UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Malicious websites blocking system using Deep Learning algorithms

Project Thesis
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by Norma Gutiérrez

In partial fulfillment
of the requirements for the degree in
TELECOMMUNICATIONS ENGINEERING

Advisor: Beatriz Otero
Barcelona, Date: July, 2021

Revision history and approval record

Revision	Date	Purpose
0	15/04/2021	Document creation
1	29/04/2021	Document implementation
2	05/05/2021	Document implementation
3	13/05/2021	Document implementation
4	20/05/2021	Document implementation
5	25/05/2021	Document implementation
6	02/06/2021	Document implementation
7	07/06/2021	Document implementation
8	13/06/2021	Document first revision
9	14/06/2021	Document implementation
10	15/06/2021	Document implementation
11	16/06/2021	Document implementation
12	17/06/2021	Document revision
13	18/06/2021	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Norma Gutiérrez	norma.escobar@upc.edu
Beatriz Otero	botero@ac.upc.edu

Written by:		Reviewed and approved by:	
Date	15/04/2021	Date	17/06/2021
Name	Norma Gutiérrez	Name	Beatriz Otero
Position	Project Author	Position	Project Supervisor

Contents

List of Figures

List of Tables

Abbreviations

AI Artificial Intelligence

AUC Area Under the Curve

CNN Convolutional Neural Network

CSS Cascading Style Sheets

DL Deep Learning

DNS Domain Name System

FFNN Feed Forward Neural Network

FN False Negatives

FP False Positives

FPR False Positive Rate

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IP Internet Protocol

JS JavaScript

LSTM Long Short-Term Memory

ML Machine Learning

MVC Model View Controller

NG Next Generation

NN Neural Network

RBM Restricted Boltzmann Machine

ReLU Rectified Linear Unit

ROC Receiver Operating Characteristic

SVM Support Vector Machine

TLD Top Level Domain

TN True Negatives

TP True Positives

TPR True Positive Rate

URL Uniform Resource Locator

WP Work Package

Abstract

Malicious websites are currently one of the most severe threats to internet users. Traditional methods that detect malicious websites, such as blacklists, do not update frequently, and they cannot detect new attackers. To address this need, we propose a system that can be inserted into a firewall or Google Chrome extension. Starting with a dataset that contains both malicious and benign websites, we classify it by extracting numerous features. Furthermore, the features are parsed, analyzed, and preprocessed to create easily separable and categorized data. With the preprocessed data, the study proposes a Deep Learning (DL) model to classify each sample. The model consists of a Feed-Forward Neural Network (FFNN). We evaluate different combinations of neurons in the model and computes an in-depth study of the best performing network. Our results show up to 99.88% of detection of malicious websites and 2.61% of false hits (i.e. malicious websites classified as benign). Additionally, the system was tested with 10000 unseen websites, and the false hit rate decreased to 1.026%.

To approach the system to end-users, an interface is created to insert the suspicious URL and return the prediction output. The interface response time gives a mean response time of 2.53 seconds. This value is beneath the 4-second limit where the user's start to lose attention.

Overall, the proposed system can correctly classify both previously seen and unseen data, protecting thus against new attackers. Additionally, the interface creates a space where the user can quickly consult the URL maliciousness.

Resum

Les webs malicioses en aquest moment són una de les amenaces més grans que poden tenir els usuaris d'internet. Els mètodes tradicionals que detecten pàgines web malicioses, com les llistes negres, no s'actualitzen amb la rapidesa necessària per poder detectar amb precisió nous atacs a la web. Per a solucionar aquesta necessitat proposem un sistema que es pot afegir a un tallafoc o una extensió de Google Chrome. Començant amb un conjunt de dades (dataset) que conté llocs web malicioses i benignes, els classifiquem traient nombrosos atributs. Addicionalment, els atributs es formaten, s'analitzen i es processen prèviament per crear dades fàcilment separables i classificables. Amb les dades processades, l'estudi proposa un model d'Aprenentatge Profund (DL) per a classificar cada mostra. El model consisteix en una xarxa neuronal directa (FFNN). Avaluem diferents combinacions de neurones al model i realitzem un estudi en profunditat de la xarxa amb millor rendiment. Els nostres resultats mostren una precisió de detecció de pàgines web malicioses de fins al 99,88% i el 2,61% de detecció de fals positiu (és a dir, llocs web malignes classificats com a benignes). A part, el sistema ha estat provat amb 10.000 llocs webs no vistos prèviament i la detecció de fals positiu disminueix fins a l'1,026%.

Per fer més accessible el sistema als usuaris finals, es desenvolupa una interfície que a l'inserir una URL sospitosa retorna el resultat obtingut d'aquesta. El temps de resposta d'aquesta interfície dóna un valor mitjà de 2,53 segons, que és un valor inferior al límit de 4 segons en què l'usuari comença a perdre l'atenció.

En general, el sistema proposat és capaç de classificar correctament les dades vistes amb anterioritat i les que no. Per tant, protegeix contra nous atacants. A més, la interfície crea un espai on l'usuari pot consultar ràpidament la malícia de la URL.

Resumen

Hoy en día, las páginas web maliciosas son una de las mayores amenazas que pueden tener los usuarios de internet. Los métodos tradicionales para detectar páginas web maliciosas, como las listas negras, no se actualizan con la rapidez necesaria para poder identificar con precisión nuevos tipos de ataques en la web. Para solucionar esta necesidad proponemos un sistema que se puede añadir en un cortafuegos o en una extensión de Google Chrome. Empezando con un conjunto de datos (dataset) que contiene páginas web maliciosas y benignas, los clasificamos extrayendo numerosos atributos. Además, los atributos se formatean, se analizan y se procesan previamente para crear datos fácilmente separables y clasificables. El modelo consiste en una red neuronal prealimentada (FFNN). Evaluamos distintas combinaciones de neuronas en el modelo y realizamos un estudio en profundidad de la red con mejor rendimiento. Nuestros resultados muestran una precisión de detección de páginas web maliciosas de hasta el 99,88% y, un 2,61% de detección de falso positivo (es decir, sitios web malignos clasificados como benignos). El sistema ha sido probado con 10.000 páginas webs no vistas previamente y la detección de falso positivo disminuye hasta el 1,026%.

Para hacer más accesible el sistema a los usuarios finales, se desarrolla una interfaz gráfica que al insertar una URL sospechosa devuelve el resultado obtenido de esta. También se estudia el tiempo de respuesta de la interfaz. Este proporciona un valor medio de 2,53 segundos, que es un valor inferior al límite de 4 segundos en que el usuario empieza a perder la atención.

En general el sistema propuesto es capaz de clasificar correctamente los datos tanto vistos con anterioridad como los que no. Por tanto, protege de nuevos ataques. Aparte, la interfaz crea un espacio donde el usuario puede consultar rápidamente la malicia de la URL.

1 Introduction and Statement of purpose

Web pages may contain numerous types of attacks that target web browsers vulnerabilities. Malicious web pages have become one of the most common security threats. These attacks run malware in the target system, intending to take control of it. This project aims to design a system that blocks malicious website attacks by identifying possible malicious web pages. Moreover, the work parts from existing URLs and extracts relevant information from them. The big data treatment is later used to gather existing vulnerabilities and malicious websites used in real environments. It creates a Deep Learning (DL) model to detect new emerging websites even before they are listed in a blacklist database. DL techniques enable us to model complex computational architectures, such as websites features, to predict data representation. Consequently, the mechanism can be inserted in a firewall or web browser extension to recognize previously unseen malicious websites. Furthermore, the system is highly scalable, and it can be retrained and perfected for new emerging types of attacks.

We develop a defensive solution by implementing this mechanism, meaning that malicious software cannot penetrate the private network (i.e. blocked by a firewall). Overall, the results show high effectiveness when using only website features. Thus, the proposed Neural Network (NN) correctly differentiates between malicious and benign websites. The main contributions of this work are the following:

- To study the existing products similar to the project and stand out the advantages.
- Creation of three different datasets (training, testing and validation) parting from existing URLs and extracting data directly from the internet.
- Extract and preprocess raw website data differentiating the most important attributes.
- Develop a scalable DL NN that uses existing patterns in malicious web pages to detect malicious websites in real environments.
- Produce a system able to be integrated into a firewall or a browser extension.
- Creation of an interface to visualize the environment and control the internet's malicious activity.
- To create an affordable, quick and efficient system for users.

Overall, the system functioning is depicted in Figure 1. As shown, the system is divided into two main parts: the dataset creation and training of the model and the validation of the system and its detection. Firstly, the training and testing URLs are passed through feature extraction and labelling and are preprocessed, as explained later in the thesis. Furthermore, once the datasets are computed, an FFNN is created, and the data is passed to train and test the model's validity. The training process is fine tunned and repeated until a suitable NN is found. On the other hand, the second block parts from a validation set of URLs. Then, the same features as in the training and testing datasets are extracted, and the model is used to predict the sample's output. The actual label is stored in the validation dataset and compared with the predicted values to compute classification met-

rics. Lastly, the classification phase describes the process that an inputted URL from a user would follow. First, the user enters the URL, then the system extracts the necessary features and predicts the output from the URL (benign or malicious). Finally, the output is classified between a threshold into Malicious or Benign.

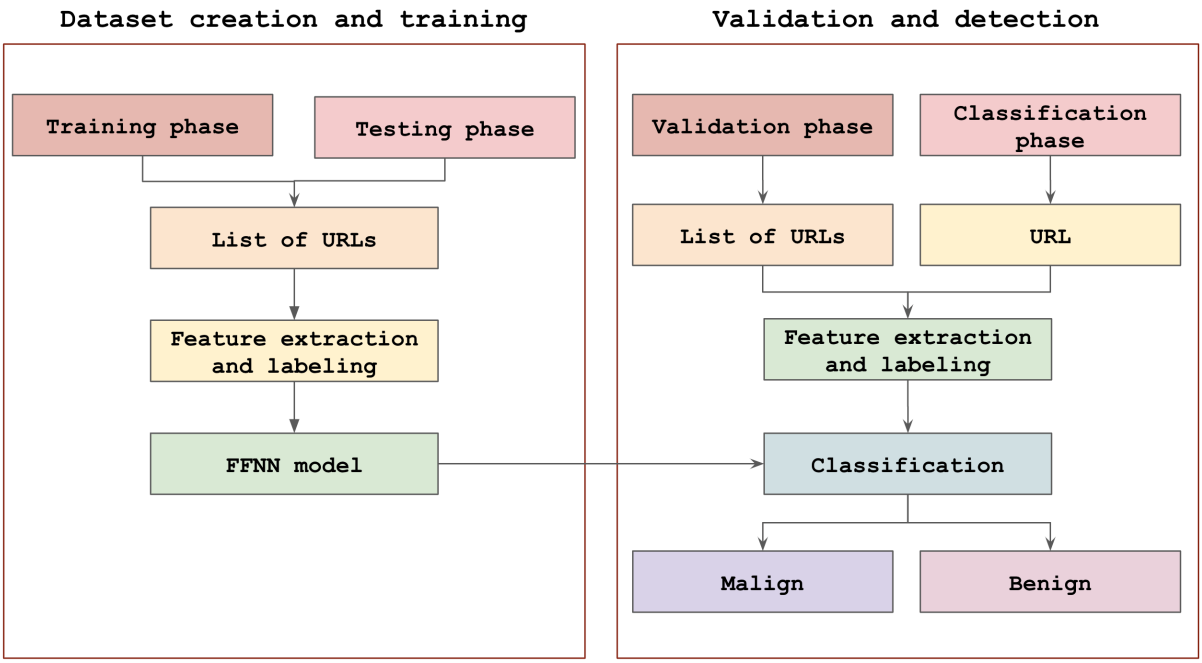


Figure 1: Block diagram of the system's functioning

The remainder of this project is organized as follows. Section 2 describes the context and problem definition, section 3 shows the work plan and the Gantt diagram. Section 4 presents the state of the art. Sections 5 and 6 describe the dataset and its preprocessing. Then, section 7 explains the used Neural Network, and section 8 describes the experiments and metrics used. Section 9 presents the results and section 10 explains the interface for the final classification phase. Additionally, section 11 describes the time limitations and the project's viability, while sections 12 and 13 show the budget and sustainability of this project. Finally, we draw the main conclusions and future work in section 14.

2 Context and problem definition

The created system treats the increasing necessity of blocking malicious or suspicious websites. Malicious content in web pages is a recurrent threat nowadays while browsing the internet. Thus, the project aims to create a system that can be downloaded into a user's computer. This system, given an URL, extracts and analyzes its features and classifies them into malicious or benign. Furthermore, the system is addressed to any user that wants secure navigation through the internet. Therefore, the importance of the product remains in keeping the users safe while navigating through the internet, avoiding malicious content into computers.

The idea of this project emerges from my curiosity about network attacks, cybersecurity and their detection. Moreover, I have previous knowledge of Artificial Intelligence (AI), and I wanted to continue that learning path. Thus, to combine network attacks with AI, particularly Deep Learning, a malicious website blocking device implemented with Deep Learning algorithms came to my mind (the author's) and was proposed. The idea is to develop from scratch the desired system that uses the URLs data extracted from the internet and predicts whether the website is malicious or benign. Finally, the project is independent of any other existing project since the project's main ideas were planned for this specific thesis.

2.1 Methods and procedures

To conduct the study, some methods and procedures were followed. Firstly, meticulous research was made to decide the thesis course of action (sections 3 and 4). We decided to separate the project into two parts: the dataset creation and training and the validation and detection. In the first part of the project, a dataset containing existent URLs was searched. The dataset needed to contain non-simulated data in order to work out of a simulated environment. Once the dataset was found, the first phase could commence.

To treat the data, several studies and similar researches were investigated. We started with a simple dataset preprocessing but realized that the selected features were feeble and not consistent enough to train a DL model. Hence, it was decided a new, improved dataset parsing, modifying almost entirely the existing dataset. Once the data was computed, new features were extracted, and further analysis (explained later on in the project) were performed. The analysis helped to decide which extracted features were necessary and optimize the dataset (sections 5 and 6).

Additionally, the DL model was created. To decide the different model parameters, we took into consideration the created dataset. Since the sample information consisted of a binary classification supervised learning dataset, an FFNN approach was selected. Furthermore, the model's parameters were chosen following the same approach (section 7). The final task of the first part of the project consisted of training, testing and saving the created model (sections 8 and 9).

The second central part of the project starts with the validation of the previously saved model (section 9). Here, by performing the validation with unseen data, the system's

efficiency can be easily seen and can be modified if needed. The validation process is the most critical part of the project since it shows the system's flaws, and its functioning is crucial to assure the system's viability. Then, for usability purposes, an interface was created, where a user can introduce an URL, and the interface will return its prediction output (section 10). Finally, some additional studies were performed to study the system's engagement, commercial outings, and sustainability (sections 11, 12 and 13). These final analyses helped us to evaluate the efficiency of the system and its future improvements.

3 Work plan

The project's work plan is divided into seven work packages. Firstly, thorough planning and research were made. During this process, related works were searched, and a course of action approach was developed. Once the project was planned and structured, the second work package (WP) consisted of the data treatment. In this part, a dataset was selected and modified so it can be used to train a DL model correctly. The data was analyzed with different techniques, and the most effective preprocessing was applied to the samples. The third work package is to implement the best NN for the data to predict the output correctly. Parting from the data, the best NN and parameters must be decided, and then several experiments must be conducted to fine-tune the model. Once the model is designed, the next task is to train and test the model. The fourth work package consists of the NN validation with unseen data and analyzes the final results. Next, in the fifth work package, the main task consisted of implementing a user-friendly interface where the user provides an URL, and the system returns the maliciousness of the website. Finally, a study of the time limitations of the project, its sustainability and its budget has been performed. The aim is to compute the system's efficiency, economic viability and engagement.

Overall the different tasks performed during the course can be seen in Figure 2. Also, note that apart from the denoted work packages (WP), an important task consisted of prettifying and fixing the code and writing all the necessary documents that the thesis required.

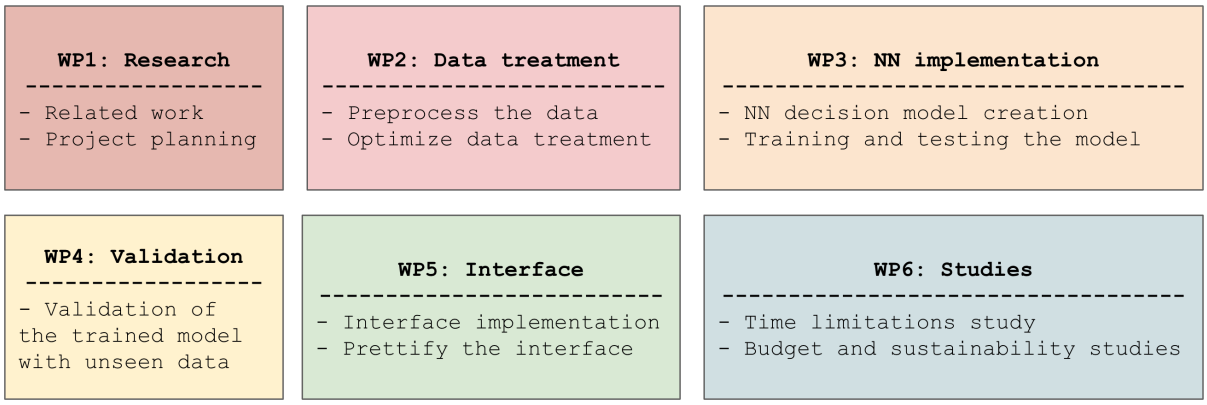


Figure 2: General view of the work packages performed through the thesis

An extensive description of these tasks as for their exact timeline is depicted in Annex A.1. All these tasks have been performed during the semester, as depicted in the Gantt diagram in Figure 3.

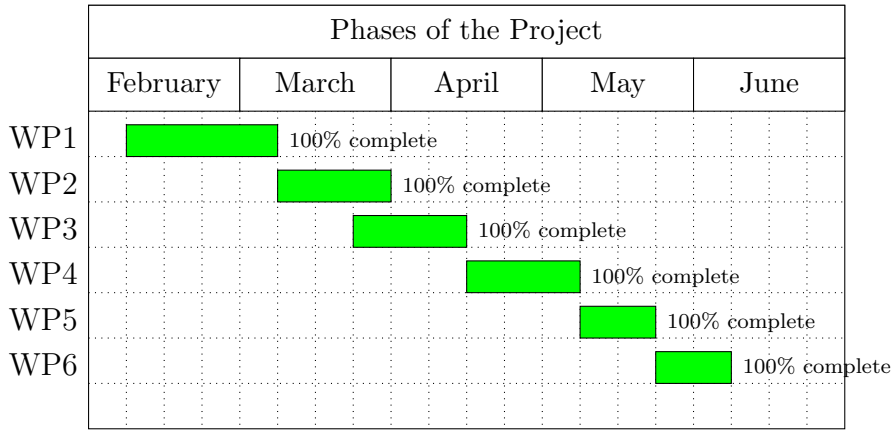


Figure 3: Gantt diagram

As shown in Figure 3, almost all tasks were sequential since all work packages required from the one before to be performed. Nevertheless, the dataset preprocessing and the NN design overlapped. This overlay is due to the dependence of the type of NN with its preprocessing, meaning that we had to decide the type of Artificial Intelligence we were going to use before completing the preprocessing process. Another overlap is the bug fix and the final thesis report with the final tasks. Some improvements were made to the code when performing the last work package studies, and all were instantaneously documented into the report.

4 State of the art

This section includes a comprehensive literature review that treats the most relevant and recent articles and works performed on the subject matter.

4.1 Related work

A firewall is a network security mechanism that controls incoming and outgoing network traffic. It decides whether to allow or block specific traffic based on a defined set of security rules. Furthermore, firewalls are a barrier between untrusted external networks and secured, controlled and trusted internal networks [1]. Thus, we aim to detect and block cyberattacks (i.e. an unauthorized attempt to harm someone's computer system or information using the internet). There are many types of cyberattacks such as endpoint attacks, malware attacks, system exploits, advanced persistent threats and network attacks [2]. Most studies focus on Network traffic and Malware. Network traffic analysis predicts whether the (network) packets are malicious or benign. On the other hand, malware detection classifies incoming traffic into different types of attacks. Their goal is to distinguish between several network attacks. These types of approaches are evaluated with simulated datasets. Other cybersecurity approaches are the study of WebApps, Software, Hosts, Email, Fraud, Honeypots, Binaries, Phishing, Password or MISC attacks.

Our proposal focuses on a non-simulated dataset where features from different websites are extracted (malicious and benign). Furthermore, our data comprises the URL and features associated with it, such as the associated continent and its JavaScript content length. These features are simple to obtain and simpler to treat than other attributes such as the HTTP or CSS content.

Chiba et al. [3] propose a system that can detect a malicious or benign website by only analyzing the IP characteristics. They create a dataset extracting campus traffic. The article preprocesses the data separating the address by bits and applies two different Machine Learning (ML) algorithms (SVM and RBM). They achieve a maximum of 90% accuracy. In contrast, our proposal uses DL techniques, and the dataset does not use IP information. Instead, it uses the URL and the JavaScript content length. Using more features enables us to make a more precise model than the one proposed by Chiba et al.

Moreover, Xuan et al. [4] uses the URL features to extract dynamic behaviours and train them with two supervised ML algorithms (Support Vector Machine and Random Forest). The differences with the presented system are that they use far more URL features than we do, expanding the computational needs of the network. Plus, they apply ML algorithms, whereas, in our system, DL techniques are applied. Finally, their performance is less than ours, having a 3% less accuracy, 4% less precision, and 1% less recall.

Saxe et al. [5] use HTML, CSS and embedded JavaScript (JS) files; they analyze the data and create a model. First, they pass the data through an SVM and then through a DL model. The model detects up to 97% of malicious traffic and can identify content not previously caught by the vendor community. Our approach reduces the data analyzed while also achieving a higher detection rate (i.e. instead of HTML, CSS and JS content, our proposal uses URL and IP information and the JS length).

Uçar et al. [6] develop two DL models (CNN and LSTM) that blacklist malicious URL. The models detect the type of data and classify it. It achieves up to 98.86% accuracy in the CNN network. The main distinctions between our approach and this paper are: (1) the type of Neural Network used. They use a more complex Neural Network. Thus their proposal uses more resources than our proposal. (2) their dataset only contains URL information, whereas our data also contains the continent and the JS length.

Another similar approach is presented by Johnson et al. [7]. The article presents the same problem as in this thesis; a binary classifier detecting if the URL is malicious or benign. Nevertheless, it adds a second multi-class classification that detects the type of attack. Overall, the article adds a further step to the classification to detect the type of attack but obtain a far more complicated NN and 2% less accuracy than in our system.

Finally, Sahoo et al. [8] propose a survey that gives a structural understanding of Malicious URL detection techniques using ML. The survey separates two distinct types of families when detecting Malicious URL: Blacklisting/heuristic approaches or ML techniques. The survey only talks about mathematical ML algorithms that previous literature has done. We advance the state-of-the-art presented in that survey by developing DL techniques to classify previously unseen data.

Our final approach uses DL to train the NN since the quantity of data enabled us to perform a model with outstanding performance. Furthermore, the chosen model was a FFNN, which was chosen since it is the most common approach for supervised learning with binary classification datasets. Also, a FFNN needs less computation than the other DL approaches mentioned above. Consequently, the resulting NN presents outstanding results. A comparison table with the presented related works and this project can be found in Annex A.2.

4.2 Related commercial products

Additionally to the research approaches, some commercial products offer an URL blocking system, such as the one presented in this project. The most significant and known is GoogleSafeBrowsing.

GoogleSafeBrowsing comes with Google Chrome Enterprise [9] and can be activated through its account configuration. It uses a URL Blocklist and a URL Allowlist to prevent or allow users to access the URL. Furthermore, the user can personalize their configuration.

WinTools URL blocker [10] is another commercial solution of our system, and it also makes some URLs inaccessible from the user's computer. Moreover, it works as a firewall, and the administrator has access to all the blocked sites and manually configure some restrictions. The system works by modifying the setting of the hosts' file and reconfiguring it. WinTools is a free solution, but it is only supported for Windows users.

Other solutions are DNS Filter [11] which is cloud-based and uses an AI_powered real-time threat detection with customizable URL filtering and off-network protection. A different approach is Cloudflare Gateway [12], which offers highly effective DNS filtering and other

technologies to keep internal employees secure.

Moreover, Cisco Umbrella [13] is one of the largest solutions and is also a cloud-based software. WebTitan Cloud [14] is a very scalable and DNS-based service, but its interface is only suitable for technical users. Finally, other systems that provide the same service are McAfee Web Gateway Cloud Service [15] and Untangle NG Firewall Complete [16].

All these approaches are priced and have to be hired by a company. Although our approach is a research-oriented project, having compared commercial products allows us to have a global vision of the actual market. Furthermore, some of the project's functionalities (such as the FFNN) could be sold to a depicted company.

5 Dataset

To create a DL model capable of distinguishing malicious websites, we modified the existing dataset *Visualisation of Malicious & Benign Web-pages Dataset (VoMBWeb)* [17] since the dataset was fitted to our solution but lacked consistency.

Thus, the system starts from a list of URLs captured from malign and benign websites and extracts the necessary information to be later trained and correctly classified. The features depicted in Table 1 were extracted from each URL crawling the internet.

Table 1: Samples feature extraction from the VoMBWeb dataset

Feature group	Feature	Description
URL	URL	Website's URL
URL	TLD	The Top Level Domain of the website
URL	HTTPS	Whether the website is HTTP or HTTPS
URL	Entropy	The URL's entropy
URL	URL length	The total URL length
URL	URL body length	The URL body length
URL	URL number of arguments	The count of the number of arguments in the URL
URL	URL path length	The URL path length
URL	URL letters length	The count of letters in the URL
URL	URL digits and symbols length	The count of digits and symbols in the URL
IP	IP address	The IP address associated to the URL
IP	Continent	The continent associated to the IP address
Content	Filtered content	The website's JavaScript filtered content
Content	JavaScript length	The content's length

As shown in Table 1, the data is extracted from three separate categories, meaning that the data is obtained through the IP, the URL and the content. The primary group is the URL group, parsed and treated as shown in Figure 4.

The first part of the URL is the protocol. The *HTTPS* feature is extracted; in the example, we have a *https* value, so the value of the feature is 1. Secondly, the body is computed where two different features are extracted, its length (*url_body_len*) and the Top Level Domain (*tld*). The third part of the URL includes the arguments where thy are quantified (*url_num_args*) and their length (*url_digits_len*) is computed. Finally, to treat the URL globally, the URL's length (*url_len*), the number of digits and symbols that it contains (*url_digits_length*), and the number of letters (*url_letters_len*) are computed.

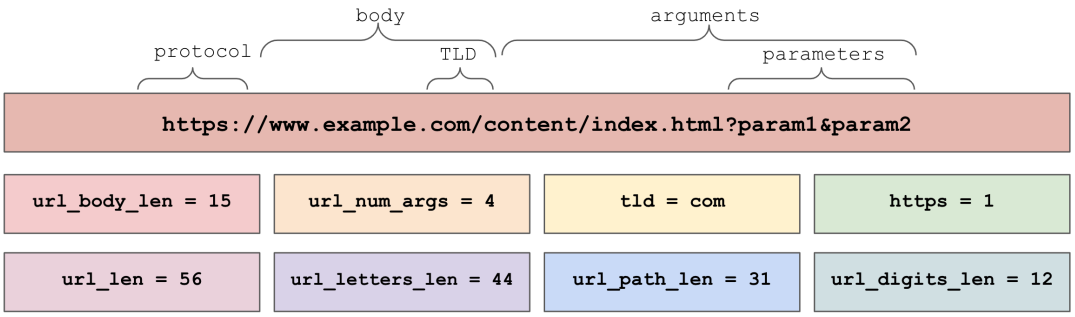


Figure 4: Example of URL preprocessing

The next group is the IP. Firstly the IP associated with the URL is extracted (*ip_addr*) as the continent where the IP address is located (*continent*). Since more than 300 countries were initially defined in the dataset, the values were grouped by continent. Furthermore, treating each country individually does not give any additional information than by doing it by continent.

Once all the IP and URL features are extracted and treated, the web site's content is withdrawn. The obtained content is the one located inside the JS code. Furthermore, the content was filtered to remove spaces, code, and punctuation. Once all was computed, the cleaned JS content was saved (*content*), such as its length in KBytes (js_len).

The final feature inserted in the dataset is the label, which has a binary value, depending on if the website is malicious (1.0) or benign (0.0). Therefore, the obtained data is non-simulated and belongs to real IP addresses and web pages. Hence, the created dataset has a total of 15 features.

Moreover, the dataset is divided into two parts, a training dataset (containing 1200000 samples) and a testing dataset (containing 350000 samples). In the full dataset, 27253 websites (values) are considered malicious, while 1172747 are considered benign, having far more benign websites than malign ones. In the testing dataset, the same happens with 7828 malicious samples and 342172 benign samples. Hence the dataset is mainly represented by benign websites, such as illustrated in Table 2.

Table 2: Samples and percentage of benign and malicious websites in the dataset

VoMBWeb	**Benign websites**	**Malicious websites**
Number of samples	1514919	35081
(%)	97.74%	2.26%

6 Preprocessing

This section explains the treatment and preprocessing of data samples parting from the 15 initial values proposed in the previous section. Since the dataset comprises extracted data from a physical environment, the preprocessing must be meticulously designed to extract the best information from the given data. We analyzed each of the 15 features thoroughly and tested it before deciding on a certain technique.

The dataset contains eight numerical features (*entropy*, *url_len*, *url_body_len*, *url_num_args*, *url_path_len*, *url_letters_len*, *url_digits_len* and *js_len*), two binary features (*https* and *label*), and features that have categorical values or require specific treatment (*ip_addr*, *url*, *continent*, *tld* and *content*).

Firstly, the continent in which the web page is hosted is preprocessed (*continent*). The parameter was converted to a one-hot encoding. A one-hot encoding consists in passing the multiple choice feature into a table where each column represents a different categorical value, hence, we created a new feature per existing continent. The column which continent corresponds the sample will be marked as 1, otherwise the value is marked as 0. Thus, after the *continent* preprocessing,it had six new binary features, one for each continent.

Next, the Top Level Domain (TLD) preprocessing was performed. Since more than 600 different values were computed, the preprocessing concentrated on the *.com* domain. The *.com* domain constitutes a total of 60% of the final data. Consequently, it acts as a suitable separator. Therefore, the feature was stored whether the TLD is *.com* or not. Conversing thus, the feature to a binary option.

Both URL, IP and *content* features were deleted since they are thoroughly represented in other features and do not give additional information. Furthermore, the IP was initially converted into a binary sequence, and the model was trained with the parameter. However, the results showed less performance than without this feature. Additionally, as each web page must be parsed differently, and no clear and helpful patterns were found, the *content* feature was deleted during the preprocessing. Moreover, all numerical values were normalized, and the binary parameters passed through a binary one-hot encoding. The label was transformed using a binary one-hot encoding with a 1 value if it is considered malicious (bad) and 0 if the website is considered benign (good). This process is depicted in Figure 5.

All this preprocessing is applied to the training, testing and validation datasets. The dataset has been divided into three sections:

- Training dataset with 1200000 samples
- Testing dataset with 350000 samples
- Validation dataset with 10000 samples

Having three distinct portions of datasets allowed us to train the model with a vast amount of data. Then by test the model we assured the performance was the desired and that there was no over or underfitting. Once the model was trained and ready, the model was exported and saved. Next, the project validated the model by predicting the label of 10000

samples (containing 195 malicious samples). We then compared the obtained labels with the expected ones and analyzed the outputs. Figure 6 shows a percentage representation of the three datasets.

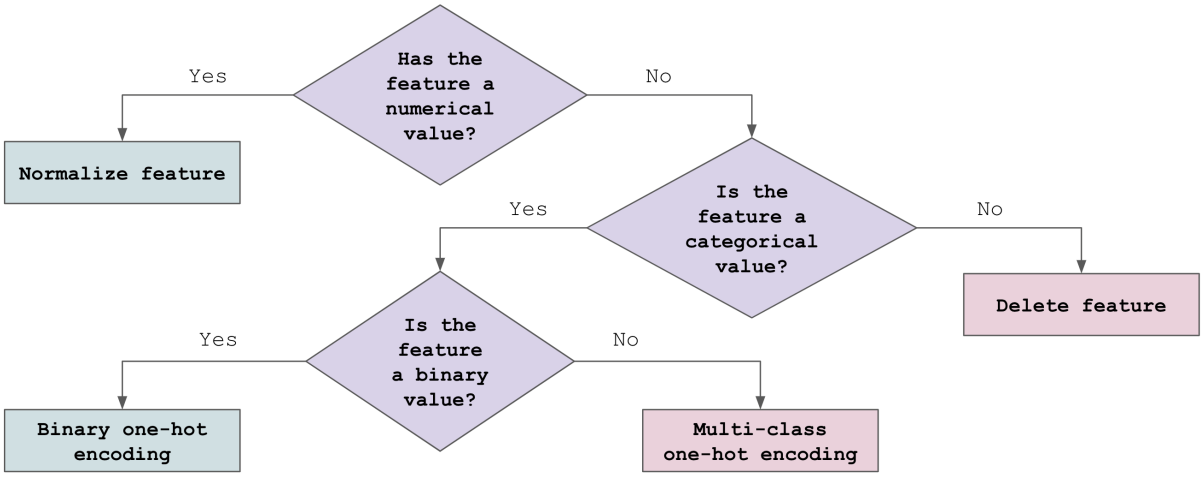


Figure 5: Preprocessing flowchart

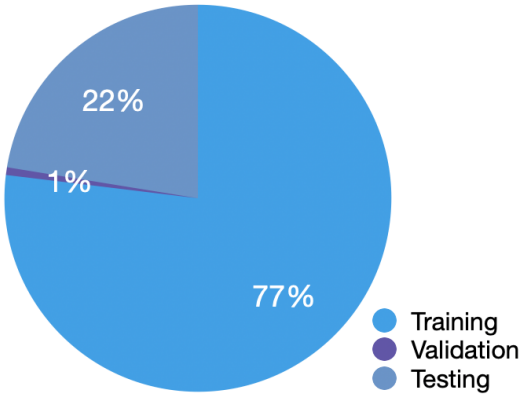


Figure 6: Percentage representation of the three dataset partitions

7 Deep Learning application

Given the preprocessed dataset generated, we propose the implementation of a Fully Connected Neural Network, specifically a Feed-Forward Neural Network (FFNN) depicted in Figure 7.

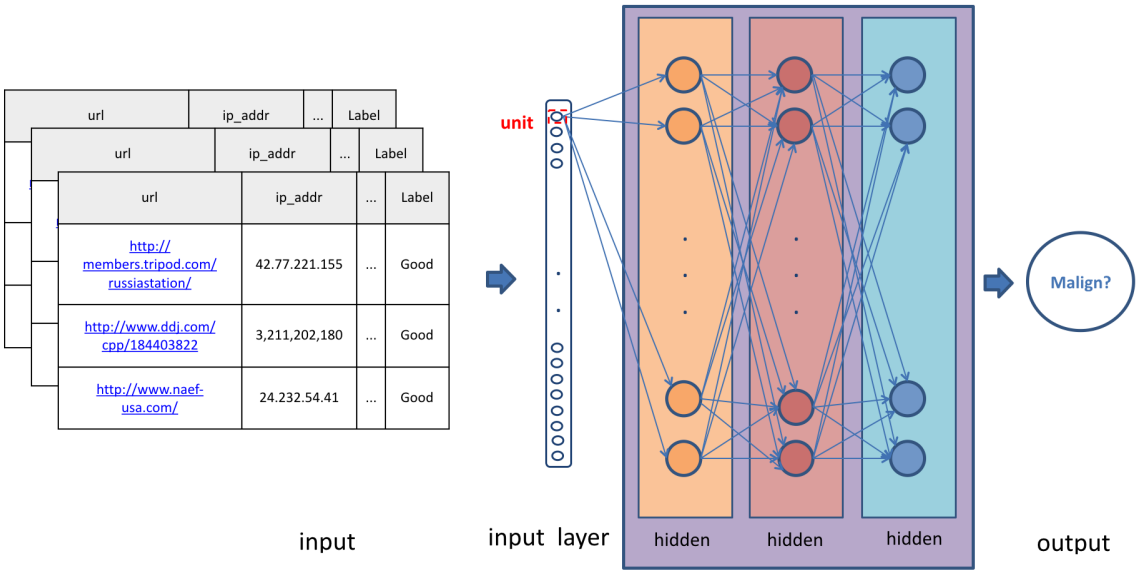


Figure 7: Feed-Forward Neural Network representation

The reason behind choosing an FFNN is due to the significant connection between parameters. Having fully connected layers enables the network to perform complex relationships between parameters, thus improving the detection capability of the system. Therefore, we apply this NN to demonstrate the effectiveness of the proposal.

A FFNN is described using several design parameters that conform a model where the training data is introduced. Additionally, the model's algorithm is run for a number of iterations or epochs. To avoid under or overfitting and to achieve good performance, the model parameters must be carefully chosen. Therefore, the proposed architecture has an input layer, three hidden layers and an output layer that uses the *Sigmoid* activation function. Each layer uses an activation function and has several neurons. In our model, the hidden layers use a *Rectified Linear Unit* (ReLU) activation function since they avoid saturation and do not stop to shape the sample weights. The input layer has an input size of 17 features. Furthermore, the first hidden layer has 64 neurons; the second layer contracts the values to 32 neurons. Finally, the third hidden layer has 64 neurons.

The final parameters that define our Neural Network are the loss, optimizer and epochs (or iterations). A *Binary cross-entropy loss* is used on the resulting vectors since it calculates the prediction error in a binary measure, just as we need for our output. The optimizer aims to sculpt the model into a precise form and to minimize the loss. We use *Adam* [18], which achieves good performance in few epochs. Finally, the number of epochs is set to 10. Ten epochs is a good compromise between stability and over-training.

8 Experiments

To evaluate the model and the proposed architecture, we conducted several experiments. First, we tested the correct implementation of the preprocessing by analyzing the correlation between attributes, computing the correlation matrix. Additionally, we conducted histograms for each feature and the label value to extract the dispersion of the parameters. In the sample's preprocessing, it is crucial that parameters can be easily distinguishable by the network. Therefore, performing a histogram between each value and the label is vital to understand its separability. Moreover, attributes are interconnected in the dataset. This interconnection can be seen in the feature correlation matrices. By performing these experiments, we can decide the optimal NN use, its parameters and identify non-useful attributes.

Second, to implement the NN depicted in the previous section, we started by implementing different combinations of FFNNs. The number of hidden layers was decided according to the dataset characteristics. In total, the dataset had an input of 17 attributes, meaning that the number of training samples considerably exceeds the number of attributes. With that in mind and the attributes dependency, we opted for medium-sized FFNN. Having less hidden layers allows the model to have a minor abstraction of the features, and having a more significant number of hidden layers allows the model to be over-complex. Additionally, to decide the number of neurons of each hidden layer, we analyzed all the possible combinations from 16 to 512 neurons (in powers of two), meaning that in total, we ran 56 different FFNN ($CR_3^6 = \frac{(6+3-1)!}{3!(8-3)!} = 56$). Evaluating the network neurons in powers of two allows us to cover a larger range of values.

Furthermore, we analyzed the most frequently used metrics: accuracy, loss, Area Under the Curve (AUC), precision, recall, and f-score for these networks. These metrics allow us to have an extensive analysis of the network's performance.

Firstly, the accuracy is defined as the True Positives (TP) plus the True Negatives (TN) divided by the sum of TP, TN, False Positives (FP) and False Negatives (FN). The formula is represented in Equation 1.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Secondly, the defined loss is the binary cross-entropy loss which formula is represented in Equation 2. Note that the y value represents the real output, whereas the $\hat{y}$ represents the output estimation.

$$\text{Loss} = -[y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})] \tag{2}$$

Precision is defined as the TP, divided by the sum of the TP and the FP. The formula is represented in Equation 3.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

Then, the recall is defined as the TP divided by the sum of the TP and the FN. The formula is defined in Equation 4.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4}$$

Next, the f-score is described as a mixture of the precision and recall formulas to evaluate the combined performance. F-score is defined as in Equation 5.

$$\text{F-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

Finally, AUC shows the performance of a classification model. It is determined as the area that defines the ROC curve, which is the graphical representation of the True Positive Rate (TPR) vs the False Positive Rate (FPR). The TPR is defined as in Equation 6, and the FPR is defined as in Equation 7.

$$TPR = \frac{TP}{TP + FN} \tag{6}$$

$$FPR = \frac{FP}{FP + TN} \tag{7}$$

We also analyzed the metrics from the validation data to extract further conclusions.

In the experiments, we performed 20 iterations of the three best performing FFNNs to ensure their stability. 20 iterations were chosen since it was crucial to assure the network's consistency. However, we did not want to over-charge the cloud server with redundant calculations.

Moreover, another experiment to test the system was the time limitations and viability study. The experiment consists of studying the detection intervals that each URL takes to be analyzed and research the project's viability based on its limitations.

To be able to execute the project, the system requires some programs installed. The end-user will need a computer equipped with internet, a bash terminal, and Python3 installed to execute the script. Python was the language used to execute the system since it is the most used programming language to develop DL programs. It has a vast platform and has extensive documentation and community. Moreover, the system has been created using a MacBook Air computer with a Dual-Core Intel Core i5 and 8 GB of RAM. The university's cluster, Sert, was used (AMD EPYC 7101p at 2.80 GHz and 128 GB of RAM) to filter the data. Besides, all the project is coded using Python and executed using Shell scripts. Additionally, to create the blocking system Deep Learning has been used, using Keras from TensorFlow. Finally, the project was preprocessed using the Python's libraries, Pandas and analyzed with Matplotlib and Sklearn to compute correlation matrixes and graphics.

9 Results

The depicted sections above explained the final procedures that were performed to achieve the definitive result. To decide the best procedures, the experiments explained in section 8 were conducted. This section shows the different results obtained from the experiments executed and a performance discussion. The results section is divided into two different parts: the processing results and plots and the FFNNs results and metrics.

9.1 Preprocessing results

To test the data preprocessing and its relevance in the dataset, we decided to conduct several experiments. Firstly we parted from the *Visualization of Malicious & Benign Webpages Dataset (VoMBWeb)* [17] since it included some of the attributes that we wanted to treat and was generated in a non-simulated environment. Considering that Keras from TensorFlow needs as an input a tensor of NumPy arrays, we decided to convert the multi-class attributes into a one-hot encoding, the IP was binarized, the webpage content was deleted, and all arguments were normalized. Once this preprocessing was performed, we observed some concerns with the dataset structure.

This dataset's main issue was that once performed the correlation matrix (depicted in Figure 20 at Annex A.3), the resulting values were very dispersed between the different attributes. There, it could be observed that the IP did not give any additional information since its correlation was nearly zero. Furthermore, treating each country independently added more than 300 features and gave misguided directions to the DL model. However, the main issue was that the dataset relied exclusively on the JS obfuscated length, meaning that the FFNN only modelled its weights regarding that feature. Hence, the dataset was severely modified to remove this dependency and add some relevant features that enable the FFNN to train using all the available features.

From the original dataset, we decided to remove the JS obfuscated length, group the country feature into continents (code depicted at Annex A.3) and treat the TLD distinguishing only by *.com* or otherwise. With this modification, the homogeneity in the features' correlation increased.

Since the URL is the only input received from the user, we decided to extract as many attributes as possible. The main goal was to find the balance between extracting trainable features and having unnecessary or misguiding information inserted to the NN. The final decision was to include at least one attribute from each URL subdivision (the URL, the protocol, the body, the TLD, the arguments and the parameters) as seen in Figure 8.

Figure 8: URL subdivisions

Additionally, we decided to add three more attributes that gave connections between the URL attributes. Those features were the URL entropy (computed in Annex A.3), the URL letters length and the URL letters and digits length.

Once all the preprocessing was done and transformed into numerical values, we performed a final analysis of the dataset (code depicted at Annex A.3). First, each parameter is compared with the label characteristic. The resulting histograms are shown in Figure 21 at Annex A.3. They provide the interconnection information and how to differentiate each value.

In this figure, we can see malicious and benign values have a very distinguishable separation in some cases. These results indicate that the Neural Network should discern between malign and benign websites easily during the classification performed in the FFNN.

We conducted another analysis to study the dependence and relevance of the parameters that do not show a very distinguishable separation. A new correlation matrix was performed with all the final preprocessed attributes. The correlation matrix can be seen in Figure 9.

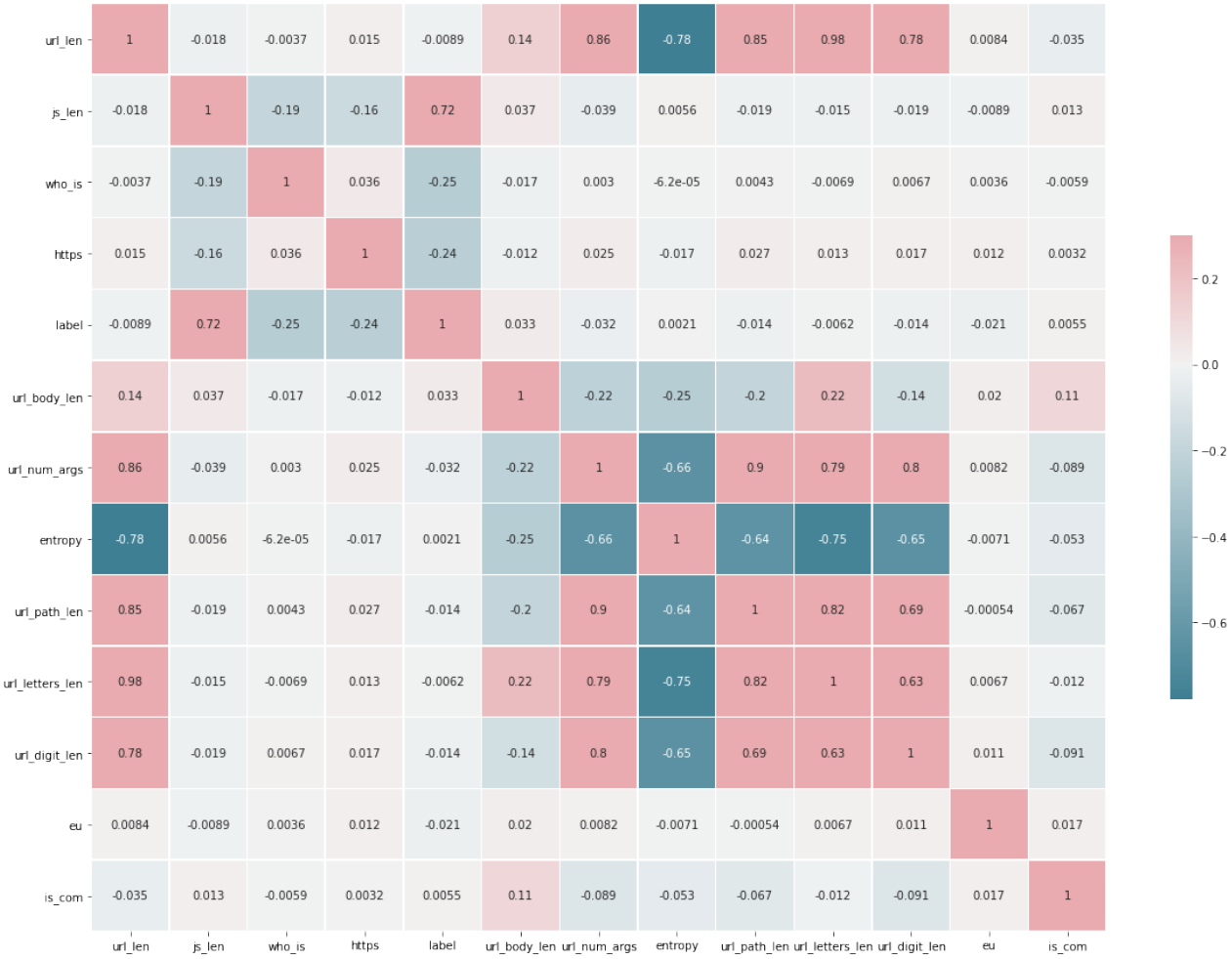


Figure 9: Preprocessed dataset parameter correlation matrix

If we look at the label column, we observe the most and least interconnected parameters. The most correlated parameters are *js_len*, *https* and *who_is*, which contain a correlation between 0.24 and 0.72, meaning that these parameters are crucial for detecting malicious websites. The parameters, such as *who_is* and *https*, which have negative correlation, indicate an inverse proportion between the label and the selected parameter. All the remaining attributes have at least some correlation with the matrix. They are highly correlated with other parameters, which improves the FFNN since it is a type of NN that relies mainly on feature interconnection and dependence.

Having performed this study, we believe that it is wise to use all the depicted parameters for evaluation in the NN. Moreover, since the parameters are correlated between them, and each column depends on other attributes, FFNNs are the best fit.

9.2 FFNN results

After preprocessing, we run the different combinations of the FFNNs. As commented in the previous section, we conducted two sets of experiments in the Neural Networks. The first set analyzed the performance of variations of several neurons in each layer and the network's depth. The second set consists of the repeated execution of the three best performing FFNN and the performance evaluation.

To finalize the network topology, we had to decide the number of layers the FFNN would have as the number of neurons in them. As stated in the previous section (section 8), three hidden layers is a valid compromise between good performance and network complexity. To prove it, we conducted three different FFNNs with a low count of neurons. The reason behind having fewer neurons in each layer is that a higher neuron count adds extra complexity to the FFNN. The number of layers is directly related to the number of attributes we have (17 in total). Hence, we tried networks with 2, 3 and 4 layers. The FFNN with two layers indicated low network complexity and a lack of resources in order to be trained. The other two networks showed similar accuracy. Since having four layers adds complexity and computation consumption to the network, it was decided to perform the FFNN with three hidden layers. The results are shown in Table 3.

Table 3: Performance of three different FFNNs with different number of layers

Neurons	Loss	Accuracy	AUC	Precision	Recall	F-score
16 - 8	0.31%	98.98%	99.95%	99.63%	94.95%	97.22%
16 - 8 - 16	0.30%	99.88%	99.93%	100.00%	94.75%	97.30%
16 - 8 - 16 - 8	0.31%	99.88%	99.95%	99.92%	94.89%	97.34%

Once the number of layers was determined, we conducted 56 different combinations of FFNNs to decide the best fit. When performed the FFNNs variations, we observed certain similarities between results. First, all the networks achieved high performance in all metrics oscillating between 94.81% and 99.88%. These results indicate that the election of an FFNN with the selected loss, optimizer, activation function and epochs is ideal. The best performing networks have a minimum of 32 neurons in each layer and 64 or more neurons in -at least- one of the layers. Table 4 shows the results for the three best-performing networks. The results of these networks are very similar. Note that the Neurons column represents the number of neurons inside each hidden layer, represented as *neurons_hidden_1* - *neurons_hidden_2* - *neurons_hidden_3*.

Table 4: Testing performance for the three best FFNNs with three layers and variation of the amount neurons per layer

Neurons	Loss	Accuracy	AUC	Precision	Recall	F-score
64 - 32 - 64	0.30%	99.88%	99.95%	97.49%	97.38%	97.42%
32 - 64 - 32	0.47%	99.88%	99.79%	99.18%	95.33%	97.22%
128 - 64 - 32	0.46%	99.87%	99.75%	99.31%	94.81%	97.01%

The best performing network (and the one used in the system) is the first one (64-32-64) since the loss decreases and f-score increases compared with other two, and it is the most stable network with lesser differences between attributes than the others. Furthermore, the network is relatively small, meaning that it is a computationally efficient network.

Figure 10 shows the mean of all the different metrics used for the best network when executed a total of 20 times. The network achieves 99.88% of detection with the validation data and only ten epochs, giving a high detection rate using little resources. As for the AUC, the network achieves 99.95% in validation data. Indicating that the performance of the classification model is almost 100%, thus demonstrating its effectiveness. The loss value is less than 0.5% in all executions, meaning that the model does not have over or underfitting. Besides, the f-score achieves 97.42% with the validation data. The f-score helps us to understand the model's combined performance. A high f-score reiterates the network effectiveness showing high precision, recall, accuracy and performance in all studied metrics. Consequently, the proposed system is very effective in detecting malicious websites.

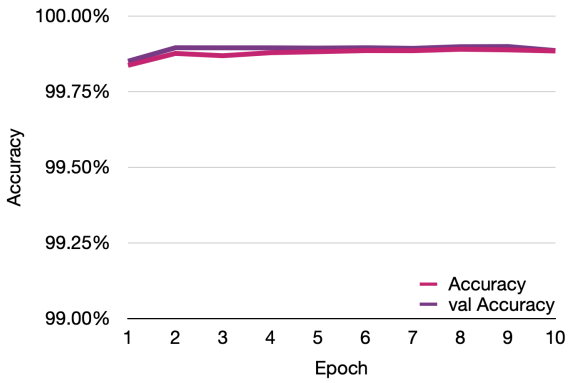
Finally, we compute the TP (True Positives), TN (True Negatives), FP (False Positives) and FN (False Negatives) of the selected network. The TP represents the samples that belong to a malicious website and is correctly classified. FN represents the samples that belong to malicious websites that are wrongly classified. The sum of TP and FN denote all the existing malicious websites. On the other hand, TN represents the benign samples that are correctly classified, and FP the benign samples that are wrongly classified. Together TN and FP denote all existing benign websites. The final goal of the system is to detect all malicious websites without misclassifying any sample, which means that the network has to minimize the FN rate. Table 5 shows the training and validation results of these metrics after the tenth epoch. The results show that the FN rate in the validation data is only 2.619%. In other words, only 205 of the 8062 malicious websites are erroneously classified as benign. We can also observe that the FN rate decreases in the validation phase. This decrease indicates that there is still room for improvement in the network (e.g. adding more data and re-training the network).

Table 5: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) for the best performing network (64-32-64) with 10 epochs

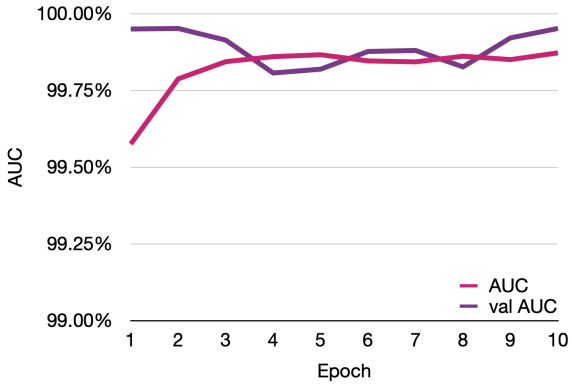
Data	**TP**	**TN**	**FP**	**FN**
Training (%)	25910 (95.070%)	1172698 (99.999%)	48 (0.001%)	1343 (4.930%)
Validation (%)	7623 (97.381%)	341977 (99.994%)	196 (0.006%)	205 (2.619%)

As commented in the preprocessing section, the dataset was divided into three different parts. The final testing of the system consists of passing through the trained network the validation datasets (containing 10000 samples). The final analysis results show a total number of wrongly classified malicious websites (a total of 1.026 %) and an accuracy

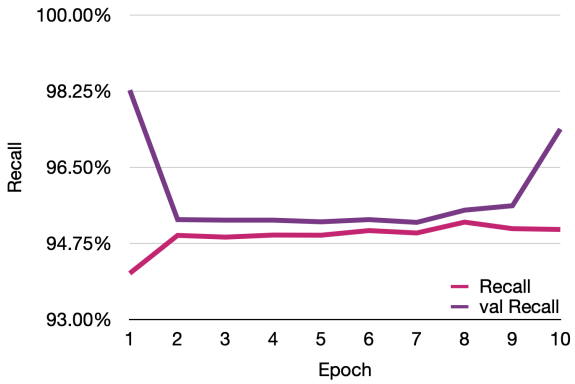
of 99.75 %. These results show a clear identification and classification of malicious websites. Overall, these outcomes demonstrate the effectiveness of our system since almost all malicious websites will be filtered, and they will not penetrate the system.



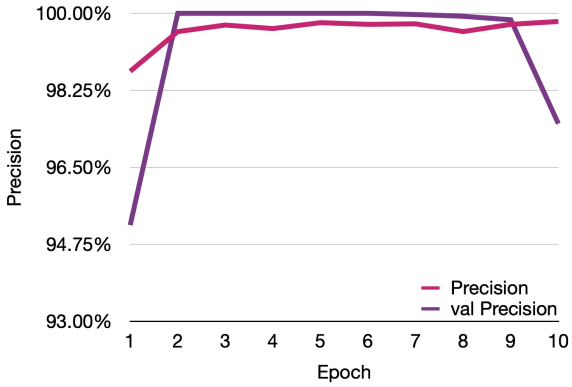
(a) Accuracy results



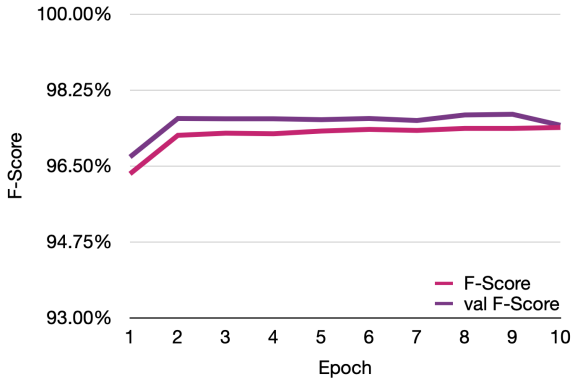
(b) AUC results



(c) Recall results



(d) Precision results



(e) F-Score results

Figure 10: Mean performance for the best operating FFNN (64-32-64) in 20 iterations

10 Interface

For the final classification phase, a user-friendly interface has been computed. The interface aims to test the project's performance and is intended to be easy to operate and engaging, assuring its usability. Hence, it is designed to be adapted into a chrome extension that ensures a safe navigation through the web.

The interface was created with a Python Library (TKinter) and was developed to be customer friendly. Also, it has been developed using the Model View Controller (MVC) pattern. The interface employs this pattern to construct two different views: the initial view and the outcome view.

In the initial view, represented in Figure 11, a welcome message and a URL input box are shown to the user. Moreover, once the user inserts the URL, the system analyzes the URL, and the interface adds a waiting indicator. This indicator aims to give the user an advancing sensation.

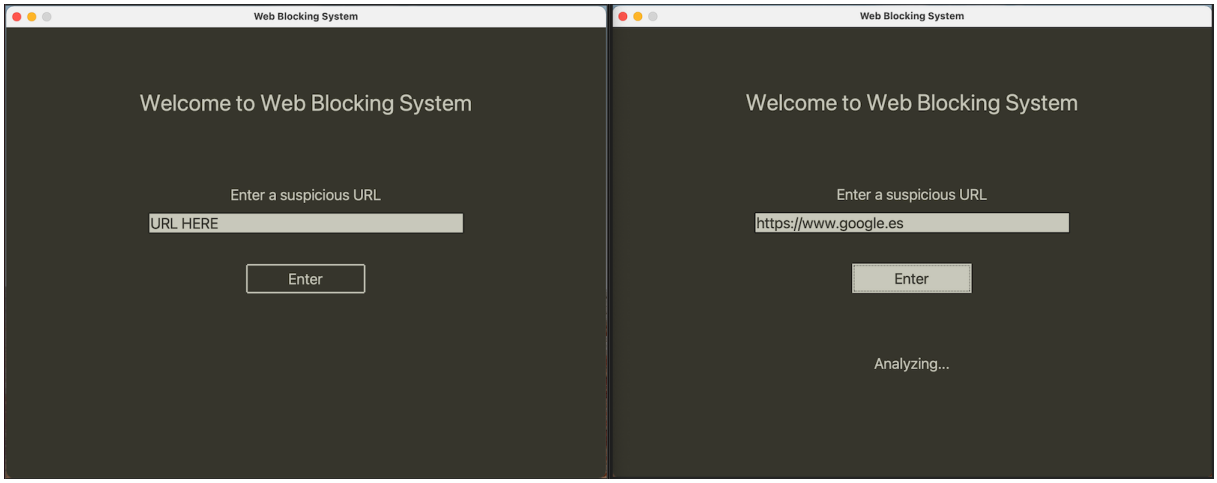


Figure 11: Initial interface view

Finally, the output is displayed with an explanatory text at the end. The system can show four different outcomes: good, bad, suspicious and unreachable URL. A good URL is one that the prediction returned a value between 0 and 0.2. A bad URL is one that the prediction returned a value between 0.8 and 1. A suspicious URL is one that the prediction returned a value between 0.2 and 0.8. Finally, an unreachable output means that the URL was wrongly written or the URL was not found browsing the internet. This system is represented in Figure 12.

To create this software, some scripts have been made. Firstly, the trained model from the first block of the project was saved. Secondly, a file containing the URL preprocessing (computed as in the validation phase) has been used. Thus, the URL from the interface is preprocessed, extracting the necessary data to compute all the needed features, and passed through a prediction method. The prediction method passes the preprocessed sample through the model and returns a prediction of the output. Moreover, since the project aims to be as quick, accurate and efficient as possible, a database containing all the

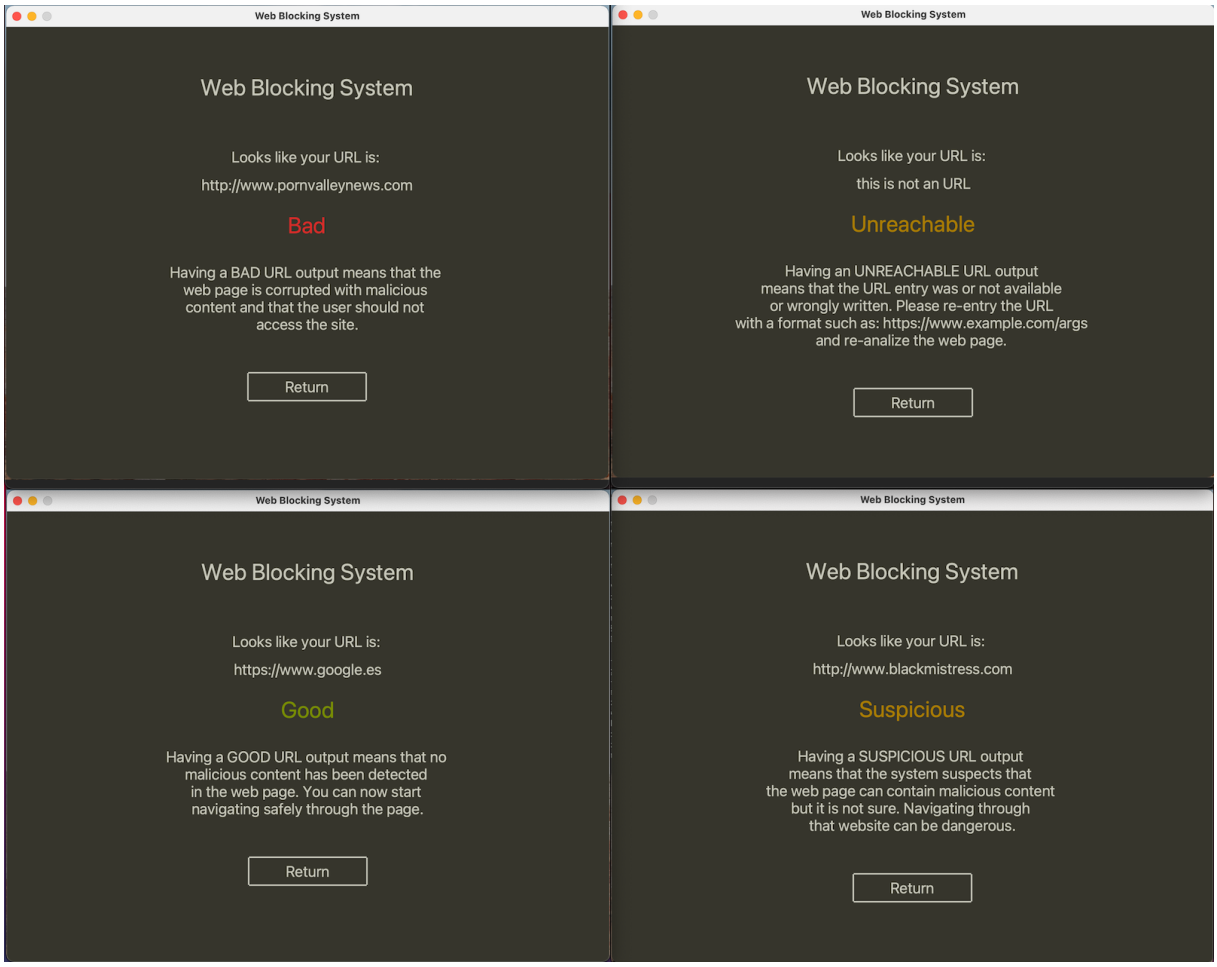


Figure 12: Output interface view

1550000 samples has been introduced. Introducing the database reduces the computation time to a fifth portion of the initial time (as explained in the following section).

Overall, the system shows an initial view that enables the users to input their URL. This URL is passed through preprocessing and the saved model to compute a prediction. The output can vary depending on the prediction performed and the entered URL, and it can have four values: good, bad, suspicious and unreachable.

11 Time limitations and viability

Once the system is computed and working, some time calculations are performed to assure the user viability of the program. The system must be quick and efficient, being fast enough to avoid exhausting the user's patience. Hence, once the system is initialized (14 seconds), each URL is treated and classified. This classification takes some time to be performed, and it is the case of our study. Thus, some URLs computing time was analyzed to research if users' waiting time will be acceptable. Two different types of URLs were executed. Firstly, 25 URLs inside the central database, and then another 25 URLs external to the database. The database acts as a URL cache and returns an already preprocessed sample, therefore, reducing the waiting time.

Figure 13 plots the different times that each URL takes. Several observations can be extracted from the graphics. Firstly, it is observed that the mean execution times are 0.37 seconds for the URLs already stored in the database and 2.53 seconds for the URLs that were not stored in the database. According to Fiona Fui-Hoon Nah [19], users expect a response in about 2 seconds for simple information retrieval tasks on the web and marks 4 seconds the maximum waiting time a user takes before feeling discomfort. The conducted study was performed with different waiting times and concluded that a 2-seconds response is needed to ensure continuous interactions.

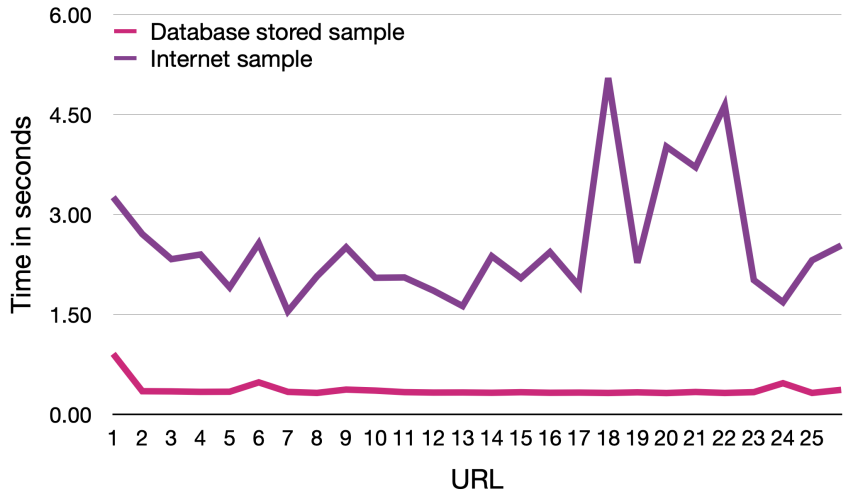


Figure 13: Time performances of URLs extracted from the internet or the database

Furthermore, Jakob Nielsen [20] performed a study of the user's behaviour during the last thirty years and gave three basic rules to assure a user keeps its attention on the desired website. The article gives three precepts: firstly, that 0.1 seconds marks the limit to making a user feel like a system is working instantaneously. Secondly, that 1 second is about the limit to assure the user's navigation flow remains uninterrupted. Thirdly, 10 seconds marks the limit for keeping the user's attention. In this project, since the largest mean execution time is 2.53 seconds, the system surpasses neither the 10-second limit nor the 4-second limit. Hence, the system is suitable to keep the user's attention and perform smoothly.

It can be observed that in the case that the URL's content is already stored in a database decreases in more than 2 seconds the computation time. Hence, storing the most frequent and used URLs in the database is recommended.

Additionally, in Figure 13, some peaks are shown. These peaks represent delays in some websites since the computation time of each URL varies. Nevertheless, 94% of executions finish before the 4-second limit, and 100% of executions finish before the 10-second limit, making the system moderately constant in their time measures and satisfactory to the final user.

12 Budget

The proposed project is mainly a software research project. There is only needed a single junior engineer with previous Python, TensorFlow, and Artificial Intelligence skills to elaborate the system. Furthermore, the engineer must be provided with a computer, internet connection, a cloud environment to compute the experiments, and free software and tools.

Firstly, the junior engineer cost yearly is 26200€ for full working days. That is to say, a mean salary of 12.6€/h. Thus, the total engineer expense for four months (480h) is 6048€and adding the company taxes (34%), the final expense sums up to 8104€. Since the employee works remotely, there are no indirect expenses related to the housing or supplies (water and light) except the internet connection.

Secondly, an internet connection during 4 months is needed. The monthly cost of a 600Mbps internet connection is 30€/month. Thus, having a total cost of 120€.

Thirdly, the needed computer has a total cost of 950€, with a lifetime period of 4 years and a residual value of 250€. The depreciation value is calculated as $Depreciation = \frac{asset\_cost - residual\_value}{useful\_life}$. Furthermore, with a depreciation value of 175€/year, the total computer cost for 4 months is 58€.

Additionally, a Cloud environment must be hired since the computation needs to perform the dataset and train the network cannot be done with the provided computer. Furthermore, having an external system provides efficiency and assures the programmer fewer blackouts.

The need of the Cloud environment is distributed in the dataset generation, the preprocessing generation and the network training. For the dataset generation, a total of 4 days (96h) of computation time with 8 different processes and 32Gb of RAM were needed. Additionally, we performed 56 different network combinations for the network testing and generation, and the three best performing ones were repeated 20 times. Each network has a mean computation time of 2h, meaning that the total hour estimation of Cloud computing is $56 \cdot 2 + 20 \cdot 3 \cdot 2 = 232h$ with the same computing specifications as for the dataset.

Hence, considering the dataset computation and the network training, a total of 328h of Cloud computing are needed. The mean estimation cost for one Debian node with 8 vCPUs, 32Gb of RAM and 50Gb of storage SSD of Amazon Web Services and Google Cloud Computing is 161€/month (720h). Leaving a 20% margin for unforeseen events, a total of 394h of computation are needed. That is to say an estimation of approximately 88€ of Cloud computing costs.

Finally, the software and tools must be computed. Since the project has been developed with free tools and software supplies, there is no added cost. Overall, all the costs are summarized in Table 6.

Table 6: Total budget estimation of the project's direct and indirect costs

Concept	Time	Price	Amount
Engineer salary	480h	12.6€/h	8104€
Internet connection	4 months	30€/month	120€
Computer	4 months	175€/year	58€
Cloud environment	394h	161€/720h	88€
Software and tools	4 months	0€/month	0€
Total			**8370€**

13 Sustainability

The project aims to minimize the negative impact that the system causes and be the most sustainable as possible. To achieve maximum sustainability, a study of the project's social impact, economic impact, and environmental impact has been made.

13.1 Social impact

The social impact represents the influence that the system has on society. In our project, the social impact is the most crucial and influential of the sustainability analysis. The project aims to develop software capable of detecting malicious activity, enabling the users' comfort and trust while navigating through their browser.

Nowadays, the internet is plagued with cybercriminals that use malicious websites to scatter attacks across the web. Their aim is to take advantage of the browsers' different vulnerabilities and install malicious content to control and access the user's computer. Hence, the final user's data protection is essential since giving users private browsing maintaining their information secure produces a beneficial impact in society. Besides, fewer website attackers will profit from people allowing less cybercrime on the internet.

Furthermore, by using non-simulated data, the system can be updated and re-nourished, enabling it to be improved. The used techniques allow research to evolve in the development of Artificial Intelligence, such as taking this project as a reference for future studies.

13.2 Economic impact

The primary purpose of malicious websites is to promote mainly frauds and scams. By entering a malicious website, the attacker can insert numerous types of malicious content, such as ransomware, trojans or viruses, that can compromise the user's computer as its information. Furthermore, if the malware is installed in a company's computer with a shared network, this network could be hacked. Thus, a single URL can cause computer catastrophes. The possible damage is so severe that one of the biggest threats of the digital world is considered malicious sites.

The system's economic impact can result in being of great importance. Firstly, since the software restricts a user from entering a malicious website, it prevents viruses from entering different devices. The lack of malicious content inserted into computers saves considerable money to users who will not have to buy another computer ahead of time.

Additionally, some of the hacked users are requested money in exchange for their stolen information; having a system that prevents this hacking from happening allows users to save considerable money. Nevertheless, the system uses Cloud servers to compute some of their calculations and a computer that the engineer must use. Nonetheless, the economic loss that the company has by using this hardware is much less than the one saved by all the benefactors of the system (the end-users).

13.3 Environmental impact

If a system gets infected by website malware, it could lead to a computer malfunctioning or scamming. These are two factors that could drive a negative environmental impact.

Nonetheless, the system reduces the malfunctioning computers due to malware prevention and saves them from going to waste before they have reached total capacity. Additionally, the system prevents attackers from downloading the user's private information preventing them from swindling or blackmailing. Also, since the study is only composed of software, no environmental damage must be introduced to our ecosystem.

In conclusion, the study is highly sustainable since it allows advancement in society, withdrawing a constant threat such as malicious website activity while allowing research to progress. Furthermore, the hardware saving by the end-user highly exceeds the hardware used by the engineer to develop the project. Finally, the project is constituted entirely by software, meaning no damage to the ecosystem must be introduced.

14 Conclusions and Future work

Malicious URL detection plays a critical role for many cybersecurity applications, and clearly, Machine Learning approaches are a promising direction. The importance of this detection remains in assuring the user a safe browsing, blocking the non-desired content. In this project, we conducted a study on Malicious URL Detection using DL techniques. In particular, we offered a comparison between existing technologies (research or commercial wise) and our approach. Additionally, the proposed system evaluates the website's features and classifies them by preprocessing and entering them in a previously trained DL model.

We reformated the *Visualisation of Malicious & Benign Web-pages Dataset (VoMB-Web)* [17] and extracted basic features of existing websites (malicious and benign) to conduct the study. We measured the dispersion and correlation between the features to observe the label separability and the feature interconnections. The preprocessing results showed the unnecessary features, which were deleted to lower the burden of the NN.

Moreover, we proposed an FFNN to compute the classification of the labels. After studying different variations of an FFNN and conducting several experiments, we conclude that our network can discern malicious from benign websites efficiently and effectively. The proposed mechanism achieves a 99.88% accuracy, 99.95% AUC, 97.49% precision, 97.38% recall and 97.42% f-score. Furthermore, the proposed NN only incorrectly classifies 2.619% of the malicious websites. This slight inaccuracy is due to the complexity of identifying all possible patterns out of malicious content.

Additionally, a validation process was performed, which consisted of entering into the model 10000 unseen samples and evaluating its results. The validation results showed only a 1.026% of inaccuracy in wrongly classified malicious websites, thus decreasing the error. Then, the trained NN and the sample designed preprocessing were entered into a user-friendly interface. The user can enter a URL in the interface, and the system will return its prediction output (malicious or benign).

Finally, time consumption and user engagement were studied to assure the product's engagement. Results showed a maximum mean waiting time of 2.53 seconds, which, according to the previously mentioned studies, meets the 4-second limit, reaffirming the system's user engagement.

Therefore, the project's main contributions are an interface capable of detecting if a provided URL is malicious or benign. The interface has a model which is trained with DL techniques and a dataset generated from non-simulated samples. Furthermore, the FFNN and data preprocessing were optimized to achieve maximum accuracy. Besides, the system's results are shown to be beneficial and viable to apply.

As future works, the system can be improved by adding more data and re-training the network. Moreover, the NN could be adapted to be automatically updated. An automatization would mean that the NN would improve with every NN search and add new features into the model. Additionally, the execution times could be reduced to give the users the illusion that the system works instantaneously. Another improvement could be that the system detects the type of attack that the malicious URL has. Finally, the interface could be converted into a Google Chrome extension for easy access and application.

A Annex

A.1 Extended work plans

In this section the different updates in the **work packages** of the project are described. You can find them inserted below.

Project: Malicious websites blocking system using Deep Learning algorithms	WP ref: 1	
Major constituent: Research	Sheet 1 of 6	
Short description: The research work package consists of a thorough investigation of related projects. This study analyzes datasets, preprocessing techniques, artificial intelligence methods and possible outcomes. The section continues with a planning of the project development and finishes with a literature review with comparison tables.	Planned start date: 15/02/2021 Planned end date: 03/03/2021	
Internal tasks: - Related work investigation. - Related commercial products - Thesis planning - Related works writing and summarizing in a table. - Write report.	Deliverables: Final draft of the investigation.	Dates: 04/03/2021

Figure 14: Research work package

Project: Malicious websites blocking system using Deep Learning algorithms	WP ref: 2	
Major constituent: Data treatment	Sheet 2 of 6	
Short description: In this work package, the aim is to chose a suitable dataset containing non simulated samples. To treat the data some articles must be researched and by fine-tuning the preprocessing the data will be computed. One the dataset is correctly modified new features must be extracted and analyzed (by doing the correlation and histograms). Finally the unnecessary features will be deleted allowing the dataset to be optimized.	Planned start date: 04/03/2021 Planned end date: 30/03/2021	
Internal tasks: - Investigate different type of databases containing the needed data. - Preprocess the selected non-simulated data - Optimize the data preprocessing - Write report.	Deliverables: Demonstration of the functionality and WP report.	Dates: 31/03/2021

Figure 15: Data treatment work package

Project: Malicious websites blocking system using Deep Learning algorithms	WP ref: 3	
Major constituent: Deep Learning Neural Network implementation	Sheet 3 of 6	
Short description: This work package starts with the optimized preprocessed data and will perform several variations of Neural Networks. The best performing model will be thoroughly studied and analyzed. The predictions from the best performing Neural Network will later be filtered and be classified into a danger region. The final task of the first part of the project will consist of training, testing and saving the created model.	Planned start date: 21/03/2021 Planned end date: 17/04/2021	
Internal tasks: - Create an optimized Neural Network that classifies malicious and benign websites. - Select the best performing network - Study the outputs of the different metrics and extract conclusions. - Save the trained model - Write conference report.	Deliverables: Working NN that predicts malicious websites and a conference report.	Dates: 18/04/2021

Figure 16: Deep Learning Neural Network implementation work package

Project: Malicious websites blocking system using Deep Learning algorithms	WP ref: 4	
Major constituent: System validation	Sheet 4 of 6	
Short description: This is the most critical work package since its only goal is to achieve a good functioning of the system. The aim of the work package is to validate all the previously done work by testing the system with 10000 unseen data samples. Once this validation is performed new actions and modifications must be performed to achieve a good system functioning.	Planned start date: 18/04/2021 Planned end date: 6/05/2021	
Internal tasks: - Extract the trained model. - Given the results from the NN predict new information; thus, validating the system. - Make necessary modifications to the algorithm. - Adapt the network to achieve full capacity and test it. - Write report.	Deliverables: Physical demonstration of the results plus the WP report.	Dates: 7/05/2021

Figure 17: System validation work package

Project: Malicious websites blocking system using Deep Learning algorithms	WP ref: 5	
Major constituent: Interface creation	Sheet 5 of 6	
Short description: In the final work package, based on the results obtained in the previous deliverable, the aim is to create an interface that given a URL is able to extract the necessary information and predict whether the URL comes from a malicious website or not. To do so, all the results will be put into an interface where the blocked URL will be displayed.	Planned start date: 7/05/2021 Planned end date: 20/05/2021	
Internal tasks: - Create an interface to interpret the results. - Upload the NN model into an interface where a sample can be inserted. - Display everything in the created interface. - Interface prettifying - Write report.	Deliverables: Final report with testing of the whole project	Dates: 21/05/2021

Figure 18: Interface creation work package

Project: Malicious websites blocking system using Deep Learning algorithms	WP ref: 6	
Major constituent: Additional studies	Sheet 6 of 6	
Short description: In this work package, a study of possible time limitations as for the possibilities for commercial solutions and its sustainability will be computed. These final analyses will be used to evaluate the system's efficiency and its future improvements.	Planned start date: 21/05/2021 Planned end date: 6/06/2021	
Internal tasks: - Study the time limitations. - Study the system's final cost and its budget to re-produce - Study the system's sustainability - Write report.	Deliverables: Physical demonstration of the results plus the WP report.	Dates: 7/06/2021

Figure 19: Analysis work package

A.2 Related works comparison table

Table 7: Performance of the related works and comparison with this project

Work reference	Publication year	Dataset	ML or/and DL	Used algorithm	Maximum Accuracy
[5]	2018	Own compilation	ML and DL	FFNN and SVM	97.2%
[3]	2012	From blacklists and campus information	ML	SVM and RBM	85.7%
[6]	2019	ISCX-URL-2016 data	DL	LSTM and CNN	98.86% (with CNN)
[4]	2020	Own compilation	ML	SVM and RF	99.7% (with RF)
[7]	2020	ISCX-URL-2016 data	DL and ML	Fast.ia, Keras and Random Forest (RF)	97.55% (with RF)
This project	2021	modification of the VoMBWeb dataset	DL	FFNN	99.88%

A.3 Preprocessing results

In Figure 20 we can see the **correlation matrix between the initial features** given by the VoMBWeb Dataset. As comented in section 9 the network depended almost entirely in the length of the JS obfuscated web content. Note that in the attributes of the IP, TLD and continent only one variation is shown for demonstration purposes.

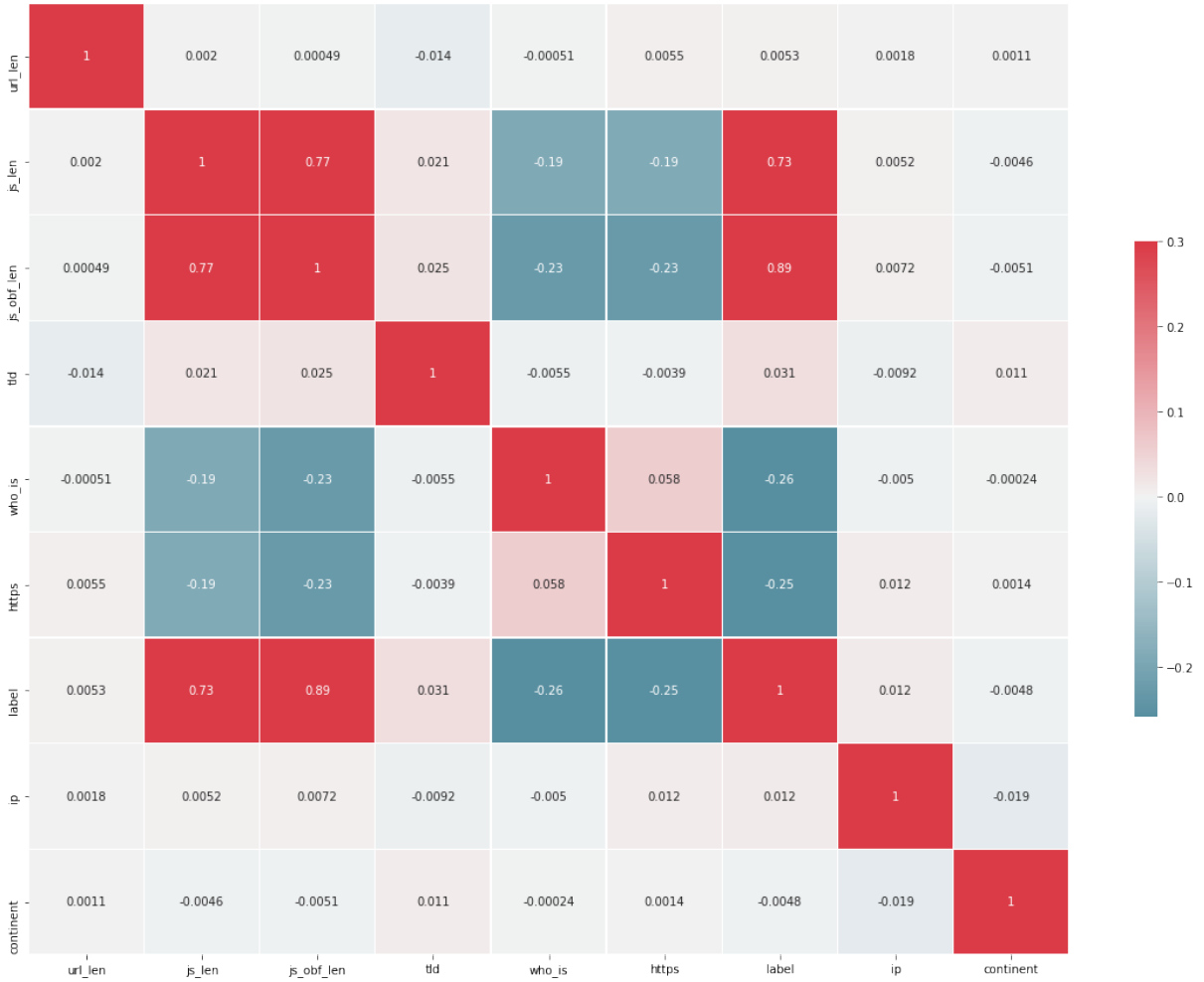


Figure 20: Correlation matrix of the initial dataset

The following code depicts the **URL entropy computation**.

```
import math

# Entropy computation
def get_entropy(row):
    string = row.url.strip()
    prob = [float(string.count(c)) / len(string) for c in dict.
        fromkeys(list(string))]
    entropy = sum([(p * math.log(p) / math.log(2.0)) for p in
        prob])
    return entropy

df['entropy'] = df.apply(get_entropy, axis=1)
```

The following code represents the computation of the **continent** and the feature **one-hot encoding** functions.

```
import pandas as pd
import numpy as np

# Convert a categorical value to ONE-HOT
def to_one_hot(df, col:str, exclude:set=None):
    if exclude is None:
        exclude = set()

    cs = set(df[col])
    for c in cs - exclude:
        df[c.lower()] = df[col] == c
        df[c.lower()] = df[c.lower()].astype(int)
    del df[col]

# Convert a binary value to ONE-HOT
def to_num(df, new_name:str, col_name:str, value:str):
    df[new_name] = df[col_name] == value
    df[new_name] = df[new_name].astype(int)
```

As we can observe there are two functions of one-hot encoding. The first one is to transform multi-class features (more than two values), whereas the second function transforms binary features into one-hot.

The continent code is depicted below.

```
from pycountry_convert import country_alpha2_to_continent_code
from ip2geotools.databases.noncommercial import DbIpCity

def country2continent (self, country):
        if country != 'Unknown':
            return country_alpha2_to_continent_code(country)
        return country

    def get_continent(self):
        response = DbIpCity.get(self.ip, api_key='free')
        return self.country2continent(response.country)
```

As we can observe firstly the function gets the city associated with the IP address and then converts the obtained country to a continent.

In Figure 21, we can observe the **preprocessing results histograms** between the dataset's features and we differentiate two colors. The blue color represents the benign label, whereas the orange color represents the malicious label. Each feature is plotted with the label attribute and its dispersion.

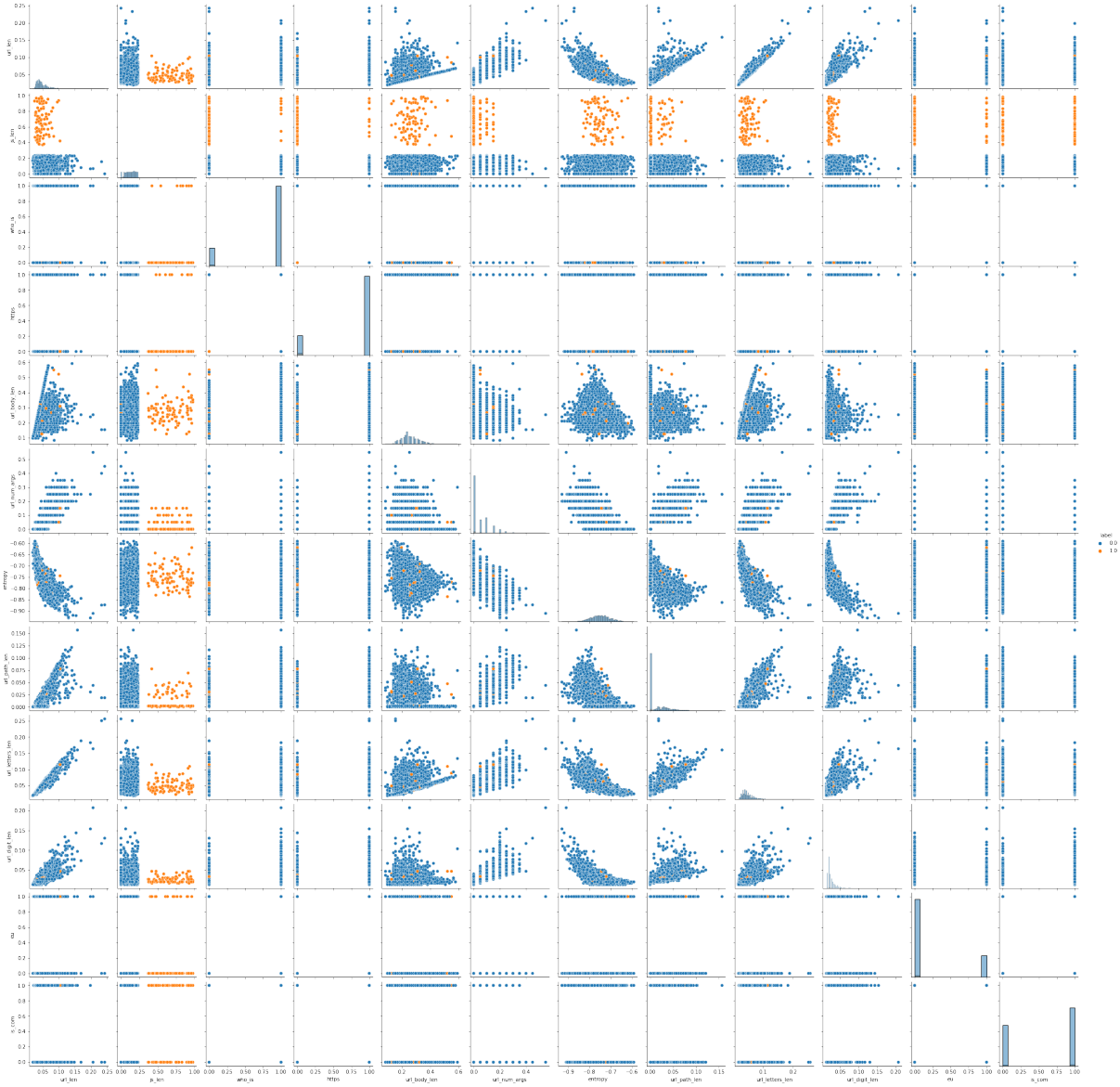


Figure 21: Preprocessed dataset parameter histograms

References

[1] CISCO. What is a firewall?, 2021.

[2] Santiago Ramos, Daniele Sartiano, and Abu Syeed Sajid Ahmed. Awesome-cybersecurity-datasets, 2021.

[3] Daiki Chiba, Kazuhiro Tobe, Tatsuya Mori, and Shigeki Goto. Detecting malicious websites by learning IP address features. In *Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on*, pages 29–39, 2012.

[4] Cho Xuan, Hoa Dinh, and Tisenko Victor. Malicious url detection based on machine learning. *International Journal of Advanced Computer Science and Applications*, 11, 01 2020.

[5] Joshua Saxe, Richard Harang, Cody Wild, and Hillary Sanders. A deep learning approach to fast, format-agnostic detection of malicious web content, 2018.

[6] Emine Uçar, Murat Ucar, and Mürsel İncetaş. A deep learning approach for detection of malicious urls. In *International Management Information Systems Conference*, 2019.

[7] Clayton Johnson, Bishal Khadka Ram B. Basnet, and Tenzin Doleck. Towards detecting and classifying malicious urls using deep learning. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 2020.

[8] Doyen Sahoo, Chenghao Liu, and Steven C. H. Hoi. Malicious URL detection using machine learning: A survey, 2019.

[9] Google. Chrome enterprise. `https://chromeenterprise.google/`, 2021. [Online; accessed 2-June-2021].

[10] WinTools. Wintools url blocker. `https://www.wintools.info/index.php/url-blocker`, 2021. [Online; accessed 2-June-2021].

[11] DNS Filter Company. DNS filter. `https://www.dnsfilter.com/`, 2021. [Online; accessed 2-June-2021].

[12] CloudFlare. Cloudflare gateway. `https://www.cloudflare.com/`, 2021. [Online; accessed 2-June-2021].

[13] CISCO. Cisco umbrella. `https://www.cisco.com/c/en/us/products/security/umbrella/index.html`, 2021. [Online; accessed 2-June-2021].

[14] TITAN HQ. Web TITAN. `https://www.webtitan.com/webtitan-cloud/`, 2021. [Online; accessed 2-June-2021].

[15] McAfee. Mcafee web gateway cloud service. `https://www.mcafee.com/enterprise/en-gb/products/web-gateway-cloud-service.html?AID=11552066&PID=6361382&SID=trd-es-1177665543907469300`, 2021. [Online; accessed 2-June-2021].

[16] Untangle. Ng firewall complete. https://www.untangle.com/shop/NG-Firewall-Complete/, 2021. [Online; accessed 2-June-2021].

[17] Amit Kumar Singh. Dataset of malicious and benign webpages, 2020.

[18] TensorFlow. tf.keras.optimizers.adam, 2020.

[19] Fiona Nah. A study on tolerable waiting time: How long are web users willing to wait? volume 23, page 285, 01 2003.

[20] Jakob Nielsen. *Usability Engineering*. Nielsen Norman Group, 1993.