Deliverable 3.1

Report on the State of the Art of Vulnerability Management

Technical References

Document Version	:	1
Submission Date	:	27/02/2021
Dissemination Level	:	Public
Contribution to	:	WP3- Vulnerability Management
Document Owner	:	GRAD
File Name	:	Report of the State of the Art of Vulnerability Management
Revision	:	3.0

Project Acronym	:	BIECO
Project Title	:	Building Trust in Ecosystem and Ecosystem Components
Grant Agreement n.	:	952702
Call	:	H2020-SU-ICT-2018-2020
Project Duration	:	36 months, from 01/09/2020 to 31/08/2023
Website	:	https://www.bieco.org

Revision History

REVISION	DATE	INVOLVED PARTNERS	DESCRIPTION
0.0	09/11/2020	GRAD	Table of Contents
0.1	08/12/2020	GRAD	Contributions to Vulnerability Detection and Exploitability Forecasting
0.2	10/12/2020	UMU	Introduction
03	20/12/2020	UMU	Contributions to Section 2
04	04/01/2021	UMU	MUD Contribution
05	06/01/2021	UTC	Contributions to Section 2 and 4
06	10/01/2021	7B	Contribution to Section 4
07	22/01/2021	GRAD	Contributions to Subsection 3.4 and Section 5
08	11/02/2021	GRAD	Contributions to Executive Summary and Conclusions
1.0	16/02/2021	GRAD	Review by Internal Reviewer and Work Package leader Review, Lilian Adkinson
1.1	19.02.2021	GRAD	Implementing Reviewers Suggestion and Update
2.0	20.02.2021	IESE	Review by External Reviewer, Emilia Cioroaica
2.1	23/02/2021	UNI	Review by External Reviewer, Sanaz Nikghadam
2.2	24/02/2021	GRAD	Implementing Reviewers Suggestion and Update
2.4	25.02.2021	UNI	Review by Coordinator, Jose Barata
3.0	27.02.2021	UNI	Final Version (3th)

List of Contributors

Deliverable Creator(s):Nora M. Villanueva (GRAD), Eva Sotos (GRAD), Borja Pintos (GRAD), Javier Yépez (GRAD), Sara Matheu (UMU), Ovidiu Cosma (UTC), Ioana Zelina (UTC), Mara Macelaru (UTC), Paweł Skrzypek (7B), Radosław Piliszek (7B)

Reviewer: Lilian Adkinson (GRAD), Emilia Cioroaica (IESE), Sanaz Nikghadam-Hojjat(UNI), José Barata (UNI)

BIECO project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **952702**.

Acronyms

Acronym	Term
ABOD	Angle-based Outlier Detection
ACL	Access Control List
ACO	Ant Colony Optimization
AI	Artificial Intelligence
AR	Autoregressive
ARCH	Autoregressive Conditional Heteroskedasticity
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
AST	Abstract Syntax Tree
AVR	Attribute Vulnerability Ratio
BFS	Breadth First Search
CFG	Control Flow Graph
CIA	Change Impact Analysis
CIA (triad)	Confidentiality, Integrity and Availability
CNN	Convolutional Neural Network
CPE	Common Platform Enumeration
CSRF	Cross-Site Request Forgery
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DBSCAN	Density-Based Spatial Clustering of Applications with Noire
EDB	Exploit Database Archive
ENISA	European Union Cyber Security Agency
EPDG	Enhanced Procedure Dependence Graphs
EPSS	Exploit Prediction Scoring System
FL	Federated Learning

GARCH	Generalized Autoregressive Conditional Heteroskedasticity
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IVS	Input Validation and Scalable attributes
LOF	Local Outlier Factor
LSTM	Long Short-Term Memory
MA	Moving Average
ML	Machine Learning
MSPL	Medium-level Security Policy Language
MUD	Manufacturer Usage Description
N-BEATS	Neural basis expansion analysis for interpretable time series forecasting
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NTBD	National Business Behaviour Database
NVD	National Vulnerability Database
PoC	Proof of Concepts
QoS	Quality of Service
RCE	Remote Code Execution
RNN	Recurrent Neural Network
SARIMA	Seasonal Autoregressive Integrated Moving Average
SDN	Software Defined Networks
SFM	State-Frequency Memory
SOD	Subspace Outlier Degree
SRAM	Security risk analysis model
STAR	Smooth Threshold Autoregressive
SVM	Support Vector Machine
SUIT	Internet of Things Software Update
TAR	Threshold Autoregressive
VND	Vulnerability Notes Database

VPM	Vulnerability Prediction Models
XACML	eXtensible Access Control Markup Language
XSS	cross-site scripting attacks
ZDI	Zero Day Initiative

Executive Summary

One of the main purposes of cybersecurity is to guarantee the properties of the CIA triad (Confidentiality, Integrity, and Availability) [1], also known as the CIA triangle of data and services (Figure 1). Confidentiality refers to the prevention of an information disclosure to unauthorized entities or individuals; integrity implies that data cannot be modified without detection, ensuring its correctness; and availability seeks to provide uninterrupted access to the system by legitimate users, whenever it is required. Hence, CIA properties become an important gear for information security. These properties are the security requirements that a computer system should accomplish, and they are directly linked to each other, so a balance between them guarantees high levels of security and trust.

Figure 1 CIA triad standing for Confidentiality, Integrity and Availability.

As a consequence, a failure in any of these three properties can impact the rest of them, affecting the trust of the user on the system. As an example, the lack of confidentiality can lead to a high probability of integrity violation, and the modification of the integrity data can provoke applications to stop working in a proper way, affecting availability. In particular, one of the main issues we can face when it comes to the fulfilment of such properties is the existence of security vulnerabilities. This leads their detection and analysis to be crucial for ensuring the security and trustworthiness of the system.

More in particular, a security vulnerability is a weakness that can be exploited by an attacker in order to compromise the confidentiality, availability or integrity of a system[1]**.** Nowadays, the number of vulnerabilities disclosed are increasing every year[2], and those that are present in widely-used systems can cause severe economic, reputational and even societal harms. Therefore, it is essential to identify these vulnerabilities on an early stage of the systems' development life cycle, and improve the assessment processes and tools that allow to detect, classify, evaluate and mitigate vulnerabilities on an accurate manner.

As the first step of the vulnerability assessment process, the identification is critical. In this sense, substantial research has been devoted to techniques that analyse source code in order to detect and characterize security vulnerabilities [2], but also to evaluate how a vulnerability could propagate to other elements of the software supply chain [e.g., 3, 4]. In BIECO project, the vulnerability identification and characterization process will focus mainly on three topics:

1) **Detection:** it consists on the accurate identification of software vulnerabilities within a source code. For this purpose, BIECO will explore the use of Machine Learning and data mining techniques, such as anomaly detection-based techniques [e.g., 5, 6], vulnerable code pattern recognition [e.g., 7, 8] and vulnerability prediction models [e.g., 9, 10].

[1] https://cve.mitre.org/about/terminology.html
[2] https://www.cvedetails.com/browse-by-date.php

2) **Forecasting:** it allows to make predictions of future data on time series domain, i.e., where data are collected at regular intervals over time (e.g., hourly, daily, monthly, annually). In the context of BIECO, the idea is two-fold: i) forecasting the number of vulnerabilities [e.g., 11, 12, 13] and ii) forecasting the period of time in which these vulnerabilities could be exploited (e.g., within the next 12 months) [e.g., 14, 15].
3) **Propagation:** it offers an estimation of how a localized vulnerability can affect the rest of the code. For this purpose, studies based on graph theory will be analysed [e.g., 16, 17], as well as optimization path algorithms such as Ant Colony Optimization (ACO) [18, 19]. Moreover, the applicability of the recent standard Manufacturer Usage Description (MUD)[3] will be also assessed.

In particular, this document provides a review on the current state of the art about vulnerabilities assessment related to the three topics mentioned before (detection, forecasting and propagation). The deliverable starts with an introduction to the concept of security vulnerabilities and continues with a summary of the most representative standards in the field, as well as a compilation of vulnerability datasets, including the NVD (National Vulnerability Database)[4]. The document presents then a review of several state-of-the-art techniques for the assessment of software vulnerabilities, including their identification, forecasting and propagation. Finally, some conclusions are provided in order to summarize the most important reviewed topics.

Project Summary

Nowadays most of the ICT solutions developed by companies require the integration or collaboration with other ICT components, which are typically developed by third parties. Even though this kind of procedures are key in order to maintain productivity and competitiveness, the fragmentation of the supply chain can pose a high risk regarding security, as in most of the cases there is no way to verify if these other solutions have vulnerabilities or if they have been built taking into account the best security practices.

In order to deal with these issues, it is important that companies make a change on their mindset, assuming an "untrusted by default" position. According to a recent study only 29% of IT business know that their ecosystem partners are compliant and resilient with regard to security. However, cybersecurity attacks have a high economic impact and it is not enough to rely only on trust. ICT components need to be able to provide verifiable guarantees regarding their security and privacy properties. It is also imperative to detect more accurately vulnerabilities from ICT components and understand how they can propagate over the supply chain and impact on ICT ecosystems. However, it is well known that most of the vulnerabilities can remain undetected for years, so it is necessary to provide advanced tools for guaranteeing resilience and also better mitigation strategies, as cybersecurity incidents will happen. Finally, it is necessary to expand the horizons of the current risk assessment and auditing processes, taking into account a much wider threat landscape. BIECO is a holistic framework that will provide these mechanisms in order to help companies to understand and manage the cybersecurity risks and threats they are subject to when they become part of the ICT supply chain. The framework, composed by a set of tools and methodologies, will address the challenges related to vulnerability management, resilience, and auditing of complex systems.

[3] https://tools.ietf.org/html/rfc8520
[4] https://nvd.nist.gov

Partners


Disclaimer

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

In recent years, an alarming increase in cybersecurity attacks has been detected[5]. Especially with the COVID pandemic, where teleworking is the new lifestyle, attackers have at their disposal a huge and often insufficiently protected attack surface. One of the best-known examples is the *WannaCry* ransomware, which affected more than 230,000 computers in 150 countries. The most affected countries were Russia; Ukraine; India; Great Britain, where the National Health Service was compromised; Spain, for the attack on Telefónica and Germany, where the German railway company Deutsche Bahn AG was the main target. Cyber attackers collected more than 140,000 dollars in bitcoins.

This is causing companies huge economic losses, service interruptions and great social concern. These attacks damage the company's reputation, causing customers to be lost. In addition to the security of information systems, such as firewall mechanisms to prevent DDoS attacks, organizations must focus on the development of their software applications. According to the IBM security summit in 2016, 60% of cyberattacks benefits from inside[6], benefiting from bugs.

Bugs are nothing more than programming errors, and most of them are completely harmless beyond affecting the performance of the product. However, some bugs can be exploited by malicious external entities, in order to obtain certain benefits (private data, access to the system, interruption of the service, etc.). In this case we are no longer talking about bugs, but about vulnerabilities and weaknesses.

Taking into account the impact that a simple vulnerability can have, it is essential to further research and improve the existent vulnerability assessment mechanisms. This document presents a review on state-of-the-art techniques for vulnerability assessment, as well as a compilation of relevant vulnerability related standards and databases. The results of this review will be used as an input for the design and development of the vulnerability assessment tools that will be produced by the WP3 of the BIECO project.

1.2. Background

A **security vulnerability** is defined by the European Union Agency for Cybersecurity (ENISA)[7] as a weakness an adversary could take advantage of to compromise the confidentiality, availability, or integrity of a resource. At the same time, a **weakness** refers to implementation flaws or security implications due to design choices. For example, a lack of control over the length of the data entered could lead to a buffer overflow vulnerability, allowing attackers to steal or corrupt private information, or even run malicious code.

An added problem is the propagation of vulnerabilities either within the same component or between different components. Although a certain functionality can be designed in a secure way, the reality is that its interaction with a vulnerable functionality or component can make the entire product insecure. An example is the one that we can find within the Maven project[8] , where the POM file in org.wso2.carbon.security.policy[9] has a dependency with components from org.apache.derby[10]. This dependency provokes that vulnerabilities existing in org.apache.derby,

[5] https://www.cpomagazine.com/cyber-security/new-security-report-breaks-down-increase-in-cyber-attacks-due-to-remote-work-lack-of-training-overwhelmed-it-departments-are-the-main-issues/
[6] https://www.ituser.es/seguridad/2016/09/el-ibm-security-summit-2016-pone-foco-en-la-seguridad-cognitiva
[7] https://www.enisa.europa.eu/topics/csirts-in-europe/glossary/vulnerabilities-and-exploits
[8] http://maven.apache.org/
[9] https://mvnrepository.com/artifact/org.wso2.carbon/org.wso2.carbon.security.policy
[10] https://mvnrepository.com/artifact/org.apache.derby/derby/10.11.1.1

such as CVE-2015-1832[11], can cause potential vulnerability threats in the Maven project. Therefore, it is not only necessary to detect and correct the vulnerabilities of our component in an isolated way, but also to analyse the possible consequences derived from the propagation of vulnerabilities coming from another component.

Vulnerability management arises as a way to identify, classify, evaluate and mitigate vulnerabilities. Following the definition from ENISA, vulnerability management comprises several steps:

- **Preparation:** defining the scope of the vulnerability management process.
- **Vulnerability scanning:** vulnerability scanners are automated tools that scan a system for known security vulnerabilities providing a report with all the identified vulnerabilities sorted based on their severity.
- **Identification, classification and evaluation of the vulnerabilities**: the vulnerability scanner provides a report of the identified vulnerabilities.
- **Remediating actions:** the asset owner determines which of the vulnerabilities will be mitigated.
- **Rescan:** once the remediating actions are completed, a rescan is performed to verify their effectiveness.

This document presents an overview of the state of the art regarding vulnerability assessment and, in particular, it focuses on the review of vulnerability scanning methods based mainly on Artificial Intelligence techniques. Section 2 introduces the main standards that can be used to structure vulnerabilities information, as well as a summary of the main databases that contain information for their characterization. Section 3 reviews the main techniques for vulnerability detection (i.e., scanning), paying special attention to the techniques based on anomaly detection, patterns and prediction models. After that, Section 4 reviews the main techniques for forecasting, which aim to give an estimation on the number of vulnerabilities that could arise and the probability of their exploitation in a certain period of time, whereas Section 5 focuses on the vulnerability propagation analysis techniques, as well as the applicability of the recent standard Manufacturer Usage Description (MUD) for vulnerability assessment. Finally, the document ends with a summary of the conclusions.

[11] https://nvd.nist.gov/vuln/detail/CVE-2015-1832

2. Analysis of Vulnerability Public Information

This section presents different types of information that are essential in order to characterize adequately a security vulnerability and, therefore, that should be taken into account during the design and development of BIECO's vulnerability assessment tools. The section includes a selection of relevant vulnerability standards, a compilation of vulnerability datasets, as well as other vulnerability compilation projects.

A vulnerability assessment process can be significantly difficult without a common baseline. Therefore, when dealing with vulnerabilities, it is important to take into account the most common standards in the fields (subsection 2.1). Having a structured information about vulnerabilities simplifies their assessment process, providing a common understanding of the context of which the different vulnerabilities are discovered.

As the standardization in the vulnerability field has become more mature, several vulnerability databases have emerged. These databases are just data repositories, typically public, that compile software and hardware vulnerabilities information. To date, there have been published several databases that can be used as a support for the assessment of vulnerabilities, and that provide users with different types of information. Subsection 2.2 will review the most relevant vulnerability databases, paying special attention to National Vulnerability Database (NVD)[12].

Finally, subsection 2.3 introduces OWASP, a reference project that compiles some of the most relevant vulnerabilities.

2.1. Vulnerability Standards

In this subsection we present a selection of vulnerability related standards, such as Common Vulnerabilities and Exposures (CVE)[13], Common Weakness Enumeration (CWE)[14], Common Platform Enumeration (CPE)[15] and Common Vulnerability Scoring System (CVSS)[16]. All of them are well-known and widespread standards: from identifying vulnerabilities (CVE), to describing common weaknesses in software (CWE), to providing consistent names for referring to operating systems, hardware and applications (CPE), up to the rating of the severity of vulnerabilities (CVSS).

2.1.1. Common Vulnerabilities and Exposures (CVE)

Common Vulnerabilities and Exposures (CVE) is a list of records of public known information about security vulnerabilities, widely used by multiple IT vendors. Its identifiers (CVE-YYYY-XXXX) enable a common understanding about vulnerabilities and help with the evaluation of the coverage of vulnerability tools and services.

CVE is the industry standard for vulnerability and exposure identifiers whose records provide reference points for data exchange so that cybersecurity products and services can speak with each other. Products and services compatible with CVE provide easier interoperability, and enhanced security.

[12] https://nvd.nist.gov
[13] https://cve.mitre.org/
[14] https://cwe.mitre.org/about/index.html
[15] https://nvd.nist.gov/products/cpe
[16] https://www.first.org/cvss/v3-1/

The CVE List feeds the NVD or National Vulnerability Database (see section 2.2.1), which then builds upon the information included in CVE Records to provide enhanced information for each record such as fix information, severity scores, and impact ratings.

2.1.2. Common Weakness Enumeration (CWE)

Common Weakness Enumeration (CWE) is a dictionary of unique identifiers of common software weaknesses (also hardware weaknesses from 2020). This project is supported by the U.S. Department of Homeland Security (DHS) Cybersecurity and Infrastructure Security Agency (CISA), the Homeland Security Systems Engineering and the Development Institute (HSSEDI), which is operated by The MITRE Corporation.

It offers a wealth of options that can describe common weaknesses such as detection methods, consequences, affected resources and likelihood. Even though CWE seems to be related to CVE, CWE does not deal with specific software vulnerabilities. For example, CWE would describe a buffer overflow in multiple software types, but a CVE ID would be assigned to one specific buffer overflow vulnerability in Cisco IOS version X.

Nonetheless, CWE is useful for:

- Describing and discussing software and hardware weaknesses in a common language.
- Checking for weaknesses in existing software and hardware products.
- Leveraging a common baseline standard for weakness identification, mitigation, and prevention efforts.
- Preventing software and hardware vulnerabilities prior to deployment.

2.1.3. Common Platform Enumeration (CPE)

Common Platform Enumeration (CPE) is a structured naming scheme for information technology systems, software and packages. Based upon the generic syntax for Uniform Resource Identifiers (URI), CPE includes a formal name format, a method for checking names against a system and a description format for binding text and tests to a name.

The CPE dictionary contains the official list of CPE names. The dictionary is provided in XML format, which follows the CPE XML schema[17]. In particular a typical CPE name would follow the structure *cpe:/{part}:{vendor}:{product}:{version}:{update}:{edition}:{language}*[18].

In the context of vulnerability assessment, the CPE allows to identify unequivocally within a CVE the software and version that is affected by the vulnerability.

2.1.4. Common Vulnerability Scoring System (CVSS)

Common Vulnerability Scoring System (CVSS) is a free and open industry standard for assessing the severity of a security vulnerability. The standard provides a way to capture and understand the principal characteristics of a vulnerability by means of assigning severity scores. These scores, which can take a value on the range from 0 to 10 (being 10 the most severe), are calculated based on a multi-formula process that depends on several metrics.

[17] https://csrc.nist.gov/schema/cpe/2.3/cpe-dictionary_2.3.xsd
[18] For example, cpe:/a:microsoft:internet_explorer:8.0.6001:beta

Historically, vendors have used their own methods for scoring software vulnerabilities, usually without detailing their criteria or processes. This creates a major problem for users, particularly those who manage disparate IT systems and applications. In this sense, the goal for CVSS is to facilitate the generation of consistent scores that accurately represent the impact of vulnerabilities, as it provides full details regarding the parameters used to generate each score. The first version of CVSS was released in 2005, version 2 was released in 2007 and version 3 in 2015. The current version is 3.1. It was released in 2019.

The CVSS is calculated from three metrics groups, i.e., **Base**, **Temporal, and Environmental**. Each of these metrics are calculated taking into account a set of sub scores obtained from several features. This process is explained in subsection 2.1.4.4 and the metrics groups are detailed below.

2.1.4.1. Basic Metrics

These set of metrics reflect the severity of vulnerabilities according to their characteristics and assumes the worst-case impact across different environments. Base metrics do not change over time and are common to all environments. For a better understanding of the vulnerability, base metric is divided in three subtypes, Exploitability metrics, Scope metric and Impact metrics, which, in turn, are made up of a set of metrics.

The **Exploitability metrics** represent the characteristics of a vulnerable component, and reflect the technical means by which the vulnerability can be exploited. This metric is composed of four features:

1. **Attack Vector** reflects the context by which vulnerability exploitation is possible. Its value is greater the more distant an attacker can be to exploit the vulnerable component. The possible *values* (and sub scores) of Attack Vector are:
    - *Network* (0.85) if the vulnerable component is bound to the network stack and the set of possible attackers extends up to the entire Internet.
    - *Adjacent* (0.62) if the attack is limited at the protocol level to a logically adjacent topology. An attack must be launched from the same physical or logical network.
    - *Local* (0.55) if the vulnerable component is not bound to the network stack. The attacker must access the target system locally or remotely (e.g., Telnet, SSH), or interact with another person to perform the required actions, using social engineering techniques.
    - *Physical* (0.2) if the attack requires the physical touch or manipulation of the vulnerable component.
2. **Attack Complexity** represents the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. The possible *values* (and sub scores) are:
    - *Low* (0.77) if specialized access conditions or extenuating circumstances do not exist. Repeatable success can be expected.
    - *High* (0.44) if successful attack requires certain preparation operations performed against the vulnerable component.
3. **Privileges Required** describes the level of privileges an attacker must possess before successfully exploiting the vulnerability. The possible *values* (and sub scores) are:
    - *None* (0.85) if no access to settings or files of the vulnerable component is required to carry out an attack.

- *Low* (0.62 if the Scope metric value is *Unchanged*, or 0.68 if it is *Changed*) if an attack requires privileges that provide basic user capabilities that affect only settings and files owned by a user.
- *High* (0.27 if the Scope metric value is *Unchanged*, or 0.5 if it is *Changed*) if an attack requires privileges that provide significant (e.g., administrative) control over the vulnerable component, allowing access to component-wide settings and files.

4. **User Interaction** captures the requirement for a human user, other than the attacker, to participate in the successful compromise of the vulnerable component. The possible *values* (and sub scores) are:
    - *None* (0.85) if no interaction from any user is required to exploit the vulnerability.
    - *Required* (0.62) if user action is required for vulnerability exploitation.

The **Scope metric** indicates whether a vulnerability in one vulnerable component impacts resources in components beyond its security scope. The security scope of a component encompasses other components that provide functionality solely to that component, even if these other components have their own security authority. The possible values are:

- *Unchanged* if an exploited vulnerability can only affect resources managed by the same security authority.
- *Changed* if an exploited vulnerability can affect resources beyond the security scope managed by the security authority of the vulnerable component.

The **Impact metric** reflects the consequence of a successful exploit over the impacted component, which could be a software application, a hardware device or a network resource. The metric is composed of three groups:

1. **Confidentiality** measures the impact to the confidentiality of the information resources managed by a software component, due to a successfully exploited vulnerability. The possible *values* (and sub scores) are:
    - *High* (0.56) if there is a serious loss of confidentiality, resulting in all or some of the resources within the impacted component being divulged to the attacker.
    - *Low* (0.22) if there is some loss of confidentiality, but the attacker does not have control over what information is obtained and the attack does not cause a direct, serious loss to the impacted component.
    - *None* (0) if there is no loss of confidentiality within the impacted component.
2. **Integrity** measures the impact of a successfully exploited vulnerability to the trustworthiness and veracity of information. The possible *values* (and sub scores) are:
    - *High* (0.56) if there is a serious loss of integrity, or a complete loss of protection. A successful attack can modify protected files, resulting a serious consequence to the impacted component.
    - *Low* (0.22) if modification of data is possible, but there is no control over the consequences, or the amount of modification is limited. The modifications have a partial effect on the impacted component.
    - *None* (0) if there is no loss of integrity within the impacted component.
3. **Availability** measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. The possible *values* (and sub scores) are:

- *High* (0.56) if a successful attack can fully or partially deny access to resources, presenting a serious consequence to the impacted component. The loss can be either sustained (during the attack) or persistent if the condition persists after the attack has completed.
- *Low* (0.22) if a successful attack can reduce performance or cause interruptions in resource availability but overall, there is no serious consequence to the impacted component, there is no complete denial of service to legitimate users.
- *None* (0) if there is no impact to availability within the impacted component.

2.1.4.2. Temporal Metrics

These metrics measure the current state of exploit techniques or code availability, the existence of any patches or workarounds, or the confidence in the description of a vulnerability. These metrics adjust the Base severity of a vulnerability based on factors that change over time, but not across environments. The availability of a simple-to-use exploit kit would increase the CVSS score, while the release of a patch would decrease it.

Temporal Metrics are composed of three groups:

1. **Exploit Code Maturity** measures the probability of the vulnerability being exploited. It is based on the current state of exploit techniques, exploit code availability or active exploitation. The possible *values* (and sub scores) are:
    - *Not Defined* (1) if there is insufficient information to choose one of the other values. It has the same effect on scoring as the next variant (*High*).
    - *High* (1) if exploit development has reached the level of reliable, widely available, easy-to-use automated tools. Exploit code works in every situation or is actively being delivered via an autonomous agent (such as a worm or virus). Network-connected systems are likely to encounter scanning or exploitation attempts.
    - *Functional* (0.97) if functional exploit code is available, that works in most situations.
    - *Proof-of-Concept* (0.94) if proof-of-concept exploit code or technique is available. The code or technique is not functional in all situations and may require substantial modification by a skilled attacker.
    - *Unproven* (0.91) if no exploit code is available, the exploit is theoretical.
2. **Remediation Level** reflects the level of vulnerability mitigation techniques. The possible *values* (and sub scores) are:
    - *Not Defined* (1) if there is insufficient information to choose one of the other values. It has the same effect on scoring as assigning *Unavailable*.
    - *Unavailable* (1) if there is no available solution, or it is impossible to be applied.
    - *Workaround* (0.97) if there is an unofficial, non-vendor solution available.
    - *Temporary Fix* (0.96) if there is available an official temporary fix or workaround.
    - *Official Fix* (0.95) if an official solution is available such as a patch or an upgrade.
3. **Report Confidence** measures the degree of confidence in the existence of the vulnerability and the credibility of the known technical details. The possible *values* (and sub scores) are:

- *Not Defined* (1) if there is insufficient information to choose one of the other values. It has the same effect on scoring as assigning *Confirmed*.
- *Confirmed* (1) if detailed reports or functional exploits exist, or the vulnerability is confirmed by the author or vendor of the affected code.
- *Reasonable* (0.96) if significant details are available, but the finds are not fully confirmed. Reasonable confidence exists that the vulnerability is real and at least one impact can be verified.
- *Unknown* (0.92) if there are reports of impacts that indicate a vulnerability is present, but the true nature of the vulnerability is uncertain.

2.1.4.3. Environmental Metrics

The Environmental Metrics adjust the Base severities to a specific computing environment. They consider factors such as the presence of mitigations in that environment. These set of metrics enable the customization of the CVSS score depending on the importance of the affected IT asset to an organization and they are the modified equivalents of Base Metrics. They are composed of two groups, Security Requirements and Modified Base Metrics, which are useful to understand impact of the vulnerability:

1. **Security Requirements** enable the customization of the Impact Metrics (Confidentiality, Integrity and Availability). The full effect on the environmental score is determined by the corresponding Modified Base Impact metrics. The possible *values* (and sub scores) are:
    - *Not Defined* (1) if there is insufficient information to choose one of the other values. It has no impact on the overall Environmental Score. It has the same effect as *Medium*.
    - *High* (1.5) if the customized metric has a great importance for the organization.
    - *Medium* (1) if the metric does not need customization.
    - *Low* (0.5) if the customized metric is of little importance for the organization.
2. **Modified Base Metrics** allow the Base Metrics to be overridden, based on the specific characteristics of the environment. Thus Exploitability, Scope, or Impact can be reflected via an appropriately modified Environmental Score. These metrics modify the Environmental Score by overriding Base Metric values, prior to applying the Environmental Security Requirements. Each Modified Environmental metric has the same values as its corresponding Base metric, plus a value of *Not Defined*, that is that is equivalent with the associated Base metric.

2.1.4.4. CVSS Scoring

The CVSS scoring process is depicted in Figure 2. The tree sets of metrics mentioned above (Base, Temporal and Environmental) are those that are going to define the CVSS score. They have the following qualitative ratings: *None* (0), *Low* (0.1 – 3.9), *Medium* (4.0 – 6.9), *High* (7.0 – 8.9) and *Critical* (9.0 – 10).

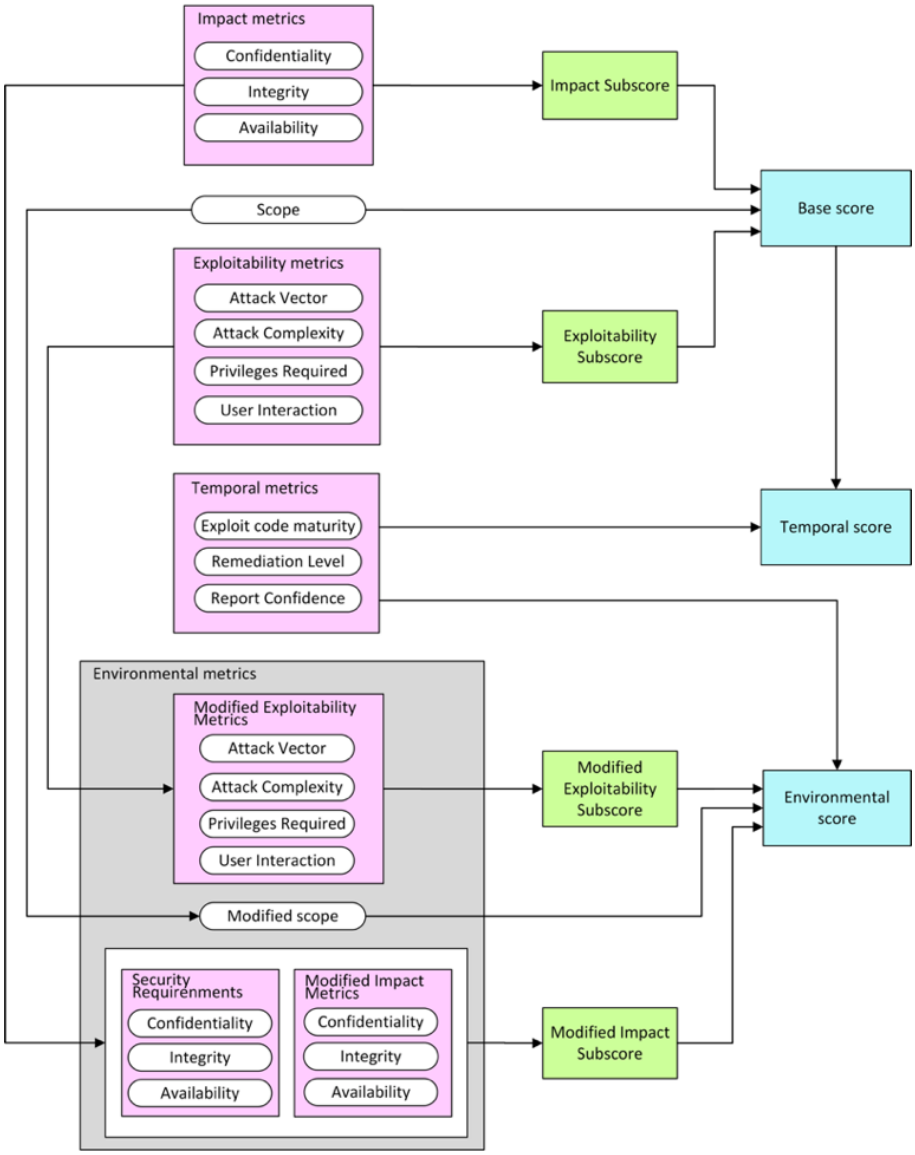

Figure 2 The CVSS scoring process[19].

The Impact and Exploitability metrics are used first, to determine Impact and Exploitability sub scores. Then the Base score is determined based on these sub scores, together with the Scope metric. The Base Score can then be refined by scoring the Temporal and Environmental metrics, in order to more accurately reflect the relative severity posed by a vulnerability to a user's environment at a specific point in time. Scoring the Temporal and Environmental metrics is not required but is recommended for more precise scores.

The Base and Temporal metrics are specified by vulnerability bulletin analysts, security product vendors, or application vendors because they typically possess the most accurate information about the characteristics of a vulnerability. The Environmental metrics are specified by end-user organizations because they are best able to assess the potential impact of a vulnerability within their own computing environment.

CVSS score is an important feature and standard in the field of vulnerabilities. This scoring provides a numerical score about vulnerability behaviour and the possible severity of its attack.

[19] https://www.first.org/cvss/specification-document#CVSS-v3-1-Equations

Thus, the use of CVSS score could be beneficial in the development of vulnerability assessment tools, although its applicability at BIECO framework is still under study.

2.2. Vulnerability Databases

2.2.1. National Vulnerability Database (NVD)

The National Vulnerability Database (NVD) is one of the reference vulnerability databases of interest of our study. It is maintained by the National Institute of Standards and Technology (NIST) Computer Security Division, Information Technology Laboratory and is sponsored by the Cybersecurity & infrastructure Security Agency. It is the U.S. government repository of standards-based vulnerability management data represented using the Security Content Automation Protocol (SCAP). This data enables automation of vulnerability management, security measurement, and compliance. Although other vulnerability databases exist, the NVD still remains widely used and the most exhaustive resource in security vulnerability data. Based on this, it would help our study providing a list of vulnerabilities and exposure data for training machine learning models.

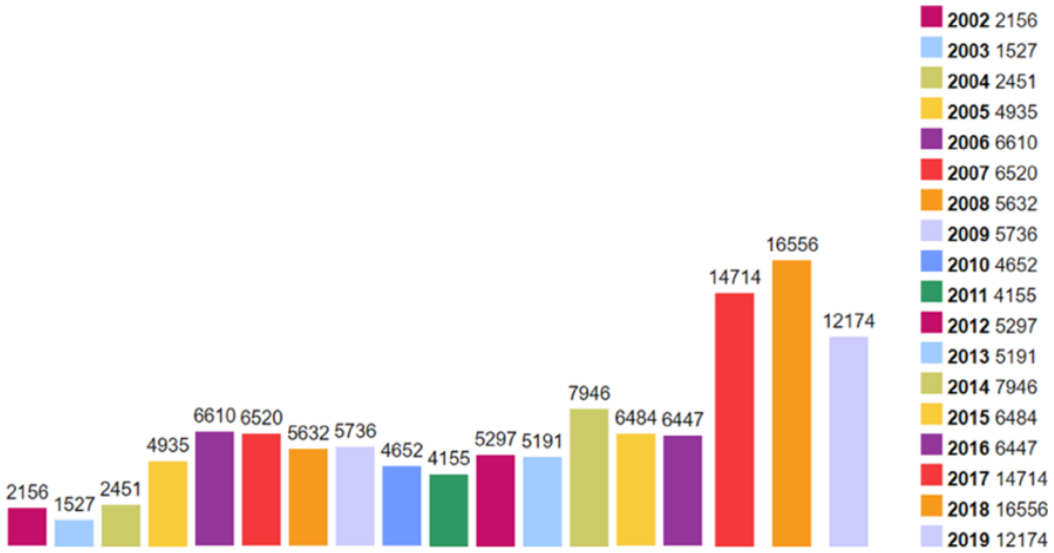

Figure 3 Number of vulnerabilities by year (from 2002 to 2019)[20].

NVD includes databases of security checklist references, security-related software flaws, misconfigurations, product names and impact metrics, and it records CVEs since 2002. The information in NVD is updated permanently, as new information becomes available, and it is fully synchronized with the CVE List so that any updates to CVE appear immediately in NVD. As it can be seen in Figure 3, in recent years the number of registered vulnerabilities has increasing trend, with a maximum value reached in 2018 when 16,556 vulnerabilities were registered.

In particular, the distribution of the main types of vulnerabilities registered in NVD is shown in Figure 4. According to the number of vulnerabilities, the first three places are taken in order by *Execute Code*, *Denial of Service* and *Overflow*.

[20] Taken from https://www.cvedetails.com/browse-by-date.php

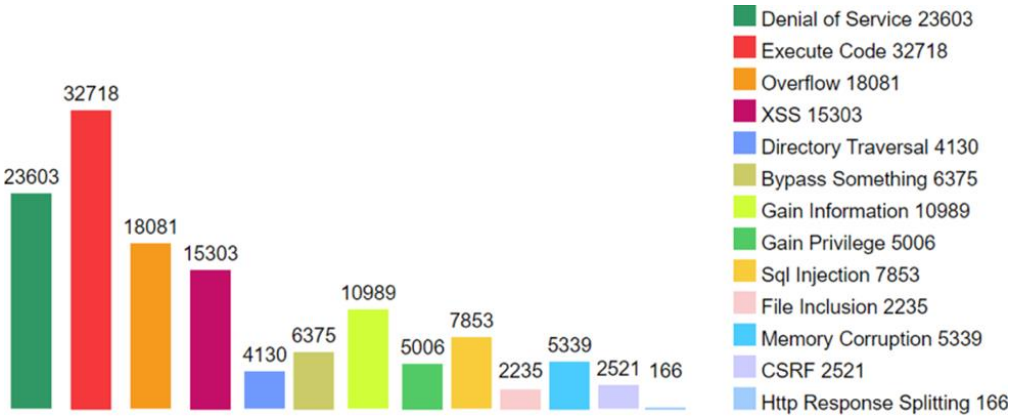

Figure 4 Number of vulnerabilities by type.[21]

The NVD data feeds are available at in JSON format[22], which follows the NVD JSON schema[23]. The most important fields that are registered for each vulnerability are the following:

- **ID:** the unique identifier of the CVE.
- **problemtype:** a list of CWEs that are related to the CVE.
- **reference:** links to external sources of information.
- **description:** description of the CVE in text format.
- **configurations:** a list of CPEs affected by the vulnerability.
- **impact:** CVSS v3.1 Base metrics, Base score, Exploitability and Impact subscores, vector string, and Base severity. There are also available CVSS v2 metrics and score.
- **publishedDate:** date of publication in NVD.
- **lastModifiedDate:** date of the last modification.

Finally, Figure 5 shows the distribution of the CVSS scores of the vulnerabilities registered in NVD. In the figure it is possible to observe that the number of vulnerabilities with CVSS scores less or equal to 3-4 is lower than those ones with CVSS scores equal or greater than 4-5, with one exception, vulnerabilities with CVSS scores between 8 and 9 are almost non-existent.

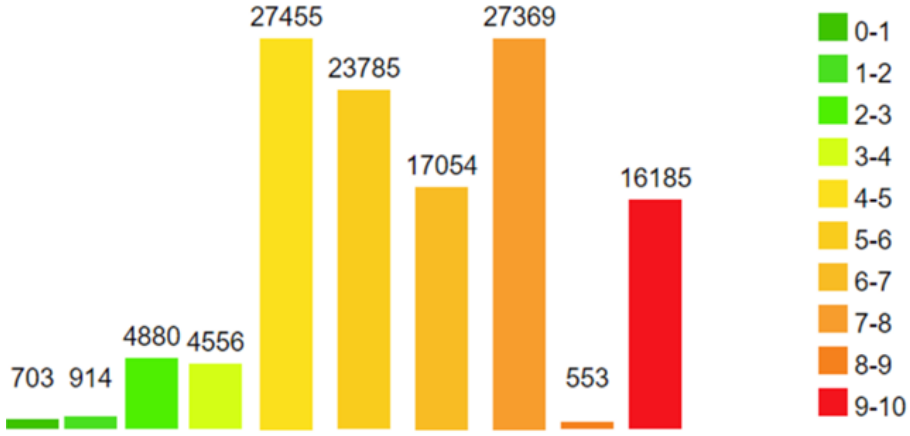

Figure 5 Number of vulnerabilities by CVSS scores.[24]

[21] Taken from: https://www.cvedetails.com/vulnerabilities-by-types.php
[22] https://nvd.nist.gov/vuln/data-feeds
[23] https://csrc.nist.gov/schema/nvd/feed/1.1/nvd_cve_feed_json_1.1.schema
[24] Taken from https://www.cvedetails.com/

2.2.2. Other Vulnerability Databases and Platforms

This section includes a compilation of other vulnerabilities datasets and platforms that are also relevant and that could be considered as an input for the vulnerability assessment process within BIECO.

Table 1 includes the name of each of these databases, a short description and its type of access ("public", "private", or "both" if the dataset has a public and private version), as some of these datasets are not completely public and require a payment.

Table 1 Other vulnerability platforms and datasets.

Name	Description	Access
0 Day Today[25]	A database of exploits and vulnerabilities written for educational purposes. The information is collected from submittals and various mailing lists.	Both
Awesome Threat Detection and Hunting[26]	A curated list of threat detection and hunting resources.	Public
CERIAS Vulnerability Database[27]	A vulnerability database maintained by Purdue University.	Private
CERT-EU[28]	The platform of the Computer Emergency Response Team for the EU institutions. It maintains a list of security advisories and information on product vulnerabilities, threats and incidents and hacking techniques.	Public
China National Vulnerability Database (CNVD)[29]	NVD similar database maintained by the Chinese national computer emergency response team (CERT). It often presents vulnerabilities unavailable in other sources	Public
Chinese national CERT's ICS branch[30]	The website contains a list of ICS and IoT vulnerabilities. These vulnerabilities are found in either CNVD or CNNVD.	Public
Chinese National Vulnerability Database of Information Security (CNNVD)[31]	Second database from China. It usually follows data found in NVD.	Public
CVE Details[32]	It provides an easy-to-use web interface to CVE vulnerability data. Information about vendors, products, versions and statistics about vendors, products and versions of products are available.	Public
DISA STIG Compliance Requirements List[33]	A STIGs "are the configuration standards for DOD [information assurance, or IA] and IA-enabled devices/systems…The STIGs contain technical guidance to 'lock down' information systems/software that might otherwise be vulnerable to a malicious computer attack."	Public

[25] https://0day.today/
[26] https://github.com/0x4D31/awesome-threat-detection
[27] https://www.cerias.purdue.edu/site/about/history/coast/projects/vdb.php
[28] https://cert.europa.eu/cert/newsletter/en/latest_SecurityBulletins_.html
[29] https://www.cnvd.org.cn/
[30] https://www.cert.org.cn/publish/english/indix.html
[31] http://www.cnnvd.org.cn/
[32] https://www.cvedetails.com/
[33] https://www.stigviewer.com/stigs

Draper VDISC Dataset[34]	A dataset that containing the source code of 1.27 million functions mined from open-source software, labelled by static analysis for potential vulnerabilities.	Public
Exploit Database[35]	A CVE compliant archive of public exploits and corresponding vulnerable software, developed for use by penetration testers and vulnerability researchers. It contains a comprehensive collection of exploits gathered through direct submissions, mailing lists, as well as other public sources. The Exploit Database is a repository for exploits and proof-of-concepts rather than advisories, making it a valuable resource for research.	Public
IBM X-Force Exchange[36]	Cloud-based threat platform that enables the research on the latest global security threats, consulting, and collaboration with peers. It contains both human and machine-generated information.	Public
ICS Vulnerability Database	From a Chinese ICS security company Winicssec[37]. Contains data from other sources (NVD, CNVD and CNNVD).	Public
ICS-CERT[38]	The Industrial Control Systems Cyber Emergency Response Team platform. It shares vulnerability information and threat analysis through information products and alerts. It provides vulnerability and malware analysis, onsite support for incident response and forensic analysis.	Public
MISP[39]	It is used to store, share, collaborate on cyber security indicators, malware analysis, and to detect and prevent attacks, against ICT infrastructures. It is used to store, share, collaborate on cyber security indicators, malware analysis, and to detect and prevent attacks, against ICT infrastructures.	Public
National Cyber Security Centre[40]	Located in Finland, it develops and monitors the operational reliability and security of communications networks and services. It provides situational awareness of cyber security.	Public
Netsparker[41]	A fully automatic vulnerability assessment tool that crawls and scans web applications. Vulnerabilities are automatically assigned a severity level to highlight the potential damage and the urgency with which they must be fixed.	Private
NIST Software Assurance Reference Dataset Project[42]	It provides a set of known security flaws in order to allow users to evaluate tools and to test their methods. The dataset includes "wild" (production), "synthetic" (written to test or generated), and "academic" (from students) test cases. The dataset intends to encompass a wide variety of possible vulnerabilities, languages, platforms, and compilers.	Public
Packet Storm[43]	An information security website offering current and historical computer security tools, exploits, and security advisories.	Public

[34] https://osf.io/d45bw/
[35] https://www.exploit-db.com/
[36] https://exchange.xforce.ibmcloud.com/
[37] http://ivd.winicssec.com/
[38] https://www.us-cert.gov/ics/advisories
[39] https://www.misp-project.org/features.html
[40] https://www.kyberturvallisuuskeskus.fi/en/homepage
[41] https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/
[42] https://samate.nist.gov/SARD/index.php
[43] https://packetstormsecurity.com/

SecuriTeam[44]	A security portal containing security information from mailing lists, information channels and tools.	Public
Security Focus[45]	Focuses on a few key areas that are of greatest importance to security: a mailing list for discussion and announcements related to computer security and a vulnerability database.	Public
Snyk Intel Vulnerability Database[46]	An open-source vulnerability database, that also includes additional non-CVE vulnerabilities derived from numerous sources. Numerous vulnerabilities are exposed before they are added to public databases.	Both
Talos[47]	A regular intelligence update from Cisco Talos, highlighting the biggest threats each week and other security news.	Public
The Global Platform MUD File Service[48]	It provides a MUD files database, helping device manufacturers to publish, in a unique location, the MUD file library associated with their products. Publication in the MUD File Service simplifies the access and consumption of MUD files from networks hosting these devices.	Private
The Vulnerability Notes Database (VND)[49]	It provides information about software vulnerabilities. Vulnerability notes include summaries, technical details, remediation information, and lists of affected vendors. Most vulnerability notes are the result of private coordination and disclosure efforts. The CERT/CC Vulnerability Notes Database is run by the CERT Division, which is part of the Software Engineering Institute, a federally funded research and development centre operated by Carnegie Mellon University.	Public
Veracode[50]	An agent-based scan software composition analysis for securing web, mobile and third-party enterprise applications. Veracode provides multiple security analysis technologies on a cloud-based platform, including static analysis, dynamic analysis, mobile application behavioural analysis and software composition analysis.	Private
Vulnerabilities in open-source systems[51]	A project representing a dataset of vulnerabilities in open-source projects, as published in Mining Software Repositories 2018 (MSR) conference.	Public
Vulnerability Assessment Platform[52]	A platform aggregating vulnerability and exploit data from over 130 sources.	Both
Vulnerability Database Catalogue[53]	A catalogue initially of vulnerability databases, underlining differences in identifiers, coverage and scope, size, abstraction and other characteristics. Vulnerability databases are loosely defined as sites that provide vulnerability information, such as advisories, with identifiers.	Both

[44] https://securiteam.com/
[45] https://www.securityfocus.com/vulnerabilities
[46] https://snyk.io/features/vulnerability-database/
[47] https://www.talosintelligence.com/
[48] https://globalplatform.org/iotopia/mud-file-service/
[49] https://www.kb.cert.org/vuls/
[50] https://help.veracode.com/reader/hHHR3gv0wYc2WbCclECf_A/lQYKhC8AvpIbz5_ULOCYMw
[51] https://github.com/AUEB-BALab/VulinOSS
[52] https://vulners.com/
[53] https://www.first.org/global/sigs/vrdx/vdb-catalog

VulnDB[54]	A commercial vulnerability intelligence mechanism developed by Risk-Based Security that provides actionable information about the latest in security vulnerabilities via a SaaS Portal, or a RESTful API. The tool tracks over 2,000 software libraries looking for security issues and it has a direct mapping with CVE and NVD. The client can configure email alerts to receive a notification when a new vulnerability is released and he can ask for guidance on how to mitigate the vulnerability and for product and vendor evaluations.	Private
Vulnerability Database[55]	A database with more than 166000 entries available. The information is updated daily since 1970. Besides technical details, there are additional threat intelligence information like current risk levels and exploit price forecasts provided.	Both
WordPress Vulnerability Database[56]	A database of WordPress vulnerabilities, plugin vulnerabilities and theme vulnerabilities.	Both
Zero Day Initiative[57]	Platform for reporting of 0-day vulnerabilities privately to the affected vendors by the researchers. There is available a list of publicly disclosed vulnerabilities discovered by Zero Day Initiative researchers.	Both

2.3. **Open Web Application Security Project**

The Open Web Application Security Project (OWASP)[58], which is focused on the compilation of software vulnerabilities, developed the **OWASP TOP 10 list**, a standard awareness document for developers and web application security, similar to the list provided by MITRE[59]. Both lists share some common vulnerabilities considered as critical, which should be considered when developing software.

Listing the most critical vulnerabilities will allow us to know their relevance in terms of security, and to prioritize them for their subsequent analysis. In particular, the main risks considered by the OWASP Top 10 list are the following:

1. **Injection:** Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
    - CWE-78 Improper Neutralization of Special Elements Used in an OS Command ('OS Command Injection').
    - CWE-89 SQL Injection.
    - CWE-94 Improper Control of Generation of Code ('Code Injection').
    - CWE-434 Unrestricted Upload of File with Dangerous Type.
2. **Broken Authentication:** Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise

[54] https://vulndb.cyberriskanalytics.com/
[55] https://vuldb.com/
[56] https://wpvulndb.com
[57] https://www.zerodayinitiative.com/advisories/published/
[58] https://owasp.org/www-project-top-ten/
[59] https://www.mitre.org

passwords, keys or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

- CWE-862 Missing Authentication for Critical Function.
- CWE-287 Improper Authentication.
- CWE-798 Use of Hard-coded Credentials.

3. **Sensitive Data Exposure:** Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
    - CWE-200 Exposure of Sensitive Information to an Unauthorized Actor.
4. **XML External Entities**: Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
5. **Broken Access Control:** Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
    - CWE-862 Missing Authorization.
    - CWE-269 Improper Privilege Management.
    - CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal').
6. **Security Misconfiguration:** Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.
    - CWE-522 Insufficiently Protected Credentials.
    - CWE-732 Incorrect Permission Assignment for Critical Resource.
7. **Cross-Site Scripting (XSS)**: XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
    - CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting').
    - CWE-352 Cross-Site Request Forgery (CSRF).
    - CWE-611 Improper Restriction of XML External Entity Reference.
8. **Insecure Deserialization**: Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to

perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

- CWE-502 Deserialization of Untrusted Data.

9. **Using Components with Known Vulnerabilities**: Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defences and enable various attacks and impacts.
    - CWE-416 Use After Free.
    - CWE-787 Out-of-bounds Write.
    - CWE-20 Improper Input validation.
    - CWE-125 Out-of-bounds Read.
    - CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer.
    - CWE-190 Integer Overflow or Wraparound.
    - CWE-476 NULL Pointer Dereference.
    - CWE-400 Uncontrolled Resource Consumption.
10. **Insufficient Logging and Monitoring**: Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

3. Vulnerability Detection Techniques

In general, vulnerabilities require a different identification process than software defects, and can be considered as a subset of faults which occur (and are discovered) much less frequently. They are often not realized by developers during the normal operation of the system, while defects are more easily noticed. For example, some authors report that 21% of files in Mozilla Firefox have defects, while only a 3% have vulnerabilities [20].

To date, several tools and techniques can be used for detecting vulnerabilities in software applications. There are two traditional approaches: static code analysis and dynamic analysis through symbolic execution. In the former, the code is examined for weaknesses without executing it. Besides, it can be considered as a defensive and preventive technique, as it attempts to identify weaknesses in the program source code before its actual use. By contrast, in dynamic analysis the code is executed to check how software will perform in a runtime environment. There are several static and dynamic analysis tools which can be obtained from the market and also some of them are open-source products [21]-[23].

The selection of a proper and efficient technique to detect vulnerabilities is one of the main goals for BIECO. The effectiveness of the tool and in consequence the security of the system will depend on the chosen technique. Therefore, this section presents some contributions published in the literature that are related to the detection of software vulnerabilities and, in particular, that rely on the use of Machine Learning (ML) and artificial intelligence (AI) techniques. The contributions reviewed here can be considered as a subset of the static analysis methods, as they aim at analysing the contents of a source code prior its execution.

Being this a broad field, and to acquire a better overview of the different approaches, they will be divided regarding their detection methodology to be able to create a comparative of them. Particularly, we will focus on three main topics: anomaly detection-based techniques, code pattern recognition, and vulnerability prediction models. For **anomaly detection-based** techniques and **code pattern recognition** we have included studies that analyse program syntax and semantics, unlike **vulnerability prediction model** approaches in which the majority of works do not analyse program syntax and semantics (Figure 6).

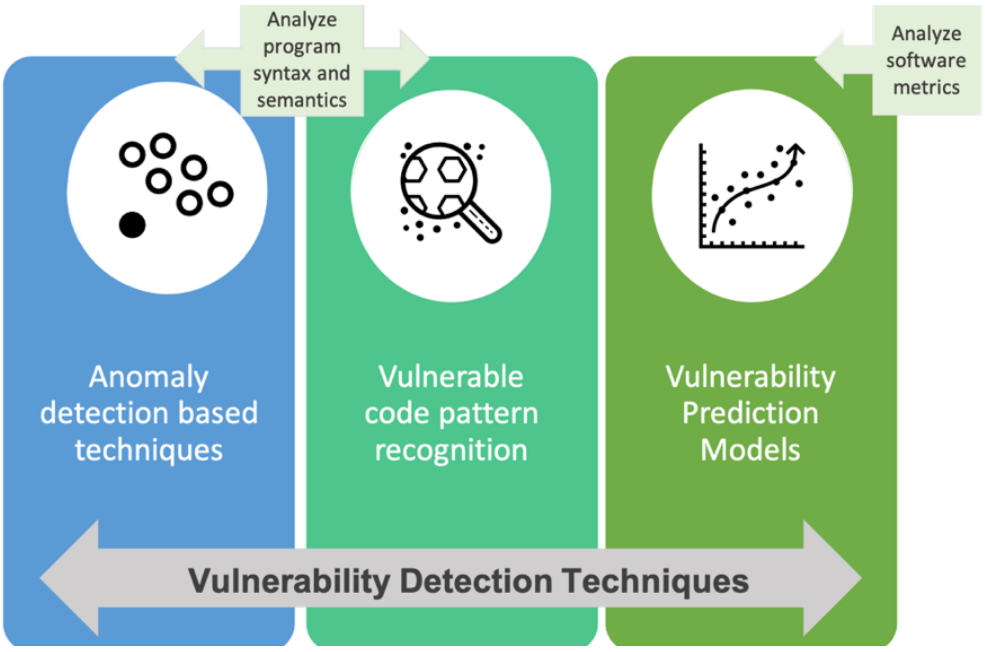

Figure 6 Vulnerability detection techniques classification

3.1. Anomaly Detection-Based Techniques

Anomaly detection addresses the problem of finding unusual events that differ significantly from the majority of the data applying Machine Learning algorithms (mostly unsupervised). These events are often referred to as anomalies, exceptions or outliers [24]. Some of the well-known techniques that can be used to perform anomaly detection in a general-purpose scenario are the density-based such as Local Outlier Factor (LOF) [25] or Density-Based Spatial Clustering of Applications with Noire (DBSCAN) [26], K-Nearest Neighbour [27] or Isolation Forest [28]. Some other examples consider deviation from association rules and frequent itemset [29] or they are more focused on high dimensional spaces such as subspace-based such as Subspace Outlier Degree (SOD) [30] or the Angle-based Outlier Detection (ABOD) [31], among others.

In the context of software security, vulnerabilities can be considered as an "anomaly", as they are undesired events that can appear during the development of a software, and which occurrence should be minimized as far as possible. Under this context, the aforementioned anomaly detection techniques could be used. Here, we pay special focus on **association rules** and **frequent itemset mining techniques**, included in data mining field. In the particular, Engler at el. [32] were among the first authors to point out the need for extracting rules that could be used in bug-finding tools. In their approach, they demonstrated a static analysis technique based on simple function-pair based programming rules that allowed to automatically extract rules from the source code without a priori knowledge of the system, decreasing as well the amount of manual labour required to analyse other systems.

Years later, Li and Zhou [5] presented a general technique called PR-Miner to extract implicit undocumented programming rules and detect violations on large software code written in C, with little effort from their developers. The tool used frequent itemset mining to find sets of functions, variables and data types that tend to appear together. Hence, a later study [33] extended this work taking also ordering into account, which allowed to identify additional defects that remain undetected by PR-Miner. In this new approach, Wasylkowski et al. stated the fact that interacting with objects often requires following a protocol which is not always documented, and its violation can lead to defects. To automatically extract valid sequences of method calls, the authors proposed to use frequent itemset mining on closed patterns taking code examples which are then used to detect anomalies. Furthermore, they introduce a defect indicator, implemented by the tool JADET (Java Anomaly Detector), which ranks the identified anomalies based on several factors.

Working on the anomaly detection mechanism of JADET, Gruska et al. [6] improved the technique adding cross-project anomaly detection. In this approach authors introduced a method based on functions calls and program structure, which allowed to analyse only selected parts of a source code. With this, a lightweight parser is achieved which is effective enough to mine usage rules from large bodies of almost arbitrary source code. Furthermore, this language-independent parser is applicable for analysing programs written in several languages that follow a similar syntax, like C, C++, Java or PHP. Compared with JADET, this new approach based on association rules offered a set of frequent temporal properties more expressive, and an improvement on the ranking system.

The mentioned previous works [5]-[6] used frequent itemset mining approach to mine frequent API patterns, these methods have some limitations especially when multiple APIs are involved across different procedures. For that reason, Acharya et al. [34] presented an approach to automatically extract frequent partial-orders from API client code. They employ inter procedural analysis to discover rules across function boundaries. However, their approach is limited to mining function call ordering rules. In contrast, Chang et al. [35] created an approach general enough to discover function call ordering rules, as well as their preconditions and postconditions ordering rules. Chang et al. emphasize the importance of neglected conditions as a difficult-to-find class software defect. In this approach, the authors integrate static program analysis and

advanced data mining techniques to reveal implicit conditional rules as well as rule violations that indicate neglected conditions. Thus, the user indicates just a few restrictions on the context of the rules that have to be searched, rather than specific rule templates. To do so, rules are modelled as graph minors of Enhanced Procedure Dependence Graphs (EPDGs).

An approach related to the one developed by Chang et al., is Alattin [36]. This novel approach tried to reduce false positives produced by the frequent sub-graph mining approaches developed in previous works. To this end, they introduced a new mining algorithm and technique, ImMiner, which extracts alternative patterns and classifies them into two categories: balanced, where all patterns are frequent, and imbalanced, where some of them are infrequent. In both ([35], [36]), there are targeted neglected conditions, but Chang et al.'s approach cannot mine infrequent alternatives and it is heavily limited by its underlying graph mining algorithm which is known to suffer from scalability issues. In this context, ImMiner is much more scalable due to its imbalance frequent itemset mining algorithm.

Another form of analysis that can be complementary to Chang et al.'s approach is DynaMine [37]. This tool presented by Livshits and Zimmermann proposes an automatic way to extract likely error patterns combining revision history information with dynamic analysis. For that purpose, the tool analyses source code check-ins to find highly correlated method calls and common bugs fixes. DynaMine evaluates incremental changes, obtaining more precise results. However, this approach requires extensive history of software revisions in repositories to be effective and, as with PR-Miner, it may suffer from issues of a high number of false positives since rule elements are not necessarily associated with program dependencies.

The aforementioned studies are just some examples of anomaly detection techniques carried on in the last years applied to the context of software vulnerabilities. Even though some of the results exposed reveal good results in terms of detection, there are still high false positive rates since they do not take into account rare events. In addition, these types of techniques are focused only on the detection process, and they do not provide the type of the vulnerability or even its location.

3.2. **Vulnerable Code Pattern Recognition**

Vulnerable code pattern recognition methods use Machine Learning algorithms (mostly supervised) to identify in an automatic way patterns of vulnerable code segments. As with the anomaly detection approach, this category of works requires the analysis and extraction of different features from the source code, but with the difference that it is focused on defining models and patterns of vulnerable code segments, instead of obtaining a model of the normal behaviour of the software.

In order to extract the required software features, different techniques can be used such as conventional code parsers, static data and control flow analysis, dynamic analysis or text mining, among others. Taking the resulting features as an input, different approaches have been proposed to statistically detect vulnerable code patterns and, in particular, concepts from the area of software verification have been successfully adapted for tracking vulnerabilities. However, the biggest challenge with these approaches is to avoid as much as possible burdensome manual audits which require considerable time and experience.

Yamaguchi et al. [38] addressed particularly this issue and proposed a method to assist a security analyst during source code audit. The purpose of this approach was to make a manual audit more effective by guiding the search for vulnerabilities rather than looking for an automated solution. To this end, they based their study on the concept of "vulnerability extrapolation" [39], which focuses on the possibility of discovering vulnerabilities by looking for functions that share the same code structure, since they are usually linked to the same faulty programming patterns. The

method suggested by Yamaguchi et al. extracts an Abstract Syntax Tree (AST) from the source code and determines its structural patterns, in such a way that each function in the code can be described as a mixture of the obtained patterns. These patterns contain subtrees in which each node corresponds to types, functions and syntactic constructions of the codebase. Thanks to this representation, it is possible to break down a known vulnerability and suggest code with similar properties to the analyst.

The main limitation of the previous approach was that even though the AST can be useful when it comes to transform simple code and identify code with similar semantics, it is not suited for more advanced analysis. For this reason, and continuing this direction of research, Yamaguchi et al. [40] presented a method to effectively mine large amounts of source code and find vulnerabilities. In this approach, authors combined classic concepts of program analysis with recent developments in the field of graph mining. They introduced a new graph representation named *code property graph* that combined in the same data structure properties of ASTs, Control Flow Graphs (CFGs)[60] and Program Dependence Graphs (PDGs)[61]. With this approach, they were able to create templates for vulnerabilities using graph traversal representations in such a way that could identify buffer overflows, integer overflows, format string vulnerabilities and memory disclosures, among others. Code property graphs and graph traversals are suitable to find common types of vulnerabilities but more importantly, they can be well adapted to identify vulnerabilities specific to a code base.

Using the code property graph representation, in a latter work Yamaguchi et al. [41] presented a method for automatically deducing search patterns for taint-style vulnerabilities from C source code. This definition of vulnerabilities has its origin in taint analysis, a technique for tracing the propagation of data though a program. The method proposed automatically identifies source sink systems in a code base given, analyses their data flow and generates search patterns in the form of graph transversals that enable uncovering vulnerabilities in the data flow to the sink. In order to generate search patterns, the approach combines techniques from unsupervised machine learning (i.e., clustering techniques) and static program analysis, using an extension of the platform Joern[62] for the generation of graph transversals. It should be noted that although this automatic search significantly speeds up the analysis in large code bases, the approach still requires a considerable amount of manual auditing and analysis work.

A more general approach is the one given by Scandariato et al. [7] using text analysis and machine learning techniques, such us Naïve Bayes and Random Forest, in order to predict vulnerable software components. The approach is based on text mining the source code mainly with bag-of-words techniques, in which a software component (source code file) is seen as a series of terms with associated frequencies. These terms are the features that are used to predict whether each component is likely to contain vulnerabilities. Hence, the set of features used for modelling is not fixed but rather depends on the vocabulary used by the developers.

A related approach was proposed by Pang et al. [42] who investigated the possibility of predicting vulnerable software components employing a hybrid technique combining *N*-gram text mining and feature selection techniques. The *N*-gram text mining technique is an advanced version of bag-of-words, handling not only single tokens but also sequences of tokens. However, dealing with high dimensionality -curse of dimensionality [43]- can affect the performance and effectiveness of training datasets and therefore classification models. In order to solve this challenge, they proposed to select the most relevant and important attributes from a dataset, reducing the space of features.

[60] CFG are graphical representations of the control flow of a software during its execution.
[61] Program representation that includes the data and control dependences for each possible operation.
[62] Joern, Open-Source Code Querying Engine: https://joern.io/

Most of the reviewed approaches are focused on locating vulnerable code only at software component or file levels. However, from a web developers' point of view, input validation and input sanitization are also essential secure coding techniques that can be used to protect programs from common vulnerabilities. Under this premise, Shar and Tan [44] proposed a set of input sanitization code attributes that can be statically collected and based on Control Flow Graphs (CFG), and which can be used to predict if a certain program statement could be vulnerable to SQL-injection (SQLI) or cross-site scripting (XSS) attacks. These static code attributes are used to characterize input sanitization code patterns, which are then analysed in order to check if the associated program statements are vulnerable to SQLI or XSS. As a result, they developed an automated data collection tool called PhpMinerI[63] to extract the data of the proposed input sanitization code attributes from PHP programs, in order to build vulnerability prediction models based on supervised learning algorithms by means of manually tag the data with vulnerable labels. This new tool was evaluated and compared on some open-source web applications, i.e., a static analysis tool named Pixy [45]. On average, Pixy discovered more vulnerabilities, but also produced much more false positives than PhpMinerI. As expected, even though the proposed static attributes are a good predictor, their predictive capability is limited as it depends on the classification of the input validation and sanitization code patterns.

Being aware of this limitation, Shar et al. [46] presented a more extensive empirical study proposing a hybrid analysis. Their idea was to use static analysis for the classification of nodes that have unambiguous security related purpose, and avoid the lack of precision that it provides by the use of dynamic analysis. In order to predict vulnerabilities, they proposed the use of both supervised machine learning models such as Logistic Regression and Multi-Layer Perceptron, and unsupervised machine learning models, such as the k-means algorithm (for the presence or absence of labelled training data, respectively).

In a latter work, the study was extended [47] adding new contributions and changes. The authors proposed to use static slicing[64] and dynamic execution techniques that mine data dependency and control dependency information. Moreover, they used a semi-unsupervised approach, unexplored in this domain till the date, along the supervised approach, to predict vulnerabilities from a new set of code attributes. These attributes are called Input Validation and Sanitization (IVS) attributes from which vulnerability-built predictions are fine-grained, accurate and scalable. The authors also extended the support to predict vulnerabilities to SQLi and XSS by adding the detection of Remote Code Execution (RCE) and File Inclusion (FI) web vulnerabilities, implementing a modification of their previous tool.

The aforementioned works, with the exception of Shar et al. [46], [47], are based on static analysis techniques for the detection of vulnerabilities. In addition, these techniques focus on the analysis of the source code. Grieco et al. [8] presented the first large scale study on vulnerability discovery for binary code. The objective of this work was to create a scalable Machine Learning approach that used lightweight hybrid features of static and dynamic analysis techniques to predict if a binary program is likely to contain an easily exploitable memory corruption vulnerability. They also developed and implemented VDiscover, a tool based on Machine Learning techniques such as logistic regression, stochastic gradient descent or random forest, to predict vulnerabilities in test cases. This study increases the possibility to find a greater number of vulnerabilities at operating system scale.

In the last years, deep learning-based techniques have had a big success in many domains, which has led their application study to a new trend in the field of software vulnerability detection. The first systematic framework for using deep learning to detect vulnerabilities was the one presented

[63] http://aharlwinkhin.com/phpminer.html
[64] Technique that allows to represent a simplified version of source code, ensuring that the effects of the software on a certain variable is preserved.

by Zhen Li et al. [48]. The framework, called SySeVR, was focused on extracting program representations that can accommodate syntax and semantic information that are relevant for the vulnerabilities such us function call and pointer usage. Zhen Li et al. [49] also propose the use of deep learning to detect vulnerabilities at the slice level[65]. Since then, deep learning was used to detect vulnerabilities at a coarser granularity such as in a function level. In this approach, authors focus on multiple lines of code that are semantically related to each other in terms of data or control dependency. To this end, they design and implement a deep learning-based vulnerability detection system called VulDeePecker.

Each of the approaches previously introduced present a different perspective and tools for the detection of vulnerabilities. In this case, the goal is not detecting anomalies in the source code, but rather to define vulnerability patterns for their detection. Although vulnerable code pattern recognition is a promising approach for the detection of vulnerabilities, especially in comparison with the previous one, it still has some limitations. As with anomaly detection approaches, the studies presented do not identify the type of the vulnerability and, while it is possible to detect the location of a the vulnerability within the source code, it is still not fine-grained.

3.3. Vulnerability Prediction Models

Prediction techniques rely on Machine Learning models (mostly supervised) and Artificial Intelligence techniques in order to determine the location of the vulnerabilities within a source code. In other words, these techniques determine which software components are most likely to contain a certain vulnerability. These particular models are commonly referred as Vulnerability Prediction Models (VPM) [50]. Yet, there are many other prediction models that can be used to forecast different aspects of the nature of a vulnerability, which will be presented in detail in Section 4.

As we mentioned at the beginning of Section 3, vulnerabilities occur less frequently than defects, and in consequence, VPMs have to deal typically with highly unbalanced datasets. There are still a number of open issues in the construction of effective VPM: i) choice of granularity: model granularity, the selection of a unit to collect data and make predictions for binary, source file, class, and function/method; ii) statistical learner choice: it can be considered the most important step. For a good introduction of all the techniques see e.g. [51], or for more advanced details see [52]; iii) classification performance: in this sense, [20] and others [53] have suggested specific precision and recall values and they can be considered as a good starting point.

The first work related to VPM was published in 2007 by Neuhaus et al [54]. The authors proposed to fit a ML model by means of incorporating the imports and the function call contained in a file as independent variables. This approach is based on the intuition that vulnerable files are likely to share similar sets of imports and function calls which could be used to identify them. The idea is in accordance with the work performed by Schroter et al. [55], which aims at predicting defects. The authors begin this study by performing a correlation analysis of import/function calls and vulnerabilities on a dataset of Mozilla Firefox vulnerabilities to validate the intuition. In their evaluation, the authors perform random splitting and experiment two approaches, one using the includes of the file as features while the other use its function calls. To build the models, the authors opt for the Support Vector Machine (SVM) algorithm.

There is extensive literature for VPMs approaches that use software engineering metrics computed from the source code to build their models. The starting point of all studies that use complexity metrics is the study carried out by Shin et al. [56]. Some years later, in addition to

[65] i.e., Multiple lines of code that are semantically related to each other in terms of e.g., data dependency or control dependency.

complexity metrics, code churn and develop activity metrics are evaluated as indicators of software vulnerabilities [57]. Other studies use code complexity, code churn, and other static alerts to predict attack-prone or vulnerable components, [58] and [57] among others.

Additionally, Morrison et al. [59] replicate the study of Zimmermann et al. [9] who studied typical code metrics as well as dependency metrics and evaluated their correlation with post-release vulnerabilities in Windows Vista binaries. Besides, Morrison et al. [59] investigated the performance of the models using the same metrics on finer levels of granularity (source file level). They used a total of 29 metrics classified as follows: churn metrics, complexity metrics, dependency metrics and legacy metrics and size metrics.

In a more recent study carried out by Younis et al. [60] was described the attributes of code that contain vulnerabilities that are more likely to be exploitable. To this end the authors gather 183 vulnerabilities from the Linux kernel and Apache HTTPD web server projects, which includes 82 exploitable vulnerabilities.

Recently, Bilgin et al. [10] examined how to predict software vulnerabilities from source code by employing ML techniques prior to their release. To this end, they developed a source code representation that enables us to perform intelligent analysis on the Abstract Syntax Tree (AST) form of source code and then investigate whether ML can distinguish vulnerable and non-vulnerable code fragments.

After reviewing several works for detecting security vulnerabilities, and among the different approaches described, VPMs seem to be a good starting point for the development of a vulnerability detection tool in BIECO. To this end, and according to various experiments provided by the researchers, the goal is to build a VPM based on advanced software metrics, determining which ones allow to obtain more accurate results on the vulnerability detection process.

3.4. Privacy-Preserving Strategies

Source code is considered in general a sensitive asset, as it is part of the intellectual property of the companies that develop software and, in consequence, it should be protected adequately in order to avoid leakages that could derive in an economical and reputational impact.

The tools that will be developed in BIECO for the vulnerability assessment will deal directly with the analysis of source code. Due to its sensitiveness, the project will explore the feasibility of using of privacy preserving mechanisms as means of protecting the confidentiality of the source code during the vulnerability assessment process.

As a result, in this section a review of existing privacy-preserving mechanisms to protect sensitive data and intellectual property is provided. These mechanisms could include differential privacy, secure multi-party computation and Federated Learning, between others, although we will pay special attention to Federated Learning technique as an interesting mechanism to further explore its usage within BIECO.

Current and emerging techniques for privacy and preservation can be found in [61]. They show a general visual overview of the privacy-preserving field in Figure 7.

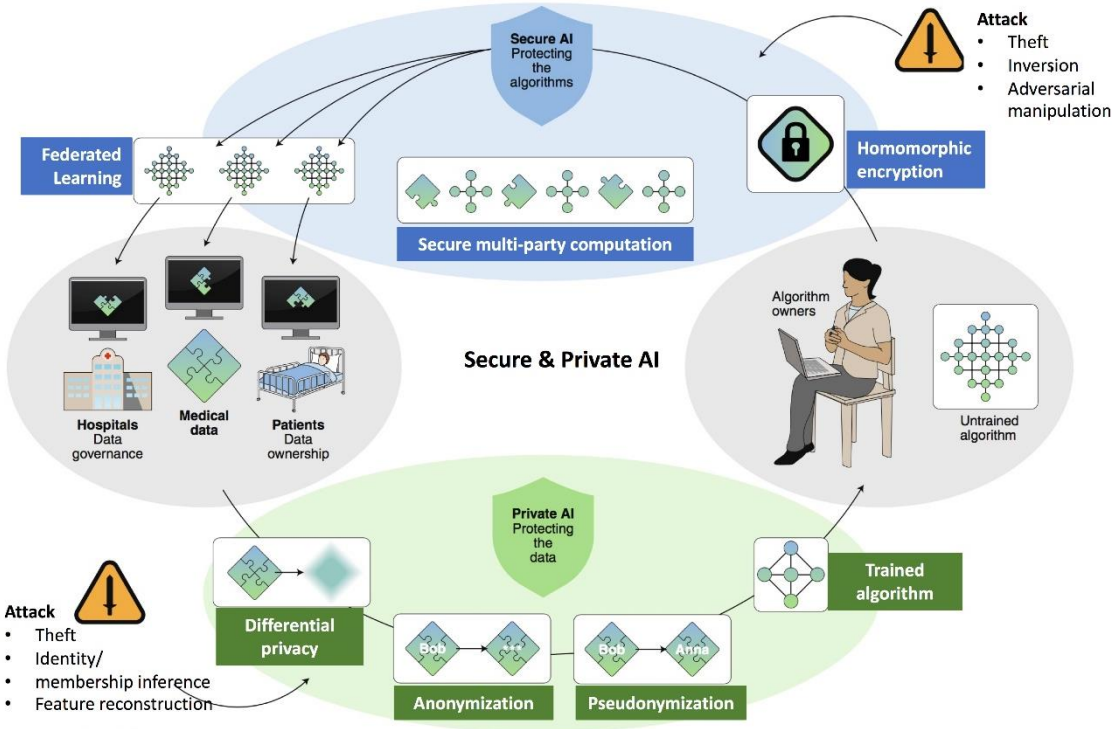

Figure 7 Overview of the relationship between data, algorithms, actors, and techniques in the field of secure and private AI.[66]

On the one hand, secure AI includes methods concerned with safeguarding algorithms and, on the other hand, private AI includes methods for systems allowing data processing without revealing the data itself. A summary of the terms presented in the previous figure is shown below:

- **Differential privacy:** modification of a dataset to obfuscate individual information as a means of ensuring that the risk incurred by participating in a dataset is only marginally greater than the risk of not included in it. It can also be applied to algorithms.
- **Anonymization:** removal or transformation of personally identifiable information from a dataset in such a way that the observable data cannot be used to breach user's privacy. It is an irreversible technique.
- **Pseudonymization:** replacement of sensitive data with realistic synthetic data that cannot be attributed to a specific individual without additional information which, according to GDPR, is to be "kept separately and subject to technical and organization measures to ensure non-attribution to an identified or identifiable person". It is an optional reversible technique.
- **Secure multiparty computation:** collection of techniques and protocols enabling two or more parties to split up data among them to perform joint computations in a way that prevents any single party from gaining knowledge of the data but preserving the computational result.
- **Homomorphic encryption:** cryptographic technique that preserves the ability to perform mathematical operations on data as if it was unencrypted (in plain text).

[66] Taken from [62]

- **Federated learning:** Machine Learning system relying on distributing the algorithm to where the data is instead of gathering the data where the algorithm is.

Next, we will focus on federated learning technique, which belongs to a class of decentralized/distributed systems.

3.4.1. Federated Learning (FL)

It is well known that Machine Learning techniques require centralizing the training data on one machine or datacentre for building accurate and robust models. However, these techniques have a disadvantage from the security and privacy perspective, that is, the coupling of the training model with the need for direct access to the raw training data. Because of the increasing concern in data privacy (e.g., General Data Protection Regulation, GDPR), restrict access to the data is still a major challenge.

Recently, a new paradigm was proposed by McMahan et al. [62]. They investigate a learning technique that lets users collectively take the benefits of shared models trained from data, without the need to centrally store it. They coined the term Federated Learning (FL) since the learning task is solved by a loose federation of participating devices (clients) which are coordinated by a central server. Briefly, the idea is that each client (e.g., mobile devices) has a local training dataset which never is transferred to the server or other clients. Instead, each client computes an update to the current global model maintained by the *server*, and only this update is communicated. This is an example of application of the principle of *focused collection* [63].

Federated Learning is considered as an iterative process wherein each iteration the central Machine Learning model is improved. FL implementations can be generalized into the following three steps: (1) The central pre-trained ML model (i.e., global model) and its initial parameters are initiated and then the global ML model is shared with all the clients in the FL environment. (2) After sharing the initial ML model and parameters with all clients, the initial ML model at the client level (called local ML models) is trained with individuals training data. (3) Local models are trained at the client level and updates are sent to the central server in order to aggregate and train the global ML model. The global model is updated and the improved model is shared among the individual clients for the next iteration. Federated Learning is in a continuous iterative learning process that repeats the training steps of 2 and 3 above to keep the global Machine Learning model updated across all the clients. A scheme of the architecture and the training approach can be seen in Figure 8 (taken from [64]).

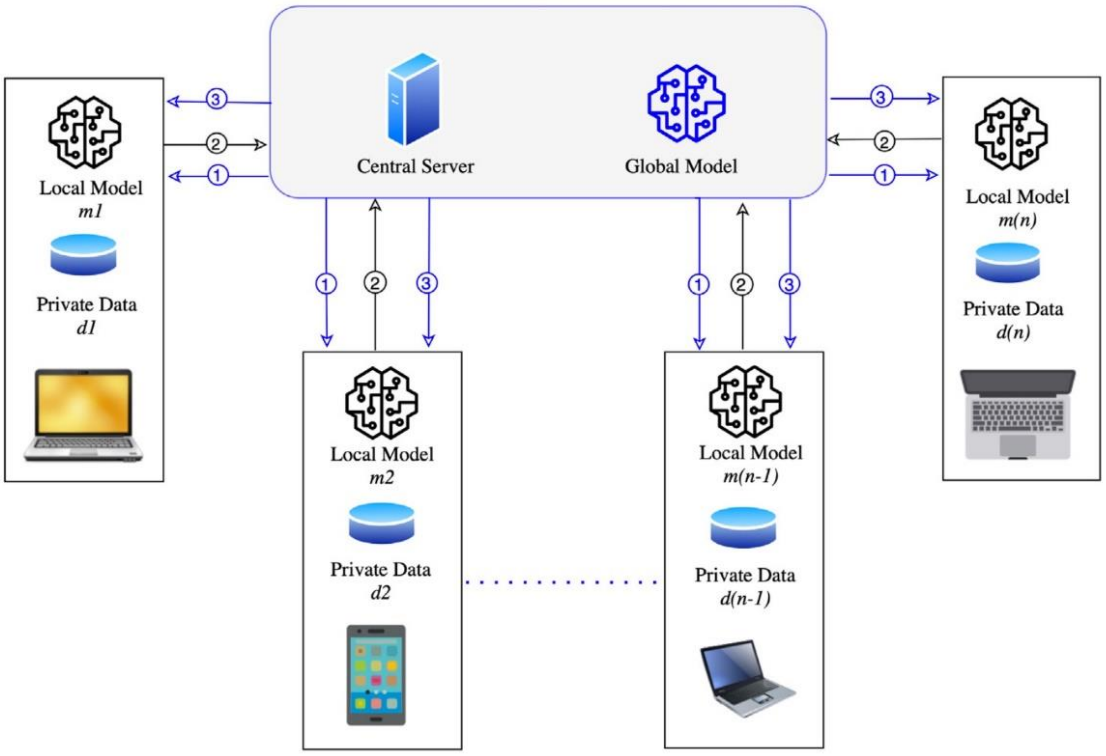

Figure 8 Federated Learning process flow.[67]

Federated Learning is a recent technique which is still under development and uses different approaches in order to apply it in practice. Some of the most current and common approaches are, for example, included in data partition category with Horizontal Federated Learning, Vertical Federated Learning and Transfer Federated Learning. In the aggregation/optimization algorithms classification it should be highlighted the Federated Averaging which is Google´s implementation of FL, or FedMA (Federated Learning with Matched Averaging) which is useful for constructing a shared model for convolutional neural networks (CNNs) and LSTM based in FL environments, between others.

[67] Taken from [65]

4. Vulnerability Forecasting Techniques

As it has been explained in Section 3.3, prediction is a more general term than the topic related to this section. In particular, the term forecasting refers to the process of making predictions of the future as accurately as possible, based on past and present data available or even knowledge of any future events that might impact the forecasts.

In the early stages of a forecasting analysis, decisions need to be made about what should be forecasted, taking into account that some things can be easier to forecast than others. In particular, the predictability of an event or quantity depends on: i) how well understand the factors that affect the event, ii) how much data are available, and iii) whether the forecast can affect the thing we are trying to forecast. Depending on the study field, one could be interested in short-term, medium-term or long-term forecasts. Thus, it is also relevant to consider forecasting horizon, as a different type of technique must be applied depending on how the forecasting horizon is set.

Moreover, in forecasting it is important to be able to capture the relationships that exist in the historical data, but do not replicate past events that will not occur again; in other words, to distinguish between random fluctuations in the past data that should be not taken into account, and a genuine pattern that should be modelled and extrapolated. To this end, a variety of methods can be used: from the simplest ones, such as the usage of the most recent observation as a forecast (named as naïve method) or more complex ones, such as neural networks.

There are many forecasting techniques that have been described in the literature, often developed within specific fields for specific goals. Each technique has its own accuracies, properties and assumptions that are relevant for selecting the method to be used [65].

Forecasting future data can be addressed based on time series domain, where data are collected at regular intervals over time (e.g., hourly, daily, monthly, annually). Time series analysis try to find a model that describes the pattern of data with natural temporal ordering and they include models such as moving average (MA), autoregressive models (AR), the more general Autoregressive Moving Average (ARMA) [66]-[68], Autoregressive Integrated Moving Average models (ARIMA) [69]-[72] and Seasonal Autoregressive Integrated Moving Average (SARIMA) [75], all of them based on autoregressive moving averages. Focusing on a family models based on autoregressive conditional heteroskedasticity, Autoregressive Conditional Heteroskedasticity (ARCH) or Generalized Autoregressive Conditional Heteroskedasticity (GARCH) [73] can be used. Another option could be models such as Threshold Autoregressive (TAR) or Smooth Threshold Autoregressive (STAR) [74], nonlinear models of mean and non-linear variances, as well as others such as the Hamilton mode switch [75] and the Theta method [76].

Other of the most well-known techniques for forecasting analysis are the exponential smoothing models such as single exponential smoothing, Holt´s linear method, or Holt´s winter method [77]-[80].

In order to estimate how the sequence of observations will continue into the future, it can also be addressed by predictor variables based on an explanatory model [81][82]. This type of model incorporates information about other variables instead of only historical values of the variable to be forecast. There is also another type of models, called mixed models, which combines the features of explanatory models and time series models. They are known as dynamic regression models, longitudinal models, panel data models, etc. [78][83].

The appropriate selection of the forecasting method depend on what resources and data are available, the accuracy of the competing models and the way in which the forecasting model is to be used.

4.1. Forecasting Techniques Regarding the Exploitability of a Vulnerability

Vulnerability forecasting refers to the estimation of certain information such as the number, type or time of occurrence of an event (i.e., software vulnerabilities) in the future. As mentioned earlier, the number of software vulnerabilities disclosed every year is staging and, consequently, the risks for system security officers are also increasing. To date, substantial research has been dedicated to techniques that analyse source code and detect security vulnerabilities. However, only limited research has focused on forecasting security vulnerabilities that are detected and reported after the release of a software.

One interesting and widely studied approach is estimating the probability that a vulnerability will be exploited. Some contributions have been described in the literature such as Jacob et al. [14] and Bhatt et al. [84]. Unlike these research works which study if a Common Vulnerability and Exposure (CVE) is used in an exploit, an important question growing between researchers is forecasting when an exploit might appear. This issue is especially important for system administrators, because of their need to devote scarce resources to take corrective actions when a new vulnerability appears [15]. When vulnerability information is disclosed, there is an attacker´s tendency to exploit vulnerabilities, and consequently, vendors' tendency to release patches. Because patching is expensive, many vulnerabilities go unpatched because of the limited resources to tackle with large patching tasks [85]. So, it is crucial to prioritize patching based on the results of prediction models about when a vulnerability will be exploited.

It should be highlighted that there are two types of exploits: i) Proof of concepts (PoC) exploits that are one where someone generates a sample exploit code to demonstrate vulnerability in a controlled environment and ii) real world exploits that were used in real world attacks. In particular, [86], [87] and [88] focus on predicting PoC exploits (published exploits) rather than real world exploits. Sabottke et al. [89] developed methods to predict both PoC and real-world exploits. They were the first to demonstrate that prediction accuracy could be improved by using (one year of) Twitter data.

Recently, much more efforts have been made in order to studied real world exploits [90] where, in addition to National Vulnerability Database (NVD) and Exploit Database Archive (EDB), data from the Zero Day Initiative (ZDI)[68] and dark web and deep web posts are used.

On the other hand, even though Common Vulnerability Scoring System (CVSS) has become an industry standard for assessing fundamental characteristics of vulnerabilities, some limitations have been identified, such as the absence of an authoritative entity to update the metric values and lack of data to inform the score. Following this idea, Haipeng et al. [15] described a method to predict when a vulnerability will be exploited based on CVE ID and Twitter discussion data, without the need for Common Vulnerability Scoring System (CVSS) scores. In addition, Jacob et al. [14] proposed a first open, data-driven threat scoring system, called Exploit Prediction Scoring System (EPSS), for predicting the probability that a vulnerability will be exploited within the 12 months following public disclosure.

Taking into account the increasing number of vulnerabilities and their consequently need for prioritizing them, the goal in BIECO is to develop a tool that allows to predict when a certain vulnerability will be exploited. Following the ideas proposed by Sabottke et al. [89], Haipeng et al. [15], Jacob et al. [14], among others, different features or variables will be tested as input for training the Machine Learning model and, also, a period time window will be established and adjusted in order to provide accurate forecasting results.

[68] http://www.zerodayinitiative.com/

4.2. Other Vulnerability Forecasting Techniques

Between all of the challenges that are still open or not completely studied in forecasting vulnerabilities, predictions of the numbers of vulnerabilities in the next period of time are an important input for several managerial decisions [91], [11], [92]. Some well-known approaches are described in order to achieve this goal.

From a time series perspective, Roumani et al. [11] implemented Autoregressive Integrated Moving Average (ARIMA) and exponential smoothing models for predicting the number of vulnerabilities, but also, other authors applied regression techniques [56] [12] or reliability growth models [93].

However, none of the research takes into account the rareness of the vulnerability occurrence and high volatility, non-stationarity and seasonality. To solve this gap in the literature, recently, Yasasin et al. [94] concentrate on forecasting the number of vulnerabilities implementing a multiple forecasting approach, in which they compare several methodologies and evaluate their performance in terms of forecasting accuracy. On the one hand, they applied exponential smoothing model and Box-Jenkins models (such as ARIMA). On the other hand, they apply zero-inflated time series which is especially suitable in this context because there are multiple periods of zero values [13], particularly, Croston´s methodology [95] and Neural Network based approach [96], [97]. The conclusion is that the Croston's method and ARIMA are recommended for forecasting IT vulnerabilities since they achieved low forecasting errors for the considered software, for both metrics. Exponential smoothing methods are not recommended because of their susceptibility to the time series' nature of IT vulnerabilities. The feed forward neural network with a single hidden layer used achieved also poorer prediction accuracy.

To achieve better forecasting results, it could be interesting to explore in depth some properties of security vulnerabilities: vulnerabilities are rare events so it is common that no vulnerabilities are reported, but also, there are some periods in which high numbers of vulnerabilities are reported. In this regard, vulnerabilities found by software engineering, and closed without publicly announced could be taken into account, as well as the relationship between monetary incentives such as third-party bug bounty platforms [98] and the number of vulnerabilities reported could be studied harder, leading to a decrease in zero values.

Forecasting zero-day vulnerabilities is also an important issue in software security. David Last described in [92] three vulnerability discovery forecast model suites: i) Composite Regression Models that use linear and quadratic regression to fit trendlines of the training period, ii) ML over Cumulative Vulnerabilities Models in which cumulative data for vulnerabilities to the beginning of the forecast period are used for training and iii) Machine Learning over Monthly Vulnerabilities Models, in which the number of vulnerabilities discovered each month in the training period is used for training instead the cumulative vulnerabilities. Because it is impossible to say which of these forecast models will be the most accurate, a consensus model influenced by all the components model is created.

One of the latest achievements in the time series prediction, presented in M4 Competition[69] [99] show that in the last few years a significant advancement in accuracy of predicting time series has been observed, mostly due to the use of ML [100]. To our best knowledge they have not been used for the vulnerabilities forecasting but, based on their accuracy, it might be very promising to use them in this field.

One of the methods that presents significant progress in state of the art of time series forecasting is hybrid ES-RNN method by Slawek Smyl (from Uber Technologies; the winner of M4

[69] The M Competitions (also known as Makridakis Competitions) are led by prof. Spyros Makridakis from the University of Nicosia and are the most comprehensive studies of time series forecasting methods.

Competition), which is a hybrid statistical (Exponential Smoothing) and Machine Learning (Recurrent Neural Networks) method [101]. In the M4 competition, it outperforms the benchmark method for almost 10% of accuracy on 100 000 of time series, which is a really significant improvement to the previous state of the art. The second method, the one provided by Montero-Manso et al. [102] from the University of A Coruña and Monash University, uses a novel combination (ensemble) of one machine learning model and several statistical ones.

One approach which introduces novel concepts of Machine Learning for time series forecasting and has already outperformed the aforementioned methods, was the one presented as N-BEATS (Neural basis expansion analysis for interpretable time series forecasting) [103]. Another interesting venue are the methods based on recurrent embedding kernels which predict based on similarities between time series [104]. This kind of methods might give promising results for vulnerabilities predictions as they are based on complex pattern learning. The similarity in this case is learned by the neural network during the training as opposed to relying on clustering methods with fixed metrics. Some other interesting methods include those based on Echo State Networks with deep reservoir [105] and SFM (State-Frequency Memory) which combines LSTM networks with wavelet time series analysis [106].

Finally, there are two very successful ML methods not originally targeted at time series analysis: Differentiable Neural Computer [107] and Tsetlin Machine [108]. According to our knowledge, they have not been used yet for time series forecasting and vulnerabilities forecasting, but we expect them to achieve good performance after some improvements. It should be noted that these methods are generally far more effective than any others previously used for time series forecasting, including vulnerabilities forecasting. Another key approach that can be undertaken is focused on the novel architectures of the neural network's models, like the Transformer architecture [109] and Capsule Networks [110]. These architectures exhibited remarkable results in NLP and image recognition tasks, outperforming previous models by a solid margin. The usage of these architectures for vulnerabilities prediction is not researched fully yet, but nonetheless we consider it a very promising approach.

5. Vulnerability Propagation Techniques

A software system can be composed by several entities or components that are related to each other. A change in one of them, such as adding new features or a patch, can affect their related entities, creating unexpected effects and causing inconsistencies with other parts of the original software.

This interdependency can also impact the software security, as a vulnerability located in a specific area of a source code could be propagated and escalated through the supply chain via multiple paths, making a fragment of the code with no vulnerability being part of vulnerable path, and leading to a different system security risk.

In order to deal with these issues, in BIECO we want to create a tool with which it is possible to detect the propagation of vulnerabilities across interconnected ICT systems and modules. As a first step, in this task we will focus on vulnerability propagation through the code, doing a review of the state-of-art of different techniques to be able to evaluate the most prone one. As a complement of vulnerability code propagation, MUD standard is introduced, where the expected behaviour of an ITC component is defined, helping to avoid or mitigate future potential security attacks.

The next section presents a review of several approaches published in the literature to evaluate how a vulnerability can be propagated across the software supply chain.

5.1. Review on Vulnerability Propagation Methods

Program dependency relations, such as functions and variables, have been the goal of many researchers in order to evaluate how a change in one source code entity propagates to other entities [111]. In the last decades, there have been many investigations for identifying the effects of those changes using Change Impact Analysis (CIA) [112]. CIA is a technique that can be employed to predict the impact on a change in software, as well as estimating what needs to be modified to accomplish a certain change and their cost. Depending on the level of abstraction between elements, CIA methods can be divided into those that are based on the traceability examination (traceability-based CIA) and those that are on dependence relationships (dependence-based CIA). In the former, the analysis tries to trace the existing dependencies between elements from different levels of abstraction, while the latter analyses the dependencies between program entities from the same level.

These dependencies above mentioned can lead to the introduction of known (or unknown) security vulnerabilities in software systems by third-party components, as for example in the case of integrating open source libraries with vulnerable code. Several tools have been developed in order to detect if a third-party component with known security vulnerability has been used. This is the case of Cadariu et al. [3], who develop a tool-based process to track known vulnerabilities in software systems named Vulnerability Alert Service (VAS). In this approach, inputs (software project and vulnerability disclosures) are destined for the vulnerability checker (OWASP Dependency Check) which extracts dependency data, recognizes them and matches them with known vulnerabilities. Another perspective is the one introduced by Plate et al. [113]. This approach presents a dynamic analysis technique to automatically determine if the changes made by a security patch in a library are propagated to their source code, in such a way that they can identify where they may have a vulnerability when using the library from third parties. To identify changes, they consider headers and function constructors that change from the original library source code relative to the security patch, subsequently identifying their use in their source code.

Another interesting approach to take under consideration is the impact of using unvalidated input, also known as taint data. Taintedness is propagated in the obvious way, as strings derived from

tainted string are also considered tainted. Perl's taint mode [114] prevents the use of taint data as arguments for sensitive functions that can affect the local system creating files, sending data over the network or running local commands among others. Inspired by this idea, Haldar et al. [115] introduce a technique for tagging, tracking and detecting the taintedness of untrusted input throughout the lifetime of the application. To this end, both data originating from the client as tainted and data derived from tainted data are marked. In this approach, they track taintedness from sources to methods that should not use taint data, and prevent tainted data from being passed into them, which could lead to an improper use in a security-sensitive context.

Moreover, many researchers have proposed the use of graph methods, such as Bayesian Networks and attack graphs among others, for providing an estimation of risk factors and being able to analyse the complexity and uncertainty of the propagation of a vulnerability. To make these methods more effective, it is important to use vulnerability propagation analysis tools that are capable of proactively assessing propagation. This can be obtained by using graph optimization methods.

One of these methods is the known Ant Colony Optimization [19]. Ant Colony Optimization is a metaheuristic method that is inspired by the behaviour of real ants. This algorithm is used to solve computational problems that can be solved by finding a good path through a graph. The combination of this method together with the use of Bayesian Networks [16] built a security risk analysis model (SRAM) to which determine the propagation paths with the highest probability and the largest estimated risk value.

Other authors like Hu et al. [116] propose a search algorithm based on lazy strategy, in terms of the macro component level, and on propagation difficulty when it comes to the microcode class level. With it, it is possible to reduce the search space complexity from exponential to polynomial. Furthermore, the algorithm can effectively identify software vulnerabilities that may affect a specific project.

Although numerous works have already been done in attack graph analysis, most of the works have focused on vulnerability prioritization instead of attack path prioritization. Many algorithms have been developed to prioritize individual vulnerability, but these do not work well in multi-step attack situations. Thus, it is essential to propose an overall security rate to indicate the risk level of the complete attack path and not just the single vulnerability. Agrawal and Khan [17] used the idea of Breadth First Search (BFS) algorithm (a graph and tree search algorithm) to propose an algorithm for computing vulnerability propagation of an attribute. To do so, they introduce a measure called Attribute Vulnerability Ratio (AVR). More recently, Garg et al. [4] developed a hybrid methodology to estimate attack path score by first calculating the risk score of individual vulnerabilities using CVSS score metrics (base, temporal and environmental metrics) to prioritize them, and using Page rank model and Markov model to calculate the attack path risk score, thus being able to be prioritized. Hence, it will be easy to decide which vulnerability needs to be patched first to avoid a complete attack path.

5.2. Review on the Application of MUD Files for vulnerability assessment based on Security Policies

The specification of the intended behaviour of IoT devices could help to avoid or mitigate potential security attacks. In this direction, policy-based approaches have been traditionally considered to define the set of allowed and denied actions for a particular system. One of the most

prominent examples is the eXtensible Access Control Markup Language (XACML)[70], which is considered the de facto standard for the definition and enforcement of access control policies.

One of the main advantages of XACML is the interoperability it provides between different vendor access control implementations. Also related to access control, token-based methods have been widely used to perform authorization. It provides a simple method where the token contains the permissions of a specific entity.

In addition to the access control aspect, other recommendations are also defined to express other restrictions on system behaviour. Specifically, authors in [117] propose a layered architecture in which an access control policy model is used for security and privacy. The method is based on the model-based security toolkit Seckit, which provides different meta-models to represent the security requirements of the system behaviour.

Regarding network-layer security aspects, authors of [118] and [119] believe that the network behaviour of a device is predictable and therefore easily restricted. Indeed, they propose a network security policy enforcement architecture based on this idea by restricting the network behaviour of the devices.

Network aspects are also the focus of the recently created (in 2019) Internet Engineering Task Force (IETF) Manufacturer Usage Description (MUD)[71] standard, which is intended to represent the intended network behaviour of IoT devices. The MUD standard defines the architecture and data model necessary to restrict the communication from and to a device. This file is intended to be generated by the manufacturer, as a way to describe the intended network behaviour of their devices, which can be used later to detect unwanted behaviours that could derive on vulnerabilities and attacks.

MUD files define the type of communications and access of a certain device in the form of policies or ACLs. In this sense, the MUD standard is capable of specify policies of the following types: “allow the communication between devices of the same manufacturer”, “allow the access to the controller”, or “deny the communications coming from a specific port”. MUD is based on YANG[72] standard to model such restrictions, and JSON[73] for serialization purposes. It also provides mechanisms to extend the MUD model, so manufacturers can express other conditions not contemplated in the standardized MUD data model (e.g., Quality of Service (QoS) proposed in [120]).

Hamza et al. [121], as well as authors in [122] and [123] used the MUD rules as input for an Intrusion Detection System (IDS). They also discussed the limitation of the MUD in protecting the device from local attacks, as local endpoints are not defined in the MUD file. Authors in [124] focus on the battery life extension within the 802.11ax devices, optimizing the wake time. In [125], the authors focus on flooding attacks, using Software Defined Networks (SDNs) and MUD rules. Also, the project proposed in [126] reduces the vulnerabilities and increase the resilience of devices in the smart home domain, combining the MUD with threat signalling and updates. However, this documents only gives some recommendations and it does not enter into details. The guidelines of the document are completed in the NIST Cybersecurity Practice Guide[74] (SP 1800-15), which explains how MUD protocols and tools can reduce the vulnerability of IoT devices to botnets and other network-based threats as well as reduce the potential for harm from exploited IoT devices.

[70] http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html
[71] https://tools.ietf.org/html/rfc8520
[72] https://tools.ietf.org/html/rfc6020
[73] https://datatracker.ietf.org/doc/rfc8259/
[74] https://csrc.nist.gov/publications/detail/sp/1800-15/draft

Regarding validation, the thesis developed in [127] pre certifies the security of a device based on its MUD file, validating that the behaviour is the one described in the file. Furthermore, the thesis considers existing vulnerabilities (e.g., for a specific version of a protocol) and all is performed in an automated way.

However, MUD files have not been applied only to simple IoT devices. Authors in [128] combines the MUD file with machine learning techniques within the 5G environment for predicting the resources the user will need. Also, MUD files have been applied to more complex devices such as smartphones [129] with the objective to mitigate the threat of malicious apps and IoT devices in smart home networks. Even so, the MUD model, as described in the standard is limited to certain network layer aspects. In this sense, and coping with this, current research has addressed this limitation by extending the model to integrate more fine-grained information that could be useful to detect more types of vulnerabilities. Authors in [130] extend the MUD model to consider dynamic aspects in the context of smart homes, whereas in [131], authors propose a new behavioural profile based on the MUD with the aim of creating a feature vector. Moreover, authors in [132] define an augmented MUD profile to include security properties such as key sizes or cryptographic algorithms and to limit the maximum number of simultaneous connections. The same authors in [133] extend the MUD model to integrate the Medium-level Security Policy Language (MSPL), which provides flexibility to define other policies (privacy, data protection, channel protection, authorization, etc.).

MUD files have been widely used in current research to validate the configuration of the device, detecting misalignments from the recommended behaviour. Although in the majority of the cases, these detections come hand by hand with an IDS, MUD files can be also used during design phase as a way to validate the conformity of the manufacturer recommendations with the system, before it is released into the market, adjusting it in case it is necessary. Moreover, although the provided semantics does not allow the specification of more fine-grained security aspects to be defined, some of the analysed works provide extensions to integrate more information in the MUD model that can be also analysed during the design phase (e.g., key lengths or cryptographic algorithms used). Due to the significant growth of IoT devices, the use of a standardized approach such as MUD will be crucial to face existing and new security threats, as well as the heterogeneity of existing devices and technologies. Furthermore, although the original MUD file was oriented to IoT devices, the research has proved that the potential of the MUD goes beyond, providing behavioural profiles to more complex systems such as mobile phones or 5G environments, and providing extensions to reflect more types of security policies.

6. Conclusions

In this document a review of the main vulnerability datasets and standards was provided, as well as a review of the state of the art on vulnerability detection, forecasting and propagation.

Based on the idea that vulnerability identification is one of the first steps in software security lifecycle, the techniques for this purpose have been reviewed and categorized in the following three topics: anomaly detection, vulnerable pattern recognition and vulnerability prediction models.

Anomaly detection approaches can be used to find vulnerabilities caused by the use of improper APIs as well as missing checks or neglected conditions. However, the assumption that missing checks or the use of improper APIs are rare events makes deviations that cannot be easily detected, leading to high false-positive rates. On the other hand, vulnerable pattern recognition is focused on discovering vulnerabilities by defining patterns of vulnerable code segments. Even though the results provided from the aforementioned works were acceptable, they suffer from some important shortcomings since they employ approaches based on very limited or shallow information about the software. An effective modelling and discovery of vulnerabilities requires information about different aspects of software like syntactic, control-flow and data-flow. A different perspective for trying to predict the presence of software vulnerabilities is by means of VPM based on software metrics.

It should be noted that, although detection is a crucial step and can be obtained by means of the aforementioned techniques, identifying which ones are more likely to be exploitable could be even more important in order to prioritize the vulnerability management. Being aware of this, in the VPM context based on metrics, some studies were performed focused on this goal, unlike the other two approaches.

To the best of our knowledge, anomaly detection, vulnerable pattern recognition and vulnerability prediction models are focused on the detection of vulnerabilities in a general way. Although detection is the first step to reduce the risk in software security, it is also interesting to determine the type of vulnerability, in order to prioritize which ones could be exploited sooner. Taking this into account, BIECO will focus on the **research of mechanisms that allow not only to detect but also to try to identify the type of vulnerability**, as well as to **provide their fine-graine location**. Furthermore, the application of Federated Learning techniques will be explored more in depth for the training of the Machine Learning models used in the vulnerability detection process. **The application of Federated Learning, besides not being used in the field of vulnerability detection until the date, could allow to solve confidentiality issues** (avoiding the sharing of sensitive source code).

Related to the forecasting topic, **there are two Machine Learning methods**, Differentiable Neural Computer and Tsetlin Machine**, that have not been used yet in vulnerabilities forecasting**, but we expect them to achieve good performance after some improvements. It should be noted that these methods are generally far more effective than any other methods previously used for time series forecasting, including vulnerabilities forecasting. Another key approach that could be undertaken is focused on the **novel architectures of the neural network's models**, like the Transformer architecture and Capsule Networks. These architectures exhibited remarkable results in NLP (Natural Language Processing) and image recognition tasks, outperforming previous models by a solid margin. The usage of these architectures for vulnerabilities prediction is not researched fully yet, but we consider it a very promising approach nonetheless. Additionally, based on the reviewed literature of the subsection 4.1 **for predicting when a vulnerability will be exploited within a period of time, our idea is to provide a methodology** applied to the BIECO´s use cases **considering a closest period of time** (e.g., 6 months), or at least, the same period (within the next 12 months).

In terms of propagation, and after the reviewed studies, **graph algorithms will be explored to model the path that a vulnerability can follow across a source code**. Furthermore, since a vulnerability can be propagated through different paths, several optimization graphs algorithms are going to be tested in order to obtain the most prone one. In addition, the use of severity rating methods will be assessed as a complement to the path optimization.

Finally, policy-based approaches, and specially the recently approved **MUD standard**, appears as a promising tool that could allow not only to mitigate suspicious behaviours during the runtime phase, but also to **detect misalignment with the manufacturer's security specifications during the design phase**, applying the corrections that are necessary before its market release.

The review of the different approaches and the works mentioned in this document provides us with a good starting point for the design of the different tools that will be developed in order to perform the vulnerability assessment process of BIECO. These approaches will be further researched within the tasks “T3.3 Vulnerability detection and forecasting” and “T3.4 Vulnerability propagation”, and the preliminary results documented in deliverables “D3.3 Report of the tools for vulnerability detection and forecasting” and “D3.4 Report of the tools for vulnerabilities propagation”.

7. References

[1] S. Samonas and D. Coss, "The CIA strikes back: redefining Confidentiality, Integrity and Availability in security," *Journal of Information System Security*, vol.10, no. 3, pp. 21–45, Jul. 2014

[2] R. Amankah, P. K. Kudjo, S. Y. Antwi "Evaluation of software Vulnerability Detection Methods and Tools: A review," *International Journal of Computer Applications*, vol 69, no. 8, pp. 22—27, 2017

[3] M. Cadariu, E. Bouwers, J. Visser, and A. van Deursen, "Tracking known security vulnerabilities in proprietary software systems," *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 516–519, Mar. 2015

[4] U. Garg, G. Sikka, and L. K. Awasthi, "Empirical analysis of attack graphs for mitigating critical paths and vulnerabilities," *Comput. Secur.*, vol. 77, pp. 349–359, Aug. 2018.

[5] Z. Li and Y. Zhou, "PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 5, pp. 306–315, Sep. 2005.

[6] N. Gruska, A. Wasylkowski, and A. Zeller, "Learning from 6,000 projects: lightweight cross-project anomaly detection," *Proceedings of the 19th international symposium on Software testing and analysis*, pp.119-130, Jul. 2010.

[7] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting Vulnerable Software Components via Text Mining," *IEEE Trans. Software Eng.*, vol. 40, no. 10, pp. 993–1006, Oct. 2014.

[8] G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, and L. Mounier, "Toward Large-Scale Vulnerability Discovery using Machine Learning," *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pp. 85-96, Mar. 2016.

[9] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista," *Third International Conference on Software Testing, Verification and Validation*, pp.421-428, Apr. 2010.

[10] Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Çomak, and L. Karaçay, "Vulnerability prediction from source code using Machine Learning," *IEEE Access*, vol. 8, pp. 150672–150684, 2020.

[11] Y. Roumani, J. K. Nwankpa, and Y. F. Roumani, "Time series modelling of vulnerabilities," *Comput. Secur.*, vol. 51, pp. 32–40, Jun. 2015.

[12] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Int. J. High Perform. Syst. Archit.*, vol. 57, no. 3, pp. 294–313, Mar. 2011.

[13] G. O. Kaya and O. F. Demirel, "Parameter optimization of intermittent demand forecasting by using spreadsheet," *Kybernetes*, vol. 44, no. 4. pp. 576–587, 2015.

[14] J. Jacobs, S. Romanosky, B. Edwards, M. Roytman, and I. Adjerid, "Exploit Prediction Scoring System (EPSS)," *arXiv [cs.CR]*, Aug. 13, 2019.

[15] H. Chen, R. Liu, N. Park, and V. S. Subrahmanian, "Using Twitter to Predict When Vulnerabilities will be Exploited," *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019.

[16] N. Feng, H. J. Wang, and M. Li, "A security risk analysis model for information systems: Causal relationships of risk factors and vulnerability propagation analysis," *Inf. Sci.*, vol. 256, pp. 57–73, Jan. 2014.

[17] A. Agrawal and R. A. Khan, "Impact of inheritance on vulnerability propagation at design phase," *SIGSOFT Softw. Eng. Notes*, vol. 34, no. 4, pp. 1–5, Jul. 2009

[18] M. Dorigo and L.M. Gambardella, "Ant Colony System: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66. 1997.

[19] M. Abadi and S. Jalili, "An ant colony optimization algorithm for network vulnerability analysis" *Iranian Journal of Electrical and Electronic Engineering*, vol. 2, no. 3 & 4, July 2006.

[20] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?," *Empirical Software Engineering*, vol. 18, no. 1, pp. 25–59, 2013.

[21] N. Dor, M. Rodeh, and M. Sagiv, "CSSV: towards a realistic tool for statically detecting all buffer overflows in C," *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pp. 155–167, May 2003.

[22] D. Larochelle and D. Evans, "Statically detecting likely buffer overflow vulnerabilities," *10th USENIX Security Symposium*, 2001.

[23] J. Viega, J. T. Bloch, Y. Kohno, and G. McGraw, "ITS4: a static vulnerability scanner for C and C++ code," *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)*, pp. 257-267, Dec. 2000.

[24] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[25] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 93–104, May 2000.

[26] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, and Others, "A density-based algorithm for discovering clusters in large spatial databases with noise," *Kdd*, vol. 96, pp. 226–231, 1996.

[27] P. Cunningham and S.J. Delany, "k-Nearest neighbour classifiers," *Multiple Class. Syst.*, vol 34, pp. 1-17, 2007.

[28] F. T. Liu, K. M. Ting and Z. Zhou, "Isolation Forest," *Eighth IEEE International Conference on Data Mining*, pp. 413-422, 2008.

[29] R. Agrawal, T. Imieliński, A. Swami, "Mining association rules between sets of items in large databases," *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, p. 207, 1993.

[30] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Outlier Detection in Axis-Parallel Subspaces of High Dimensional Data," in *Advances in Knowledge Discovery and Data Mining*, 2009, pp. 831–838.

[31] H.P. Kriegel, M. Schubert, and A. Zimek, "Angle-based outlier detection in high-dimensional data," *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 444–452, Aug. 2008.

[32] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf, "Bugs as deviant behaviour: A general approach to inferring errors in systems code", *Proc. 18th SOSP*, pp. 57-72, 2001.

[33] A. Wasylkowski, A. Zeller, and C. Lindig, "Detecting object usage anomalies," *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 35–44, Sep. 2007.

[34] M. Acharya, T. Xie, J. Pei, and J. Xu, "Mining API patterns as partial orders from source code: from usage scenarios to specifications," *Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 25–34, Sep. 2007.

[35] R. Chang, A. Podgurski, and J. Yang, "Discovering Neglected Conditions in Software by Mining Dependence Graphs," *IEEE Trans. Software Eng.*, vol. 34, no. 5, pp. 579–596, Sep. 2008.

[36] S. Thummalapenta and T. Xie, "Alattin: Mining Alternative Patterns for Detecting Neglected Conditions," *2009 IEEE/ACM International Conference on Automated Software Engineering*, pp. 283–294, Nov. 2009.

[37] B. Livshits and T. Zimmermann, "DynaMine: finding common error patterns by mining software revision histories," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 5, pp. 296–305, Sep. 2005.

[38] F. Yamaguchi, M. Lottmann, and K. Rieck, "Generalized vulnerability extrapolation using abstract syntax trees," *Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 359–368, Dec. 2012.

[39] F. Yamaguchi, F. Lindner, and K. Rieck, "Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning," *Proceedings of the 5th USENIX conference on Offensive technologies*, p. 13, Aug. 2011.

[40] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modelling and Discovering Vulnerabilities with Code Property Graphs," *2014 IEEE Symposium on Security and Privacy*, pp. 590–604, May 2014.

[41] F. Yamaguchi, A. Maier, H. Gascon, and K. Rieck, "Automatic Inference of Search Patterns for Taint-Style Vulnerabilities," *2015 IEEE Symposium on Security and Privacy*, pp. 797–812., May 2015.

[42] Y. Pang, X. Xue, and A. S. Namin, "Predicting Vulnerable Software Components through N-Gram Analysis and Statistical Feature Selection," *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. 2015.

[43] R.E. Bellman, "Adaptive Control Processes: a guided tour*," Princeton University Press, NJ*, 1961.

[44] L. K. Shar and H. B. K. Tan, "Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns," *Information and Software Technology*, vol. 55, no. 10, pp. 1767–1780, Oct. 2013.

[45] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: a static analysis tool for detecting Web application vulnerabilities," *2006 IEEE Symposium on Security and Privacy (S P'06)*, p. 6 pp.–263, May 2006.

[46] L. K. Shar, H. Beng Kuan Tan, and L. C. Briand, "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis," *2013 35th International Conference on Software Engineering (ICSE)*, pp. 642–651, May 2013.

[47] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 6, pp. 688–707, Nov. 2015.

[48] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "Sysevr: A framework for using deep learning to detect software vulnerabilities," *arXiv*, 2018.

[49] Z. Li *et al.*, "Vuldeepecker: A deep learning-based system for vulnerability detection," *arXiv*, 2018

[50] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 529–540, Oct. 2007.

[51] G. James, D. Witten, T. Hastie, and R. Tibshirani, "An Introduction to Statistical Learning: with Applications in R," *Springer*, 2013.

[52] T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction," *Second Edition. Springer Science & Business Media*, 2009.

[53] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4–5, pp. 531–577, 2012.

[54] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 529–540, Oct. 2007.

[55] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting component failures at design time," *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pp. 18–27, Sep. 2006.

[56] Y. Shin and L. Williams, "An empirical model to predict security vulnerabilities using code complexity metrics," *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pp. 315–317, Oct 2008.

[57] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities," *IEEE Trans. Software Eng.*, vol. 37, no. 6, pp. 772–787, Nov. 2011.

[58] M. Gegick, L. Williams, J. Osborne, and M. Vouk, "Prioritizing software security fortification throughcode-level metrics," *Proceedings of the 4th ACM workshop on Quality of protection*, pp. 31–38, Oct 2008.

[59] P. Morrison, K. Herzig, B. Murphy, and L. Williams, "Challenges with applying vulnerability prediction models," *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, pp. 1–9, Apr. 2015.

[60] A. Younis, Y. Malaiya, C. Anderson, I. Ray "To fear or not to fear that is the question: Code characteristics of a vulnerable function with an existing exploit," *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, 2016

[61] G. A. Kaissis, M. R. Makowski, D. Rückert, and R. F. Braren, "Secure, privacy-preserving and federated machine learning in medical imaging," *Nature Machine Intelligence*, vol. 2, no. 6, pp. 305–311, Jun. 2020.

[62] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, vol. 54, pp. 1273–1282, 2017.

[63] "Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy," White House, Washington, DC, pp. 1–62, 2012.

[64] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Gener. Comput. Syst.*, vol. 115, pp. 619–640, Feb. 2021.

[65] R. J. Hyndman and G. Athanasopoulos, "Forecasting: principles and practice*," OTexts*, 2018.

[66] E. Bee Dagum and S. Bianconcini, "Seasonal Adjustment Methods and Real Time Trend-Cycle Estimation," *Springer, Cham*, 2016.

[67] R. B. Cleveland and Others, "STL: A seasonal-trend decomposition procedure based on loess," 1990.

[68] M. Theodosiou, "Forecasting monthly and quarterly time series using STL decomposition," *Int. J. Forecast.*, vol. 27, no. 4, pp. 1178–1195, Oct. 2011.

[69] R. G. Brown, "Statistical Forecasting for Inventory Control," *McGraw-Hill*, 1959.

[70] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages*," Defense Technical Information Center*, 1957.

[71] P. R. Winters, "Forecasting Sales by Exponentially Weighted Moving Averages," *Manage. Sci.*, vol. 6, no. 3, pp. 324–342, Apr. 1960.

[72] R. Hyndman, A. B. Koehler, J. Keith Ord, and R. D. Snyder, "Forecasting with Exponential Smoothing: The State Space Approach," *Springer Science & Business Media*, 2008.

[73] B. Abraham and J. Ledolter, "Statistical methods for forecasting," *John Wiley & Sons*, vol. 234, 2009

[74] Z. Cai, L. Chen, and Y. Fang, "A new forecasting model for usd/cny exchange rate," *Studies in Nonlinear Dynamics & Econometrics*, vol. 16, no. 3, 2012.

[75] C.-J. Kim, "Dynamic linear models with markov-switching," *Journal of Econometrics*, vol. 60, no. 1-2, pp. 1–22, 1994.

[76] V. Assimakopoulos and K. Nikolopoulos, "The theta model: a decomposition approach to forecasting," *International journal of forecasting*, vol. 16, no. 4, pp. 521–530, 2000.

[77] R. J. Hyndman, Y. Khandakar, and Others, "Automatic time series for forecasting: the forecast package for R," *Monash University, Department of Econometrics and Business Statistics*, 2007.

[78] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, "Time Series Analysis: Forecasting and Control," *John Wiley & Sons*, 2015.

[79] P. J. Brockwell and R. A. Davis, "Introduction to Time Series and Forecasting," *Springer, Cham*, 2016.

[80] D. Peña, G. C. Tiao, and R. S. Tsay, "A Course in Time Series Analysis," *John Wiley & Sons*, 2011.

[81] K. Ord and R. Fildes, "Principles of business forecasting," *Nelson Education*, 2012.

[82] F. E. Harrell and Jr., "Regression Modelling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis," *Springer*, 2015.

[83] A. Pankratz, "Forecasting with Dynamic Regression Models," *John Wiley & Sons*, 2012.

[84] N. Bhatt, A. Anand, and V.S.S. Yadavalli, "Exploitability prediction of software vulnerabilities," *Quality and Reliability Engineering International*, 2020.

[85] K. A. Farris, A. Shah, G. Cybenko, R. Ganesan, and S. Jajodia, "VULCON," *ACM Transactions on Privacy and Security*, vol. 21, no. 4. pp. 1–28, 2018.

[86] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker. "Beyond heuristics: learning to classify vulnerabilities and predict exploits," *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 105-113, 2010.

[87] M. Edkrantz and A. Said. "Predicting cyber vulnerability exploits with machine learning," *Thirteenth Scandinavian Conference on Artificial Intelligence*, pp. 48-57, 2015.

[88] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian and P. Shakarian, "Proactive identification of exploits in the wild through vulnerability mentions online," *2017 International Conference on Cyber Conflict (CyCon U.S.)*, 2017.

[89] C. Sabottke, O. Suciu, and T. Dumitraș, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits," *24th USENIX Security Symposium*, pp. 1041–1056, 2015.

[90] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian, and P. Shakarian, "Proactive identification of exploits in the wild through vulnerability mentions online," *2017 International Conference on Cyber Conflict (CyCon U.S.)*. 2017

[91] H. S. Venter and J. H. P. Eloff, "Vulnerability forecasting—a conceptual model," *Comput. Secur.*, vol. 23, no. 6, pp. 489–497, Sep. 2004.

[92] D. Last, "Forecasting Zero-Day Vulnerabilities," *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, pp. 1–4, Apr. 2016.

[93] A. Ozment, S. E. Schechter, and R. Dhamija, "Web sites should not need to rely on users to secure communications," 2006.

[94] E. Yasasin, J. Prester, G. Wagner, and G. Schryen, "Forecasting IT security vulnerabilities – An empirical analysis," *Computers & Security*, vol. 88. p. 101610, 2020.

[95] J. D. Croston, "Forecasting and Stock Control for Intermittent Demands," *J. Oper. Res. Soc.*, vol. 23, no. 3, pp. 289–303, Sep. 1972.

[96] M. R. Amin-Naseri and B. R. Tabar, "Neural network approach to lumpy demand forecasting for spare parts in process industries," *2008 International Conference on Computer and Communication Engineering*, pp. 1378–1382, May 2008.

[97] N. Kourentzes, “Intermittent demand forecasts with neural networks,” *Int. J. Prod. Econ.*, vol. 143, no. 1, pp. 198–206, May 2013.

[98] M. Zhao, J. Grossklags, and P. Liu, “An Empirical Study of Web Vulnerability Discovery Ecosystems,” *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1105–1117, Oc. 2015.

[99] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, “The m4 competition: results, findings, conclusion and way forward,” *International Journal of Forecasting*, vol. 34, no. 4, pp. 802–808, 2018.

[100] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, “The m4 competition: 100,000 time series and 61 forecasting methods,” *International Journal of Forecasting*, 2019.

[101] S. Smyl, “A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting,” *International Journal of Forecasting*, vol. 36, no. 1, pp. 75–85, 2020.

[102] P. Montero-Manso, G. Athanasopoulos, R.J. Hyndman, and T.S. Tala- gala, “FFORMA: feature-based forecast model averaging,” *International Journal of Forecasting, vol.* 36, no. 1, pp. 86–92, 2020.

[103] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, “N-beats: neural basis expansion analysis for interpretable time series forecasting,” *arXiv*, 2019.

[104] L. Le and Y. Xie, “Deep embedding kernel,” *Neurocomputing*, vol. 339, pp. 292–302, 2019.

[105] L. Shen, J. Chen, Z. Zeng, J. Yang, and J. Jin, “A novel echo state network for multivariate and nonlinear time series prediction,” *Applied Soft Computing*, vol. 62, pp. 524–535, 2018.

[106] Hu, Hao, and Guo-Jun Qi. "State-frequency memory recurrent neural networks." *International Conference on Machine Learning*. 2017.

[107] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al., “Hybrid computing using a neural network with dynamic external memory,” *Nature*, vol. 538, no. 7626, p. 471, 2016.

[108] O.-C. Granmo, S. Glimsdal, L. Jiao, M. Goodwin, C. W. Omlin, and G. T. Berge, “The convolutional tsetlin machine,” *arXiv*, 2019.

[109] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[110] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” *Advances in neural information processing systems*, pp. 3856–3866, 2017.

[111] A. E. Hassan and R. C. Holt, “Predicting change propagation in software systems,” *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, pp. 284–293, Sep. 2004.

[112] B. Li, X. Sun, H. Leung, and S. Zhang, “A survey of code-based change impact analysis techniques: A SURVEY OF CODE-BASED CIA TECHNIQUES,” *Softw. Test. Verif. Reliab.*, vol. 23, no. 8, pp. 613–646, Dec. 2013.

[113] H. Plate, S. E. Ponta, and A. Sabetta, "Impact assessment for vulnerabilities in open-source software libraries," *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 411–420, Sep. 2015.

[114] L. Wall and T. Christiansen, with J. Orwant, "Programming Perl," *O'Reilly*, 3rd edition, 2000.

[115] V. Haldar, D. Chandra, and M. Franz, "Dynamic taint propagation for Java," *21st Annual Computer Security Applications Conference (ACSAC'05)*, p. 9 pp.–311, Dec 2005.

[116] W. Hu, Y. Wang, X. Liu, J. Sun, Q. Gao, and Y. Huang, "Open Source Software vulnerability propagation analysis algorithm based on Knowledge Graph," *2019 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 121–127, Dec. 2019.

[117] C. Sarkar, A. U. Nambi S. N., R. V. Prasad, A. Rahim, R. Neisse, and G. Baldini, "DIAT: A scalable distributed architecture for IoT," *IEEE Internet Things J.*, vol. 2, no. 3, pp. 230-239, Jun. 2015.

[118] D. Barrera, I. Molloy, and H. Huang, "Standardizing IoT network security policy enforcement," *Proc. Workshop Decentralized IoT Secur. Standards*, pp. 1-6, 2018.

[119] D. Barrera, I. Molloy, and H. Huang, "IDIoT: Securing the Internet of Things like it's 1994," *arXiv*, Dec. 2017.

[120] E. Lear, J. Henry, and R. Barton, "Determining nominal quality of service needs of a device,"

[121] A. Hamza, H.H. Gharakheili, and V. Sivaraman, "Combining MUDPolicies with SDN for IoT Intrusion Detection," *Proceedings of the 2018 Workshop on IoT Security and Privacy*, pp. 1–7, ACM,2018.

[122] H. J. Hadi, S. M. Sajjad, and K. un Nisa, "BoDMitM: Botnet Detection and Mitigation System for Home Router Base on MUD," *International Conference on Frontiers of Information Technology (FIT)*, pp. 139–1394, 2019.

[123] Y. Afek, A. Bremler-Barr, D. Hay, L. Shafir, and I. Zhaika, "Demo: NFV-based IoT Security at the ISP Level," p.2.

[124] V. Lade, A. Mohan, and S. Patil, "802.11AX for the Internet of Things- Machine Learning assisted optimized power save techniques for IoT devices using 802.11AX target wake time"

[125] L. Chang, "A Proactive Approach to Detect IoT Based Flooding Attacks by Using Software Defined Networks and Manufacturer Usage Descriptions."

[126] T. Polk, M. Souppaya, and W. C. Barker, "Mitigating IoT-Based Automated Distributed Threats," 2017.

[127] C. Gangurde, "Automation of IoT pre-certification security testing environment based on the Manufacturing Usage Description."

[128] M. Hanes, C. Byers, J. Clarke, and G. Salgueiro, "Human usage description for 5G networks endpoints"

[129] I. Berenice, M. Serror, and K. Wehrle, "Extending MUD to Smartphones," presented at the *45th IEEE Conference on Local Computer Networks (LCN)*, 2020.

[130] Z. Jin, Y. M. Lee, C. H. Copass, and Y. Park, "Building system with dynamic manufacturer usage description (MUD) files based on building model queries," Apr. 30 2020.

[131] A. Singh, S. Murali, L. Rieger, R. Li, S. Hommes, R. State, G. Ormazabal, and H. Schulzrinne, "Hanzo: Collaborative network defence for connected things," *2018 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, pp. 1–8, IEEE, 2018.

[132] S. N. Matheu, J. L. Hernandez-Ramos, S. Perez, and A. F. Skarmeta, "Extending MUD profiles through an Automated IoT Security Testing Methodology," *IEEE Access*, pp. 1–20, 2019.

[133] S. N. Matheu, A. Robles Enciso, A. Molina Zarca, D. Garcia-Carrillo, J. L. Hernández-Ramos, J. Bernal Bernabe, and A. F. Skarmeta, "Security architecture for defining and enforcing security profiles in dlt/sdnbased iot systems," *Sensors*, vol. 20, no. 7, p. 1882, 2020.