Deliverable 3.3

Report of the Tools for Vulnerability Detection and Forecasting

Technical References

Document version	:	**1.0**
Submission Date	:	28/02/ 2022
Dissemination Level	:	Public
Contribution to	:	WP3 – Vulnerability Management
Document Owner	:	GRAD
File Name	:	BIECO_D3.3_28.02.2022_V1.0
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

This project has received funding from the European Union´s Horizon 2020 Research and Innovation Programme under Grand agreement No. 952702.

Revision History

REVISION	DATE	INVOLVED PARTNERS	DESCRIPTION
0.0	05/10/2021	GRAD	Draft structure of the document
0.1	06/10/2021	GRAD	Table of Contents
0.2	26/11/2021	GRAD	Contribution to Sections 2 and 3.
0.3	19/12/2021	GRAD	Contributions to Subsections 2.1, 2.2 and 2.3
0.4	13/01/2022	GRAD	Contributions to Section 1 and Executive Summary
0.5	23/01/2022	UTC	Added section 3.2, updated the acronyms table
0.6	26/01/2022	GRAD	Contributions to Subsection 2.4
0.7	30/01/2022	UTC	Added sections 3.2.4 and 3.2.5
0.8	02/02/2022	7B	Added section 3.3
0.9	03/02/2022	GRAD	Contributions to Section 4
0.10	04/02/2022	UTC	Contributions to Section 4
1.0	04/02/2022	GRAD	Internal review by Borja Pintos and implementation of suggestions
1.1	11/02/2022	IESE	First review
1.2	15/02/2022	GRAD, 7B, UTC	Comments from review addressed
1.3	23/02/2022	CNR	Second review
2.0	24/02/2022	GRAD	Comments from review addressed
2.1	26.02. 2022	UNI	Final Revision and correction by Coordinator
3.0	28.02.2022	UNI	Finalizing deliverable and submission

List of Contributors

Deliverable Creator(s): Eva Sotos (GRAD), Laura Pérez (GRAD), Javier Yépez (GRAD), Mónica Alonso (GRAD), Ovidiu Cosma (UTC), Paweł Skrzypek (7B), Radosław Piliszek (7B), Katarzyna Karnas (7B)

Reviewer(s): Borja Pintos (GRAD), Emilia Cioroaica (IESE), Felicita Di Giandomenico (CNR), Giulio Masetti (CNR); Sanaz Nikghadam-Hojjati(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
ANN	Artificial Neural Networks
AST	Abstract Syntactic Tree
AUC	Area Under the ROC Curve
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
DCT	Data Collection Tool
DM	Data Minning
EDA	Exploratory Data Analysis
EDB	Exploit Database
GLM	General Linear Models
GAM	Generalized Additive Models
GBT	Gradient Boosting Tree
GD-BP	Gradient Descent – Back Propagation
ICT	Information and Communication Technology
IT	Information Technology
LLoC	Logical Lines of Code
LoC	Lines of Code
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multilayer Perceptron
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
PoC	Proof-of-concept
PVE	Provisional CVE
RF	Randon Forest
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SLoC	Source Lines of Code
SVM	Support Vector Machine
TFT	Temporal Fusion Transformer
VPM	Vulnerability Prediction Models
WP	Work Package
XSS	Cross-site scripting attacks

Executive Summary

The goal of Work Package (WP3) is to research and develop a set of cybersecurity tools oriented to the detection, forecasting and propagation of vulnerabilities across complex Information and Communication Technology (ICT) systems. Many issues have to be considered when designing a tool for vulnerability management and, specially, when it relies on the use of Machine Learning (ML) models. The functionality of the tool as well as the definition of the concrete algorithms to be used are the main points to consider.

The core of the deliverable is to present the status and progress of the different tools developed within the task T3.3. The tools that make up the aforementioned task are those focused on the detection of vulnerabilities, as well as the forecast of different vulnerability parameters which will help to better assess system vulnerabilities. In terms of detection, a vulnerability detection tool is being developed with the goal to detect and identify vulnerabilities within the source code by the use of supervised learning algorithms. With regard to vulnerability forecasting, three different tools are developed. These tools are focused on the prediction of different parameters, the possibility of a vulnerability being exploited in a time window, the number of vulnerabilities detected in a period of time and the severity of a vulnerability.

The deliverable includes an introduction of the concept of security vulnerabilities and their assessment as well as a description of the different tools. For a better understanding of their development, some theoretical concepts of different ML models are included along with a selection of functions and data to be used in their training. Once the theoretical notions have been put into practice, a comparison of the results is presented together with the conclusions to summarize the most important advances made to date, as well as possible future actions.

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.

Partners


Disclaimer

Table of Contents

List of Figures

List of Tables

1. Introduction

In recent years a significant rise of cyberattacks has been detected[1]. The new normality derived from the COVID pandemic has led to an increase in the use of teleworking, as well as the use of more telematic processes that replace those previously carried out in person. This new lifestyle has become an attractive target for attackers, due to the fact that they have a large amount of information. In many cases this information is unprotected and with easily breakable access, which loss can provoke economical or even reputational damage. That is why cybersecurity has become such an important field in recent years.

One of the main factors which can help attackers to corrupt a system is the existence of vulnerabilities in the code. A simple vulnerability in the code can affect a whole system. Thus, it is vital to execute a suitable vulnerability assessment to the code.

To perform a good vulnerability and risk assessment process, it is important to complement a secure development methodology [1]. One of its main steps is to perform a code review or, in our case, the detection of critical or vulnerable code that can lead to a security breach. To this purpose, many have been the approaches used, having great relevance those focused on the use of Machine Learning (ML) and Data Mining (DM) algorithms. In the context of BIECO, we want to develop a tool which through ML and DM techniques is capable of detecting vulnerabilities within a static source code.

Detecting the existence of vulnerabilities within a system source is the first step in terms of vulnerability assessment. Nevertheless, such process alone is not enough to ensure a proper vulnerability assessment. Evaluating the severity or possible future impact of each discovered vulnerability are crucial steps when it comes to the security assessment in ICT components, as well as the prediction of future security threats for their risk identification. With it an improvement is achieved in order to prioritize mitigation efforts. Thus, and to provide the most comprehensive vulnerability assessment possible, three different forecasting tools are deployed. They are: a forecasting tool which estimates the likelihood of a given vulnerability to be exploited within a time window (e.g., 3, 6 or 12 months), a vulnerability forecasting tool which provides an estimation of the number of vulnerabilities to be expected in the main system software components for certain times frames, and a severity forecasting tool which predicts the severity of newly discovered and related vulnerabilities.

1.1 Connection with other tasks

The main purpose of WP3 is the research and development of the security tools and methodologies of BIECO's framework that are oriented to risk and security assessment. Particularly, it is focused on the detection, forecasting and propagation of vulnerabilities across complex ICT systems.

In the case of the task T3.3, four different tools are deployed to be implemented within the design phase: one focused on the detection of vulnerabilities in the source code, and another three which forecast different parameters related with the vulnerabilities.

[1] https://blog.checkpoint.com/2022/01/10/check-point-research-cyber-attacks-increased-50-year-over-year/

The dependency between the tools developed in WP3 within the BIECO framework is shown in Figure 1. If we focus on those corresponding to task T3.3, it is possible to observe how these depend on vulnerability database (mostly from the Data Collection Tool developed in T3.2), as well as how their different outputs provide the input to the tools created in WP7 for the security claim evaluation.

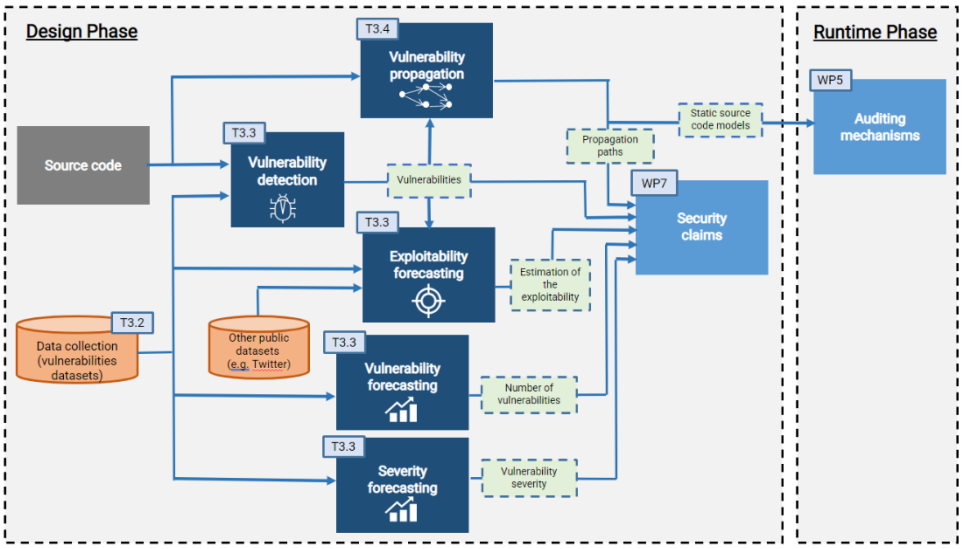

Figure 1: Dependency between tools

For a better understanding of the tools aforementioned, the deliverable is organized as follows:

Section 2 introduces the vulnerability detection tool, focusing on the description of the process to acquire the dataset to be used for the training of the ML algorithms, a theoretical description of the different methods adopted in the process of evaluation as well as the reasons for their selection, and the implementation and obtained results of the different models trained.

Section 3 presents the three different forecasting tools developed within WP3: exploitability forecasting tool, vulnerability forecasting tool and a new severity forecasting tool, surged at the beginning of this task. In it, a description of each tool as well as the models considered for their implementation, possible features and results, if any, are provided. Due to the recent incorporation of the development of a severity forecasting tool, in this case, only the tool is presented offering a brief state of the art and possible algorithms to be used.

Section 4 concludes the deliverable by reporting the conclusions obtained in the tool development process as well as future actions to be taken.

2. Vulnerability Detection

As it was mentioned in previous deliverables, [2] and [3], there are several procedures and techniques developed for the detection of vulnerabilities in the source code. After an extensive study and once the different existing possibilities have been assessed, a combination of techniques has been chosen boarding ML and DM techniques. In a first stage of the project, it has been decided to design a tool using Vulnerability Prediction Models (VPM) to detect vulnerable libraries in Python code. Detecting if a component could contain a vulnerability, helps to focus future efforts when analyzing the type of vulnerability contained. Once the approach has been chosen, it is necessary to obtain and transform the data to be used, following the steps of a typical data analytic project (Figure 2).

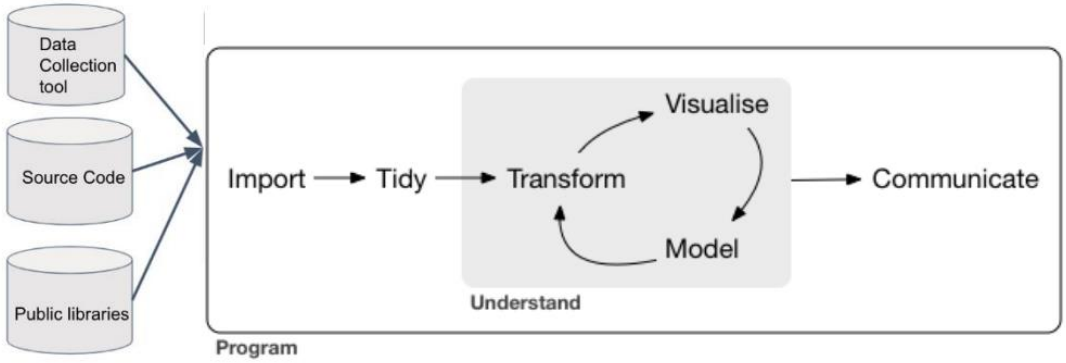

Figure 2: Structure of a data analytic project [4]

For the development of the aforementioned techniques, the task has been divided in different phases:

- Obtention of the dataset to use for the analysis of the models, which includes the search for different public repositories of vulnerability datasets, its download or *import*, and its *tidiness* (**Section 2.1**),
- a *transformation* of the dataset to acquire the selection of the different features which characterize the inputs/outputs of the models,
- an analysis of different *ML prediction models* (**Section 2.2**),
- and the *implementation* and results obtained from those (**Section 2.3**).

2.1 Dataset

In order to obtain a public dataset for the training of ML algorithms, a library download tool has been created. This tool takes public libraries from Pypi repositories[2] and divides them into vulnerable and not vulnerable. For having this classification, it has been consulted a repository[3] that indicates which libraries and versions are vulnerable to a

[2] https://pypi.org/
[3] https://raw.githubusercontent.com/pyupio/safety-db/master/data/insecure_full.json

given CVE (Common Vulnerabilities and Exposures) or PVE, in case that the CVE is still provisional.

After parsing the repository, the data is processed to obtain extra information, such as the versions in a library that are not vulnerable or their corresponding CWE (Common Weakness Enumeration), if available, which will be used by further analysis. Once the data is processed a total of 3204 versions of different libraries have been downloaded. For a first phase of the evaluation, libraries and their versions are divided in two categories:

1. vulnerable, that are those which have at least one known vulnerability, and
2. safe, those that have no known vulnerabilities.

After having carried out this classification, 1930 vulnerable and 1274 not vulnerable libraries have been obtained. This division is going to characterize the analysis which will determine if a library is vulnerable or not.

2.1.1 Features

Once the data is provided, it is necessary to define the variables that will provide the necessary information to predict the existence or not of vulnerabilities. To this end, it is required a preprocessing of the acquired information. Data preprocessing is understood as the computation needed to transform the collected data into suitable input data for modeling. This procedure implies tidiness of the data by storing them in a consistent form which corresponds to the descriptive characteristics of the collected data, i.e., features. This transformation implies the creation of new variables of interest, to calculate a set of summary statistics, etc. These features are used as inputs to the ML models to characterize the problem to solve.

The obtention of the descriptive variables that will appear in the dataset is made by the use of different tools. These tools perform a static analysis of the source code and extract different characteristics such as the number of lines, possible dependencies with security issues or the complexity, among others. To acquire the features to be used in the ML models for the detection tool, different existing tools have been used, as well as an internal developed tool which will provide those features not able to be obtained by the existing ones (Figure 3). These tools are described below:

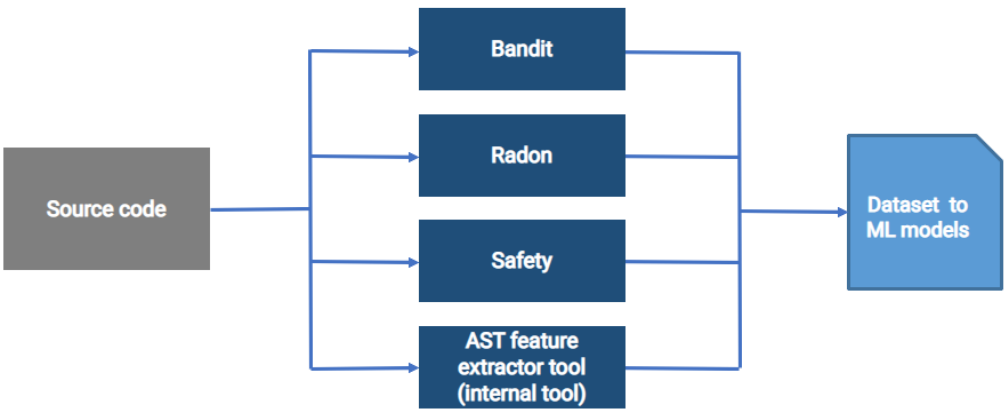

Figure 3: Feature extractor tools

Bandit (static code analysis)

Bandit[4] is a tool which provides common security issues in Python code. To do this, it processes each file, acquiring its Abstract Syntactic Tree (AST), and runs different plugins against the AST nodes. Once the process is finished, Bandit generates a report, available in different formats, with different issues together with the number of evaluated lines in the code and the number of issues classified by confidentiality and severity. The features obtained are the following:

- Plugin tests: which encompasses misc. tests, application/framework misconfiguration, blacklists (calls and imports), cryptography, injection and XSS (Cross Site Scripting)
- Confidence: high, low, medium and undefined
- Severity: high, low medium and undefined

Radon

Radon[5] is a tool based in Python which computes various code metrics to obtain information about code complexity. These supported metrics are:

- Raw metrics: based on the evaluation about the lines of the code. This information includes: LOC (lines of code), LLOC (logical lines of code), SLOC (source lines of code), comment lines and blank lines.
- Cyclomatic Complexity (i.e., McCabe's Complexity): a quantitative measure of the logical complexity presented by a code calculated from the associated AST.
- Halstead metrics: focused on the different Python operators (arithmetic, logic, assign...) which can lead to the existence of bugs.
- Maintainability Index (a Visual Studio metric): a software metric to assess the level of difficulty to support or modify a code.

Safety (dependency check)

Safety[6] is a command line tool that checks the local virtual environment, required files or any input from *stdin* for dependencies with security issues. As a result of the analysis, the tool provides a dependency report indicating those vulnerable libraries on which they are dependent on.

AST feature extractor tool

In addition to the previously described tools, an internal one has been developed to acquire those features of interest that the existing tools are not able to provide.

The tool, developed in Python, uses the public library *ast*[7] to obtain the AST code representation. AST is an abstract representation of the source code that keeps the structural and context details, excluding all the inessential punctuation and

[4] https://bandit.readthedocs.io/en/latest/index.html
[5] https://radon.readthedocs.io/en/latest/
[6] https://pyup.io/safety/
[7] https://docs.python.org/3/library/ast.html

delimiters such as braces, semicolons or parentheses. This is often used by compilers as an intermediate representation before generating the binary executable since they can be modified and enhanced programmatically (e.g., to check the correctness of the syntax, remove death code or apply some high-level optimizations.) (detailed in Annex A)

In our case we use the AST to extract or calculate different features which will describe the static characteristics of the code such as the number lines, function classes or the imported libraries between others. Even though it is possible to obtain these features without the use of this representation, the extraction of the AST makes it easier and, in some cases, instantaneous.

In order to obtain the final dataset and given the heterogeneity of the downloaded files, it is necessary to perform certain adaptation tasks to the downloaded libraries for being able to use the different tools selected and, therefore, obtain the features in a standardized way. After processing the filtered data, a total of 2159 records (1398 vulnerable and 761 no vulnerable) and 144 different features have been evaluated.

2.2 Models

Once the different features have been obtained, they are analyzed through a training process using different ML metrics based on predictive models. These models are based on the use of a unique equation applied to an entire sample space. Occasionally, its implementation is hard due to the fact that it can result difficult to find a single global model capable of reflecting the relationship between the variables.

As previous works reviewed in the state of the art in [2], and taking into account the complexity of the calculated features and the results to be obtained, as starting point it has been chosen the tree-based method Random Forest (RF). Thinking ahead about the possibility of obtaining results that are not clarifying enough, three more models are implemented to provide a complete analysis and achieve the best results through their comparison. These models are Gradient Boosting Tree (GBT), Support Vector Machine (SVM) and Generalized Additive Models (GAM).

2.2.1 Tree-based methods

Tree-based methods have become one of the benchmarks within the predictive fields since they provide good results in different areas, either regression or classification problems [5]. These methods encompass a set of supervised techniques that are able to split an entire sample space into simple regions, which makes it easier to handle their interactions. To do so, an initial node is taken as the beginning, which is formed by the entire training sample, and subdivided conditioned by a certain feature into two new subsets, giving rise to two new nodes. This process is performed recursively with subsequent created nodes until a predetermined finite number of times, thus obtaining a final decision tree whose nodes are used to perform the prediction.

In the case of predicting if a certain library is vulnerable or not, the nodes generated by the tree-based method should be pure, that is, vulnerable or no vulnerable observations. Nevertheless, the fulfillment of this condition in practice is difficult to achieve, due to the nature of the available data. This results in models that, even though they have low bias,

they present high variability, that is, a small data variation can lead to the creation of a different decision tree. Acquiring a balance between bias and variance is one of the main issues we can face when it comes to the selection of the tree-based model to use.

Random Forest

As discussed above, one of the main issues we can face when it comes to the use of tree-based methods, is to achieve a balance between bias and variance. To solve a possible decompensation between them, a combination of multiple models is used, better known as ensemble models, improving the predictions obtained by any of the original ones used individually. One of the most used ensemble models is *bagging*. These models adjust multiple models, each one trained with different training data, and provide the mean of the different obtained predictions or the most frequent class. One example of algorithm that is based on these models is Random Forest (RF) [5].

RF models offer suitable analysis results by implementing it in those scenarios in which a sheer number of features are handled. Some of the main advantages of such models are that they provide an automatic selection of predictors, which can be numerical or categorical, good scalability, low dependence on outliers and no need for standardization, among others.

This model is made up of an ensemble of individual decision trees which are trained individually with a random sample extracted from the original training data. The observations are distributed through the nodes generating the tree structure until reaching a terminal node or leaf. The set of predictions of each of the trees that make up the general model, generates the total prediction of the new observation (Figure 4).

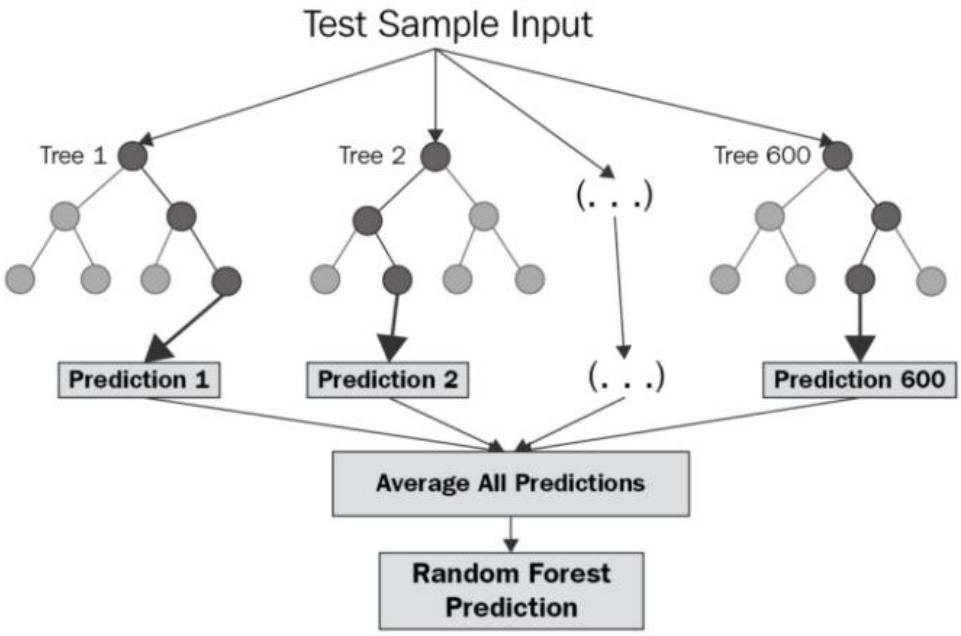

Figure 4: Random Forest Structure (8)

[8] https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f

The steps to follow when it comes to execution of the algorithm, are described as:

1. Being B the number of trees, for *b* = 1 to B
    a. Draw a bootstrap sample **Z*** of size *N* from the training data.
    b. Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
        i. Select *m* variables at random from the *p* variables.
        ii. Pick the best variable/split-point among the *m*.
        iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point *x*:

Regression: $f'^{B}_{rf}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $C'_b(x)$ be the class prediction of the *b*th random-forest tree. Then $C'^{B}_{rf}(x)$ = *average all predictions*$\{C'_b(x)\}_1^B$.

Gradient Boosting Tree

Another approach used to find the balance between bias and variance is the use of the ensemble models based on the use of *boosting*. Unlike bagging, these adjust sequentially multiple simple models. An example of these kind of models which use this kind of metric is Gradient Boosting Tree (GBT).

GBT are models made up of a set of individual decision trees, trained sequentially and achieving weak learners by the use of trees with one or few branches. Being run sequentially, each tree is trained by taking into account the information provided by the previous tree, correcting the prediction errors made to improve each iteration. This process is executed recursively until it achieves a final node, creating a complete tree structure. The prediction resulting from having executed all of them is the mean of all the individual results, or the most frequent class (Figure 5).

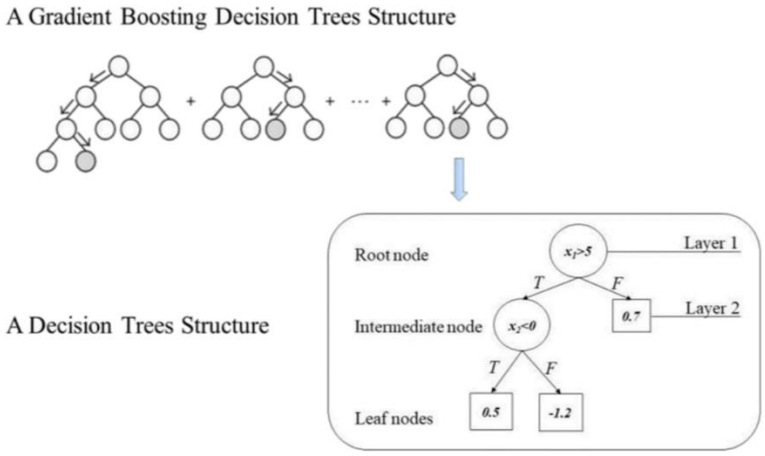

Figure 5: Gradient Boosting Decision Tree Structure (9)

[9]https://www.researchgate.net/figure/Structure-of-the-gradient-boosting-decision-trees_fig2_344395470

The algorithm for the implementation of the GBT models, can be described as follows:

1. Initialize $f_0(x) = \arg \min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$, where L is the loss function
2. For $m = 1$ to M:
   a. For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\delta L(y_i, f(x_i))}{\delta f(x_i)}\right]_{f=f_{m-1}}$$

   b. Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$.
   c. For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

   d. Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $f'(x) = f_M(x)$.

2.2.2 Support Vector Machine

Another model to take into account is the use of the algorithm Support Vector Machine (SVM) [6]. This algorithm provides good results when it comes to datasets with binary classification outputs, which can suggest a good performance in the case of detecting the vulnerability or non-vulnerability of a library.

SVM is a supervised and linear ML algorithm that looks at data and sorts it into one of two categories. These types of methods were created to solve binary classification problems, based on the idea of dividing data through hyperplanes, and maintaining all the main features that characterize the algorithm.

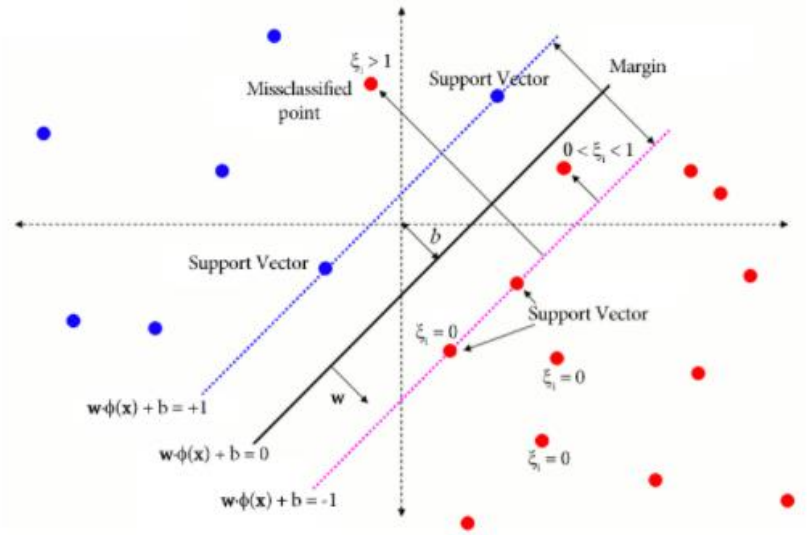

Figure 6: Classification in two groups by using SVM (10)

[10] https://fhernanb.github.io/libro_mod_pred/svm-clas.html

The objective is to build a tolerance margin (ε) with which there will be observations (support vectors) within the margins and outside them (Figure 6). Using only the observations outside the margins, the errors ξ are calculated and with them the objective function (FO) to be minimized is constructed[11], i.e.:

Minimize:

$$FO = \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{N} \xi_i \text{ , subject to } y_i (w, x_i) \geq 1, \forall i.$$

Where:

- w is the vector with the slopes associated with each of the variables
- C is the penalty value for the errors

2.2.3 Generalized Additive Models

An alternative methodology in terms of predictive models is the use of linear models, and in particular the General Linear Models (GLM) [7]. GLMs are a generalization of the stacking approach to ensemble learning that follows the concepts of the Super Learner models [8]. These models offer an approachable way to acquire a soft transition to more flexible models, while retaining some of the interpretability, as opposed to the opacity of many ML models.

To evaluate the possible improvement of results and make a more complete study, a model with a different approach was executed. In this case, it is decided to implement a model that allows greater flexibility in the dependence of each covariate with the response variable. This functionality can be obtained by the use of GLM which allows to interpret the features information knowledge. Nevertheless, despite the advantages offered by these kinds of models, they present some limitations when it comes to the relationship between predictors as well as the need for the variable response mean to be linear and constant. Thus, an extension of GLM which allows the use of non-linear relationships is used: GAM [9].

GAMs are regression models which require the assumption that the response variable follows a certain parametric distribution (normal, beta, and gamma), but with the particularity that the parameters used can be modeled, each one independently, following non-parametric functions (I.e., linear, additive or non-linear). This makes these models being considered as semi-parametric. Thanks to this versatility, GAMs are a suitable tool for the modeling of features which follow a wide range of distributions.

In these models, the relation between each predictor and the variable response mean is not direct, but it is made through a function. The most used ones are the smooth non-linear functions, such as cubic regression splines, tin plate regression splines or penalized splines.

The purpose of GAM is to maximize the prediction accuracy of a dependent variable and several distributions, by specific non-parametric functions of the predictor variable

[11] https://fhernanb.github.io/libro_mod_pred/svm-reg.html

which are connected to the dependent one through a link function. In other words, GAMs structures can be expressed as:

$$g(E(Y)) = \beta + f_1(x_1) + f_2(x_2) + \ldots + f_m(x_m)$$

Where:

- $g(E(Y))$ is the link function which associates the expected value with the predictive variables x_1 , x_2 , … , x_m.
- $f_1(x_1) + f_2(x_2) + \ldots + f_m(x_m)$ is the functional form with an additive series that generates the response variable Y .

2.3 Implementation and evaluation

When implementing ML tools, the selection of the input dataset for model training as well as the features to be used is crucial. For the implementation of the aforementioned methods, it is necessary to preprocess the acquired features through an Exploratory Data Analysis (EDA) from where different inferences can be drawn to ensure a selection of efficient features. After performing this EDA, it has been possible to verify a large presence of outlier data, as well as the existence of similar records, dependence between numerous features and low dependence of them with the response variable, among others. This assessment leads to the discarding of those features that could result in repetitions or that could have dependencies with others already calculated. After this preprocessing, a total of 84 features have been selected for the use of the models based on decision tree and the SVM.

Besides this filtering, algorithms based on logistic regression require a recompilation of features which are the most significant or provide the greatest contribution to the model. Thus, a selection has been carried out by discarding those variables with a high correlation between them. Furthermore, an own selection has been done taking those features which are considered the best significant option for a vulnerability prediction. After the analyses, a total of 48 features have been collected to be used in the model.

When it comes to the selection of the input dataset, and to avoid possible input data which could compromise the results, a balancing has been carried out, i.e., an approximately equal number of vulnerable and non-vulnerable samples have been selected. Likewise, and in order to make a complete evaluation of the models, two different scenarios have been created: one in which a set of paired data is provided, and another whose samples are unpair. For the former, a vulnerable and non-vulnerable version of each of the libraries has been chosen, while for the latter, the vulnerable and non-vulnerable samples correspond to different libraries. In the case of models GAM, the use of paired datasets does not provide satisfactory results, since they can lead to confusion. Therefore, the dataset used will be the one that, in addition to being balanced, has unpaired data.

To obtain the optimum results of the model, samples have been divided into those used for the training process and those used for testing. In addition, a grid search has been

implemented by cross validation for obtaining the hyperparameters corresponding to the best result of each model.

When implementing the different models, the use of the Python programming language has been chosen, since it includes specific libraries for performing their execution. In the case of RF, the public library *RandomForestClassifier*[12] is used. GBT is implemented by the library *GradientBoostingClassifier*[13]**.** SVM is configured by the use of the *SVC*[14] from the public library *sklearn*[15]. By last, for the implementation of linear generalized models two different methodologies have been applied: one based on the logistic regression model by the use of *linear_model.LogisticRegression*[16] included in the Python library *sklearn* and another by implementing the generalization to the additive case of the linear model, using *LogisticGAM()*[17] from the Python library *sklearn***.**

The proportion between the records obtained and variables, together with the conclusions aforementioned and the exploratory nature of the study presented, favor the possibility of evaluating several different data sets. These are the result of performing certain transformations such as the grouping of certain variables, the elimination of paired data, the suppression of certain categorical variables, etc.

In a first test, it has been taken as a dataset those libraries vulnerable and no-vulnerable to a known CVE, that is, it has been discarded those libraries and versions marked with provisional CVE or PVE. The results obtained in the different models with said dataset are not conclusive since the number of existing samples in it is not enough for a correct implementation of the selected models.

To solve the lack of samples in the dataset, it has been taken those libraries with PVE, considering them as vulnerable, as well as their no-vulnerable version. These samples have been evaluated in both scenarios paired data and unpaired data, except for the GAM models which nature only allows their use with unpaired data.

Table 1: Results with paired and unpaired data

Scenario	Model	Accuracy training	Accuracy testing	AUC
Paired data	**RF**	0.509	0.502	0.518
	GBT	0.602	0.579	0.539
	SVM	0.6	0.581	0.541
Unpaired data	**RF**	0.497	0.496	0.528
	GBT	0.532	0.558	0.569
	SVM	0.552	0.554	0.552
	GAM	0.521	0.523	0.498

For the evaluation of the results, it has been taken into account the accuracy training, accuracy testing and the AUC (Area Under the ROC Curve) whose optimal values have to be around the 0.7 and 0.9. In this scenario, the obtained results (Table 1) showed low adjustments in all the models and scenarios indicating that a data extension has not

[12] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[13] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
[14] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[15] https://scikit-learn.org/stable/user_guide.html
[16] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[17] https://pygam.readthedocs.io/en/latest/api/logisticgam.html

provided solutions in improving the modeling. When making a comparison between the different models and scenarios, there is no indication of a significant difference. In addition, having a low accuracy with the training data indicates that models are not being able to learn properly, which leads to consider performing an improvement of the problem context.

As a way of improvement of the model's results, it has been chosen to add a contextualization on the dataset. For this, the possibility of adding new features to the existing ones has been evaluated, which can characterize more accurately the response feature of the same. After various analyses, a reconfiguration of the output feature is performed.

The results obtained after the execution of the models with the new reconfiguration are those shown in Table 2. Since the results obtained with the paired and unpaired data do not show any difference, only those whose dataset are paired are shown, but the GAM models which only allows their use with unpaired data.

Table 2: Results with a reconfiguration of the output feature

Model	Accuracy training	Accuracy testing	AUC
RF	0.566	0.561	0.623
GBT	0.651	0.634	0.628
SVM	0.653	0.631	0.535
GAM	0.642	0.564	0.573

After the obtention of the results, it can be seen that, in general terms, all the models present a slight improvement after the response reconfiguration. Likewise, the trend in the variation within the training test settings is maintained. However, the differences between the models are still not significant, and the unpaired scenario presents a weaker improvement regarding this information.

During the process of reformulating the problem, the possibility of modifying the response feature is explored taking into account the different types of CVE contained in the libraries. Since a certain library can have several different vulnerabilities, the possibility of obtaining a new dataset is contemplated, in which the records corresponding to a certain library appear as many times as different CVEs contain.

In this way, a new dataset has been generated which includes all the original libraries considered no-vulnerable, and those vulnerable to a given CVE, discarding those with PVE label. Due to the fact that data samples are paired, GAM is not performed as its use confuses the model giving unsatisfactory results. This dataset contains a total of 1613 records.

Table 3: Results considering the number of CVE per library

Model	Accuracy training	Accuracy testing	AUC
RF	0.754	0.727	0.776
GBT	0.751	0.702	0.719
SVM	0.653	0.621	0.637

The execution of the algorithms with the new dataset provides results in which a significant improvement can be verified with respect to the previous one (Table 3). In this case, the evaluated models present an improvement in the adjustment reached around 75%, being the model RF the one that provides the best results. The evaluation

of the performance of said model offers a confusion matrix with false positive and false negative rates greater than 10% (Figure 7).

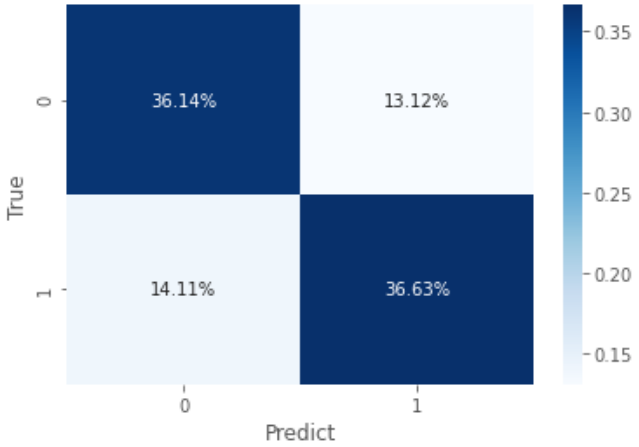

Figure 7: Confusion Matrix for the RF model

In addition, the obtained report details other representative measures such as *precision* (number of true positives that are actually positive compared to the total number of predicted positive values), *recall* (number of true positives that the model has classified based on the total number of positive values) and *F1 score* (harmonic mean of the above) (Table 4), which values are around 70% in both precision and recall.

Table 4: Descriptive Parameters of the Results for the RF model

Outputs	Precision	Recall	F1-score	% Support
0	0.72	0.754	0.727	0.493
1	0.74	0.653	0.621	0.507

By focusing on the ROC curve (Figure 8) and the AUC, it is possible to observe a summary of the model's performance, which is close to 80%.

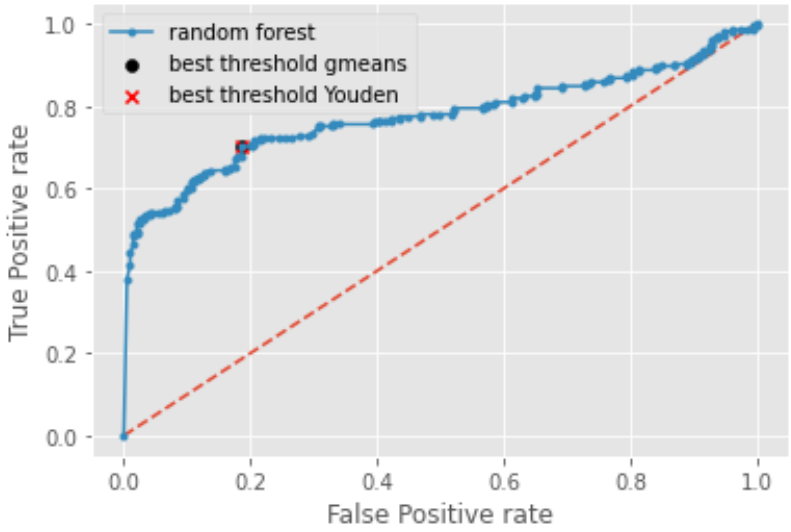

Figure 8: ROC curve for the RF model

As a result of the different transformations carried out to improve the evaluations, it has been verified that the results do not depend to a great extent on the chosen model, since they provide similar results between them. Moreover, adding records to the sample does not provide relevant improvements. Only by adding context to the problem and recoding the response feature provides slight improvements to the results.

3. Vulnerability Forecasting

In order to evaluate how a detected vulnerability can affect a system, from the context of BIECO, it is proposed the implementation of tools focused on the prediction of different vulnerability aspects. With it, a more complete security assessment and risk identification within the ICT system is provided.

One of the first steps when it comes to a forecasting analysis is to concretize which parameters are going to be predicted. When it comes to vulnerabilities, it is important to predict those aspects that may be most relevance when compromising the security of the system under analysis. Under this premise, and taking into account the life cycle of a vulnerability [10], it has been stipulated to forecast three types of vulnerability aspects: the probability of a vulnerability to be exploited in a selected time window, the number of vulnerabilities that may appear in the next period of time in a given system, and how severe a vulnerability may be. To this end, three different tools are going to be developed, that will focus on each of the previously mentioned approaches.

3.1 Exploitability Forecasting

In an initial phase, a tool is going to be developed to predict if a vulnerability will be exploited in a window time period, such as 3, 6, or 12 months. The selection of this period has been set according to the different information provided by the review of the state of the art [2] centered in the exploitability forecasting and by the results obtained in the training step of the ML model.

The following subsections detail the dataset under analysis and the academic description of the model that will be implemented to obtain the probabilities. The idea aims to improve the proposal of Jacobs et al. [11] by expanding the set of features with the contribution of the information provided by social networks about vulnerabilities.

3.1.1 Features

As mentioned in previous deliverables ([2], [3]) a selection of different features is evaluated in order to obtain the best results. These features are taken from different public data sources, which provide details about vulnerability disclosures, proof-of-concept (PoC) exploits and vulnerability descriptions.

For the obtention of features regarding vulnerability information, the Data Collection Tool (DTC[18]) developed in T3.2 is used. This tool is a web application which stores information from relevant vulnerability datasets. The information offered by the DCT, is taken by public repositories such as the National Vulnerability Database (NVD[19]), MITRE[20] and Exploit Database (EDB[21]) among others. Thanks to the information provided by the DCT, different features are obtained, which will be taken into consideration in the development of the forecasting tool. Some of these features are: CVE identificatory, CVE publication date, CVE description.

[18] https://dct.bieco.org/
[19] https://nvd.nist.gov/
[20] https://cve.mitre.org/
[21] https://www.exploit-db.com/

Another public data source to be used in the acquisition of vulnerability information features and particularly regarding its exploitability, is EDB. This is a web repository for exploits and proof-of-concepts where penetration testers and vulnerability researchers upload application vulnerabilities and its corresponding vulnerable software to exploit them. Some of the information provided by EDB useful for the creation of the forecasting tool, is the CVE associated with a given exploitability, as well as the discovered date.

As a complementation of the previous public repositories for the obtention of useful data to predict the exploitability, it has been studied the possibility of the use of public data sources. In previous works [12], the social network Twitter has been proposed to be used to obtain various useful information. Following the same path, for the development of the tool, queries will be made to different social networks in order to obtain useful data in the forecasting of exploits. The required information is acquired by parsing different specific channels with topics related to cybersecurity, counting the occurrences keywords for a given vulnerability within different time windows.

3.1.2 Logistic regression models

In a first assessment, for the implementation of the tool, the possibility of implementing algorithms based on logistic regression models is considered, since these allow explaining the contribution offered by the predictor features on the behavior of the response variable.

A linear regression model predicts the value of the response feature using the equation (E1):

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i \quad \text{(E1)}$$

Where:

- y_i is a specific observation i
- β_0 is the intercept
- β_p represents the partial regression coefficients
- ϵ is the residue or error

In the case of binary logistic regression, the objective is to model the probability of a binary qualitative variable as a function of one or more independent variables. To do this, the value returned by the linear regression is transformed with a function whose results are always between 0 and 1. One of the most used functions is the logistic function, well known as sigmoide function (E2)

$$sigmoide = \sigma(y) = \frac{1}{1+e^{-y}} \quad \text{(E2)}$$

Substituting the y of the previous equation (E2) by the linear function (E1), the general equation of the logistic regression is obtained:

$$P(y = 1\ /X = x) = \frac{e^{\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p}} \quad \text{(E3)}$$

in which $P(y = 1\ /X = x)$ is the probability that the response feature y is equal to 1, given the predictors $x_1, \ldots, x_p$.

The resulting model has the regression coefficient in the exponents, which implies that it is not a linear model and, therefore, the strategy to fit the model will also be different from that of the linear model.

With this, it can be affirmed that the objective of logistic regression is not to predict the value of the variable from one or several predictor variables, but to predict the probability that it occurs knowing the variable values.

3.2 Vulnerability Forecasting

When it comes to vulnerabilities, forecasting the number of them that will appear in the next period of time is an important information for various managerial decisions [13],[14]. The forecasting of vulnerabilities deals with the main system software components used within a complex ICT system. The results represent a measure of the ICT system level of trust, since each vulnerability that affects a major component, can represent a vulnerability of the entire system that can be exploited by a malicious attack. The forecasting tools developed in T3.3 will provide an estimation of the number of vulnerabilities to be expected for the main system software components in certain time frames.

The vulnerability forecasting tool uses artificial neural network models which are made up of a set of interconnected components, called neurons or perceptrons. The neurons are placed in several layers. The inputs of each neuron are connected to the outputs of the neurons in the previous layer. The neurons in the first layer get the inputs of the model, and the outputs of the neurons in the final layer provide the result.

The structure of a neuron is presented in Figure 9, where $x_i, i = 1,..,n$ are the inputs, $w_i, i = 1,..,n$ are the weights associated to the inputs, b is the bias, f is the activation function and y is the output.

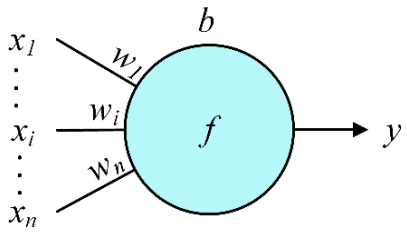

Figure 9: Perceptron structure

The neuron operation is defined by relation (E4). It calculates a weighted sum of the inputs and bias, and then it passes it through the activation function to determine the

output. The most widely used activation functions are *Identity* (E5), *Sigmoid* (E6), *ReLu* (E7), *Tanh* (E8) and *Softplus* (E9), which are shown in Figure 10.

$$y = f\left(b + \sum_{i=1}^{n} w_i x_i\right) \tag{E4}$$

$$f_1(x) = x \tag{E5}$$

$$f_2(x) = \frac{1}{1 + e^{-x}} \tag{E6}$$

$$f_3(x) = \max(0, x) \tag{E7}$$

$$f_4(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{E8}$$

$$f_5(x) = \ln(1 + e^{x}) \tag{E9}$$

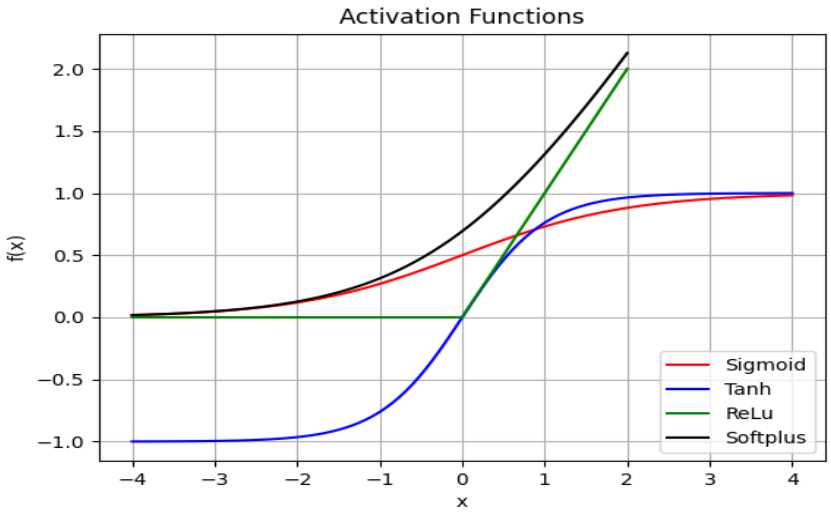

Figure 10: Common activation functions

3.2.1. Multilayer Perceptron Models

A Multilayer Perceptron (MLP) model with an input layer, a hidden layer, and an output layer is shown in Figure 11. The outputs of the model are given by relations (E10) and (E11), where f is the activation function of the output layer neurons and g is the activation function of the hidden layer neurons.

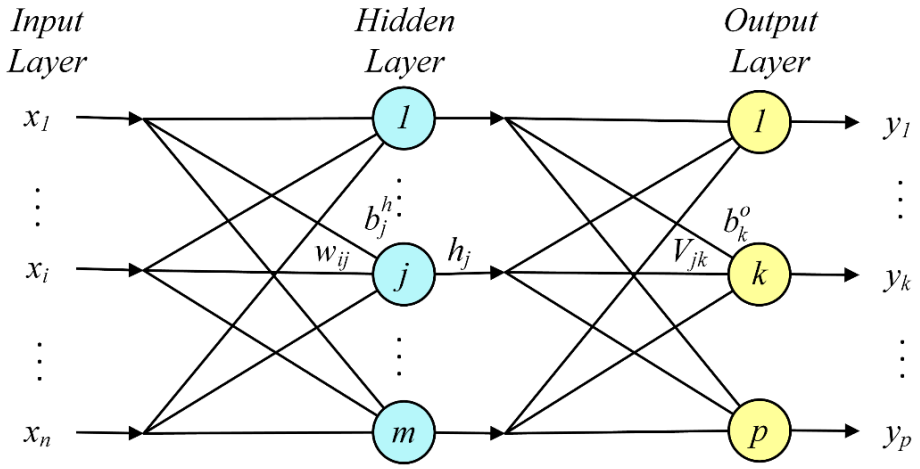

Figure 11: MLP model with one hidden layer

$$\mathrm{h}_j = g\left(b_j^h + \sum_{i=1}^{n} W_{ij}x_i\right), j = 1, \dots, m \quad \text{(E10)}$$

$$\mathrm{y}_k = f\left(b_k^o + \sum_{j=1}^{m} V_{jk}h_j\right), k = 1, \dots p \quad \text{(E11)}$$

A more compact representation of this model is given in Figure 12, where x and y represent the input and the output vectors, W and V are the hidden and the output layers weights matrices, b^h and b^o are the hidden and the output layers bias vectors, h is the hidden layer output vector, and g and f are the activation functions of the hidden and the output layers. The output is given by relations (E12) and (E13), that are compact representations of (E10) and (E11). The model can be extended by adding more hidden layers. Training such a model involves adjusting the weights and biases in such a way as to provide an output as close as possible to the correct value for each input. The most widely used methods of training Artificial Neural Networks (ANNs) are based on the Gradient Descent – Back Propagation (GD-BP) algorithms [15] or genetic algorithms.

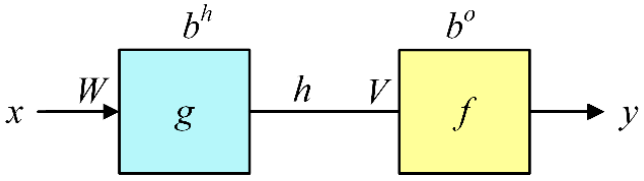

Figure 12: Compact representation of the MLP model with one hidden layer

$$h = g(b^h + xW) \quad \text{(E12)}$$

$$y = f(b^o + hV) \quad \text{(E13)}$$

3.2.2. Recurrent Neural Networks

Recurrent Neural Nets (RNNs) [16] were designed to process sequential data. They can memorize previous states and use them to determine the current state. The structure of a RNN is shown in Figure 13.

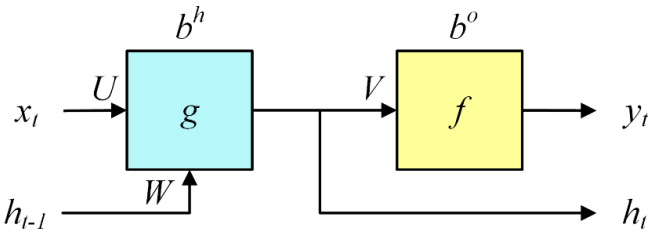

Figure 13: RNN structure

The operation of a RNN is shown in Figure 14. Its state is composed by the hidden layer output vector h. The state vector h calculated at step $t-1$ is processed as an entry at step t. The total number of trainable parameters does not depend on the number of steps, but only on the dimensions of the layers. Thus, for the network structure in Figure 13, the total number of parameters is $(n+m+1)m+(m+1)p$, were n is the size of the input vector x, m is the size of the status vector h, and p is the size of the output vector y. The outputs at step t are given by relations (E14) and (E15). They depend not only on the entries at step t, but on their entire evolution, starting from the initial step.

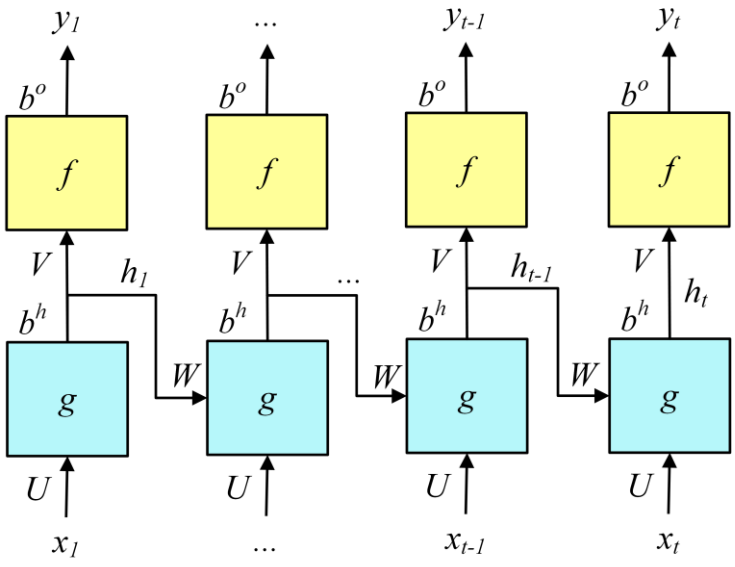

Figure 14: RNN operation

$$h_t = g(b^h + x_t \mathrm{U} + h_{t-1} \mathrm{W}) \quad \text{(E14)}$$

$$y_t = f(b^o + h_t \mathrm{V}) \quad \text{(E15)}$$

3.2.3. Long Short-Term Memory Models

The main disadvantage of RNNs lies in the fact that they cannot learn long sequences, because of the vanishing gradient problem [15]. This problem occurs in deep networks with many hidden layers. RNNs do not have long-term memory. The Long Short-Term Memory (LSTM) models have been projected to solve this problem. The structure of a LSTM cell is presented in Figure 15, where f is the forget gate, i is the input gate, g is the input updater, O is the output gate, C_t is the current cell state and h is the hidden state. Its operation is defined by relations (E16) – (E21).

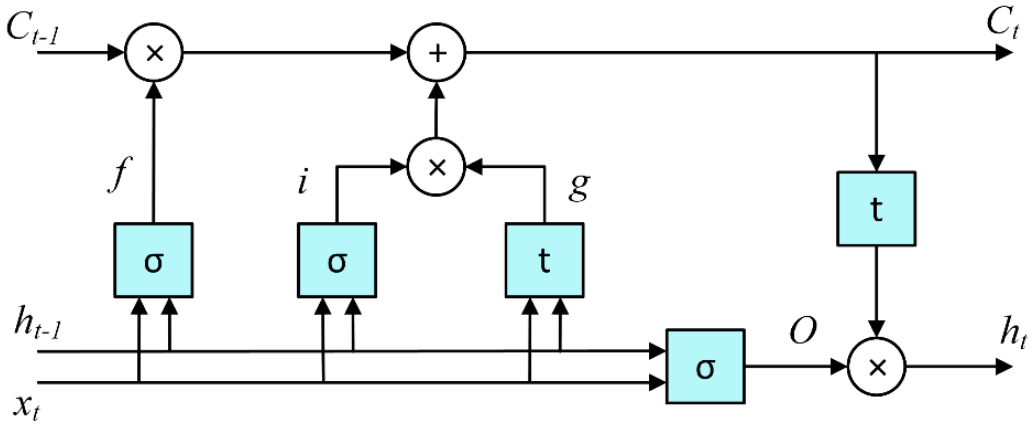

Figure 15: LSTM cell structure.

$$f = \sigma(x_t U^f + h_{t-1} W^f) \quad \text{(E16)}$$

$$i = \sigma(x_t U^i + h_{t-1} W^i) \quad \text{(E17)}$$

$$g = \tanh(x_t U^g + h_{t-1} W^g) \quad \text{(E18)}$$

$$C_t = C_{t-1} \circ f + g \circ i \quad \text{(E19)}$$

$$O = \sigma(x_t U^o + h_{t-1} W^o) \quad \text{(E20)}$$

$$h_t = \tanh(C_t) \circ O \quad \text{(E21)}$$

3.2.4. Dataset preparation and model training

One of the main software components used in the BIECO use cases is the UBUNTU operating system, produced by Canonical Ltd. The first variant of this operating system appeared in 2004 and its development continues to this day. Common versions of this operating system that were released before 2013 have an operating life of 18 months and are reviewed approximately every 6 months. Since 2013, the service life of these versions has been reduced to 9 months. Every fourth version of UBUNTU benefits from Long Term Support, for a minimum period of 5 years. The versions of this operating system are shown in Table 5.

Table 5: UBUNTU operating system versions.

Version	Name	Release date	End of support
4.10	Warty Warthog	2004-10-20	2006-04-30
5.04	Hoary Hedgehog	2005-04-08	2006-10-31
5.10	Breezy Badger	2005-10-13	2007-04-13
6.06 LTS	Dapper Drake	2006-06-01	2011-06-01
6.10	Edgy Eft	2006-10-26	2008-04-25
7.04	Feisty Fawn	2007-04-19	2008-10-19
7.10	Gutsy Gibbon	2007-10-18	2009-04-18
8.04 LTS	Hardy Heron	2008-04-24	2013-05-09
8.10	Intrepid Ibex	2008-10-30	2010-04-30
9.04	Jaunty Jackalope	2009-04-23	2010-10-23
9.10	Karmic Koala	2009-10-29	2011-04-30
10.04 LTS	Lucid Lynx	2010-04-29	2015-04-30
10.10	Maverick Meerkat	2010-10-10	2012-04-10
11.04	Natty Narwhal	2011-04-28	2012-10-28
11.10	Oneiric Ocelot	2011-10-13	2013-05-09
12.04 LTS	Precise Pangolin	2012-04-26	2019-04-26
12.10	Quantal Quetzal	2012-10-18	2014-05-16
13.04	Raring Ringtail	2013-04-25	2014-01-27
13.10	Saucy Salamander	2013-10-17	2014-07-17
14.04 LTS	Trusty Tahr	2014-04-17	2024-04-25
14.10	Utopic Unicorn	2014-10-23	2015-07-23
15.04	Vivid Vervet	2015-04-23	2016-02-04
15.10	Wily Werewolf	2015-10-22	2016-07-28
16.04 LTS	Xenial Xerus	2016-04-21	2026-04-23
16.10	Yakkety Yak	2016-10-13	2017-07-20
17.04	Zesty Zapus	2017-04-13	2018-01-13
17.10	Artful Aardvark	2017-10-19	2018-07-19
18.04 LTS	Bionic Beaver	2018-04-26	2028-04-26
18.10	Cosmic Cuttlefish	2018-10-18	2019-07-18
19.04	Disco Dingo	2019-04-18	2020-01-23
19.10	Eoan Ermine	2019-10-17	2020-07-17
20.04 LTS	Focal Fossa	2020-04-23	2030-04-23
20.10	Groovy Gorilla	2020-10-22	2021-07-22
21.04	Hirsute Hippo	2021-04-22	2022-01-20
21.10	Impish Indri	2021-10-14	2022-07-14
22.04 LTS	Jammy Jellyfish	2022-04-21	2032-04-21

Figures 16 and 17, obtained from the analysis carried out in the vulnerability forecasting tool, show the vulnerabilities discovered for LTS releases of the UBUNTU operating system. The red portion represents vulnerabilities that were discovered before the release date, and the green area represents vulnerabilities discovered after the end of the support period.

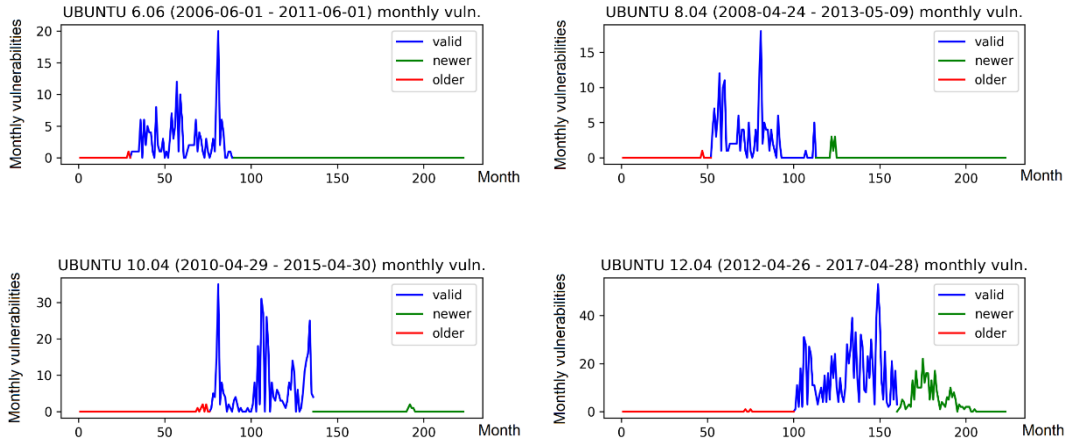

Figure 16: Monthly vulnerabilities of LTS UBUBTU releases

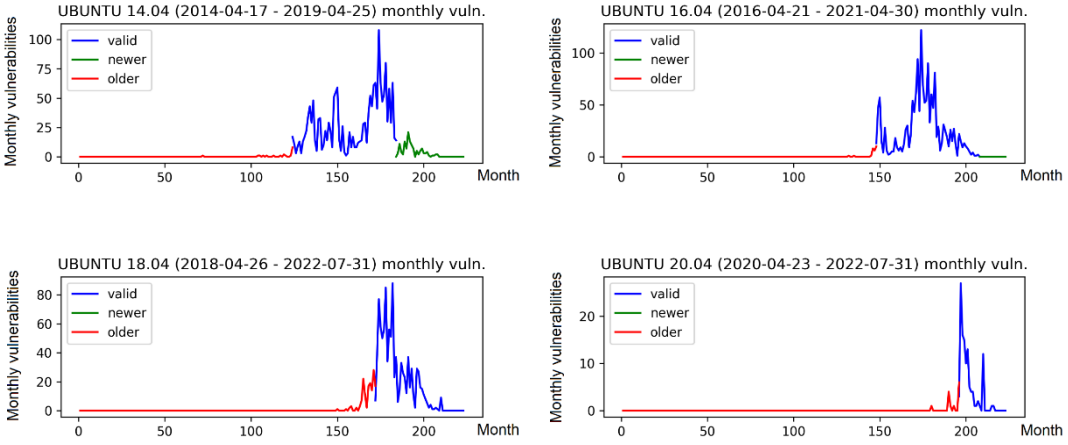

Figure 17: Monthly vulnerabilities of LTS UBUBTU releases

Figure 18 represents all the vulnerabilities discovered for the UBUNTU operating system, in the 206 months of its existence, since its launch, until now. The red line represents the age of the last release of UBUNTU, in months.

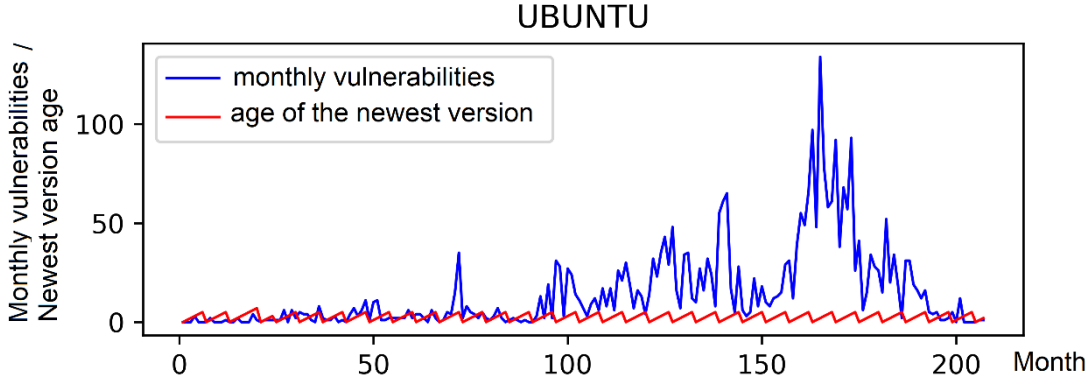

Figure 18: UBUNTU monthly vulnerabilities

3.2.5 Vulnerabilities Forecasting Tool

The vulnerability forecasting tool is based on LSTM models trained with the iterated k-fold cross validation technique. The models are equipped with dropout layers in order to avoid overtraining. The models' parameters were optimized using grid search. The inputs of the model are the history of UBUNTU vulnerabilities and the age of the newest version in months.

At this stage, the vulnerability forecasting tool contains four models for predicting the monthly number of vulnerabilities for the UBUNTU operating system, as well as the average number of vulnerabilities for 2, 3 and 6 months.

The results of the monthly vulnerabilities prediction model are presented in Figure 19. The red line represents the real number of vulnerabilities, and the blue one represents the forecasting tool prediction. The mean absolute error of the forecasts obtained for the test dataset is 6.12.

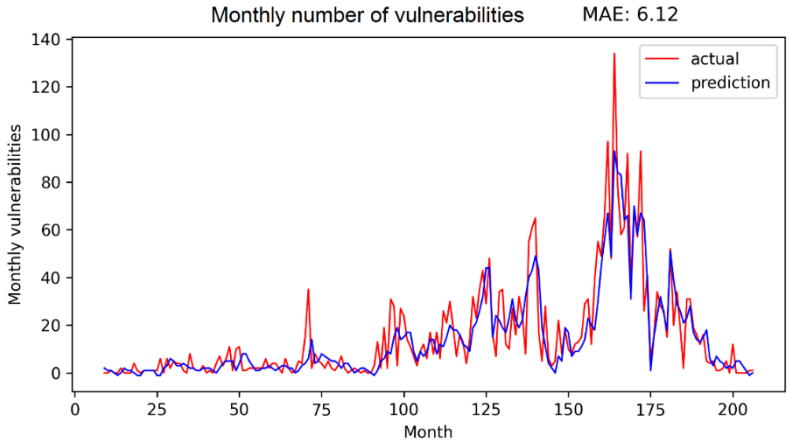

Figure 19: UBUNTU monthly vulnerabilities forecasting

The results for predicting the average number of vulnerabilities for 2, 3 and 6 months periods are presented in Figures 20, 21 and 22. The mean absolute errors obtained for the test dataset are 3.84, 3.03 and 1.87 respectively.

The red lines represent the actual data, and the blue lines represent the results obtained by the forecasting tool for each month, based on the information from the previous months. The horizontal labels represent the age of the studied component, which is at this stage the UBUNTU operating system. The first month (0) corresponds to the release of the first UBUNTU version (October 2004) and the last month (206) corresponds to the date on which the work on this document began (December 2021).

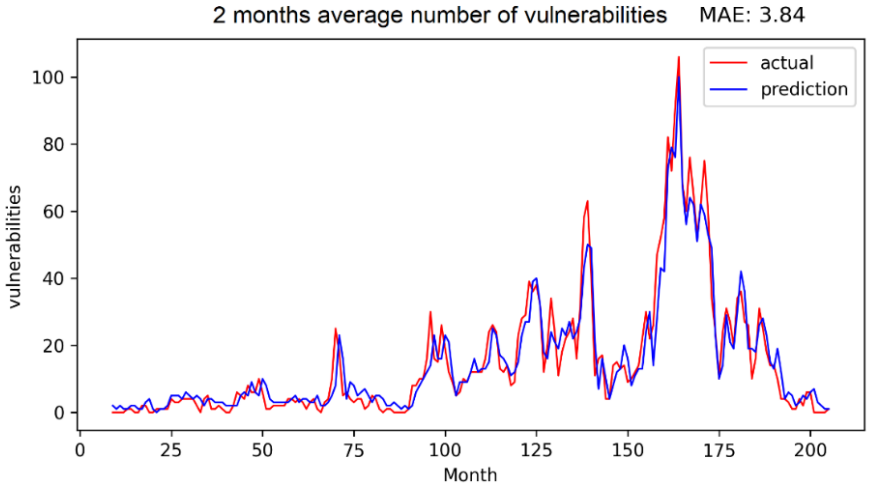

Figure 20: UBUNTU 2 months average number of vulnerabilities forecasting

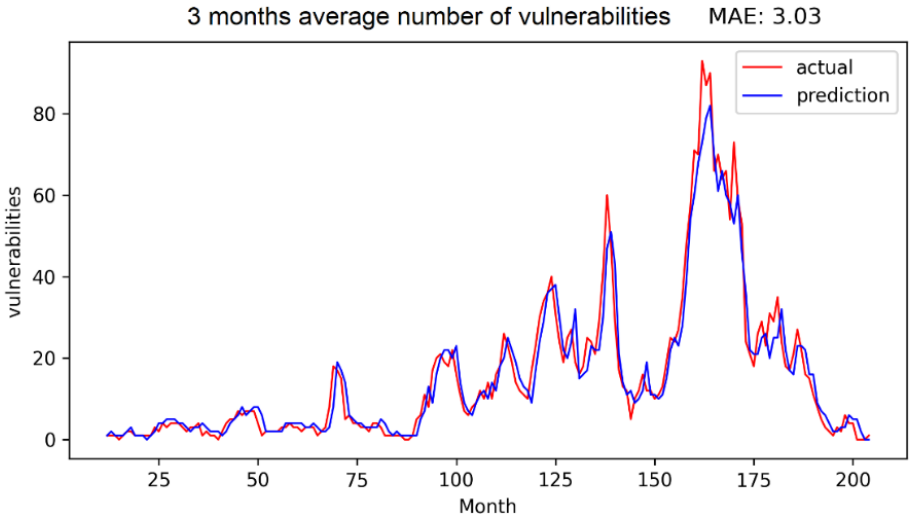

Figure 21: UBUNTU 3 months average number of vulnerabilities forecasting

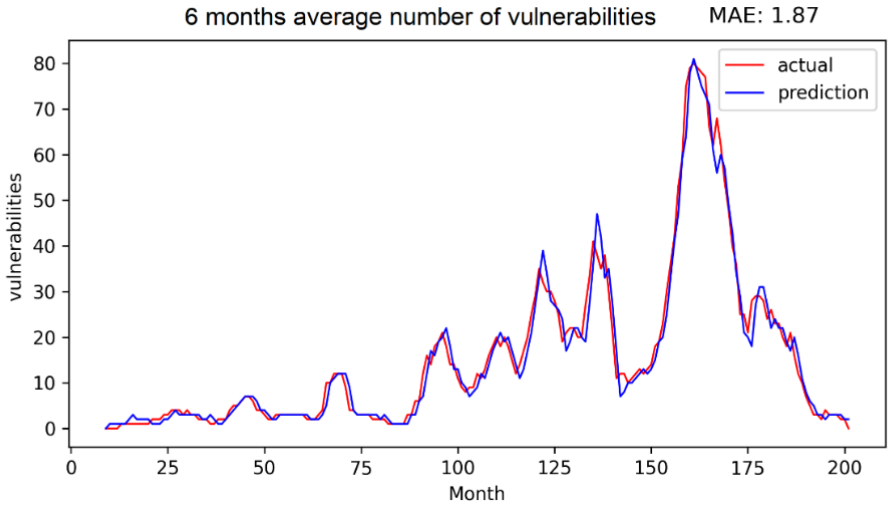

Figure 22: UBUNTU 6 months average number of vulnerabilities forecasting

3.3 Severity Forecasting

Vulnerabilities can be characterized by their severity which depends on the risk they may generate or the consequences caused by being exploited. For this reason, in addition to the prediction tools described above, the development of a new tool is considered which can offer a forecast of it. To achieve said prediction, it is intended to forecast the severity of newly discovered and related vulnerabilities, i.e., considering time correlation of related vulnerabilities among the already registered ones and comparing that to newly discovered ones. The model will be trained on a data set containing all previously discovered vulnerabilities. For this purpose, we use the NVD provided by NIST[22] (National Institute of Standards and Technology) of the United States. For training and evaluation, the input data set is split into three parts, training set, validation set and test set.

We consider novel algorithms for time series forecasting: ES-Hybrid and N-Beats, as the basis of our model. We also consider advances in neural network methods such as the Transformer architecture and Capsule Network.

3.3.1. Forecasting methods state of the art overview

One of the methods that presents significant progress in the state of the art of forecasting is ES-Hybrid method by Slawek Smyl, which is a hybrid machine learning and statistical method [17]. In the M4 competition [18], it outperforms the benchmark method for almost 10% of accuracy on 100 000 of time series, which is a really significant improvement to the current state of the art.

Another approach, which introduces novel concepts of machine learning for forecasting and has already overperformed the aforementioned method was presented in [19] and is called N-Beats. N-Beats method was designed to use deep learning for time series forecasting and proposes a deep neural architecture based on backward and forward residual links and a very deep stack of fully-connected layers. The architecture has a number of desirable properties, being interpretable, applicable without modification to a wide array of target domains, and fast to train. We discuss it further later in this document.

We also consider novel architectures of the neural networks' models, like Transformer architecture [20] and Capsule Networks [21]. These architectures displayed very good results in natural language processing and image recognition tasks, outperforming previous models for a solid margin. The usage of these architectures for forecasting is not researched yet fully. The first experiments show very promising results [22] [23]. The latest method evaluated and reviewed is Temporal Fusion Transformer [24], considered the current state of the art in the field of forecasting and based on the Transformer architecture. It combines high-performance multi-horizon forecasting with interpretable insights into temporal dynamics. We discuss it further later in this document.

Based on the above review, we plan to use two methods: N-Beats and Temporal Fusion Transformer as forecasting algorithms in our tool.

[22] https://nvd.nist.gov/

3.3.2. N-Beats method

The N-Beats [19] architecture design has certain assumptions. The base architecture is simple and generic, yet expressive (deep). It does not rely on timeseries-specific feature engineering or input scaling and is extendable for interpretability.

The basic building block is a multi-layer fully-connected network with ReLU (Rectified Linear Unit) nonlinearities. It predicts basis expansion coefficients both forward and backward. Blocks are organized into stacks using doubly residual stacking principle. Forecasts are aggregated in hierarchical fashion. This enables building a very deep neural network with interpretable outputs.

N-Beats proposes a novel, hierarchical, doubly-residual topology depicted in Figure 23. There are two residual branches, one running over backcast prediction of each layer and the other one is running over the forecast branch of each layer. The figure shows the architecture in detail. There are *M* stacks each of which is made of *K* blocks while each block has 4 common, fully-connected (FC) layers and another, separate, fully-connected layer for backcast and forecast. The inputs and outputs are passing through stacks, blocks and FC layers as shown. The initial stack (stack 1) receives the original data for lookback (lookback period/window). The forecast output of all stacks is combined to obtain the global forecast (model output). The backcast output of a stack (also called its residual) is passed to the next stack as its input. The inputs to blocks, except the first one which receives the stack input, are composed of the backcast of the previous block and the input of that previous block.

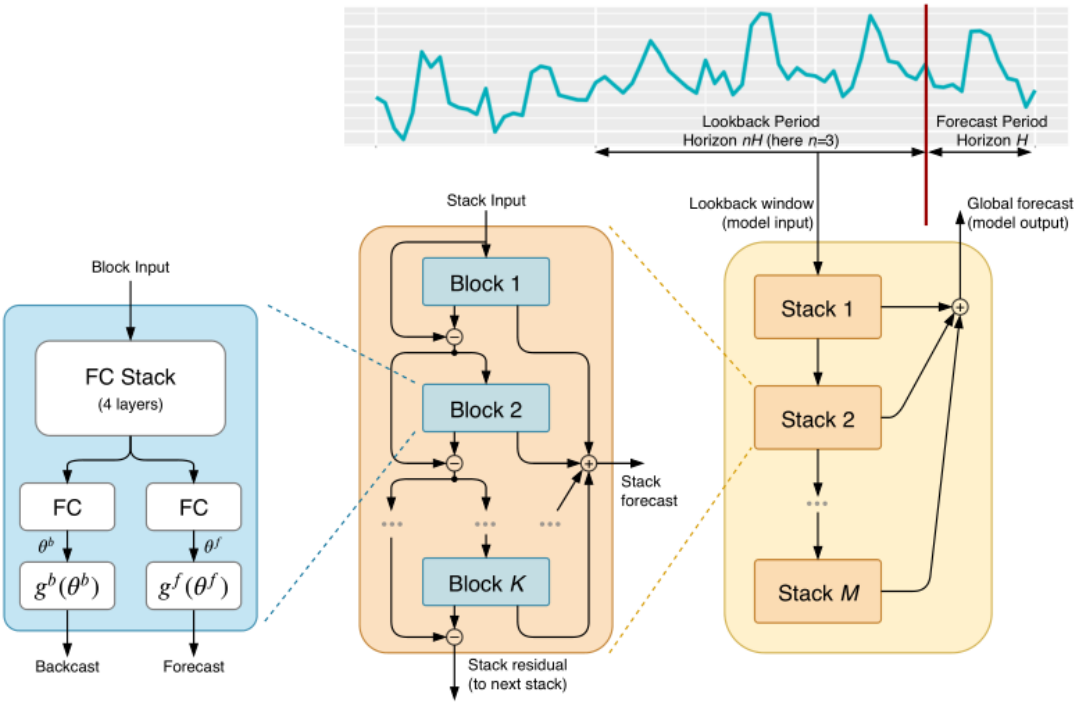

Figure 23: N-Beats architecture [19]

3.3.3. Temporal Fusion Transformer method

The Temporal Fusion Transformer (TFT) [24] is a novel attention-based architecture that combines high-performance multi-horizon forecasting with interpretable insights into

temporal dynamics. To learn temporal relationships at different scales, TFT uses recurrent layers for local processing and interpretable self-attention layers for long-term dependencies. TFT utilizes specialized components to select relevant features and a series of gating layers to suppress unnecessary components, enabling high performance in a wide range of scenarios.

"*To obtain significant performance improvements in state-of-the-art benchmarks, TFT introduces multiple novel ideas to align the architecture with the full range of potential inputs and temporal relationships common to multi-horizon forecasting – specifically incorporating (1) static covariate encoders which encode context vectors for use in other parts of the network, (2) gating mechanisms throughout and sample-dependent variable selection to minimize the contributions of irrelevant inputs, (3) a sequence-to-sequence layer to locally process known and observed inputs, and (4) a temporal self-attention decoder to learn any long-term dependencies present within the dataset. The use of these specialized components also facilitates interpretability; in particular, TFT enables three valuable interpretability use cases: helping users identify (i) globally-important variables for the prediction problem, (ii) persistent temporal patterns, and (iii) significant events.*" [24].

On a variety of real-world datasets, TFT demonstrates significant performance improvements over existing benchmarks.

Detailed architecture of Temporal Fusion Transformer architecture is presented in Figure 24. In particular, one can see that the basic building block in this architecture is a Gated Residual Network (GRN) – architecture of which can be seen in the upper right corner – with dense neural network layers, Exponential Linear Unit (ELU) activation functions, and a residual connection forwarding the input to the add and norm layer. All the input data goes through dedicated Variable Selection Networks (their architecture is depicted in the bottom right corner). The applied input processing depends on the kind of input data – the Temporal Fusion Transformer allows for encoded static metadata to be fed into GRNs for static enrichment. The variable selection residual is also passed to the add and norm layer of the (Variable Selection, LSTM [Long Short-Term Memory] Encoder/Decoder) block and encoded static metadata is used also for LSTM Encoding and Decoding (directly in the first layer and then after the LSTM encoding/decoding process). The final thing to notice is that the residuals from decoding blocks are passed as far as the last add and norm layers.

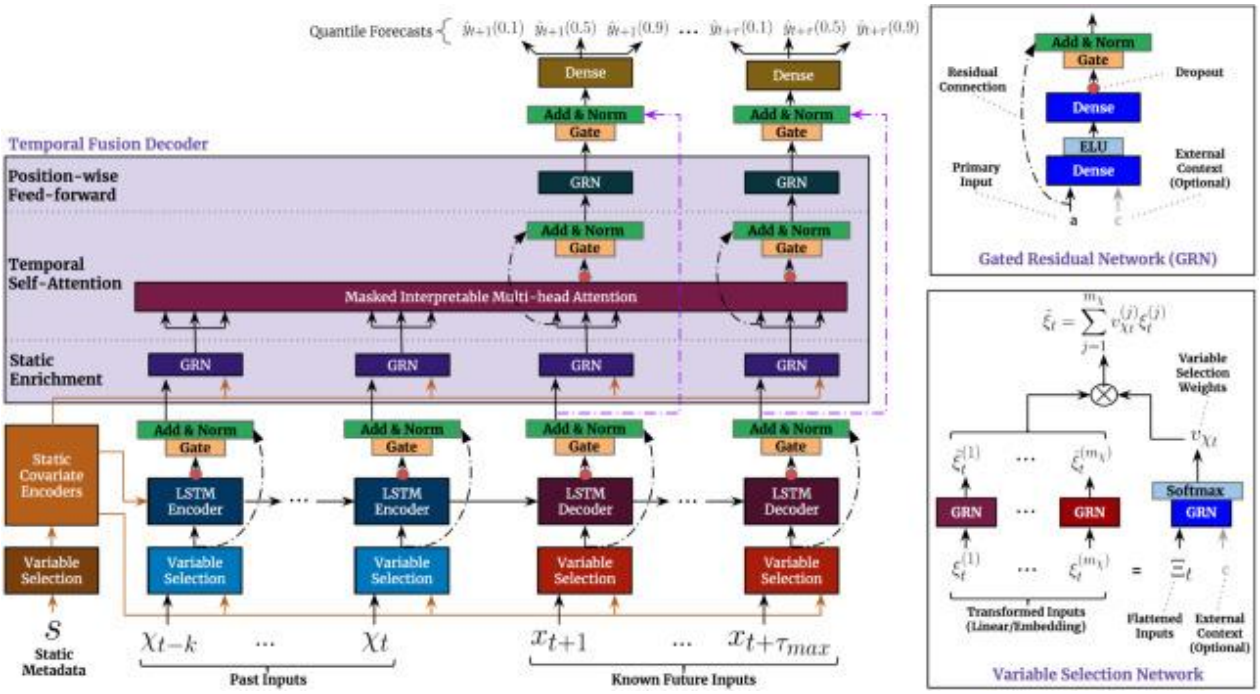

Figure 24: Temporal Fusion Transformer – architecture [24]

4. Conclusions and future actions

This deliverable has focused on presenting and overviewing different tools for an adequate vulnerability assessment.

First, a **vulnerability detection tool** has been presented which, for a first phase of study, will focus on the analysis of public libraries in Python language, to detect whether they are vulnerable or non-vulnerable. This deployment is breaking down in different stages: the obtention of the database and features which best represents the data to analyze, selection and study of different models and approaches to carry out, and the implementation and result of the different models executed.

The results obtained from different models under the same sample to be analyzed do not show a great difference between them, being RF models the ones that present a small improvement. In addition, it has been possible to observe the dependence of the results on the input features when reformulating the problem through their variation, obtaining considerable improvements in them. These results show how crucial the choice of variables is when developing the ML detection tools.

Despite having obtained decent results in the last analyzed scenario, the option of changing the current binary response feature (vulnerable/non-vulnerable) to a multiple one is contemplated as the next step in the development, this being the type of CWE detected. In this way, once the type of vulnerability is detected in a software component, its more specific location can be focused by using patterns techniques for a specific type of vulnerability. Furthermore, the study will be extended to the remaining programming languages, I.e., Java and C.

When it comes to the forecasting assessment, different tools have been presented: exploitability, vulnerability and severity forecasting.

For the development of the **exploitability forecasting tool**, several data sources have been explored for the obtention of the features to be used in the prediction: the deployed DCT for acquiring those features related with vulnerability descriptions, the public dataset EDB which provides exploits information such as dates and exploits id, and different social networks to obtain the day-by-day information. Carrying out an efficient parsing of the different sources to obtain the features is an essential step in the development, since these will be the ones that describe the model. Specifically, future actions will be focused on obtaining different features from the social networks Twitter and Telegram, as they are the least immediate to obtain. Likewise, the aforementioned logistic regression models will be implemented.

The **vulnerability forecasting tool** contains 4 models based on LSTM neural networks, which have been optimized and trained for the UBUNTU operating system. The training data was taken from the data collection tool, which is part of D3.2. [2]. Since there is relatively little training data, the iterated k-fold cross validation technique was used. To avoid the overtraining, it has been chosen the use of dropout layers. Likewise, the forecasting models have been optimized through grid search. As future lines of work, it is intended to continue to experiment with other optimization techniques, such as, for example, genetic algorithms. In the next stage of development, new models of vulnerability forecasting will be created, for other software components that are used in complex ICT systems. It is also aimed to experiment with expanding the models created so that they process new features obtained from the components developer and through feature engineering.

Finally, a new tool to be developed within the BIECO project has been presented: **severity forecasting tool**. The goal of this tool will be to predict the severity of newly discovered and related vulnerabilities. After analyzing the different existing methods in the state of the art, it has been planned to use two methods for the implementation of said tool: N-Beats and Temporal Fusion Transformer.

5. References

[1] G. McGraw, "Software Security: Building Security In," 17th International Symposium on Software Reliability Engineering, 2006, pp. 6-6.

[2] Deliverable D3.1 “Report on the State of the Art of Vulnerability Management”

[3] Deliverable D3.2 “Dataset with Software Vulnerabilities”

[4] H. Wickham and G. Grolemund. “R for Data Science: visualize, model, transform, tidy, and import data”. O’Reilly (2017).

[5] G. James, et al. “An introduction to statistical learning.” Vol. 112. New York: springer, 2013.

[6] S. Suthaharan. “Support Vector Machine”, in Machine learning models and algorithms for big data classification (pp. 207-235). Springer, Boston, MA. (2016)

[7] A. J. Dobson and A. G. Barnett. “An introduction to generalized linear models”. Chapman and Hall/CRC, 2018.

[8] M. J. Van der Laan, E. C. Polley and A. E. Hubbard. “Super learner”. Statistical applications in genetics and molecular biology, 6(1), 2007.

[9] T. Hastie and R. Tibshirani. "Generalized additive models: some applications." Journal of the American Statistical Association 82.398 (1987): 371-386.

[10] M. Shahzad, M.Z. Shafiq, & A.X. Liu. “A large scale exploratory analysis of software vulnerability life cycles”. In 2012 34th International Conference on Software Engineering (ICSE) (pp. 771-781), 2012

[11] J. Jacobs et al. “Exploit prediction scoring system (EPSS)”. arXiv preprint arXiv:1908.04856, 2019.

[12] H. Cheng, R. Liu, N. Park, and V. S. Subrahmaian, “Using Twitter to Predict When Vulnerabilities will be Exploited”, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019.

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

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

[15] Venkata Reddy Konasani, Shailendra Kadre: Machine Learning and Deep Learning Using Python and Tensor Flow. McGraw Hill: New York, 2021.

[16] A. Sherstinsky "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network." Physica D: Nonlinear Phenomena 404. 2020.

[17] S. Smyl, “A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting,” International Journal of Forecasting, 2019.

[18] S. Makridakis E. Spiliotis, and V. Assimakopoulos, "The M4 Competition: 100,000 time series and 61 forecasting methods." International Journal of Forecasting 36.1 (2020): 54-74.

[19] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, “N-beats: neural basis expansion analysis for interpretable time series forecasting,” arXiv preprint arXiv:1905.10437, 2019.

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

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

[22] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, et al., “Gluonts: probabilistic time series models in python”, 2019.

[23] J. Liu, X. Liu, H. Lin, B. Xu, Y. Ren, Y. Diao, and L. Yang, “Transformer-based capsule network for stock movements prediction,” in The First Workshop on Financial Technology and Natural Language Processing in conjunction with IJCAI 2019, p. 66.

[24] B Lim et al., "Temporal fusion transformers for interpretable multi-horizon time series forecasting." International Journal of Forecasting, 2021.

Annex A.

AST example

Next, a simple Python program is shown (Figure 25) along with part of the AST representation generated from it (Figure 26).

```python
def check_triangle(a: int, b: int, c: int):
    if a == b:
        if a == c:
            if b == c:
                return "Equilateral"
            else:
                return "Isosceles"
        else:
            return "Isosceles"
    else:
        if b != c:
            if a == c:
                return "Isosceles"
            else:
                return "Scalene"
        else:
            return "Isosceles"
```

Figure 25: Example source code of a function that checks the type of a triangle

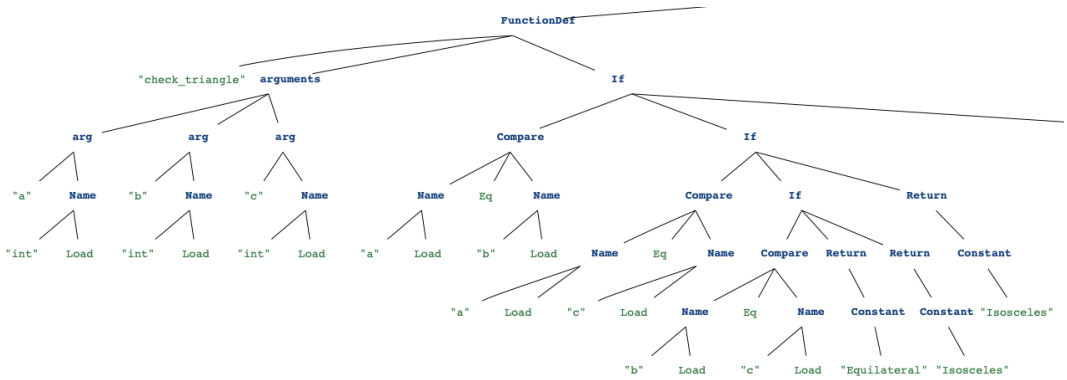

Figure 26: Part of the AST generated for the example in Figure 25