Malicious website detection through Deep Learning algorithms

Norma Gutiérrez, Beatriz Otero, Eva Rodríguez, and Ramon Canal

Universitat Politècnica de Catalunya (UPC). Barcelona, Spain
{norma, botero, evar, rcanal}@ac.upc.edu

Abstract. Traditional methods that detect malicious websites, such as blacklists, do not update frequently, and they cannot detect new attackers. A system capable of detecting malicious activity using Deep Learning (DL) has been proposed to address this need. Starting from a dataset that contains both malevolent and benign websites, classification is done by extracting, parsing, analysing, and preprocessing the data. Additionally, the study proposes a Feed-Forward Neural Network (FFNN) to classify each sample. We evaluate different combinations of neurons in the model and perform in-depth research of the best performing network. The results show up to 99.88% of detection of malicious websites and 2.61% of false hits in the testing phase (i.e. malicious websites classified as benign), and 1.026% in the validation phase.

Keywords: Network attacks· Deep learning · Feed Forward Neural Network · Preprocessing.

1 Introduction

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, as stated in Abdulghani [1]. These attacks run malware in the target system, intending to take control of it. This article 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 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.

A defensive solution is developed 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. The main contributions of this work are the following:

– 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 DL NN that uses existing patterns in malicious web pages to detect malicious websites in real environments.

The system is divided into two main parts: the dataset creation and training of the model and the system's validation and 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 metrics. 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).

The remainder of this paper is organized as follows. Section 2 describes the related work. Section 3 and 4 illustrate the dataset and its preprocessing. Then, Section 5 explains the used Neural Network, and Section 6 describes the experiments and presents the results. We draw the main conclusions in Section 8.

2 Related work

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 characteristics associated with it, such as the related 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. [2] 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 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. [9] 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. [8] develop two DL models (CNN and LSTM) that add to a blacklist malicious URLs. 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. [3]. The article presents the same problem as in this project; 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. [4] propose a survey that gives a structural understanding of Malicious URL detection techniques using Machine Learning. The survey separates two types of families when detecting Malicious URLs: Blacklisting/heuristic approaches or Machine Learning techniques. The survey only talks about mathematical Machine Learning algorithms that previous literature has implemented, such as SVM, Logistic Regression, Decision Trees, and online learning. 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 an FFNN, which was chosen since it is the most common approach for supervised learning with binary classification datasets. Also, an 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 Table 1.

3 Dataset

To create a DL model capable of distinguishing malicious websites, we modified the existing dataset *Visualisation of Malicious & Benign Web-pages Dataset (VoMBWeb)* [6] 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 were extracted from each URL crawling the

Table 1: Summary of the related works to malicious website detection using ML/DL techniques

Publication year	Work reference	Dataset	ML or/and DL	Used algorithm	Maximum accuracy
2012	[2]	From blacklists and campus information	ML	SVM and RBM	85.7%
2018	[5]	Own compilation	ML and DL	FFNN and SVM	97.2%
2019	[8]	ISCX-URL-2016 data	DL	LSTM and CNN	98.86% with CNN
2020	[9]	Own compilation	ML	SVM and RF	99.7% with RF
2020	[3]	ISCX-URL-2016 data	DL and ML	Fast.ia, Keras and Random Forest (RF)	97.55% with RF

internet. The data is extracted from three separate categories: the IP, the URL and the content. The primary group is the URL group, parsed and treated as shown in Figure 1.

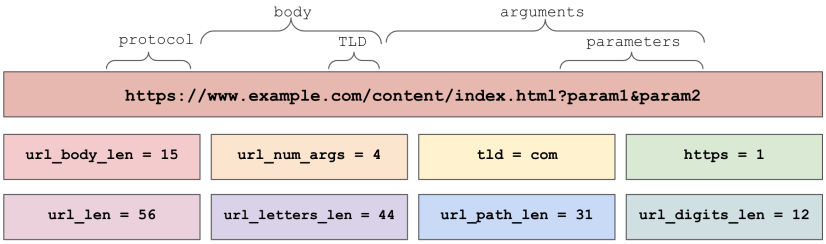


Fig. 1: Example of URL preprocessing

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 they 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.

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 website'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). 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 whole 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%

4 Preprocessing

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 particular 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 of passing the categorical feature into a table where each column represents a different value; hence, we created a new feature per existing continent. The column which continent corresponds to 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.

All URL, IP and *content* features were deleted since they are thoroughly represented in other features and do not give additional information. Furthermore, the IP address 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 deemed to be benign (good). This process is depicted in Figure 2.

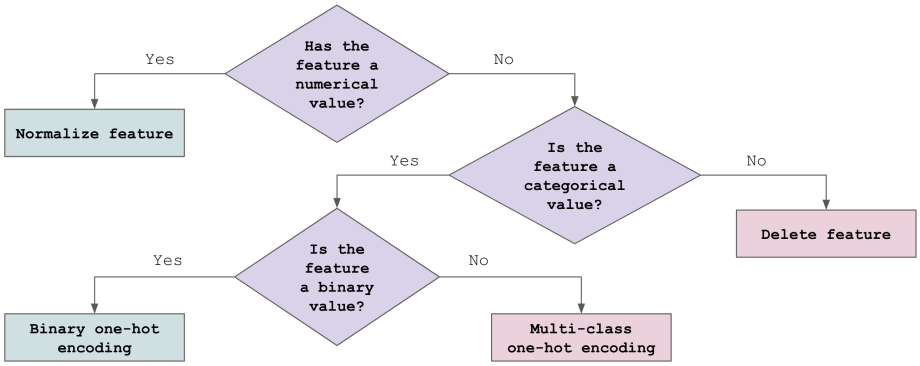


Fig. 2: Preprocessing flowchart

All the 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 to train the model with a vast amount of data. Then by testing 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.

5 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 3.

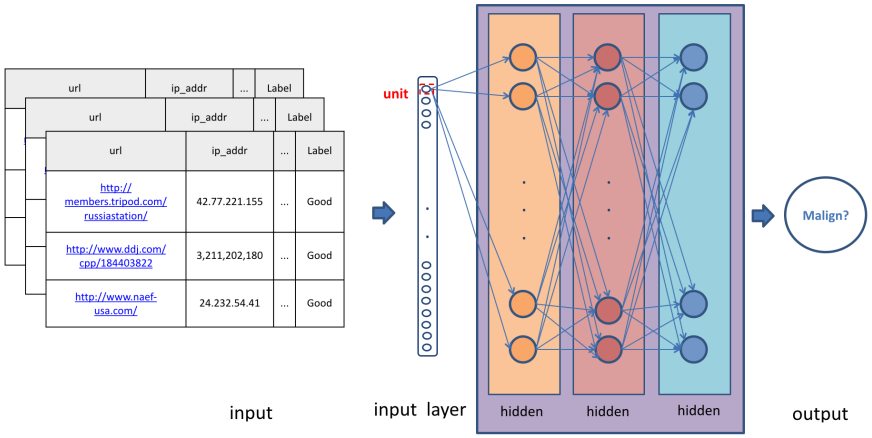


Fig. 3: Feed-Forward Neural Network representation

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 system's detection capability. Therefore, we apply this NN to demonstrate the effectiveness of the proposal.

An 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 several 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 the 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 the 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* [7], which achieves good perfor-

mance in few epochs. Finally, the number of epochs is set to 10, which is a good compromise between stability and over-training.

6 Experiments and Results

The correct implementation of the preprocessing was tested through the analysis of the correlation between attributes (correlation matrix). In the sample's preprocessing, it is crucial that parameters can be easily distinguishable by the network. Moreover, attributes are interconnected in the dataset. This interconnection can be seen in the feature correlation matrices. Performing these experiments, we can decide the optimal NN use, its parameters and identify non-useful attributes.

Secondly, 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 run 56 different FFNN ($CR_3^6 = \frac{(6+3-1)!}{3!(8-3)!} = 56$).

Furthermore, we analyzed the most frequently used metrics: accuracy, loss, Area Under the Curve (AUC), 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.

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

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.

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

Next, the f-score is defined as a mixture of the precision and recall formulas to evaluate the combined performance. F-score is defined as in Equation 3.

$$F - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3}$$

Where the precision is defined in Equation 4, and the recall is defined in Equation 5.

$$Precision = \frac{TP}{TP + FP} \quad (4) \qquad Recall = \frac{TP}{TP + FN} \quad (5)$$

Finally, the AUC shows the performance of a classification model. It is defined 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.

$$TPR = \frac{TP}{TP + FN} \quad (6) \qquad FPR = \frac{FP}{FP + TN} \quad (7)$$

The same metrics from the validation data was analyzed to extract further conclusions.

In the experiments, we perform 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.

Furthermore, Python was the language used to execute the system, and Keras from Tensorflow was used to create the DL blocking system. 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.

6.1 Preprocessing results

To test the data preprocessing and its relevance in the dataset, several experiments were conducted. Firstly, we parted from the VoMBWeb dataset [6] since it included some of the attributes we wanted to treat and generated in a non-simulated environment. Considering that Keras from TensorFlow needs as an input a tensor of NumPy arrays, the multi-class attributes were converted 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. The resulting values were very dispersed between the different attributes. There, it could be observed that the IP address 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 modified to remove this dependency and add some relevant features that enable the FFNN to train correctly. The JS obfuscated length attribute was removed from the original dataset, the

country feature was grouped into continents, and the TLD was treated by 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 from the input. The main goal was to find the balance between extracting trainable features and having unnecessary or misguiding information inserted into 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 4.

Fig. 4: URL subdivisions

Additionally, we decided to add three more attributes that gave connections between the URL attributes. Those features were the URL entropy, 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. First, to study the parameters' dependence and relevance, a new correlation matrix was performed with all the final preprocessed attributes. The correlation matrix can be seen in Figure 5.

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 a 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.

6.2 Neural Network 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.

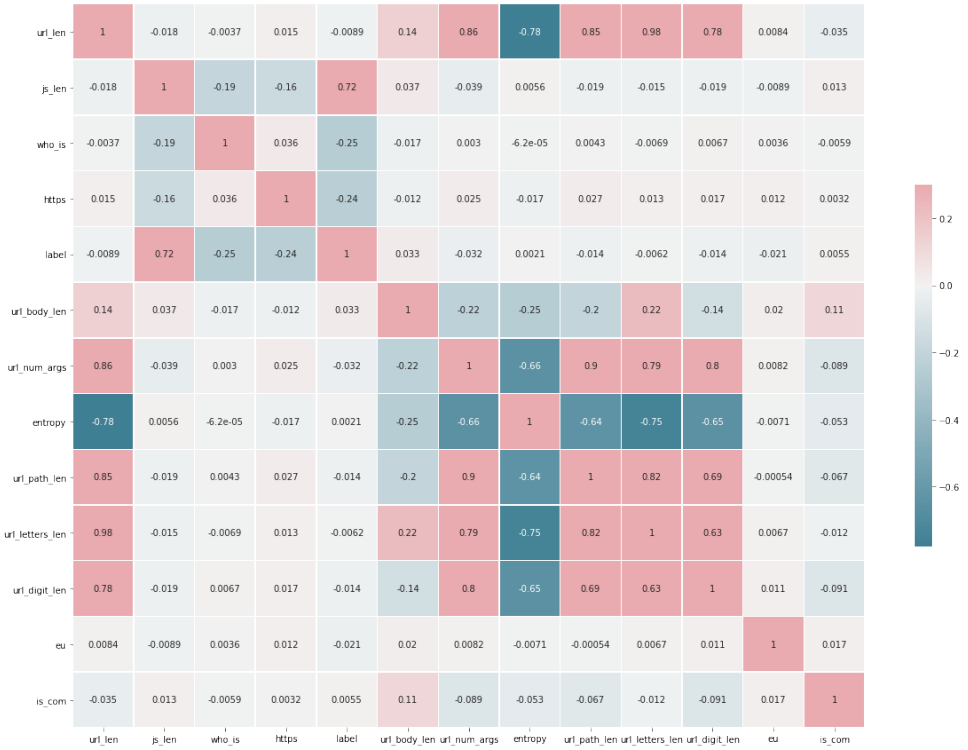


Fig. 5: Preprocessed dataset parameter correlation matrix

To finalize the network topology, we had to decide the number of layers the FFNN would have as the number of neurons in them. 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 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.

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

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

Neurons	Loss	Accuracy	AUC	F-score
16 - 8	0.31%	98.98%	99.95%	97.22%
16 - 8 - 16	0.30%	99.88%	99.93%	97.30%
16 - 8 - 16 - 8	0.31%	99.88%	99.95%	97.34%

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.

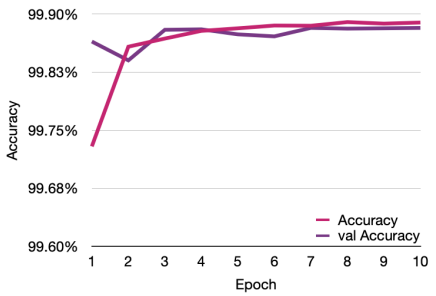
Table 4: Testing performance for the three best FFNNs with three layers and variation of the amount neurons per layer

Neurons	Loss	Accuracy	AUC	F-score
64 - 32 - 64	0.30%	99.88%	99.95%	97.42%
32 - 64 - 32	0.47%	99.88%	99.79%	97.22%
128 - 64 - 32	0.46%	99.87%	99.75%	97.01%

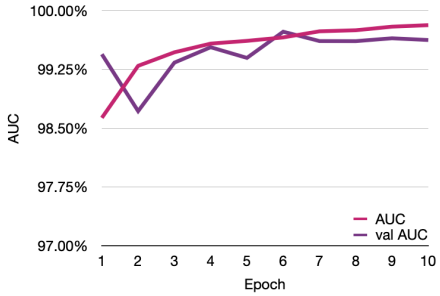
The best performing network (and the one used in the system) is the first one (64-32-64) since the loss decreases and f-score increases compared with the other two, and it is the most stable network with lesser differences between attributes than the others. Furthermore, the network is relatively small,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

Figure 6 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 AUC, accuracy and performance in all studied metrics. Consequently, the proposed system is very effective in detecting malicious websites.

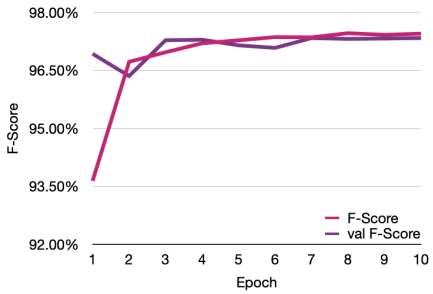
Finally, we compute the TP (True Positives), TN (True Negatives), FP (False Positives) and FN (False Negatives) of the selected network. The TP represents the samples that belong to a malicious website and is correctly classified. FN represents the samples that belong to malicious websites that are wrongly clas-



(a) Accuracy results



(b) AUC results



(c) F-Score results

Fig. 6: Mean performance for the best operating FFNN (64-32-64) in 20 iterations

sified. 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 (i.e. adding more data and re-training the network).

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 accuracy of 99.75%. These results show a clear identification and classification of malicious websites. Overall, these outcomes demonstrate the

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	1172698	48	1343
(%)	(95.070%)	(99.999%)	(0.001%)	(4.930%)
Validation	7623	341977	196	205
(%)	(97.381%)	(99.994%)	(0.006%)	(2.619%)

effectiveness of our system since almost all malicious websites will be filtered, and they will not penetrate the system.

Furthermore, a comparison with traditional Machine Learning algorithms, such as Random Forest (RF), Logistic Regression (LR) and Gaussian Naïve Bayes (GNB), was performed. The results show a similar accuracy and f-score, but the Loss and AUC results decrease considerably. Furthermore, the use of DL is chosen due to its supremacy in performance when using large quantities of data such as in the depicted project. Hence, DL trains quickly and effectively large subsets of data, contrary to traditional ML algorithms. The results for all approaches are depicted in Figure 6.

Table 6: Testing performance for RF, LR and GNB algorithms

Algorithm	Loss	Accuracy	AUC	F-score
RF	24%	99.88%	97.99%	97.34%
LR	16%	99.82%	96.43%	96.29%
GNB	19%	99.80%	97.27%	95.69%
FFNN (ours)	0.30%	99.88%	99.95%	97.42%

7 Conclusions

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. This work proposes a study on Malicious URL Detection using DL techniques. The proposed system evaluates the website's features, and it classifies them by preprocessing and entering them in a DL model. We evaluate a real dataset that extracts basic features of real websites (malicious and benign) to conduct the study. Furthermore, we measured the dispersion and correlation between the dataset's features to observe the label separability and the feature interconnections.

Moreover, we proposed an FFNN to compute the classification of the labels. The proposed mechanism achieves a 99.88% accuracy, 99.95% AUC 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, the results showed only a 1.026% of inaccuracy in wrongly classified malicious websites, thus decreasing the error.

As future works, the system can be improved by adding more data and retraining 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.

Acknowledgments. This work was supported in part by the Catalan Government, through the program 2017-SGR-962 and the RIS3CAT DRAC project.

References

1. Abdulghani, A.: Malicious website detection: A review. Journal of Forensic Sciences & Criminal Investigation **7** (02 2018). https://doi.org/10.19080/JFSCI.2018.07.555712
2. Chiba, D., Tobe, K., Mori, T., Goto, S.: Detecting malicious websites by learning ip address features. In: Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on. pp. 29–39 (2012). https://doi.org/10.1109/SAINT.2012.14
3. Johnson, C., Basnet, B.K.R.B., Doleck, T.: Towards detecting and classifying malicious urls using deep learning. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA) (2020). https://doi.org/10.22667/JOWUA.2020.12.31.031
4. Sahoo, D., Liu, C., Hoi, S.C.H.: Malicious url detection using machine learning: A survey (2019)
5. Saxe, J., Harang, R., Wild, C., Sanders, H.: A deep learning approach to fast, format-agnostic detection of malicious web content (2018)
6. Singh, A.K.: Dataset of malicious and benign webpages (2020). https://doi.org/10.17632/gdx3pkwp47.2, `https://data.mendeley.com/datasets/gdx3pkwp47/2`
7. TensorFlow: tf.keras.optimizers.adam (2020), `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam`
8. Uçar, E., Ucar, M., İncetaş, M.: A deep learning approach for detection of malicious urls. In: International Management Information Systems Conference (2019)
9. Xuan, C., Dinh, H., Victor, T.: Malicious url detection based on machine learning. International Journal of Advanced Computer Science and Applications **11** (01 2020). https://doi.org/10.14569/IJACSA.2020.0110119