IJCNN 2019. International Joint Conference on Neural Networks. Budapest, Hungary. 14-19 July 2019

NeuralAS: Deep Word-Based Spoofed URLs Detection Against Strong Similar Samples

Jing Ya[1,2], Tingwen Liu[1,2], Panpan Zhang[1,2], Jinqiao Shi[1,2], Li Guo[1,2], Zhaojun Gu[3]
[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3]Information Security Evaluation Center of Civil Aviation, Civil Aviation University of China, Tianjin, China
Email: {yajing, liutingwen, zhangpanpan, shijinqiao, guoli}@iie.ac.cn, 15620968007@163.com

***Abstract*—Spoofed URLs are associated with various cyber crimes such as phishing and ransomware *etc*. Most existing detection approaches design a set of hand-crafted features and feed them to machine learning classifiers. However, designing such features is a time consuming and labor intensive process. This paper proposes an approach named NeuralAS (Neural Anti-Spoofing) by segmenting URLs into word sequences and detecting spoofed URLs with recurrent neural networks. As a result, NeuralAS can perform detection with high-abstract and poor-interpretable features learned automatically, and achieve accurate detection with contextual information in sequences. We also propose a novel method to construct indistinguishable data sets of strong similar samples, which can be used to evaluate the robustness of different approaches. Extensive experimental results show that NeuralAS works well on spoofed URLs detection, and has a significant effectiveness and robustness even on strong similar data sets.**

***Index Terms*—Spoofed URLs, Strong Similar Samples, Recurrent Neural Networks**

I. Introduction

A Uniform Resource Locator (URL), colloquially termed a web address, is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. It typically reveals key information about the site, *e.g.*, the host of the server, the path of the resource and some PHP information as well. Most users usually glance at URLs but not seriously examine the information they contain. Cyber crooks utilize users' neglection to construct spoofed URLs by mixing terms of normal URLs, as shown in Table I. Such spoofed URLs are often sent to users through emails, chat sessions, or other websites. They are usually associated with various cyber crimes such as phishing and ransomware [1]. Spoofed URLs provide "entry points" to lure crime victims for attackers. So, detecting spoofed URLs is a critical component for discovering cyber attacks like phishing and ransomware in networks.

To date, many detection approaches based on URL analysis have been proposed [1]–[5]. Most of these approaches design a set of hand-crafted features based on the URLs (*e.g.*, lexical features and statistical frequencies of URLs *etc*), and feed them to a binary Machine Learning (ML) classifier. Unfortunately, designing such features is time consuming and labor intensive, although the process can be done offline.

TABLE I: Examples of **S**(poofed) URLs and **N**(ormal) URLs

#	URL	Label
1	http://`paypal`.services-users.com/webapps /4dcfe/websrc	S
2	http://resturkrant.com/wp-includes/pomo/`p aypal`/index.php	S
3	http://login-confirmation.gq/`www.paypal.com`/help/	S
4	https://www.`paypal`.com/uk/signin	N
5	http://time.com/4197704/tech-`paypal`-usersearnings/	N
6	https://twitter.com/`paypal`	N

Recently, several Deep Learning (DL) classifiers have been proposed to perform the detection with the character sequences of URLs [6], [7].

However, the existing detection approaches based on URL analysis experience the following three main limitations. First, prior ML classifiers are sequence-independent, although some lexical and statistical features of URLs are designed based on words inside URLs. The rich contextual information is discarded, and attackers can easily confuse the classifiers by putting some common words in fixed positions of URLs. Second, prior DL classifiers work with character sequences of URLs as inputs, which ignore the meaningful and readable words inside URLs. For the same URL, character sequences are much longer than word sequences. Then it may be harder for DL classifiers to memory the contextual information during the training process. The detection accuracy experiences a decrease penalty or the training set should be expanded. Third, experimental data sets used in existing approaches are highly skewed: spoofed samples are from famous blacklists (*e.g.*, PhishTank[2]), and normal samples are chosen independently from DMOZ[3] or websites of high Alexa Rank[4]. Thus, spoofed samples have very few common words with normal samples. This contradicts the aforementioned fact that attackers try to forge spoofed URLs similar with normal ones. As a result, a weak classifier may achieve as high accuracy as prior detection approaches.

In this paper, we propose a novel approach named NeuralAS (Neural Anti-Spoofing) to address the above three limitations. NeuralAS takes word sequences of URLs as inputs, trains with Recurrent Neural Networks (RNN), and achieves high

[1]https://en.wikipedia.org/wiki/Spoofed_URL

[2]http://www.phishtank.com
[3]https://dmoztools.net/docs/en/rdf.html
[4]https://www.alexa.com/topsites

accuracy even on strong similar samples.

We first propose a word-level bidirectional RNN to detect spoofed URLs. The key insight is to make full use of contextual information inside the word sequences of URLs. We use BiLSTM (Bidirectional Long Short-Term Memory) to automatically learn high-abstract and poor-interpretable features. LSTM is a special kind of RNN, using three gates (*i.e.*, forget gate, input gate and output gate) to protect and control the cell state over sequences. In our problem, the state status is intended to capture combinations of words that are discriminating spoofed URLs from normal ones. This flexible architecture generalizes manual feature extraction and learns dependencies of one or multiple words, whether in succession or with arbitrary separation. BiLSTM is to connect two hidden layers of opposite directions to the same output. By this structure, the output layer can get context information of words from front and back positions in a URL.

Second, we propose a novel method to construct indistinguishable data sets of strong similar samples. The spoofed samples are also from famous blacklists, just same as traditional methods. We choose some representative words from the word sequences of spoofed samples, and use the search operator “allinurl:” to instruct Google to search for pages that contain the chosen words in the URL. Then, we obtain the URLs of the top 100 searching results and use the ones that are labelled with “clean site” by VirusTotal[5] as normal samples. We reduce the problem of choosing representative words to a problem of frequent itemset mining. In this way, we can construct a data set of strong similar samples, which are in consistence with the reality.

We evaluate our work on two datasets constructed by traditional methods and our new proposed method respectively, and conduct an extensive set of comparisons with classical ML classifiers and other DL classifiers. Plenty of experimental results show that our work achieves higher accuracy than prior classifiers, even if on strong similar samples. Moreover, our work is robust and achieves the least decrease in accuracy with the decrease of the number of training samples.

The main contributions of this paper are summarized as follows:

- We propose a word-level BiLSTM for spoofed URLs detection, which makes full use of contextual information inside the word sequences of URLs.
- We propose a novel method to construct indistinguishable spoofed data sets of strong similar samples, which can be used to evaluate the robustness of different detection approaches.
- We evaluate our work on real-world data sets and conduct an extensive set of comparisons with other classifiers. The results show that our detection approach has a significant effectiveness and robustness.

[5]https://www.virustotal.com

II. Related Work

A. Spoofed URLs Detection

A spoofed URL is a web address that illuminates an immense amount of deception through its ability to appear as a normal URL. Spoofed URLs detection is a critical component for discovering cyber crimes like phishing and ransomware in networks. Many URL analysis based approaches used in detecting phishing and ransomware [8] can be used to detect spoofed URLs as well. Most of the existing detection approaches design a set of hand-crafted features of URLs, and feed them to a binary ML classifier. Such features can be divided into two categories: character-level features and word-level features.

Character-level Features. Anh *et al.* derived lexical features using heuristics with the objective of being obfuscation resistant [1]. They proposed five categories of features: URL-related features (length, keywords, *etc.*), domain name features (length, IP address, *etc.*), directory related features (length, number of subdirectory tokens, *etc.*), file name features (length of filename, number of delimiters, *etc.*), and argument features (length of the argument, number of variables, *etc.*). D. Kevin and Minaxi studied the characteristics of phishing URLs, discovering that various lexical features such as length, frequencies of certain characters, and the presence of certain brand names in URLs were associated with spoofed URLs [9]. Leyla *et al.* designed two lexical features in their detection approach, namely percentage of numbers, and proportion of the longest English word [10]. However, they did not use individual words as features. They compare the performance of several distance metrics, including KL-distance, edit distance and Jaccard measurement to identify new domains that are suspiciously similar (measured via distance) to detect malicious domains. Weibo *et al.* proposed distance based features, called domain brand name distance and path brand name distance [11] . These features are essentially types of edit distance between strings aimed at detecting those URLs which try to fake famous brands or websites.

Word-level Features. He *et al.* used several Markov models on different text corpora to create features, and got the result that legitimate URLs often contain meaningful English words [12], while many malicious domain names do not. Justin *et al.* selected words as features, but they limited their vocabulary of words to those that are separated by a specific set of punctuation marks in the domain name or URL path [13]. Aaron *et al.* tried to enhance the lexical features by considering the usage of bigram features [14], *i.e.*, they construct a dictionary, where in addition to single-words the presence of a set of 2-words in the same URL is considered a feature. In addition, they record the position of sensitive tokens and bigrams to exploit the token context sensitivity. Marchal *et al.* defined a new concept of intra-URL relatedness which was used to quantify the relations between different words in the domain part and the remaining part of a URL [15], [16].

B. URL Analysis with Deep Learning

Designing the above hand-crafted features of URLs is time consuming and labor intensive, although some of the features can be extracted by statistical analysis offline. Recently, several DL classifiers have been proposed to analyse URLs [17] [7], [18]. They took character sequences of URLs as input, and performed the detection process by learning features automatically through deep neural networks. However, they ignored the meaningful and readable words inside URLs. Other authors [18] applied Convolutional Neural Networks to both characters and words of the URL String to learn the URL embedding in a jointly optimized framework. But, they did not consider the contextual information inside the sequences. Our work is based on a word-level DL classifier, and make full use of contextual information inside the word sequences of URLs.

III. Model Design

A. URL Segmentation

In order to get the word sequences of URLs, we use a word segmentation algorithm described by Norvig [19]. In our work, "word" is used in an informal sense, which includes the words in natural language, symbols contained in URLs, and 'uncuttable' segmentations. Using a sample from Google's web corpus[6] which includes words and their counts, Norvig defines a probabilistic model for consecutive words. Specifically, Norvig describes a dynamic programming algorithm to compute the most likely segmentation of a string of characters into a set of one or more tokens.

In this paper, we first treat symbols contained in URLs as known token boundaries. Then, we segment the resulting substrings using Norvig's algorithm to extract additional tokens that are not symbols-separated.

For URL `http://shen.mansell.tripod.com/games/gameboy.html`, we get the word sequence ['`http`', '`://`', '`shen`', '`.`', '`man`', '`sell`', '`.`', '`tripod`', '`.`' '`com`', '`/`', '`games`', '`/`', '`game`', '`boy`', '`.`', '`html`'].

B. Detection Model

After getting the word sequences of URLs, we can automatically learn high-abstract and poor-interpretable features among words, and classify each URL's status (spoofed or normal). The architecture of our model is shown in Figure 1.

Our model accepts a sequence of encoded words as input. The encoding is done by prescribing a vocabulary of size m for input word sequences. And, each word is quantized with its index in the vocabulary using "one-hot" encoding. Then, the input word sequences are transformed to vectors with fixed length n. The detection model consists of six layers: one embedding layer, one BiLSTM layers, three fully connected layers and one output layer.

The embedding layer is generally the first layer of most complex neural networks [20]. It is responsible to transform words into the first internal distributed representations, and

[6]http://norvig.com/ngrams/count_1w.txt

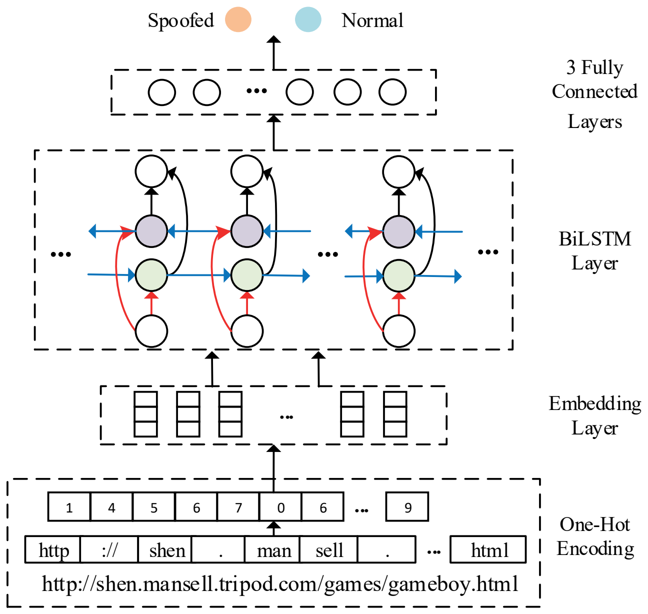

Fig. 1: Architecture of Our Detection Model

is shaped by the entire overall learning process. In this way, we can capture syntactic and semantic information about the words. Given an embedding matrix $W \in R^{d \times m}$, where m is the size of word vocabulary. For the given sequence $\{x_1, x_2, \ldots, x_n\}$, each word x_i has its embedding by using the matrix-vector product: $W \times v^{x_i}$, where v^{x_i} is one-hot representation, to get the corresponding column of the matrix W. The size of the word embedding d is a tunable hyperparameter that represents an embedding.

The BiLSTM layer can be regarded as implicit feature extraction, as opposed to explicit feature extraction used in traditional feature-based approaches. BiLSTM networks contain one node for each of the words in the input layer. Thus the network has n input nodes. BiLSTM networks are able to access context in both directions along the input sequence [21]. Each node in the input layer is connected with two separate hidden layers, one of which processes the input sequence of words forward, while the other processes it backward. Both hidden layers are connected to the same output layer, providing it with access to the past and future context of every word in the sequence. BiLSTM networks calculate the forward hidden sequence h^f, the backward hidden sequence h^b and the output sequence y by iterating over the following equations:

$$h_n^f = \sigma(W_x^f x_n + W_{h^f} h_{n-1} + b_{h^f}) \tag{1}$$

$$h_n^b = \sigma(W_x^b x_n + W_{h^b} h_{n-1} + b_{h^b}) \tag{2}$$

$$y_n = W_h^f h_n^f + W_h^b h_n^f + b_y \tag{3}$$

The output activations of the nodes in the output layer are then normalized to sum up to 1. Hence they can be treated as a vector indicating the probability for each label to occur at a particular position.

We apply a dropout function in the BiLSTM layer to prevent overfitting [22]. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. When testing, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. It significantly reduces overfitting and gives major improvements over other regularization methods.

In order to make full use of the rich features obtained from BiLSTM layer, we use three fully connected layers to map the distributed feature representation to the sample's marker space. We use ReLU as the activation function here in our work. In these three fully connected layers, we can also apply dropout functions to prevent overfitting.

The output layer is to convert the learned vectors into probabilities for classification. We use a sigmoid function as the activation function to optimize the model. Sigmoid functions refer to a special case of the logistic function which is defined by the formula $S(x) = \frac{1}{1+e^{-x}}$.

A sigmoid function is a mathematical function having a characteristic 'S'-shaped curve or sigmoid curve. Sigmoid functions have domain of all real numbers, and return value monotonically increasing most often from 0 to 1 or alternatively from -1 to 1, depending on convention. Note that spoofed URL detection is just exactly a binary classification problem (spoofed or normal).

IV. Dataset Construction

In order to construct a dataset of strong similar samples, we must ensure that spoofed samples have as many common words with normal samples as possible. The usual construction method is to generate spoofed samples which are similar with the given normal ones. However, this process has two limitations. First, normal URLs are massive, and how to select appropriate feeds is hard. Second, it costs too much to get enough generated URLs that are real spoofed URLs used by cyber crooks although we can obtain a large number of generated URLs. As a results, we can not verify the realness of the generated spoofed samples. So, we use an opposite method where we try to find normal samples which are similar with the known spoofed ones.

The process of our dataset construction method is shown in Figure 2. The key idea is to find legitimate URLs that are similar with spoofed URLs as normal URLs, but not choose normal URLs randomly as traditional methods. In this paper, we carry out the constraint that normal URLs should have a similar frequency words distribution with spoofed URLs. That means each normal URL should have at least one common word (or one common combination of several words) with at least one spoofed URL in the data sets. Given spoofed URLs, our method tries to find similar legitimate URLs based on the open Internet services provided by Google and VirusTotal. Specifically, our method consists of five steps: URL Segmentation, Matrix Construction, Word Selection, Google Hacking, and VirusTotal Verification.

1) URL Segmentation: The input spoofed URLs are choosen from a public blacklist available on PhishTank. PhishTank is a community-based phish verification system where users can submit, verify, and share phishing URLs. We consider phishing URLs which contain popular brand names (*e.g.*, google, paypal) as spoofed URLs created by cyber crooks. We extract the popular brand names through the Alexa's top 500 global sites. In this step, we use the URL segmentation algorithm described in Section III-A to divide spoofed URLs into word sequences.

2) Matrix Construction: This step is to construct a URL-word matrix from word sequences (without symbols) of spoofed URLs. Each row in the matrix represents a spoofed URL, and each column represents a word in word sequences. Each value in the matrix is the times of the corresponding word appearing in the word sequence of the corresponding spoofed URL.

3) Word Selection: This step tries to select representative words based on the URL-word matrix, which are used to collect normal URLs in the following step. Representative words should be able to represent the distribution of words in spoofed URLs. How to select representative words is yet an open problem to be addressed. On one hand, we want as few representative words as possible, as more words mean much more overheads. On the other hand, we want these selected words can cover as many spoofed samples as possible. Because more spoofed URLs are covered means the selected words are more representative and the distribution of collected URLs will be more even. We reduce the problem of selecting representative words to a problem of frequent itemset mining [23]. Traditional frequent itemset mining is used to mine the relationships among items in a large database [24]. In this paper, we use it to find the frequent words (including a single word and a combination of several words appear synchronously). Thus, the open problem is NP-hard. In this paper, we use the dynamic programming strategy to obtain the optimal result.

4) Google Hacking: This steps use a computer hacking technique that involves using advanced operators in the Google search engine to locate specific strings of text within search results [25]. Search operator "allinurl:" is to instruct Google to search for pages that contain all the words in the input text in the URL. And, the input texts come from the representative words. For example, inputting "allinurl:paypal update" in Google searching box will get pages with words "paypal" and "update" in the URL. In this paper, we collect the URLs of the first 100 pages for each combination of representative words. According to the main idea of PageRank algorithm that is the cornerstone of Google search, the first 100 URLs are very likely to be legitimate.

5) VirusTotal Verification: We use VirusTotal to verify whether the collected URLs are normal or not, in order to avoid potential and negligible errors. VirusTotal aggregates at least 65 URL scanners from BitDefender, ESET and Kaspersky *etc.* to check if the inputting URL is clean. We think a clean URL is normal and add it into our data set if the URL is classified as "clean site" by all URL scanners.

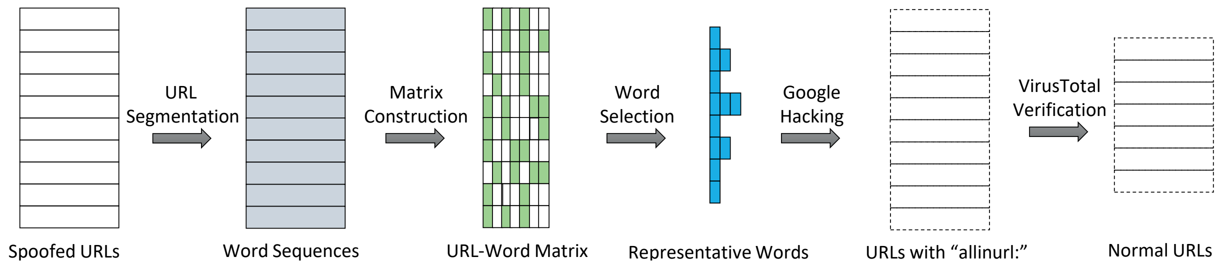

Fig. 2: Overview of our proposed spoofed dataset construction process.

V. Experimental Results

This section presents our experimental results. We first give a brief description of our experimental setup, including experimental data sets and the baseline approaches used to compare with our work. Then we evaluate the performance of NeuralAS, and make a comparison with the baseline approaches using three different metrics. Finally, we give some further discussions.

A. Experimental Datasets

In order to evaluate the effectiveness and robustness of NeuralAS, we use two different experimental datasets. One is the same as most existing approaches used, spoofed and normal URLs all come from public datasets, and we call it 'TraditionalSet'. Another is constructed with our proposed construction method described in Section IV, and we call it 'ConstrutedSet'. The overview of the two experimental datasets are shown in Table II and Table III.

We extract 120,166 spoofed URLs from public blacklist available on PhishTank. As we describe in Section IV, we consider the phishing URLs which contain popular brand names as spoofed URLs. We use the same spoofed URLs in the both datasets. The difference between the two datasets is the construction of normal URLs.

In 'TraditionalSet', we randomly choose 300,000 URLs from DMOZ as normal URLs. In 'ConstrutedSet', we first segment 120,166 spoofed URLs into word sequences. And, there are 41,462 words appearing in the resulting word sequences, except the words appears in the scheme component and the TLD part of the host component, using the URL segmentation algorithm. Then, we obtain 2,316 representative words. For each representative word, we collect the URLs of the first 100 searching pages using Google Hacking with operator "allinurl:". In our experiments, we only collect 228,322 URLs, which is less than the expected number of collected URLs (231,600 = 2,316$\times$100), owing to the crawling restrictions and the removing of duplicate URLs. We use VirusTotal to verify the collected URLs and finally only 227,783 of them are labelled as "clean site". These 227,783 "clean site" are used as normal URLs in 'ConstrutedSet'.

To train and test the detection models, we randomly separate the two datasets into two parts respectively (90% for training, 10% for testing).

TABLE II: Overview of 'TraditionalSet'

	Size	**Source**
Spoofed URLs	120,166	PhishTank
Normal URLs	300,000	DMOZ
Total URLs	420,166 (120,166+300,000)	

TABLE III: Overview of 'ConstrutedSet'

	Size	**Source**
Spoofed URLs	120,166	PhishTank
Words in Spoofed	41,462	URL Segmentation
Representative Sets	2,051	Words Selection
Collected URLs	228,322	Google Hacking
Normal URLs	227,783	VirusTotal Verification
Total URLs	347,949 (120,166+227,783)	

B. Evaluation Metrics

We evaluate the effectiveness and robustness of our work on detecting spoofed URLs with three metrics: precision rate, recall rate, and F_1 measure. When the classifier correctly predicts a spoofed URL, it is a true positive (TP). When the classifier correctly predicts a normal URL, it is a true negative (TN). When the classifier predicts a normal URL as a spoofed URL, it is a false positive (FP). When the classifier predicts a spoofed URL as a normal URL, it is a false negative (FN). The precision rate is defined as the proportion of predicted spoofed URLs that are really spoofed, which is $precision = \frac{TP}{TP+FP}$. The recall rate is defined as the proportion real spoofed URLs that are correctly predicted, which is $recall = \frac{TP}{TP+FN}$. The F_1 measure is the harmonic mean of the precision rate and the recall rate, and defined as $F_1 = \frac{2*precision*recall}{precision+recall}$.

C. Baseline Approaches

In this paper, we design four different detection approaches as the baselines to compare with NeuralAS. Two of them are machine learning based approaches, and the other two are variants of our work using different neural networks. Specifically speaking, we use decision tree and random forest to classify an unlabelled URL. The feature vector of each URL in the two machine learning approaches is obtained with bag-of-word model, where each vector item corresponds to a word, and the value represents the number of the word occurring in the URL. The two variants are word-level Convolutional Neural Network (word-level CNN) and character-level BiLSTM (char-level BiLSTM), which are the combinations of word sequences and CNN, and the combination of character

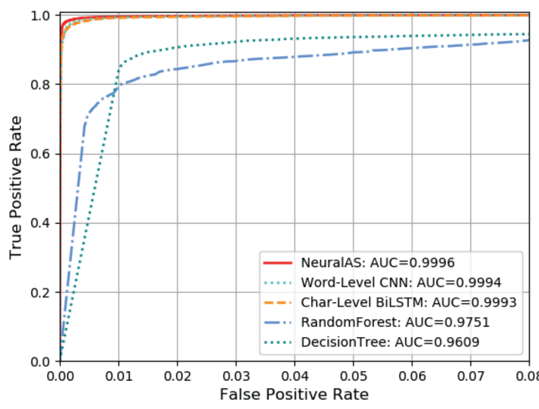

Fig. 3: ROC curves for NeuralAS and other detection methods on 'TraditionalSet'.

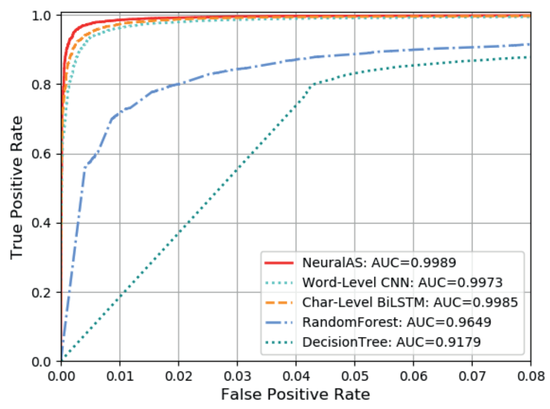

Fig. 4: ROC curves for NeuralAS and other detection methods on 'ConstrutedSet'.

sequences and BiLSTM. The processing details are the same with our work in this paper. We implement all the approaches on the both datasets.

D. *Effectiveness of Our Work*

To evaluate the effectiveness of our work, we compare NeuralAS with traditional feature-based methods (*i.e.*, decision tree and random forest) and other deep learning models (*i.e.*, word-level CNN and char-level BiLSTM) on the two datasets shown in Table II and Table III. The ROC curves for all the detection models on the two datasets are shown in Figure 3 and Figure 4 respectively. We calculate the AUC value for each curve, which is equivalent to the probability that a randomly chosen positive example is ranked higher than a randomly chosen negative example [26].

From the results, we can observe that NeuralAS provides the best performance with an AUC of 0.9996 on 'TraditionalSet' and an AUC of 0.9989 on 'ConstrutedSet'. And, the performance of NeuralAS is relatively stable. The feature-based methods (*i.e.*, ML classifiers) perform well on 'TraditionalSet', but shift down significantly on 'ConstrutedSet'. The reason is that the traditional data sets are obviously distinguishable, in which the spoofed URLs are very different with the normal ones. While in our constructed datasets, spoofed URLs are much more similar with normal URLs. It also proves that feature-based detection methods are not able to guard against attackers' different strategies in reality. While, DL classifiers can adjust to the changes of attack strategies very well.

E. *Robustness of Our Work*

Note that, performance of supervised learning models usually depends on training data. We have proved that NeuralAS performs well even if on strong similar samples by deeply mine the contextual information inside the word sequences of URLs. In order to evaluate the robustness of NeuralAS, we conduct another experiment on a smaller data set. We randomly select 30% of the training data (as 'SubSet'), and compare the results with all the training data (as 'FullSet') in 'ConstrutedSet'.

The results of the comparison of 'FullSet' and 'SubSet' are given in Table IV. We can find that NeuralAS achieves the highest Precision, Recall, and F_1 measure even on the subset, and experiences the least decrease comparing with the values on the full set, as the results in bold in Table IV. In contrast, character-level classifiers experience the most decrease among DL classifiers. The reason is that for the same URL, character sequences are much longer than word sequences. Then it is much harder for DL classifiers to memory the contextual information during the training process. When the training data decreases, the detection accuracy will experience a decrease penalty.

F. *Further Discussions*

Our NeuralAS approach aims at detecting spoofed URLs, which is a critical component for discovering spoofing attacks such as phishing and ransomware in networks. Many existing URL analysis approaches have been proposed to provide a clue to phishing attacks. Actually, phishing is a complex procedure. And, cyber crooks can also achieve URLs obfuscation with IP address and URL shortening.

However, relying on IP address rather than domain names deprives attckers from the flexibility brought by the DNS to change the hosting location of their deceptive content while keeping the same link. Moreover, IP blacklisting is widely used to prevent access to malicious hosting infrastructure, so attackers would have to face other issues [27].

As for shortened URLs, they are still redirected to a real and long URL. We can get the pending URLs with web browser monitors (*e.g.*, Selenium), and put them into our detection model[7].

VI. Conclusion and Future Work

In this paper, we propose a novel approach named NeuralAS (Neural Anti-Spoofing) which is the first work using

[7]http://www.seleniumhq.org

TABLE IV: Overview of Experimental Results of Performance Decline for each model with Data Sets Decrease.

	Precision			Recall			F_1 measure		
	FullSet	**SubSet**	**Decline**	**FullSet**	**SubSet**	**Decline**	**FullSet**	**SubSet**	**Decline**
DecisionTree	0.8840	0.8569	0.0271	0.8500	0.8266	0.0234	0.8667	0.8415	0.0252
RandomForest	0.9099	0.8897	0.0202	0.8797	0.8575	0.0222	0.8945	0.8734	0.0211
Char-Level BiLSTM	0.9732	0.9396	0.0336	0.9813	0.9729	0.0084	0.9772	0.9559	0.0213
Word-Level CNN	0.9763	0.9637	0.0126	0.9684	0.9534	0.0150	0.9723	0.9585	0.0138
NeuralAS	**0.9840**	**0.9737**	**0.0103**	**0.9833**	**0.9765**	**0.0068**	**0.9836**	**0.9751**	**0.0085**

contextual information inside word sequences of URLs to detect spoofed URLs. The main idea is to segment URLs into word sequences, and utilize bidirectional Recurrent Neural Networks (RNN) to perform the detection with high-abstract and poor-interpretable features learned automatically. First, we use a URL segmentation algorithm to divide URLs into word sequences. Second, we input the word sequences into deep learning classifier to detect spoofed URLs. To evaluate the performance of NeuralAS, we also propose a novel method to construct indistinguishable spoofed data sets of strong similar samples, which is in consistence with the reality. An extensive set of comparisons is offered with traditional feature-based models and other neural network models in the experiments. The results show that our proposed method works well on spoofed URLs detection and has a significantly better performance over other detection models.

In the future work, we would like to investigate the adversarial strategies for avoiding the detection with DL classifiers. As for ML classifiers, attackers can avoid the detection by learning the details of features used in classifiers. While, for DL classifiers, attacker maybe avoid the detection in a similar way by learning the mechanism of neural network architectures.

ACKNOWLEDGEMENT

This work was supported by the Open Project Foundation of Information Security Evaluation Center of Civil Aviation, Civil Aviation University of China (NO. CAAC-ISECCA-201801).

REFERENCES

[1] L. Anh, M. Athina, and M. Faloutsos, "PhishDef: URL Names Say It All," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2011, pp. 191–195.
[2] L. Min-Sheng, C. Chien-Yi, L. Yuh-Jye, and P. Hsing-Kuo, "Malicious URL Filtering - A Big Data Application," in *IEEE Big Data*, 2013, pp. 589–596.
[3] W. Wei and S. Kenneth E, "Breaking Bad: Detecting Malicious Domains using Word Segmentation," in *IEEE Web 2.0 Security and Privacy Workshop*, 2015.
[4] V. Rakesh and D. Keith, "On the Character of Phishing URLs: Accurate and Robust Statistical Learning Classifiers," in *ACM Conference on Data and Application Security and Privacy*, 2015, pp. 111–121.
[5] V. Rakesh and D. Avisha, "What's in a URL: Fast Feature Extraction and Malicious URL Detection," in *Proc of International Workshop on Security and Privacy Analytics (IWSPA)*, 2017, pp. 55–63.
[6] J. Saxe and K. Berlin, "eXpose: A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys," *arXiv:1702.08568*, 2017.
[7] A. C. Bahnsen, E. C. Bohorquez, S. Villegas, J. Vargas, and F. A. Gonzalez, "Classifying Phishing URLs Using Recurrent Neural Networks," in *2017 APWG Symposium on Electronic Crime Research (eCrime)*, 2017, pp. 1–8.
[8] D. Sahoo, L. Chenghao, and C. H. Steven, "Malicious URL Detection using Machine Learning: A survey," *arXiv:1701.07179*, 2017.
[9] M. D. Kevin and G. Minaxi, "Behind Phishing: An Examination of Phisher Modi Operandi," in *Usenix Workshop on Large-Scale Exploits and Emergent Threats*, 2008, pp. 41–48.
[10] B. Leyla, K. Engin, K. Christopher, and B. Marco, "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis," in *NDSS*, 2011.
[11] C. Weibo, Z. Bin, B, X. Feng, G. Xiaohong, and C. Zhongmin, "Protect Sensitive Sites from Phishing Attacks Using Features Extractable from Inaccessible Phishing URLs," in *IEEE ICC*, 2013, pp. 1990–1994.
[12] H. Yuanchen, Z. Zhenyu, K. Sven, and T. Yuchun, "Mining DNS for Malicious Domain Registrations," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2010, pp. 1–6.
[13] M. Justin, S. Lawrence, K, S. Stefan, and V. Geoffrey, M, "Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs," in *Proc. ACM SIGKDD*, 2009, pp. 1245–1254.
[14] B. Aaron, W. Brad, S. Thamar, and W. Gary, "Lexical Feature Based Phishing URL Detection Using Online Learning," in *ACM Workshop on Artificial Intelligence and Security*, 2010, pp. 54–60.
[15] S. Marchal, J. Francois, R. State, and T. Engel, "PhishScore: Hacking Phishers' Minds," in *International Conference on Network and Service Management (CNSM)*, 2014, pp. 46–54.
[16] ——, "PhishStorm: Detecting Phishing With Streaming Analytics," *IEEE Transactions on Network and Service Management*, vol. 11, no. 4, pp. 458–471, 2014.
[17] S. Joshua and B. Konstantin, "eXpose: A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys," *arXiv:1702.08568*, 2017.
[18] L. Hung, P. Quang, S. Doyen, and C. H. Steven, "URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection," in *In Proceedings of ACM Conference, Washington, DC, USA*, 2017, pp. 1–13.
[19] S. Toby and H. Jeff, "Beautiful Data: The Stories Behind Elegant Data Solutions," http://books.google.com/books?id=zxNglqU1FKgC, 2009.
[20] F. Lorenzo and F. M. Zanzotto, "Symbolic, Distributed and Distributional Representations for Natural language Processing in the Era of Deep Learning, a Survey," *arXiv preprint arXiv:1702.00764*, 2017.
[21] G. Alex, F. Santiago, and S. Jurgen, "Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition," in *International Conference on Artificial Neural Networks*, 2005, pp. 799–804.
[22] S. Nitish, H. Geoffrey, K. Alex, S. Ilya, and S. Ruslan, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[23] C. Borgelt, "Frequent item set mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 6, pp. 437–456, 2012.
[24] Y. Junrui, Z. Yingjie, and W. Yanjun, "An Improved Vertical Algorithm for Frequent Itemset Mining from Uncertain Database," in *Intelligent Human-Machine Systems and Cybernetics*, 2017, pp. 355–358.
[25] N. Blachman, "Google guide ," http://exactas.udea.edu.co/~carlopez/google/Quick_Guide_Google_Search_Operators.pdf, 2005.
[26] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters, Special issue: ROC analysis in pattern*, vol. 27, no. 8, pp. 861–874, 2006.
[27] M. Samuel, S. Kalle, S. Nidhi, and A. N, "Know Your Phish: Novel Techniques for Detecting Phishing Sites and their Targets," in *Proc. IEEE ICDCS*, 2016, pp. 323–333.