of Figures

List of Tables

List of Abbreviations

CAPEC	**C**ommon **A**ttack **P**attern **E**numeration and **C**lassification
CC	**C**ommon **C**riteria
CPS	**C**yber-**P**hysical **S**ystem
CVSS	**C**ommon-**V**ulnerability **S**coring **S**ystem
CVE	**C**ommon-**V**ulnerabilities and **E**xposures
CWE	**C**ommon-**W**eakness **E**numeration
ES	**E**mbedded **S**ystem
SM	**S**ecurity **M**etric
ECSO	**E**uropean **C**yber **S**ecurity **O**rganisation
ICS	**I**ndustrial **C**ontrol **S**ystem
IoT	**I**nternet **o**f **T**hings
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
OSSTMM	**O**pen **S**ource **S**ecurity **T**esting **M**ethodology **M**anual
OWASP	**O**pen **W**eb **A**pplication **S**ecurity **P**roject
PC	**P**ersonal **C**omputer
SUT	**S**ystem **U**nder **T**est

Chapter 1

Introduction

Since the First Industrial Revolution in the 17th century, the society we live in has evolved along with the advances brought about by industry. Today, we live in the fourth industrial revolution, or Industry 4.0, which is characterized by trends such as automation and data exchange, including technologies like Cyber-Physical Systems (CPS), Internet of Things (IoT), and edge computing. Unfortunately, quite often, technological advances of each industrial revolution bring some drawbacks which need to be addressed. All the advances introduced so far have one issue in common that was not a key aspect until now: cybersecurity.

In this chapter, the critical role of current CPS is highlighted. Second, we describe the features that are more critical to the security of CPS to set the motivation of this thesis. Then, we put forward the hypotheses that this work uses as a basis. Finally, we review the publication associated with this research and the structure of this dissertation.

1.1 Motivation and Objective of the Research

Embedded Systems (ES) evolved from fully analog devices to digital devices with the Third Industrial Revolution in the late 20th century, when the industry started to adopt digital computers. With this digital revolution, and the proliferation of digital record-keeping, data became an even more critical asset, and so did security. Nevertheless, the breakthrough for ESs came later, in the 21st century. The so-called Fourth Industrial Revolution (or Industry 4.0) introduced interconnectivity and smart automation to the landscape of the existing industries. With these new trends, ESs evolved again to integrate all these features, serving as a starting point for developing other technologies (*e.g.*, the Internet of Things). The new connectivity capabilities of ESs increase their exposures, and therefore, their attack surface, making them a promising target for attackers.

1.1.1 Motivation

ESs, and more broadly industrial components, are the driving force of almost every industrial field, such as automotive, energy production, and transportation, including critical infrastructures [1–7]. A successful malicious attack could have severe consequences, turning security into a key issue for ESs. These types of components are rapidly evolving [8, 9] and increasing in number [10]. This fact is related to several factors:

- **The reuse of hardware and software:** Open-source hardware and software, and Commercial Off-The-Shelf (COTS) components are being integrated to speed up the development of industrial components [11–13]. COTS are easy to use, but they can introduce vulnerabilities, creating potential entry points for attackers [14,15].
- **New connectivity features:** Industrial components are providing more advanced connectivity features, enabling new automation applications, services, and data exchange. This new connectivity, boost by the fifth generation (5G) of wireless technology for cellular networks, is further opening the window of exposure to any threat [6,10,16,17].
- **More complex systems:** The complexity of industrial components is also increasing with the integration of new trends, such as the Internet of Things (IoT) [17–20], cloud computing, Artificial Intelligence (AI) [20,21], and big data. The extensive use of these technologies further opens the windows for attackers [22–27]. Complexity is a critical aspect of industrial components design, because it is closely related to the number of vulnerabilities [28,29].

This scenario points security is a key aspect of industrial components. Moreover, numerous attacks have been reported targeting industrial enterprises across the globe since 2010 [30]. An exponential rise in such attacks is predicted for future years [31,32].

In summary, the rapid evolution of ESs, their connectivity, and the integration of more and more features, increase their attack surface. This makes it essential to protect their use in environments such as critical infrastructures [33,34]. The sophistication of attacks, a larger attack surface, and the ease of attacks thanks to exploits and tools that decrease the necessary knowledge of the attackers, highlights the need to invest more in cybersecurity. The numerous attacks targeting industrial enterprises across the globe since 2010 reinforce this fact [30], and an exponential rise in such attacks is predicted for the upcoming years [31,32]. As a consequence, security is turning into a critical issue for ESs [35]. However, security by itself is not enough, and the degree of coverage of the implemented countermeasures also has to be evaluated to know whether they are sufficient. Tracking the security status of an ES [36,37] and considering both software and hardware in the evaluation would also be desirable [38–41].

1.2 Research Hypothesis

In this section, the hypotheses of the research are defined. To overcome the process of definition of hypotheses, it has been necessary to analyze the research gaps having as basis the context of the area described in Chapter 2 and related scientific contributions analyzed in Chapter 3.

The hypotheses to which this research seeks to provide an answer are set out and explained below:

H1. **The intrinsic characteristics of embedded systems make it necessary to create a tailored security evaluation methodology.**

ES have two main factors that define them: (1) They are resource-constrained, so power consumption, memory, or functionality are usually limited as a requirement; (2) Their security level is also related to the location and environment where they are working. These differences have to be taken into account when considering existing metrics or creating new ones for ESs, when classifying security metrics, and when assessing the level of security of an ES.

H2. **The result of the evaluation process should be a combination of a numeric value and a detailed description.**

The evaluation result is not enough if the conditions under which the test was carried out are not set. Suppose the report describing the procedure followed during the evaluation is attached to the very first result of the evaluation. In that case, it is possible to re-evaluate the device under the same conditions, obtaining a similar and comparable result over time.

H3. **Not all existing security metrics provide a quantitative, repeatable, and reproducible value that reflects the security level of the ES.**

A metric that returns a similar result under the same circumstances is needed as a solid foundation for an evaluation methodology. Moreover, this stability allows extracting conclusions, and make decisions.

H4. **Breaking down the ES into its assets makes the security evaluation more repeatable:**

This break down gives the evaluator the ability to understand the inner of the System Under Test (SUT). Moreover, this break down let us track possible issues that can appear during the life cycle of ES, effectively implementing a constant security evaluation. It is also a powerful tool to develop more security metrics.

H5. **The aggregation of metrics can serve to identify the more critical elements in an ES, while helping to make decisions about the security status of the device.**

An aggregation mechanism can be applied partially to know which parts are the more vulnerable, and therefore, the ones that need to be addressed first (decision-making tool). Moreover, they can give also an overview of the security status of an ES.

1.2.1 Objective of the Research

Measuring security is not an easy task, but it can be a useful tool to make decisions. Nevertheless, the result of a security evaluation process does not have to be extremely accurate, but meaningful and useful, as well as representative to serves as a decision making tool. This idea is the foundation of this research that can be summarized in the following main, high-level objective:

MAIN OBJECTIVE

Design, implement, and validate a security assessment methodology tailored to embedded systems.

This research is based on two main subjects: (1) evaluation methodologies, and (2) security metrics. Independent sub-objectives are set for each:

1.2.1.1 Evaluation Methodology objectives

EM1. **Identify the main differences between evaluating an ES and a traditional IT system:** Knowing which key aspects are different in each device type will serve as the basis for building the security evaluation methodology oriented to ES.

EM2. **Identify and define repeatable, reproducible and quantitative security metrics, considering dimensions such monetary cost and time:** These properties ensure that the security evaluation methodology is objective, so evaluations carried out under the same conditions and using the same devices give the same result. Moreover, the methodology will track the evolution of the security state of the device through multiple evaluations over time. The cost in money and time of the evaluation process will also be considered to make the methodology applicable.

EM3. **Identify and define how the security evaluation process should be documented:** The evaluation result will be accompanied by a document detailing the conditions under which it was carried out. This ensures the assessment result is repeatable, and reproducible.

EM4. **Break down the ES into its minimal assets for their joint and individual evaluation:** Breaking down an ES into its most basic assets can give us a more detailed picture of the security status. Moreover, we can use this information to extract conclusions about the relationships between assets.

EM5. **Re-evaluate ESs over time:** Tracking the evolution of the security status of ESs is a crucial to monitor potential vulnerabilities, prioritize patching activities, and track the development cycle.

1.2.1.2 Security Metric objectives

SM1. **Analyze the applicability of the existing security metrics and the proposed taxonomies for classifying them:** Limitations in resources (*e.g.*, computation, and power consumption) mean that not every metric is suitable for evaluating the security of ESs.

SM2. **Develop a meaningful method for metric aggregation:** Metrics carry a wealth of information, but a summarized value can be useful for non-technical users.

To better understand the relationships among hypothesis and objectives, in Table 1.1 a mapping of the objectives and the hypothesis is presented.

TABLE 1.1: Mapping of the initial objectives of the thesis to the proposed hypothesis.

Objectives	Hypothesis
EM1	H1
EM2	H3, H4
EM3	H2
EM4	H3, H4
EM5	H2, H4
SM1	H1, H3
SM2	H2, H3, H5

1.3 Scope

Developing a comprehensive security evaluation methodology for ES is an ambitious goal. Therefore, in this section we explain which aspects of our objectives we have been able to achieve, and which have remained outside the scope of this research work. The latter will be considered as future work.

1.3.1 Evaluation Methodologies Objectives

EM1. **Identify the main differences between evaluating an ES and a traditional IT system:** We found that there are differences when evaluating ES and IT systems, being the most important one that ESs usually have restrictive safety requirements.

EM2. **Identify and define repeatable, reproducible and quantitative security metrics, considering dimensions such a monetary cost and time:** We were able to identify existing security metrics, but we were not able to test them for repeatability and reproducibility. Nevertheless, we proposed a set of security metrics oriented to vulnerabilities that are indeed repeatable and reproducible.

EM3. **Identify and define how the security evaluation process will be documented:** Although we explored how each one of the analyzed standards document their findings, we have not proposed a novel documentation flow.

EM4. **Break down the ES into its minimal assets for their joined and individual evaluation:** In our use case, we were able to break down the SUT into its minimal assets. Moreover, we used a directed graph-based representation to also take their relationship into account.

EM5. **The developed evaluation should be carried out over time:** Our vulnerability analysis model makes it possible to evaluate an ES during its whole life spam in a continuous manner.

1.3.2 Security Metrics Objectives

SM1. **Check the applicability of the existing security metrics and the proposed taxonomies for classifying them:** Although we could obtain a list with the existing metrics, and we analyzed their possible applicability to ESs, we did not test them. Nevertheless, we proposed a new taxonomy to classify them according to the structure and properties of ESs.

SM2. **Develop a meaningful method for metric aggregation. Validate the developed method on a real use case:** We proposed a method to aggregate CVSS scores specifically.

1.4 Contributions

The contributions of this thesis are described below:

1. Systematic review of existing security metrics.
2. Proposed ES-oriented taxonomy to classify security metrics.
3. Directed graph-based model to track vulnerabilities over time in ESs.
4. Quantitative metrics to measure the vulnerability status of ES over time.
5. CVSS scores aggregation method.

1.5 Publications

Table 1.2 shows the publications of this thesis, and Table 1.4 shows the correspondence between the publications and the state of the art.

TABLE 1.2: List of publications achieved in the thesis.

Type of Publication	Name	Status
INDIN 2020 Conference	How to Quantify the Security Level of Embedded Systems? A Taxonomy of Security Metrics	Published
MDPI Sensors	A Novel Model for Vulnerability Analysis through Enhanced Directed Graphs and Quantitative Metrics	Published
Conference	Gotta Catch 'em All: Aggregating CVSS Scores	Published

TABLE 1.4: Mapping of the topics reviewed in the state of the art to the publications of the thesis.

Topic	Publication
1) Evaluation methodologies	How to Quantify the Security Level of Embedded Systems? A Taxonomy of Security Metrics
2) Fundamental features of a comprehensive methodology	How to Quantify the Security Level of Embedded Systems? A Taxonomy of Security Metrics
3) Testing types and techniques	Thesis Research Plan
4) Existing standards	Thesis Research Plan
5) Security metrics	How to Quantify the Security Level of Embedded Systems? A Taxonomy of Security Metrics
6) Directed Graphs	A Novel Model for Vulnerability Analysis through Enhanced Directed Graphs and Quantitative Metrics
7) Vulnerability Analysis	A Novel Model for Vulnerability Analysis through Enhanced Directed Graphs and Quantitative Metrics
8) Metric Aggregation	Are you Gonna Buggy?: Aggregating CVSS Scores in EDG

1.6 Document Structure

This document is structured as follows:

1. **Introduction:** In this chapter, the motivations of this research work is described, summarizing the main reasons why ES security is important, and describing the problem to solve in a simplified way.
2. **Background:** In order to understand every aspect of this thesis, the most important concepts and key terms are described and explained, such as ES, security evaluation, and security metrics.
3. **Literature Review:** To set the foundations, and better understand the context, and the gaps in ESs security, a systematic literature review is presented in this chapter. In this section, the context of this research work is also introduced.
4. **How to Quantify the Security Level of Embedded Systems? A Taxonomy of Security Metrics:** In this chapter, we systematically review the literature to gather all existing security metrics to analyze them. With our conclusions, we

proposed a new taxonomy to classify them, according to the structure of ES. This is the first contribution of this thesis.

5. **A Novel Model for Vulnerability Analysis through Enhanced Directed Graphs and Quantitative Metrics:** In this chapter, we introduce a new model based on directed graphs to track known vulnerabilities of ESs over time. To reinforce the proposed model, we also developed an initial set of quantitative metrics. This is the second contribution of the thesis.
6. **Gotta Catch 'em All: Aggregating CVSS Scores:** In the last contribution of this thesis, we proposed a novel method of aggregation of CVSS values.
7. **Discussion, Conclusions and Future Work:** Finally, in this last chapter, we extract the conclusions from the contributions that we have proposed during this document, and we discuss them. The future work is also explained.

Chapter 2

Background

Cybersecurity is a very complex subject, and so are CPSs. Commonly, CPSs are designed and developed to solve a particular problem. The interactions between the different components and systems have to be taken into account to really make an CPS secure, including its connection to the Internet, if any.

In this chapter, the basic concepts related to embedded systems, embedded security, and security evaluation are presented.

2.1 Embedded System

An ES is an applied computer system. Although the definition of ES is not standardized, and it is difficult to pin down, there are some common properties (summarized in Figure 2.1) that characterize them when compared with a Personal Computer (PC) [42–46]:

- **ESs objective:** most embedded devices are primarily designed to accomplish a particular task or a group of specific functions (e.g., an automatic dishwasher, or a digital camera).
- **ESs hardware constrains:** contains in processing performance, power consumption, memory, hardware functionality, and so forth.
- **ESs software limitations:** fewer applications, scaled-down applications, no Operating System (OS) or a limited OS, or less abstraction-level code.
- **ESs quality and reliability requirements:** some families of ESs have a very high threshold of quality, reliability and safety requirements. However, there are also embedded devices in which a malfunction is an inconvenience but not usually a life-threatening situation (e.g., a vending machine).

From this point, any definition of ES must be augmented with examples to better understand the concept. And one of the most significant examples is an automobile, more precisely, all the components and systems that integrate it. An automobile itself is not an ES, but its infotainment head-unit, anti-lock breaking system, powertrain engine control unit, digital instrument cluster, and a plethora of other electronic subsystems - dozens in the typical modern card - are all examples of embedded systems (see Figure 2.2).

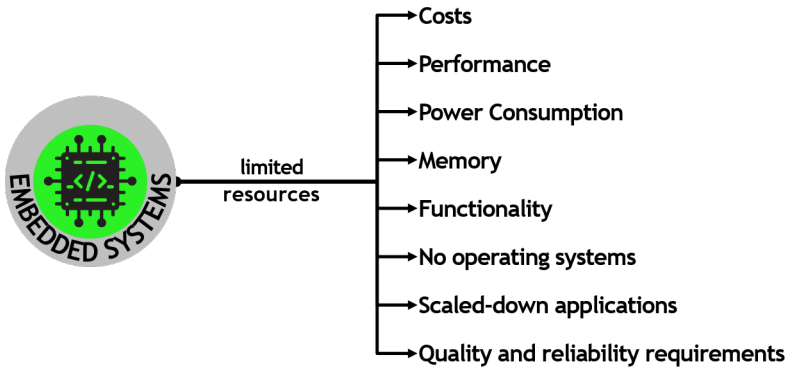

FIGURE 2.1: Most common limited resources in an embedded system.

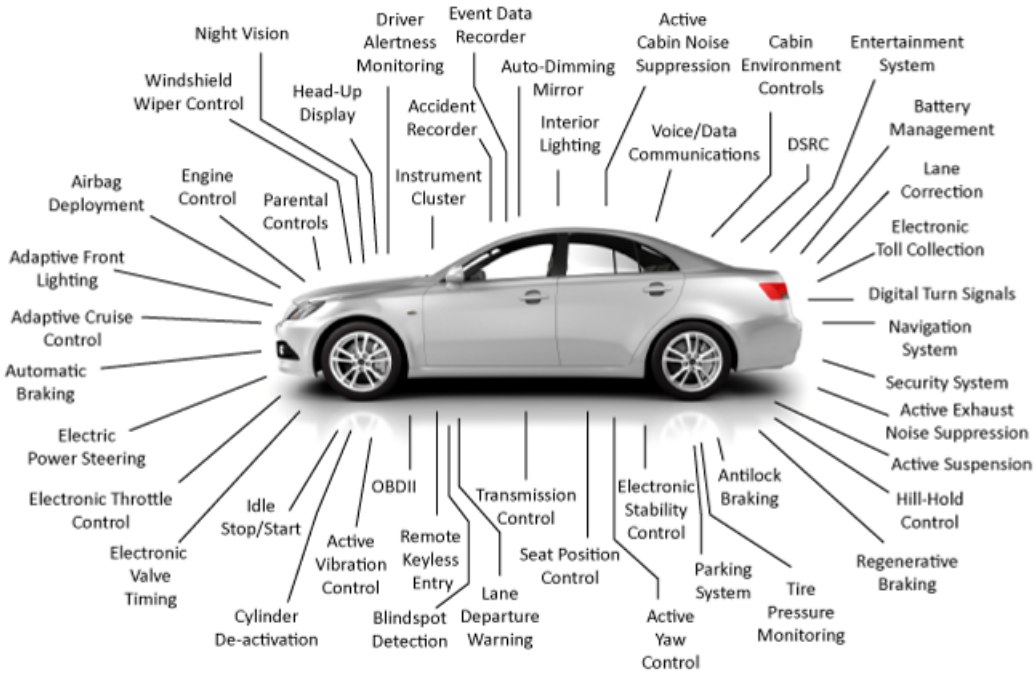

FIGURE 2.2: Summary of some ESs that can be found inside a car.

Nowadays, ESs are present in almost every electronic device: automotive, railway, consumer electronic (microwaves, ovens, washing machines...), vending machines, and manufacturing, to name a few. Even though this definition seems clear, there is some controversy that arises when classifying smartphones. When they were invented, they were conceived to perform just one task - i.e., calling to another device. They were simple devices whose functions were very limited. Nowadays, smartphones have way more functionalities that those initial mobile phones, and calling to another mobile phone is no longer the priority tasks. Some say that smartphones are not ES, but just a miniature desktop computer because of this. Nevertheless, there is little debate that individual components within the phone are ESs.

Finally, it is worth mentioning that ESs are also present in critical infrastructures, such as nuclear power plants, or the power grid. Because of their key role in the modern society, these kinds of infrastructures require a special protection and care, as

a malicious attack could have severe consequences.

2.1.1 Cyber-Physycal System

CPS [7] is a term equivalent to ES, but it makes emphases in the interaction with the physical world. They integrate computation, networking, and physical processes. Embedded computers and networks monitor and control the physical processes, with feedback loops where physical processes affect computations and vice versa. CPS integrates the dynamics of the physical processes with those of the software and networking, providing abstractions and modelling, design, and analysis techniques for the integrated whole.

2.1.2 Industrial Control System (ICS)

Industrial Control System (ICS) [7] is a general term that encompasses several types of control systems and associated instrumentation used for industrial process control. They are usually considered a special kind of ES. Such systems can range in size from a few modular panel-mounted controllers to large interconnected and interactive distributed control systems with many thousands of field connections.

2.1.3 Internet of Things

The IoT can be defined as an ES with internet connection [47]. Initially, this referred to the trend of connecting everyday objects to the internet, connecting to it more objects than people. Nowadays, the definition of the Internet of things has evolved due to the convergence of multiple technologies, real-time analytics, machine learning, commodity sensors, and embedded systems. Traditional fields of embedded systems, wireless sensor networks, control systems, automation (including home and building automation), and others all contribute to enabling the Internet of Things. Nowadays, they can be seen as an open and comprehensive network of intelligent objects that have the capacity to auto-organize, share information, data and resources, reacting and acting in face of situations and changes in the environment.

2.1.4 Industrial Automation and Control Systems (IACS)

According to ISA/IEC 62443 [48] Industrial Automation and Control System (IACS) are the collection of personnel, hardware, software, procedures, and policies involved in the operation of the industrial process and that can affect or influence its safe, secure and reliable operation.

2.1.5 Industrial Component

According to the ISA/IEC 62443-4-2 standard, an Industrial Component (IC) is one of the parts that make up an industrial product or system, including hardware, software, or other components [48]. The ISA/IEC 62443-4-2 document defines four types of components [49]:

- **Software Application**: One or more software programs and their dependencies[1] that are used to interface with the process or the control system itself (*e.g.*,

[1] Any software program that is necessary for the software application to function, such as database packages, reporting tools, or any third party or open source software.

configuration software and historian). Software applications typically execute on host devices or embedded devices.

- **Embedded Device**: Special purpose device designed to directly monitor or control an industrial process (*e.g.*, PLCs, wired or wireless field sensor devices, wired or wireless field actuator devices, Safety Instrumented System (SIS) controllers, distributed control system (DCS) controllers). According to the ISA/IEC 62443, typical attributes of these devices are limited storage, limited number of exposed services, programmed through an external interface, embedded operating systems (OSs), or firmware equivalent, real-time scheduler, may have an attached control panel, and may have a communications interface.
- **Host Devices**: General purpose device running an operating system capable of hosting one or more software applications, data stores or functions from one or more suppliers. This kind of devices typically include attributes such as filesystem, programmable services, no real-time scheduler, and full HMI (keyboard, mouse, etc.).
- **Network Device**: Device that facilitates data flow between devices, or restricts the flow of data, but may not directly interact with a control process. According to the standard, these devices typically include embedded OS or firmware, no HMI, no real-time scheduler, and configured through an external interface.

2.2 System Under Test (SUT)

The System Under Test (SUT) is defined as a system such as a Control System (CS), SCADA system or Monitoring System that is available from a single system supplier [50]. A SUT may be comprised of hardware and software components from several manufacturers but must be integrated into a single system and supported, as a whole, by a single supplier. The SUT may be an IC, a part of an IC, a set of ICs, a unique technology that may never be made into a product, or a combination of these.

2.3 Weakness

Weaknesses are flaws, faults, bugs, and other errors in software and hardware design, architecture, code, or implementation that, if left unaddressed, could result in systems, networks, and hardware being vulnerable to attacks [51] (*e.g.*, buffer overflow).

2.4 Vulnerability

A vulnerability is a flaw in a software, firmware, hardware, or service component resulting from a weakness that can be exploited, causing a negative impact to the confidentiality, integrity, or availability of an impacted component or components. It can be used to gain remote or physical access to a system [52]. To put it in other words: All vulnerabilities rely on weaknesses, but not all weaknesses entail vulnerabilities.

Vulnerabilities can be classified as both known and unknown [53]. Known vulnerabilities are those that have been published in publicly available sources such as the National Vulnerability Database (NVD) [54]. On the other hand, unknown vulnerabilities are dormant vulnerabilities that have not been publicly exposed or exploited.

These potential vulnerabilities have to be discovered by using other methods such as penetration testing [55]. In some cases, unknown vulnerabilities might be known for a group of attackers that do not want to disclose their knowledge to take malicious advantages of it (zero-day vulnerabilities) [56].

2.5 Attack Pattern

An attack pattern is a description of the common attributes and approaches employed by adversaries to exploit known weaknesses in cyber-enabled capabilities [57]. Attack patterns define the challenges that an adversary may face and how they go about solving it.

2.6 Common Platform Enumeration (CPE) Scheme

CPE is an abstract structured naming scheme for describing and identifying applications, operating systems, software, and hardware, including industrial control systems, such as Supervisory Control And Data Acquisition (SCADA) [58]. The logical construct of a CPE is called "Well-Formed CPE Name (WFN)" [59]. The CPE scheme, however, cannot describe and identify specific instances of products (e.g., using serial numbers, particular licenses, or physically discernible products) [60]. CPE is operated by the NIST [61]. The latest version at the time this paper was written is version 2.3.

Version 8.0.6001 of Internet Explorer for its *beta* update can be represented using version 2.3 of the CPE naming specification[2]. The "*a*" after the CPE version indicates that the following element is a software application.

2.7 Common Weakness Enumeration (CWE)

CWE is a community-developed list of common software and hardware weakness types, each one associated with some CVEs (explained in the next subsection) [51, 62]. CWE is operated by the MITRE Corporation [63]. The latest version at the time this paper was written is version 4.3.

CWE-119 is an example of a weakness present in all updates of version 8 of Internet Explorer. It is related to improper restriction of operations within the bounds of a memory buffer.

2.8 Common Vulnerabilities and Exposures (CVE)

CVE is a list of common identifiers for publicly known cybersecurity vulnerabilities [51, 64, 65] operated by the MITRE Corporation [63]. Each CVE includes a unique identification number, a description, one public reference, a reference to the software or hardware that is affected (using CPE), a reference to a weakness, and its severity [66]. As Dimitriadis *et al.* state in [66], "it can be assumed that CVEs are "known knowns" (things we are aware of and understand) or specific vulnerabilities, while

[2] *cpe:2.3:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:**

CWEs are "unknown knowns" (things we understand but are not aware of) or generic vulnerability types." The latest CVE version is always available in its official site[3].

CVE-2015-6154 is an example of a vulnerability that exploits the CWE-119 weakness present in Internet Explorer 8, allowing remote attackers to execute arbitrary code or cause a denial of service (memory corruption) via a crafted website.

2.9 Common Vulnerability Scoring System (CVSS)

CVSS is a public framework that provides a standardized method for assigning quantitative values (scores) to security vulnerabilities (CVE) [65] according to their severity [67]. A CVSS score is a decimal number in the range [0.0, 10.0]. The latest version at the time this paper was written is version 3.1.

The CVSS value for vulnerability CVE-2015-6154 is 9.3 out of 10. This value is classified as critical.

2.10 Common Attack Pattern Enumeration and Classification (CAPEC)

CAPEC is a comprehensive dictionary that provides a publicly available classification taxonomy of known attack patterns (security threats) [68]. In [66], Dimitriadis *et al.* explain that each CAPEC describes the common characteristics of a cyber threat (aka attack-pattern), helping to explain how applications and other cyber-enabled capabilities are likely to be attacked. CAPEC also provides the likelihood that the attack pattern can occur and its impact. CAPEC utilizes a qualitative approach, rating both likelihood and impact in a five-step value scale ranging from very low to very high. Finally, each CAPEC records the weaknesses (CWEs) that the attack pattern can exploit. The latest version at the time this article was written is version 3.4. Figure 2.3 shows the relationship between the different standards [66].

2.11 White-box Testing

White-Box Testing is the detailed investigation of internal logic and structure of the code. In white box testing, it is necessary for a tester to have full knowledge of source code [69].

2.12 Black-box Testing

Black-Box Testing is a technique of testing without having any knowledge of the internal working of the application. It only examines the fundamental aspects of the system and has no or little relevance with the internal logical structure of the system [69].

[3]https://cve.mitre.org/

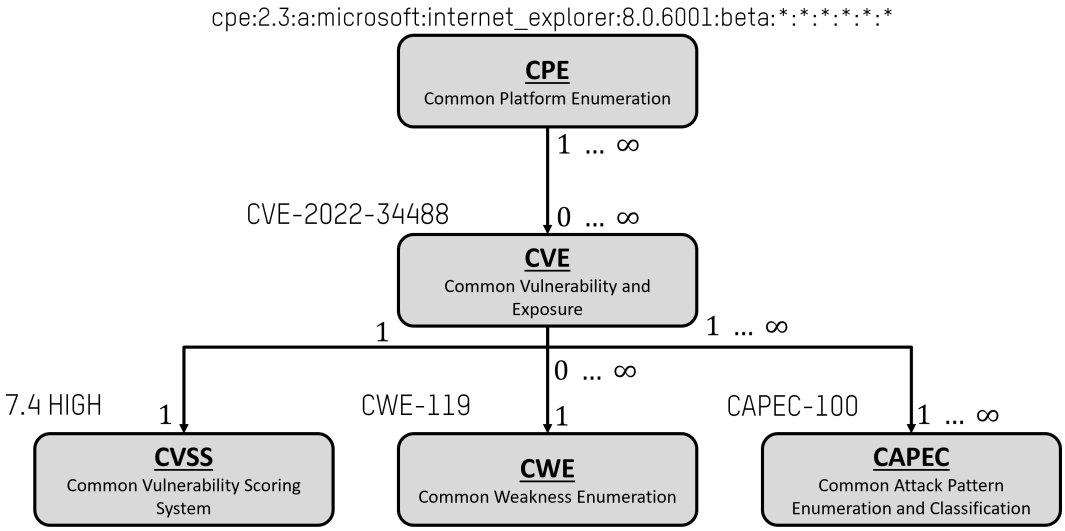

FIGURE 2.3: Relationship between the security standards defined by MITRE and NIST. Taken from [66].

2.13 Grey-box Testing

Grey-Box Testing is a technique to test the application with limited knowledge of the internal working of an application and also has the knowledge of fundamental aspects of the system [69]. It is an intermediate point between white-box and black-box testing.

Chapter 3

Literature Review: State of the Art

Cybersecurity assessment involves not only cybersecurity assessment methodologies, but cybersecurity metrics as well. This chapter describes and discusses relevant scientific works performed within those two cybersecurity assessment categories. Bellow, we list the points that we have researched in our state of the art:

1. **Security metrics:** Proposed security metrics in scientific research papers and standards.
2. **Security Evaluation methodologies:** Existing security evaluation methodologies, in both literature and standards.
3. **Fundamental features of a comprehensive security assessment methodology:** Analysis of the features that an evaluation methodology has to meet to be comprehensive.
4. **Testing types and techniques:** Existing testing techniques in the literature and standards.
5. **Existing cybersecurity evaluation standards:** Both national and international.
6. **Directed Graphs:** Contemporary applications of directed graph in vulnerability analysis.
7. **Vulnerability Analysis:** Existing techniques o perform vulnerability analysis.
8. **Metric Aggregation:** Existing techniques to aggregate security metrics.

In the following sections, we will review each one in more detail.

3.1 Security Metrics

Previously, we described evaluation methodologies as the instructions and steps that evaluators must follow, and testing techniques as the tools that they have to perform each step. Following this example, metrics would be the way for the evaluators to interpret the results obtained.

First, to understand what a security metric is, it is important to distinguish between “metric" and “measurement".

MEASUREMENT

A measurement is a concrete and objective attribute that provides a single-point-in-time view of a specific and discrete factor [70].

An example of measurement would be the number of spams detected.

METRIC

A metric is generated from the analysis of the raw data provided by measurements [70].

A possible corresponding metric for the previous measurement could be the number of spams detected last month compared to the number of spam detected during this month.

Measuring security, both qualitatively and quantitatively, is not an easy task [71–73]. It is a long-standing open problem to the research community, and it is of practical importance to software industry today. Suitable metrics are needed to achieve a foundational science to guide system design and development, and to reveal the safety, security, and possible fragility of complex systems. Using metrics allows systems to be compared and evaluated, so we can increase accountability, demonstrate compliance and determine how much our investments in products and processes are making our systems more secure [74, 75]. There is clearly a growing interest in adopting and using security metrics, yet a lack of widely accepted solutions.

3.1.1 Metrics in the literature

Security cannot be measured as a universal concept due to the complexity, uncertainty, non-stationary, limited observability of operational systems, and malice of attackers [73, 76]. For this reason, there is a growing interest in adopting and using security metrics to measure concrete properties of systems [77]. The most exhaustive surveys about security metrics are commented below:

Atzeni *et al.* discussed the importance of security metrics, and summarized security system assessment methods and metrics in [71]. They concluded that measurements and metrics are necessary to justify investments in security and to manage security.

In [78], Pendleton *et al.* provided a comprehensive survey on quantitative system security metrics. In their work, they analyzed software, network, economic, and effectiveness security metrics. To classify the surveyed metrics, a hierarchical ontology was proposed, and then, each metric was briefly described.

In [79], Ramos *et al.* presented a survey that focuses on model-based quantitative security metrics. The authors review the state of the art of network security metrics in depth, more specifically, in the realm of model-based quantitative metrics. They present a complete and thorough review of the main metrics proposals, remarking the pros and the cons of each metric. Finally, the authors review the open issues and suggestions for future research directions.

3.1.2 Metrics in the standards

We reviewed the industrial standards that are more cited in different articles to find whether they encourage the use of metrics during the evaluation process [80].

OSSTMM [81] defines the *rav*, a scale measurement of the attack surface. In this scale, 100 *rav* means perfect balance and a lower value indicates insufficient controls and therefore a greater attack surface. A value greater than 100 *rav* shows more controls than necessary, which might be inefficient and sometimes a problem, since excessive controls often increase complexity and maintenance issues. The *rav* does not measure risk for an attack surface, rather it enables the measurement of it.

The Common Criteria (CC) [50] defines the Evaluation Assurance Levels (EALs) as an increasing scale to describe the rigor and depth of evaluation. CC list seven levels, from EAL1 (the most basic one) to EAL7 (the most stringent security level). It is important to notice that the EAL levels do not measure security itself. Instead, the emphasis is given to "functional testing", confirming the overall security architecture and design, and performing some testing depending on the EAL to be achieved. In other words, the highest level means that more testing was done, but not that the product is necessarily more secure.

The CC suggests that the output documentation should include the numeric values of the metrics used, as well as the actions taken as a result of the measurements, but there is no reference about how to compute these metrics [82].

The National Institute of Standards and Technology (NIST) has done a lot of research on security metrics, and has proposed nine security metrics for three different aspects: (1) implementation, (2) effectiveness/efficiency, and (3) impact [83]. Moreover, NIST proposes a comprehensive taxonomy to classify the metrics presenting three security categories [84, 85]: (1) management, (2) technical, and (3) operational; and 17 subcategories. According to NIST, metrics should be used from the moment systems are conceived, from their design, creation to operation. Such metrics help identify security problems early and assist in faster and more efficient management and governance [86, 87].

The IEC 62443-1-3 standard [88] includes cybersecurity conformance metrics for managing a security program throughout the life cycle of an industrial automation control system. These metrics provide evidence of conformance for verifiability, completeness, and accuracy. They can also be of assistance to others for assessing the cyber robustness of their control system solutions. This standard also provides a list with the characteristics of a good metric, a metrics development and implementation process, and the steps to be followed for creating metrics. The 62443-4-1 [89] standard also includes some examples of metrics that can be used at the component level, that is, for ES, network components, host software and applications, e.g., functional security, deployment security and current backlog. In Annex A of [88], possible metrics are described to be used in the case of highest maturity level for an organization. Some of these metrics are: results of static code analysis, analysis of attack surface and deviations from Secure Coding Guidelines. They show the effectiveness of the development process with measurable improvements.

3.1.3 Current Issues with Security Metrics

There are many recurrent problems associated with security metrics that have been identified in the literature [90]:

- There are hardly any empirical results confirming the predictive power of security indicators. Hardly any security indicator has a solid theoretical foundation. Many indicators seem to correlate only weakly to actual security properties. A proper method for evaluating the correlation between the degree of process compliance and the actual security level is missing in the literature.
- It is essential to explain how the measure relates to the attribute it captures and to define how it is calculated.
- Empirical evidence in support of the claimed correlation is missing.
- Many security metrics lack an adequate description of the measure's scale, unit and reference values for comparison and interpreting the results. The applicability criteria for the method as well as its assumptions, possible advantages and disadvantages, eventual openly available tool support, and comparison to other properties is not always explained.
- Some indicators implicitly or explicitly rely on extrapolation.
- Non-existing mapping of security metrics and measurements to concrete security goals.

Sentilles *et al.* [91] carried out a literature survey using a systematic mapping methodology to analyze the security metrics proposed in the publications until that moment. The main research question was "What percentage of evaluation method descriptions contains pertinent information to facilitate its use?". They wanted to know (1) what proportion of properties were explicitly defined, (2) what proportion of evaluation methods provided explicitly key information to enable their use (*e.g.*, formulas) and (3) what proportion of other supporting elements of the evaluation methods was explicit (*e.g.*, available implementations). The authors identified the possible gaps concerning security metrics. Only half of the papers clearly specify the assumptions and hypotheses that are assumed to hold when using the method. Despite being

often mentioned in the papers as part of the solution being implemented, in practice, only a few implementations or programs are available to directly support the method evaluation. Sentilles *et al.* could find mentions of available tool support in only 12.5% papers, and only one performed some kind of benchmarking or comparison to similar properties or evaluation methods. This corresponds to only 6 % of all the papers. The authors concluded to a large extent that the information provided in the papers is insufficient to directly apply the method, especially by non-experts.

Even though metrics and methodologies have been proposed in the literature, it is difficult to know about them and understand whether they are applicable in a given context and if they are, how to use them. As Sentilles *et al.* concluded in [91]: "Despite the abundance of metrics found in the literature, those available give us an incomplete, disjointed, hazy view of software security." According with that, it seem reasonable that the future research should be focused on the development of a convincing theoretical foundation, empirical evaluation, and systematic improvement of existing approaches. It should not be focused on the development of more security metrics or approaches.

3.1.4 Granularity vs. Overall Security

As Pendleton *et al.* commented in [78], there is an issue that is far from being trivial when determining whether to aggregate multiple metrics into one or not. If multiple metrics are aggregated into a single value, granularity is lost. On the other hand, if every metric is kept as a single value, reporting details of attack, impact or countermeasures would be endless. This is known as the challenge in "measurability." The authors identify the advantages and the disadvantages of both options, and different proposed alternatives for each one.

3.1.5 Scales of Metrics

Attributes of an object should be measured by certain scales [92]. The following classification has been used in various previous works [78, 93]:

- **Nominal:** also known as *categorical scale*, they are represented by a discrete unordered set. Measurement in a nominal scale can be compared using standard set operations. Nominal scales can be transformed between each other via some methods for one-to-one mapping.
- **Ordinal:** this scale is a discrete ordered set, in which we can use operators such as "greater than" to compare two measurements on the same ordinal scale. Ordinal scales can be transformed between each other based on monotonic (i.e., order-preserving) one-to-one mapping methods. Ordinal scales are one step higher (i.e., more accurate in the measurement) than nominal scales in the hierarchy of scales because it can downgrade to nominal scale at the cost of information loss.
- **Interval:** this scale extends the ordinal scale such that the distance between adjacent points on the scale is constant. This equal-distance property allows the difference operator by which one can compare the difference between two points and the difference between another two points. One can define linear transformations between two interval scales, which preserve constant distance

between two points on the same scale. Interval scales can be analyzed via statistical analysis, such as mean, variance, and even higher moments. Interval scales can be downgraded to ordinal (and therefore nominal) scales by relaxing the equal-distance condition on the scale.

- **Ratio:** this scale extends the interval scale by defining the origin (i.e., value of 0) in a natural fashion. Ratio scales can be operated with multiplication, logarithm, and finding roots. Transformations can be defined between ratio scales.
- **Absolute:** this measures things measurable only in a single method. Counting is a simple example of absolute scale. Any kind of statistical analysis can be applied to measurements in the absolute scale.
- **Distribution:** this scale is used to measure an attribute with inherent uncertainty.

As it is said in [78], "It is worth noting that using a different scale gives a different level of accuracy, leading to a different degree of understanding in a given value as a metric." A higher level of scale in the hierarchy gives higher accuracy that can be easily transformed to a low level of scale. Measurements using a higher level of scale permits richer statistical inference.

3.1.6 Features of a Comprehensive Metric

Metrics are important for quantifying the security level achieved; however, computing metrics can be exhausting and time-consuming resulting in not being practical. Furthermore, it is not resource-effective to measure everything, thus using the appropriate metrics is critical.

Some authors have tried to characterize which criteria should meet a comprehensive metric. Among their efforts, three main ideas can be found in the literature: (1) the SMART criterion [94], (2) the PRAGMATIC criterion [95], and (3) the characteristics identified in the work carried out by Savola [77].

Since Doran proposed the "SMART" (Specific, Measurable, Attainable, Repeatable, Time-dependent) criteria in [94] (see Table 3.1), it has been widely used, developed and extended, and it has been also adapted for security metrics development [96]. Although several interpretations have arisen about what the letters in SMART actually mean, it is important to note that, while the exact definition of each letter is not critical, it should be meaningful and not redundant in a larger context. On the other hand, Krag *et al.* [95] propose a set of nine criteria for assessing and selecting metrics, using the acronym "PRAGMATIC" (Predictive, Relevant, Actionable, Genuine, Meaningful, Accurate, Timely, Independent, Cheap) (see Table 3.2).

According to Jaquith [97], a good metric should be consistently measured, without subjective criteria; cheap to gather, preferably in an automated way; expressed as a cardinal number or percentage and not with quality labels; expressed using at least one unit of measure; and should be contextually specific, so it is relevant enough to take action based on their values.

In their survey [78], Pendleton *et al.* discuss three main perspectives towards a good metric: (1) the conceptual perspective, that says that a good metric should be intuitively easy to understand and with high usability; (2) the measurement

TABLE 3.1: Definition for each letter of the SMART criterion.

SMART	Definition
Specific	Well-defined, unambiguous wording
Measurable	Quantitative when feasible
Attainable	Within budgetary and technical limitations
Repeatable	Do not vary depending on the person taking them
Time-dependent	Multiple time slices

TABLE 3.2: Definition for each letter of the PRAGMATIC criterion.

PRAGMATIC	Definition
Predictive	Predicts security outcomes
Relevant	Useful information concerning aspects that we truly care about
Actionable	Less lag between measurement and reaction
Genuine	Unambiguous, straightforward, credible, real, and objective
Meaningful	Meaningful to the intended recipients, audiences, or users of the metrics
Accurate	Accuracy and precision
Timely	Minimising the delay between collecting, analysing, and presenting the data
Independent	Truthful, honest, credible, objective representation, and resistant to manipulation
Cheap	Cost of collecting, analysing, and using metrics

perspective (as done in [97]); and (3) the utility perspective, that claims that a metric should allow both horizontal and temporal comparison between enterprise systems.

In [77,98], Savola analyses the literature to identify the quality criteria and dimensions of security metrics and measurement process. In [98], the pre-existing goodness criteria are listed. Afterwards, in [77], Savola takes research one step further by conducting expert surveys and interviews to 141 security experts from 21 different countries, and extracting 19 quality criteria (see Table 3.3). According to his results, correctness, measurability, and meaningfulness are the main quality criteria along with usability.

The quality criteria that Savola proposes are intended to support human decision-making in every step of the life cycle, e.g., security engineering activities at design and development practices or in security management activities at run time. In his work, Savola concludes that there is no security approach that would allow the measurement of security as a universal property, since there is no security metric that could fulfill all the quality criteria, so security metrics cannot be used to measure security as a whole. He finally points out that there is a need for widely-accepted approaches for security measuring.

3.1.7 Taxonomies for security metrics

In [93,99], Savola reviews the most common taxonomies for security metrics, and also provides a new model to taxonomize security metrics for technical systems to systematize and organize their development activities. The most common classification for metrics divides them as organizational, technical and operational [93,100].

- **Organizational security metrics:** They evaluate the security level of a whole organization. They describe and track how effectively organizational programs

TABLE 3.3: Desired quality criteria of Security Metrics (SM) identified in [77,98].

Dimension	Explanation
Correctness	The SM are correctly implemented and error-free.
Granularity	The SM allow the measurement results that differ from each other to be distinguished at an adequate level.
Objectivity Unbiasedness	Results are not influenced by the measurer's will, beliefs, or feelings. Results are not influenced by any bias.
Controllability	Results should be kept within the defined limits or the measurement window.
Time dependability	The time-dependent behavior of SM can be leading, coincident, or lagging. The time dependability of the SM should be part of their specification.
Comparability	Support comparison of the targets that they represent.
Measurability	Capable of having dimensions, quantity or capacity ascertained in the SuI.
Attainability Availability Easiness	Measurement results can be achieved from the SuI Results are generally available How easy it is to achieve the measurement results.
Reproducibility, repeatability Scale reliability	The same results are achieved if a measurement is repeated in the same context, with exactly the same conditions. Reproducibility of the measurement by different persons.
Cost effectiveness	The measurement gathering and measurement approaches should be cost effective.
Scalability Portability	The SM should be applicable to SuIs of different sizes Applicability of the SM to various target systems.
Non-intrusiveness	The measurements should not harm the normal operation of the SuI, require only minimum changes to the SuI, and not affect the measurement results.
Meaningfulness	The SM should be relevant and respond to the needs.
Effectiveness	The expectations of the adequacy of the SM sufficiency in the final use environment are satisfied.
Efficiency	The adequate requirements of the SM are achieved with only minimal undesired use of effort and time.
Representativeness, contextual specificity	The SM correspond to the actual system characteristics in the SuI in a contextually focused way.
Clarity Succinctness	The SM are clearly formulized. Only important parameters are considered.
Ability to show progression	The SM should be able to show progression on the dimension it addresses.
Completeness	The collection of SM should be complete from a measurement objectives perspective.

and processes achieve cybersecurity [74,100–103]. They do not just include all the systems, but could also take into account all the people involved for the enterprise to work. Organizations can use measures and metrics to set goals (or benchmarks), and determine success or failure against the benchmarks. Some more references on this topic can be found in [95,104–106].

- **Technical security metrics:** They indicate the level of security a specific system exhibits. They describe and compare technical objects, *e.g.*, algorithms, specifications and architectures, and to indicate the level of security a specific system exhibits. Since the definition of what a "system" is covers a broad spectrum, technical security metrics also cover a wide range of systems, going from network security metrics to embedded devices security metrics.
- **Operational security metrics:** They are describe and manage the risk to an operational environment, including as-used systems and operating practices [100]. They include measures of operational readiness or security postures, measures used in risk management, metrics describing threat environment, and metrics

supporting incident response and vulnerability analysis. They could also include metrics produced as part of normal operations that can serve as input to other security metrics. The estimation of operational metrics generally requires experimental or empirical measurements.

3.1.8 Gaps Detected and Critics to the Current Status

Security metrics are a key piece to obtain an objective result in every evaluation process. Metrics are developed to measure concrete properties in very specific contexts. This means that they are not usually adaptable to a different target. As happens with security evaluation methodologies and security standards, there is also a bunch of security metrics to choose from. They can be used to evaluate almost every single aspect of a system or an organization, and they can be found in many forms. Under this scenario, it is not difficult to realize that choosing the right metrics is also an overwhelming task.

On the other hand, when talking about metrics, generalization is not always possible. This means that each dimension of the ES has associated a set of metrics that is the most suitable for evaluating that concrete aspect of the device. So having a vast amount of metrics is useless if there is no method to choose among them each time a security evaluation has to be made. It is as important having metrics to choose from, as it is to have a method to choose the right metric for each case, and how to apply each one.

3.2 Security Evaluation Methodologies and Standards

There are many security evaluation methodologies available in the literature. In this section, we review the most cited security evaluation methodologies surveyed by ENISA [107]. Table 3.4 shows them in alphabetical order.

TABLE 3.4: Comparison of security evaluation methodologies.

Methods	Last version	Year	Status [1] [2]	Standard Type [1] [2]	Metric proposal [1] [2]	Scopeof the evaluation [1] [2] [3] [4] [5] [6] [7] [8]	Documentation accesibility [1] [2] [3]
			1. Up to date 2. Deprecated	1. International 2. National	1. Yes 2. No	1. Organization 2. Network 3. Device 4. Hardware 5. Software 6. Web 7. User 8. Data	1. Free 2. Payment 3. Unaccesible
CC [108]	3.1	2017	■ □	■ □	□ ■	□ □ ■ ■ ■ □ ■ ■	■ □ □
CPA [109,110]	1.4	2018	■ □	□ ■	□ ■	□ □ ■ ■ ■ □ □ ■	□ ■ □
CSPN [111]	3.0	2021	■ □	□ ■	□ ■	□ □ ■ ■ ■ □ □ ■	□ ■ □
ICCF [112]	1.0	2018	■ □	■ □	□ ■	□ □ ■ ■ ■ □ ■ ■	■ □ □
ISA/IEC 62443 [48]	1.0	2018	■ □	■ □	□ ■	■ ■ ■ ■ ■ □ □ ■	□ ■ □
ISSAF [113,114]	0.2.1	2016	□ ■	■ □	□ ■	■ ■ □ □ □ ■ ■ □	■ □ □
NESCOR [115]	3	2015	■ □	□ ■	□ ■	■ ■ □ □ □ ■ ■ □	■ □ □
NITES [116]	2	2009	□ ■	□ ■	□ ■	□ □ ■ ■ ■ □ ■ ■	■ □ ■
NIST [117]	2	2008	■ □	□ ■	■ □	■ ■ □ □ □ ■ ■ ■	■ □ □
OSSTMM [118]	3	2010	■ □	■ □	■ □	■ ■ □ □ □ ■ ■ ■	■ □ □
OWASP [119]	4.2	2014	■ □	■ □	□ ■	□ □ □ □ □ ■ ■ ■	■ □ □
PTES [120,121]	1	2014	■ □	■ □	□ ■	■ ■ □ □ □ ■ ■ □	■ □ □

3.2.1 Common Criteria (CC)

The Common Criteria for Information Technology Security Evaluation (Common Criteria or CC) is an international standard (ISO/IEC 15408) for computer security certification [108, 122]. CC is a framework which provides assurance that the process of specification, implementation, and evaluation of a computer security product has been conducted in a rigorous, standard and repeatable manner [50, 82]. CC defines seven levels of security or Evaluation Assurance Level (EAL) of a product or system, from EAL 1 (the lowest one) to EAL 7 (the highest one). EAL 7 indicates the product satisfies the most demanding development process. The CC Certification is recognized by 30 countries around the world that have signed the Common Criteria Recognition Agreement (CCRA).

CC has become so popular that it has served as the foundation of a vast amount of other methodologies, and standards all over the world. Nevertheless, CC exhibits some drawback [123, 124]:

- The evaluation is a costly process and the vendor's return on that investment is not necessarily a more secure product.
- The evaluation focuses primarily on assessing the evaluation documentation, not on the actual security, technical correctness or merits of the product itself.

- The effort and time necessary to prepare evaluation evidence and other evaluation-related documentation is so cumbersome that by the time the work is completed, the product in evaluation is generally obsolete.
- CC recognizes the need to limit the scope of evaluation, therefore, evaluation activities are only performed to a certain depth, use of time, and resources.
- Certification schemes are different depending on the country, and they can lead to different assessment results.

3.2.2 CPA

The Commercial Product Assurance (CPA) scheme is a UK scheme to assess commercial products [109,110]. The certification scheme is governed by the National Cyber Security Center (NCSC). The CPA is based on the Common Criteria.

The evaluation is mainly black box because it does not require confidential information. The products are tested against the so-called security features defined by the CPA, and the security features depend on the product categories (e.g., encryption of data, software execution control, remote desktop, firewall, real-time secure communication, virtualization platforms, or smart meters).

Nowadays, the CPA only certifies smart meters or smart metering products. Certification is usually valid for six years.

3.2.3 CSPN

The Certification Sécuritaire de Premier Niveau (CSPN) is a French IT security certification scheme established in 2008 [111]. Created by ANSSI in 2008, the CSPN's main objective is to offer a faster and cheaper alternative for IT security certification compared to Common Criteria. CSPN assessments follow a black box testing approach within a fixed and limited time frame. Therefore, it is less rigorous and targeted to low threat or risk environments. The CSPN covers mostly IT security products such as antivirus, firewall access control mechanisms and secure storage devices. It can also be applied to hardware.

3.2.4 IACS Cybersecurity Certification Framework (ICCF)

The IACS components Certification Framework (ICCF) is a certification scheme developed by the European Reference Network for Critical Infrastructure Protection (ERNCIP) [112]. ICCF focuses on the security of IACS components, rather than on subsystems or even entire IACS.

The ICCF scheme takes particular account of the work done on the ISA/EC 62443 standards. For example, IEC 62443-4-2 was adopted as the basic set of cybersecurity requirements. In addition, the proposed assessment scheme is heavily influenced by the Common Criteria.

3.2.5 IEC 62443-4-1

Document IEC 62443-4-1 serves as a reference for the definition of a life cycle for the development of industrial cybersecure components. These components must

comply with a series of technical requirements for cybersecurity that are defined in document IEC 62443-4-2 and which, after implementation, would result in product instances configured to be integrated within a complete industrial solution (system). System-level integration should comply with IEC 62443-2-4 and IEC 62443-3-2, which should also consider the technical requirements at the system level defined in IEC 62443-3-3.

IEC 62443-4-1 standard describes the requirements that apply in the life cycle of a product or industrial component. IEC 62443-4-1 classifies the requirements into the following practices:

1. **Practice 1:** Security management.
2. **Practice 2:** Specification of security requirements.
3. **Practice 3:** Secure by design.
4. **Practice 4:** Secure implementation.
5. **Practice 5:** Security verification and validation testing.
6. **Practice 6:** Management of security-related issues.
7. **Practice 7:** Security update management.
8. **Practice 8:** Security guidelines.

The standard also tries to establish how to perform the concrete tests and how to document them, however, this process is not clear. For example, the following table shows the different tests to be performed in an IACS, however, these are defined in an ambiguous way and there is no relationship between testing and the required Security Level (SL).

Security Levels (SL) are described in IEC 62443-4-2, and for each component must decide what level of protection is desired for each of the fundamental protection requirements. The following table shows the relationship between the SL and the type of attacker, with SL 1 being classified as protection against accidental errors and SL 4 as protection against intentional attacks with sophisticated and highly motivated means.

3.2.6 Information Systems Security Assessment Framework (ISSAF)

It is a well-established penetration testing methodology developed by the Open Information Systems Security Group [125]. ISSAF provides comprehensive step-by-step penetration testing technical guidance. It is designed to evaluate the security of networks, systems and application controls.

The methodology outlines three well-defined action areas, and details the nine steps composing the main one (see Figure 3.1) as follows [113, 114]:

1. **Planning and Preparation:** The first phase encompasses the steps needed to set the testing environment up, such as: planning and preparing test tools, contracts and legal protection, definition of the engagement team, deadlines, requirements and structure of the final reports.

2. **Assessment:** This phase is the core of the methodology, where the real penetration tests are carried out. The assessment phase is articulated in the following activities: (1) Information Gathering, (2) Network Mapping, (3) Vulnerability Identification, (4) Penetration, (5) Gaining Access and Privilege Escalation, (6) Enumerating Further, (7) Compromise Remote Users Sites, (8) Maintaining Access, (9) Covering Tracks.
3. **Reporting, Clean-up and Destroy Artifacts:** During this phase, at the very end of the active parts of the methodology, testers have to write a complete report and to destroy artifacts built during the Assessment phase.

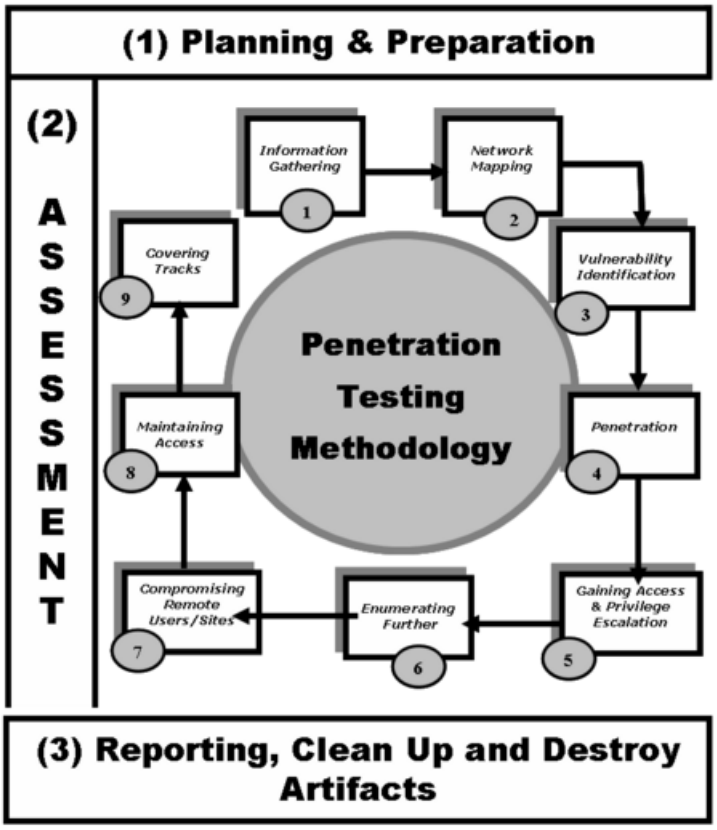

FIGURE 3.1: ISSAF workflow [113].

ISSAF has a clear and very intuitive structure, which guides the tester through the complicated assessment steps. The order in which the methodology describes the penetration testing process is optimized to help the tester perform a complete and correct penetration testing, avoiding the mistakes commonly associated with randomly selected attack strategies.

On the negative side, the reporting section is poorly implemented. There is not a well-defined and accurate guideline to develop final reports, and some suggestions are outdated. For example, the methodology suggests destroying and cleaning up the developed artifacts, while the most current practice is to leave the developed artifacts on the tested system, to ease further and deeper analysis [126].

3.2.7 NESCOR Guide to Penetration Testing for Electric Utilities

The National Electric Sector Cybersecurity Organization Resource (NESCOR) [127] has created a test template to provide network operators with a guide to help them perform penetration testing of smart grids [115]. It can be used for advanced metering infrastructures, distributed energy resources, and Electric Transportation [128].

The methodology provides for a linear sequence of tests and is divided into six phases with nine work packages, as Figure 3.2 shows.

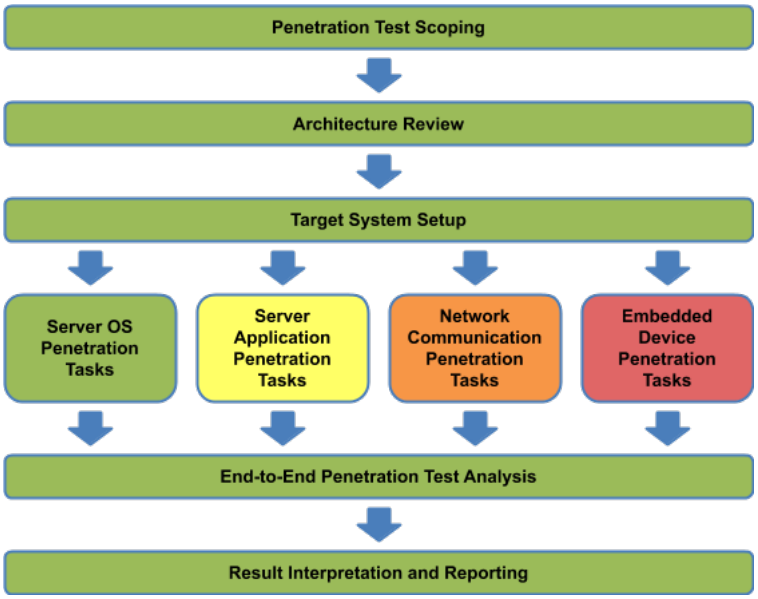

FIGURE 3.2: NESCOR workflow.

This methodology is based on the assumption that testers will initially gain a comprehensive overview of the network to be investigated by means of expert interviews, among other things.

3.2.8 NITES

The Singaporean National IT Evaluation Scheme (NITES) was established by the Security Accreditation Committee (SAC) [116]. NITES is an adaptation of Common Criteria v3.1, equivalent to the EAL4+ package.

There are four categories of products that can be assessed under NITES:

1. Secure portable storage.
2. Network related devices (i.e. VPN).
3. File/folder encryption.
4. Key management solutions (i.e. HSM).

As the SAC was accredited by Common Criteria as an issuing certification body in January 2019, the NITES scheme has lost relevance. The Common Criteria certification scheme is called the Singapore Common Criteria Scheme (SCCS).

3.2.9 NIST Guidelines on Network Security Testing (GNST)

The Guideline on Network Security Testing (GNST) [117] was the first methodology that introduced a formal process for reporting and to take advantage of inducted hypotheses [126]. GNST was developed by National Institute of Standards and Technology (NIST) [61].

GNST follows four main steps (Figure 3.3):

- **Planning:** The system is analyzed to find out the most interesting test targets.
- **Discovery:** The tester searches the system, looking for vulnerabilities.
- **Attack:** The tester verifies whether the found vulnerabilities can be exploited.
- **Reporting:** In the last step, every result is reported.

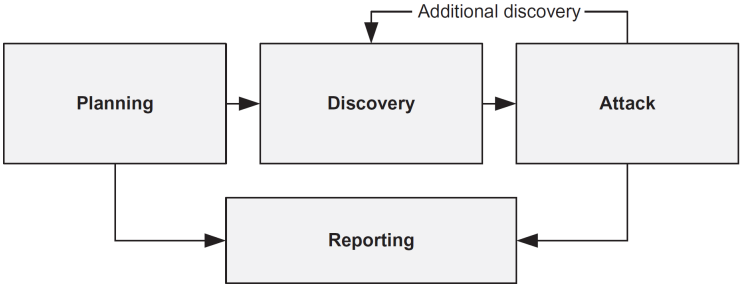

FIGURE 3.3: GNST workflow.

GNST was updated by Technical Guide to Information Security Testing and Assessment in 2008 [129].

3.2.10 Open Source Security Testing Methodology Manual (OSSTMM)

The Open Source Security Testing Methodology Manual (OSSTMM) was developed by the Institute for Security and Open Methodologies (ISECOM) in January 2001 [130]. It was the first methodology to include human factors in the tests, taking into account the established fact that humans may be very dangerous for the system. Capturing the human factors such as insider attacks and social engineering attacks is a valuable feature, which no one else have never tried to capture.

OSSTMM is the *de facto* standard for security testers [126] to find vulnerabilities. It describes a complete testing methodology, offering comprehensive tools to report the result set. It is designed to test the operational security of physical locations, workflows, human security testing, physical security testing, wireless security testing, telecommunication security testing, data networks security testing and compliance [118].

Although it is a great methodology, OSSTMM fails to deliver some key concepts, namely [126]:

- **Control flow analysis:** It is very biased towards communication analysis, forcing the tester to put a lot of effort on data flow, while control flow and application analysis is, comparatively, neglected.

- **Readable diagrams:** OSSTMM does not provide intuitive diagrams, mainly because of the high number of items and of many messy loops inside the flow.
- **data loss process on writing reports and traceable reports:** It offers nice templates to fill up the reports, but unfortunately these templates follow a strictly linear reading process. A good practice would be writing reports after each action; otherwise, many details easily fall into oblivion.

3.2.11 Open Web Application Security Project Testing Guide (OWASP)

The OWASP Testing Project [119] is a methodology developed by the Open Web Application Security Project (OWASP) to help people understand the what, why, when, where, and how of testing web applications [131, 132]. Figure 3.4 shows the workflow of the OWASP Testing Guide.

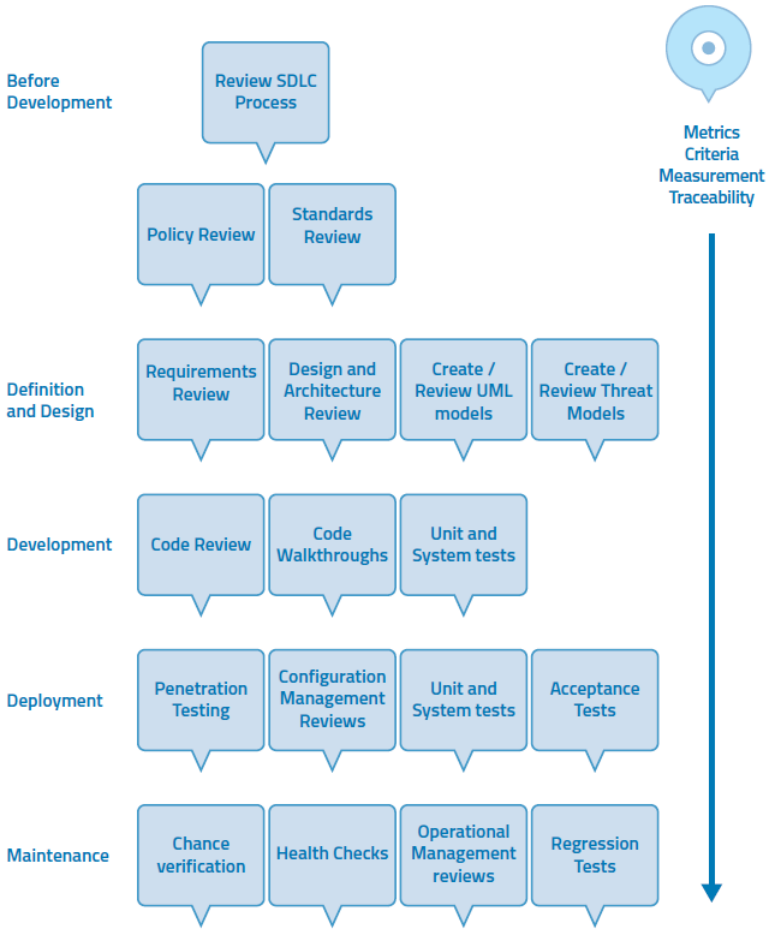

FIGURE 3.4: OWASP workflow [119].

3.2.12 Penetration Testing Execution Standard (PTES)

The Penetration Testing Execution Standard (PTES) consists of seven main sections [120]. These cover everything related to a penetration test. The main sections (Figure 3.5) defined by the standard as the basis for penetration testing execution are:

(1) Pre-engagement Interactions, (2) Intelligence Gathering, (3) Threat Modelling, (4) Vulnerability Analysis, (5) Exploitation, (6) Post Exploitation, (7) Reporting.

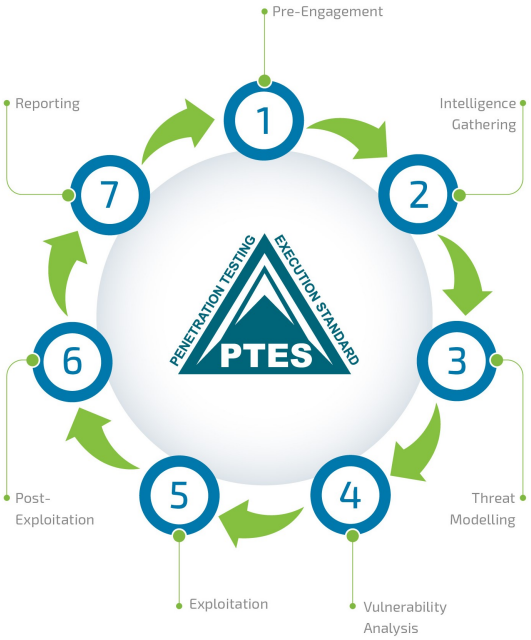

FIGURE 3.5: PTES workflow [133].

As the standard does not provide any technical guidelines as far as how to execute an actual pentest, a technical guide to accompany the standard itself was proposed [121].

3.2.13 Fundamental Features of a Comprehensive Evaluation Methodology

We have already reviewed the most relevant security evaluation methodologies for the industry in the previous section, but we lack a means of comparing them with each other. For that reason, we also reviewed the literature to find which are the fundamental features that a comprehensive evaluation methodology should have. In this section, we analyze them, and apply them to compare the security evaluation methodologies described in Section 3.2.

In [126], the fundamental features of a comprehensive methodology were identified:

1. **Modelling:** The methodology should explicitly define the key concepts in order to facilitate the tester in modelling both the system and the evaluation process, by removing potential ambiguities and leading the tester to the kind of model that better suits the subsequent activities.

2. **Planning:** The methodology should support the tester in laying detailed test plans out, *i.e.*, definition of phases, prerequisites for each phase, tools to use in each phase, expected outcomes. During planning, the methodology should help the tester in selecting the most appropriate and useful activities for the specific system.

3. **Flexibility:** While statically defining a test plan is an important step, an even more powerful feature would be providing structured means of dynamically

integrating additions (deriving from the results that are acquired at each step) in the initially defined plan, leading to richer, or more specific, new plans. Without this possibility, the tester would be strongly exposed to the risk of defining either too general or too specific targets; in the former case, the procedure to achieve them would lack the right amount of detail to ensure its correctness; in the latter, significant alternatives would possibly be excluded a priori.

4. **Adaptation:** The concepts and models defined within a methodology should certainly be unambiguous, but this quality should not hinder the possibility to adapt them to many variations of the real systems to be tested.

5. **Guidance:** Given the huge amount of different aspects involved in security evaluation, the methodology should offer practical guidance about what activities compose a testing session, and which tasks are needed before, during, and after each activity, for example by means of up-to-date checklists related to the vulnerabilities and the most effective testing procedures for different testing contexts.

6. **Reporting:** Guidance should not be limited to the active phases of testing, but also extend to the documentation of every useful information related to the test setup, environment, progress and results. Supporting the tester in the reporting activity means not only helping him not to omit important details, but also letting him format the information in one or more ways that are suitable for different kinds of readers (technicians, policy-makers, managers, etc.).

7. **Granularity:** Finding good guidance and reporting features in a methodology, commonly means having lots of highly-detailed information at hand. However, capturing the details only where needed, while not uselessly encumbering the testing and reporting activities, is equally important. This criterion applies both to data collection and to task planning. With regard to the former, the methodology should not force filling out detailed reports about low-severity, low-priority or low-probability scenarios. With regard to the latter, the methodology should cater for the easy selection of sensible steps and the provision for skipping the useless ones, possibly foreseeing nested levels of planned tasks.

Table 3.5 shows an at-a-glance comparison of the methodologies previously discussed with respect to the properties defined above.

TABLE 3.5: Feature map of the security evaluation methodologies (+ good coverage, = average coverage, - limited or no coverage) [126, 128, 134].

Features	OSSTMM	ISSAF	CEM	PTES	OWASP	NESCOR	GNST
Modeling	=	+	-	=	+	+	=
Planning	+	+	=	+	+	+	=
Flexibility	-	-	=	+	-	-	+
Adaptation	+	=	-	+	=	-	=
Guidance	=	=	=	=		+	+
Reporting	-	-	-	=	=	=	-
Granularity	=	+	-	=		=	=

The main issue with this scheme is that it is developed from a theoretical point of view, and it does not consider the cost of the evaluation process (both time and money). This is a key aspect, because it is the one that best reflects the reality of the evaluation process: a security evaluation methodology could theoretically be perfect in all aspects, but if it takes too long, or if it is too expensive to obtain a result, the methodology will be useless. This is why a last factor called "applicability", or "cost" should be introduced to balance the mix, and to adapt it to the industry.

3.2.14 Gaps Detected and Critics to the Current Status

Each analyzed methodology was devised with a different goal, and a different environment in mind. There are so many standards and methodologies that choosing one for any application can be overwhelming. So it is very common to find a bunch of standards for the same application, with different requirements. Moreover, they differ from country to country, and international standards are not always used. This scenario shows that it is not an easy task to develop a security evaluation methodology that can be applied to every single device, case and situation.

Taking a step forward in the analysis, the majority of the analyzed methodologies are conceived for a high-level system, namely network or organization level. This is because of the high demand of the sector, where every organization wants to be sure that they are protected both from competitors, and possible attackers. This encourages most efforts to focus on developing robust and comprehensive methodologies to protect business. As a consequence, other scenarios, which also need to be evaluated to know their security level, are neglected, and the methodologies available are more scarce, as is the case with ESs.

The security evaluation methodologies shown here can be classified according to their drawbacks when trying to apply them to an ES. Three scenarios can be differentiated:

1. **Application is possible with modifications:** This means that the general structure and content of the methodology have potential to assess the security level of ESs, but because of the nature of ESs, the methodology has to be adapted in order to be used. This is the case of OSSTMM, that was conceived for organizations, but it is so systematic that can be extrapolated to ESs.
2. **Application is possible, but the evaluation takes a long time (with or without modification):** This means that the evaluation process takes too long, and depending on the product, it could be obsolete by the time the evaluation is done. Sometimes the evaluation process can be expensive not only in time but also in money. This is the case of CEM of Common Criteria, where the evaluations time can take somewhat between six months up to one year.
3. **Impossibility to apply the methodology:** This means that there is no possible way to alter or modify the methodology to apply it to an ES. This is usually cased by very specialized methodologies developed for very concrete scenarios. This is the case of the OWASP testing methodology that is oriented to web applications.

As can be seen, a lot of work has been done at the organization level, but to the authors' knowledge, there is no security evaluation methodology that is specifically design to assess the security level of ESs.

Finally, it is very common to find recommendations of using metrics in the security evaluation process, and the advantages that they introduce, but they are not usually defined in any way. Analyzed methodologies let the evaluators choose their own set of metrics for each target of evaluation, increasing the subjectivity of the process.

3.3 Testing Types and Techniques in the Literature

Security evaluation methodologies, and in general, any methodology or standard, set the foundations for a given process. They standardize and list the steps to follow, and how to report the results. They are developed in this way in order to cover the largest number of cases in a specific domain. But it is the case that most of the time, standards specify what has to be done, but not how it should be done. If standards are like the instructions that the evaluators have to follow, testing techniques are the tools that the evaluator has to carry out the tasks described in the standards.

In this section, we define what is security testing, and we reviewed the literature to find the most relevant testing types and techniques used today. We classify them according to when they can be applied in the development life cycle.

3.3.1 Security Testing

Security testing is one type of assessment method used for determining whether the information system being assessed meets the security objectives (confidentiality, integrity, and availability). Security testing, as defined by the National Institute of Standards and Technology (NIST), is the "process of exercising one or more assessment objects under specified conditions to compare actual with expected behavior, the results of which are used to support the determination of security control existence, functionality, correctness, completeness, and potential for improvement over time" [135] (see Figure 3.6). Security testing can be used to determine if technical mechanisms or organizational activities meet a set of predefined specifications, or it can be used to evaluate whether security controls can be circumvented.

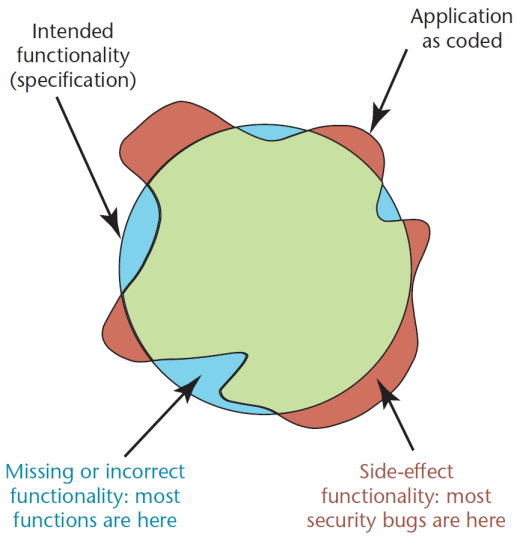

FIGURE 3.6: Intended versus implemented software behavior in applications. The circle represents the software's intended behavior as defined by the specification. The amorphous shape superimposed on it represents the application's true behavior. Most security bugs lay in the areas of the figure beyond the circle, as side effects of normal application functionality [136].

Security testing can be conducted using manual or automated techniques. Both testing techniques have advantages and disadvantages. Manual security testing is not only time-consuming; it can also be more expensive than automated testing, and qualified testers are required. Automated testing uses tools to execute tests and automate tasks, thereby improving the efficiency of security testing. Automated tools can also improve the time to complete security testing, and can be more frequently updated based on new types of vulnerabilities and exploits. However, automated security testing reduces the types of testing techniques (e.g., social engineering) that can be performed based on the limitation of automated tools to mimic human behavior. Therefore, both manually and automated testing should be considered against the depth and coverage required for the security testing [137].

It is essential to take testing into account in all phases of the secure software development life cycle [138]. Thus, security testing must be holistic, covering the whole secure software development life cycle. In concrete terms, Fig. 3.7 shows a recommended distribution of static and dynamic testing efforts among the phases of secure software development life cycle [139].

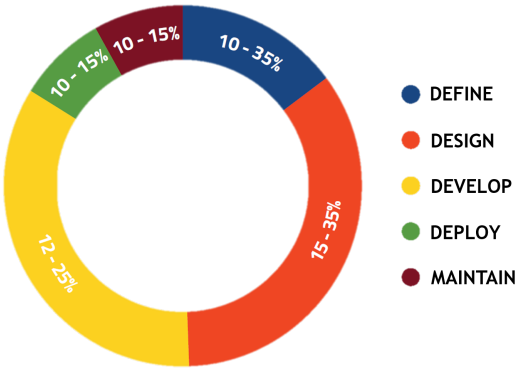

FIGURE 3.7: Proportion of test effort in secure software development life cycle [139].

3.3.2 Security Testing Techniques

In [140], a classification according to the test basis within the secure software development life cycle (Fig.3.8) is presented:

1. **Model-based security testing (MBST):** It is a Model-based testing (MBT) approach that validates software system requirements related to security properties. It combines security properties like confidentiality, integrity, availability, authentication, authorization, and non-repudiation with a model of the System Under Test (SUT) and identifies whether the specified or intended security features hold in the model.

2. **Code-based testing and static analysis:** Many vulnerabilities can only be detected by looking at the code. Static analysis of the program code is an important part of any security development process as it allows detecting vulnerabilities at early stages of the development life cycle, where fixing vulnerabilities is comparatively cheap. This is also known as white-box testing.

3. **Penetration testing and dynamic analysis:** These testing techniques do not require access to the source code or other development artifacts of the application under test. Instead, the security test is conducted via interaction with the running software. This is also known as black-box testing.

4. **Security regression testing:** It is a combination of regression and security testing which ensures that changes made to a system do not harm its security and do not cause unintended effects, as unchanged parts and changed parts of the software behave as intended.

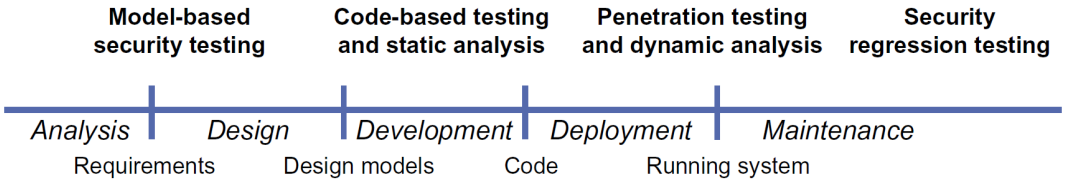

FIGURE 3.8: Security testing techniques in the secure software development life cycle [140].

Table 3.6 describes the testing techniques that are associated to each step in the development life cycle [81, 117, 140–142].

TABLE 3.6: Testing techniques according to the step in the development life cycle step.

Software Development Life-Cycle Step	Testing Technique	Description
Model-Based Security Testing	—-	Selected algorithms generate test cases automatically from a set of models of the system under test or of its environment.
Code-Based Testing and Static Analysis	Domain Testing	Only valid input.
	Error-Handling Testing	Properly process erroneous transactions.
	Log Review	Operate according to policies.
	Loop Testing	Exercise program loops.
	Mutation Testing	Using artificial defects, keeping them minimal.
	Path Testing	Satisfy coverage criteria for each logical path through the program.
	Recovery Testing	How well a system recovers from crashes, hardware failures, or other catastrophic problems.
	Statement Testing	Each statement in a program is executed at least once during program testing.
	Static Analysis	Sanity of the code, algorithm, or document. The software is not actually executed.
	Storage Testing	Store data files in the correct directories. Reserve sufficient space to prevent unexpected termination resulting from lack of space.
Penetrations Testing and Dynamic Analysis	Combinatorial Software Testing	Identify and test all unique combinations of software inputs.
	Denial of Service Testing (DoS)	Make the system unavailable by overloading it, or exploiting an existing vulnerability.

TABLE 3.6: Testing techniques according to the step in the development life cycle step.

Software Development Life-Cycle Step	Testing Technique	Description
	Destructive Testing	Understand a SUT's structural performance or material behaviour under different loads. Carried out to the SUT's failure.
	Endurance Testing	Check for memory leaks or other problems that may occur with prolonged execution.
	Fault Injection	Physically exploit an existing vulnerability through voltage, or temperature variations or laser light exposure.
	Functional Testing	Check test cases based on the specifications of the software SUT.
	Fuzzing	Sumit random, malformed data as inputs into software programs to determine if they will crash.
	Invasive Attack	The aim is to capture information stored in memory areas, or data flowing through the data bus, registers. Such techniques are also used to disconnect circuits, to override sensors, or to defeat blown fuse links. The physical properties of the SUT are irreversibly modified.
	Network Scanning	Identify all host potentially connected to an organization's network services operating on those hosts, and the specific application running the identified service.
	Password Cracking	Identify weak passwords.

TABLE 3.6: Testing techniques according to the step in the development life cycle step.

Software Development Life-Cycle Step	Testing Technique	Description
	Penetration Testing	Test from the outside in a setup that is comparable to an actual attack from a malicious third party. Limited information.
	Security Scanning	Identify network and system weaknesses, and later provide solutions for reducing these risks.
	Side-channel Attack	Use existence of physically observable phenomenons caused by the execution of computing tasks in present microelectronic devices (power consumption, electromagnetic emision or time) to extract information.
	Test Vector Leakage Assessment (TVLA)	Know whether the SUT leakages information through any side channel. TVLA cannot determine the magnitude of the leakage o whether it is exploitable.
	Social Engineering Testing	Querying personnel.
	Vulnerability Scanning / Testing	Identify hosts and host attributes (operating systems, applications, open ports), but it also attempts to identify vulnerabilities rather than relying on human interpretation of the scanning results.
Security Regression Testing	Age testing	Evaluate the SUT'sability to perform in the future.
	Comparison Testing	compares the product strengths and weaknesses with previous versions or other similar products.

TABLE 3.6: Testing techniques according to the step in the development life cycle step.

Software Development Life-Cycle Step	Testing Technique	Description
	Parallel Testing	Check that the new application which has replaced its older version has been installed and is running correctly.
Others	A/B Testing	Determine whether a proposed change is more effective than the current approach.
	Compatibility Testing	Check for backward compatibility.
	Compliance Testing	Compliance with internal or external standards
	Configuration Testing	Find minimal and optimal configuration of hardware and software, and the effect of adding or modifying resources such as memory, disk drives and CPU.
	Conformance Testing	SUT conforms to the specification on which it is based.
	Dependency Testing	Examine an application's requirements for pre-existing software, initial states and configuration in order to maintain proper functionality.
	File Integrity Checkers	Check whether the SUT computes and stores a checksum for every guarded file and establishes a database of file checksums.
	Formal verification Testing	Proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.

TABLE 3.6: Testing techniques according to the step in the development life cycle step.

Software Development Life-Cycle Step	Testing Technique	Description
	Posture Assessment	Security scanning, Ethical Hacking and Risk Assessments to show an overall security posture of an organization.
	Services Probing	Active examination of the application listening behind the service.
	Smoke Testing	Preliminary testing to reveal simple failures severe enough to, for example, reject a prospective software release.
	SQL Injection	Database errors
	Stability Testing	Check the quality and how the SUT behaves in different environmental parameters like temperature, voltage etc.
	Stress Testing	Evaluate the SUT at or beyond the limits of its specified requirements.

3.3.3 Test Case Prioritization

With the increasing size of software systems and the continuous changes that are committed to the software's codebase, regression testing has become very expensive for real-world software applications. Test case prioritization is a classic solution in this context. Test case prioritization is the process of ranking existing test cases for execution with the goal of finding defects sooner. It is useful when the testing budget is limited and one needs to limit their test execution cost, by only running top n test cases, according to the testing budget. There are many heuristics and algorithms to rank test cases [143].

3.3.4 Gaps Detected and Critics to the Current Status

Even though some described test techniques can be applied to ES, most of them were developed for high-level systems. This means that those that cannot be applied directly —penetration testing, fuzzing, or A/B testing— should be adapted [144].

3.4 Vulnerability Analysis

This section presents the current status of vulnerability analysis. This review aims to find similar approaches in the literature, including the current standard.

3.4.1 Vulnerability Analysis in Security Standards

This review is focused on how standards conduct vulnerability analysis, the use of metrics, their management of the life cycle of the device, the techniques that they propose, and the security evaluation of both software and hardware.

3.4.1.1 ISA/IEC 62443

The ISA/IEC 62443-4-1 technical document is divided into eight practices, which specify the secure product development life cycle requirements for both the development and the maintenance phases [89]. "Practice 5 - Security verification and validation testing" (SVV) section of this document specifies that a process shall be employed to identify and characterize potential security vulnerabilities in the product, including known and unknown vulnerabilities [53, 56]. Two requirements in Practice 5 are in charge of the task of analyzing vulnerabilities, as follows [89]:

- **Requirement SVV-3. Vulnerability Testing**. This requirement states that a process shall be employed to perform tests that focus on identifying and characterizing potential and known security vulnerabilities in the product (*i.e.*, fuzz testing, attack surface analysis, black box known vulnerability scanning, software composition analysis, and dynamic runtime resource management testing).
- **Requirement SVV-4. Penetration Testing**. This requirement states that a process shall be employed to identify and characterize security-related issues via tests that focus on discovering and exploiting security vulnerabilities in the product (*i.e.*, penetration testing).

Although the ISA/IEC 62443-4-1 document considers the possibility of analyzing and characterizing the vulnerabilities of an industrial component, it does not propose a technique to perform this task, but instead refers to other standards for vulnerability handling processes [145]. In addition, it does not indicate how the data obtained from the analysis should be interpreted, and it does not define metrics or reference values for the current state of compliance with the requirement. Finally, it does not take into account neither the dependencies among the assets of the industrial component (dependency trees), nor their evolution of the number of vulnerabilities over time.

3.4.1.2 Common Criteria

The CC defines five tasks in the Vulnerability Assessment class, which manage the deepness of the vulnerability assessment. The higher the EAL to be achieved, the greater the number of tasks in the list to be performed [55]:

1. Vulnerability survey,
2. Vulnerability analysis,
3. Focused vulnerability analysis,

4. Methodical vulnerability analysis, and
5. Advanced methodical vulnerability analysis.

Every task checks for the presence of publicly known vulnerabilities. Penetration testing is also performed. The main difference among the five levels of vulnerability analysis described here is the deepness of the analysis of known vulnerabilities and the penetration testing.

The CC scheme defines the general activities, but it does not specify how to perform them, therefore no technique for analyzing vulnerabilities is proposed. The evaluator decides the most appropriated techniques for each test in each scenario and for each device, which adds a large degree of subjectivity to the evaluation. Furthermore, dependencies among vulnerabilities and assets are not considered in the analysis. Moreover, the CC does not define a procedure to manage the life cycle of the device. In other words, when updated, the whole device has to be reevaluated [124, 146–148]. Finally, although the usage of metrics is encouraged by the CC, it does not propose any explicitly defined metric to be used during the evaluation.

3.4.2 Vulnerability Analysis in the Literature: Directed Graphs

Directed graphs, and structures derived from them (*e.g.*, attack graphs), are graphs in which the edges have a defined direction. This direction is usually indicated with an arrow on the edge. They are mainly used for vulnerability analysis in computer networks. Vulnerability analysis is a key step towards the security evaluation of a device. Consequently, many research efforts have been focused on solving this issue. In this section, the most relevant works related to vulnerability analysis are reviewed.

Homer *et al.* [149] present a quantitative model for computer networks that objectively measures the likelihood of a vulnerability. Attack graphs and individual vulnerability metrics, such as CVSS, and probabilistic reasoning are applied to produce a sound risk measurement. However, the main drawback is that their work is only applicable to computer networks. Although they propose new metrics based on the CVSS for probabilistic calculations, they do not integrate standards such as CAPEC to enhance their approach centered on possible attacks and privilege escalation. They also fail to establish a relationship among existing vulnerabilities, and they fail to obtain the source problem causing each vulnerability.

Zhang *et al.* [150, 151] developed a quantitative model that can be used to aggregate vulnerability metrics in an enterprise network based on attack graphs. Their model measures the likelihood that breaches can occur within a given network configuration, taking into consideration the effects of all possible interplays between vulnerabilities. This research is centered on computer networks, using attack graphs. Although the proposed model is capable of managing shared dependencies and cycles, only CVSS-related metrics are used. Moreover, this model assumes that the attacker knows all the information in the generated attack graphs. Finally, the method that they proposed for the aggregation of metrics is only valid for attack graphs, and is not valid for vulnerability analysis.

George *et al.* [152] propose a graph-based model to address the security issues in Industrial IoT (IIoT) networks. Their model is useful because it represents the relationships among entities and their vulnerabilities, serving as a security framework for the

risk assessment of the network. Risk mitigation strategies are also proposed. Finally, the authors discuss a method to identify the strongly connected vulnerabilities. However, the main drawback of this work is that each node of the generated attack graph represents a vulnerability instead of representing a device or an asset of that device. This leads to a loss of information in the analysis, because there is no way to know which vulnerability belongs to which device. Moreover, these methods need to know the relationships among present vulnerabilities in the devices. This information is not trivially obtained, and a human in the loop is needed. The proposals of [153] and [154] follow a similar graph-based approach to study the effects of cascade failures in the power grid, and a subway network.

Poolsappasit *et al.* [155] propose a risk management framework using Bayesian networks that enables a system administrator to quantify the chances of network compromise at various levels. The authors are able to model attacks on the network, and also to integrate standardized information of the vulnerabilities involved, such as their CVSS score. Although their proposed model lends itself to dynamic analysis during the deployed phase of the network, these results can only be applied to computer networks. Meanwhile, the prior probabilities that are used in the model are assigned by network administrators, and hence are subjective. The proposed model also has some issues related to scalability.

Muñoz-González *et al.* [156] propose the use of efficient algorithms to make an exact inference in Bayesian attack graphs, which enables static and dynamic network risk assessments. This model is able to compute the likelihood of a vulnerability, and can be extended to include zero-day vulnerabilities, attacker's capabilities, or dependencies between vulnerability types. Although this model is centered on studying possible attacks, it fails to integrate standards (such as CAPEC) that are related to attack patterns. Moreover, the generated graphs are focused on privilege escalation, trust, and users, rather than including information about vulnerabilities and the analyzed device.

Liu *et al.* [157] carry out a detailed assessment of vulnerabilities in IoT-based critical infrastructures from the perspectives of applications, networking, operating systems, software, firmware, and hardware. They highlight the three key critical infrastructure IoT-based cyber-physical systems (*i.e.*, smart transportation, smart manufacturing, and smart grid). They also provide a broad collection of attack examples upon each of the key applications. Finally, the authors provide a set of best practices and address the necessary steps to enact countermeasures for any generic IoT-based critical infrastructure system. Nevertheless, their proposal is focused on attacks and countermeasures, and it leaves aside the inner analysis of the targets. Continuous evaluation over time is not considered in this proposal, and no enhancements of the development process are generated.

Hu *et al.* [158] Hu et al. propose a network security risk assessment method that is based on the improved hidden Markov model (I-HMM). The proposed model reflects the security risk status in a timely and intuitive manner, and it detects the degree of risk that different hosts pose to the network. Although this is a promising approach, it is centered on computer networks and is at a higher abstraction level. No countermeasure or enhancement in the development process is proposed or generated.

Zografopoulos *et al.* [159] provide a comprehensive overview of the Cyber-Physical System (CPS) security landscape, with an emphasis on Cyber-Physical Energy Systems (CPES). Specifically, they demonstrate a threat modeling methodology to accurately represent the CPS elements, their interdependencies, as well as the possible attack entry points and system vulnerabilities. They present a CPS framework that is designed to delineate the hardware, software, and modeling resources that are required to simulate the CPS. They also construct high-fidelity models that can be used to evaluate the system's performance under adverse scenarios. The performance of the system is assessed using scenario-specific metrics. Meanwhile, risk assessment enables system vulnerability prioritization, while factoring the impact on the system's operation. Although this research work is comprehensive, it is focused on enhancing the existing adversary and attack modeling techniques of CPSs of the energy industry. Moreover, their model does not integrate the internal structure of the target of evaluation, and it does not take both software and hardware into account for the evaluation. Continuous evaluation over time is not considered. Finally, they do not propose countermeasures, or any kind of mechanism to enhance the security or the development of the CPSs.

3.4.3 Gaps Detected and Critics to the Current Status

Most of the works reviewed here are more focused on modeling threats and attacks. They do not propose solutions to protect CPSs, enhancing their development, or manage their update throughout their whole life cycle. It is worth noting that they are still more focused on software evaluation, while hardware is usually neglected in their proposals.

As shown in this review, most of the research has adopted dependency trees, attack graphs, or directed graphs as the main tool to manage and assess vulnerabilities in computer networks. Graphs are an efficient technique to represent the relationships between entities, and they can also effectively encode the vulnerability relations in the network. Furthermore, the analysis of the graph can reveal the security-relevant properties of the network. For fixed infrastructure networks, graphical representations, such as attack graphs, are developed to represent the possible attack paths by exploiting the vulnerability relationships. For these reasons, vulnerability analysis techniques based on directed graphs are frequently found in the literature [160]. However, despite their potential, these analysis techniques have been relegated to vulnerability analysis in computer networks. Graph-based analysis has rarely been applied to industrial components.

3.5 Metric Aggregation

System security quantification is not an easy task [75]. There exist both a lack of consensus and standardization around security metrics [71–74, 90, 91, 161]. For this reason, research efforts keep aiming to unify this field [162].

Among these efforts, the Common Vulnerability Scoring System (CVSS) is a widely extended standard for vulnerability quantification [67]. CVSS is a public framework that provides a standardized method for assigning quantitative values to security vulnerabilities according to their severity. A CVSS score is a decimal number in the range [0, 10][1] [65].

The CVSS is aimed to quantify the severity of vulnerabilities in individual and specific software items, however the majority of systems are actually a composition of simpler isolated items with different interdependencies. This situation highlights one of the biggest problems related to security quantification [163], the difficulty to really measure the global security state of a composite system. To do so, it would be necessary to aggregate each individual CVSS value into a global one in a consistent and coherent way.

The official CVSS documentation does not propose any kind of aggregation mechanism, and nowadays, there is no standardized method [164]. In addition to this, previous research works do not usually integrate contextual or interdependency information about the vulnerabilities to update the CVSS. This means that aspects such as whether affected functionalities, the environment of deployment, or the existence of exploits are usually neglected.

Context is a critical aspect to integrate in the aggregation process. This can be illustrated using a device implementing multiple functionalities as an example. To perform those functionalities, usually it will contain assets that implement those functionalities. But depending on the context where the device is deployed, some of its functionalities might not be needed. So the assets implementing unused functionalities would be disabled, and therefore, their vulnerabilities could not be exploited. It can also be the case that the asset implementing a functionality is simply inaccessible, so it could not also be exploited.

Nowadays, there is no widely-accepted method to aggregate CVSS values for software composition. All of them can be classified into one of the following categories [165, 166]: (1) Arithmetic Aggregation, (2) Attack Graph-based Aggregation, and (3) Bayesian Network-based Aggregation.

3.5.1 Arithmetic Aggregation

This method uses arithmetic operations to aggregate the values [167–170]. Common examples of this approach are taking the maximum of the CVSS values, their arithmetic mean, or a combination of them. For example, Heyman *et al.* [167], proposed an algorithm to aggregate CVSS values in dependency graph that is based on taking the maximum value in each case, according to certain conditions.

Although their simplicity makes them suitable for initial approximations, their results can be biased in two ways:

[1] The latest version at the time this paper was written is version 3.1.

1. **Exploitable by quantity:** When a system poses several vulnerabilities that by their own are not critical and cannot be exploited, they can sum up to an aggregated value of a high impact vulnerability (overfitting). This can happen when multiple simple mechanisms are combined as the aggregation algorithm.
2. **Exploitable by criticality:** When there exist a critical vulnerability, the whole system will be usually classified as critical. Nevertheless, that vulnerability might not be exploitable, nor being affecting the functionality of the system. This is specially common when using the maximum as the aggregation algorithm.

Since these metrics are defined over a set, they do not depend on where vulnerabilities are located, or how they are related to each other. These naive approaches lead to misleading results because they ignore causal relationships between vulnerabilities.

3.5.2 Attack Graph-based Aggregation

This approach models the relationships between vulnerabilities using attack graphs, converting CVSS scores into probabilities [171–176]. In this way, both the CVSS value and the place of the vulnerability in the whole graph are taken into account.

Cheng *et al.* in [165] proposed a graph-based aggregation method that uses the underlying metrics of CVSS, where the dependency relationships between vulnerabilities are usually visible. As the center of the aggregation algorithm, they use the product of the CVSS used as probabilities, also known as the join probability of both vulnerability.

The main drawback with these approaches is that the relationship between individual vulnerabilities cannot be obtained straightforwardly from existing databases. This means that establishing a relation between two vulnerabilities implies that they can be chained during an attack, which is not always obvious. Moreover, factors such as exploitability of the vulnerabilities, or existing exploits are not taken into account.

3.5.3 Bayesian Network-based Aggregation

Going a step further, these methods integrate the conditional relationship between vulnerabilities, modeling them using Bayesian networks [177–179]. Poolsappasit *et al.* [178] proposed a CVSS aggregation framework using Bayesian networks. They used the Bayesian probability factorization formula as the aggregation mechanism:

$$p(x) = \prod_{i=0} p(x_v | x_{pa(v)})$$

Bayesian network-based approaches have to deal with establishing the relationships between the vulnerabilities, but also with the calculation of conditional probabilities, that have to be usually estimated. As the previous ones, these techniques do not integrate information about how functionality if affected by existing vulnerabilities, or the possibility to actually exploit them.

3.5.4 Gaps Detected and Critics to the Current Status

The aggregation methods found in the literature have evolved to address for an increasing complexity in the environment. So they evolved from simple averages,

to more complex models based on Bayesian networks. Nevertheless, their main drawback is not related to how they work, but to the data they need to work: CVSS scores where design, so most of the vulnerabilities have a value higher than 6.0. In fact, the current weighted average CVSS score is 6.5. This, together with the fact that any given device will have a high amount of vulnerabilities, makes the aggregated result to tend easily to 10. This is not desirable as the results are no longer meaningful.

Finally, these methods do not take the context or their functionality into account. In other words, a device could carry out several functionalities while only one of those functionalities could depend on a library such as OpenSSL. If there would be an OpenSSL vulnerability with a CVSS value of 10, and that particular device would be used to accomplish other functionalities not depending on OpenSSL, that would not mean that the CVSS value of the whole device would be 10. The resulting CVSS value for the whole device should be adjusted according to the used or accessible functionalities. But it can be the case that that vulnerability only affects the specific functions of OpenSSL that the devices does not use. Furthermore, it can happen that the device is indeed affected by the vulnerability, but its presence does not have any repercussion in its functionality.

Chapter 4

How to Quantify the Security Level of Embedded Systems? A Taxonomy of Security Metrics

This chapter covers the existing evaluation methodologies and security metrics in order to classify the existing security metrics based on a new taxonomy. In order to propose a new taxonomy, existing evaluation methodologies and standards have been analyzed to check whether they propose or define any kind of security metric. Then, we perform a systematic review of the literature, gathering existing security metrics. We highlight the importance of security metrics, describe the existing metrics proposed in international standards, and analyzes the proposed taxonomies. Finally, we classify them by developing a new taxonomy.

4.1 Features of Comprehensive Security Metrics for Embedded System

Some authors have tried to characterize which criteria should meet a comprehensive metric. Among their efforts, three main ideas can be found in the literature: (1) the SMART criterion [94], (2) the PRAGMATIC criterion [95] and (3) the characteristics identified in the work carried out by Savola [77].

Figure 4.1 shows the mapping among the three different concepts explained by each criterion. This mapping was carried out using the definition and the data given by the authors, to find the common points among them[1].

The approach proposed by Savola classifies properties into 19 items; whereas the other two criteria have nine and five criteria, resulting in a higher mapping ratio. With this figure it is not possible to deduce which property is more important in any context, but it is possible to conclude that cost, accuracy, genuineness and repeatability are the most common properties in all three criteria.

It should be noted that none of these criteria explicitly mentions that metrics should be quantitative. This may be because, depending on the context, a nominal scale could be used instead. Some authors explicitly avoid this approach and prefer metrics to be quantitative [97]. In this mapping, many criteria have associated the cheap

[1]This figure can be better visualized at `https://embeddedsecuritymetrics.github.io/`.

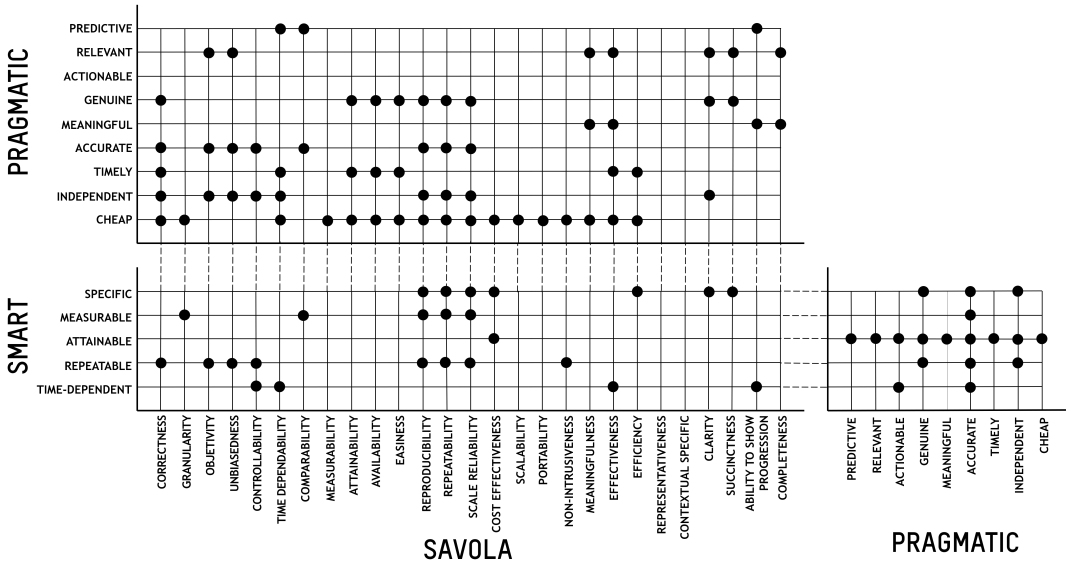

FIGURE 4.1: Mapping of criteria for identifying a good metric, according to the SMART, and PRAGMATIC criteria; and the survey carried out by Savola. Each black circle represents a common point between each one of the properties.

feature. This is because there are properties that directly affect the cost of computing a metric (the more properties it meets, the more expensive it is).

On the other hand, there are two properties identified by Savola (*i.e.*, contextual specific and representativeness) that do not have any correspondence with the rest of the criteria.

In the context of ES, we propose that the quality criteria that Security Metrics (SM) should address are (in no particular order):

- **Comparability:** SM should support comparison of the targets that they represent.
- **Cost effectiveness:** measurement gathering and approaches should be cost-effective.
- **Measurability:** SM are capable of having dimensions, quantity or capacity ascertained in the ES.
- **Repeatability:** The same results are achieved if a measurement is repeated in the same context, with exactly the same conditions.
- **Reproducibility:** The same results are achieved if a measurement is repeated in the same context, with exactly the same conditions, and different persons.

4.2 Selecting Metrics for Embedded Systems

There is neither a single system metric nor “one-perfect" set of metrics that could be suitable for every single case. The set of metrics that will be most suitable depends on multiple factors. For that reason, it is convenient to analyze whether the existing

metrics can be applied to ES. We carry out a systematic review of the state of the art to search for the currently available security metrics. The objective is to analyze to what extent the proposed metrics can be applicable to the security evaluation of ES. The details of this analysis are explained in the following points.

4.2.1 Search and selection strategy

The data sources used are online databases, conference proceedings and academic journals of IEEE Xplore, Elsevier, AMC Digital Library, Springer, along with Google Scholar search engine, Web Of Science and Scopus. The search terms included the keywords: "security", "metric", "measure", "evaluation" and "assessment", considering another synonyms. The inclusion criteria included:

- Security measurements or metrics are the main subjects.
- Surveys collecting security metrics were preferred.
- The paper is primarily related to measuring security.

The initial number of metrics analyzed was 531, obtained mainly from [78, 180–182]. For each one, the following data was collected:

- **Definition:** Definition of the metric given by the author.
- **Scale:** Nominal, ordinal, interval, ratio, absolute and distribution [78].
- **Scope:** User, software, hardware, device or organization.
- **Automation:** Automatic, semi-automatic or manual.
- **Measurement:** Static or dynamic measurement.

4.2.2 Filtering and exclusion criteria

In order to identify the subset of metrics that can be also applied to ES from that initial set, the collected metrics were filtered according to the following criteria:

- Organizational specific metrics.
- Network specific metrics.
- Metrics without a clear definition.
- Repeated metrics.

When the concept of a metric was found to be applicable with some modification, it was considered to be eligible.

4.2.3 Data extraction and interpretation

After filtering the data, the final count of metrics was 169. An interesting, albeit not very surprising fact, is that most of the metrics (77.5% of them) were related exclusively to software (*e.g.*, lines of code, number of functions and so on). On the other hand, only 0.6% of them were related exclusively to hardware (*e.g.*, side-channel vulnerability factor metric); and finally, 14.8% of them could be applied to both software and hardware (*e.g.*, the historically exploited vulnerability metric that measures the number of vulnerabilities exploited in the past). The remaining 7.1%

is focused on other aspects, such as user usability. This shows that there is a clear lack of hardware security metrics in the literature, and main efforts are centered in developing software-related metrics. It is worth commenting these metrics are not only valid for ES, but could also be applied to other systems.

All the data extracted from the articles, including the 169 security metrics, as well as the filtering and selection processes, are available at GitHub.

4.3 A New Taxonomy for Security Metrics

Although we discussed the most commonly used taxonomy for classifying security metrics in Chapter 3, to the best of our knowledge, there is no taxonomy that focuses on the specific assessment needs and limitations of ES. On the other hand, the taxonomies proposed in the literature are usually at a very high level, and therefore not applicable to ES. Thus, we propose a taxonomy that is based on which assets have to be protected in ES. Figure 4.2 shows the structure of the proposed taxonomy.

According to the research work in [183], embedded security cannot be addressed at a single level of abstraction; instead, it must be addressed at all abstraction levels. The main structure of the proposed taxonomy is built on three main blocks that aim to address all those abstraction levels: (1) Assets: elements of the ES that can be evaluated, as hardware, software, data, services, crypto keys, or communications; (2) Security Dimension: which attribute is to be protected (foundational requirements from the IEC 62443); and (3) Metric Properties: which are the characteristic parameters of the metric (such as automation, or computing cost). It is important to notice that a metric might cover more than one asset, and/or more than one security dimension, but at least, one value for each one have to be assigned (dashed line in Figure 4.2). Nevertheless, all values in metric properties are needed for each metric, as they represent the way the metric behaves, the cost associated to a metric and the information it returns (solid line in Figure 4.2).

The use of this taxonomy can be further explained using the Side-Channel Leakage (SCL) metric as an example. This metric uses statistical tests to measure if a device is prone to a side channel attack by comparing two sets of data; a random one and a fixed one. The SCL metric can be classified as follows:

- **Assets:** system hardware.
- **Security dimension:** data confidentiality.
- **Metric properties:**
  - A**utomation:** semi-automation.
  - **Scale:** nominal (yes/no).
  - **Measurement:** dynamic.
  - **Computing cost:** constant.

As can be seen, the main feature of the proposed taxonomy is that considers all assets in ES. Unlike other proposals in the literature that are at a high level (*e.g.*, enterprise level) this one is specifically designed to evaluate all aspects that comprise ES. Furthermore, it includes not only the CIA triad (*i.e.*, Confidentiality, Integrity

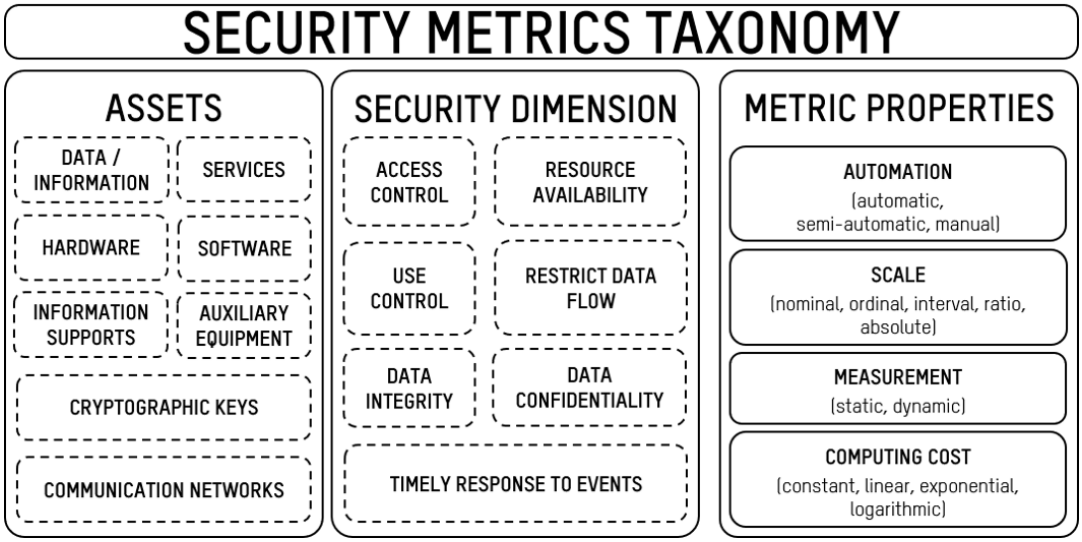

FIGURE 4.2: Taxonomy structure for security evaluation metrics for ES. Dashed line indicates AND/OR condition, and solid line AND condition.

and Availability), but also other dimensions, such as access control, use control, restrict data flow and timely-response to events. These dimensions are foundational requirements in IEC 62443 that need to be evaluated in ES.

4.4 Conclusions and Future Work

Security metrics are not standardized, and they would help in quantifying the security level of ES, that is, in giving the degree of confidence in the effectiveness of protections and countermeasures. This research work proposes a new taxonomy for security metrics, which takes into account the different assets of ES. Since it is critical to identify proper and non-exhausting security metrics for this security assurance, a conceptual mapping of criteria for choosing the most appropriate metrics is given. On the other hand, the literature has been analyzed, and 169 security metrics have been identified. 77.5% of them are dedicated exclusively to evaluating software security, while only 0.6% of them are oriented to hardware evaluation. This difference clearly shows the wider adoption by the community of software-related security metrics.

As a future work, we propose to select and refine a reduced set of metrics from those that have been analyzed, based on the 5 criteria that we have proposed, and applied to a real ES. This will be the foundation for proposing a security assessment methodology specifically designed for ES, which will provide qualitative and quantitative results. These results will give trust to end-users, make it possible to monitor the evolution of the security level of a device over time, and to compare different versions or devices.

Chapter 5

A Novel Model for Vulnerability Analysis through Enhanced Directed Graphs and Quantitative Metrics

Chapter 4 presented an analysis of the current status of security metrics, both in the literature and in the standards. The analysis showed that even though there is a plethora of available metrics to use, existing standards and methodologies do not integrate any of them in their workflows. Nevertheless, they encourage their integration in the evaluation process. With this analysis, we have covered the current status of security metrics and evaluation methodologies.

In this chapter, we focus on other aspects of ESs: A methodology that enables the monitoring of the security of ESs during their whole life cycle. To achieve this, we developed a model for continuous vulnerability assessment to manage the security of existing industrial components during their whole lifespan to deal with emerging threats.

5.1 Proposed Approach

In this contribution, we propose an Extended Dependency Graph (EDG) model that performs continuous vulnerability assessment to determine the source and nature of vulnerabilities, and enhance security throughout the entire life cycle of industrial components. The proposed model is built on a directed graph-based structure, and a set of metrics based on globally accepted security standards. The proposed EDG model is intended to:

- Identify the root causes and nature of vulnerabilities.
- Extract new requirements and test cases.
- Support the prioritization of patching.
- Track vulnerabilities during the whole lifespan of industrial components.
- Support the development and maintenance of industrial components.

To accomplish this task, the proposed model comprises two basic elements: the model itself, which is capable of representing the internal structure of the system under test; and a set of metrics, which allow conclusions to be drawn about the origin, distribution, and severity of vulnerabilities. Both the model and metrics are very flexible and exhibit some properties that make them suitable for industrial components, and can also be applied to enhance the ISA/IEC 62443 standard.

The content in this section is distributed into four sections, namely:

1. **Model:** The proposed model is explained, together with the systems in which it can be applied and the algorithms that are used to built it.
2. **Metrics:** Metrics are a great tool to measure the state of the system and to track its evolution. The proposed metrics and their usage are described in this section.
3. **Properties:** The main features of the proposed model and metrics (*e.g.*, granularity of the analysis, analysis over time, and patching policy prioritization support) are described in detail.
4. **Applicability:** Even though the reviewed standards exhibit some gaps, the proposed model aims to serve as the first step towards generating a set of tools to perform a vulnerability analysis in a reliable and continuous way. This last section will discuss the requirements of the ISA/IEC 62443-4-1 that can be enhanced using our model.

5.1.1 Description of the Model

The proposed model in this research work is based on directed graphs, and requires knowledge of the internal structure of the device to be evaluated (i.e., the assets, both hardware and software, that comprise it and the relationships between them). This section defines the most basic elements that make up the model, the algorithms to build it for any give system, and its graphical representation.

Definition 5.1.1. A System Under Test (SUT[1]) is now represented by an Extended Dependency Graph (EDG) model $G = (\langle A, V \rangle, E)$ that is based on directed graphs, where A and V represent the nodes of the graphs, and E represents its edges or dependencies:

- $A = \{a_1, ..., a_n\}$ represents the set of assets in which the SUT can be broken down, where n is the total number of obtained assets. An asset a is any component of the SUT that supports information-related activities, and includes both hardware and software [107, 184, 185]. Each asset is characterized by its corresponding CPE identifier, while its weaknesses are characterized by the corresponding CWE identifier. In the EDG model, the assets are represented by three types of nodes in the directed graphs (*i.e.*, root nodes, asset nodes and cluster).
- $V = \{v_1, ..., v_q\}$ represents the set of known vulnerabilities that are present in each asset of A, where q is the total number of vulnerabilities. They are characterized by the corresponding CVE and CVSS values. In the EDG model, vulnerabilities are represented using two types of nodes in the directed graphs (*i.e.*, known vulnerability nodes and clusters).
- $E = \{e_{ij} | \forall i, j \in \{1, ..., n+q\}$ such that $i \neq j\}$ represents the set of edges or dependencies among the assets, and between assets and vulnerabilities. e_{ij} indicates that a dependency relation is established from asset a_i to asset a_j. Dependencies are represented using two different types of edges in the EDG (*i.e.*, normal dependency and deprecated asset/updated vulnerability edges).

In other words, the EDG model can represent a system, from its assets to its vulnerabilities, and its dependencies as a directed graph. Assets and vulnerabilities are represented as nodes, whose dependencies are represented as arcs in the graph. The information in the EDG is further enhanced by introducing metrics.

The EDG model of a given SUT will include four types of node and two types of dependency. The graphical representation for each element is shown in Table 5.1. Figure 5.1 shows an example of a simple EDG and its basic elements. All of the elements that make up an EDG will be explained in more detail below:

5.1.1.1 Types of Node

The EDG model uses four types of node:

- **Root nodes** represent the SUT,

[1]Following the denomination in the ISA/IEC 62443 standard [48], the SUT may be an industrial component, a part of an industrial component, a set of industrial components, a unique technology that may never be made into a product, or a combination of these.

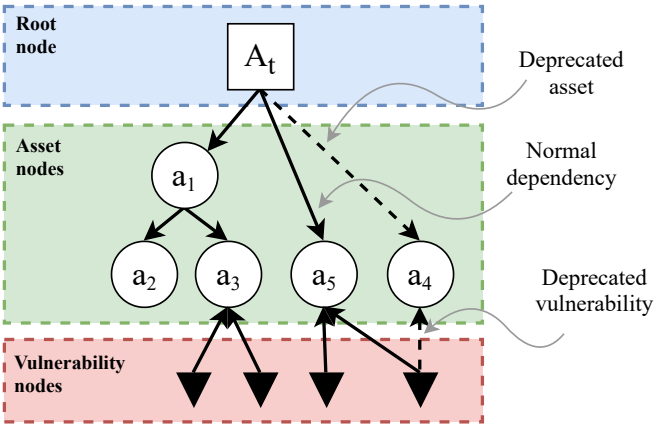

FIGURE 5.1: Basic elements of an EDG. Note that clusters are not displayed in this figure. For clusters, see Figure 5.3. For metrics definition, see Section 5.1.2.

TABLE 5.1: Overview of the information that is necessary to define each of the EDG elements.

SYMBOL	NOTATION	MEANING	VALUES
□	$A(t)$	Root Node / Device Node	$CPE_{current}$
○	$a(t)$	Asset Node	$CPE_{previous}, CPE_{current}, CWE_{a_i}(t)$
◌	$\underline{a}(t)$	cluster	$\{CPE_{previous}, CPE_{current}, CWE_{a_i}(t)\}, \{CVE_{a_i}(t), CVSS_{v_i}(t), CAPEC_{w_i}(t)\}, \{Dependencies\}$
▼	$v(t)$	Known Vulnerability Node	$CVE_{a_i}(t), CVSS_{v_i}(t), CAPEC_{w_i}(t)$
⟶	$e(t)$	Dependency Relation	—
- - →	$e(t)$	Updated Asset / Patched Vulnerability	—

- **Asset nodes** represent each one of the assets of the SUT,
- **Known vulnerability nodes** represent the vulnerabilities in the SUT, and
- **Clusters** summarize the information in a subgraph.

Root nodes (collectively, set G_R) are a special type of node that represent the whole SUT. Any EDG starts in a root node and each EDG will only have one single root node, with an associated timestamp (t) that indicates when the last check for changes was done. This timestamp is formatted following the structure defined in the ISO 8601 standard for date and time [186].

Asset nodes (collectively, set G_A) represent the assets that comprise the SUT. The EDG model does not impose any restrictions on the minimum number of assets that the graph must have. However, the SUT can be better monitored over time when there is a higher number of assets. Moreover, the results and conclusions obtained will be much more accurate. Nevertheless, each EDG will have as many asset nodes as necessary, and the break-down of assets can go as far and to as low-level as needed.

Each known vulnerability node will be characterized by the following set of values:

- $CPE_{current}$: Current value for the CPE. This points to the current version of the asset it refers to.
- $CPE_{previous}$: Value of the CPE that identifies the previous version of this asset. This will be used by the model to trace back all the versions of the same asset over time, from the current version, to the very first version.
- $CWE_{a_i}(t)$: Set of all the weaknesses that are related to the vulnerabilities present in the asset. The content of this list can vary depending on the version of the asset.

Figure 5.2 illustrates how the tracking of the versions of an asset using CPE works. On the one hand, version a_i is the current version of asset a. It contains its current CPE value, and the CPE of its previous version. On the other hand, a_2 and a_1 are previous versions of asset a. The last value of a_1 points to a null value. This indicates that it is the last value in the chain, and therefore the very first version of the asset a.

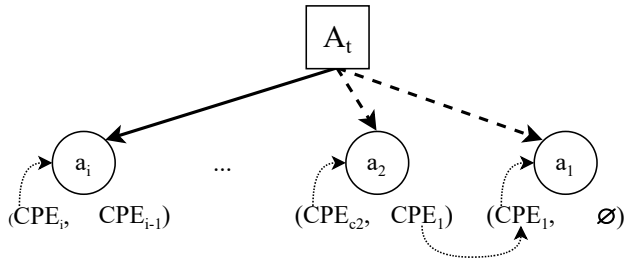

FIGURE 5.2: Tracking dependencies between the previous and current CPE values for asset a.

Known vulnerability nodes (collectively, set G_V) represent a known vulnerability present in the asset that it relates to. Each asset will have a known vulnerability node for each known vulnerability belonging to that asset. Assets alone cannot tell how severe or dangerous the vulnerabilities might be, so unique characterization of vulnerabilities is crucial [152].

To identify each known vulnerability node, each will be characterized by the following set of features (formally defined in 5.1.2:

- $CVE_{a_i}(t)$: This serves as the identifier of a vulnerability of asset a_i.
- $CVSS_{v_i}(t)$: This metric assigns a numeric value to the severity of vulnerability v_i. Each CVE has a corresponding CVSS value.
- $CAPEC_{w_i}(t)$: Each vulnerability (CVE) is a materialization of a weakness (CWE) w_i that can be exploited using a concrete attack pattern (CAPEC). In many cases, each CWE has more than one CAPEC associated. Consequently, this field is a set that contains all the possible attack patterns that can exploit the vulnerability that is being analyzed.

Clusters (collectively, set G_S) are a special type of node that summarizes and simplifies the information contained in a subgraph in an EDG. Figure 5.3 shows how the clusters work.

To identify each cluster, and to be able to recover the information that they summarize, each is characterized by the data that define each of the elements that they contain:

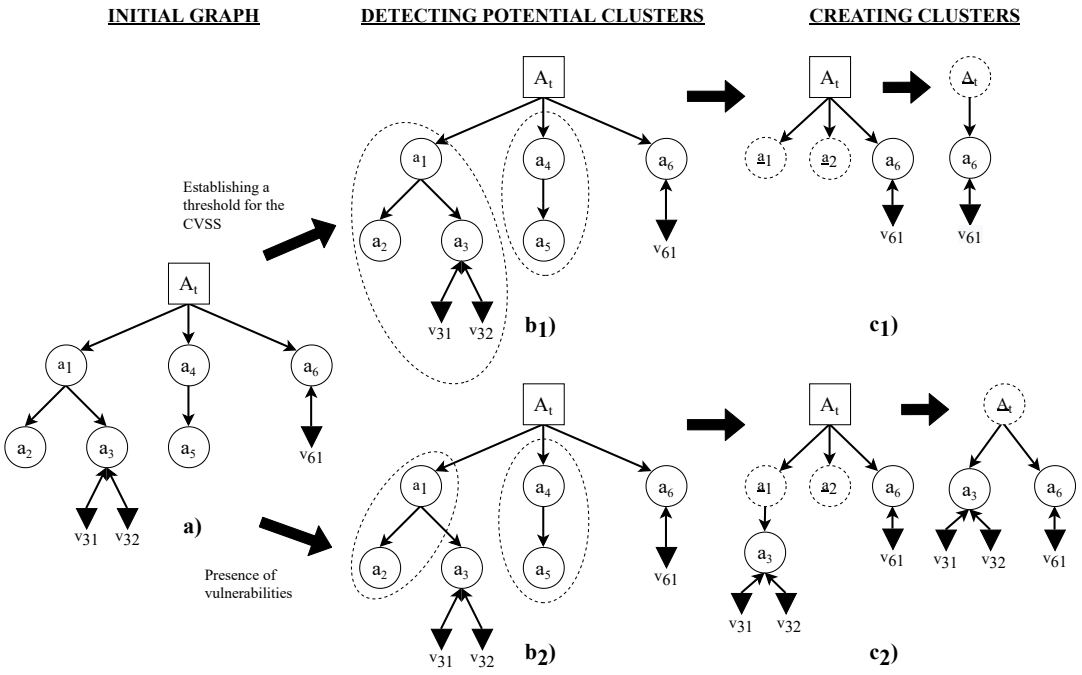

FIGURE 5.3: Creating clusters. Application of the two proposed criteria to the creation of clusters to simplify the graph: (1) Establishing a threshold to select which vulnerability stays outside the cluster (upper side). (2) Choosing the absence of vulnerability as the criterion to create clusters (lower side). The severity value (CVSS) for v_{211} and v_{212} is supposed to be lower than the establish threshold.

$\{CPE_{previous}, CPE_{current}, CWE_{a_i}(t)\}$, $(CVE_{a_i}(t), CVSS_{v_i}(t), \{CAPEC_{w_i}(t))\}$, and their dependencies.

Two types of criteria can be used to create clusters and to simplify the obtained graph (Figure 5.3):

1. **Absence of vulnerabilities**: Using this criterion, clusters will group all nodes that contain no associated vulnerabilities.
2. **CVSS score below a certain threshold**: With this criterion, a threshold for the CVSS scores will be chosen. Nodes whose CVSS score is less than the defined threshold will be grouped into a cluster.

5.1.1.2 Types of Edge

In the EDG model, edges plays a key role representing dependencies. Two types of edge can be identified:

- **Normal dependencies** relate two assets, or an asset and a vulnerability. They represent that the destination element depends on the source element. Collectively, they are known as set G_D.
- **Deprecated asset or patched vulnerability dependencies** indicate when an asset or a vulnerability is updated or patched. They represent that the destination

element used to depend on the source element. Collectively, they are known as set G_U.

The possibility of representing old dependencies brings the opportunity to reflect the evolution of the SUT over time. When a new version of an asset is released, or a vulnerability is patched, the model will be updated. Their dependencies will change then from a normal dependency to a deprecated asset or vulnerability dependency to reflect that change.

5.1.1.3 Conditions of Application of EDGs

The EDG model is applicable to SUTs that meet the following set of criteria:

- **Software and hardware composition:** In our approach, the model is created by means of a white-box analysis. The absence or impossibility to perform a white-box analysis limits the ability to create an accurate model. Some knowledge about the internal structure and code is expected. This information is usually only known by the manufacturer of the component, unless the component is publicly available or open-source. It should be also possible to break down the SUT into simpler assets to generate a relevant EDG.
- **Existence of publicly known vulnerabilities:** The EDG model focuses on known vulnerabilities. This is not critical because many industrial components use commercial or open-source elements. The SUT must be composed of assets for which public information is available. If the majority of SUT assets are proprietary, or the SUT is an *ad hoc* development that is never exposed, then the generated EDG will not evolve. Therefore, the analysis will not be relevant.

5.1.1.4 Steps to Build the Model

This section explains the process and algorithms that were used to build the corresponding EDG of a given SUT. The main scenarios that can be found are also described.

Before extracting useful information about the SUT, the directed graph associated with the SUT has to be built. This comprises several steps, which are described in the following paragraphs (see the flowchart in Fig. 5.4, and Fig. 5.5):

Step 1 — Break down the SUT into assets. For the model to work properly, it relies on the SUT being able to be broken down into assets. With this in mind, the first step involves obtaining the assets of the SUT, either software or hardware. In the CC, this process is called modular decomposition of the SUT [50]. Ideally, every asset should be represented in the decomposition process, but this is not compulsory for the model to work properly. Each one of the assets obtained in this step will be represented as an asset node. In this step, the dependencies among the obtained assets are also added as normal dependencies.

Step 2 — Assign a CPE to each asset. Once the assets and their dependencies have been identified, the next task is to assign the corresponding CPE identifier to each asset. If there is no publicly available information of a certain asset, and therefore, it does not have a CPE identifier, then it is always possible to generate one using the fields described in the CPE naming specification documents [59] for internal use in the model.

Step 3 — Add known vulnerabilities to the assets. In this step, the vulnerabilities ($CVE_{a_i}(t)$) of each asset are set. This is done by consulting public databases of known vulnerabilities [54, 64] looking for existing vulnerabilities for each asset. When a vulnerability is found, it is added to the model of the SUT, including its dependencies. If there were no known vulnerabilities in an asset, then the asset would become the last leaf of its branch. In this step, the corresponding value of the CVSS of each vulnerability is also added to the model.

Step 4 — Assign to each asset its weaknesses and possible CAPECs. After the vulnerabilities, the corresponding weaknesses to each vulnerability ($CWE_{a_i}(t)$) are added, along with the corresponding attack patterns ($CAPEC_{w_i}(t)$) for each weakness. If there is no known vulnerability in an asset, then there will be no weaknesses. Meanwhile, it would be possible to have a known vulnerability in an asset, but no known weakness or attack pattern for that vulnerability. Finally, more than one CAPEC can be assigned to the same weakness. Consequently, it would be common to have a set of possible CAPECs that can be used to exploit the same weakness. It is worth noting that not all of them could be applied in every scenario.

Step 5 — Computing Metrics and tracking the SUT. At this point, the EDG of the SUT is completed with all the public information that can be gathered. This last step is to calculate the metrics defined (for further information, see Section 5.1.2.), generating the corresponding reports, and tracking the state of the SUT for possible updates in the information of the model. This step is always triggered when the SUT is updated. This can imply that a new asset can appear, an old asset can disappear, an old vulnerability can be patched, or a new one can appear in the SUT. All of these scenarios will be reflected in the model as they arise during its life cycle.

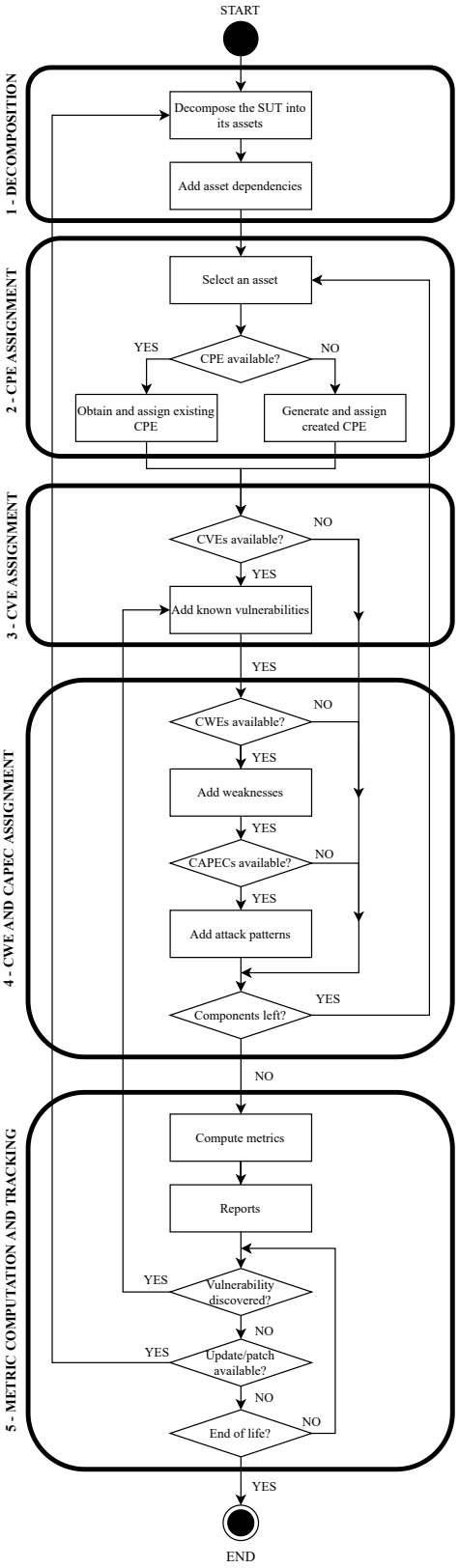

FIGURE 5.4: Algorithm to generate the initial EDG of a give SUT.

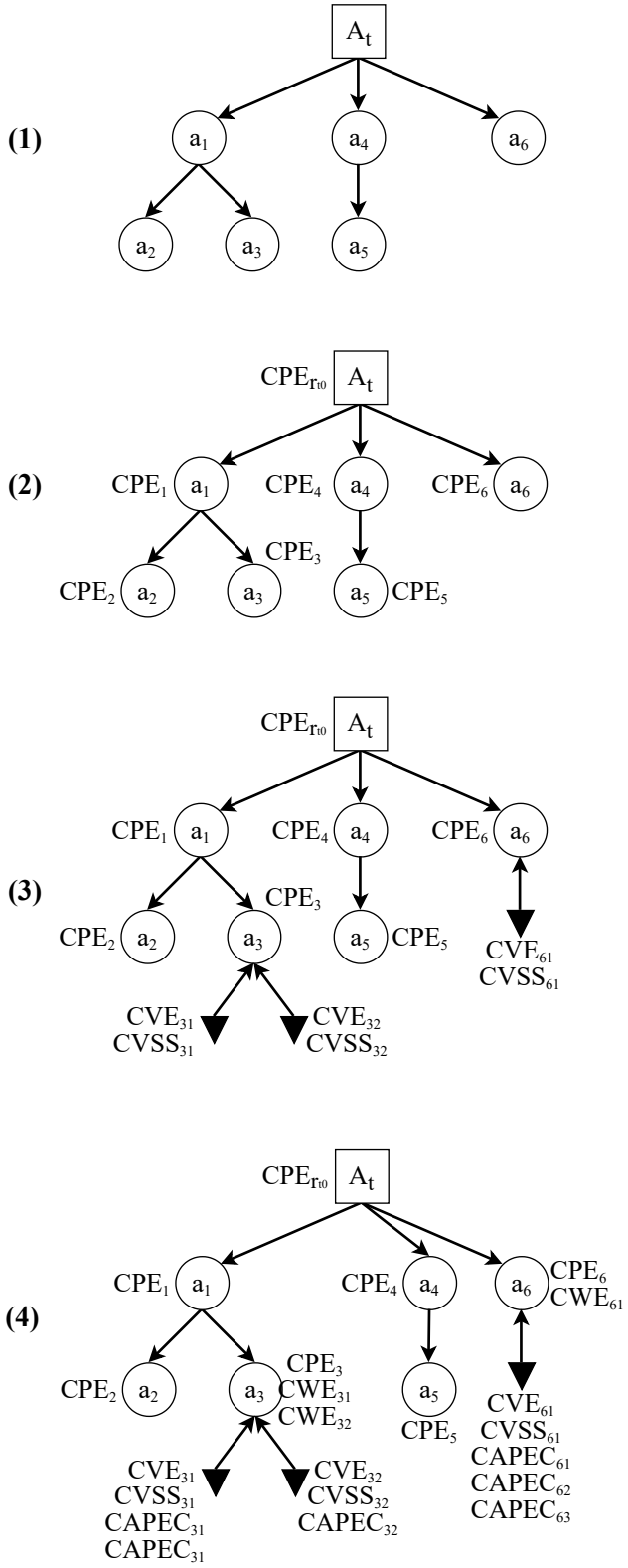

FIGURE 5.5: Example of the process of building the EDG model of a given SUT A.

5.1.2 Security Metrics

The EDG model that was proposed in the previous sections is by itself capable of representing the internal structure of the SUT, and it can display it graphically for the user. This representation not only includes the internal assets of the SUT, but it also captures their relationships, existing vulnerabilities, and weaknesses. Moreover, assets, vulnerabilities, and weaknesses are easily identified using their corresponding CPE, CVE, and CWE values, respectively. All together, this constitutes a plethora of information that the model can use to improve the development and maintenance steps of the SUT, enhance its security, and track its status during its whole life cycle. Metrics are a great tool to integrate these features into the model.

Metrics can serve as a tool to manage security, make decisions, and compare results over time. They can also be used to systematically improve the security level of an industrial component or to predict its security level in a future point in time.

In this section, the basic definitions that serve as the foundation of the metrics are described. Then, the proposed metrics are introduced to complement the functionality of the EDG model. The main feature of these metrics is that they all depend on time as a variable, so it is possible to capture the actual state of the SUT, track its evolution over time, and compare the results.

5.1.2.1 Basic Definitions

In this section, the basic concepts on which the definitions of the metrics will be based are formalized.

Definition 5.1.2. The set of all possible weaknesses at a time t is represented as $CWE(t)$, where

$$CWE(t) = \{cwe_1, ..., cwe_m\} \tag{5.1}$$

and m is the total number of weaknesses at time t. This set contains the whole CWE database defined by MITRE [62].

Definition 5.1.3. The set of all of the possible vulnerabilities at a time t is represented as $CVE(t)$ where

$$CVE(t) = \{cve_1, ..., cve_p\} \tag{5.2}$$

and p is the total number of vulnerabilities. This set contains the whole CVE database defined by MITRE [64].

Definition 5.1.4. The set of all possible attack patterns at a time t is represented as $CAPEC(t)$, where

$$CAPEC(t) = \{capec_1, ..., capec_q\} \tag{5.3}$$

and q is the total number of attack patterns at time t. This set contains the whole CAPEC database defined by MITRE [68].

Definition 5.1.5. The set of weaknesses of an asset a_i at a time t is defined as

$$\begin{aligned} CWE_{a_i}(t) = \{cwe_j | cwe_j \text{ is in the asset } a_i \text{ at time } t \wedge cwe_j \in CWE(t) \\ \wedge \forall k \neq j, cwe_j \neq cwe_k\} \end{aligned} \tag{5.4}$$

From this expression, the set of all the weaknesses of a particular asset throughout its life cycle is defined as

$$CWE_{a_i} = \bigcup_{t=1}^{T} CWE_{a_i}(t) \tag{5.5}$$

where $|CWE_{a_i}|$ is the total number of non-repeated weaknesses in its entire life cycle.

Definition 5.1.6. The set of vulnerabilities of an asset a_i at a time t is defined as

$$CVE_{a_i}(t) = \{cve_j | cve_j \text{ is in the asset } a_i \text{ at time } t \wedge cve_j \in CVE(t)\} \tag{5.6}$$

From this expression, the set of vulnerabilities of an asset throughout its entire life cycle is defined as

$$CVE_{a_i} = \bigcup_{t=1}^{T} CVE_{a_i}(t) \tag{5.7}$$

where $|CVE_{a_i}|$ is the total number of vulnerabilities in its entire life cycle.

Definition 5.1.7. The set of weaknesses of a SUT A with n assets at a time t is defined as:

$$CWE_A(t) = \bigcup_{i=1}^{n} CWE_{a_i}(t) \tag{5.8}$$

Definition 5.1.8. The set of vulnerabilities of a SUT A with n assets at a time t is defined as:

$$CVE_A(t) = \bigcup_{i=1}^{n} CVE_{a_i}(t) \tag{5.9}$$

Definition 5.1.9. The set of vulnerabilities associated to the weakness cwe_j and to the asset a_i at a time t is defined as:

$$CVE_{a_i|cwe_j}(t) = \{cve_k | cve_k \text{ associated to weakness } cwe_j \text{ and to asset } a_i \text{ at time } t\} \tag{5.10}$$

It is worth noting that CWE is used as a classification mechanism that differentiates CVEs by the type of vulnerability that they represent. A vulnerability will usually have only one associated weakness, and weaknesses can have one or more associated vulnerabilities [66].

Definition 5.1.10. The partition j of an asset a_i at time t conditioned by a weakness cwe_k is defined as

$$CVE_{a_i|cwe_k}(t) = \{cwe_l | cwe_l = cwe_k \wedge cwe_l \in CVE_{a_i}(t)\} \tag{5.11}$$

Definition 5.1.11. The partition j of the SUT A at time t conditioned by a weakness cwe_k is defined as

$$CVE_{A|cwe_k}(t) = \{cwe_l | cwe_l = cwe_k \wedge cwe_l \in CVE_A(t)\} \tag{5.12}$$

Definition 5.1.12. The set of attack patterns associated to a weakness w_i at a time t is defined as

$$CAPEC_{w_i}(t) = \{capec_j | capec_j \text{ can exploit weakness } w_i \text{ at time } t \wedge capec_j \in CAPEC(t)\} \tag{5.13}$$

.

Definition 5.1.13. The set of metrics that are defined in this research work based on the EDG model is defined as

$$M = \{m_1, ..., m_r\} \tag{5.14}$$

where r is the total number of metrics. This set can be extended, defining more metrics according to the nature of the SUT.

5.1.2.2 Metrics

This section will describe the metrics that were defined based on the EDG model and the previous definitions. Although it might seem trivial, the most interesting feature of these metrics is that they all depend on time. Using time as an input variable for the computation of the metrics opens the opportunity to track results over time, compare them, and analyze the evolution of the status of the SUT. Furthermore, some metrics take advantage of time to generate an accumulated value, giving information about the life cycle of the SUT. Table 5.2 shows all of the proposed metrics, their definition, and their reference values.

In addition to the metrics in Table 5.2, the model allows the definition of other types of metrics according to the analysis to be performed, and the nature of the SUT (*e.g.*, the vulnerability evolution function for SUT A up to time t for all vulnerabilities could be defined as the linear regression of the total number of vulnerabilities in each time t for SUT A, or using any other statistical model).

TABLE 5.2: Proposed metrics for the model.

	METRIC	DEFINITION	REFERENCE VALUE
VULNERABILITIES	$M_0(A) = \frac{\lvert CVE_A(t)\rvert}{n(t)}$	Arithmetic mean of vulnerabilities in the SUT A, where $n(t)$ is the number of assets in a SUT at a time t. M_0 shows how many vulnerabilities would be present in each asset if they were evenly distributed among the assets of the SUT. The result of M_0 can serve as a preliminary analysis of the SUT, related to the criticality of its state. From Equation 8.	$M_0 < 1$: The number of vulnerabilities is lower than the number of assets. $M_0 \geq 1$: Every asset has at least one vulnerability.
	$M_1(A,t) = \lvert CVE_A(t)\rvert$	Number of vulnerabilities in a SUT A at time t. From Equation 8.	Ideally, the values of M_1 should be zero (no vulnerability in A), but the lower the value of M_1, the better.
	$M_2(A) = \sum_{t=1}^{T} \lvert CVE_A(t)\rvert = \sum_{t=1}^{T} M_1(A,t)$	Number of vulnerabilities in a SUT A throughout its entire life cycle T. This metric computes the accumulated value of the number of vulnerabilities of a SUT throughout its entire life cycle. From Equation 8.	The lower the value of M_2, the better.
	$M_3(a_i,t) = \lvert CVE_{a_i}(t)\rvert$	Number of vulnerabilities in an asset a_k at time t The values of M_3 can be useful during a vulnerability analysis, or when performing a penetration test, to identify the asset with more vulnerabilities. From Equation 6.	Ideally, the value of M_3 should be zero.
	$M_4(a_k,t) = \frac{\lvert CVE_{a_k}(t)\rvert}{\sum_{i=1}^{n} \lvert CVE_{a_i}(t)\rvert}$	Relative frequency of vulnerabilities of the asset a_k at a time t. From Equation 6.	Ideally, the value of M_4 should be zero, or at least $M_4 \leq \frac{1}{n(t)}$, being $n(t)$ the number of assets in the SUT. This value can also be expressed as the percentage of vulnerabilities of asset a_i respect to the total number of vulnerabilities in the SUT, $M_4(a_k,t) = \frac{\lvert CVE_{a_k}(t)\rvert}{\sum_{i=1}^{n} \lvert CVE_{a_i}(t)\rvert} \cdot 100$
	$M_5(a_i, cwe_j, t) = \lvert CVE_{a_i \lvert cwe_j}(t)\rvert$	Multiplicity of weakness cwe_j of the asset a_i at a time t. This metric represents the number of times a weakness is present among the vulnerabilities of the asset a_i. This is possible because a vulnerability can have associated the same weakness as other vulnerabilities. From Equation From Equation 9.	Ideally, the value of M_5 should be zero, or at least, $M_5 \leq \frac{\lvert CVE_{A \lvert cwe_j}(t)\rvert}{n(t)}$, being $n(t)$ the number of assets in the SUT. The value of the metric could be further narrowed by assuming that cwe_j will be present in all but one asset, so $M_5 \leq \frac{\lvert CVE_{A \lvert cwe_j}(t)\rvert}{n(t)-1}$ to be in acceptable values.
	$M_6(A, cwe_j, t) = \lvert CVE_{A \lvert cwe_j}(t)\rvert$	Multiplicity of weakness cwe_j of the SUT A at a time t. This metric represents the number of times a weakness is present among the vulnerabilities of the SUT A. From Equation 11.	Ideally, the value of M_6 should be zero.
WEAKNESSES	$M_7(A,t) = \lvert CWE_A(t)\rvert$	Number of weaknesses in a SUT A at time t. From Equation 7.	Ideally, the value of M_7 should be zero (no weakness in A), but the lower the value of M_7, the better.
	$M_8(A) = \sum_{t=1}^{T} \lvert CWE_A(t)\rvert = \sum_{t=1}^{T} M_7(A,t)$	Number of weaknesses in a SUT A throughout its entire life cycle T. This metric computes the accumulated value of weaknesses of a SUT throughout its entire life cycle. From Equation 7	The lower the value of M_8, the better.

5.1.3 Properties

Together, the EDG model and the defined metrics exhibit a series of characteristics that make them suitable for vulnerability assessment. These properties represent an advantage over the techniques reviewed in the state of the art, including automatic inference of root causes, spatial and temporal distribution of vulnerabilities, and prioritization of patching, which will be described in the following subsections.

5.1.3.1 Automatic Inference of Root Causes

Each CWE natively contains information that is directly related to the root cause of a vulnerability. From this information, new requirements and test cases can be proposed.

5.1.3.2 Spatial and Temporal Distribution of Vulnerabilities

The key feature of the proposed model is the addition of the temporal dimension in the analysis of vulnerabilities. This makes it possible to analyze the location of the vulnerabilities both in space (in which asset) and time (their recurrence), which allows us to track the state of the device throughout the whole life cycle. This approach also enables a further analysis of the SUT, by updating data in the model, such as new vulnerabilities that are found or new patches that are released.

Each time that a new vulnerability is found, or an asset is patched (*i.e.*, via an update), the initial EDG is updated to reflect those changes. An example of this process can be seen in Fig. 5.6.

At time t_0, the initial graph of the SUT A is depicted in Fig. 5.6. Because there is no vulnerability at that time, this graph can be simplified using the cluster notation, with just a cluster containing all assets. At time t_1, a new vulnerability that affects the asset a_2 is discovered. At time t_2, the asset a_2 is updated. This action creates a new version of asset a_2, asset a_3. Because the vulnerability was not corrected in the new update, both versions contain the vulnerability that was initially presented in asset a_2. Finally, at time t_3, the asset a_3 is updated to its new version a_4, and the vulnerability is corrected.

This approach enables a further analysis of the SUT, including updated data, according to new vulnerabilities that are found or new patches that are released.

5.1.3.3 Patching Policies Prioritization Support

The proposed model is not only able to include known vulnerabilities associated with an asset, but it also provides a relative importance sorting of vulnerabilities by CVSS. Relying on the resulting value, it is possible to assist in the vulnerability patching prioritization process. Furthermore, the presence of an existing exploit for a known vulnerability can be also be taken into account, when deciding which vulnerabilities need to be patched first. A high CVSS value combined with an available exploit for a given vulnerability is a priority when patching.

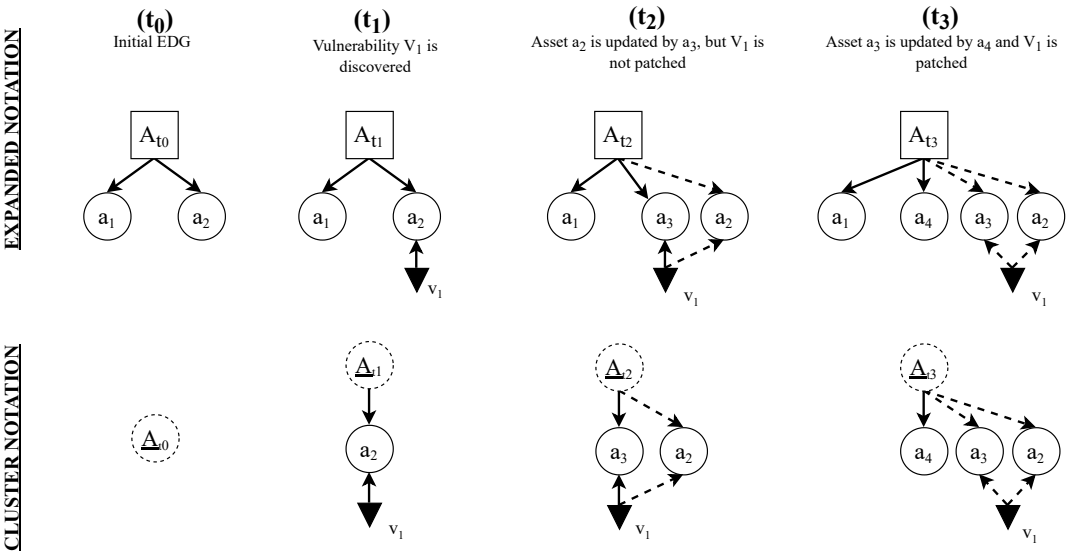

FIGURE 5.6: Representation of the temporal behavior in the graphical model using the two kinds of dependencies of the model. It is worth mentioning that these graphs could be further simplified by taking advantage of the cluster notation, as shown at the bottom of this figure.

5.1.4 Applicability in the Context of ISA/IEC 62443

In this section, the potential application of the proposed EDG model to the existing security standards is described. The proposed EDG model can be used isolated by itself, or in combination with other techniques that complement the analysis. In this sense, the EDG model can be used to enhance some task in the security evolution processes defined by security standards.

The ISA/IEC 62443-4-1 standard specifies 47 process requirements for the secure development of products used in industrial automation and control systems [89]. Thus, the EDG model was developed to enhance the execution of one of those requirements defined by the standard: the "SVV-3: Vulnerability testing" requirement, serving as a support for the execution of Practice 5 — Security Verification and Validation testing. According to the SVV-3 requirement, both known and unknown vulnerability analysis has to be performed. The EDG model proposed in this research work is intended to support the identification of known vulnerabilities, their dependencies, and the possible consequences of their propagation, yielding the opportunity to analyze them systematically. Nevertheless, more requirements of the ISA/IEC 62443 can be mapped to one or more of the metrics defined in this research work. Using this relationship, it is possible to apply the EDG model to enhance the analysis and review of the following requirements:

5.1.4.1 Security Requirements - 2: Threat Model (SR-2)

"A process shall be employed to ensure that all products have a threat model specific to the current development scope of the product. The threat model shall be reviewed and verified periodically" [89]. The proposed EDG model can serve as an abstraction of the threat model that has to be obtained. Moreover, the standard states that this threat model has to be reviewed periodically for updates. Given that the EDG of a given SUT evolves with every update, the threat model would be always up-to-date. Potential threats and their severity using the CVSS can also be analyzed with this proposal. Finally, these results can be used to enhance the risk assessment of the SUT.

5.1.4.2 Security Management - 13: Continuous Improvement (SM-13)

"A process shall be employed for continuously improving the secure development life cycle" [89]. The EDG model can be used to identify recurrent issues in the development of an industrial component, due to its ability to track the state of a SUT over time. Consider the scenario where a piece of code contains an unknown vulnerability. For example, this code can implement a communication protocol, or the generation of a cryptographic key. If this piece of code is recurrently integrated in many type of devices, then when they are released to the market, the end users can identify that vulnerability and report it to the product supplier. The EDG can reflect the presence of that vulnerability. If an EDG is done for each type of device, then this problem can be detected beforehand. Using the CWE, the root problem can be detected. With this information, new training and corrective actions can be proposed to avoid this issue.

5.1.4.3 Specification of Security Requirements - 5: Security Requirements Review (SR-5)

"A process shall be employed to ensure that security requirements are reviewed, updated, and approved" [89]. As before, taking advantage of the previous scenario, the information extracted from the generated EDG model can be used to propose new requirements or to update the existing requirements.

5.1.4.4 Security Verification and Validation Testing - 4: Penetration Testing (SVV-4)

"A process shall be employed to identify and characterize security-related issues via tests that focus on discovering and exploiting security vulnerabilities in the product" [89]. The EDG model facilitates the identification of possible entry points to the SUT when carrying out a penetration test. In addition, existing attack patterns (CAPEC) and weaknesses (CWE) can serve as a starting point to discover unknown vulnerabilities and exploits.

5.1.4.5 Management of Security-related Issues - 3: Assessing Security-related issues (DM-3)

"A process shall be employed for analyzing security-related issues in the product" [89]. When a new vulnerability is detected, end users will report it to the product suppliers. Then, the corresponding EDG model of that SUT will be updated to reflect that change. This information, in addition to that previously contained in the EDG, can be used to obtain the severity value of the discovered vulnerability using the CVSS. This also facilitates the identification of root causes, related security issues, or the impact.

Finally, the ISA/IEC 62443-4-2 document defines four types of components of an IACS (*i.e.*, software applications, embedded devices, host devices, network devices) [49]. The proposed model is capable of representing the inherent complexity of each of them.

TABLE 5.3: Mapping between the developed metrics and the requirements they refer in the ISA/IEC 62443.
SR (Security Requirements), SM (Security Management), SVV (Security Validation and Verification), DM (Management of Security-Related Issues).

METRIC	SR-2	SR-5	SM-13	SVV-4	DM-3
$M_0(A) = \frac{\lvert CVE_A(t)\rvert}{n(t)}$	■	■	■	■	■
$M_1(A,t) = \lvert CVE_A(t)\rvert$	■	■	■	■	■
$M_2(A) = \sum_{t=1}^{T} \lvert CVE_A(t)\rvert = \sum_{t=1}^{T} M_1(A,t)$	□	■	■	□	□
$M_3(A,t) = \lvert CVE_{a_i}(t)\rvert$	■	■	■	■	□
$M_4(a_k,t) = \frac{\lvert CVE_{a_k}(t)\rvert}{\sum_{i=1}^{n} \lvert CVE_{a_i}(t)\rvert}$	□	■	■	□	□
$M_5(a_i, cwe_j, t) = \lvert CVE_{a_i \lvert cwe_j}(t)\rvert$	■	■	■	■	□
$M_6(A, cwe_j, t) = \lvert CVE_{A \lvert cwe_j}(t)\rvert$	■	■	□	■	■
$M_7(A,t) = \lvert CWE_A(t)\rvert$	■	□	□	■	■
$M_8(A) = \bigcup_{t=1}^{T} \lvert CWE_A(t)\rvert = \bigcup_{t=1}^{T} M_7(A,t)$	□	■	■	□	□

5.2 Real Use Case: General Analysis of OpenPLC

In this section, the EDG model and the proposed metrics will be applied to perform a vulnerability assessment of the OpenPLC project, which will be the SUT. In the subsections, we will assess the three available versions of the OpenPLC project. For each, the EDG model will be obtained, and the proposed metrics will be applied to draw conclusions about the vulnerability status of each version.

OpenPLC is the first functional standardized open source Programmable Logic Controller (PLC), both in software and hardware [187]. It was mainly created for research purposes in the areas of industrial and home automation, Internet of Things (IoT), and SCADA. Given that it is the only controller that provides its entire source code, it represents an engaging low-cost industrial solution — not only for academic research but also for real-world automation [188,189].

5.2.1 Structure of OpenPLC

The OpenPLC project consists of three parts:

1. **Runtime:** It is the software that plays the same role as the firmware in a traditional PLC. It executes the control program. The runtime can be installed in a variety of embedded platforms, such as the Raspberry Pi, and in Operating Systems (OSs) such as Windows or Linux. Industrial Modbus slave devices can be attached to expand the number of inputs and outputs.
2. **Editor:** An application that runs on a Windows or Linux OS that is used to write and compile the control programs that will be later executed by the runtime.
3. **HMI Builder:** This software is to create web-based animations that will reflect the state of the process, in the same manner as a traditional HMI.

When installed, the OpenPLC runtime executes a built-in webserver that allows OpenPLC to be configured and new programs for it to run to be uploaded.

5.2.2 Initial Scenario

The OpenPLC project is constituted by three different versions [190–192] as can be seen in Table 5.4. For this research work, Ubuntu Linux was selected as the platform to install the OpenPLC runtime. Ubuntu Linux provides comprehensive documentation, previous versions are accessible, and software dependencies can easily be obtained.

To make the analysis of OpenPLC fair, a contemporary operating system was selected for each of the OpenPLC versions, according to the version of Ubuntu that was available at the release time of each OpenPLC version (see Table 5.4). The Long Term Support (LTS) version was chosen, given that the industry tends to work with the most stable version available of any software and security updates are provided for a longer time.

The scenario used for the analysis consists of OpenPLC installed on Ubuntu Linux in a virtual machine, following the OSs shown in Table 5.4.

TABLE 5.4: Versions and release dates of OpenPLC and the available Ubuntu Linux LTS at that time for each date.

VERSION	RELEASE	OS	OS RELEASE
OpenPLC V1	2016/02/05	Ubuntu 14.04 LTS	2014/04/17
OpenPLC V2	2016/05/13	Ubuntu 16.04 LTS	2016/04/21
OpenPLC V3	2018/06/14	Ubuntu 18.04 LTS	2018/04/26

5.2.3 Steps of the Analysis

The EDG model of OpenPLC was built following the steps described in Section 5.1.1.4 (see flowchart in Fig. 5.4). It is worth noting that these steps are followed for each one of the three versions available of OpenPLC, so an EDG is generated for each one. Obtaining the EDG for each version of OpenPLC will give information about the evolution of vulnerabilities over time.

For the sake of clarity and ease of analysis, the three obtained dependency graphs for each OpenPLC version were not merged into a single diagram. Fig. 5.7, Fig. 5.8, and Fig. 5.9, show the obtained EDG for versions V1, V2, and V3, respectively. In reality, these three diagrams would be the result of applying this method over time, updating the graph each time that a new vulnerability is discovered or a new patch/update is issued.

5.2.4 Analysis

In this stage, the analysis of the SUT is performed based on the generated EDG and the value of the computed metrics. This process can be structured into three main steps, as follows:

1. **Analysis of the induced EDG model:** This step involves the analysis of the obtained directed-graph model. The structure, assets and dependencies are the focus of this first step.
2. **Vulnerability analysis:** Vulnerability number, distribution, and severity are analyzed in this step, which is supported by metrics. A proposal for vulnerability prioritization is also proposed.
3. **Root causes analysis (weaknesses):** Finally, the root cause of each vulnerability if found (related to the associated weakness). In this step, new requirements, test cases, and training activities are proposed based on the results of the analysis.

For each of the previously described steps, the analysis is done in both the spatial and temporal dimensions:

1. **Spatial Dimension:** This kind of analysis focuses on the distribution of vulnerabilities among the assets at a time t. In this example, this corresponds to independently analyzing each obtained EDG, because each one represents an instant in time (a different version of OpenPLC).
2. **Temporal Dimension:** This kind of analysis focuses on the evolution and distribution of vulnerabilities over time. In this example, this corresponds to

analyzing the changes in the number of assets and vulnerabilities, and their distribution over all three EDGs (over all three versions of OpenPLC).

5.2.4.1 Analysis of the Induced EDG Model

OpenPLC V1 is analyzed in this subsection, focusing on the internal structure and dependencies among the assets.

The first result of the EDG model is thegraph obtained for OpenPLC V1 (Fig. 5.7). From the **spatial dimension** point of view, assets depend on a main service, `server.js`, based in Java. This web server offers a web GUI for the user to configure, start, and stop the PLC execution. Below this level, the main components of OpenPLC can be seen: OPLC Starter (responsible to start the OpenPLC and constantly monitor if it is running or not), OPLC Compiler (compiler from ladder logic to ANSI C code), and openplc (initialization procedures for the hardware, network and the main loop). The other assets are dynamic libraries of the system, such as `libstdc++`, `libm`, textttlibc (for C programming), and textttlibssl (C library for SSL and TLS).

Moreover, the `libc` library is a wrapper around the system calls of the Linux kernel, which provides and defines system calls and other basic functions. Thus, it is expected that all of the identified assets depend on this library in every version of OpenPLC. Fig. 5.7 shows that this is indeed the case.

From the **temporal dimension** point of view, OpenPLC V2 has to be analyzed in comparison to the previous version of OpenPLC to find changes in the structure and the number of assets. Fig. 5.8 shows the EDG for OpenPLC V2. Comparing both OpenPLC V1 and OpenPLC v2, it can be noted that their structure is very similar. In OpenPLC V2, new assets were introduced to provide new features to the project: Matiec compiler (which is an open-source compiler for programming languages defined in the IEC 61131-3 standard), ST optimizer (which is responsible for the optimization process after the initial compilation from OpenPLC Editor), Glue Generator (which is responsible for gluing the variables from the IEC program to the OpenPLC memory pointers), and OpenDNP3 (which is an implementation of the DNP3 protocol stack written in C++11).

In OpenPLC V2, and in comparison with OpenPLC V1, the OPLC compiler has disappeared and has been substituted by the Matiec Compiler, which supports all programming languages defined in the IEC 61131-3 standard. Moreover, the ST Optimizer and the Glue Generator were added to support the compilation process. Finally, the OpenDNP3 library has been added for high-performance applications, such as many concurrent TCP sessions.

Finally, and keeping the analysis in the temporal dimension, OpenPLC V3 can be compared to its ancestor, OpenPLC V2. Fig. 5.9 shows the EDG for OpenPLC V3. This last version of the project is the simplest. Now, the Java-based web server has been replaced by a Python-based web server, *webserver.py*. The main components of this version are: Matiec compiler, ST optimizer, Glue Generator, OpenDNP3, and LibModbus.

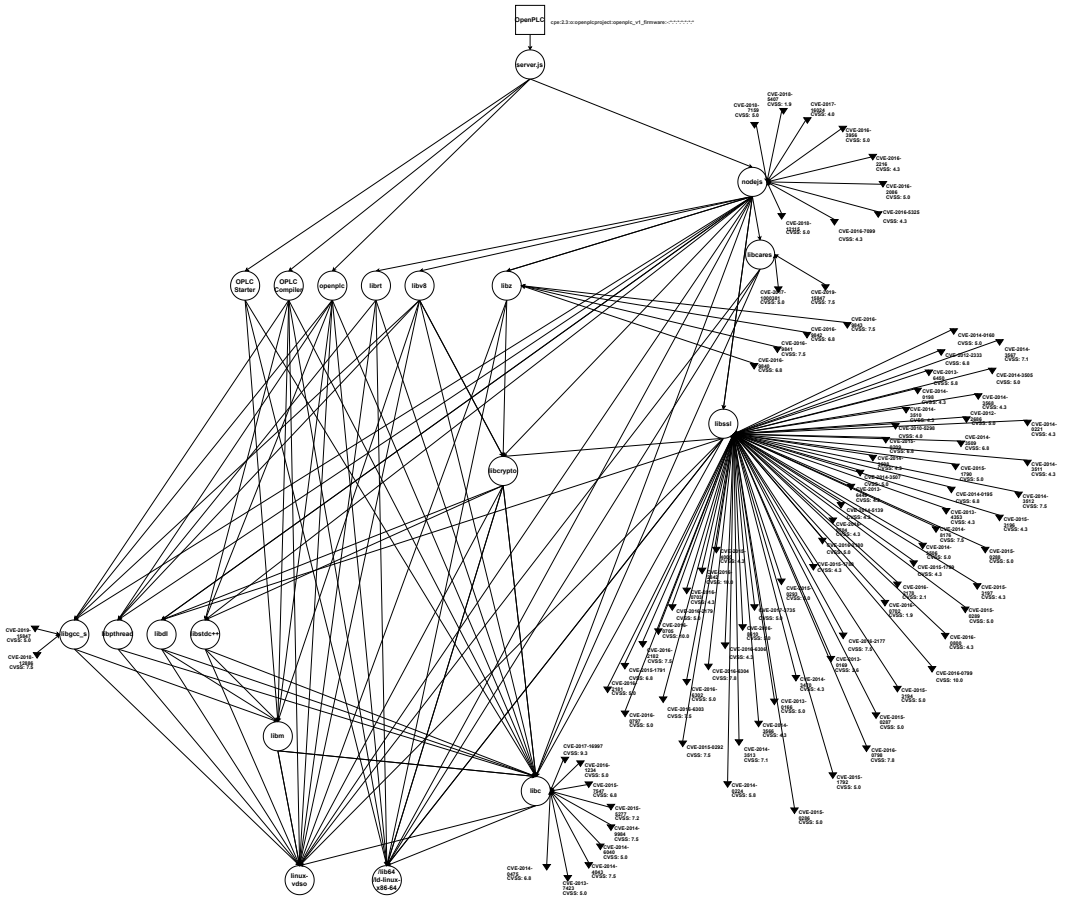

FIGURE 5.7: EDG for OpenPLC V1. Notice that, for simplicity, CWE, and CAPEC values are omitted, and only the CPE identifier of the SUT is shown.

5.2.4.2 Vulnerability Analysis

In this step, all three available versions of the OpenPLC project are analyzed from a vulnerability perspective, both in the spatial and temporal dimensions. The goal here is to use the defined metrics as a support of the analysis, obtaining the number of vulnerabilities, their distribution, and their severity score. Table 5.5 shows the values of the proposed metrics for all three versions of OpenPLC. Finally, a ranked prioritization by severity for the vulnerabilities of each version is proposed.

From the **spatial dimension** perspective, just by observing the generated EDG for OpenPLC V1 (Fig. 5.7), it can be stated that:

- Most of the vulnerabilities are present in third-party open-source components, such as `libssl` ($M_3 = 65$), and `node-js` ($M_3 = 9$), having `libssl` the $M_4 = 71.4\%$ of the vulnerabilities in this version. *A priori*, just by looking the number of vulnerabilities in each asset, it can be said that `libssl` is the most vulnerable asset in this version. Nevertheless, it is never enough to analyze the number of vulnerabilities (M_3), and the CVSS score for each has to be taken into account, as well as the existence of known exploits (*e.g.*, updating the value of the CVSS using the temporal score).

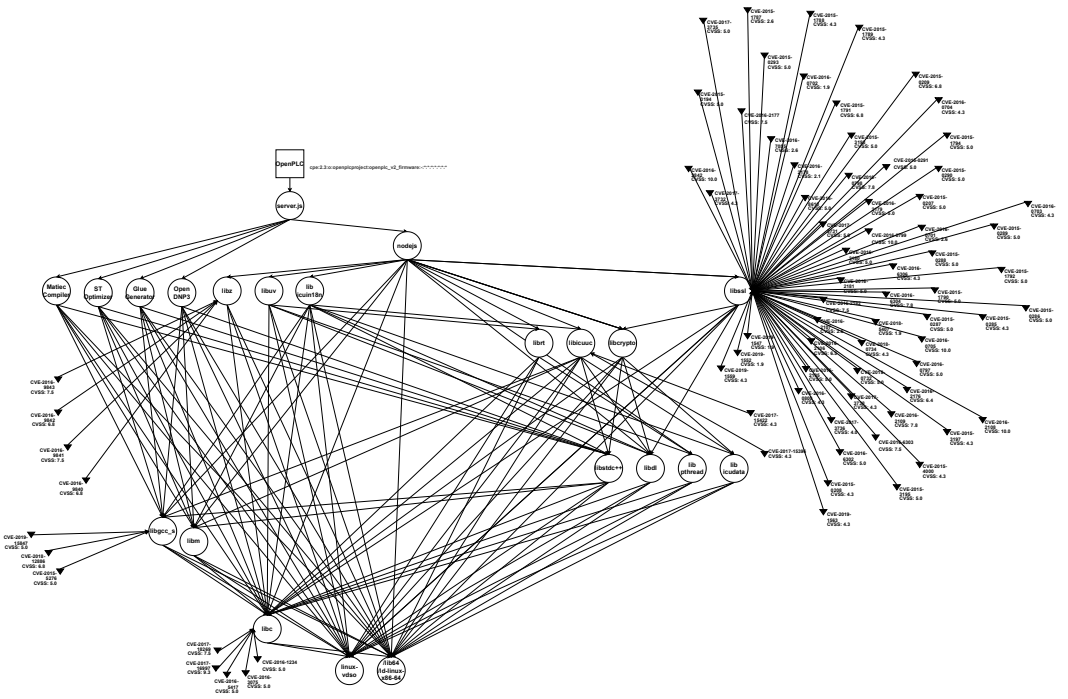

FIGURE 5.8: EDG for OpenPLC V2. Note that for the sake of simplicity, CWE, and CAPEC values are omitted and only the CPE identifier of the SUT is shown.

- The other vulnerabilities in OpenPLC V1 affect the assets of the operating system, such as `libc` ($M_3 = 9$).
- The *ad hoc* assets developed for this project have no known vulnerabilities. This does not mean that they are secure, but rather that there are no known vulnerabilities available for them. Zero-day vulnerabilities could be present in the SUT.

Values of $M_1 = 91$ suggests a high number of vulnerabilities in this project. This is expected given that OpenPLC V1 was the first version of OpenPLC. In the following versions, the value of M_1 is expected to decrease.

The most striking fact when observing the EDG for OpenPLC V1 is the large number of vulnerabilities that are present in *libssl*. From this perspective, if a pen-tester were to evaluate OpenPLC V1, *libssl* would be a promising target. When the values of the CVSS are checked, three vulnerabilities[2] have a score of 10.0 out of 10.0. As can be seen, EDGs are a powerful tool for inspecting the structure of the SUT, and they can be used to analyze how the exploitation of a vulnerability can affect the rest of the SUT.

All of the assets point to *libc*, which is a wrapper around the system calls of the Linux kernel. An evaluator, even without this information, can understand the importance of this dependency with *libc* using the EDG. On closer inspection, the highest score of CVSS for the vulnerabilities of *libc* is 9.3 out of 10.0[3]. If the exploitation of this

[2]CVE-2016-2842, CVE-2016-0799, and CVE-2016-0705.
[3]CVE-2017-16997

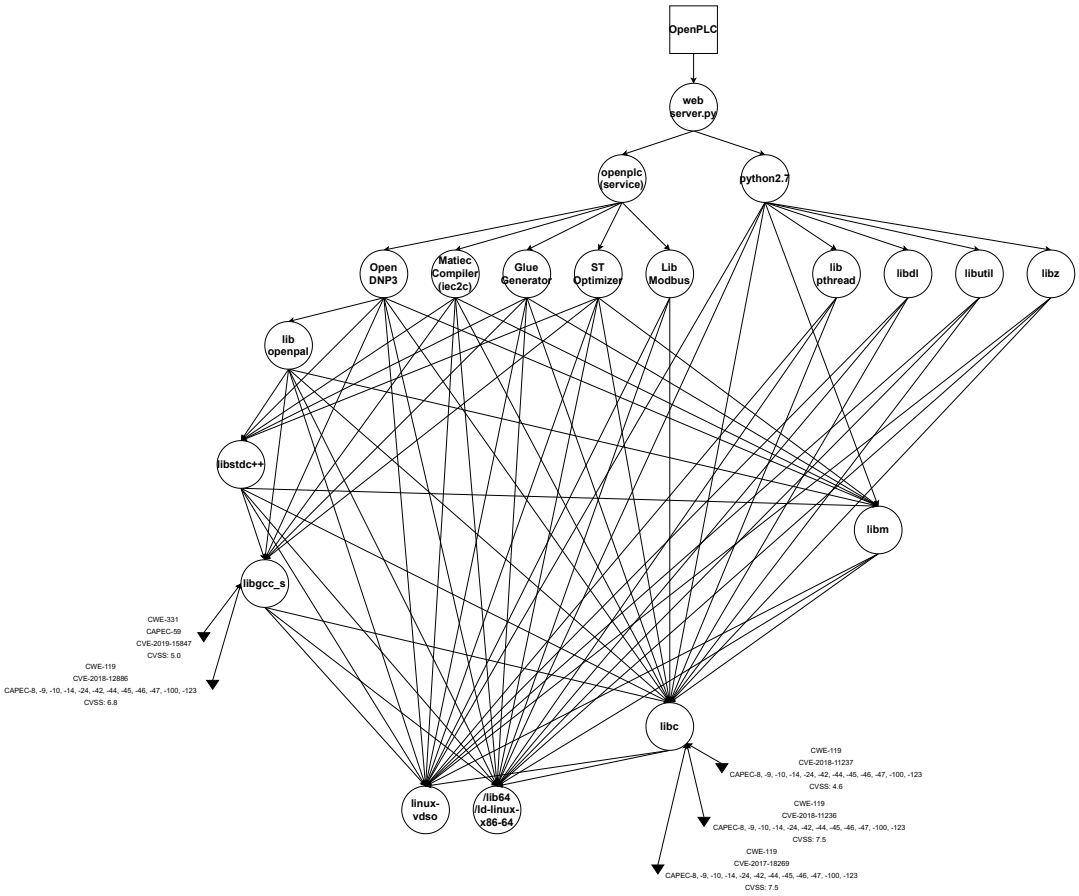

FIGURE 5.9: EDG for OpenPLC V3. Note that only the CPE of SUT is shown for the sake of simplicity.

vulnerability is possible, then it would allow a local user to gain privileges, exposing other assets of the SUT.

Using the value of severity of each vulnerability, it is possible to generate a list of vulnerabilities to be patched. This list can either be ordered by the global CVSS or by asset. Table 5.6 shows all the vulnerabilities whose CVSS value is between 6.0 and 10.0 ordered by asset and by descending CVSS. This can be used to decide in which order the vulnerabilities have to be patched, both at asset level or at SUT level.

From the information in Table 5.6, it is clear that all the three vulnerabilities with a CVSS value of 10.0 are in `libssl`, which makes this asset the most vulnerable by number of vulnerabilities and by CVSS. These vulnerabilities should be a priority during the patching stages. It is worth noting that `libc`, which is an important asset in Ubuntu Linux, has a vulnerability whose score is 9.3 (CVE-2017-16997). This should also be a priority when patching.

Moving now to OpenPLC V2, and comparing it with OpenPLC V1 (temporal dimension), similar conclusions can be drawn:

- Most of the vulnerabilities are in third-party open-source components. Moreover, `libssl` ($M_3 = 63$) remains as the asset with the majority of vulnerabilities (with $M_4 = 81.8\%$ of them), followed by `libz` ($M_3 = 4$) from `node-js`.

TABLE 5.5: Metric values for each asset and version of OpenPLC. Notice that "CWE-NULL" refers to a void value of CWE for a certain CVE value.

	METRIC		OpenPLC V1							OpenPLC V2						OpenPLC V3		
			libgcc_s	libc	libz	libcares	nodejs	libssl	others	libgcc_s	libc	libz	libicuuc	libssl	others	libgcc_s	libc	others
vulnerabilities	$n(t)$		19							22						19		
	$M_0(A) = \frac{\lvert CVE_A(t)\rvert}{n(t)}$		4.79							3.50						0.26		
	$M_1(A,t) = \lvert CVE_A(t)\rvert$		91							77						5		
	$M_2(A) = \sum_{t=1}^{I} \lvert CVE_A(t)\rvert = \sum_{t=1}^{I} M_1(A,t)$		173															
	$M_3(A) = \lvert CVE_{a_i}(t)\rvert$		2	9	4	2	9	65	0	3	5	4	2	63	0	2	3	0
	$M_4(a_k,t) = \frac{\lvert CVE_{a_k}(t)\rvert}{\sum_{i=1}^{n} \lvert CVE_{a_i}(t)\rvert}$		0.02	0.10	0.04	0.02	0.10	0.71	0.00	0.04	0.06	0.05	0.03	0.82	0.00	0.40	0.60	0.00
weaknesses	$M_5(a_i,cwe_j,t) = \lvert CVE_{a_i\lvert cwe_j}(t)\rvert$	CWE-17	-	1	-	-	-	2	-	-	-	-	-	3	-	-	-	-
		CWE-19	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
		CWE-20	-	-	-	-	3	5	-	-	-	-	-	3	-	-	-	-
		CWE-22	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		CWE-94	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		CWE-113	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
		CWE-119	1	5	-	-	-	9	-	1	3	-	1	6	-	1	3	-
		CWE-125	-	-	-	-	-	2	-	-	-	-	-	3	-	-	-	-
		CWE-189	-	-	4	-	-	2	-	-	-	4	-	4	-	-	-	-
		CWE-190	-	-	-	-	-	1	-	-	-	-	1	1	-	-	-	-
		CWE-200	-	-	-	1	3	5	-	1	-	-	-	12	-	-	-	-
		CWE-295	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-
		CWE-310	-	-	-	-	-	12	-	-	-	-	-	5	-	-	-	-
		CWE-311	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-
		CWE-320	-	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-
		CWE-331	1	-	-	-	-	-	-	1	-	-	-	-	-	1	-	-
		CWE-362	-	-	-	-	-	4	-	-	-	-	-	1	-	-	-	-
		CWE-399	-	-	-	-	-	8	-	-	1	-	-	6	-	-	-	-
		CWE-400	-	-	-	-	-	1	-	-	-	-	-	1	-	-	-	-
		CWE-426	-	1	-	-	-	-	-	-	1	-	-	-	-	-	-	-
		CWE-787	-	-	-	1	1	2	-	-	-	-	-	2	-	-	-	-
		CWE-NULL	-	-	-	-	-	12	-	-	-	-	-	10	-	-	-	-
	$M_6(A,cwe_j,t) = \lvert CVE_{A\lvert cwe_j}(t)\rvert$	CWE-17	3							3						-		
		CWE-19	1							-						-		
		CWE-20	8							3						-		
		CWE-22	1							-						-		
		CWE-94	1							-						-		
		CWE-113	1							-						-		
		CWE-119	15							11						4		
		CWE-125	2							3						-		
		CWE-189	6							8						-		
		CWE-190	1							2						-		
		CWE-200	9							13						-		
		CWE-295	-							1						-		
		CWE-310	12							5						-		
		CWE-311	-							2						-		
		CWE-320	-							3						-		
		CWE-331	1							1						1		
		CWE-362	4							1						-		
		CWE-399	8							7						-		
		CWE-400	1							1						-		
		CWE-426	1							1						-		
		CWE-787	4							2						-		
		CWE-NULL	12							10						-		
	$M_7(A,t) = \lvert CWE_A(t)\rvert$		19							18						2		
	$M_8(A) = \bigcup_{t=1}^{I} \lvert CWE_A(t)\rvert = \bigcup_{t=1}^{I} M_7(A,t)$		22															

- The remaining vulnerabilities are related to the operating system, just as in OpenPLC V1, `libc` ($M_3 = 5$).

For this version, $M_1 = 77$, less than $M_1 = 98$ for the previous version. As was expected, the total number of vulnerabilities has decreased. From the **spatial dimension**, it is worth noting that `libssl` is still the asset with the highest number of vulnerabilities and is still a promising target for a pen-tester or an attacker. Nevertheless, the number of vulnerabilities for `libssl` (M_3) has decreased from OpenPLC V1 to OpenPLC V2, as it was expected.

As was done earlier, a list of vulnerabilities can be generated that is ordered by asset and by descending CVSS. Table 5.7 shows all of the vulnerabilities for OpenPLC V2 whose CVSS value is between 6.0 and 10.0.

Table 5.7 shows that `libssl` now has four instead of three vulnerabilities whose CVSS value is 10.0 (CVE-2016-0799, CVE-2016-2108, CVE-2016-0705, CVE-2016-2842). In OpenPLC V2, `libc` is still affected by the same vulnerability with a score of 9.3 (CVE-2017-16997). All of these vulnerabilities should be a priority during patching activities.

TABLE 5.6: Vulnerability prioritization by asset and by CVSS for OpenPLC V1. Only the vulnerabilities whose value is between 6.0 and 10.0 are shown.

CVE	CVSS	ASSET
CVE-2019-15847	7.5	libcares
CVE-2018-12886	7.5	libgcc
CVE-2016-9843	7.5	libz
CVE-2016-9841	7.5	libz
CVE-2016-9840	6.8	libz
CVE-2016-9842	6.8	libz
CVE-2017-16997	9.3	libc
CVE-2014-9984	7.5	libc
CVE-2014-4043	7.5	libc
CVE-2015-5277	7.2	libc
CVE-2015-7547	6.8	libc
CVE-2014-0475	6.8	libc
CVE-2016-2842	10.0	libssl
CVE-2016-0705	10.0	libssl
CVE-2016-0799	10.0	libssl
CVE-2016-6304	7.8	libssl
CVE-2016-0798	7.8	libssl
CVE-2014-8176	7.5	libssl
CVE-2016-2182	7.5	libssl
CVE-2014-3512	7.5	libssl
CVE-2016-6303	7.5	libssl
CVE-2015-0292	7.5	libssl
CVE-2016-2177	7.5	libssl
CVE-2014-3567	7.1	libssl
CVE-2014-3513	7.1	libssl
CVE-2015-1791	6.8	libssl
CVE-2012-2333	6.8	libssl
CVE-2015-0209	6.8	libssl
CVE-2014-3509	6.8	libssl
CVE-2014-0195	6.8	libssl
CVE-2014-3505	5.0	libssl

Finally, OpenPLC V3, which is the last available version, is analyzed. The most striking fact when comparing OpenPLC V3 with the other versions (**temporal dimension**) is the significant reduction in the global number of vulnerabilities. This effect is caused by the absence of `libssl` in this version.

Focusing on the **spatial dimension** for this version, vulnerabilities in OpenPLC V3 are only related to assets in the operating system, *libgcc_s* ($M_3 = 2$), and *libc* ($M_3 = 3$). In OpenPLC V3, neither third-party open-source assets nor *ad hoc* assets have any known vulnerability. In fact, OpenPLC V3 has the lowest values of $M_1 = 5$ for the whole life cycle of the project, and therefore has the lowest number of vulnerabilities.

Table 5.8 shows an ordered list of all the vulnerabilities in OpenPLC V3 whose CVSS value is between 6.0 and 10.0.

For OpenPLC V3, the most vulnerable asset is `libc`, according to the CVSS score shown in Table 5.8.

TABLE 5.7: Vulnerability prioritization by asset and by CVSS for OpenPLC V2. Only the vulnerabilities whose value is between 6.0 and 10.0 are shown.

CVE	CVSS	ASSET
CVE-2018-12886	6.8	libgcc_s
CVE-2017-16997	9.3	libc
CVE-2017-18269	7.5	libc
CVE-2016-9843	7.5	libz
CVE-2016-9841	7.5	libz
CVE-2016-9842	6.8	libz
CVE-2016-9840	6.8	libz
CVE-2016-0799	10.0	libssl
CVE-2016-2108	10.0	libssl
CVE-2016-0705	10.0	libssl
CVE-2016-2842	10.0	libssl
CVE-2016-0798	7.8	libssl
CVE-2016-6304	7.8	libssl
CVE-2016-2109	7.8	libssl
CVE-2016-2177	7.5	libssl
CVE-2016-2182	7.5	libssl
CVE-2016-6303	7.5	libssl
CVE-2015-0209	6.8	libssl
CVE-2015-1791	6.8	libssl
CVE-2016-2106	6.5	libssl
CVE-2016-2176	6.4	libssl

TABLE 5.8: Vulnerability prioritization by asset and by CVSS for OpenPLC V3. Only the vulnerabilities whose value is between 6.0 and 10.0 are shown.

CVE	CVSS	ASSET
CVE-2018-12886	6.8	libgcc_s
CVE-2018-11236	7.5	libc
CVE-2017-18269	7.5	libc

5.2.4.3 Root Causes Analysis (Weaknesses)

In this last step of the analysis, the information about the root causes of the vulnerabilities is extracted. This can be achieved through weaknesses. Each CWE contains data about the main issue to be solved for any given vulnerability. This information is useful to propose new requirements, test cases, and training. The analysis of the root causes will also be carried out in both the spatial and temporal dimensions.

The analysis of the root causes starts by extracting the weaknesses for each vulnerability for all three versions of OpenPLC. All of the extracted weaknesses for all three available versions of OpenPLC are shown in Fig. 5.10 and in Table 5.9.

Focusing on the spatial dimension, OpenPLC V1 has the widest range of weaknesses ($M_7 = 19$), and therefore has the widest range of root causes. From the total number of vulnerabilities in OpenPLC V1 ($M_3 = 91$), $M_6 = 15$ of them are related to weaknesses CWE-119[4]: "The software performs operations on a memory buffer, but it can read

[4]Description for CWE-119: `https://cwe.mitre.org/data/definitions/119.html`

TABLE 5.9: Weakness analysis for all versions of OpenPLC.

	METRIC	OpenPLC V1	OpenPLC V2	OpenPLC V3	$\sum_{i=1}^{T} M_8(A, cwe_j, t)$
$M_8(A, cwe_j, t) = \lvert CVE_{A\lvert cwe_j}(t)\rvert$	CWE-17	3	3	-	6
	CWE-19	1	-	-	1
	CWE-20	8	3	-	11
	CWE-22	1	-	-	1
	CWE-94	1	-	-	1
	CWE-113	1	-	-	1
	CWE-119	15	11	4	30
	CWE-125	2	3	-	5
	CWE-189	6	8	-	14
	CWE-190	1	2	-	3
	CWE-200	9	13	-	22
	CWE-295	-	1	-	1
	CWE-310	12	5	-	17
	CWE-311	-	2	-	2
	CWE-320	-	3	-	3
	CWE-331	1	1	1	3
	CWE-362	4	1	-	5
	CWE-399	8	7	-	15
	CWE-400	1	1	-	2
	CWE-426	1	1	-	2
	CWE-787	4	2	-	6
	CWE-NULL	12	10	-	22
$M_7(A, t) = \lvert CWE_A(t)\rvert$		19	18	2	22

from or write to a memory location that is outside the intended boundary of the buffer." $M_6 = 12$ of them are related to CWE-310 [5]: "Weaknesses in this category are related to the design and implementation of data confidentiality and integrity. Frequently, these deal with the use of encoding techniques, encryption libraries, and hashing algorithms. The weaknesses in this category could lead to a degradation of the quality of data if they are not addressed."

Repeating this same analysis with OpenPLC V2, we can see that the number of different weaknesses ($M_7 = 18$) is almost the same as before. But in this version, the most repeated weakness is CWE-200[6]: "The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information." Followed by CWE-119.

Comparing the previous versions with OpenPLC V3, we find now that the total number of weaknesses ($M_7 = 2$) has been reduced drastically. Nevertheless, weakness CWE-119 is still present in this version, and is the most repeated.

This analysis can also be done by combining the information of all three versions, drawing conclusions for the whole life cycle of OpenPLC. Requirements and test cases can be extracted from the most recurrent weaknesses, and training can be proposed to avoid weaknesses that repeat over time. In our example, CWE-119 has the greatest

[5]Description for CWE-310: `https://cwe.mitre.org/data/definitions/310.html`.
[6]Description for CWE-200: `https://cwe.mitre.org/data/definitions/200.html`

frequency over time ($\sum_{i=1}^{T} M_6 = 30$), followed by CWE-200 ($\sum_{i=1}^{T} M_6 = 22$), and CWE-310 ($\sum_{i=1}^{T} M_6 = 17$). Table 5.10 shows the generated requirements, Table 5.11 shows the proposed training, and Table 5.12 shows the proposed test cases for OpenPLC.

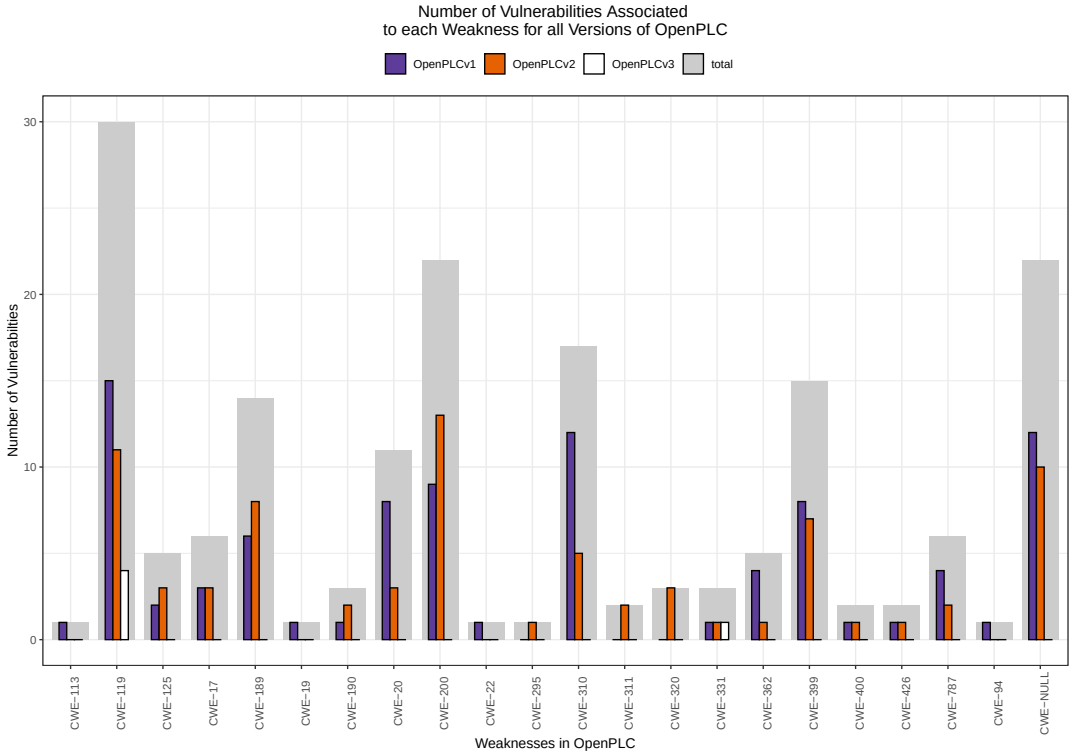

FIGURE 5.10: Evolution of the number of vulnerabilities over consecutive versions of OpenPLC.

TABLE 5.10: An example of generated requirements for OpenPLC.

CWE ID	REQUIREMENTS
CWE-19, CWE-20, CWE-119, CWE-125, CWE-189, CWE-190, CWE-362, CWE-399, CWE-400, CWE-787	Use languages that perform their own memory management.
CWE-19, CWE-20, CWE-119, CWE-125, CWE-189, CWE-190, CWE-787	Use libraries or frameworks that make it easier to handle numbers without unexpected consequences. Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++).
CWE-19, CWE-20, CWE-119, CWE-125, CWE-189, CWE-190, CWE-200, CWE-362, CWE-787	Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent.
CWE-19, CWE-20, CWE-125, CWE-189, CWE-190, CWE-200, CWE-362	Ensure that all protocols are strictly defined, such that all out-of-bounds behaviors can be identified simply, and require strict conformance to the protocol.
CWE-19, CWE-20, CWE-399, CWE-400	Run or compile the software using features or extensions that randomly arrange the positions of a program's executable and libraries in memory (*e.g.*, Address Space Layout Randomization (ASLR), or Position-Independent Executables (PIE)).
CWE-19, CWE-295, CWE-310, CWE-311, CWE-320, CWE-331	Clearly specify which data or resources are valuable enough that they should be protected by encryption. Require that any transmission or storage of this data/resource should use well-vetted encryption algorithms. Up-to-date algorithms must be used, and the entropy of the keys must be sufficient for the application.
CWE-22, CWE-94, CWE-113, CWE-399, CWE-400, CWE-426	Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party.
CWE-20, CWE-22, CWE-94, CWE-113, CWE-399, CWE-400, CWE-426	Use an input validation framework such as Struts or the OWASP ESAPI Validation API.
CWE-19, CWE-20, CWE-22, CWE-94, CWE-113, CWE-295, CWE-426	Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
CWE-19, CWE-20, CWE-119, CWE-125, CWE-399, CWE-400, CWE-787	Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows.
CWE-19, CWE-22, CWE-119, CWE-125, CWE-399, CWE-400, CWE-787	Replace unbounded copy functions with analogous functions that support length arguments, such as strcpy with strncpy. Create these if they are not available.

TABLE 5.11: Example of proposed training for OpenPLC.

CWE ID	TRAINING
CWE-19, CWE-20, CWE-22, CWE-94, CWE-113, CWE-119, CWE-125, CWE-426	Identification of all potentially relevant properties of an input (length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields).
CWE-19, CWE-20, CWE-22, CWE-94, CWE-113, CWE-119, CWE-125, CWE-426	Input validation strategies.
CWE-19, CWE-20, CWE-22, CWE-94, CWE-113, CWE-119, CWE-125, CWE-200, CWE-426	Allowlists and Denylists.
CWE-19, CWE-20, CWE-22, CWE-94, CWE-113, CWE-119, CWE-125, CWE-189, CWE-190, CWE-426	Character encoding compatibility.
CWE-19, CWE-20, CWE-94, CWE-113, CWE-119, CWE-125, CWE-426	Buffer overflow detection during compilation (*e.g.*, Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice).
CWE-19, CWE-20, CWE-94, CWE-113, CWE-119, CWE-125, CWE-200, CWE-426	Secure functions, such as *strcpy* with *strncpy*. Create these if they are not available.
CWE-19, CWE-20, CWE-94, CWE-113, CWE-119, CWE-125, CWE-189, CWE-190, CWE-426,CWE-787	Secure programming: memory management.
CWE-19, CWE-20, CWE-22, CWE-94, CWE-113, CWE-119, CWE-125, CWE-189, CWE-190, CWE-787	Understand the programming language's underlying representation and how it interacts with numeric calculation.
CWE-19, CWE-20, CWE-22, CWE-94, CWE-113, CWE-119, CWE-125	System compartmentalization.
CWE-200, CWE-295, CWE-310, CWE-311, CWE-320, CWE-331	Certificate management.
CWE-200, CWE-295, CWE-310, CWE-311, CWE-320, CWE-331	Certificate pinning.
CWE-310, CWE-311, CWE-320, CWE-331	Encryption integration (Do not develop custom or private cryptographic algorithms).
CWE-310, CWE-311, CWE-320, CWE-331	Secure up-to-date cryptographic algorithms.
CWE-189, CWE-200, CWE-362, CWE-399, CWE-400, CWE-426	Shared resources management.
CWE-200, CWE-362, CWE-399, CWE-400, CWE-426	Thread-safe functions.

TABLE 5.12: Example of generated test cases for OpenPLC.

CAPEC ID	TEST CASES
CAPEC-10	Check for buffer overflows through manipulation of environment variables. This test leverages implicit trust often placed in environment variables.
CAPEC-14	Static analysis of the code: secure functions and buffer overflow.
CAPEC-24	Feed overly long input strings to the program in an attempt to overwhelm the filter (by causing a buffer overflow) and hoping that the filter does not fail securely (i.e. the user input is let into the system unfiltered)
CAPEC-45	This test uses symbolic links to cause buffer overflows. The evaluator can try to create or manipulate a symbolic link file such that its contents result in out of bounds data. When the target software processes the symbolic link file, it could potentially overflow internal buffers with insufficient bounds checking.
CAPEC-46	Static analysis of the code: secure functions and buffer overflow.
CAPEC-47	In this test, the target software is given input that the evaluator knows will be modified and expanded in size during processing. This test relies on the target software failing to anticipate that the expanded data may exceed some internal limit, thereby creating a buffer overflow.
CAPEC-59	This test targets predictable session ID in order to gain privileges. The evaluator can try to predict the session ID used during a transaction to perform spoofing and session hijacking.
CAPEC-97	Automated measurement of the entropy of the keys generated.
CAPEC-475	Fuzz testing to externally provided data, such as directories and filenames.

5.2.5 Second Scenario

For this use case, the setup consisted of OpenPLC installed on 14.04 LTS Ubuntu Linux in a virtual machine. All configuration options were by default.

5.2.6 Building the EDG

Using the generated EDG for OpenPLC V1 shown in Figure 5.7, we extracted the information about security updates (discarding updates that introduced more functionalities), for both `libssl` and `nodejs`. Table 5.13 shows the security updates and their date of availability for both `libssl` [193] and `nodejs` [194] for Ubuntu 14.04 LTS. There were two security updates available for the amd64 architecture for each asset.

TABLE 5.13: Update information of both `libssl` and `nodejs`.

ASSET	$1_{st} UPDATE$	SOLVED VULNERABILITIES (CVSS)	$2_{nd} UPDATE$	SOLVED VULNERABILITIES (CVSS)
libssl	2014/04/07	CVE-2014-0076 (1.9) CVE-2014-0160 (5.0)	2018/12/06	CVE-2018-5407 (1.9) CVE-2018-0734 (4.3)
nodejs	2014/03/27	—	2018/08/10	CVE-2016-5325 (4.3)

From this data, we can extract that:

- Updates for `nodejs` were released before the updates for `libssl`.
- `libssl` shows more vulnerabilities than `nodejs`.
- The highest CVSS score in the period of this analysis is 5.

Then, the EDG for these two assets and their updates were built. Figure 5.11 shows the updates over time of the EDG, whereas Figure 5.12 shows the final EDG with all the information included.

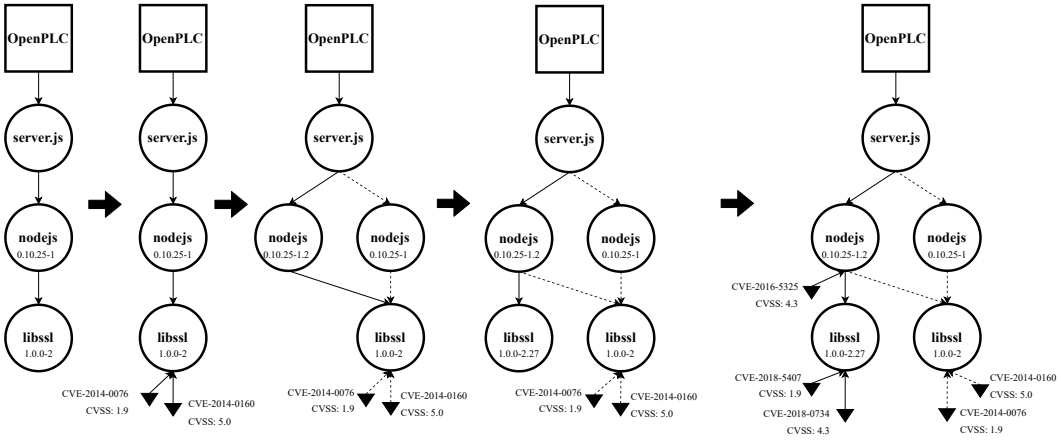

FIGURE 5.11: Temporal evolution of the EDG for OpenPLC V1 for both libss and nodejs.

5.2.7 Analysis of the EDG

Using Figure 5.12, and Table 5.14, we can analyze the obtained EDG:

1. **Analysis of the induced EDG model:** The structure, assets, and dependencies are the focus of this first step.

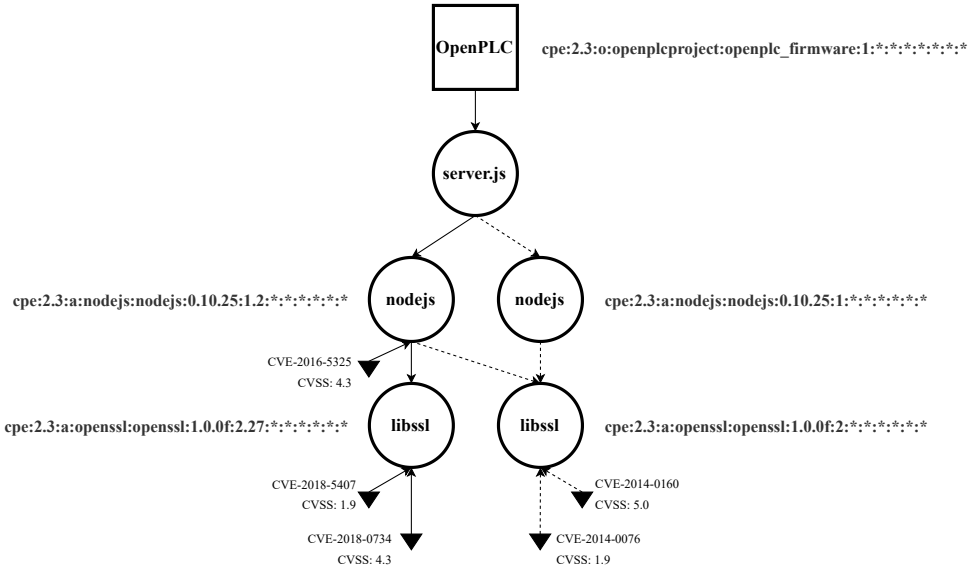

FIGURE 5.12: Final EDG for libssl and nodejs integrating all the updates for Ubuntu Linux 14.04 for amd64 architecture.

We can observe that `libssl` is used by `nodejs`, and they are not at the same level of the hierarchy. So vulnerabilities could propagate upwards and downwards through the EDG.

2. **Vulnerability analysis:** Vulnerability number, distribution, and severity are analyzed in this step. A proposal for vulnerability prioritization is also generated.

   We can highlight that `nodejs` had one vulnerability discovered after its first update, whereas `libssl` had vulnerabilities in both periods of time. We could argue that, as `nodejs` is the most accessible asset from the exterior, its vulnerabilities should be first addressed, even though the associated CVSS is not the highest one.

3. **Weaknesses analysis:** Finally, the root cause of each vulnerability is found. In this step, new requirements, test cases, and training activities are proposed based on the results of the analysis.

   Table 5.14 shows the root cause for each vulnerability. Using this data, new requirements (Table 5.15), test cases (Table 5.16) and training activities (Table 5.17) were proposed.

It is worth noticing that this use case is focused on reflecting the temporal evolution of the EDG. For this reason, metrics cannot be computed here, because of the low number of vulnerabilities available.

TABLE 5.14: Relationship between vulnerabilities and weaknesses for both `libssl` and `nodejs`.

CVE	CVSS	CWE	DESCRIPTION
CVE-2014-0076	1.9	CWE-310	Cryptographic Issues
CVE-2014-0160	7.5	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer
CVE-2016-5325	6.1	CWE-113	Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
CVE-2018-0734	5.9	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CVE-2018-5407	4.7	CWE-203 CWE-200	Observable Discrepancy Exposure of Sensitive Information to an Unauthorized Actor

TABLE 5.15: An example of generated requirements for OpenPLC V1.

CWE ID	REQUIREMENTS
CWE-119	Use languages that perform their own memory management.
CWE-119	Use libraries or frameworks that make it easier to handle numbers without unexpected consequences. Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++).
CWE-119, CWE-200	Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent.
CWE-190, CWE-200	Ensure that all protocols are strictly defined, such that all out-of-bounds behaviors can be identified simply, and require strict conformance to the protocol.
CWE-310	Clearly specify which data or resources are valuable enough that they should be protected by encryption. Require that any transmission or storage of this data/resource should use well-vetted encryption algorithms. Up-to-date algorithms must be used, and the entropy of the keys must be sufficient for the application.
CWE-113	Use an input validation framework such as Struts or the OWASP ESAPI Validation API.
CWE-113	Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
CWE-113	Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party.
CWE-119	Run or compile the software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows.
CWE-119	Replace unbounded copy functions with analogous functions that support length arguments, such as strcpy with strncpy. Create these if they are not available.

TABLE 5.16: Example of generated test cases for OpenPLC V1.

CAPEC ID	TEST CASES
CAPEC-119	Check for buffer overflows through manipulation of environment variables. This test leverages implicit trust often placed in environment variables.
CAPEC-119	Static analysis of the code: secure functions and buffer overflow.
CAPEC-119	Feed overly long input strings to the program in an attempt to overwhelm the filter (by causing a buffer overflow) and hoping that the filter does not fail securely (i.e. the user input is let into the system unfiltered)
CAPEC-119	This test uses symbolic links to cause buffer overflows. The evaluator can try to create or manipulate a symbolic link file such that its contents result in out of bounds data. When the target software processes the symbolic link file, it could potentially overflow internal buffers with insufficient bounds checking.
CAPEC-119	Static analysis of the code: secure functions and buffer overflow.

TABLE 5.17: Example of proposed training for OpenPLC V1.

CWE ID	TRAINING
CWE-113, CWE-119	Identification of all potentially relevant properties of an input (length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields).
CWE-113, CWE-119	Input validation strategies.
CWE-113, CWE-119, CWE-200	Allowlists and Denylists.
CWE-113, CWE-119	Character encoding compatibility.
CWE-113, CWE-119	Buffer overflow detection during compilation (*e.g.*, Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice).
CWE-113, CWE-119CWE-200	Secure functions, such as *strcpy* with *strncpy*. Create these if they are not available.
CWE-113, CWE-119CWE-190	Secure programming: memory management.
CWE-113, CWE-119	Understand the programming language's underlying representation and how it interacts with numeric calculation.
CWE-113, CWE-119	System compartmentalization.
CWE-200, CWE-310	Certificate management.
CWE-200, CWE-310	Certificate pinning.
CWE-310	Encryption integration (Do not develop custom or private cryptographic algorithms).
CWE-310	Secure up-to-date cryptographic algorithms.
CWE-200	Shared resource management.
CWE-200	Thread-safe functions.

5.3 Conclusions and Future Work

Vulnerability analysis is a critical task which ensures the security of industrial components. The EDG model that we propose performs continuous vulnerability assessment throughout the entire life cycle of industrial components. The model is built on a directed graph-based structure and a set of metrics based on globally accepted security standards. Metrics can be used by the model to improve the development process of the SUT, enhance its security, and track its status. The key feature of the proposed model is the addition of the temporal dimension in the analysis of vulnerabilities. The location of vulnerabilities can be analyzed in both space (in which asset) and time (their recurrence), which allows the state of the device to be tracked throughout the whole life cycle.

The model was successfully applied to the OpenPLC use case, which demonstrated its advantages, applicability, and potential. The use case showed that the model can assist in updating management activities, applying patching policies, launching training activities, and generating new test cases, and requirements. This has significant implications for cybersecurity evaluators, as it can serve as a starting point for identifying vulnerabilities, weaknesses, and attack patterns.

Further research will enhance the EDG by adding a mathematical model to aggregate the values of the CVSS metric for each asset, and a value for the whole SUT. This will enable the comparison of different SUTs over time. More improvements will be made in the prioritization of patching, taking into account the context and the functionalities of the SUT. Finally, historical information about the developers can be integrated into the EDG model to predict future vulnerabilities.

Chapter 6

Gotta Catch 'em All: Aggregating CVSS Scores

In this chapter, we propose a novel aggregation algorithm for a set of CVSS values which takes into account context-related properties of the SUT.[1]. This approach is based on the EDGs proposed in Chapter 5. Because EDGs are capable of modeling dependencies, this algorithm can also be applied to computer networks. Our proposal is capable of selecting the most relevant CVSS to be aggregated, taking into account four different context-related properties of the SUT:

1. Functionality disruption.
2. Exploitation difficulty.
3. Existence of exploits, and their development state.
4. Context of deployment.

This approach increases the granularity of the CVSS base, environment and temporal metrics, where not every possible value in the scale $[0, 10]$ is achievable, or the result of changing the value of a submetric has almost no effect on the final CVSS [164, 195]. Moreover, our proposal is capable of detecting which branch in the EDG, (in the case of an ES the most critical set of assets), is contributing the most (more critical) to the final score.

[1]The Python code implementing the aggregation algorithm is available at GitHub `https://github.com/aaalongueira/CVSS_Aggregation`.

6.1 Proposed Approach for Metric Aggregation

In this section, we propose a CVSS aggregation algorithm inspired by the risk propagation formula [196] described in MAGERIT [197,198]. First, we describe the corrections factors involved in our proposal. Then, the aggregation formula is introduced. Finally, the algorithm and the interpretation of the results is explained in detail.

6.1.1 Correction Factors

The proposed aggregation algorithm integrates correction factors to adapt the formula described in MAGERIT. These correction factors apply individually for each CVSS, except for the average and summarized factors. Correction factors are summarized in Table 6.1.

TABLE 6.1: Correction factors proposed for adapting the Bayesian sum proposed in MAGERIT.

CORRECTION FACTOR	DESCRIPTION
Functionality factor (ρ)	Binary value indicating whether a vulnerability affects or not the functionality of the SUT.
Deepness factor (β)	Value between $[0, 1]$ proportional to the position of the affected asset in the EDG of the SUT.
Context factor (γ)	Binary value indicating whether the vulnerability can be exploited in the real and particular conditions of the SUT.
Exploit factor (μ)	This factor accounts for the existence of a public exploit for a given vulnerability, being proportional to its state of development: Not defined ($\mu = 0.5$), Theoretical ($\mu = 1.25$), Proof-Of-Concept ($\mu = 1.5$), Functional ($\mu = 1.75$), and Automated ($\mu = 2$).
Summarized factor (λ)	This factor summarizes the effect of all the above ones, $\lambda = \rho\beta\gamma\mu\sigma$.
Average factor (σ)	Function that adjust the value of the sum to avoid its rapid evolution to 10.

1. *Functionality factor (ρ):* This correction factor represents whether any functionality of the systems is affected by its vulnerabilities.

   It is represented by a binary value, being 0 when no functionality is affected, and 1 when any of them is affected.

   For example, a cryptographic library with a vulnerability in SHA1. If the SUT does not make use of SHA1 in any way, the vulnerability would not be exploitable, and could be removed from the analysis ($\rho = 0$).

2. *Deepness Factor (β):* This factor represents the difficulty of chained exploitation of each vulnerability. It is represented by a value between $[0, 1]$ inversely proportional to the amount of assets to compromise in order to exploit vulnerability. Vulnerabilities close to the entry point will account more for the final aggregation, whereas those that are far away will account less. In this approach, linear

interpolation is proposed to calculate the weight of each layer, because of its simplicity. Nevertheless, different interpolations could be used according to the criticality of the system. Figure 6.1 shows the corresponding β for a four-layer system.

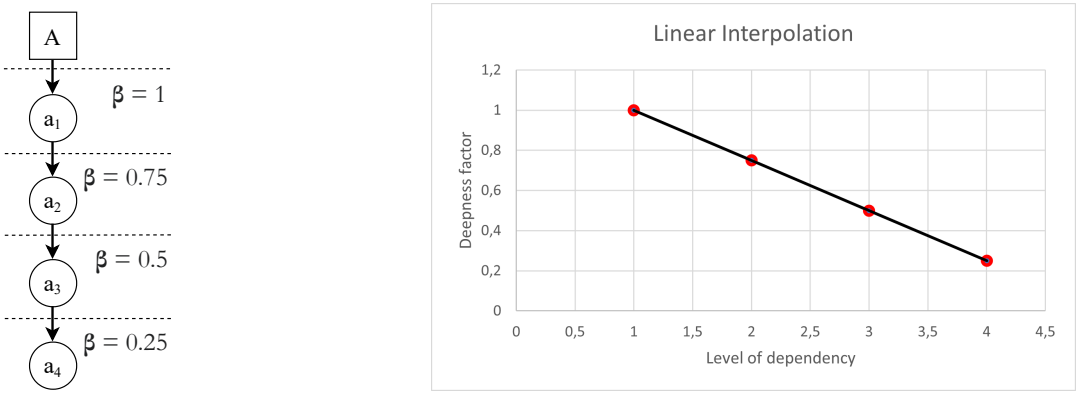

FIGURE 6.1: Calculation of the deepness factor for a four-layer of dependency example.

3. *Context factor (γ):* This factor considers whether the exploitation of a vulnerability is actually possible in the real scenario where the system is deployed. It is represented by a binary value, where 0 indicated that it is not possible, and 1, that it is possible. It is calculated comparing the attack vector of the CVSS with the real conditions where the device is deployed. For example, this can happen when a vulnerability with a high CVSS score needs physical access to be exploited, but in reality the device is physically isolated. To reflect this, the CVSS should be updated, lowering the resulting value [165]. This factor aims to complement the existing submetrics in the temporal and the environment metrics of the CVSS. Both the temporal and the environmental scores lack of an "isolated" value for the attack vector.

4. *Exploit factor (μ):* This factor accounts for the existence of a public exploit for a given vulnerability, being proportional to its state of development. The temporal score of the CVSS already implements this feature, but the CVSS values are not updated in practice [195]. Moreover, taking into account the temporal score has almost no effect as opposed to using the raw initial base score. This means that a CVSS just considering the base score is higher than a CVSS considering an exploit code maturity of "functional exploit exists". To solve this issue, we introduce the following values for the exploit factor: Not defined ($\mu = 0.5$), Theoretical ($\mu = 1.25$), Proof-Of-Concept ($\mu = 1.5$), Functional ($\mu = 1.75$), and Automated ($\mu = 2$). These values are equivalent to the scale defined in the CVSS Specification Document [67].

5. *Summarized factor (λ):* The λ factor accounts for the effect of all the factors above:

$$\lambda = \rho\beta\gamma\mu \tag{6.1}$$

6. *Average factor (σ):* This factor defines the behavior of the aggregation function. It can be chosen as needed (*e.g.*, the arithmetic or harmonic mean), but taking into account all the values to be added.

6.1.2 Aggregation Formula

The aggregation function is defined as:

$$\Gamma(\overrightarrow{V}) = 10 - \frac{1}{\sigma} f(\overrightarrow{V}) \tag{6.2}$$

Where $\overrightarrow{V}$ is a vector $(cvss_0, cvss_1, \ldots, cvss_n)$ with all the corrected CVSS values to be added, $cvss$, being n the last value to be added. $f(\overrightarrow{V}) = a_n$ is defined as the following recursive function:

$$a_n = 10 \left[1 - \left(1 - \frac{\lambda_{a_{n-1}}}{10} a_{n-1} \right) \cdot \left(1 - \frac{\lambda_{cvss_n}}{10} cvss_n \right) \right] \tag{6.3}$$

Where the base case is defined as:

$$a_0 = \lambda_{cvss_0} cvss_0 \tag{6.4}$$

6.1.3 Algorithm

The proposed aggregation algorithm is divided into the following steps (see Figure 6.2):

1. Calculation of the correction factors for each CVSS,
2. Calculation of the summarized factor for each CVSS,
3. Calculation of the corrected CVSS values,
4. Calculation of the average correction function, and
5. Aggregation.

Notice that the dependency graph of the SUT, the vulnerabilities associated to each element of the dependency graph, and their CVSS value are needed.

Correction factors for each CVSS

The first step obtains the values of each correction factor for each CVSS:

1. **Functionality factor (ρ):** This factor is obtained using the description provided in the corresponding CVE of each CVSS. The description provides enough information to decide whether the functionality of the system is affected.
2. **Context factor (γ):** This factor is obtained by comparing the value of the Attack Vector (AV) submetric of the CVSS, with the real environment of deployment of the SUT.
3. **Exploit factor (μ):** To obtain this factor, public databases have to be queried to find any potential exploit for each vulnerability.
4. **Deepness factor (β):** For any given CVSS, its value is obtained according to the deepness in the exploit chain for the SUT [199].

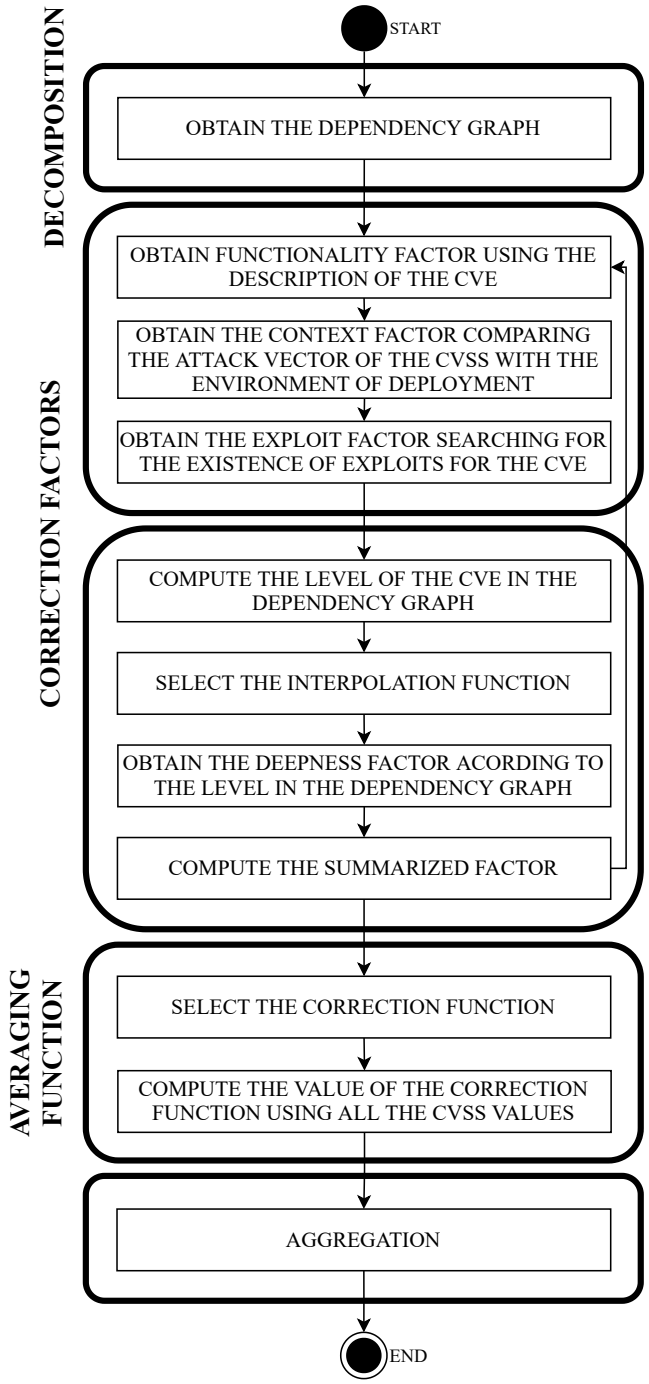

FIGURE 6.2: Flowchart showing the main steps of the aggregation algorithm for each CVSS.

Summarized factor for each CVSS

The summarized factor, λ, is obtained by multiplying all the corrections factors obtained in the previous step, following Equation 6.1.

Corrected CVSS values

The corrected CVSS values are obtained by multiplying each CVSS by its corresponding summarized factor, λ. At this point, it is necessary to check for overflows, because the exploitation factor generated corrected CVSS values higher than 10. Values higher than 10 are set to 10 at this stage.

Correction function

At this point, it is necessary to choose an averaging function. Choosing one function over the other will cause the aggregation result to grow slower or faster toward 10 in each addition. In this case, and for the sake of clarity, we chose the arithmetic mean, but any other kind of mean (*e.g., harmonic mean*) could be used according to each scenario.

Aggregation

Finally, the aggregated value is computed using Equation 6.2.

6.1.4 Interpretation of the result

The advantage of this method is that the result can be interpreted in the same way that a normal CVSS would be interpreted. This is because of the correction factors in Equation 6.2, that only let the algorithm return high values when vulnerabilities with high CVSS values are exploitable in reality (λ is close to 1). This mechanism ensures that multiple aggregated low CVSS values do not result in a critical score just because there are a large number of them.

6.2 Use Case

To test the potential of our proposal, we analyzed Version 3 of OpenPLC project, obtaining a CVSS aggregated value for its vulnerabilities using the proposed algorithm.

OpenPLC is the first functional open source Programmable Logic Controller (PLC), both in software and hardware [187]. It was mainly created for research purposes, because it provides its entire source code [188, 189]. The current version of the project is OpenPLC V3 [192].

6.2.1 Use Case Scenario

For this use case, we are going to make the next assumptions:

- The system executing OpenPLC V3 is deployed in an isolated network.
- The system running OpenPLC V3 is physically isolated.
- The attacker is an insider without access to the systems.
- The reference point for the deepness factor will be the `webserver.py` in Figure 6.3.

6.2.2 Structure of OpenPLC

The first step was to obtain the inner structure of OpenPLC V3 using the Extended Dependency Graph (EDG) proposed in [199]. To simplify the obtained graph, we only represented the shortest path to each node, so the worst case scenario (more accessible from the outside) is considered. The result is shown in Figure 6.3.

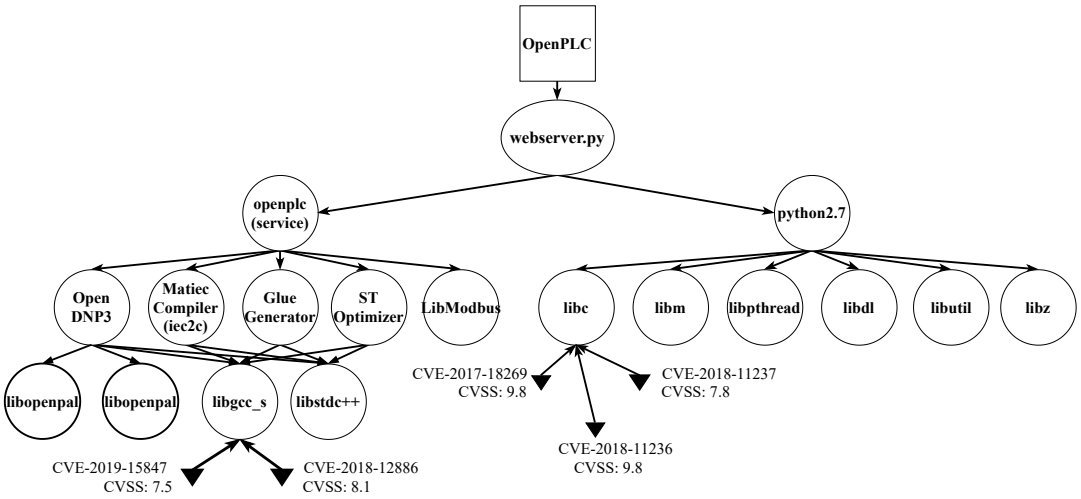

FIGURE 6.3: Extended Dependency Graph of OpenPLC V3. Circles represent individual assets, black triangles are the vulnerabilities associated to each asset, and the square represent the entry point to the system, or root node of dependency.

TABLE 6.3: Vulnerabilities present in OpenPLC V3. For each one, the CVSS is shown, together with their associated Attack Vector (AV), and their correction factors.

CVE	CVSS	Attack Vector	Functionality (ρ)	Deepness (β)	Context (γ)	Exploit (μ)	Summarized (λ)	Corrected CVSS
CVE-2017-18269	9.8	Network	1	0.5	1	1.25	0.625	6.125
CVE-2018-11236	9.8	Network	0	0.5	1	0	0	0
CVE-2018-11237	7.8	Local	1	0.5	0	1.25	0	0
CVE-2018-12886	8.1	Network	1	0.25	1	1.25	0.313	2.530
CVE-2019-15847	7.5	Network	1	0.25	1	1.25	0.313	2.344

6.2.3 Calculation of the Correcting Factors

OpenPLC V3 has five vulnerabilities: two vulnerabilities affecting `libgcc_s`, and three vulnerabilities affecting `libc`. Table 6.3 shows each vulnerability in more detail.

From these data, it is possible to obtain all corrections factors for each vulnerability, as follows (Table 6.3 summarizes the results):

Functionality Factor (ρ)

This factor is obtained from the analysis of the description of each CVE. From these data, we have to decide whether the functionality of OpenPLC V3 is affected ("1") or not ("0").

Deepness Factor (β)

By taking a look at Figure 6.3, it can be seen that the maximum deepness level is four. So the possible values for the deepness factor are the ones shown in Figure 6.1. More precisely, vulnerabilities CVE-2019-15847 and CVE-2018-12886 have a deepness factor of 0.25, because they are at level four. By contrast, vulnerabilities CVE-2017-18269, CVE-2018-11236, and CVE-2018-11237 have a deepness factor of 0.5, because they are at level three.

Context Factor (γ)

From the initial assumptions, insiders can only exploit the existing vulnerabilities from the local network. This means that every vulnerability that has an attack vector of "network" (N) can be exploited, thus CVE-2017-18269, CVE-2018-11236, CVE-2018-12886, and CVE-2019-15847 are exploitable by the attacker. Vulnerabilities whose attack vector is "local" (L) cannot be exploited, because physical access is needed. Therefore, CVE-2018-11237 cannot be exploited.

Exploit Factor (μ)

Public databases have to be queried to find existing exploits for each vulnerability. According to their state of development, a different value is assigned.

Summarized Factor (λ)

The summarized factor for each vulnerability is obtained as the product of the previous factors, as shown in Equation 6.1. At this step, by taking a look at the resulting values of λ, it is possible to know which CVSS will contribute to the final aggregation and in which percentage ($\lambda > 0$), and which ones will not contribute at all ($\lambda = 0$).

Average Factor (σ)

Finally, we obtained the average factor by calculating the arithmetic mean of all the initial CVSS values: $\sigma = 8.6$.

6.2.4 Aggregation

The previous step before the aggregation is obtaining the corrected CVSS value for each initial CVSS. This is done by multiplying each CVSS by their corresponding summarized value (λ). The corrected values are shown in Table 6.3.

Finally, the aggregation is performed using the corrected CVSS values. The aggregation is an iterative process that takes the first two values to be added, and adds them using Equation 6.2. Then, this result is added to the third value to be added, and so on, until there are no more values.

For OpenPLC V3, this process returns a final aggregated value of 9.1. Without the correction factors, the result would be 10. Nevertheless, taking into account features such as the exploitability of the vulnerabilities, the context of the SUT, or its functionalities, we can select the most important CVSS values to be aggregated. With such process, the total amount of CVSS values to be added is simplified. This also helps to simplify potential attack paths.

This result was obtained aggregating three of the five CVSS values present in OpenPLC V3. The associated CVSS for CVE-2018-11236 and CVE-2018-11237 were not taking into account for the aggregation, because they do not affect to any functionality of the system, Moreover, CVE-2018-11237 cannot be exploited in the conditions described in the use case.

CVE-2017-18269 (with an associated CVSS of 9.8) is the vulnerability with the highest value for λ. Therefore, it is going to contribute the most to the final aggregated value. CVE-2018-12886 and CVE-2019-15847 follow with a CVSS of 8.1 and 7.5 respectively. As it is shown, the selected vulnerabilities have a high CVSS, so it is expected that the aggregated value would be also high. This is reflected in the obtained result of 9.1.

Finally, it is worth highlighting that the final result is lower than the highest CVSS value present in OpenPLC V3. This difference is due to the effect of the correction factors: as the CVE-2017-18269 is further away from the entry point of the system (in layer 3), its real CVSS value in lower.

6.3 Conclusions and Future Work

In this research work, we proposed a new aggregation algorithm for CVSS values. The proposed approach integrates correction factors to select the most relevant CVSS values to be added based on contextual information. For each vulnerability, we check for:

1. Functionality disruption.
2. Exploitation difficulty.
3. Existence of exploits, and their development state.
4. Context of deployment.

We assigned a different correction factor to each one of the previous properties to further ponder the initial CVSS value and adjust it to the real context where the system is operating.

The proposed aggregation algorithm was applied to OpenPLC V3 in a use case. Two of the existing vulnerabilities were filtered out by the algorithm, as they cannot be exploited in the described context of OpenPLC V3. The rest of the vulnerabilities were aggregated, and the result (9.1) was indeed lower than the highest CVSS present in the system (9.8). This shows that the CVSS for each vulnerability was correctly adjusted to the real context of deployment of OpenPLC V3.

As future work, we plan to perform the aggregation at the submetric level of the CVSS, instead of using the base metric value, giving more granular values for each factor.

Chapter 7

Conclusions and Future Work

ESs constitute a critical piece for the industry, where they are present in almost every field, including critical infrastructures. Moreover, they are also rapidly evolving, and increasing in number. Under this scenario, a successful attack could have disastrous consequences, both economic and human losses. These characteristics together highlight the necessity of a methodology to assess the level of security of ESs.

This thesis focuses on quantifying the security level of ESs. For this reason, we reviewed both industrial standards and scientific literature to collect existing metrics. The properties of a comprehensive metrics were also found during this analysis. After analyzing the data, we found that most of the collected metrics are design to assess the level of security of an organization, and cannot be applied to ESs. Those that can be applied to ES are focused on software-related aspect of the system, while hardware-related ones are often neglected. Because existing taxonomies are defined to be used at a very high level (organization level), we proposed a new ES-centered taxonomy to classify security metrics. This taxonomy focuses on the specific assessment needs and constrains of ESs.

On the other hand, we found that graph-based techniques that are typically used in computer networks analysis can be applied to ES. Representing both the inner dependencies of an ES and its vulnerabilities helps to find root causes, make a decision about patching prioritization, and to generate new requirements and test cases. With this idea in mind, we proposed the Extended Dependency Graphs (EDGs), a directed graph-based structure that is capable of representing the inner structure of an ES. Moreover, we integrate security standards such as the CVSS, and the CWE to quantify, and find potential root cause for each vulnerability. In addition, we were also able to generate new requirements, new test cases, and propose training.

Finally, to complement the previously proposed model, we developed an aggregation algorithm for CVSS values. The proposed approach integrates correction factors to select the most relevant CVSS values to be added based on contextual information. For each vulnerability, we check for functionality disruption, exploitation difficulty, existence of exploits, and context of deployment.

As future work, analyzing the obtained metrics in more deepness could be promising. Choosing a smaller set of metrics, and test them empirically to check their real applicability to ESs. This would allow a more detailed knowledge of existing metrics, their applicability, and the relation between the property they are measuring and the

returned value. Moreover, it would be interesting to know which metrics are indeed more used in the industry.

With regard to the EDGs, we plan to generate a tool that automates the generation of EDGs. This tool could also integrate automatic updating of the current state of the SUT. So with every change (discovered vulnerability, update release) the corresponding EDG would be updated. A system to include warnings, available exploits, or patching prioritization suggestions would be also interesting. Historical information about the developers can also be integrated into the EDG model to predict future vulnerabilities, and tendencies among the developers. Additionally, according to the properties of a comprehensive methodology described in Section 3.2.13 (Table 3.5), we should enhance the guidance and reporting of the EDGs.

Finally, we plan to explore the advantages of performing the aggregation at the submetric level of the CVSS, instead of using the base metric value, giving more granular values for each factor. Integrating this aggregation algorithm to the EDG tool, could lead to better analysis of each SUT.

Chapter 8

Bibliography

[1] O. Qingyu, L. Fang, and H. Kai, "High-security system primitive for embedded systems," in *2009 International Conference on Multimedia Information Networking and Security*, vol. 2, pp. 319–321, 2009. https://ieeexplore.ieee.org/document/5368926.

[2] T. M. Chen and S. Abu-Nimeh, "Lessons from stuxnet," *Computer*, vol. 44, pp. 91–93, Apr. 2011.

[3] M. Vai, B. Nahill, J. Kramer, M. Geis, D. Utin, D. Whelihan, and R. Khazan, "Secure architecture for embedded systems," in *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–5, 2015. https://ieeexplore.ieee.org/document/7322461.

[4] C.-W. Ten, G. Manimaran, and C.-C. Liu, "Cybersecurity for critical infrastructures: Attack and defense modeling," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 40, no. 4, pp. 853–865, 2010. https://ieeexplore.ieee.org/document/5477189.

[5] L. Gressl, C. Steger, and U. Neffe, "Design space exploration for secure iot devices and cyber-physical systems," *ACM Trans. Embed. Comput. Syst.*, vol. 20, May 2021. https://dl.acm.org/doi/10.1145/3430372.

[6] M. Gupta, M. Abdelsalam, S. Khorsandroo, and S. Mittal, "Security and privacy in smart farming: Challenges and opportunities," *IEEE Access*, vol. 8, pp. 34564–34584, 2020. https://ieeexplore.ieee.org/document/9003290.

[7] P. Marwedel, *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Springer, Jan. 2010.

[8] S. Mumtaz, A. Alsohaily, Z. Pang, A. Rayes, K. F. Tsang, and J. Rodriguez, "Massive internet of things for industrial applications: Addressing wireless iiot connectivity challenges and ecosystem fragmentation," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 28–33, 2017. https://ieeexplore.ieee.org/document/7883984.

[9] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A review of low-end, middle-end, and high-end iot devices," *IEEE Access*, vol. 6, pp. 70528–70554, 2018. https://ieeexplore.ieee.org/document/8528362.

[10] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim, "Internet of Things (IoT) for Next-Generation Smart Systems: A Review of Current Challenges, Future Trends and Prospects for Emerging 5G-IoT Scenarios," *IEEE Access*, vol. 8, pp. 23022–23040, 2020. https://ieeexplore.ieee.org/document/9103025.

[11] S. E. Ponta, H. Plate, and A. Sabetta, "Detection, assessment and mitigation of vulnerabilities in open source dependencies," *Empirical Software Engineering*, vol. 25, pp. 3175–3215, Sept. 2020. https://link.springer.com/content/pdf/10.1007/s10664-020-09830-x.pdf.

[12] Hejderup, J.I. and Van Deursen, A. and Mesbah, A., *In Dependencies We Trust: How vulnerable are dependencies in software modules?* PhD thesis, Department of Software Technology, TU Delft, may 2015. http://resolver.tudelft.nl/uuid:3a15293b-16f6-4e9d-b6a2-f02cd52f1a9e.

[13] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vulnerable open source dependencies: Counting those that matter," in *Proceedings of the 12th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Oct 2018. https://dl.acm.org/doi/10.1145/3239235.3268920.

[14] I. Zografopoulos, J. Ospina, X. Liu, and C. Konstantinou, "Cyber-physical energy systems security: Threat modeling, risk assessment, resources, metrics, and case studies," *IEEE Access*, vol. 9, pp. 29775–29818, 2021. https://ieeexplore.ieee.org/document/9351954.

[15] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A.-R. Sadeghi, M. Maniatakos, and R. Karri, "The cybersecurity landscape in industrial control systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1039–1057, 2016. https://ieeexplore.ieee.org/document/7434576?reload=true&arnumber=7434576.

[16] A. Mathew, "Network slicing in 5g and the security concerns," in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 75–78, 2020. https://ieeexplore.ieee.org/abstract/document/9076479.

[17] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016. https://ieeexplore.ieee.org/document/7467408.

[18] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on iot security: Application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019. https://ieeexplore.ieee.org/document/8742551.

[19] M. Ayaz, M. Ammad-Uddin, Z. Sharif, A. Mansour, and E.-H. M. Aggoune, "Internet-of-things (iot)-based smart agriculture: Toward making the fields talk," *IEEE Access*, vol. 7, pp. 129551–129583, 2019. https://ieeexplore.ieee.org/document/8784034.

[20] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital twin: Enabling technologies, challenges and open research," *IEEE Access*, vol. 8, pp. 108952–108971, 2020.

[21] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018. https://ieeexplore.ieee.org/document/8359287.

[22] N. Benias and A. P. Markopoulos, "A review on the readiness level and cybersecurity challenges in industry 4.0," in *2017 South Eastern European Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, pp. 1–5, 2017. https://ieeexplore.ieee.org/document/8088234.

[23] W. Matsuda, M. Fujimoto, T. Aoyama, and T. Mitsunaga, "Cyber security risk assessment on industry 4.0 using ics testbed with ai and cloud," in *2019 IEEE Conference on Application, Information and Network Security (AINS)*, pp. 54–59, 2019. https://ieeexplore.ieee.org/document/8968698.

[24] G. Culot, F. Fattori, M. Podrecca, and M. Sartor, "Addressing industry 4.0 cybersecurity challenges," *IEEE Engineering Management Review*, vol. 47, no. 3, pp. 79–86, 2019. https://ieeexplore.ieee.org/document/8758411.

[25] M. Lezzi, M. Lazoi, and A. Corallo, "Cybersecurity for industry 4.0 in the current literature: A reference framework," *Computers in Industry*, vol. 103, pp. 97–110, 2018. https://www.sciencedirect.com/science/article/abs/pii/S0166361518303658.

[26] A. Ustundag and E. Cevikcan, *Industry 4.0: Managing The Digital Transformation*. Springer International Publishing, 2018. https://link.springer.com/book/10.1007/978-3-319-57870-5.

[27] L. Thames and D. Schaefer, eds., *Cybersecurity for Industry 4.0*. Springer International Publishing, 2017. https://link.springer.com/book/10.1007/978-3-319-50660-9.

[28] N. Medeiros, N. Ivaki, P. Costa, and M. Vieira, "Software metrics as indicators of security vulnerabilities," in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 216–227, 2017. https://ieeexplore.ieee.org/document/8109088.

[29] M. Alenezi and M. Zarour, "On the relationship between software complexity and security," *International Journal of Software Engineering & Applications (IJSEA)*, vol. 11, no. 1, 2020. https://aircconline.com/abstract/ijsea/v11n1/11120ijsea04.html.

[30] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security Privacy*, vol. 9, pp. 49–51, May 2011.

[31] G. George and S. M. Thampi, "A graph-based security framework for securing industrial iot networks from vulnerability exploitations," *IEEE Access*, vol. 6, pp. 43586–43601, 2018. https://ieeexplore.ieee.org/document/8430731.

[32] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, pp. 145–152, July 2015.

[33] B. B. Nielsen, M. T. Torp, and A. Møller, "Modular call graph construction for security scanning of node.js applications," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2021, (New York, NY, USA), p. 29–41, Association for Computing Machinery, 2021. https://dl.acm.org/doi/10.1145/3460319.3464836.

[34] R. E. Sawilla and X. Ou, "Identifying critical attack assets in dependency attack graphs," in *Computer Security - ESORICS 2008* (S. Jajodia and J. Lopez, eds.), (Berlin, Heidelberg), pp. 18–34, Springer Berlin Heidelberg, 2008. https://link.springer.com/chapter/10.1007/978-3-540-88313-5_2#citeas.

[35] I. Garitano, S. Fayyad, and J. Noll, "Multi-metrics approach for security, privacy and dependability in embedded systems," *Wireless Personal Communications*, 2015. https://link.springer.com/article/10.1007/s11277-015-2478-z.

[36] D. Kleidermacher and M. Kleidermacher, "Practical methods for safe and secure software and systems development," in *Embedded Systems Security* (D. Kleidermacher and M. Kleidermacher, eds.), Oxford: Newnes, 2012. https://www.sciencedirect.com/science/article/pii/B9780123868862000011.

[37] O. Andreeva, S. Gordeychik, G. Gritsai, O. Kochetova, E. Potseluevskaya, S. Sidorov, and A. Timorin, "Industrial control systems vulnerabilities statistics," tech. rep., 03 2016. https://www.researchgate.net/publication/337732465_INDUSTRIAL_CONTROL_SYSTEMS_VULNERABILITIES_STATISTICS.

[38] D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede, "Securing embedded systems," *IEEE Security Privacy*, vol. 4, no. 2, pp. 40–49, 2006. https://ieeexplore.ieee.org/document/1621059.

[39] J. Viega and H. Thompson, "The state of embedded-device security (spoiler alert: It's bad)," *IEEE Security Privacy*, vol. 10, no. 5, pp. 68–70, 2012. https://ieeexplore.ieee.org/document/6322974?section=abstract.

[40] P. Marwedel, "Embedded systems foundations of cyber-physical systems, and the internet of things," in *Embedded System Design*, Switzerland: Springer Nature, 2018. https://link.springer.com/book/10.1007%2F978-3-319-56045-8.

[41] P. Arpaia, F. Bonavolontà, A. Cioffi, and N. Moccaldi, "Reproducibility enhancement by optimized power analysis attacks in vulnerability assessment of iot transducers," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–8, 2021. https://ieeexplore.ieee.org/document/9521880.

[42] D. Kleidermacher and M. Kleidermacher, *Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development*. Newton, MA, USA: Newnes, 1st ed., 2012.

[43] T. Noergaard, *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Newton, MA, USA: Newnes, 2005.

[44] T. Noergaard, *Embedded systems architecture - a comprehensive guide for engineers and programmers*. Elsevier, 2005. https://www.elsevier.com/books/embedded-systems-architecture/noergaard/978-0-12-382196-6.

[45] S. Heath, *Embedded Systems Design*. Newnes, 2003.

[46] R. Kamal, *Embedded Systems: Architecture, Programming, and Design*. 01 2003.

[47] "Internet of things (iot): A review of enabling technologies, challenges, and open research issues," *Computer Networks*, vol. 144, pp. 17 – 39, 2018.

[48] "IEC 62443: Industrial Communication Networks—Network and System Security," standard, IEC Central Office, Geneva, Switzerland, 2010.

[49] International Electrotechnical Commission, "IEC 62443: Security for Industrial Automation and Control Systems — Part 4-2: Technical Security Requirements for IACS Components," standard, International Electrotechnical Commission, Geneva, Switzerland, 2019. https://www.isa.org/products/ansi-isa-62443-4-1-2018-security-for-industrial-au.

[50] CC, "The Common Criteria for Information Technology Security Evaluation - Introduction and General Model." https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf.

[51] MITRE Corporation, "CWE - Common Weakness Enumeration." https://cwe.mitre.org/about/faq.html.

[52] MITRE Corporation, "CVE - Common Vulnerabilities and Exposures." https://cve.mitre.org/about/terminology.html.

[53] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004. https://ieeexplore.ieee.org/document/1335465.

[54] NIST - National Institute of Standards and Technology, "National Vulnerability database (NVD)." https://nvd.nist.gov/.

[55] Common Criteria (CC), "Part 3: Security Assurance Components." https://commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf.

[56] W. He, H. Li, and J. Li, "Unknown vulnerability risk assessment based on directed graph models: A survey," *IEEE Access*, vol. 7, pp. 168201–168225, 2019. https://ieeexplore.ieee.org/abstract/document/8906081.

[57] MITRE Corporation, "CAPEC - Common Attack Pattern Enumeration and Classification." https://capec.mitre.org/about/glossary.html.

[58] National Institute for Standards and Technology (NIST), "CPE - Common Platform Enumeration." https://nvd.nist.gov/products/cpe.

[59] Brant A. Cheikes and David Waltermire and Karen Scarfone, "NIST Interagency Report 7695 - Common Platform Enumeration: Naming Specification Version 2.3," nist interagency report, National Institute for Standards and Technology (NIST), Gaithersburg, Maryland, 2011. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=909010.

[60] Mary C. Parmelee and Harold Booth and David Waltermire and Karen Scarfone, "NIST Interagency Report 7696 - Common Platform Enumeration: Name

Matching Specification Version 2.3," nist interagency report, National Institute for Standards and Technology (NIST), Gaithersburg, Maryland, 2011. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=909008.

[61] "National Institute for Standards and Technology (NIST)." https://www.nist.gov/.

[62] MITRE Corporation, "CWE - Common Weakness Enumeration." https://cwe.mitre.org/index.html.

[63] "MITRE Corporation." https://www.mitre.org/.

[64] MITRE Corporation, "CVE - Common Vulnerability and Exposures." https://cve.mitre.org/index.html.

[65] National Institute for Standards and Technology (NIST), "National Vulnerability Database NVD — Vulnerabilities." https://nvd.nist.gov/vuln/full-listing.

[66] A. Dimitriadis, J. L. Flores, B. Kulvatunyou, N. Ivezic, and I. Mavridis, "Ares: Automated risk estimation in smart sensor environments," *Sensors*, vol. 20, no. 16, 2020. https://www.mdpi.com/1424-8220/20/16/4617.

[67] FIRST - global Forum of Incident Response and Security Teams, "Common Vulnerability Scoring System (CVSS)." https://www.first.org/cvss/.

[68] MITRE Corporation, "CAPEC - Common Attack Pattern Enumeration and Classification." https://capec.mitre.org/.

[69] M. E. Khan and F. Khan, "A comparative study of white box, black box and grey box testing techniques," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 6, 2012.

[70] T. Zeb, M. Yousaf, H. Afzal, and M. R. Mufti, "A quantitative security metric model for security controls: Secure virtual machine migration protocol as target of assessment," *China Communications*, vol. 15, no. 8, pp. 126–140, 2018. https://ieeexplore.ieee.org/document/8438279.

[71] A. Atzeni and A. Lioy, "Why to adopt a security metric? a brief survey," *Advances in Information Security*, vol. 23, pp. 1 – 12, 2006. https://link.springer.com/chapter/10.1007%2F978-0-387-36584-8_1.

[72] S. M. Bellovin, "On the brittleness of software and the infeasibility of security metrics," *IEEE Security & Privacy*, vol. 4, no. 4, pp. 96–96, 2006.

[73] V. Verendel, "Quantified security is a weak hypothesis: A critical survey of results and assumptions," in *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*, NSPW '09, (New York, NY, USA), pp. 37–50, ACM, 2009.

[74] S. Stolfo, S. M. Bellovin, and D. Evans, "Measuring security," *IEEE Security Privacy*, May 2011. https://ieeexplore.ieee.org/document/5432146.

[75] S. Pfleeger and R. Cunningham, "Why measuring security is hard," *IEEE Security Privacy*, July 2010. https://ieeexplore.ieee.org/document/5432146.

[76] V. Verendel, *Some problems in quantified security*. Chalmers University of Technology, 2010.

[77] R. M. Savola, "Quality of security metrics and measurements," *Computers & Security*, vol. 37, pp. 78 – 90, 2013. https://www.sciencedirect.com/science/article/pii/S0167404813000850.

[78] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," *ACM Comput. Surv.*, December 2016. https://ieeexplore.ieee.org/document/5482589.

[79] A. Ramos, M. Lazar, R. H. Filho, and J. J. P. C. Rodrigues, "Model-based quantitative network security metrics: A survey," *IEEE Communications Surveys Tutorials*, vol. 19, pp. 2704–2734, Fourthquarter 2017. https://ieeexplore.ieee.org/document/8017389.

[80] K. Beckers, I. Côté, S. Fenz, D. Hatebur, and M. Heisel, *A Structured Comparison of Security Standards*, pp. 1–34. Cham: Springer International Publishing, 2014.

[81] ISECOM, *The Open Source Security Testing Methodology Manual (OSSTMM)*. Institute for Security and Open Methodologies, 2010. https://www.isecom.org/OSSTMM.3.pdf.

[82] CC, "Common Methodology for Information Security Technology Evaluation." https://www.commoncriteriaportal.org/files/ccfiles/CEMV3.1R5.pdf.

[83] E. Chew, M. Swanson, K. Stine, N. Bartol, A. Brown, and W. Robinson, "Performance measurement guide for information security (draft)," *NIST Special Publication 800-55*, July 2008. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=152183.

[84] M. Swanson, "Security self-assessment guide for information technology systems," *NIST Special Publication 800-26*, November 2001. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-26.pdf.

[85] M. Swanson, N. Bartol, J. Sabato, J. Hash, and L. Graffo, "Security metrics guide for information technology systems," *NIST Special Publication 800-55*, July 2003. https://pdfs.semanticscholar.org/ad77/fc87f1bf93225d2c1a0bb5ad831b907ae6fc.pdf.

[86] W. Jansen, "Directions in security metrics research," tech. rep., National Institute of Standards and Technology (NIST), April 2009. https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7564.pdf.

[87] IIC, "Industrial Internet of Things Volume G4: Security Framework," 2016. https://www.iiconsortium.org/pdf/IIC_PUB_G4_V1.00_PB.pdf.

[88] I. S. for Automation, "Security for industrial automation and control systems," standard, ISA, 2009.

[89] International Electrotechnical Commission, "IEC 62443: Security for Industrial Automation and Control Systems — Part 4-1: Secure Product Development Lifecycle Requirements," standard, International Electrotechnical Commission, Geneva, Switzerland, 2018.

[90] M. Rudolph and R. Schwarz, "A critical survey of security indicator approaches," in *2012 Seventh International Conference on Availability, Reliability and Security*, pp. 291–300, Aug 2012. https://ieeexplore.ieee.org/document/6329197.

[91] S. Sentilles, E. Papatheocharous, and F. Ciccozzi, "What Do We Know about Software Security Evaluation? A Preliminary Study," in *QuASoQ@APSEC*, 2018. http://ceur-ws.org/Vol-2273/QuASoQ-04.pdf.

[92] S. S. Stevens, "On the theory of scales of measurement," *Science*, vol. 103, no. 2684, pp. 677–680, 1946. https://science.sciencemag.org/content/103/2684/677.

[93] R. Savola, "A security metrics taxonomization model for software-intensive systems," *Journal of Information Processing Systems*, 2009. http://jips-k.org/q.jips?cp=pp&pn=103.

[94] G. T. Doran, "There's a s.m.a.r.t. way to write managements's goals and objectives.," *Management Review*, vol. 70, no. 11, pp. 35–36, 1981.

[95] W. Krag Brotby and Gary Hinson, *PRAGMATIC Security Metrics. Applying Metametrics to Information Security*. CRC Press, 01 2013.

[96] G. Jelen, "SSE-CMM Security Metrics," in *National Institute of Standards and Techonology (NIST) and Computer System Security and Privacy Advisory Board (CSSPAB) Workshop*, June 2000.

[97] A. Jaquith, *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional, 01 2007.

[98] R. Savola, "On the feasibility of utilizing security metrics in software-intensive systems," *International Journal of Computer Science and Network Security*, vol. 10, pp. 230–239, Jan. 2010.

[99] R. Savola, "Towards a security metrics taxonomy for the information and communication technology industry," in *International Conference on Software Engineering Advances (ICSEA 2007)*, pp. 60–60, Aug. 2007.

[100] W. H. Sanders, "Quantitative security metrics: Unattainable holy grail or a vital breakthrough within our reach?," *IEEE Security Privacy*, Mar 2014. https://ieeexplore.ieee.org/document/6798561.

[101] P. Black, K. Scarfone, and M. Souppaya, *Cyber Security Metrics and Measures*, ch. 2. John Wiley & Sons, 2008. https://www.researchgate.net/publication/227988213_Cyber_Security_Metrics_and_Measures.

[102] P. Mukherjee and C. Mazumdar, "Security concern" as a metric for enterprise business processes," *IEEE Systems Journal*, 2019. https://ieeexplore.ieee.org/abstract/document/8732670.

[103] G. O. M. Yee, "Designing good security metrics," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Jul 2019. https://ieeexplore.ieee.org/document/8754182.

[104] A. Jaquith, *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional, 2007.

[105] L. Hayden, *IT Security Metrics: A Practical Framework for Measuring Security and Protecting Data*. McGraw-Hill Education, 08 2010.

[106] George Campbell, *Measuring and Communicating Security's Value A Compendium of Metrics for Enterprise Protection*. Elsevier, 04 2015.

[107] Marnix Dekker, and Christoffer Karsberg, "Guideline on Threats and Assets: Technical guidance on threats and assets in Article 13a," tech. rep., European Union Agency For Network And Information Security, 2015. `https://www.enisa.europa.eu/publications/technical-guideline-on-threats-and-assets`.

[108] CC, "The Common Criteria for Information Technology Security Evaluation." `https://www.commoncriteriaportal.org/`.

[109] National Cyber Secure Centre (NCSC), "Commercial Product Assurance (CPA): The CPA Build Standard," Standard, National Cyber Secure Centre (NCSC), Oct 2018.

[110] National Cyber Secure Centre (NCSC), "Process for Performing Commercial Product Assurance (CPA) Foundation Grade Evaluations," Standard, National Cyber Secure Centre (NCSC), Oct 2018.

[111] Agence nationale de la sécurité des systèmes d'information (ANSSI), "Certification de Sécurité de Premier Niveau (CSPN)," Standard, Agence nationale de la sécurité des systèmes d'information (ANSSI)l, Apr 2014.

[112] European Commission, "IACS Cybersecurity Certification Framework (ICCF): Lessons from the 2017 study of the state of the art," standard, ERNCIP Thematic Group: European IACS Cybersecurity Certification, 2018.

[113] OISSG, "Information Systems Security Assessment Framework (ISSAF)." `http://cuchillac.net/archivos/pre_seguridad_pymes/2_hakeo_etico/lects/metodologia_oissg.pdf`.

[114] OISSG, "Information Systems Security Assessment Framework (ISSAF)." `https://sourceforge.net/projects/isstf/`.

[115] J. Searle, G. Rasche, A. Wright, and S. Dinnage, "NESCOR Guide to Penetration Testing for Electric Utilities." `http://smartgrid.epri.com/doc/NESCORGuidetoPenetrationTestingforElectricUtilities-v3-Final.pdf`.

[116] Cybersecurity Certification Centre of Singapore, "Cybersecurity Certification Guide," Standard, Cyber Security Agency of Singapore (CSA), 2021.

[117] J. Wack, M. Tracy, and M. Souppaya, *Guideline on Network Security Testing*. Washington, United States: National Institute of Standards and Technology (NIST), 2001. `http://www.elitesuites.com.bh/downloads/IT5%20Guideline%20for%20Network%20Security%20Testing_NIST.pdf`.

[118] ISECOM, "The Open Source Security Testing Methodology Manual (OSSTMM)." `http://www.isecom.org/research/osstmm.html`.

[119] OWASP, "The Open Web Application Security Project." https://www.owasp.org/images/1/19/OTGv4.pdf.

[120] PTES, "The Penetration Testing Execution Standard - High Level Organization." http://www.pentest-standard.org/index.php/Main_Page.

[121] PTES, "The Penetration Testing Execution Standard - Techincal Guidelines." http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines.

[122] CC, "The Common Criteria for Information Technology Security Evaluation - Publications." https://www.commoncriteriaportal.org/cc/.

[123] "Under attack – GCN."

[124] S. N. Matheu, J. L. Hernandez-Ramos, and A. F. Skarmeta, "Toward a cybersecurity certification framework for the internet of things," *IEEE Security Privacy*, vol. 17, no. 3, pp. 66–76, 2019.

[125] OISSG, "The Open Information Systems Security Group." http://www.oissg.org/.

[126] M. Prandini and M. Ramilli, "Towards a practical and effective security testing methodology," in *The IEEE symposium on Computers and Communications*, pp. 320–325, June 2010. https://ieeexplore.ieee.org/document/5546813.

[127] J. Searle, G. Rasche, A. Wright, and S. Dinnage, "The National Electric Sector Cybersecurity Organization Resource (NESCOR)." http://smartgrid.epri.com/NESCOR.aspx.

[128] M. Ring, *Systematische Security-Tests von Kraftfahrzeugen*. PhD thesis, Universität Ulm, 2019. https://oparu.uni-ulm.de/xmlui/handle/123456789/12225.

[129] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, "NIST SP 800-115," Jan. 2020. https://www.nist.gov/privacy-framework/nist-sp-800-115.

[130] ISECOM, "Institute for Security and Open Methodologies." http://www.isecom.org/.

[131] OWASP, "The Open Web Application Security Project." https://www.owasp.org/index.php/OWASP_Testing_Project.

[132] OWASP, "The Open Web Application Security Project." https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project.

[133] O. Diachuk, "Guide to Modern Penetration Testing [Part 2] | Grey Box Penetration Testing | Infopulse." https://www.infopulse.com/blog/guide-to-modern-penetration-testing-part-2-fifty-shades-of-grey-box/.

[134] D. D. Bertoglio and A. F. Zorzo, "Tramonto: Uma estratégia de recomendação para testes de penetração," in *XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, 2016. http://hdl.handle.net/10923/13183.

[135] B. E. A. Plans, "Assessing security and privacy controls in federal information systems and organizations," *NIST Special Publication*, vol. 800, p. 53A, 2014. https://rmf.org/wp-content/uploads/2017/10/NIST.SP_.800-53Ar4.pdf.

[136] H. H. Thompson, "Why security testing is hard," *IEEE Security Privacy*, July 2003. https://ieeexplore.ieee.org/document/1219078.

[137] M. Metheny, "Chapter 10 - security testing: Vulnerability assessments and penetration testing," in *Federal Cloud Computing (Second Edition)* (M. Metheny, ed.), pp. 379 – 400, Syngress, second edition ed., 2017. http://www.sciencedirect.com/science/article/pii/B978012809710600010X.

[138] R. Bachmann and A. D. Brucker, "Developing secure software," *Datenschutz und Datensicherheit - DuD*, vol. 38, pp. 257–261, Apr 2014. https://doi.org/10.1007/s11623-014-0102-0.

[139] M. Meucci and A. Muller, *OWASP Testing Guide 4.0*. OWASP Foundation, 2014. https://www.owasp.org/images/1/19/OTGv4.pdf.

[140] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Chapter one - security testing: A survey," in *Advances in Computers* (A. Memon, ed.), vol. 101, pp. 1 – 51, Elsevier, 2016. http://www.sciencedirect.com/science/article/pii/S0065245815000649.

[141] E. Conrad, S. Misenar, and J. Feldman, "Chapter 7 - domain 6: Security assessment and testing (designing, performing, and analyzing security testing)"," in *CISSP Study Guide (Third Edition)* (E. Conrad, S. Misenar, and J. Feldman, eds.), pp. 329 – 345, Boston: Syngress, third edition ed., 2016. http://www.sciencedirect.com/science/article/pii/B9780128024379000072.

[142] Guru99, "What is Security Testing?," last visited 01/04/2019. https://www.guru99.com/what-is-security-testing.html.

[143] H. Hemmati, "Chapter four - advances in techniques for test prioritization," in *Advances in Computers* (A. M. Memon, ed.), vol. 112 of *Advances in Computers*, pp. 185–221, Elsevier, 2019. https://www.sciencedirect.com/science/article/pii/S0065245817300566.

[144] M. Eceiza, J. L. Flores, and M. Iturbe, "Fuzzing the internet of things: A review on the techniques and challenges for efficient vulnerability discovery in embedded systems," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10390–10411, 2021. https://ieeexplore.ieee.org/abstract/document/9344712.

[145] "Information technology —– Security techniques —– Vulnerability handling processes," standard, International Organization for Standardization, Geneva, CH, oct 2019. https://www.iso.org/standard/69725.html.

[146] D. Herrmann, *Using the Common Criteria for IT Security Evaluation*. 12 2002. https://www.taylorfrancis.com/books/mono/10.1201/9781420031423/using-common-criteria-security-evaluation-debra-herrmann.

[147] D. Mellado, E. Fernández-Medina, and M. Piattini, "A common criteria based security requirements engineering process for the development of secure information systems," *Computer Standards & Interfaces*, vol. 29, no. 2, pp. 244–253, 2007. https://www.sciencedirect.com/science/article/pii/S0920548906000511.

[148] A. Hohenegger, G. Krummeck, J. Baños, A. Ortega, M. Hager, J. Sterba, T. Kertis, P. Novobilsky, J. Prochazka, B. Caracuel, A. L. Sanz, F. Ramos, H. Blasum, M. Brotz, C. Gries, T. Vögler, J. Neškudla, J. Rollo, L. Burgstaller, M. Truskaller, K.-M. Koch, Technikon, R. Hametner, S. Rauscher, P. Tummeltshammer, T. Austria, F. Golatowski, and T. Schulz, "Security certification experience for industrial cyberphysical systems using common criteria and iec 62443 certifications in certmils," in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, pp. 25–30, 2021. https://ieeexplore.ieee.org/document/9468241.

[149] J. Homer, X. Ou, and D. Schmidt, "A sound and practical approach to quantifying security risk in enterprise networks," tech. rep., 2009. https://www.cse.usf.edu/~xou/publications/tr_homer_0809.pdf.

[150] S. Zhang, X. Ou, A. Singhal, and J. Homer, "An empirical study of a vulnerability metric aggregation method," tech. rep., KANSAS STATE UNIV MANHATTAN, 2011. https://www.cse.usf.edu/~xou/publications/stmacip11.pdf.

[151] J. Homer, S. Zhang, X. Ou, D. Schmidt, Y. Du, S. R. Rajagopalan, and A. Singhal, "Aggregating vulnerability metrics in enterprise networks using attack graphs," *Journal of Computer Security*, vol. 21, no. 4, pp. 561–597, 2013. https://content.iospress.com/articles/journal-of-computer-security/jcs475.

[152] G. George and S. M. Thampi, "A graph-based security framework for securing industrial iot networks from vulnerability exploitations," *IEEE Access*, vol. 6, pp. 43586–43601, 2018. https://ieeexplore.ieee.org/document/8430731.

[153] S. Li, Y. Chen, X. Wu, X. Cheng, and Z. Tian, "Power Grid-Oriented Cascading Failure Vulnerability Identifying Method Based on Wireless Sensors," *Journal of Sensors*, vol. 2021, p. 8820413, June 2021. https://www.hindawi.com/journals/js/2021/8820413/.

[154] B. Liu, G. Zhu, X. Li, and R. Sun, "Vulnerability assessment of the urban rail transit network based on travel behavior analysis," *IEEE Access*, vol. 9, pp. 1407–1419, 2021. https://ieeexplore.ieee.org/document/9306836.

[155] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012. https://ieeexplore.ieee.org/document/5936075.

[156] L. Muñoz-González, D. Sgandurra, M. Barrère, and E. C. Lupu, "Exact inference techniques for the analysis of bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 231–244, 2019. https://ieeexplore.ieee.org/document/7885532.

[157] X. Liu, C. Qian, W. G. Hatcher, H. Xu, W. Liao, and W. Yu, "Secure internet of things (iot)-based smart-world critical infrastructures: Survey, case study and research opportunities," *IEEE Access*, vol. 7, pp. 79523–79544, 2019. https://ieeexplore.ieee.org/document/8730298.

[158] J. Hu, S. Guo, X. Kuang, F. Meng, D. Hu, and Z. Shi, "I-hmm-based multidimensional network security risk assessment," *IEEE Access*, vol. 8, pp. 1431–1442, 2020. https://ieeexplore.ieee.org/document/8941077.

[159] I. Zografopoulos, J. Ospina, X. Liu, and C. Konstantinou, "Cyber-physical energy systems security: Threat modeling, risk assessment, resources, metrics, and case studies," *IEEE Access*, vol. 9, pp. 29775–29818, 2021. https://ieeexplore.ieee.org/document/9351954.

[160] M. Khosravi-Farmad and A. Bafghi, "Bayesian decision network-based security risk management framework," *Journal of Network and Systems Management*, vol. 28, 10 2020. https://link.springer.com/article/10.1007/s10922-020-09558-5.

[161] W. H. Sanders, "Quantitative security metrics: Unattainable holy grail or a vital breakthrough within our reach?," *IEEE Security Privacy*, vol. 12, no. 2, pp. 67–69, 2014.

[162] D. G. Ángel Longueira-Romero, Rosa Iglesias and I. Garitano, "How to Quantify the Security Level of Embedded Systems? A Taxonomy of Security Metrics," in *Proceedings of the 2020 18th IEEE International Conference on Industrial Informatics (INDIN)*, 2020.

[163] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," vol. 49, dec 2016.

[164] H. Howland, "Cvss: Ubiquitous and broken," *Digital Threats*, sep 2021.

[165] P. Cheng, L. Wang, S. Jajodia, and A. Singhal, "Aggregating cvss base scores for semantics-rich network security metrics," in *2012 IEEE 31st Symposium on Reliable Distributed Systems*, pp. 31–40, 2012.

[166] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, "Dag-based attack and defense modeling: Don't miss the forest for the attack trees," *Computer Science Review*, vol. 13-14, pp. 1–38, 2014.

[167] T. Heyman, R. Scandariato, C. Huygens, and W. Joosen, "Using security patterns to combine security metrics," in *2008 Third International Conference on Availability, Reliability and Security*, pp. 1156–1163, 2008.

[168] Z. Song, Y. Wang, P. Zong, Z. Ren, and D. Qi, "An empirical study of comparison of code metric aggregation methods–on embedded software," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 114–119, 2019.

[169] M. Walkowski, M. Krakowiak, M. Jaroszewski, J. Oko, and S. Sujecki, "Automatic cvss-based vulnerability prioritization and response with context information," in *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pp. 1–6, 2021.

[170] C. Fruhwirth and T. Mannisto, "Improving cvss-based vulnerability prioritization and response with context information," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 535–544, 2009.

[171] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An attack graph-based probabilistic security metric," in *Data and Applications Security XXII* (V. Atluri, ed.), (Berlin, Heidelberg), pp. 283–296, Springer Berlin Heidelberg, 2008.

[172] S. Zhang, X. Ou, A. Singhal, and J. Homer, "An empirical study of a vulnerability metric aggregation method," in *Mission Assurance and Critical Infrastructure Protection*, 2011 World Congress in Computer Science, 2011-08-18 00:08:00 2011.

[173] N. Idika and B. Bhargava, "Extending attack graph-based security metrics and aggregating their application," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 75–85, 2012.

[174] M. Zhang, L. Wang, S. Jajodia, and A. Singhal, "Network attack surface: Lifting the concept of attack surface to the network level for evaluating networks' resilience against zero-day attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 310–324, 2021.

[175] J. Homer, S. Zhang, X. Ou, D. Schmidt, Y. Du, S. R. Rajagopalan, and A. Singhal, "Aggregating vulnerability metrics in enterprise networks using attack graphs," *Journal of Computer Security*, vol. 21, no. 4, pp. 561–597, 2013.

[176] L. Gallon and J.-J. Bascou, "Cvss attack graphs," in *2011 Seventh International Conference on Signal Image Technology Internet-Based Systems*, pp. 24–31, 2011.

[177] M. Frigault, L. Wang, A. Singhal, and S. Jajodia, "Measuring network security using dynamic bayesian network," in *Proceedings of the 4th ACM Workshop on Quality of Protection*, QoP '08, (New York, NY, USA), p. 23–30, Association for Computing Machinery, 2008.

[178] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic security risk management using bayesian attack graphs," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012.

[179] P. Xie, J. H. Li, X. Ou, P. Liu, and R. Levy, "Using bayesian networks for cyber security analysis," in *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, pp. 211–220, 2010.

[180] A. Ramos, M. Lazar, R. H. Filho, and J. J. P. C. Rodrigues, "Model-based quantitative network security metrics: A survey," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2704–2734, 2017.

[181] S. Yusuf Enoch, J. Hong, M. Ge, and D. Kim, "Composite metrics for network security analysis," vol. 2017, pp. 137–160, Feb. 2017.

[182] P. Morrison, D. Moye, and L. L. Williams, "Mapping the field of software security metrics," 2014.

[183] D. D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede, "Securing embedded systems," *IEEE Security Privacy*, vol. 4, no. 2, pp. 40–49, 2006.

[184] M. A. Amutio, J. Candau, and J. A. Mañas, "MAGERIT V3.0. Methodology for Information Systems Risk Analysis and Management. Book I - The Method," National Standard, Ministry of Finance and Public Administration, Madrid, Spain, 2014.

[185] ISO, *ISO/IEC 13335-1:2004 - Information technology — Security techniques — Management of information and communications technology security — Part 1: Concepts and models for information and communications technology security management*. Geneva, Switzerland: International Organization for Standardization.

[186] ISO, *ISO 8601:2019. Data and time -Representation for information interchange - Part 1: Basic rules*. Geneva, Switzerland: International Organization for Standardization.

[187] Thiago Alves, "OpenPLC Project." `https://www.openplcproject.com/`.

[188] T. R. Alves, M. Buratto, F. M. de Souza, and T. V. Rodrigues, "Openplc: An open source alternative to automation," in *IEEE Global Humanitarian Technology Conference (GHTC 2014)*, pp. 585–589, 2014. `https://ieeexplore.ieee.org/document/6970342`.

[189] T. Alves and T. Morris, "Openplc: An iec 61,131–3 compliant open source industrial controller for cyber security research," *Computers & Security*, vol. 78, pp. 364–379, 2018. `https://www.sciencedirect.com/science/article/pii/S0167404818305388?via%3Dihub`.

[190] Thiago Alves, "OpenPLC V1." `https://github.com/thiagoralves/OpenPLC`.

[191] Thiago Alves, "OpenPLC V2." `https://github.com/thiagoralves/OpenPLC_v2`.

[192] Thiago Alves, "OpenPLC V3." `https://github.com/thiagoralves/OpenPLC_v3`.

[193] "libssl1.0.0 : Trusty (14.04) : Ubuntu." `https://launchpad.net/ubuntu/trusty/+package/libssl1.0.0/+index`.

[194] "nodejs : Trusty (14.04) : Ubuntu." `https://launchpad.net/ubuntu/trusty/+package/nodejs/+index`.

[195] J. Spring, E. Hatleback, A. Householder, A. Manion, and D. Shick, "Time to change the cvss?," *IEEE Security & Privacy*, vol. 19, no. 2, pp. 74–78, 2021.

[196] M. A. Amutio, J. Candau, and J. A. Mañas, "MAGERIT V3.0. Methodology for Information Systems Risk Analysis and Management. Book III - Technical Guide," National Standard, Ministry of Finance and Public Administration, Madrid, Spain, 2012.

[197] M. A. Amutio, J. Candau, and J. A. Mañas, "MAGERIT V3.0. Methodology for Information Systems Risk Analysis and Management. Book I - The Method," National Standard, Ministry of Finance and Public Administration, Madrid, Spain, 2014.

[198] A. Syalim, Y. Hori, and K. Sakurai, "Comparison of risk analysis methods: Mehari, magerit, nist800-30 and microsoft's security management guide," in *2009 International Conference on Availability, Reliability and Security*, pp. 726–731, 2009.

[199] A. Longueira-Romero, R. Iglesias, J. L. Flores, and I. Garitano, "A novel model for vulnerability analysis through enhanced directed graphs and quantitative metrics," *Sensors*, vol. 22, no. 6, 2022.

Appendix A

Collected Metrics

A.1 Initial Set of Metrics

#	METRIC	DEFINITION	SCALE	SCOPE	AUTOMATION	MEASUREMENT	USABLE	REASON
1	False positives (FP)	Percentage of flagging genuine email as phishing email	RATIO	USER		DYNAMIC	NO	Phishing affects only people. It exploits a user vulnerability. This evaluates the user, not the device
2	False negatives (FN)	Percentage of detecting a phishing email as a genuine email	RATIO	USER		DYNAMIC	NO	Phishing affects only people. It exploits a user vulnerability. This evaluates the user, not the device
3	User's online behavior		RATIO	USER		DYNAMIC	NO	Phishing affects only people. It exploits a user vulnerability. This evaluates the user, not the device. This is only a suggestion. Not defined
4	Users' susceptibility to attacks		RATIO	USER		UNKNOWN	NO	This evaluates the user, not the device. This is only a suggestion. Not defined
5	Entropy			USER	AUTOMATIC	STATIC	YES	This could be used with keys instead of passwords to measure randomness
6	Password guessability (statistical / parameterized)	This metric measures the guessability of a set of passwords (rather than for individual passwords) by an "idealized" attacker with the assumption of a perfect guess / This metric measures the number of guesses an attacker needs to make via a particular cracking algorithm (i.e., a particular threat model) before recovering a particular password and training data. Different password cracking algorithms leads to different results	RATIO ABSOLUTE	USER	AUTOMATIC	STATIC	NO	Passwords are for people. Devices use keys (that are much more larger)
7	Password strength meter	This metric measures the password strength via an estimated effort to guess the password, while considering factors such as the character set (e.g., special symbols are required or not), password length, whitespace permissibility, password entropy, and blacklisted passwords being prevented or not. One variant metric is adaptive password strength, which is used to improve the accuracy of strength estimation	ORDINAL	USER	AUTOMATIC	STATIC	NO	Passwords are for people. Devices use keys (that are much more larger)
8	Attack surface	It aim to measure the ways by which an attacker can compromise a targeted software	ABSOLUTE	DEVICE		STATIC	YES	
9	Historical vulnerability metric	It measures the degree that a system is vulnerable (i.e., frequency of vulnerabilities) in the past	RATIO ABSOLUTE DISTRIBUTION	DEVICE		STATIC	YES	This could be part of a checklist of task, to check if there is any vulnerability that affects the device
10	Historically exploited vulnerability metric	It measures the number of vulnerabilities exploited in the past	RATIO ABSOLUTE DISTRIBUTION	DEVICE		STATIC	YES	This could be part of a checklist of task, to check if there is any vulnerability that affects the device
11	Future vulnerability metric	It measures the number of vulnerabilities that will be discovered during a future period of time	RATIO ABSOLUTE DISTRIBUTION	DEVICE		STATIC	NO	If this has to do with statistic and probabilities, I think it is better not to use it
12	Future exploited vulnerability metric	It measures the number of vulnerabilities that will be exploited during a future period of time	RATIO ABSOLUTE DISTRIBUTION	DEVICE		STATIC	NO	If this has to do with statistic and probabilities, I think it is better not to use it
13	Tendency-to-be-exploited metric	It measures the tendency that a vulnerability may be exploited, where the "tendency" may be derived from information sources such as posts on Twitter before vulnerability disclosures	RATIO ABSOLUTE DISTRIBUTION	DEVICE		STATIC	¿?	This research work might not be part of a security evaluation, but could be applied if so
14	Vulnerability lifetime	Client-end vulnerabilities Server-end vulnerabilities Cloud-end vulnerabilities	ABSOLUTE	NETWORK		STATIC	NO	Clearly related to nerworks
15	Attack vector	Describes whether a vulnerability can be exploited remotely, adjacently, locally, or physically (i.e., attacker must have physical access to the computer)	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
16	Attack complexity	Describes the conditions that must hold before an attacker can exploit the vulnerability, such as low or high	ORDINA	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
17	Privilege required	Describes the level of privileges that an attacker must have in order to exploit a vulnerability, such as none, low, or high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
18	User interaction	Describes whether the exploitation of a vulnerability requires the participation of a user (other than the attacker), such as none or required	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
19	Authorization scope	Describes whether or not a vulnerability has an impact on resources beyond its privileges (e.g., sandbox or virtual machine), such as unchanged or changed	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
20	Confidentiality impact	The impact of a successful exploitation of a vulnerability in terms of confidentiality such as none, low, high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
21	Integrity impact	The impact of a successful exploitation of a vulnerability in terms of integrity such as none, low, high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
22	Availability impact	The impact of a successful exploitation of a vulnerability in terms of availability, such as none, low, high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
23	Exploit code maturity	The likelihood a vulnerability can be attacked based on the current exploitation techniques, such as undefined, unproven, proof-of-concept, functional, or high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
24	Remediation level	Describes whether or not a remediation method is available for a given vulnerability, such as undefined, unavailable, workaround, temporary fix, or official fix	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
25	Report confidence	Measures the level of confidence for a given vulnerability as well as known technical details, such as unknown, reasonable, or confirmed	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
26	Security requirement	Describes environment-dependent security requirements in terms of confidentiality, integrity, and availability, such as not defined, low, medium, or high	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
27	Depth metric	Ratio of the diameter of a domain-level attack graph over the diameter in the most secure case, implying that the larger the diameter, the more secure the network	-	NETWORK		STATIC	NO	It is related to the topological properties of attack graphs
28	Existence, number, and lengths of attack paths metrics	It uses the attributes of attack paths from an initial state to the goal state. These metrics can be used to compare two attacks	-	NETWORK		STATIC	NO	It is related to the topological properties of attack graphs

29	Necessary defense metric	It estimates a minimal set of defense countermeasures necessary for thwarting a certain attack	-	DEVICE	STATIC	YES	This is related with the defenses that are necesary to implement so the system is secure for a certain vulnerability
30	Effort-to-security-failure metric	It measures an attacker's effort to reach its goal state	-	DEVICE	DYNAMIC*	YES	Although it might be difficult to estimate, this could be also included in the evaluation process
31	Weakest adversary metric	It estimates minimum adversary capabilities required to achieve an attack goal	-	DEVICE	DYNAMIC*	¿?	If the effort can be calculated in an objective manner, this could be used in embedded systems
32	k-zero-day-safety metric	It measures a number of zero-day vulnerabilities for an attacker to compromise a target	-	DEVICE	STATIC	NO	This has to do with probabilities. Can be applied, but I won't
33	Likelihood of exploitation	It measures the probability that an exploit will be executed by an attacker with certain capabilities	-	SOFTWARE	STATIC	¿?	This is related with statistics and risk assessment. It could be applied
34	Effective base metric / scope	Given an attack graph and a subset of exploits, the effective base metric aims to adjust the CVSS base metric by taking the highest value of the ancestors of an exploit to reflect the worst-case scenario, while the effective base score metric is calculated based on the effective base metric	-	SOFTWARE	STATIC	-	-
35	Reaction time metric	It captures the delay between the observation of the malicious entity at time t and the blacklisting of the malicious entity at time t (i.e., t - t)	ABSOLUTE	DEVICE	DYNAMIC	YES	If the device has any kind of detection mechanism, this could be used. Detection and reaction might have different times
36	Coverage metric	It estimates the portion of blacklisted malicious players	ORDINAL	DEVICE	STATIC	YES	This could be used, but the device has to have a blacklist
37	NO METRIC IS DEFINED	No metrics have been defined to measure the effectiveness of DEP. The effectiveness of DEP can be measured based on the probability of being compromised by a certain attack A(t) over all possible classes of attacks	-	DEVICE	-	-	No metric is defined
38	Average indirect target reduction	It counts the overall reduction of targets for any indirect control-flow transfer. It measures the overall reduction in terms of the number of targets exploitable by the attacker where smaller targets are more secure	ABSOLUTE	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
39	Average target size	Ratio between the size of the largest target and the number of targets	-	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
40	Evasion resistance	It is measured against control flow bending attacks, reflecting the effort (or premises) that an attacker must make (or satisfy) for evading the CFI scheme	-	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
41	Coverage metric	It measures the fraction of events detectable by a specific sensor deployment, reflecting a defender's need in monitoring events	-	DEVICE	DYNAMIC	UNKNOWN	UNKNOWN
42	Redundancy metric	It estimates the amount of evidence provided by a specific sensor deployment to detect an event	-	DEVICE	DYNAMIC	YES	This can be applied to devices as a whole, to check any mechanism of intrusion detection (mostly, tanti-tampering and software)
43	Confidence metric	It measures how well-deployed sensors detect an event in the presence of compromised sensors	-	DEVICE	DYNAMIC	YES	This can be applied to devices as a whole, to check any mechanism of intrusion detection (mostly, tanti-tampering and software)
44	Cost metric	It measures the amount of resources consumed by deploying sensors including the cost for operating and maintaining sensors	-	ORGANISATION	STATIC	NO	This is related with the cost of a specific countermeasure. This is more likely to be applied in early steps of the development
45	Detection time	For instrument-based attack detection, this metric is used to measure the delay between the time t0 at which a compromised computer sends its first scan packet and the time t that a scan packet is observed by the instrument	-	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
46	True-positive rate	It is the probability that an intrusion is detected as an attack	-	NETWORK	DYNAMIC	NO	This is clearly a network-oriented metric
47	False-negative rate	It is the probability that an intrusion is not detected as an attack	-	NETWORK	DYNAMIC	NO	This is clearly a network-oriented metric
48	True-negative rate	It is the probability that a non-intrusion is not detected as an attack	-	NETWORK	DYNAMIC	NO	This is clearly a network-oriented metric
49	False-positive rate	Also known as false alarm rate, it is the probability that a nonintrusion is detected as an attack	-	NETWORK	DYNAMIC	NO	This is clearly a network-oriented metric
50	Intrusion detection capability metric	It is the normalized metric of I(I, O) with respect to H(I) based on the base rate where I is the input to the IDS as a stream of 0/1 random variables (0 for benign/normal and 1 for malicious/abnormal), O is the output of the IDS as a stream of 0/1 random variables (0 for no alert or normal; 1 for alert / abnormal), H(I) and H(O), respectively, denote the entropy of I and O, and $I(I,O) = H(I) - H(I\|O)$ is the mutual information between I and O	-	NETWORK	UNKNOWN	NO	This is clearly a network-oriented metric
51	Receiver operating characteristic (ROC)	It reflects the dependence of the true-positive rate on the false-positive rate, reflecting a tradeoff between the true-positive and the false-positive rates	-	NETWORK	STATIC	NO	This is clearly a network-oriented metric
52	Intrusion detection operating characteristic (IDOC)	It describes the dependence of the true positive rate Pr(A | I) on the Bayesian detection rate Pr(I | A), while accommodating the base rate Pr(I)	-	NETWORK	STATIC	NO	This is clearly a network-oriented metric
53	Cost metric (damage / response / operational)	It includes the damage cost incurred by undetected attacks, the response cost spent on the reaction to detected attacks including both true and false alarms, and the operational cost for running an IDS	-	NETWORK	STATIC	NO	This is clearly a network-oriented metric, that aims to translate the impact into moneu
54	Relative effectiviness	This metric reflects the strength of a defense tool when employed in addition to other defense tools. A defense tool does not offer any extra strength if it cannot detect any attack undetected by other defense tools in place	RATIO	NETWORK	DYNAMIC	NO	This is clearly a network-oriented metric
55	Collective effectiviness	This metric measures the collective strength of IDSs and anti-malware programs	RATIO	NETWORK	DYNAMIC	NO	This is clearly a network-oriented metric
56	ASLR-induced Entropy metric		-				This is clearly a network-oriented metric
57	ASLR-induced Effective entropy metric	Measure of entropy in user space memory which quantitatively considers an adversary's ability to leverage low entropy regions of memory via absolute and dynamic inter-section connections	-	SOFTWARE	DYNAMIC	YES	If the device runs an OS, this could be applied
58	Moving Target Defense (MTD)	It measures the degree that an enterprise system can tolerate some undesirable security configurations in order to direct the global security state S(t) towards a desired stable system state	-	NETWORK	DYNAMIC	NO	This is only applicable to an enterprise system
59	Penetration resistance (PR)	Running a penetration test to estimate the level of effort (e.g., person-day or cost) required for a red team to penetrate into a system	-	DEVICE NETWORK	DYNAMIC	YES	Penetration testing is a really common way to test this kind of device

60	Network diversity (ND)	It measures the least or average effort an attacker must make to compromise a target entity based on the causal relationships between resource types to be considered as the inclusion in an attack graph	-	NETWORK	DYNAMIC	NO	This is clearly a network-oriented metric
61	Lifetime of zero-day attacks	It measures the period of time between when an attack was launched and when the corresponding vulnerability is disclosed to the public	-	SOFTWARE HARDWARE DEVICE	DYNAMIC	NO	This can be computed, but it is not feasible for a security evaluation
62	Victims by zero-day attacks	It measures the number of computers compromised by zero-day attacks	-	ORGANISATION	DYNAMIC*	NO	This is usefull for networks or an organisation. It can be computed for a network of devices, but it is not useful for a security evaluation
63	Susceptibility of a device to zero-day attacks	If we can capture the degree of the susceptibility of a device to zero-day attacks, then we can prioritize resource allocation to the ones requiring higher attention for mitigating the damage	-	DEVICE			This metric is proposed in [2] as a future concept, but it is not developed
64	Targeted threat index	Level of targeted malware attacks	-	NETWORK ORGANISATION	DYNAMIC	NO	This is usefull for networks or an organisation. It can be computed for a network of devices, but it is not useful for a security evaluation
65	Botnet size	Number of bots, x, that can be instructed to launch attacks (e.g., distributed denial-of-service attacks) at time t, denoted by y(t). Due to time zone difference, y(t) is often much smaller than the actual x as some of x is turned off during night time at time zones	-	NETWORK	STATIC	NO	This is clearly a network-oriented metric
66	Network bandwidth	Network bandwidth that a botnet can use to launch denial-of-service attacks	-	NETWORK	STATIC DYNAMIC	NO	This is clearly a network-oriented metric
67	Botnet efficiency	Network diameter of the botnet network topology. It measures a botnet's capability in communicating command-and-control messages and updating bot programs	-	NETWORK	¿?	NO	This is clearly a network-oriented metric
68	Botnet robustness	Robustness of botnets under random or intelligent disruptions. The idea was derived from the concept of complex network robustness, characterized by the percolation threshold under which a network is disrupted into small components	-	NETWORK	¿?	NO	This is clearly a network-oriented metric
69	Infection rate	Average number of vulnerable computers that are infected by a compromised computer (per time unit) at the early stage of spreading	-	NETWORK	DYNAMIC	NO	This is usefull for networks or an organisation. It can be computed for a network of devices, but it is not useful for a security evaluation
70	Increased false-positive rate	The strength of attacks as a consequence of applying a certain evasion method	-	DEVICE NETWORK	DYNAMIC	NO	This can only be calculated if the embedded system has implemented an evasion mechanism against machine learning attacks
71	Increased false-negative rate	The strength of attacks as a consequence of applying a certain evasion method	-	DEVICE NETWORK	DYNAMIC	NO	This can only be calculated if the embedded system has implemented an evasion mechanism against machine learning attacks
72	Obfuscation prevalence metric	Occurrence of obfuscation in malware samples	-	MALWARE	STATIC	NO	Malware-related
73	Structural complexity metric	Runtime complexity of packers in terms of their layers or granularity	-	MALWARE	¿?	NO	Malware-related
74	Evasion capability of attacks	Allows comparing the evasion power of two attacks, but also allows computing the damage caused by evasion attacks	-	MALWARE	DYNAMIC	NO	Proposed, but not defined
75	Obfuscation sophistication	In terms of the amount of effort required for unpacking packed malware	-	MALWARE	STATIC	NO	Proposed, but not defined. Malware related
76	Network maliciousness metric	It estimates the fraction of blacklisted IP addresses in a network	-	NETWORK	STATIC DYNAMIC	NO	This metric works at network level, for systems in a network. Cannot be applied to embedded systems
77	Rogue networkmetric	Population of networks used to launch drive-by download or phishing attacks	-	NETWORK	DYNAMIC	NO	This metric works at network level, for systems in a network. Phishing cannot be applied to embedded systems
78	ISP badness metric	Quantifies the effect of spam from one ISP or Autonomous System (AS) on the rest of the Internet	-	NETWORK	DYNAMIC	NO	This is clearly a network-oriented metric
79	Control-plane reputation metric	Calibrates the maliciousness of attacker-owned (i.e., rather than legitimate but mismanaged/abused) ASs based on their control plane information (e.g., routing behavior), which can achieve an early-detection time of 50–60 days (before these malicious ASs are noticed by other defense means)	-	ATTACKER	¿?	NO	This is related to external systems that could be potential attackers
80	Cybersecurity posturemetric	Measures the dynamic threat imposed by the attacking computers	-	ATTACKER	DYNAMIC	NO	This is related to attacks themselves
81	Fraction of compromised computers at time t		-	NETWORK	DYNAMIC	NO	Directly related to the systems in a network
82	Probability a computer is compromised at time t	It quantifies the degree of security in enterprise systems by using modeling techniques based on a holistic perspective	-	DEVICE NETWORK	¿?	¿?	Although this is a network-oriented metric, this might be adapted to be used with an embedded system
83	Encounter rate	Measures the fraction of computers that encountered some malware or attack during a period of time	-	NETWORK	STATIC	NO	Directly related to the systems in a network
84	Incident rate	Measures the fraction of computers successfully infected or attacked at least once during a period of time	-	NETWORK	STATIC	NO	Directly related to the systems in a network
85	Blocking rate	The rate an encountered attack is successfully defended by a deployed defense	-	NETWORK	DYNAMIC	¿?	Although this is a network-oriented metric, this might be adapted to be used with an embedded system (maybe for a penetration test)
86	Breach frequencymetric	How often breaches occur	-	NETWORK	STATIC	NO	Directly related to the systems in a network
87	Breach size metric	Number of records breached in individual breaches	-	NETWORK	STATIC	NO	Directly related to the systems in a network
88	Time-between-incidents metric	Period of time between two incidents	-	NETWORK	STATIC	NO	Directly related to the systems in a network
89	Time-to-first-compromise metric	Estimates the duration of time between when a computer starts to run and the first malware alarm is triggered on the computer where the alarm indicates detection rather than infection	-	DEVICE	DYNAMIC	YES	This can be used in embedded systems, but it might not be suitable for a security evaluation (maybe for a penetration test)
90	Delay in incident detection	Time between the occurrence and detection	-	ORGANISATION	DYNAMIC	NO	This metric could be used to transmit the importance of security in the organisation, but cannot be used in embedded systems
91	Cost of incidents	-	-	ORGANISATION	STATIC	NO	Similar to 56
92	Security spending	Percentage of IT budget	-	ORGANISATION	STATIC	NO	This metric shows the investing in security in the organization. It Cannot be applied to embedded systems

93	Security budget allocation	It estimates how the security budget is allocated to various security activities and resources	-	ORGANISATION	STATIC	NO	This metric shows the investing in security in the organization. It Cannot be applied to embedded systems
94	Return on security investment (ROSI)	It is a variant of the classic return on investment (ROI) metric, measuring the financial net gain of an investment based on the gain from investment minus the cost of investment	-	ORGANISATION	STATIC	NO	This metric shows the investing in security in the organization. It Cannot be applied to embedded systems
95	Net present value	It measures the difference between the present economic value of future inflows and the present economic value of outflows with respect to an investment	-	ORGANISATION	STATIC	NO	This metric shows the investing in security in the organization. It Cannot be applied to embedded systems
96	Attack Impact	It is the quantitative measure of the potential harm caused by an attacker to exploit a vulnerability	-	NETWORK	STATIC		
97	Attack Cost	It is the cost spent by an attacker to successfully exploit a vulnerability (i.e., security weakness) on a host	-	NETWORK	STATIC DYNAMIC	YES	This is related to network, but this could be applicable to devices too
98	Structural Important Measured	It it used to qualitatively determine the most critical event (attack, detection or mitigation) in a graphical attack model	-	NETWORK	STATIC	NO	Related to networks
99	Mincut Analysis	??	-	NETWORK			
100	Mean-time-to-compromise (MTTC)	It is used to measure how quickly a network can be penetrated	-	NETWORK	DYNAMIC	¿?	This is no directly applicable, but its equivalent would be a penetration attack in an embedded system
101	Mean-time-to-recovery (MTTR)	It is used to assess the effectiveness of a network to recovery from an attack incidents. It is defined as the average amount of time required to restore a system out of attack state	-	NETWORK	DYNAMIC	¿?	Although this is developed for networks, the same concept can be translated into embeded systems
102	Mean-time-to-First-Failure (MTFF)	??	-	NETWORK			
103	Mean-Time-To-Breach (MTTB)	??	-	NETWORK			
104	Return on Investment	It measures the financial net gain of an investment based on the gain from investment minus the cost of investment	-	NETWORK	STATIC	NO	Network-related and cost related. Useful for organisation managing
105	Return on Attack	The gain the attacker expects from successful attack over the losses he sustains due to the countermeasure deployed by his target	-	NETWORK	¿?	¿?	Network-related, but it scent can be translated into embedded systems
106	Common Vulnerability Scoring System (CVSS) - Overall value	The overall score is determined by generating a base score and modifying it through the temporal and environmental formulas	-	DEVICE	STATIC	YES	This could be used tohether with other metics to obtain a numeric value
107	Probability of Vulnerability Exploited	It is used to assess the likelihood of an attacker exploiting a specific vulnerability on a host	-	SOFTWARE HARDWARE NETWORK	STATIC	¿?	Network-related metric. This is also related to risk assessment. Can be traslated into embedded systems
108	Probability of attack detection	It is used to assess the likelihood of a countermeasure to successfully identify the event of an attack on a target.	-	SOFTWARE HARDWARE NETWORK	DYNAMIC	¿?	Network-related metric. This is also related to risk assessment. Can be traslated into embedded systems
109	Probability of host compromised	It is used to assess the likelihood of an attacker to successfully compromise a target	-	HARDWARE SOFTWARE NETWORK	STATIC DYNAMIC	¿?	Network-related metric. This is also related to risk assessment. Can be traslated into embedded systems
110	Network Compromise Percentage	It quantifies the percentage of hosts on the network on which an attacker can obtain an user or administration level privilege	-	NETWORK	STATIC DYNAMIC	NO	It's about networks, and we can't predict where the device will be used, so this metric is not representative
111	Waekest Adversary	It is used to assess the security strength of a network in terms of the weakest part of the network that an attacker can successfully penetrate	-	SOFTWARE HARDWARE NETWORK	STATIC DYNAMIC	NO	Similar to 31. It's about networks, and we can't predict where the device will be used, so this metric is not representative
112	Vulnerable Host Percentage	This metric quantifies the percentage of hosts with vulnerability on a network. It is used to assess the overall security of a network	-	NETWORK	DYNAMIC	NO	It's about networks, and we can't predict where the device will be used, so this metric is not representative
113	Attack Shortest Path	It is the smallest distance from the attacker to the target. This metric represents the minimum number of hosts an attacker will use to compromise the target host	-	NETWORK	AUTOMATIC* DYNAMIC	NO	If it has to do with paths, it has to do with networks, and we can't assume where is this going to be used
114	Number of Attack Paths	It is the total number of ways an attacker can compromise the target	-	NETWORK	AUTOMATIC* DYNAMIC	NO	If it has to do with paths, it has to do with networks, and we can't assume where is this going to be used
115	Mean of Attack Path Lengths	It is the average of all path lengths. It gives the expected effort that an attacker may use to breach a network policy	-	NETWORK	AUTOMATIC* DYNAMIC	NO	If it has to do with paths, it has to do with networks, and we can't assume where is this going to be used
116	Normalised Mean of Path Lengths	Expected number of exploits an attacker should execute in order to reach the target	-	NETWORK	AUTOMATIC* DYNAMIC	NO	If it has to do with paths, it has to do with networks, and we can't assume where is this going to be used
117	Standand Deviation of Path Lengths	It is used to determine the attack paths of interest	-	NETWORK	AUTOMATIC* DYNAMIC	NO	If it has to do with paths, it has to do with networks, and we can't assume where is this going to be used
118	Mode of Path Lengths	It is the attack path length that occurs most frequently	-	NETWORK	AUTOMATIC* DYNAMIC	NO	If it has to do with paths, it has to do with networks, and we can't assume where is this going to be used
119	Median of Path Lengths	It is used by network administrator to determine how close is an attack path length to the value of the median path length (i.e. path length that is at the middle of all the path length values)	-	NETWORK	AUTOMATIC* DYNAMIC	NO	If it has to do with paths, it has to do with networks, and we can't assume where is this going to be used
120	Attack Resistance Metric	It is use to assess the resistance of a network configuration based on the composition of measures of individual exploits. It is also use for assessing and comparing the security of different network configurations	-	NETWORK	¿?	NO	If it has to do with paths, it has to do with networks, and we can't assume where is this going to be used
121	Mean Time To Failure (MTTF) - Time to compromise	Mean time for a potential intruder to reach the specified target	-	NETWORK	AUTOMATIC DYNAMIC	NO	
122	Mean Effort To Failure (METF) - Effort to compromise	Mean effort for a potential attacker to reach the specified target	-	NETWORK	SEMI-AUTOMATIC DYNAMIC	NO	
123	Mean Time to Security Failure (MTTSF) - Time to compromise	Expected amount of time attackers need to successfully complete an attack process	-	NETWORK	SEMI-AUTOMATIC STATIC	NO	
124	Mean Time to First Failure (MTFF) - Time to compromise	Mean time interval from system initiation to the first system failure	-	NETWORK	SEMI-AUTOMATIC STATIC	¿?	Similar to 102. Depending on the type of test, the time to compromise can be measured (penetration test, fuzzing…), but the result could not be representative

125	Mean Time To Compromise (MTTC) - Time to compromise	Mean time needed for an attacker to gain some level of privilege on some system component	-	NETWORK	SEMI-AUTOMATIC	STATIC	NO	Similar to 100. How can the effort be calculated in an objective manner? This might be the result after some tests
126	Shortest path			NETWORK	AUTOMATIC	DYNAMIC	NO	Similar to 113
127	Number of paths			NETWORK	AUTOMATIC	DYNAMIC	NO	Similar to 114
128	Mean of Path lengths			NETWORK	AUTOMATIC	DYNAMIC	NO	Similar to 115
129	Weakest adversary			NETWORK	SEMI-AUTOMATIC	STATIC	NO	Similar to 31
130	Network compromise percentage			NETWORK	AUTOMATIC	DYNAMIC	NO	If we are evaluating a device, it has no sense talking about networks
131	Compromise probability			NETWORK	SEMI-AUTOMATIC	DYNAMIC	NO	
132	Attack Resistance	It is use to assess the resistance of a network configuration based on the composition of measures of individual exploits. It is also use for assessing and comparing the security of different network configurations	-	NETWORK	SEMI-AUTOMATIC	STATIC	NO	Similar to 120
133	Expected dificulty			NETWORK	AUTOMATIC	STATIC	NO	
134	Percentage of distinct hosts (d1-Diversity)			NETWORK	SEMI-AUTOMATIC	STATIC	NO	This is related with servers in a network, cannot be applied to a single device
135	Number of bits leaked (mean privacy)			NETWORK	SEMI-AUTOMATIC	STATIC	¿?	If adapted, this metric can be used to test memory related vulnerabilities
136	K-zero day safety	Instead of attempting to rank unknown vulnerabilities, the metric counts how many such vulnerabilities would be required for compromising network assets; a larger count implies more security because the likelihood of having more unknown vulnerabilities available, applicable, and exploitable all at the same time will be significantly lower	-	NETWORK	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 32. This has to do with probabilities. Can be applied, but I won't
137	rav	It is a scale measurement of the attack surface, the amount of uncontrolled interactions with a target, which is calculated by the quantitative balance between operations, limitations, and controls	-					
138	30-Day Churn			SOFTWARE				
139	Churn	Count of Source Lines of Code that has been added or changed in a component since the previous revision of the software	-	SOFTWARE	-	-	-	
140	Frequency	Number of times that a binary was edited during its development cycle	-	SOFTWARE	AUTOMATIC*	STATIC*	NO	This is more related to the development of the software
141	Lines Changed	The cumulated number of code lines changed since the creation of a file	-	SOFTWARE	AUTOMATIC*	STATIC*	NO	This is more related to the development of the software
142	Lines new	The cumulated number of new code lines since the creation of a file	-	SOFTWARE	AUTOMATIC*	STATIC*	NO	This is more related to the development of the software
143	Number of lines deleted between revisions			SOFTWARE	AUTOMATIC*		NO	This is more related to the development of the software
144	Number of commits	Counting the commits a given developer made	-	SOFTWARE	AUTOMATIC*	STATIC*	NO	This is more related to the development of the software
145	Percentage of interactive churn (PIC)							
146	Relative churn	Normalized by parameters such as lines of code, files counts	-	SOFTWARE	AUTOMATIC*	STATIC*		
147	Repeat frequency	The number of consecutive edits that are performed on a binary	-	SOFTWARE	AUTOMATIC*	STATIC*		
148	Total churn	The total added, modified, and deleted lines of code of a binary during the development	-	SOFTWARE	AUTOMATIC*	STATIC*		
149	Average service depth (ASD)							
150	Code complexity			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
151	Complexity			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
152	Count path complexity			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
153	Cyclomatic number	M = E - N + P, where E = the number of edges of the graph. N = the number of nodes of the graph. P = the number of connected components.		SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
154	Dependency network complexity							
155	Execution complexity							
156	Fan in (FI)	Number of inputs a function uses. Inputs include parameters and global variables that are used (read) in the function / Number of functions calling a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
157	Fan out (FO)	Number of outputs that are set. The outputs can be parameters or global variables (modified) / Number of functions called by a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
158	Henry Kafura: SLOC *((FI*FO)$\hat{2}$)							
159	Lack of cohesion of methods (LCOM)							
160	Max fan in	The largest number of inputs to a function such as parameters and global variables	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
161	Max fan out	The largest number of assignment to the parameters to call a function or global variables	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
162	MaxMaxNesting	The maximum of MaxNesting in a file	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
163	MaxNesting	Maximum nesting level of control constructs such as if or while statements in a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
164	Nesting complexity	Maximum nesting level of control constructs (if, while, for, switch, etc.) in the function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software

165	Number of children (NOC)	Number of immediate sub-classes of a class or the count of derived classes. If class CA inherits class CB, then CB is the base class and CA is the derived class. In other words, CA is the children of class CB, and CB is the parent of class CB.	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
166	SumCyclomaticStrict	The sum of the strict cyclomatic complexity, where strict cyclomatic complexity is defined as the number of conditional statements in a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
167	SumEssential	The sum of essential complexity, where essential complexity is defined as the number of branches after reducing all the programming primitives such as a for loop in a function's control flow graph into a node iteratively until the graph cannot be reduced any further. Completely well-structured code has essential complexity 1	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
168	SumFanIn	The sum of FanIn, where FanIn is defined as the number of inputs to a function such as parameters and global variables	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
169	SumFanOut	The sum of FanOut, where FanOut is defined as the number of assignment to the parameters to call a function or global variables	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
170	SumMaxNesting	The sum of the MaxNesting in a file, where MaxNesting is defined as the maximum nesting level of control constructs such as if or while statements in a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
171	McCabe Cyclomatic Complexity	M = E - N + 2P, where E = the number of edges of the graph. N = the number of nodes of the graph. P = the number of connected components	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
172	Weighted methods per class (WMC)	Number of local methods defined in the class	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
173	Annual Loss Expectancy (ALE)							
174	Cost of security breach						NO	
175	Remediation Impact (RI)	The remediation impact provides an overview of how much impact the remediation could have on the affected system	-				NO	
176	Risk Potential							
177	Threat-to-impact transitions							
178	Access accuracy	Number of correctly configured user accounts, against the overall number of accounts created, including badly configured accounts and hanging accounts	-	ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	This is not related to embedded systems
179	Administrator and operator logs	Total number corrective actions taken after specific event is recorded in log / total number of events recorded in log * 100	-					
180	Anomalous session count	The first phase derives a SessionTableAccessProfile by correlating application server user log entries for a user session with accessed database tables; the resulting value of SessionTableAccessProfile is represented as a user ID followed by a set of ordered pairs with a table name and a count. The second phase derives the AnomalousSessionCount by counting how many SessionTableAccessCounts don't fit a predefined user profile. If AnamalousSessionCount is greater than one for any user, especially a privileged user, it could indicate the need for significant refactoring and redesign of the Web application's persistence layer. This is a clear case in which detection at design time is preferable.	-	SOFTWARE	-	-	¿?	If adapted, it could be used
181	Approval accuracy	Number of approved provisioning activities, against the overall provisioning activities, including the unauthorized ones	-	ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	NO	This is not related to embedded systems
182	Arc coverage							
183	Audit logging	Total number of log files monitored in specific time interval / number of available log files * 100	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	NO	Although this has to do with larger systems, embedded systems could not be able to produce log files
184	Block coverage							
185	Classified Accessor Attribute Interactions (CAAI)	The ratio of the sum of all interactions between accessors and classified attributes to the possible maximum number of interactions between accessors and classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
186	Classified Attributes Interaction Weight (CAIW)	The ratio of the number of all interactions with classified attributes to the total number of all interactions with all attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
187	Classified Class Data Accessibility (CCDA)	The ratio of the number of non-private classified class attributes to the number of classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
188	Classified Instance Data Accessibility (CIDA)	The ratio of the number of non-private classified instance attributes to the number of classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
189	Classified Method Extensibility (CME)	The ratio of the number of the non-finalised classified methods in program to the total number of classified methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
190	Classified Methods Weight (CMW)	The ratio of the number of classified methods to the total number of methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
191	Classified Mutator Attribute Interactions (CMAI)	The ratio of the sum of all interactions between mutators and classified attributes to the possible maximum number of interactions between mutators and classified attributes in the program.	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
192	Classified Operation Accessibility (COA)	The ratio of the number of non-private classified methods to the number of classified methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
193	Classified Writing Methods Proportion (CWMP)	The ratio of the number of methods which write classified attributes to the total number of classified methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
194	Composite-Part Critical Classes (CPCC)	The ratio of the number of critical composed-part classes to the total number of critical classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
195	Classes Design Proportion (CDP)	Ratio of the number of critical classes to the total number of classes, from the group of Design Size Metrics.	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software

196	Controls against malicious code	Number of incidents as results of malware (malicious software) outbreaks on system / number of detected and blocked malware occurrences * 100	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	NO	This metric could be adapted to embedded systems, but it might not make sense in the testing phase
197	Countermeasure-effectiveness							
198	Coverage							
199	Coverage of Hazard Analysis							
200	Critical Class Extensibility (CCE)	The ratio of the number of the non-finalised critical classes in program to the total number of critical classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
201	Critical Design Proportion (CDP)	The ratio of the number of critical classes to the total number of classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
202	Critical Element Ratio	A process may not require all aspects of an object to be instantiated. Critical Elements Ratio = (Critical Data Elements in an Object) / (Total number of elements in the object)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
203	Critical Serialized Classes Proportion (CSCP)	The ratio of the number of critical serialized classes to the total number of critical classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
204	Critical Superclasses Inheritance (CSI)	The ratio of the sum of classes which may inherit from each critical superclass to the number of possible inheritances from all critical classes in the program's inheritance hierarchy	-	SOFTWARE	-	-	¿?	
205	Critical Superclasses Proportion (CSP)	The ratio of the number of critical superclasses to the total number of critical classes in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
206	Depth of inspection	It calculates the depth of inspection by collecting the statistical data of inspection and testing approaches and finds out the defect capturing capability as a ratio. DI can be measured phase-wise or as a whole before the deployment of the product. # defects captured by inspection process / # defects captured by both inspection and testing approaches	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
207	EIR - ratio between external and internal data flow							
208	Fail-Safe Defaults (PFSD)							
209	Grant Least Privilege (PLP)							
210	Hazard Analysis Achieved							
211	Isolation (PI)	This assesses the level of security isolation between system components. This means getting privileges to a component does not imply accessibility of other co-located components. This metric can be assessed using system architecture and deployment models. Components marked as confidential should not be hosted with nonconfidential (public) components. Methods that are not marked as confidential should not have access to confidential attributes or methods	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	I am not sure if this metric can be applied to embedded software
212	Least Common Mechanism (PLCM)							
213	Number of Catch Blocks Per Class (NCBC)	It measures the exception handing factor (EHF) in some way. It is defined as the percentage of catch blocks in a class over the theoretical maximum number of catch blocks in the class	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
214	Percentage High-, medium- and moderate-Risk Software Hazards with Safety Requirements	It reveals whether all high-, medium- and moderate-risk software hazards have resulted in applicable safety requirements through hazard analysis	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	This is related to software and the design
215	Percentage of Software Safety Requirements (PSSR)	How sufficient hazard analysis has been performed. (# Software Safety Requirements / # SoftwareRequirements) * 100	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	This is related to software and the design
216	Percentage of Software Safety Requirements Traceable to Hazards	Ensuring traceability to system hazards or System of Systems hazards increases the validation case. Percentage indicator of traceability of requirements. All derived software safety requirements must be traceable to system hazards or System of Systems hazards (# Traceable Software Safety Requirements / # Software Safety Requirements) * 100	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	This is related to software and the design
217	Percentage of identified corrective action that has not been implemented			ORGANISATION			¿?	
218	Percentage of incidents that are a recurrence of previous incidents			ORGANISATION	MANUAL*	DYNAMIC*	NO	This a high level metric
219	Percentage of increases in reported security incidents			ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
220	Percentage of IT assets for which fault tolerance mechanisms have been implemented			ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
221	Percentage of IT assets for which recovery procedures have been defined and implemented			ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
222	Percentage of IT assets for which redundancy mechanisms have been implemented			ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
223	Percentage of new components that were deployed in the system all at once			ORGANISATION				
224	Percentage of organization attended security training			ORGANISATION				
225	Percentage of Organization contributing to development			ORGANISATION				
226	Percentage of reported incidents that have been followed up and mitigated			ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
227	Percentage of reported security incidents where the cause of the incident was identified			ORGANISATION				

228	Percentage of security incidents related to incorrect, incomplete, missing or compromised audit data		ORGANISATION	MANUAL*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
229	Percentage of security incidents related to lack of an audit capability		ORGANISATION	MANUAL*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
230	Percentage of security incidents related to the ability to bypass an audit function		ORGANISATION	MANUAL*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
231	Percentage of security incidents that exploited existing vulnerabilities with known solutions		ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
232	Percentage of servers with installed and active automatic hard-disk encryption		NETWORK			NO	Although this cannot be applied directly, this can be something to check for in the checklist
233	Percentage of system architecture for which failure modes have been thoroughly identified		ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	¿?	This is a high level metric, but the concept might be applied to embedded systems
234	Percentage of system changes that were reviewed for security impacts before implementation		ORGANISATION				
235	Percentage of system for which approved configuration settings has been implemented		ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	This is a high level metric. Different configurations of the device can be tested, but this might not be suitable for assessment
236	Percentage of system that depends on external components that the organization lacks control over		ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	¿?	This can be applied to components of thirds parties, as libraries that have not been tested
237	Percentage of system that has been subject to risk analysis		ORGANISATION	AUTOMATIC*	STATIC*	¿?	If the risk analysis can be done just to a percentage of the device, this can be applied. Although it might not make sense to do that
238	Percentage of system that is continuously monitored for configuration policy compliance		ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	This is a high level metric
239	Percentage of system where permissions to install non-standard software is limited or prohibited		ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	Software cannot be installed on a embedded system
240	Percentage of system where the authority to make configuration changes is limited in accordance to policy		ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	This is a high level metric
241	Percentage of systems with the latest approved patches installed		ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	This is related to an organisation security
242	Percentage of sytems exposed at time of malware						
243	PercentValidatedInput	To compute this metric, let T equal the count of the amount of input forms or interfaces the application exposes (the number of HTML form POSTs, GETs, and so on) and let V equal the number of these interfaces that use input validation mechanisms. The ratio V/T makes a strong statement about the Web application's vulnerability to exploits from invalid input	-	SOFTWARE	-	-	YES	This could be adapted to other pieces of code, rather than just HTML
244	Ratio of Design Decisions (RDD)	Ratio of the number of design decisions related to security to the total number of design decisions of the entire system. The objective is to assess the portion of design dedicated to security	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	This is related to design phase
245	Ratio of Design Flaws Related to Security (RDF)	Ratio of design flaws related to security to the total number of design flaws applicable to the whole system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	This is related to design phase
246	Ratio of extension misuse cases extended once to the total number of extension misuse cases	Are the functional decompositions between misuse cases correctly handled? 1 - (# Extension Misuse Cases included once / # Extension Misuse Cases)	-	SOFTWARE	MANUAL*	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
247	Ratio of Implementation Errors That Have an Impact on Security (RSERR)	Ratio of the number of errors related to security to the total number of errors in the implementation of the system (i.e. NERR)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	This is related to design phase
248	Ratio of inclusion misuse cases included once to the total number of inclusion misuse cases	Are the functional decompositions between misuse cases correctly handled? (# Inclusion Misuse Cases included once) / (# Inclusion Misuse Cases)	-	SOFTWARE	MANUAL*	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
249	Ratio of misuse cases used as pre/post conditions of other misuse cases to the total number of misuse cases	Are the functional decompositions between misuse cases correctly handled? (# Misuse Cases used as pre/post conditions) / (# Misuse Cases)	-	SOFTWARE	MANUAL*	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
250	Ratio of Patches Issued to Address Security Vulnerabilities (RP)	Ratio of the number of patches that are issued to address security vulnerabilities to the total number of patches of the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
251	Ratio of Security Requirements (RSR)	Ratio of the number of requirements that have direct impact on security to the total number of requirements of the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
252	Ratio of Security Test Cases (RTC)	Number of test cases designed to detect security issues to the total number of test cases of the entire system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
253	Ratio of Security Test Cases that Fail (RTCP)	Number of test cases that detect implementation errors (i.e. the ones that fail) to the total number of test cases, designed specifically to target security issues	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
254	Ratio of Shared Resources (RSR)							
255	Ratio of Software Changes Due to Security Considerations (RSC)	Number of changes in the system triggered by a new set of security requirements. Software changes due to security considerations include patches that are released after a system is delivered, or any other security updates	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
256	Ratio of the number of included security requirements to the total number of stated security requirements	How vulnerable is the application based on the stated security requirements? 1 - ((# Stated Security Requirements - # Excluded Stated Security Requirements) / # Stated Security Requirements)	-	SOFTWARE	MANUAL*	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
257	Ratio of the number of misuse cases that do not threaten the application to the total number of misuse cases	Do the misuse cases correctly represent the application vulnerabilities and are they consistent with application security use cases? (# Non-Threatening Misuse Cases) / (# Misuse Cases)	-	SOFTWARE	MANUAL*	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
258	Ratio of the Number of Omitted Exceptions (ROEX)	Ratio of the number of omitted exceptions (i.e. Noex) to the total number of exceptions that are related to security	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the software is available, it can be applied. It can be also applied to embedded software

259	Ratio of the Number of Omitted Security Requirements (ROSR)	Ratio of the number of security requirements that have not been considered during the analysis phase (i.e. NOSR) to the total number of security requirements identified during the analysis phase (NSR)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
260	Ratio of the number of the base misuse cases associated to one misuser to the total number of base misuse cases	Are the misusers presented and handled correctly in the misuse case model? 1 - (# Base Misuse Cases Associated to one misuser / # Misuse Cases)	-	SOFTWARE	MANUAL*	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
261	Ratio of the number of unmitigated misuse cases that threaten the application to the total number of misuse cases	Do the misuse cases correctly represent the application vulnerabilities and are they consistent with application security use cases? (# Unmitigated Misuse Cases) / (# Misuse Cases)	-	SOFTWARE	MANUAL*	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
262	Readability of Classified Attributes (RCA)							
263	Readability via Classified Methods (RCM)							
264	Readability via Critical Classes (RCC)							
265	Readability via Critical Classes (RCC)							
266	Software Hazard Analysis Depth	Hazardous software, or safety-critical software, allocated as high- or medium-risk, according to the Software Hazard Criticality Matrix, requires analysis of second- and third-order causal factors (should they exist)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	Hazard is similar to risk.
267	Static analysis alert density			SOFTWARE				
268	Static analysis vulnerability density	Vulnerability metrics are typically normalized by code size to create vulnerability density metrics. Normalizing by size lets us compare software components of varying size. Number of vulnerabilities a static-analysis tool finds per thousand LOC	-					
269	Target Distribution (TD)							
270	Unaccessed Assigned Classified Attribute (UACA)	The ratio of the number of classified attributes that are assigned but never used to the total number of classified attributes in the program	-	SOFTWARE	-	-	-	Object-oriented programs
271	Uncalled Classified Accessor Method (UCAM)	The ratio of the number of classified methods that access a classified attribute but are never called by other methods to the total number of classified methods in the program	-	SOFTWARE	-	-	-	Object-oriented programs
272	Unused Critical Accessor Class (UCAC)	The ratio of the number of classes which contain classified methods that access classified attributes but are never used by other classes to the total number of critical classes in the program	-	SOFTWARE	-	-	-	Object-oriented programs
273	Window of exposure							
274	Writability of Classified Attributes (WCA)							
275	Writability via Classified Methods (WCM)							
276	Writability via Critical Classes (WCC)							
277	Average internal data flow (AIDF)							
278	Average external data flow (EDF)							
279	Average incoming data flow (EDFIN)							
280	Average outgoing data flow (EDFOUT)							
281	Classified Attributes Inheritance (CAI)	The ratio of the number of classified attributes which can be inherited in a hierarchy to the total number of classified attributes in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
282	Classified Methods Inheritance (CMI)	The ratio of the number of classified methods which can be inherited in a hierarchy to the total number of classified methods in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
283	Coupling	Measures the following three coupling dimensions between modules: referential dependency, structural dependency, and data integrity dependency	-					
284	Coupling between components							
285	Coupling Between Object classes (CBOC)	Number of other classes coupled to a class C	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
286	Coupling Between Objects (CBO)							
287	Coupling Corruption Propagation	Coupling corruption propagation is meant to measure the total number of methods that could be affected by erroneous originating method. Coupling Corruption Propagation = Number of child methods invoked with the parameter(s) based on the parameter(s) of the original invocation	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
288	Critical Classes Coupling (CCC)	The ratio of the number of all classes' links with classified attributes to the total number of possible links with classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
289	Depth of Inheritance Tree (DIT)	Maximum depth of the class in the inheritance tree	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
290	Eigenvector Centrality (EvCent)							
291	Flow_ Betweenness							
292	Incoming closure							
293	Incoming data flow of (EDFIN)							
294	Incoming direct							
295	InDegree							
296	InDegree_w							

297	Internal data flow (IDF)							
298	Lack of cohesion of methods (LCOM)	The LCOM value for a class C is defined as $LCOM(C) = \frac{(1-\lvert E(C)\rvert}{\lvert V(C)\rvert\lvert M(C)\rvert} \cdot 100\%$, where V(C) is the set of instance variables, M(C) is the set of instance methods, and E(C) is the set of pairs (v,m) for each instance variable v in V(C) that is used by method m in M(C)	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
299	Layer information							
300	Number of edges between two arbitrary nodes (IE)							
301	NumCalls	The number of calls to the functions defined in an entity		SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
302	OutDegree							
303	OutDegree_w							
304	Outgoing closure							
305	Outgoing data flow of (EDFOUT)							
306	Outgoing direct							
307	Paths							
308	Ratio between average outgoing and incoming data flow (OIR)							
309	Reflection Package Boolean (RPB)	A boolean value representing whether the Java program imports the reflection package (1) or not (0)	-	SOFTWARE	-	-	-	Object-oriented programs
310	RW_ Betweenness							
311	SP_Betweenness							
312	Vulnerability Propagation (VP)	Vulnerability Propagation (VP) of a class C, denoted by VP(C), is the set containing classes in the hierarchy which directly or indirectly inherit class C. Cardinality of set VP(C) represents the number of classes which have become vulnerable due to class C. Alternatively, Vulnerability Propagation of a class C is the total number of classes which directly or indirectly inherit class C.	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	This metric can be applied to embedded software
313	Vulnerability Propagation Factor (VPF)	An Inheritance hierarchy may contain no class or one or more vulnerable classes. Also, there are more than one Inheritance hierarchies present in a design. So, calculation of Vulnerability Propagation Factor (VPF) of a design requires calculation of Vulnerability Propagation due to each vulnerable class in Inheritance hierarchies present in the design. Then, union of Vulnerability Propagation due to each vulnerable class will give overall Vulnerability Propagation in design.	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	This metric can be applied to embedded software
314	Access Complexity (AC)	This metric measures the complexity of attacks exploiting the vulnerability. A vulnerability with low complexity can be for example a buffer overflow in a web server, the vulnerability can be exploited at will						
315	Access Vector (AV)	This metric indicates from where an attacker can exploit the vulnerability						
316	Adversary Work Factor	Adversary work factor is an informative metric for gauging the relative strengths and weaknesses of modem information systems. However, this metric can be difficult to measure and evaluate	-	NETWORK	MANUAL*	DYNAMIC*	NO	This metric has nothing to do with embedded systems and it is directly retaled to networks
317	Attackability	The number of inbound edges of a node in the hostbased attack graph represents all the direct attack paths that other hosts in the considered network can compromise to that node. This number can be used to compute the attackability metric of a host in the context of the system under study. The metric is based on intuitive properties derived from common sense. For example, the metric will indicate reduced confidence when more inbound edges exist. (1 - # inbound edges of the node/ total inbound edges)	-	NETWORK	AUTOMATIC*	STATIC*	NO	This metric has nothing to do with embedded systems and it is directly retaled to networks
318	Authentication (AU)	This metric counts how often an attacker must authenticate before the vulnerability can be exploited						
319	Confidentiality Requirement (CR)							
320	Damage potential-effort ratio	It indicates the level of damage an attacker can potentially cause to the system and the effort required for the attacker to cause such damage	-	DEVICE	SEMI-AUTOMATIC*	DYNAMIC*	YES	It seems that it can be applied to embeded systems, but I am not sure if it is applicable in the evaluation phase
321	Depth							
322	ExclusiveExeTime	Execution time for the set of functions, S, defined in an entity excluding the execution time spent by the functions called by the functions in S						
323	Expected Time to Completion							
324	Exploitability (TE)							
325	InclusiveExeTime	Execution time for the set of functions, S, defined in an entity including all the execution time spent by the functions called directly or indirectly by the functions in S						
326	Integrity Requirement (IR)							
327	Mean Effort to security Failure							
328	Minimal cost of attack	Minimal cost of attack represents the minimal cost that the attacker has to pay for the execution of an attack on a system	-	NETWORK*	-	-	NO	Formal model. Formal verification. Too much notation. I think this is more related to networks
329	Minimal length of attacks	An intuition behind this metric is the following: the less steps an attacker has to make, the simpler is to execute the attack successfully, and the less secure the system is	-	NETWORK*	-	-	NO	Formal model. Formal verification. Too much notation. I think this is more related to networks

330	Protection Rate (PR)	It expresses the efficiency of security mechanisms. Percentage of the known vulnerabilities protected by a given security mechanism. It is based on the Attack Surface metric. PR = (1 - AttackSurfaceEvaluatedSystem / AttackSurfaceReferenceSystem) * 100	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	¿?	This metric was developed specificaly for OSGi platform, but I think it can be adapted to be applied to embedded systems
331	Rigor							
332	Side-channel Vulnerability Factor (SVF)	SVF quantifies patterns in attackers' observations and measures their correlation to the victim's actual execution patterns and in doing so captures systems' vulnerability to side-channel attacks. we can measure information leakage by computing the correlation between ground-truth patterns and attacker observed patterns. We call this correlation Side-channel Vulnerability Factor (SVF). SVF measures the signal-to-noise ratio in an attacker's observations. While any amount of leakage could compromise a system, a low signal-to-noise ratio means that the attacker must either make do with inaccurate results (and thus make many observations to create an accurate result) or become much more intelligent about recovering the original signal. We use phase detection techniques to find patterns in both sets of data then compute the correlation between actual patterns in the victim and observed patterns.	-	HARDWARE	SEMI-AUTOMATIC*	DYNAMIC*	YES	Embedded devices can be vulnerable to side-channel attacks and they can be tested
333	Social Engineering Resistance (SER)	SER = <SERlow, SERhigh>; SERlow = (1 - (P + N - A)/N) * 100; SERhigh = (1 - P/N) * 100; where N is the number of individuals selected or invited to take part in the experiment; A is the number of active participants; P is the total number of valid password/username pairs obtained during the experiment	-	USER	SEMI-AUTOMATIC*	DYNAMIC*	NO	Social engineering cannot be applied to a device
334	Structural severity	Measure that uses software attributes to evaluate the risk of an attacker reaching a vulnerability location from attack surface entry points. It is measured based on three values: high (reachable from an entry point), medium (reachable from an entry point with no dangerous system calls), low (not reachable from any entry points)	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	¿?	I am not sure if this metric can be applied to embedded software
335	Vulnerability exploitability							
336	Weakest adversary	It measures the security strength of a network in terms of the strength of the weakest adversary (i.e., requiring least amount of effort) who can successfully penetrate the network. We define a network configuration that is vulnerable to a weaker set of initial attacker attributes as less secure than a network configuration vulnerable to a stronger set of attributes	-	NETWORK	AUTOMATIC*	DYNAMIC*	NO	Similar to 31. This is for networks, not for embedded systems
337	Vulnerability Index (VI)	Probability of a component's vulnerability being exposed in a single execution	-	DEVICE	MANUAL*	DYNAMIC*	¿?	It seems that it can be applied to embeded systems, but I am not sure if it is applicable in the evaluation phase
338	Effective Vulnerability Index (EVI)	Relative measure of the impact of a component on the system's insecurity	-	DEVICE	SEMI-AUTOMATIC*	DYNAMIC*	¿?	It seems that it can be applied to embeded systems, but I am not sure if it is applicable in the evaluation phase
339	CNBetweenness						NO	
340	CNCloseness						NO	
341	Depth of Master Ownership	This metric determines the level of ownership of the binary depending on the number of edits done. The organization level of the person whose reporting engineers perform more than 75% of the rolled up edits is considered as the DMO. This metric determines the binary owner based on activity on that binary. Our choice of 75% is based on prior historical information on Windows to quantify ownership.					NO	
342	DNAvgBetweenness						NO	
343	DNAvgCloseness						NO	
344	DNAvgEdgeBetweenness						NO	
345	DNMaxCloseness						NO	
346	DNMaxDegree						NO	
347	DNMaxEdgeBetweenness						NO	
348	DNMinBetweenness						NO	
349	DNMinDegree						NO	
350	Edit Frequency	Total number of times the source code that makes up the binary was edited. An edit is when an engineer checks out code from the version control system, alters it, and checks it in again. This is independent of the number of lines of code altered during the edit	-				NO	
351	Level of Organizational Code Ownership	The percent of edits from the organization that contains the binary owner (or if there is no owner the percent of edits from the organization that made the majority of the edits to that binary)	-	ORGANISATION			NO	It is at higher level. It is related to the organisation's security
352	New Effective Author (NEA)						NO	
353	Number of Developers			ORGANISATION			NO	It is at higher level. It is related to the organisation's security
354	Number of Ex-Engineers	Total number of unique engineers who have touched a binary and have left the company as of the release date of the software system	-	ORGANISATION			NO	It is at higher level. It is related to the organisation's security
355	Organization Intersection Factor	The number of different organizations that contribute more than 10% of edits, as measured at the level of the overall org owners	-				NO	
356	Overall Organization Ownership	This is the ratio of the people at the DMO level making edits to a binary relative to total engineers editing the binary	-	ORGANISATION			NO	It is at higher level. It is related to the organisation's security
357	Arithmetic Expression							
358	Array index							

359	Attack Surface Metric	This metric has been proposed by Howard and Manadhata and Wing. Here we consider one of the latest versions of attack surface metric presented by Manadhata et al.	-	DEVICE*	-	-	-	Similar to 8. Formal model. Formal verification. Too much notation. I think this is more related to networks
360	Blank lines			SOFTWARE				
361	Classified Attributes Total (CAT)	The total number of classified attributes in the program	-	SOFTWARE	-	-	-	Object-oriented programs
362	Classified Methods Total (CMT)	The total number of classified methods in the program	-	SOFTWARE	-	-	-	Object-oriented programs
363	Comment Lines			SOFTWARE				
364	Component interface							
365	Control operation							
366	Count of Base Classes (CBC)	Number of base classes	-	SOFTWARE	-	-	-	
367	Critical Classes Total (CCT)	The total number of critical classes in the program	-	SOFTWARE	-	-	-	Object-oriented programs
368	Data format							
369	Economy of Mechanism (PEM)							
370	I/O management							
371	Interface complexity density (I-density)							
372	Kernel control							
373	Lines of Code (LOC)			SOFTWARE				
374	LOCVarDecl			SOFTWARE				
375	Network planning			NETWORK				
376	Number of Attacks	How many attacks on a system exist. The idea behind this metric is that the more attacks on a system exist the less secure the system is. This metric is applied for the simplest analysis of attack graphs. Number of attacks also can be used for the analysis of results of the penetration testing	-	NETWORK*	-	-	NO	Formal model. Formal verification. Too much notation. I think this is more related to networks
377	Number of data items transferred in an edge							
378	Number of Design Decisions Related to Security (NDD)	The proposed metric aims at assessing the number of design decisions that address the security requirements of the system. During the design phase, it is common to end up with multiple solutions to the same problem. Software engineers make many design decisions in order to choose among alternative design solutions.	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
379	Number of Developers			ORGANISATION			NO	It has to do with the organisation and the development of the product. It is applied at the beginning of the life-cycle
380	Number of elicited security use cases *	What is the number of elicited security use cases? # Elicited Security Use Cases	-	SOFTWARE	MANUAL	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
381	Number of excluded security requirements that ensure session handling *	Is session identifier created on server side? Is new session identifier assigned to user on authentication? Is session identifier changed on reauthentication? Is logout option provided for all operations that require authentication? Is session identifier cancelled when authenticated user logs out? Is session identifier killed after a period of time without any actions? Is user's authenticated session identifier protected via secure data transmission protocol?	-	SOFTWARE	MANUAL	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
382	Number of excluded security requirements that put the system at risk of possible attacks *	Summation of the excluded security requirements that put the system at risk	-	SOFTWARE	MANUAL	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
383	Number of global variables *			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
384	Number of identified misuse cases *	What is the number of misuse cases found? # Misuse Cases	-	SOFTWARE	MANUAL	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
385	Number of incoming connections (RIN)							
386	Number of member nodes							
387	Number of outgoing connections from (ROUT)							
388	Number of parameters in the method signature							
389	Number of published vulnerabilities			DEVICE				
390	Number of return points in the method			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
391	Number of Security Algorithms (NSA)	Number of security algorithms that are supported by the application	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied, but this metric can be used in finals stages of the development life-cycle
392	Number of Security Incidents Reported (NSR)	Number of incidents related to security that are reported by the users of the system	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	NO	By definition, the values of this metric are collectec by the user. Son it can't be used in the valuation
393	Number of Security Requirements (NSR) *	Number of security requirements identified during the analysis phase of the application	-	SOFTWARE	MANUAL	STATIC*	¿?	If the design is available, it can be applied
394	Number of sub classes			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
395	NumFunctions			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
396	NumLineProcessor							

397	Paths			SOFTWARE				
398	Reduce Attack Surface (PRAS)							
399	Resource allocation							
400	Response set for a Class (RFC)	Set of methods that can potentially be executed in response to a message received by an object of that class. RFC is simply the number of methods in the set, including inherited methods	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
401	Security Absolute Measurements (SAM)							
402	Stall Ratio	Stall ratio is a measure of how much a program's progress is impeded by frivolous activities. stall ratio = (lines of non-progressive statements in a loop) / (total lines in the loop)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
403	Total Security Index (TSI)							
404	User authority			USER				
405	Volume of email correspondence with vulnerability handling team							
406	Availability Impact (A)							
407	Availability Requirement (AR)							
408	CMMI Level							
409	Comment ratio	Ratio of number of comment lines to number of code lines	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
410	CommentDensity	The ratio of lines of comments to lines of code	-					
411	Compartmentalization	Compartmentalization means that systems and their components run in different compartments, isolated from each other. Thus a compromise of any of them does not impact the others. This metric can be measured as the number of independent components that do not trust each other (performs authentication and authorization for requests/calls coming from other system components) that the system is based on to deliver its function. The higher the compartmentalization value, the more secure the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the software is available, it can be applied. It can be also applied to embedded software
412	Completeness of fix							
413	Confidentiality							
414	Confidentiality Impact (C)							Similar to 22
415	Defense-In-Depth	This metric verifies that security controls are used at different points in the system chain including network security, host security, and application security. Components that have critical data should employ security controls in the network, host, and component layers. To assess this metric we need to capture system architecture and deployment models as well as the security architecture model. Then we can calculate the ratio of components with critical data that apply the layered security principle compared to number of critical components	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	I am not sure if this metric can be applied to embedded software, or at verification phases
416	Expected Reliability							
417	Fail Securely	The system does not disclose any data that should not be disclosed ordinarily at system failure. This includes system data as well as data about the system in case of exceptions. This metric can be evaluated from the security control responses – i.e. how the control behaves in case it failed to operate. From the system architecture perspective, we can assess it as the number of critical attributes and methods that can be accessed in a given component. The smaller the metric value, the likely more secure the system in case of failure	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	This could be used in evaluation
418	Independence of Verification							
419	Inspection Performance Metric (IPM)	It measures the performance of the people during the inspection process using the aforementioned parameters. # defects captured by inspection process / inspection effort	-	TESTER	SEMI-AUTOMATIC*	STATIC*	NO	This metric is related to the tester
420	Integrity							
421	Isolation (PI)	This assesses the level of security isolation between system components. This means getting privileges to a component does not imply accessibility of other co-located components. This metric can be assessed using system architecture and deployment models. Components marked as confidential should not be hosted with nonconfidential (public) components. Methods that are not marked as confidential should not have access to confidential attributes or methods	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	I am not sure if this metric can be applied to embedded software
422	K-zero day safety	Instead of attempting to rank unknown vulnerabilities, the metric counts how many such vulnerabilities would be required for compromising network assets; a larger count implies more security because the likelihood of having more unknown vulnerabilities available, applicable, and exploitable all at the same time will be significantly lower	-	NETWORK	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 32. This has to do with probabilities. Can be applied, but I won't
423	Least Privilege	This metric states that each component and user should be granted the minimal privileges required to complete their tasks. This metric can be assessed from two perspectives: from the security controls perspective we can review users' granted privileges. From the architectural analysis perspective this can be assessed as how the system is broken down to minimal possible actions i.e. the number of components that can access critical data. The smaller the value, the more secure the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the software is available, it can be applied. It can be also applied to embedded software
424	Mean Time To Failure							
425	Report Confidence (RC)							

426	Risk control							
427	Risk identification							
428	Security resource indicator							
429	Service Mechanism Strength							
430	Software project management experience			ORGANISATION			NO	It has to do with the organisation and the development of the product. It is applied at the beginning of the life-cycle
431	Trustworthiness	Trustworthiness of software means its worthy of being trusted to fulfill requirements which may be needed for a particular software component, application, system, or network. It involves attributes of stability, data security, quality, privacy, safety and so on. Software trustworthiness is interrelated with not only risk control in the software process, but also the quality management of the software development process. Furthermore, vision is needed to avoid excessive costs and schedule delays in development and risks management costs in operation; to improve development efforts; and above all	-	SOFTWARE	-	-	-	-
432	Use of (automated) tools							
433	Variable Security Vulnerability	The security vulnerability of a variable v@l is determined by the combined security damage it might cause to the overall system security when v@l is attacked. The more security properties that can be left intact, the less vulnerable the variable is; on the other hand, the more security properties are violated, the more criticall the variable is.	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is only applicable if the source code is available
434	Vulnerability Confidence Coefficient (VCC)							
435	Vulnerability Free Days (VFD)	Percent of days in which the vendor's queue of reported vulnerabilities for the product is empty	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	NO	This metric is not suitable for the evaluation phase
436	Attack Graph Probability							
437	Average Active Vulnerabilities per day (AAV)	Median number of software vulnerabilities which are known to the vendor of a particular piece of software but for which patch has been publicly released by the vendor	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	NO	This metric is not suitable for the evaluation phase
438	Average time from incident detection until incident has been reported			ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	¿?	This a high level metric, but it might be used during a penetration testing if the device has countermeasures implemented
439	BrokenAccountCount	Number of accounts that have no activity for more than 90 days and will never expire. Such accounts represent a clear risk of password compromise and resulting illegal access.	-	SOFTWARE	-	-	¿?	
440	Collateral Damage Potential (CDP)							
441	CVSS Base Score	This group represents the properties of a vulnerability that don't change over time, such as access complexity, access vector, degree to which the vulnerability compromises the confidentiality, integrity, and availability of the system, and requirement for authentication to the system	-	DEVICE				Similar to 15 - 26
442	Developer Risk							
443	Development Risk							
444	Environment Risk							
445	Excess Privilege							
446	Expected threat frequency							
447	Expected Vulnerability							
448	ExploitedFlawCount							
449	Faults found during manual inspection			SOFTWARE			YES	It is related to software and it can be applied to embedded software
450	InjectionFlawCount							
451	Integrity Impact (I)							
452	Maximal probability of successful attack	The probability to accomplish an attack successfully is a well-known metric. The metric describes the most probable way to compromise the system	-	NETWORK*	-	-	NO	Formal model. Formal verification. Too much notation. I think this is more related to networks
453	Mean Failure Cost	Mean Failure Cost MFC) used in the operational sense because the lack of security within the system may cause damage, in terms of lost productivity, lost business, lost data, resulting in security violations. We represent this loss by a random variable, and define MFC as the mean of this random variable. As discussed further, this quantity is not intrinsic to the system, but varies by stakeholder	-	ORGANISATION	-	-	NO	This metric is not related to embeded systems, but to organisations
454	Mean time from vendor patch availability to patch approval to patch installation			ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	NO	This a high level metric, and it is related with a network inside an organisation
455	Mean time to incident detection			ORGANISATION				
456	Monitoring system use	Number of unauthorized attempts to access file, folders, device attachment, attempts to change security settings	-	ORGANISATION	AUTOMATIC*	DYNAMIC*	NO	This can be adapted to be used in embedded systems, but this metric makes no sense in the evaluation phase
457	Non-security failure reports							
458	Number of Design Flaws Related to Security (NSDF)	Security-related design flaws occur when software is planned and specified without proper consideration of security requirements and principles. For instance, clear-text passwords are considered as design flaws. Design flaws can be detected using design inspection techniques (e.g., design reviews). Identifying the number of design flaws related to security can help detect security issues earlier in the design process	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied

459	Number of Exceptions That Have Been Implemented to Handle Execution Failures Related to Security (NEX)	Number of exceptions that have been included in the code to handle possible failures of the system due to an error in a code segment that has an impact on security	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
460	Number of excluded security requirements that ensure input/output handling	Is a specific encoding scheme defined for all inputs? Is a process of canonicalization applied to all inputs? Is an appropriate validation defined and applied to all inputs, in terms of type, length, format/syntax and range? Is a whitelist Filtering approach is applied to all inputs? Are all the validations performed on the client and server side? Is all unsuccessful input handling rejected with an error message? Is all unsuccessful input handling logged? Is output data to the client filtered and encoded? Is output encoding performed on server side?	-	SOFTWARE	MANUAL	STATIC*	NO	This metric is intended to be use in the design phase. This cannot be applied in a security assessment
461	Number of function calls that don't check return values							
462	Number of Implementation Errors Found in the System (NERR)	Number of implementation errors of the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Related to the implementation phase
463	Number of Implementation Errors Related to Security (NSERR)	Number of implementation errors of the system that have a direct impact on security. One of the most common security related implementation errors is the buffer overflow	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Related to the implementation phase
464	Number of Omitted Exceptions for Handling Execution Failures Related to Security (NOEX)	Number of missing exceptions that have been omitted by software engineers when implementing the system. These exceptions can easily be determined through testing techniques.	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Related to the implementation phase
465	Number of Omitted Security Requirements (NOSR)	Number of requirements that should have been considered when building the application	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
466	Number of open security bugs							
467	Number of reported security incidents			ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	NO	This is high level metric
468	Number of security bulletins issues per year			DEVICE				
469	Number of service accounts with weak or default passwords			SOFTWARE				
470	Number of successful attempts to execute recovery this period			ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	NO	This is a high level metric, and it is related with attacks on a organisation's network
471	Number of violations of the LP principle	Number of violations of the Least Privilege principle. A component does not adhere to LP if it, based upon the permissions attributed as described before, is capable of executing tasks it is not responsible for	-	SOFTWARE	AUTOCATIC*	STATIC*	YES	This could be used in embedded systems
472	OverflowVulnCount							
473	Percentage Software Hazards (PSH)	Comparing the number of software hazards identified against historical data, it indicates the sufficiency of the software hazard identification based on the identified hazards. (# software hazards / # System hazards) * 100	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	SÍ	Hazard is similar to risk.
474	Project Management Risk							
475	Remediation Level (RL)						NO	Similar to 24
476	Remediation Potency (RP)	Remediation potency refers to how good the remediation is against the vulnerability	-				NO	
477	Remediation Scheme (RS)	It is used to describe the remediation process	-				NO	
478	Requirements Risk							
479	Security of system documentation	Total number of unauthorized accesses to system documentation / total number of access to system documentation * 100	-	ORGANISATION	SEMI-AUTOMATIC*	DYNAMIC*	NO	This might be useful to measure anything related to social engineering and information gathering and OSINT
480	Static analysis alert count (number of)			SOFTWARE				
481	Temporal Score	Temporal metrics represent the time dependent features of the vulnerabilitie	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	YES	Similar to 23, 24 and 25. This is the final result that is obtained with those partial metrics. CVSS
482	Time to close bug/vulnerability							
483	Time to Problem Correction							
484	Time to Problem Report							
485	User Risk			USER				
486	Vulnerabilities found during requirements, design and coding			DEVICE				
487	Vulnerabilities found post-development			DEVICE				
488	Vulnerability Density	Number of vulnerabilities in the unit size of a code. VD = size of the software / # vulnerabilities in the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	It seems that it can be applied to embeded systems, but I am not sure if it is applicable in the evaluation phase
489	Weakness							
490	XsiteVulnCount	It is obtained via a penetration-testing tools	-	SOFTWARE	-	-	NO	It is not clearly defined
491	Operational Environment Security Measurement							
492	Program Implementation Security Measurement							
493	Security Indictor of Software Systems (sic)							
494	Security metrics of Arithmetic Expression	The security flaws of arithmetic expression include divide by zero, overflow of arithmetic operation, misused data type of arithmetic expression, and truncation error of arithmetic operation	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
495	Security Metrics of Array Index	The security flaws of array index include index out of range, incorrect index variable and uncontrollable array index	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist

496	Security Metrics of Component Interfaces	The security flaws of component interface include inconsistent parameter numbers, inconsistent data types, and inconsistent size of data types	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
497	Security Metrics of Control Operation	The security flaws of control operations include infinite loop, deadlock situations and boundary defects	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
498	Security Metrics of I/O Management	File is an important media which can store critical information and record log data. However, file privilege definition, file access management, file contents, file organization and file size are major factors to affect the security software operation	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
499	Security Metrics of Input Format	The security flaws of data format include mismatched data contents, mismatched data type, mismatch data volume	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
500	Security Metrics of Kernel Operation	A released software system must be adapted to a suitable operating system and environment. The kernel of operating system and environment become a critical factor to affect the operating security of software system. Resource management, execution file protection, main memory control and process scheduling are major tasks of operating system. The security flaws, which embedded in the tasks of operating system, become high security risk for software system operation	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Checklist. Only when there is an OS
501	Security Metrics of Network Environment	Network environment is a critical tool in the e-commerce years. However, data transmission management, remote access control, and transmission contents always become the security holes of software system operation	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified
502	Security Metrics of Resources Allocation	The security flaws of resource allocation include exhausted memory, unreleased memory exhausted resources, and unreleased resources	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified
503	Security Metrics of User Authority	Many users may use the released system to do specific jobs. However, user privilege definition, user password management and user detailed information maintenance are important tasks to control and manage the user authority. Without user authority control and management, the operation environment of software system will be on high security risk	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified
504	Instruction Count							
505	Attack-reward (URL Jumping)							
506	Classified Accessor Attribute Interactions (CAAI)	The ratio of the sum of all interactions between accessors and classified attributes to the possible maximum number of interactions between accessors and classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 187. Object-oriented programs
507	Classified Attributes Inheritance (CAI)	The ratio of the number of classified attributes which can be inherited in a hierarchy to the total number of classified attributes in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 284. Object-oriented programs
508	Classified Attributes Interaction Weight (CAIW)	The ratio of the number of all interactions with classified attributes to the total number of all interactions with all attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 188. Object-oriented programs
509	Classified Class Data Accessibility (CCDA)	The ratio of the number of non-private classified class attributes to the number of classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 189. Object-oriented programs
510	Classified Instance Data Accessibility (CIDA)	The ratio of the number of non-private classified instance attributes to the number of classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 190. Object-oriented programs
511	Classified Method Extensibility (CME)	The ratio of the number of the non-finalised classified methods in program to the total number of classified methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 191. Object-oriented programs
512	Classified Methods Inheritance (CMI)	The ratio of the number of classified methods which can be inherited in a hierarchy to the total number of classified methods in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 258. Object-oriented programs
513	Classified Methods Weight (CMW)	The ratio of the number of classified methods to the total number of methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 192. Object-oriented programs
514	Classified Mutator Attribute Interactions (CMAI)	The ratio of the sum of all interactions between mutators and classified attributes to the possible maximum number of interactions between mutators and classified attributes in the program.	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 193. Object-oriented programs
515	Classified Operation Accessibility (COA)	The ratio of the number of non-private classified methods to the number of classified methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 194. Object-oriented programs
516	CNBetweenness							
517	Composite-Part Critical Classes (CPCC)	The ratio of the number of critical composed-part classes to the total number of critical classes in the program	-	SOFTWARE	-	-	-	Similar to 196
518	Confidentiality							Similar to 413
519	Critical Class Extensibility (CCE)	The ratio of the number of the non-finalised critical classes in program to the total number of critical classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 201. Object-oriented programs
520	Critical Design Proportion (CDP)	The ratio of the number of critical classes to the total number of classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 202. Object-oriented programs
521	Critical Superclasses Inheritance (CSI)	The ratio of the sum of classes which may inherit from each critical superclass to the number of possible inheritances from all critical classes in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 205. Object-oriented programs
522	Critical Superclasses Proportion (CSP)	The ratio of the number of critical superclasses to the total number of critical classes in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	NO	Similar to 206. Object-oriented programs
523	Design Size			SOFTWARE HARDWARE				
524	DNMaxEdgeBetweenness							
525	Knot Count							
526	Kolmogorov Complexity			SOFTWARE				

527	Measurement of Cost			
528	Number of Developers	ORGANISATION	NO	It is no related directly with the device
529	Potency			
530	Resilience			
531	Shortest Path			

A.2 Final Set of Metrics

#	METRIC	DEFINITION	SCALE	SCOPE	AUTOMATION	MEASUREMENT	USABLE	REASON
5	Entropy			USER	AUTOMATIC	STATIC	YES	This could be used with keys instead of passwords to measure randomness
8	Attack surface	It aim to measure the ways by which an attacker can compromise a targeted software	ABSOLUTE	DEVICE		STATIC	YES	
9	Historical vulnerability metric	It measures the degree that a system is vulnerable (i.e., frequency of vulnerabilities) in the past	RATIO ABSOLUTE DISTRIBUTION	DEVICE		STATIC	YES	This could be part of a checklist of task, to check if there is any vulnerability that affects the device
10	Historically exploited vulnerability metric	It measures the number of vulnerabilities exploited in the past	RATIO ABSOLUTE DISTRIBUTION	DEVICE		STATIC	YES	This could be part of a checklist of task, to check if there is any vulnerability that affects the device
13	Tendency-to-be-exploited metric	It measures the tendency that a vulnerability may be exploited, where the "tendency" may be derived from information sources such as posts on Twitter before vulnerability disclosures	RATIO ABSOLUTE DISTRIBUTION	DEVICE		STATIC	¿?	This research work might not be part of a security evaluation, but could be applied if so
15	Attack vector	Describes whether a vulnerability can be exploited remotely, adjacently, locally, or physically (i.e., attacker must have physical access to the computer)	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
16	Attack complexity	Describes the conditions that must hold before an attacker can exploit the vulnerability, such as low or high	ORDINA	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
17	Privilege required	Describes the level of privileges that an attacker must have in order to exploit a vulnerability, such as none, low, or high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
18	User interaction	Describes whether the exploitation of a vulnerability requires the participation of a user (other than the attacker), such as none or required	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
19	Authorization scope	Describes whether or not a vulnerability has an impact on resources beyond its privileges (e.g., sandbox or virtual machine), such as unchanged or changed	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
20	Confidentiality impact	The impact of a successful exploitation of a vulnerability in terms of confidentiality such as none, low, high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
21	Integrity impact	The impact of a successful exploitation of a vulnerability in terms of integrity such as none, low, high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
22	Availability impact	The impact of a successful exploitation of a vulnerability in terms of availability, such as none, low, high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
23	Exploit code maturity	The likelihood a vulnerability can be attacked based on the current exploitation techniques, such as undefined, unproven, proof-of-concept, functional, or high	ORDINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
24	Remediation level	Describes whether or not a remediation method is available for a given vulnerability, such as undefined, unavailable, workaround, temporary fix, or official fix	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
25	Report confidence	Measures the level of confidence for a given vulnerability as well as known technical details, such as unknown, reasonable, or confirmed	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
26	Security requirement	Describes environment-dependent security requirements in terms of confidentiality, integrity, and availability, such as not defined, low, medium, or high	NOMINAL	SOFTWARE		STATIC	YES	For calculating new vulnerabilities, or to know each dimension of known vulnerabilities
29	Necessary defense metric	It estimates a minimal set of defense countermeasures necessary for thwarting a certain attack	-	DEVICE		STATIC	YES	This is related with the defenses that are necesary to implement so the system is secure for a certain vulnerability
30	Effort-to-security-failure metric	It measures an attacker's effort to reach its goal state	-	DEVICE		DYNAMIC*	YES	Although it might be difficult to estimate, this could be also included in the evaluation process
31	Weakest adversary metric	It estimates minimum adversary capabilities required to achieve an attack goal	-	DEVICE		DYNAMIC*	¿?	If the effort can be calculated in an objective manner, this could be used in embedded systems
33	Likelihood of exploitation	It measures the probability that an exploit will be executed by an attacker with certain capabilities	-	SOFTWARE		STATIC	¿?	This is related with statistics and risk assessment. It could be applied
34	Effective base metric / scope	Given an attack graph and a subset of exploits, the effective base metric aims to adjust the CVSS base metric by taking the highest value of the ancestors of an exploit to reflect the worst-case scenario, while the effective base score metric is calculated based on the effective base metric	-	SOFTWARE		STATIC	-	-
35	Reaction time metric	It captures the delay between the observation of the malicious entity at time t and the blacklisting of the malicious entity at time t (i.e., t - t)	ABSOLUTE	DEVICE		DYNAMIC	YES	If the device has any kind of detection mechanism, this could be used. Detection and reaction might have different times
36	Coverage metric	It estimates the portion of blacklisted malicious players	ORDINAL	DEVICE		STATIC	YES	This could be used, but the device has to have a blacklist
38	Average indirect target reduction	It counts the overall reduction of targets for any indirect control-flow transfer. It measures the overall reduction in terms of the number of targets exploitable by the attacker where smaller targets are more secure	ABSOLUTE	UNKNOWN		UNKNOWN	UNKNOWN	UNKNOWN
39	Average target size	Ratio between the size of the largest target and the number of targets	-	UNKNOWN		UNKNOWN	UNKNOWN	UNKNOWN
40	Evasion resistance	It is measured against control flow bending attacks, reflecting the effort (or premises) that an attacker must make (or satisfy) for evading the CFI scheme	-	UNKNOWN		UNKNOWN	UNKNOWN	UNKNOWN
41	Coverage metric	It measures the fraction of events detectable by a specific sensor deployment, reflecting a defender's need in monitoring events	-	DEVICE		DYNAMIC	UNKNOWN	UNKNOWN
42	Redundancy metric	It estimates the amount of evidence provided by a specific sensor deployment to detect an event	-	DEVICE		DYNAMIC	YES	This can be applied to devices as a whole, to check any mechanism of intrusion detection (mostly, tanti-tampering and software)

43	Confidence metric	It measures how well-deployed sensors detect an event in the presence of compromised sensors	-	DEVICE	DYNAMIC	YES	This can be applied to devices as a whole, to check any mechanism of intrusion detection (mostly, tanti-tampering and software)
45	Detection time	For instrument-based attack detection, this metric is used to measure the delay between the time t0 at which a compromised computer sends its first scan packet and the time t that a scan packet is observed by the instrument	-	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
57	ASLR-induced Effective entropy metric	Measure of entropy in user space memory which quantitatively considers an adversary's ability to leverage low entropy regions of memory via absolute and dynamic intersection connections	-	SOFTWARE	DYNAMIC	YES	If the device runs an OS, this could be applied
59	Penetration resistance (PR)	Running a penetration test to estimate the level of effort (e.g., person-day or cost) required for a red team to penetrate into a system	-	DEVICE NETWORK	DYNAMIC	YES	Penetration testing is a really common way to test this kind of device
82	Probability a computer is compromised at time t	It quantifies the degree of security in enterprise systems by using modeling techniques based on a holistic perspective	-	DEVICE NETWORK	¿?	¿?	Although this is a network-oriented metric, this might be adapted to be used with an embedded system
85	Blocking rate	The rate an encountered attack is successfully defended by a deployed defense	-	NETWORK	DYNAMIC	¿?	Although this is a network-oriented metric, this might be adapted to be used with an embedded system (maybe for a penetration test)
89	Time-to-first-compromise metric	Estimates the duration of time between when a computer starts to run and the first malware alarm is triggered on the computer where the alarm indicates detection rather than infection	-	DEVICE	DYNAMIC	YES	This can be used in embedded systems, but it might not be suitable for a security evaluation (maybe for a penetration test)
96	Attack Impact	It is the quantitative measure of the potential harm caused by an attacker to exploit a vulnerability	-	NETWORK	STATIC		
97	Attack Cost	It is the cost spent by an attacker to successfully exploit a vulnerability (i.e., security weakness) on a host	-	NETWORK	STATIC DYNAMIC	YES	This is related to network, but this could be applicable to devices too
100	Mean-time-to-compromise (MTTC)	It is used to measure how quickly a network can be penetrated	-	NETWORK	DYNAMIC	¿?	This is no directly applicable, but its equivalent would be a penetration attack in an embedded system
101	Mean-time-to-recovery (MTTR)	It is used to assess the effectiveness of a network to recovery from an attack incidents. It is defined as the average amount of time required to restore a system out of attack state	-	NETWORK	DYNAMIC	¿?	Although this is developed for networks, the same concept can be translated into embeded systems
105	Return on Attack	The gain the attacker expects from successful attack over the losses he sustains due to the countermeasure deployed by his target	-	NETWORK	¿?	¿?	Network-related, but it scent can be translated into embedded systems
106	Common Vulnerability Scoring System (CVSS) - Overall value	The overall score is determined by generating a base score and modifying it through the temporal and environmental formulas	-	DEVICE	STATIC	YES	This could be used tohether with other metics to obtain a numeric value
107	Probability of Vulnerability Exploited	It is used to assess the likelihood of an attacker exploiting a specific vulnerability on a host	-	DEVICE NETWORK	STATIC	¿?	Network-related metric. This is also related to risk assessment. Can be traslated into embedded systems
108	Probability of attack detection	It is used to assess the likelihood of a countermeasure to successfully identify the event of an attack on a target	-	DEVICE NETWORK	DYNAMIC	¿?	Network-related metric. This is also related to risk assessment. Can be traslated into embedded systems
109	Probability of host compromised	It is used to assess the likelihood of an attacker to successfully compromise a target	-	DEVICE NETWORK	STATIC DYNAMIC	¿?	Network-related metric. This is also related to risk assessment. Can be traslated into embedded systems
137	Number of bits leaked (mean privacy)			NETWORK	SEMI-AUTOMATIC STATIC	¿?	If adapted, this metric can be used to test memory related vulnerabilities
139	rav	It is a scale measurement of the attack surface, the amount of uncontrolled interactions with a target, which is calculated by the quantitative balance between operations, limitations, and controls	-	DEVICE			
141	Churn	Count of Source Lines of Code that has been added or changed in a component since the previous revision of the software	-	SOFTWARE	-	-	-
148	Relative churn	Normalized by parameters such as lines of code, files counts	-	SOFTWARE	AUTOMATIC*	STATIC*	
149	Repeat frequency	The number of consecutive edits that are performed on a binary	-	SOFTWARE	AUTOMATIC*	STATIC*	
150	Total churn	The total added, modified, and deleted lines of code of a binary during the development	-	SOFTWARE	AUTOMATIC*	STATIC*	
155	Cyclomatic number	M = E - N + P, where E = the number of edges of the graph. N = the number of nodes of the graph. P = the number of connected components.		SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
158	Fan in (FI)	Number of inputs a function uses. Inputs include parameters and global variables that are used (read) in the function / Number of functions calling a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
159	Fan out (FO)	Number of outputs that are set. The outputs can be parameters or global variables (modified) / Number of functions called by a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
162	Max fan in	The largest number of inputs to a function such as parameters and global variables	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
163	Max fan out	The largest number of assignment to the parameters to call a function or global variables	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
164	MaxMaxNesting	The maximum of MaxNesting in a file	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
165	MaxNesting	Maximum nesting level of control constructs such as if or while statements in a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
166	Nesting complexity	Maximum nesting level of control constructs (if, while, for, switch, etc.) in the function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
167	Number of children (NOC)	Number of immediate sub-classes of a class or the count of derived classes. If class CA inherits class CB, then CB is the base class and CA is the derived class. In other words, CA is the children of class CB, and CB is the parent of class CB.	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
168	SumCyclomaticStrict	The sum of the strict cyclomatic complexity, where strict cyclomatic complexity is defined as the number of conditional statements in a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software

169	SumEssential	The sum of essential complexity, where essential complexity is defined as the number of branches after reducing all the programming primitives such as a for loop in a function's control flow graph into a node iteratively until the graph cannot be reduced any further. Completely well-structured code has essential complexity 1	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
170	SumFanIn	The sum of FanIn, where FanIn is defined as the number of inputs to a function such as parameters and global variables	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
171	SumFanOut	The sum of FanOut, where FanOut is defined as the number of assignment to the parameters to call a function or global variables	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
172	SumMaxNesting	The sum of the MaxNesting in a file, where MaxNesting is defined as the maximum nesting level of control constructs such as if or while statements in a function	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
173	McCabe Cyclomatic Complexity	M = E - N + 2P, where E = the number of edges of the graph. N = the number of nodes of the graph. P = the number of connected components	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
174	Weighted methods per class (WMC)	Number of local methods defined in the class	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
187	Classified Accessor Attribute Interactions (CAAI)	The ratio of the sum of all interactions between accessors and classified attributes to the possible maximum number of interactions between accessors and classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
188	Classified Attributes Interaction Weight (CAIW)	The ratio of the number of all interactions with classified attributes to the total number of all interactions with all attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
189	Classified Class Data Accessibility (CCDA)	The ratio of the number of non-private classified class attributes to the number of classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
190	Classified Instance Data Accessibility (CIDA)	The ratio of the number of non-private classified instance attributes to the number of classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
191	Classified Method Extensibility (CME)	The ratio of the number of the non-finalised classified methods in program to the total number of classified methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
192	Classified Methods Weight (CMW)	The ratio of the number of classified methods to the total number of methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
193	Classified Mutator Attribute Interactions (CMAI)	The ratio of the sum of all interactions between mutators and classified attributes to the possible maximum number of interactions between mutators and classified attributes in the program.	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
194	Classified Operation Accessibility (COA)	The ratio of the number of non-private classified methods to the number of classified methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
195	Classified Writing Methods Proportion (CWMP)	The ratio of the number of methods which write classified attributes to the total number of classified methods in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
196	Composite-Part Critical Classes (CPCC)	The ratio of the number of critical composed-part classes to the total number of critical classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
	Classes Design Proportion (CDP)	Ratio of the number of critical classes to the total number of classes, from the group of Design Size Metrics.	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
201	Critical Class Extensibility (CCE)	The ratio of the number of the non-finalised critical classes in program to the total number of critical classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
202	Critical Design Proportion (CDP)	The ratio of the number of critical classes to the total number of classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
203	Critical Element Ratio	A process may not require all aspects of an object to be instantiated. Critical Elements Ratio = (Critical Data Elements in an Object) / (Total number of elements in the object)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
204	Critical Serialized Classes Proportion (CSCP)	The ratio of the number of critical serialized classes to the total number of critical classes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
205	Critical Superclasses Inheritance (CSI)	The ratio of the sum of classes which may inherit from each critical superclass to the number of possible inheritances from all critical classes in the program's inheritance hierarchy	-	SOFTWARE	-	-	¿?	
206	Critical Superclasses Proportion (CSP)	The ratio of the number of critical superclasses to the total number of critical classes in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
212	Isolation (PI)	This assesses the level of security isolation between system components. This means getting privileges to a component does not imply accessibility of other co-located components. This metric can be assessed using system architecture and deployment models. Components marked as confidential should not be hosted with nonconfidential (public) components. Methods that are not marked as confidential should not have access to confidential attributes or methods	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	I am not sure if this metric can be applied to embedded software
214	Number of Catch Blocks Per Class (NCBC)	It measures the exception handing factor (EHF) in some way. It is defined as the percentage of catch blocks in a class over the theoretical maximum number of catch blocks in the class	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software

215	Percentage High-, medium- and moderate-Risk Software Hazards with Safety Requirements	It reveals whether all high-, medium- and moderate-risk software hazards have resulted in applicable safety requirements through hazard analysis	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	This is related to software and the design
218	Percentage of Software Safety Requirements (PSSR)	How sufficient hazard analysis has been performed. (# Software Safety Requirements / # SoftwareRequirements) * 100	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	This is related to software and the design
219	Percentage of Software Safety Requirements Traceable to Hazards	Ensuring traceability to system hazards or System of Systems hazards increases the validation case. Percentage indicator of traceability of requirements. All derived software safety requirements must be traceable to system hazards or System of Systems hazards (# Traceable Software Safety Requirements / # Software Safety Requirements) * 100	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	This is related to software and the design
246	PercentValidatedInput	To compute this metric, let T equal the count of the amount of input forms or interfaces the application exposes (the number of HTML form POSTs, GETs, and so on) and let V equal the number of these interfaces that use input validation mechanisms. The ratio V/T makes a strong statement about the Web application's vulnerability to exploits from invalid input	-	SOFTWARE	-	-	YES	This could be adapted to other pieces of code, rather than just HTML
247	Ratio of Design Decisions (RDD)	Ratio of the number of design decisions related to security to the total number of design decisions of the entire system. The objective is to assess the portion of design dedicated to security	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	This is related to design phase
248	Ratio of Design Flaws Related to Security (RDF)	Ratio of design flaws related to security to the total number of design flaws applicable to the whole system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	This is related to design phase
250	Ratio of Implementation Errors That Have an Impact on Security (RSERR)	Ratio of the number of errors related to security to the total number of errors in the implementation of the system (i.e. NERR)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	This is related to design phase
253	Ratio of Patches Issued to Address Security Vulnerabilities (RP)	Ratio of the number of patches that are issued to address security vulnerabilities to the total number of patches of the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
254	Ratio of Security Requirements (RSR)	Ratio of the number of requirements that have direct impact on security to the total number of requirements of the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
255	Ratio of Security Test Cases (RTC)	Number of test cases designed to detect security issues to the total number of test cases of the entire system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
256	Ratio of Security Test Cases that Fail (RTCP)	Number of test cases that detect implementation errors (i.e. the ones that fail) to the total number of test cases, designed specifically to target security issues	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
258	Ratio of Software Changes Due to Security Considerations (RSC)	Number of changes in the system triggered by a new set of security requirements. Software changes due to security considerations include patches that are released after a system is delivered, or any other security updates	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
261	Ratio of the Number of Omitted Exceptions (ROEX)	Ratio of the number of omitted exceptions (i.e. Noex) to the total number of exceptions that are related to security	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the software is available, it can be applied. It can be also applied to embedded software
262	Ratio of the Number of Omitted Security Requirements (ROSR)	Ratio of the number of security requirements that have not been considered during the analysis phase (i.e. NOSR) to the total number of security requirements identified during the analysis phase (NSR)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
269	Software Hazard Analysis Depth	Hazardous software, or safety-critical software, allocated as high- or medium-risk, according to the Software Hazard Criticality Matrix, requires analysis of second- and third-order causal factors (should they exist)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	Hazard is similar to risk.
273	Unaccessed Assigned Classified Attribute (UACA)	The ratio of the number of classified attributes that are assigned but never used to the total number of classified attributes in the program	-	SOFTWARE	-	-	-	Object-oriented programs
274	Uncalled Classified Accessor Method (UCAM)	The ratio of the number of classified methods that access a classified attribute but are never called by other methods to the total number of classified methods in the program	-	SOFTWARE	-	-	-	Object-oriented programs
275	Unused Critical Accessor Class (UCAC)	The ratio of the number of classes which contain classified methods that access classified attributes but are never used by other classes to the total number of critical classes in the program	-	SOFTWARE	-	-	-	Object-oriented programs
284	Classified Attributes Inheritance (CAI)	The ratio of the number of classified attributes which can be inherited in a hierarchy to the total number of classified attributes in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
285	Classified Methods Inheritance (CMI)	The ratio of the number of classified methods which can be inherited in a hierarchy to the total number of classified methods in the program's inheritance hierarchy	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
286	Coupling	Measures the following three coupling dimensions between modules: referential dependency, structural dependency, and data integrity dependency	-	SOFTWARE				
288	Coupling Between Object classes (CBOC)	Number of other classes coupled to a class C	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
290	Coupling Corruption Propagation	Coupling corruption propagation is meant to measure the total number of methods that could be affected by erroneous originating method. Coupling Corruption Propagation = Number of child methods invoked with the parameter(s) based on the parameter(s) of the original invocation	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
291	Critical Classes Coupling (CCC)	The ratio of the number of all classes' links with classified attributes to the total number of possible links with classified attributes in the program	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Object-oriented programs. If the software is available, it can be applied. It can be also applied to embedded software
292	Depth of Inheritance Tree (DIT)	Maximum depth of the class in the inheritance tree	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
301	Lack of cohesion of methods (LCOM)	The LCOM value for a class C is defined as $LCOM(C) = \frac{(1-\lvert E(C)\rvert}{\lvert V(C)\rvert\lvert M(C)\rvert} \cdot 100\%$, where V(C) is the set of instance variables, M(C) is the set of instance methods, and E(C) is the set of pairs (v,m) for each instance variable v in V(C) that is used by method m in M(C)	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
304	NumCalls	The number of calls to the functions defined in an entity	-	SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software

312	Reflection Package Boolean (RPB)	A boolean value representing whether the Java program imports the reflection package (1) or not (0)	-	SOFTWARE	-	-	-	Object-oriented programs
	Vulnerability Propagation (VP)	Vulnerability Propagation (VP) of a class C, denoted by VP(C), is the set containing classes in the hierarchy which directly or indirectly inherit class C. Cardinality of set VP(C) represents the number of classes which have become vulnerable due to class C. Alternatively, Vulnerability Propagation of a class C is the total number of classes which directly or indirectly inherit class C.	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	This metric can be applied to embedded software
315	Vulnerability Propagation Factor (VPF)	An Inheritance hierarchy may contain no class or one or more vulnerable classes. Also, there are more than one Inheritance hierarchies present in a design. So, calculation of Vulnerability Propagation Factor (VPF) of a design requires calculation of Vulnerability Propagation due to each vulnerable class in Inheritance hierarchies present in the design. Then, union of Vulnerability Propagation due to each vulnerable class will give overall Vulnerability Propagation in design.	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	This metric can be applied to embedded software
316	Access Complexity (AC)	This metric measures the complexity of attacks exploiting the vulnerability. A vulnerability with low complexity can be for example a buffer overflow in a web server, the vulnerability can be exploited at will	-	DEVICE				
317	Access Vector (AV)	This metric indicates from where an attacker can exploit the vulnerability	-	DEVICE				
320	Authentication (AU)	This metric counts how often an attacker must authenticate before the vulnerability can be exploited	-	SOFTWARE				
322	Damage potential-effort ratio	It indicates the level of damage an attacker can potentially cause to the system and the effort required for the attacker to cause such damage	-	DEVICE	SEMI-AUTOMATIC*	DYNAMIC*	YES	It seems that it can be applied to embeded systems, but I am not sure if it is applicable in the evaluation phase
332	Protection Rate (PR)	It expresses the efficiency of security mechanisms. Percentage of the known vulnerabilities protected by a given security mechanism. It is based on the Attack Surface metric. PR = (1 - AttackSurfaceEvaluatedSystem / AttackSurfaceReferenceSystem) * 100	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	¿?	This metric was developed specificaly for OSGi platform, but I think it can be adapted to be applied to embedded systems
334	Side-channel Vulnerability Factor (SVF)	SVF quantifies patterns in attackers' observations and measures their correlation to the victim's actual execution patterns and in doing so captures systems' vulnerability to side-channel attacks. we can measure information leakage by computing the correlation between ground-truth patterns and attacker observed patterns. We call this correlation Side-channel Vulnerability Factor (SVF). SVF measures the signal-to-noise ratio in an attacker's observations. While any amount of leakage could compromise a system, a low signal-to-noise ratio means that the attacker must either make do with inaccurate results (and thus make many observations to create an accurate result) or become much more intelligent about recovering the original signal. We use phase detection techniques to find patterns in both sets of data then compute the correlation between actual patterns in the victim and observed patterns.	-	HARDWARE	SEMI-AUTOMATIC*	DYNAMIC*	YES	Embedded devices can be vulnerable to side-channel attacks and they can be tested
336	Structural severity	Measure that uses software attributes to evaluate the risk of an attacker reaching a vulnerability location from attack surface entry points. It is measured based on three values: high (reachable from an entry point), medium (reachable from an entry point with no dangerous system calls), low (not reachable from any entry points)	-	SOFTWARE	SEMI-AUTOMATIC*	DYNAMIC*	¿?	I am not sure if this metric can be applied to embeded software
	Vulnerability Index (VI)	Probability of a component's vulnerability being exposed in a single execution	-	DEVICE	MANUAL*	DYNAMIC*	¿?	It seems that it can be applied to embeded systems, but I am not sure if it is applicable in the evaluation phase
	Effective Vulnerability Index (EVI)	Relative measure of the impact of a component on the system's insecurity	-	DEVICE	SEMI-AUTOMATIC*	DYNAMIC*	¿?	It seems that it can be applied to embedded systems, but I am not sure if it is applicable in the evaluation phase
361	Classified Attributes Total (CAT)	The total number of classified attributes in the program	-	SOFTWARE	-	-	-	Object-oriented programs
362	Classified Methods Total (CMT)	The total number of classified methods in the program	-	SOFTWARE	-	-	-	Object-oriented programs
366	Count of Base Classes (CBC)	Number of base classes	-	SOFTWARE	-	-	-	
367	Critical Classes Total (CCT)	The total number of critical classes in the program	-	SOFTWARE	-	-	-	Object-oriented programs
378	Number of Design Decisions Related to Security (NDD)	The proposed metric aims at assessing the number of design decisions that address the security requirements of the system. During the design phase, it is common to end up with multiple solutions to the same problem. Software engineers make many design decisions in order to choose among alternative design solutions.	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
383	Number of global variables *			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
390	Number of return points in the method			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
391	Number of Security Algorithms (NSA)	Number of security algorithms that are supported by the application	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied, but this metric can be used in finals stages of the development life-cycle
393	Number of Security Requirements (NSR) *	Number of security requirements identified during the analysis phase of the application	-	SOFTWARE	MANUAL	STATIC*	¿?	If the design is available, it can be applied
394	Number of sub classes			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
395	NumFunctions			SOFTWARE			YES	If the software is available, it can be applied. It can be also applied to embedded software
400	Response set for a Class (RFC)	Set of methods that can potentially be executed in response to a message received by an object of that class. RFC is simply the number of methods in the set, including inherited methods	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
402	Stall Ratio	Stall ratio is a measure of how much a program's progress is impeded by frivolous activities. stall ratio = (lines of non-progressive statements in a loop) / (total lines in the loop)	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software

409	Comment ratio	Ratio of number of comment lines to number of code lines	-	SOFTWARE	AUTOMATIC*	STATIC*	YES	If the software is available, it can be applied. It can be also applied to embedded software
410	CommentDensity	The ratio of lines of comments to lines of code	-	SOFTWARE				
411	Compartmentalization	Compartmentalization means that systems and their components run in different compartments, isolated from each other. Thus a compromise of any of them does not impact the others. This metric can be measured as the number of independent components that do not trust each other (performs authentication and authorization for requests/calls coming from other system components) that the system is based on to deliver its function. The higher the compartmentalization value, the more secure the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the software is available, it can be applied. It can be also applied to embedded software
415	Defense-In-Depth	This metric verifies that security controls are used at different points in the system chain including network security, host security, and application security. Components that have critical data should employ security controls in the network, host, and component layers. To assess this metric we need to capture system architecture and deployment models as well as the security architecture model. Then we can calculate the ratio of components with critical data that apply the layered security principle compared to number of critical components	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	I am not sure if this metric can be applied to embedded software, or at verification phases
417	Fail Securely	The system does not disclose any data that should not be disclosed ordinarily at system failure. This includes system data as well as data about the system in case of exceptions. This metric can be evaluated from the security control responses – i.e. how the control behaves in case it failed to operate. From the system architecture perspective, we can assess it as the number of critical attributes and methods that can be accessed in a given component. The smaller the metric value, the likely more secure the system in case of failure	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	This could be used in evaluation
421	Isolation (PI)	This assesses the level of security isolation between system components. This means getting privileges to a component does not imply accessibility of other co-located components. This metric can be assessed using system architecture and deployment models. Components marked as confidential should not be hosted with nonconfidential (public) components. Methods that are not marked as confidential should not have access to confidential attributes or methods	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	I am not sure if this metric can be applied to embedded software
423	Least Privilege	This metric states that each component and user should be granted the minimal privileges required to complete their tasks. This metric can be assessed from two perspectives: from the security controls perspective we can review users' granted privileges. From the architectural analysis perspective this can be assessed as how the system is broken down to minimal possible actions i.e. the number of components that can access critical data. The smaller the value, the more secure the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the software is available, it can be applied. It can be also applied to embedded software
431	Trustworthiness	Trustworthiness of software means its worthy of being trusted to fulfill requirements which may be needed for a particular software component, application, system, or network. It involves attributes of stability, data security, quality, privacy, safety and so on. Software trustworthiness is interrelated with not only risk control in the software process, but also the quality management of the software development process. Furthermore, vision is needed to avoid excessive costs and schedule delays in development and risks management costs in operation; to improve development efforts; and above all	-	SOFTWARE	-	-	-	-
433	Variable Security Vulnerability	The security vulnerability of a variable v@l is determined by the combined security damage it might cause to the overall system security when v@l is attacked. The more security properties that can be left intact, the less vulnerable the variable is; on the other hand, the more security properties are violated, the more criticall the variable is.	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is only applicable if the source code is available
449	Faults found during manual inspection			SOFTWARE			YES	It is related to software and it can be applied to embedded software
458	Number of Design Flaws Related to Security (NSDF)	Security-related design flaws occur when software is planned and specified without proper consideration of security requirements and principles. For instance, clear-text passwords are considered as design flaws. Design flaws can be detected using design inspection techniques (e.g., design reviews). Identifying the number of design flaws related to security can help detect security issues earlier in the design process	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
459	Number of Exceptions That Have Been Implemented to Handle Execution Failures Related to Security (NEX)	Number of exceptions that have been included in the code to handle possible failures of the system due to an error in a code segment that has an impact on security	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
462	Number of Implementation Errors Found in the System (NERR)	Number of implementation errors of the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Related to the implementation phase
463	Number of Implementation Errors Related to Security (NSERR)	Number of implementation errors of the system that have a direct impact on security. One of the most common security related implementation errors is the buffer overflow	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Related to the implementation phase
464	Number of Omitted Exceptions for Handling Execution Failures Related to Security (NOEX)	Number of missing exceptions that have been omitted by software engineers when implementing the system. These exceptions can easily be determined through testing techniques.	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	Related to the implementation phase
465	Number of Omitted Security Requirements (NOSR)	Number of requirements that should have been considered when building the application	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	¿?	If the design is available, it can be applied
471	Number of violations of the LP principle	Number of violations of the Least Privilege principle. A component does not adhere to LP if it, based upon the permissions attributed as described before, is capable of executing tasks it is not responsible for	-	SOFTWARE	AUTOCATIC*	STATIC*	YES	This could be used in embedded systems
473	Percentage Software Hazards (PSH)	Comparing the number of software hazards identified against historical data, it indicates the sufficiency of the software hazard identification based on the identified hazards. (# software hazards / # System hazards) * 100	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	SÍ	Hazard is similar to risk.
480	Static analysis alert count (number of)			SOFTWARE				
488	Vulnerability Density	Number of vulnerabilities in the unit size of a code. VD = size of the software / # vulnerabilities in the system	-	SOFTWARE	SEMI-AUTOMATIC*	STATIC*	YES	It seems that it can be applied to embeded systems, but I am not sure if it is applicable in the evaluation phase

494	Security metrics of Arithmetic Expression	The security flaws of arithmetic expression include divide by zero, overflow of arithmetic operation, misused data type of arithmetic expression, and truncation error of arithmetic operation	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
495	Security Metrics of Array Index	The security flaws of array index include index out of range, incorrect index variable and uncontrollable array index	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
496	Security Metrics of Component Interfaces	The security flaws of component interface include inconsistent parameter numbers, inconsistent data types, and inconsistent size of data types	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
497	Security Metrics of Control Operation	The security flaws of control operations include infinite loop, deadlock situations and boundary defects	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
498	Security Metrics of I/O Management	File is an important media which can store critical information and record log data. However, file privilege definition, file access management, file contents, file organization and file size are major factors to affect the security software operation	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
499	Security Metrics of Input Format	The security flaws of data format include mismatched data contents, mismatched data type, mismatch data volume	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Part of a checklist
500	Security Metrics of Kernel Operation	A released software system must be adapted to a suitable operating system and environment. The kernel of operating system and environment become a critical factor to affect the operating security of software system. Resource management, execution file protection, main memory control and process scheduling are major tasks of operating system. The security flaws, which embedded in the tasks of operating system, become high security risk for software system operation	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified. Checklist. Only when there is an OS
501	Security Metrics of Network Environment	Network environment is a critical tool in the e-commerce years. However, data transmission management, remote access control, and transmission contents always become the security holes of software system operation	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified
502	Security Metrics of Resources Allocation	The security flaws of resource allocation include exhausted memory, unreleased memory exhausted resources, and unreleased resources	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified
503	Security Metrics of User Authority	Many users may use the released system to do specific jobs. However, user privilege definition, user password management and user detailed information maintenance are important tasks to control and manage the user authority. Without user authority control and management, the operation environment of software system will be on high security risk	-	SOFTWARE	AUTOMATIC*	STATIC*	¿?	This is not a metric, but a group of them. Individually, they could be used if they are specified
526	Kolmogorov Complexity			SOFTWARE				

Appendix B

Curriculum Vitae

Ángel Longueira-Romero

Education

2019–2022 **PhD in Industrial Cybersecurity**, *Faculty of Engineering*, Mondragon University, Metric-Based Cybersecurity Evaluation Methodology for Industrial Embedded Systems.
Specialization in:
- Cybersecurity Evaluation Standards.
- Vulnerability analysis.
- Cybersecurity measurement.

2017–2019 **Master's degree in Automation Engineering and Industrial Informatics**, *Gijón Polytechnic School of Engineering*, University of Oviedo, Specialised in Optical Techniques for Industrial Inspection.
Other Activities:
- Class President.

2012–2017 **Bachelor's degree in Industrial Electronics and Automation Engineering**, *Gijón Polytechnic School of Engineering*, University of Oviedo, Specialised in Industrial Robotics.
Other Activities:
- MENTOR Programme. Providing help to new Erasmus students in the campus.
- Member of the self-evaluation committee of the Industrial Electronics and Automation Engineering degree.
- Tandem language exchange programme of the University of Oviedo.

Academic Dissertations

2019–2022 **Ph.D. Thesis**, *Mondragon University*, Arrasate, Metric-Based Cybersecurity Evaluation Methodology for Industrial Embedded Systems.

2018–2019 **Master's Thesis**, *IKERLAN*, Arrasate, Design and Implementation of a PKI Infrastucture with Automatic Reenrolment of Certificates for Industrial Embedded Systems.
With Honors

2017 **Undergraduate Thesis**, *Thermal machinery and engines team. Department of Energy*, Gijón, Design of a temperature regulation system for a Stirling engine heater.
With Honors

Training

2018 **IEC 61131 Edu Net Certificate**, *Automation technology in theory and practice according to IEC 61131.*

Experience

2022–Present **Industrial Cybersecurity Researcher**, *IKERLAN*, Arrasate (Spain).

2019–2022 **PhD Candidate**, *IKERLAN*, Arrasate (Spain).

2018–2019 **Industrial Cybersecurity Intern**, *IKERLAN*, Arrasate (Spain), Worked at the Industrial Cybersecurity Team deploying a PKI with automatic reenrolment using SCEP, EST and OCSP for embedded devices with two HSMs.

✉ *aalongueira@hotmail.com* • *LinkedIn* • *aalongueira Web Page*

2017–2018 **Amplification Equipment Repair Apprentice**, *Ortodoxo Amplification (Now: Bell Tone Vintage)*, Gijón, Worked with Marco Fernández designing, building and repairing vacuum tube and solid state musical equipment, and developing ad hoc tools.

2016–2017 **Mentoring and Tutoring**, Private lessons of industrial technology, mathematics, physics, chemistry, technical drawing and music to high school students.

Technical Skills

OSs	Windows, GNU/Linux.
Programming	C, C++, Python, assembly.
Databases	PostgreSQL, MySQL.
Software	MatLab, R, Altium.
Hardware	FPGAs, Microcontrollers, PCB design.
Standards	ISO 9001, ISO 27005, ISA/IEC 62443, Common Criteria, SAE J3061.
Methodologies	NIST 800 Serie Reports, OWASP, OSSTMM.
Data Analysis	Machine Learning, Computer vision, Data Processing, Data Visualisation.
Cybersecurity	Cryptography, Public Key Infrastructure (PKI), Risk Analysis (MAGERIT), Threat Modelling (STRIDE).
Protocolos	SCEP, EST, OCSP.
Other	Microsoft Office, LaTeX.

Languages

Spanish	**Native**	
English	**Fluent**	
Basque	**Intermediate**	*Conversationally fluent*

Transferable Skills

- Teamwork
- Written and verbal communication
- Fast learner and proactive problem solving
- Self management
- Research and analytical skills

Interests

- 3D design with Fusion 360 (Autodesk)
- Arduino projects
- Language learning
- Travelling
- 3D printing (Simplify3D and Cura)
- Raspberry projects
- CrossFit and Bike
- Music (Electric Bass and Oboe)

Other Information

- Willing to travel
- Driving license
- Interested in a wide variety of topics