Deep Approaches on Malicious URL Classification

Arijit Das[1], Ankita Das[2], Anisha Datta[3], Shukrity Si[4] and Subhas Barman[5]
Jalpaiguri Government Engineering College, Jalpaiguri, India
{ [1]ad81952, [2]ankidas44, [3]dattaanishadatta, [4]sukriti.si98, [5]subhas.barman }@gmail.com

***Abstract*—Malicious URLs are one of the biggest threats to this digital world and preventing it is one of the challenging tasks in the domain of cyber security. Previous research to tackle malicious URLs using hard-coded features have proven good indeed, but it comes with the limitation that these features are non-exhaustive and therefore detection algorithms fail to recognize new or unseen malicious URLs. However, with the deep learning revolution, this problem can be easily solved, since deep learning models extract features of their own by learning from patterns occurring in such URLs. In this paper, we have shown a comparative study of deep learning based architectures - simple RNN, simple LSTM and CNN-LSTM and how these methods can be effective for classifying URLs as malicious or benign. We have compared their performances on the basis of accuracy, precision and recall, where CNN-LSTM architecture outperforms the other two with an accuracy of 93.59%.**

***Index Terms*—Malicious URLs, cyber security, deep learning, simple RNN, simple LSTM, CNN-LSTM**

I. INTRODUCTION

With the rapid digital transformation taking place, most of our activities are happening online, thereby increasing the chances of online crimes.Attackers are making their way into the sensitive information by tricking users into revealing them , leading to theft of money, reveal of identity, blackmailing,installing malware in the user's system. Attacking techniques include explicit hacking attempts, drive-by download [5], social engineering, phishing,man-in-the middle, SQL injections, loss/theft of devices, denial of service etc.According to the research of [14] one-third of the URLs nowadays are malicious thus marking the impact it has on cyber-crimes. Now-a-days being successful in any field be it in business, education, sports an online presence proving their skills is necessary. People also need handful of resources which are just a few clicks away. As a result the importance of World Wide Web (WWW) is increasing.Websites have an address referring to them, which are known as URLs (Uniform Resource Locator).There are client side or the users and server side or the providers.The connection between the client and server is protected by some protocols or rules, yet they are vulnerable to the outsiders who have a sharp intention of attacking them.A URL constitutes of two major components- 1. Protocols identifier (it helps to find what protocol to use) and 2. Resource name (IP address or name of the domain where resource is present). Now as the modern technology advances, the risk of corruption increases.As a first step if we can detect which of the URLs are malicious and which are not a great deal of attacks can be prevented,because most of the times malicious URLs are the door-ways for these attacks. So we can realize the importance of the topic, as a result a lot of research work has also been done on this problem. But most of them used traditional machine learning methods.A few tried the deep learning methods.In our work we have used three different architectures of RNN, LSTM and CNN-LSTM for this classification task.

This paper is divided into sections, Section II discusses similar works of other researchers. Section III describes the dataset. Section IV and V describe the pre-processing pipeline and methodology respectively while the results and visualizations are shown in section VI. And the last section concludes and discusses the future scope of the work.

II. RELATED WORK

There have been quite a lot of work on malicious URL detection using the machine learning techniques which use features to predict the outcome.A good representation of these features is essential for better training of the model.The survey paper [17] has summarized the features used by many of the previous works.They are as follows - Black Listing features, Lexical features, Content based features and Host based features. Black listing features is one of the simplest but, suffers from the problem of maintaining exhaustive up-to-date lists also algorithmically generated URLs go undetected so this technique does not perform well alone and is used along with the other methods.

Lexical features [20] works behind the motivation that based on how the URL appears the algorithm should be able to identify.The commonly used lexical features include the statistical properties of URLs such as length of each URL,length of each component in the URL or the number of special characters used and also the order in which they occur. Host-based features [10] include IP Address properties, WHOIS information, Location, Domain Name Properties, and Connection Speed.Content based features [20] require downloading the entire web-page thus is heavy-weighted but,gives a better estimate.

In the recent years the usage of Short-URLs [16] have become really common, where links are to be shared the number of characters are limited, this has also become a popular obfuscation technique because the services generating such URLs rely mainly on blacklisting techniques(not much effective).Other features used designed heuristics like Page-based features (Page rank, quality, etc.), Domain-based features (presence in white domain table), Type-based features (obfuscation types) and Word-based features(presence of keywords such as "confirm", "banking", etc.).

TABLE I: Samples from our dataset

URL	*Label*
gurl.com/category/your-life	Benign
lazada.co.id/sanken-official-store	Benign
codeweavers.com/account/downloads	Benign
vvorootad.top/admin.php?f=1.dat	Malicious
fryzjer.elblag.pl/dfr/sercurity.htm	Malicious
keepgrowing.net.br/sial/New%20folder%20file	Malicious

Machine learning algorithms like batch learning, online learning [4] ,representation learning [13] [21] have been used in many previous works.Batch learning assumes that the whole training set is available before hand to which classification algorithms like SVM [1] [2] [15], Naive Bayes [9] [6], Decision trees [11] [6] along with Ensemble learning techniques have been applied.Whereas in online-learning the predicted target values are compared against actual ones and accordingly the parameters are updated to better train the model.Representation of data [12] [3] [18] plays a major role in deciding the efficiency of the model, with so many features sparsity in the data increases, so feature selection techniques have been applied to the data in order to train the models faster and to give good results. [7].

Given the above URL detection techniques ,there are a lot of problems that come into the way, the Google Search Engine crawls more than 20 billion URLs a day [19], with so many new URLs getting generated everyday, training our models on the whole set is not possible,also proper extraction of features and feature representation play a major role in model training and manual handling of these factors with such a huge volume is difficult. Our model here takes care of the latter 2 problems of feature representation and extraction of features.Unlike in machine learning learning, the features need not be explicitly specified for Deep Learning, on providing the input and target values it updates the weights and biases,to train the model.

III. DATASET

For our research, we used a balanced dataset with 50% malicious and 50% benign, containing 194798 URLs in total. The URLs were scraped from open data sources, among which many didn't include the protocol, and so the protocol (http:// or https://) and the sub domain (e.g. www) was removed from the URL string if applicable. Every URL in the dataset has a corresponding label assigned to them - 1 or 0, where 1 indicates the URL is malicious and 0 indicates it is benign. Few of our data samples is given in Table I.

For training and validation, the dataset was splitted - 70% of the data used for training and the remaining for validation. The data-distribution is given in Table II.

IV. PREPROCESSING

To prepare our model for distinguishing between malicious and benign URLs, our model must be trained on the dataset. However our model does not recognize raw URLs or any sort of alphanumeric text or symbols. So, in order to make the URLs training worthy, the URLs must be passed through a pre- processing pipeline, where the URLs pass through three stages - tokenization, character-to-index mapping and sequence-padding.

TABLE II: Data Distribution

Data	*Training*	*Test*
Malicious	68179	29220
Benign	68179	29220

- **Tokenization -** Tokenization is a compulsory pre-processing step for almost any NLP related task. It is the process of chopping a text sequence into smaller linguistic units called tokens (words, punctuations, numbers, special symbols, etc.). In our research, every single character (letters, special symbols, numbers, etc.) of the URL is considered as token.
- **Character-to-index mapping -** The tokens generated from the previous step are just alphabets, digits and special characters. However, our model only understands numeric values and so our tokens need conversion. As a result, we assign a unique integer id to each character in our vocabulary and every list of tokens is mapped to their corresponding list of ids in this pre-processing step.
- **Sequence-padding -** This process is mostly carried out to help us train the model with the numeric data programmatically. Sequence padding is done by appending additional zeros with the token vector, so as to make the all the vectors equal in shape, thus aiding in feeding multiple data into our model as batches.

A. EMBEDDING

After the three steps of preprocessing, we have a list of integers corresponding to each URL in our dataset, where each of the integer represents a unique token in our vocabulary. However this integer representation of tokens does not contain any information regarding our URLs. So, it is necessary to convert these integers to some sort of encodings, which will help our model train better on our dataset. One such commonly used representation is one-hot encoding, in which each integer is expressed as an array of length same as the size of the vocabulary, placing "1" where the vector index is equal to integer and "0"s in remaining places. But these encodings have several limitations - first the size of each encoding can be considerably large depending upon the vocabulary, thus resulting in very large and sparse matrices, making the training procedure inefficient. Second and the most important shortcoming is that these representations do not consider the relationship between the tokens, i.e. when two similar tokens are plotted in vector space, they are far apart from each other.

Thus we have used embeddings. Embeddings are dense vector representations which considers the similarity between the tokens. This has been implemented using keras as a new layer to our architecture, where each token is represented by a vector of a fixed length, initialized with some random values which get trained along with our model by repeated forward and backward passes according to the context where it has been used. The keras embedding layer takes in 3 arguments - input-dim, output-dim and input-length indicating the total vocabulary size, the embedding vector size,the length of the input sequence respectively.The input shape of this embedding layer is (batch size, sequence length) whereas the output shape is (batch size,sequence length,output dim).

V. METHODOLOGY

A. SIMPLE RNN

As the term "recurrent" describes the recurrent neural networks having recurring units i.e. the contextual information from the past (saved as hidden units) along with the present input is used in deciding the next output in the sequence. With the hyper parameters remaining the same the recurrent unit is repeated for a fixed number of times (in most cases it is the sequence length).
The information gets fed into the RNN in the form of sequences in which each embedding, x_t goes in as input at each time step along with the previous hidden unit h_{t-1} which carries information from the past. The next hidden unit h_t and output y_t is calculated as:

$$h_t = \tanh(W_a \cdot [h_{t-1}, x_t] + b_c)$$
$$y_t = \sigma(W_{hy} \cdot h_t)$$

The loss is then calculated at each time step and is backpropagated through time, the weights and biases are updated accordingly. In the above equations, W_a, W_{hy} refers to the trainable randomly initialized weights, while b_c the corresponding bias.

B. SIMPLE LSTM

RNNs have proven to be quite advantageous over simple neural networks, especially when dealing with sequence data or any NLP tasks. However RNN comes with its own limitations. Theoretically RNN takes care of the contextual dependencies present in the text sequences, but to be precise it only takes into consideration the immediate or the most recent information. LSTMs [8] on the other hand make use of not just the immediate previous context, but it retains the useful information from the past to be used later.In short, it takes care of long term dependencies.

If we take a single LSTM unit and try understand the working, we can see there are three inputs entering each cell - x_t, h_{t-1} and C_{t-1} representing the current input (token embedding), previous hidden state and previous cell state respectively. The cell state is what makes LSTMs special. It is a horizontal wire running through the top of the unit which acts as a conductor of information, and in each LSTM cell it passes through, some information is added to or removed from this, thus making all the relevant context available to the future time steps. The information that is added to or discarded from the previous cell state is determined by a series of three mathematical doorways inside each cell.

The first doorway is a sigmoid function which outputs a number d_t between 0 and 1 and this value is then multiplied with the existing cell state C_{t-1}, thus determining what fraction of previous information is to be discarded.

$$d_t = \sigma(W_d \cdot [h_{t-1}, x_t] + b_d)$$
$$C_{t-1} = C_{t-1} * d_t$$

The following doorway is a combination of both sigmoid and tanh function - the tanh function generates the new information $\bar{C}_t$, learned from the input in the current cell and the output from the sigmoid u_t is to filter how much of the newly learnt context is to be added to the cell state.

$$u_t = \sigma(W_u \cdot [h_{t-1}, x_t] + b_u)$$
$$\bar{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$C_t = C_{t-1} + u_t * \bar{C}_t$$

The last doorway is again a combination of both sigmoid and tanh - the tanh producing the output z_t to be delivered to the next cell from the updated cell state C_t and the sigmoid o_t optimally filtering this output as the next hidden state to the next cell.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$z_t = \tanh(C_t)$$
$$h_t = o_t * z_t$$

In the above equations, W_d, W_u, W_c and W_o refers to the trainable randomly initialized weights, while b_d, b_u, b_c and b_o the corresponding biases.

C. CNN

Convolutional neural networks (CNN) is a special type of deep learning algorithm, which have proven quite beneficial in the past few years, because of its ability to share weights by taking advantage of the local dependencies between neighbouring values in both image as well as sequence data. For dealing with images, mostly 2D or 3D convolutional layers are used, however since we are working with text, we have used 1D convolutional layer, which has proven to be quite effective, especially on time-series or sequence data.

A 1D or 2D or 3D convolutional layer works in exact similar manner, if seen mathematically. The only difference lies in the kernel shape and the sliding operations. The input $I \in \mathbb{R}^{M \times l}$ here is a list of token encodings, where M represents the sequence length of the URL(preprocessed to have a fixed length) and l denotes the embedding vector size (32 in our case). A kernel $H \in \mathbb{R}^{K \times l}$ is taken, where K is the height of the window and unlike 2D convolutional operation, the kernel slides only along the height.While sliding over the input data keeping a stride of 1, element-wise multiplication takes place between the kernel weights and the masked underlying matrix $M \in \mathbb{R}^{K \times l}$, followed by addition of the terms. The operation can be formulated as -

$$z = (\sum H \odot M) + b$$

where $b \in \mathbb{R}$ refers to bias term and $\odot$ denotes element-wise multiplication between two matrices. This singular value z generated is then passed through a non linear activation function (usually $ReLU$).

$$O = ReLU(z)$$

However, just by using a single kernel, our model is restricted to extract just one type of feature. Thus, to take a wide range of features into consideration, we use 256 different kernels to train our model over 256 different features.

The output of the convolution layer is again a 2D matrix, whose each of the 256 columns represents features mapped from each different kernel. This output is then fed to a maximum pooling layer defined with a window of height 4, where the number of parameters are reduced to approximately one-fourth, thus preventing over-fitting and reducing computational cost. This pooling operation runs on each feature map independently where the window is slided over each column of the matrix and only the maximum of the underlying values is extracted, discarding the others.

D. CNN-LSTM

This method is a fusion of both the above mentioned algorithms. The URL embeddings are passed to a CNN which extracts the relevant information, which are then fed to a LSTM, where the model learns to differentiate between the URLs, considering the contextual dependencies among them. The advantage of this hybrid model is that it learns to exploit the advantages of both the architectures and it is seen to outperform both of them. To further improve the results, we have appended an additional classifier at the end, which is a 2-layered simple neural network aiding in distinguishing between benign and malicious URLs, based on the features extracted from the hybrid model.

E. SIGMOID LAYER

The final set of features, O extracted from our model are fed into a sigmoid layer which determines the finally classified class for the input URL through a series of two mathematical operations. In the first step, a weighted sum of the input features is taken -

$$h = \sum W_h \cdot O$$

where W_h refers to a set of randomly initialized trainable weights. In the next operation, this singular value h generated is passed through a sigmoid or logistic activation function, which is expressed as -

$$S(h) = \frac{1}{1 + e^{-h}}$$

The output singular value $S(h)$ is a floating point number bounded between 0 and 1, symbolizing the probability of the URL being malicious. This seems to be the perfect non-linear activation function, since we are dealing with a binary classification problem. If the output value is closer to 1, we classify the URL as malicious, else it is benign.

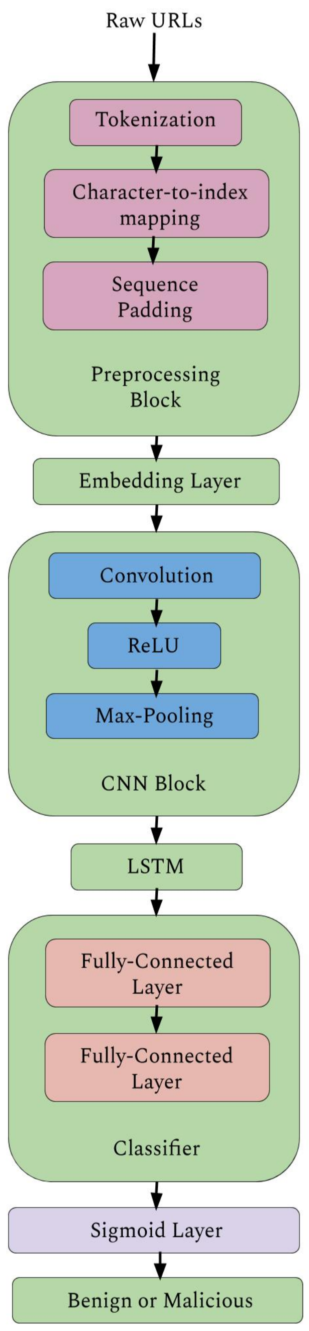
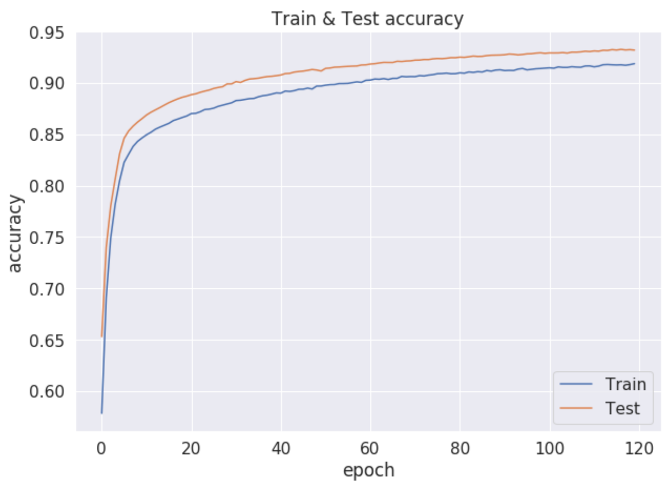


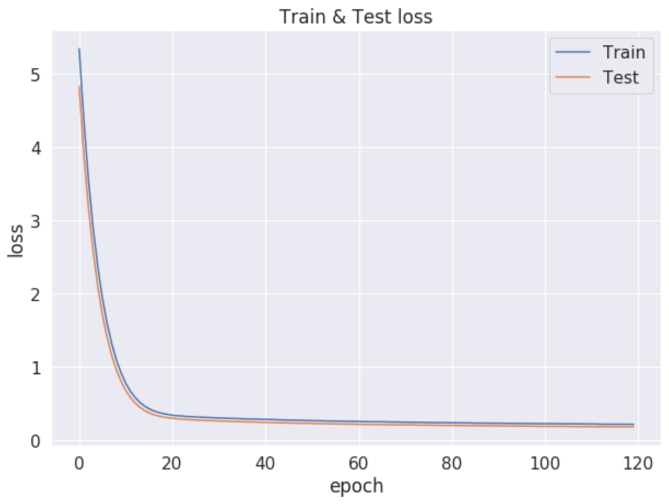
Fig. 1: Architecture

VI. EXPERIMENTAL SETUP AND RESULTS

The complete architecture of CNN-LSTM hybrid model is shown in Figure 1. Our model is trained for 120 epochs, where in each epoch, all the preprocessed URLs along with their corresponding class labels (0 or 1) are fed in batches

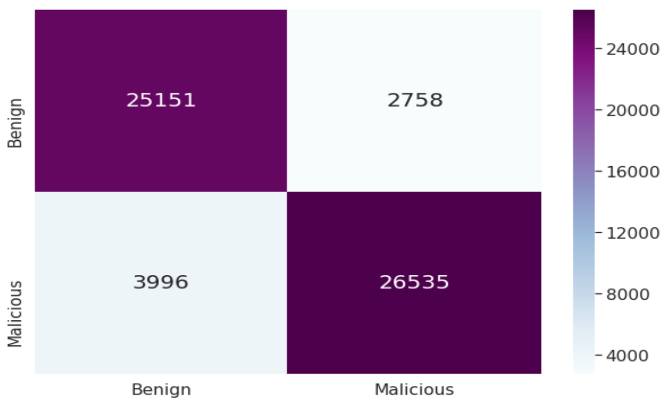


(a)

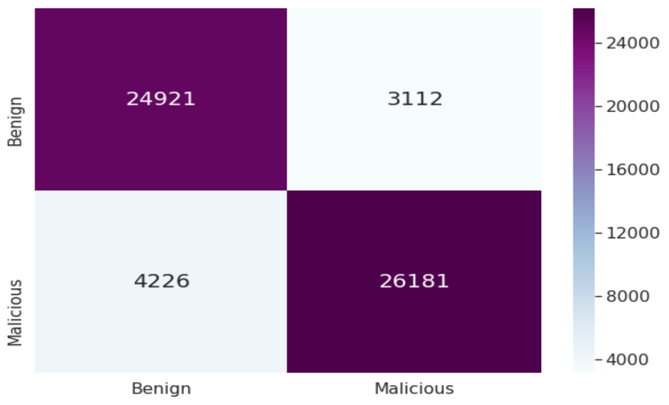


(b)

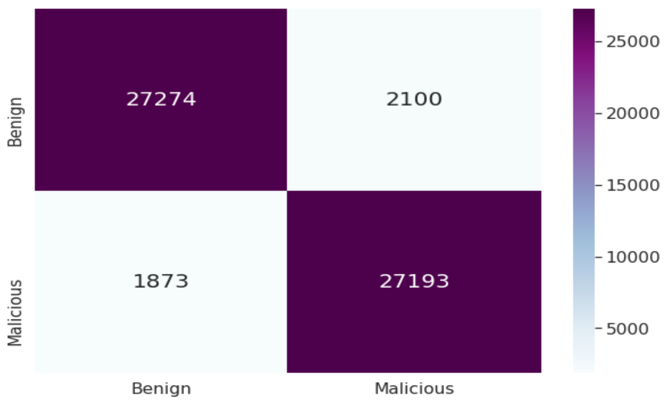
Fig. 2: (a) Accuracy curve (b) Loss curve on training & test (validation) data



(a)



(b)



(c)

Fig. 3: Confusion Matrix for (a) Simple RNN, (b) Simple LSTM & (c) CNN-LSTM

of size 32. The model predicts its own set of labels and the binary-crossentropy loss L is calculated between these predicted and the actual classes. The binary-crossentropy loss function can be expressed as -

$$L = \frac{1}{N} \sum t \cdot \log(y) + (1-t) \cdot \log(1-y)$$

The loss calculated is then optimized using Adam optimizer with a learning rate of 0.0001. After every batch is fed, the loss is back-propagated and the randomly initialized weights in our model get updated in the direction of steepest descent , in order to minimize the loss and get a better model performance. The accuracy and loss curve is shown in Figure 2

After training, we have validated our model on a test set of 58440 URLs. To demonstrate the effectiveness of our model, we have evaluated it using various performance metrics - accuracy, precision, recall and F1 score. It is seen that CNN-LSTM hybrid architecture surpasses the other algorithms in terms of all the metrices. A comparison of our proposed model with other popular deep learning architectures is shown in Table III. The confusion matrix is a plot of true values in the vertical axis verses predicted values in the horizontal axis. The diagonal elements show how many values have been correctly predicted. The confusion matrices shown in Figure 3 also confirms the effectiveness of CNN-LSTM model with maximum number of correctly predicted values.

Model	Accuracy	Precision		Recall		F-Score	
-	-	Benign	Malicious	Benign	Malicious	Benign	Malicious
RNN	0.88	0.86	0.91	0.90	0.87	0.88	0.89
LSTM	0.87	0.86	0.89	0.89	0.86	0.87	0.88
CNN-LSTM	0.94	0.94	0.93	0.93	0.94	0.94	0.94

TABLE III: Comparison between RNN, LSTM & CNN-LSTM

VII. CONCLUSION AND FUTURE SCOPE

Malicious URL detection and deep learning are two of the most trending domains of research and in this paper we have shown how to tackle the former with the latter. Experiments with three deep learning architectures are conducted - simple RNN, simple LSTM and CNN-RNN and a comparative study is shown based on their precision, recall and F-score. CNN-LSTM surpasses other two algorithms with an accuracy of 93.59%. However, considering the risks malicious URLs possess, this number is very less and further improvements must be done. With the advancement in technology, several novel and complex architectures and preprocessing techniques have been founded in recent years, some of which have been claimed to be better than the discussed ones. Thus, this leaves room for future research and better performing models are expected using these complex algorithms.

REFERENCES

[1] Yazan Alshboul, Raj Kumar Nepali, and Yong Wang. Detecting malicious short urls on twitter. In *AMCIS*, 2015.

[2] Betul Altay, Tansel Dokeroglu, and Ahmet Cosar. Context-sensitive and keyword density-based supervised machine learning techniques for malicious webpage detection. *Soft Computing*, 23(12):4177–4191, 2019.

[3] Ram B Basnet, Andrew H Sung, and Quingzhong Liu. Feature selection for improved phishing detection. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 252–261. Springer, 2012.

[4] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[5] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web*, pages 281–290, 2010.

[6] Giovanni Vigna Davide Canali, Marco Cova and Christopher Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. Proceedings of the 20th international conference on World wide web. ACM., 2011.

[7] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. Journal of machine learning research, vol. 3, no. Mar, pp.1157–1182, 2003.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Yung-Tsung Hou, Yimeng Chang, Tsuhan Chen, Chi-Sung Laih, and Chia-Mei Chen. Malicious web content detection by machine learning. *expert systems with applications*, 37(1):55–60, 2010.

[10] Bin B Zhu Hyunsang Choi and Heejo Lee. Detecting malicious web links and identifying their attack types. 2nd USENIX conference on Web application development. USENIX Association., 2011.

[11] S. Savage J. Ma, L. K. Saul and G. M. Voelker. Beyond black-lists: learning to detect malicious web sites from suspicious urls. Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2009, pp. 1245–1254., 2009.

[12] Pranam Kolari, Tim Finin, Anupam Joshi, et al. Svms for the blogosphere: Blog identification and splog detection. In *AAAI spring symposium on computational approaches to analysing weblogs*, 2006.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[14] Bin Liang, Jianjun Huang, Fang Liu, Dawei Wang, Daxiang Dong, and Zhaohui Liang. Malicious web pages detection based on abnormal visibility recognition. In *2009 International Conference on E-Business and Information System Security*, pages 1–5. IEEE, 2009.

[15] Samuel Marchal, Jérôme François, Radu State, and Thomas Engel. Phishstorm: Detecting phishing with streaming analytics. *IEEE Transactions on Network and Service Management*, 11(4):458–471, 2014.

[16] Raj Kumar Nepali and Yong Wang. You look suspicious!!: Leveraging visible attributes to classify malicious short urls on twitter. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 2648–2655. IEEE, 2016.

[17] Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. Malicious url detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179*, 2017.

[18] Peilin Zhao Steven CH Hoi, Jialei Wang and Rong Jin. Online feature selection for mining big data. 1st intl. Workshop on big data, streams and heterogeneous source mining:Algorithms, systems, programming models and applications. ACM., 2012.

[19] Danny Sullivan. Google: 100 billion searches per month, search to integrate gmail, launching enhanced search app for ios. *Search Engine Land*, 2012.

[20] Lawrence K Saul Sushma Nagesh Bannur and Stefan Savage. Judging a site by its content: learning the textual, structural, and visual features of malicious web pages. 4th ACM Workshop on Security and Artificial Intelligence., 2011.

[21] Hiba Zuhair, Ali Selamat, and Mazleena Salleh. Feature selection for phishing detection: a review of research. *International Journal of Intelligent Systems Technologies and Applications*, 15(2):147–162, 2016.