HTMLPhish: Enabling Phishing Web Page Detection by Applying Deep Learning Techniques on HTML Analysis

Chidimma Opara*, Bo Wei†, Yingke Chen*
*Teesside University, Middlesbrough, UK
c.opara@tees.ac.uk, y.chen@tees.ac.uk
†Northumbria University, Newcastle upon Tyne, UK
bo.wei@northumbria.ac.uk

Abstract*—Recently, the development and implementation of phishing attacks require little technical skills and costs. This uprising has led to an ever-growing number of phishing attacks on the World Wide Web. Consequently, proactive techniques to fight phishing attacks have become extremely necessary. In this paper, we propose HTMLPhish, a deep learning based data-driven end-to-end automatic phishing web page classification approach. Specifically, HTMLPhish receives the content of the HTML document of a web page and employs Convolutional Neural Networks (CNNs) to learn the semantic dependencies in the textual contents of the HTML. The CNNs learn appropriate feature representations from the HTML document embeddings without extensive manual feature engineering. Furthermore, our proposed approach of the concatenation of the word and character embeddings allows our model to manage new features and ensure easy extrapolation to test data. We conduct comprehensive experiments on a dataset of more than 50,000 HTML documents that provides a distribution of phishing to benign web pages obtainable in the real-world that yields over 93% Accuracy and True Positive Rate. Also, HTMLPhish is a completely language-independent and client-side strategy which can, therefore, conduct web page phishing detection regardless of the textual language.

***Keywords*-Phishing detection, Web pages, Classification model, Convolutional Neural Networks, HTML**

I. INTRODUCTION

The infamous phishing attack is a social engineering technique that manipulates internet users into revealing private information that may be exploited for fraudulent purposes [1]. This form of cybercrime has recently become common because it is carried out with little technical ability and significant cost [2]. The proliferation of phishing attacks is evident in the 46% increase in the number of phishing websites identified between October 2018 and March 2019 by the Anti-Phishing Working Group (APWG) [3]. Most phishing attacks are started by an unsuspecting Internet user merely clicking on a link in a phishing email message that leads to a bogus website. The impact of phishing attacks on individuals such as identity theft, psychological, and financial costs can be devastating.

A. Problem Definition

Recent research in phishing detection approaches has resulted in the rise of multiple technical methods such as augmenting password logins [4], and multi-factor authentication [5]. However, these techniques are usually server-side systems that require the Internet user to correspond with a remote service, which adds further delay in the communication channel. Another popular phishing detection system that relies on a centralised architecture is the phishing blacklist and whitelist methods [6]. A URL visited by an internet user will be compared with the URL in these lists in real-time. Although the list based methods tend to keep the false positive rate low, however, a significant shortcoming is that the lists are not exhaustive, and they fail to detect zero-day phishing attacks. To mitigate these limitations, researchers have developed several anti-phishing techniques using machine learning models as they are mostly client-side based and can generalise their predictions on unseen data.

Machine learning-based anti-phishing techniques typically follow specific approaches: (1) The required representation of features is firstly extracted, then (2) a phishing detection machine learning model is trained using the feature vectors. To extract the feature representation from the lexical and static components of a web page, the machine learning models rely on the assumption that the infrastructure of phishing pages are different from legitimate pages. For example, in [7], phishing web pages are automatically detected based on handcrafted features extracted from the URL, HTML content, network, and JavaScript of a web page. Furthermore, natural language processing techniques are currently used to extract specific features such as the number of common phishing words, type of ngram, etc. from the components of a web page [8], [9], [10].

While the above approaches have proven successful, they nevertheless are prone to several limitations, particularly in the context of HTML analysis: *i. inability to accommodate unseen features:* As the accuracy of existing models depends on how comprehensive the feature set is and how impervious the feature set remains to future attacks, they will be unable to correctly detect new phishing web pages with evolved content and structure without a regular update of the feature set. *ii. They require substantial manual feature engineering:* Existing phishing detection machine learning models require specialised domain knowledge in order to ascertain the needed

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

features suitable to each task (e.g., number of white spaces in the HTML content, number of redirects, and iframes, etc.). This is a tedious process, and these handcrafted features are often targeted and bypassed in future attacks. It is also challenging to know the best features for one particular application.

To address the above issues, we propose HTMLPhish, a deep learning based data-driven end-to-end automatic phishing web page classification approach. Specifically, HTMLPhish uses both the character and word embedding techniques to represent the features of each HTML document. Then Convolutional Neural Networks (CNNs) are employed to model the semantic dependencies.

The following characteristics highlight the relevance of HTMLPhish to web page phishing detection:

(1) HTMLPhish analyses HTML directly to help reserve useful information. It also removes the arduous task required for the manual feature engineering process.

(2) HTMLPhish takes into consideration all the elements of an HTML document, such as text, hyperlinks, images, tables, and lists, when training the deep neural network model.

We experimentally demonstrate the significance of character and word embedding features of HTML contents in detecting phishing web pages. We then propose a state-of-the-art HTML phishing detection model, in which the character and word embedding matrices are concatenated before employing convolutions on the represented features. Our proposed approach ensures an adequate embedding of new feature vectors that enables straightforward extrapolation of the trained model to test data. Subsequently, we conduct extensive evaluations on a dataset of over 50,000 HTML documents collected over two months. This ensures our evaluation settings reproduces real-world situations in which models are applied to data generated up to the present point and applied to new data.

We summarise the main contributions of this paper as follows:

- Different from existing methods, our proposed model, HTMLPhish, to the best of our knowledge, is the first to use only the raw content of the HTML document of a web page to train a deep neural network model for phishing detection. Manual feature engineering is reduced as HTMLPhish learns the representation in the features of the HTML document, and we do not depend on any other complicated or specialist features for the task. Our proposed approach takes advantage of the word and character embedding matrix to present a phishing detection model that automatically accommodates new features and is therefore easily applied to test data.
- We conduct extensive evaluations on a dataset of more than 50, 000 HTML documents collected in two months. The distribution of the instances in our dataset is similar to the ratio of phishing and legitimate web pages found in the real-world. This ensures that our evaluation metrics and results are relevant to existing systems.
- Furthermore, we carried out a longitudinal study on the efficiency HTMLPhish to infer the maximum retraining period, for which the accuracy of the system does not reduce. Our result only recorded a minimal 4% decrease in accuracy on the test data. This confirms that HTMLPhish remains reliable and temporally robust over a long period.

We organised the remainder of the paper as follows: the next section provides an overview of related works on proposed techniques of detecting phishing on web pages. Section III gives the prior knowledge on Convolutional Neural Networks, and Section IV provides an in-depth description of our proposed model. Section V elaborates on the dataset collection, while the detailed results on the evaluations of our proposed model are found in Section VI. Finally, we conclude our paper in Section VII.

II. Related Works

In this section, we address two most closely related topics to our work: the phishing web page detection using feature engineering and the Deep Learning method (especially for NLP).

A. Feature Engineering for Phishing Web Page Detection

These techniques extract specific features from a web page such as JavaScript, HTML web page, URL, and network features. These are fed into machine learning algorithms to build a classification model. These machine learning techniques differ in the type of heuristics and number of feature sets used and the optimisation algorithm applied to the machine learning algorithm. These techniques are based on the fact that both the phishing and benign web pages have a different content distribution of extracted features. The accuracy of heuristics and machine learning-based techniques critically depends on the type of features extracted, and the machine learning algorithm applied. Many phishing detection techniques have been built on different proposed feature sets.

Varshney et al [11] proposed LPD, a client-side based web page phishing detection mechanism. The strings from the URL and page title from a specified web page is extracted and searched on the Google search engine. If there is a match between the domain names of the top T search results and the domain name of the specified URL, the web page is considered to be legitimate. The result from their evaluations gave a true positive rate of 99.5%.

Smadi et al. [12] proposed a neural network model that can adapt to the dynamic nature of phishing emails using reinforcement learning. The proposed model can handle zero-day phishing attacks and also mitigate the problem of a limited dataset using an updated offline database. Their experiment yielded a high accuracy of 98.63% on fifty features extracted from a dataset of 12,266 emails.

The selection of features from various web page elements can be an expensive process from security risk and technological workload angle. For example, it can be prolonged and somewhat problematic to extract specific feature sets. Besides, it needs specialist domain expertise to define which features are essential.

B. Deep Learning

Due to its performance in many applications, Deep Learning has attracted increased interest in recent years [13], [14], [15]. The core concept is to learn the feature representation from unprocessed data instantaneously without any manual feature engineering. Under this premise, we want to use Deep Learning to detect phishing HTML content by directly learning how features from the raw HTML string is represented instead of using specialist features that are manually engineered.

As we want to train our Deep Learning networks using textual features, it is, therefore, essential to discuss NLP as it relates to Deep Learning. Deep learning techniques have been successful in a lot of NLP tasks, for example, in document classification [16], machine translation [17], etc. Recurrent neural networks (e.g., LSTM [18]) have been extensively applied due to their ability to exhibit temporal behaviour and capture sequential data. However, CNN has become brilliant substitutes for LSTMs, especially showing excellent performance in text classification and sentiment analysis as CNN learns to recognize patterns across space [19].

Very few attempts have been made to use Deep Learning to detect phishing web pages using web page components. Bahnsen et al. [20] proposed a phishing classifying scheme that used features of the URLs of a web page as input and implemented the model on an LSTM network. The results yielded gave an accuracy of 98.7% accuracy on a corpus of 2 million phishing and legitimate URLs. The authors of [21] proposed a CNN based model which combines the outputs of two Convolutional layers to detect malicious URLs.

However, our review did not find any existing approach that detects malicious phishing web pages using only HTML documents on Deep Learning. HTMLPhish learns the semantic information present only in the character and words in an HTML document to determine the maliciousness of the web page. Our thorough analysis shows that phishing web pages can be detected using only their HTML document content.

III. Preliminaries

We define the problem of detecting phishing web pages using their HTML content as a binary classification task for prediction of two classes: *legitimate* or *phishing*. Given a dataset with *T* HTML documents $\{(html_1, y_1), ..., (html_T, y_T)\}$, where $html_t$ for *t* = 1, . . . , *T* represents an HTML document , while $y_t \in \{0, 1\}$ is its label. $y_t = 1$ corresponds to a phishing HTML document while $y_t = 0$ is a legitimate HTML document.

A. Deep Neural Network for Phishing HTML Document Detection

The deep neural network that underlies HTMLPhish is a Convolutional Neural Network (CNN). To detail a basic CNN for HTML document classification, an HTML document is comprised of a string of characters or words. Our goal is to obtain an embedding matrix *html* $\rightarrow$s $\epsilon\mathbb{R}^{maxlen\times d}$, in a way that *s* is made up of sets of adjoining inputs $s_i \in (1, 2, ..., maxlen)$ in a string, in which the input can be individual characters or words from the HTML document. Each input is subsequently transformed in an embedding $s_i\epsilon\mathbb{R}^d$ is the i^{th} column of S and the *d-dimension* is the vector size which is automatically initialized and learnt together with the remainder of the model.

In this paper, the embedding matrix was automatically initialised, and for parallelisation, all sequences were padded to the same length *maxlen*.

The CNN performs a convolution operation $\otimes$ over $s\epsilon\mathbb{R}^{maxlen\times d}$ using:

$$c_i = f(M \otimes s_{i:i+n-1} + b_i)$$

followed by a non-linear activation where b_i is the bias, *M* is the convolving filter and *n* is the kernel size of the convolution operation. After the convolution, a pooling step is applied (which in our model is the Max Pooling) in order to decrease the feature dimension and determine the most important features.

The CNN is capable of exploiting the temporal relation of *n* kernel size in its input using the filter *M* to convolve on each segment of *n* kernel size. A CNN model typically contains several sets of filters with different kernel sizes *(n)*. Those are the model hyperparameters that are set by the user. In this deep neural network, the convolution layer is usually followed by a Pooling layer. The features from the Pooling layer are then passed to dense layers to perform the required classification. The entire network is then trained by using backpropagation.

Note: In order to differentiate our state-of-the-art model from the baseline models, for the rest of this paper, we will use the term HTMLPhish-Full to indicate HTMLPhish trained with the proposed model unless otherwise stated, while HTMLPhish-Character and HTMLPhish-Word represent the deep neural network model using only the character and word embedding respectively.

IV. The Proposed Model

In this section, we elaborate on the architecture of our proposed deep neural network model HTMLPhish-Full. The network architecture seen in Figure 3 shows HTMLPhish-Full has two input layers. The first input layer processes the raw HTML document into an embedding matrix made up of character-level feature representations, while the second input layer does the same with words. These two branches are concatenated in a dense layer called the Concatenation layer. Therefore, the embedding matrix in this model is the sum of the character-level embedding matrix and the word embedding matrix C_{em} + W_{em} where C_{em} $\rightarrow$c $\epsilon\mathbb{R}^{maxlen_1\times d}$, and W_{em} $\rightarrow$w $\epsilon\mathbb{R}^{maxlen_2\times d}$. The features in the Concatenation layer allows the preservation of the original information in the HTML content. In the concatenation layer, the content of both embedding layers are put alongside each other to yield a 3 dimensional layer [C_{em} + W_{em} $\rightarrow$(None, 180, 100) + (None, 2000, 100) = (None, 2180, 100)].

To generate the character-level embedding matrix C_{em}, the model learns an embedding, which takes the characteristics of the characters in an HTML document. To do so, all the distinct characters, including punctuation marks in the corpus, are listed. We obtained 167 unique characters. We set the length of the sequences $maxlen_1 = 180$ characters. Every HTML document with strings greater than 180 characters is cut from the 180th character, and any HTML document with characters smaller than 180 characters would be padded up to 180 with zeroes. Before each character in our work is embedded into a d-dimensional vector, we conduct a *tokenization* on the characters in the HTML document and segment the characters into *tokens* as shown in Figure 1. An index is associated with each token before being applied to a d-dimensional character embedding vector where d is set at 100, which is automatically initialised and learnt together with the remainder of the model. To facilitate its implementation, each HTML document *html* is transformed into a matrix, *html* $\rightarrow$c $\epsilon\mathbb{R}^{maxlen_1 \times d}$, where d = 100 and $maxlen_1$ = 180.

For the word embedding matrix W_{em}, firstly, the raw HTML document is processed into word-level representations by the word embedding layer. To achieve this, all the different words in the HTML document of the training corpus are listed using the following approach: An HTML document is split into individual words while treating all punctuation characters as separate tokens. For example, as shown in Figure 1, $<!DOCTYPE\ html>$, will be split into $['<'$, $'!'$, $'DOCTYPE'$, $'html']$. We surmise that punctuation marks provide important information benefits for phishing HTML document detection since punctuation marks are more prevalent and useful in the context of HTML documents than ordinary languages. HTML contains a sequence of markup tags that are used to frame the elements on a website. Tags contain keywords and punctuation marks that define the formatting and display of the content on the Web browser. The listed unique words are used to create a dictionary where every word becomes a feature. We obtained about 321,009 unique words in our dataset. We also padded the HTML documents to make the lengths of the HTML documents uniform in terms of number of words ($maxlen_2 = 2000$). Each unique word is then embedded into a d-dimensional vector, where d is set at 100, which is automatically initialised and learned together with the remainder of the model. All the HTML documents are converted to their respective matrix representation ($maxlen_2 \times d$), on which the CNN is applied where d = 100 and $maxlen_2$ = 2000. Figure 1 shows an overview of the character and word embedding layer.

We can now introduce Convolutionary layers using the HTML document matrix (for all the HTML documents $s_t \forall t = 1, ..., T$) as the corpus. We applied 32 Convolutionary filters $M \epsilon \mathbb{R}^{d \times n}$ where *n* 8. The Max-Pooling layer whose features are then passed to a 10 unit dense layer comes after the Convolutionary filters. The dense layer, which is regularised by dropout, finally connects to a Sigmoid layer. Then using the ADAM optimisation algorithm [22], we train the model through backpropagation.

A. Baseline Models

The baseline models, HTMLPhish-Character and HTMLPhish-Word, whose architectures are detailed in Figure 3, are CNN models trained either on character-level embeddings or word-level embeddings, respectively. The embedding matrices described above are applied to 32 Convolutionary filters $M \epsilon \mathbb{R}^{d \times n}$ where *n* 8. The next layer after the Convolutionary filters is the Max-Pooling layer, whose features are then passed to a 10 unit dense layer. The Dense layer, which also is regularised by dropout, finally connects to a Sigmoid layer. Also, the models are trained through backpropagation using the ADAM optimisation algorithm.

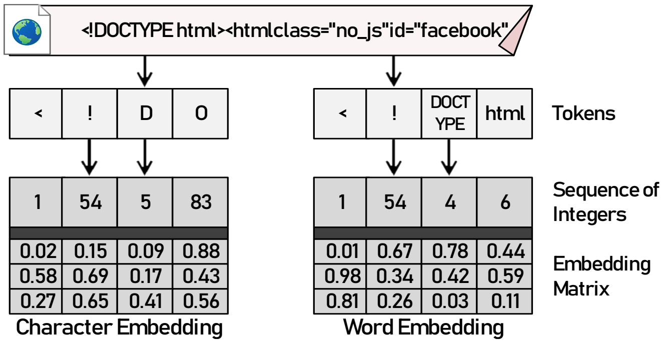

Fig. 1: Configuration of the Embedding Layer

TABLE I: HTML Documents Used in this Paper

Dataset	**D1**	**D2**
Date generated	11 - 18 Nov, 2018	10 -17 Jan, 2019
Legitimate Web Pages	23,000	24,000
Phishing Web pages	2,300	2,400
Total	25,300	26,400

V. Dataset

Data collection plays an essential role in phishing web page detection. In our approach, we collated HTML documents using a web crawler. We used the Beautiful Soup [23] library in Python to create a parser that dynamically extracted the HTML document from each final landing page. We chose to use Beautiful Soup for the following reasons:

(1) it has functional versatility and speed in parsing HTML contents, and

(2) Beautiful Soup does not correct errors when analysing the HTML Document Object Model (DOM). The HTML documents in our corpus include all the contents of an HTML document, such as text, hyperlinks, images, tables, lists, etc. Figure 2 shows an overview of the data collection stage.

A. Data Collection

Since phishing campaigns follow temporal trends in the composition of web pages, the earliest data obtained should always be used for training and the most recent data collected for testing [24]. Different phishing pages created during the same time may probably have the same infrastructure. This could exaggerate an over-trained classification model's predictive output. To ensure our evaluation settings reproduces real-world situations in which models are applied on data generated up to the present point and applied on new web pages, we

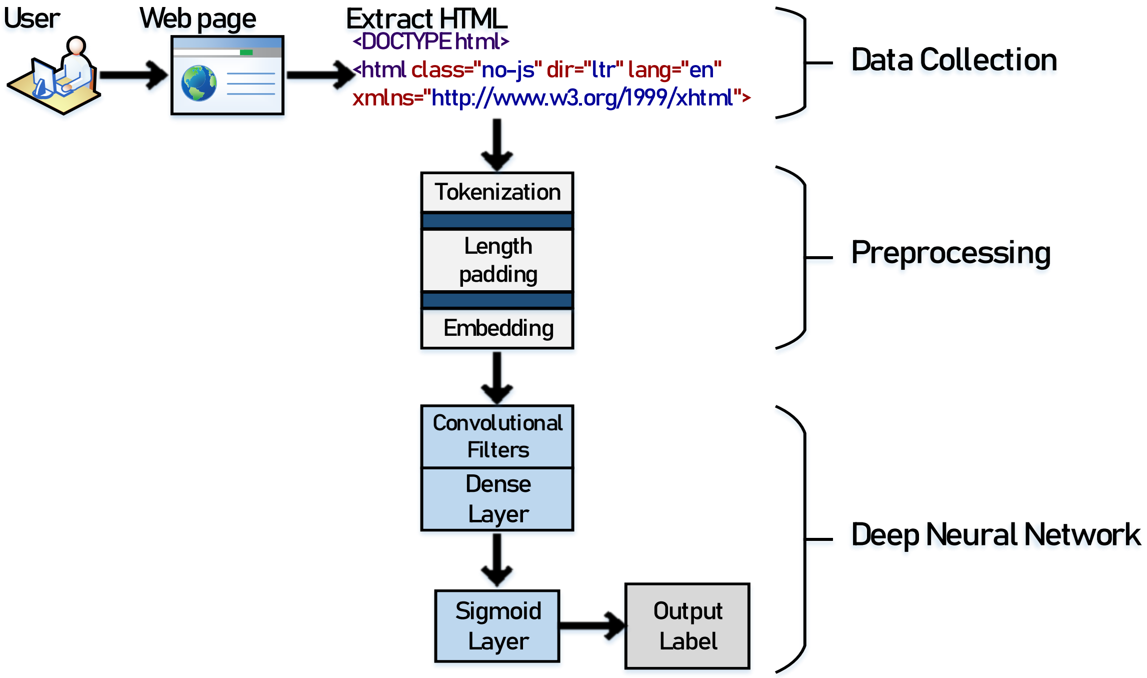

Fig. 2: A Schematic Overview of the Stages Involved in Our Proposed Model

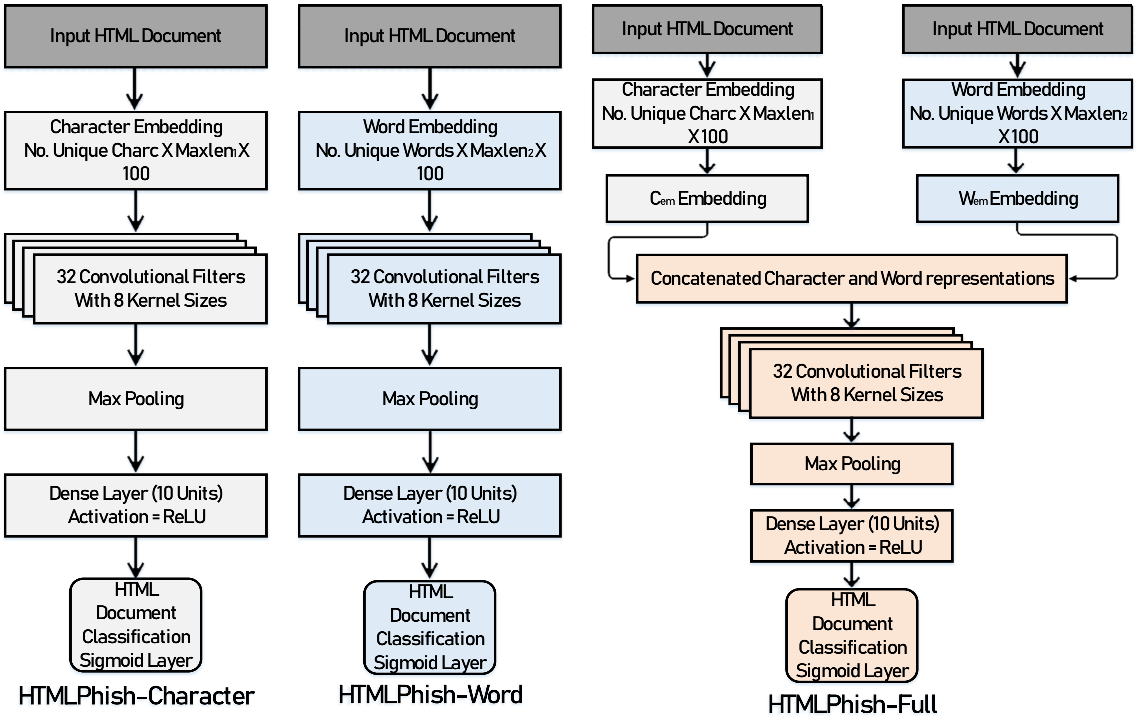
Input HTML Document
Character Embedding
No. Unique Charc X Maxlen_1 X 100
32 Convolutional Filters With 8 Kernel Sizes
Max Pooling
Dense Layer (10 Units) Activation = ReLU
HTML Document Classification Sigmoid Layer
HTMLPhish-Character
Input HTML Document
Word Embedding
No. Unique Words X Maxlen_2 X 100
32 Convolutional Filters With 8 Kernel Sizes
Max Pooling
Dense Layer (10 Units) Activation = ReLU
HTML Document Classification Sigmoid Layer
HTMLPhish-Word
Input HTML Document
Character Embedding
No. Unique Charc X Maxlen_1 X100
C_{em} Embedding
Input HTML Document
Word Embedding
No. Unique Words X Maxlen_2 X100
W_{em} Embedding
Concatenated Character and Word representations
32 Convolutional Filters With 8 Kernel Sizes
Max Pooling
Dense Layer (10 Units) Activation = ReLU
HTML Document Classification Sigmoid Layer
HTMLPhish-Full

Fig. 3: The Overall Architecture of HTMLPhish Variants

TABLE II: HTMLPhish-Full Deep Neural Network

Layers	Values	Activation
Embedding	Dimension = 100	-
Convolution	Filter = 32, Filter Size = 8	ReLU
Max Pooling	Pool Size = 2	-
Dense1	No. of Neurons = 10, Dropout = 0.5	ReLU
Dense2	No. of Neurons = 1	Sigmoid
Total Number of Trainable Parameters	412,388,597	-

collected a dataset of HTML documents from phishing and legitimate web pages over 60 days.

Also, to ensure the deployability of our model to real-word systems, our data set is required to provide a distribution of phishing to benign web pages obtainable on the Internet in the real-world ($\approx 10/100$) [25], [26]. Given that when a balanced dataset (1/1), is used, the results can yield a baseline error [27]. Consequently, our training dataset **D1** consisting of HTML documents from 23,000 legitimate URLs and 2,300 phishing URLs was collected between 11 November 2018 to 18 November 2018. **D1** dataset was used to train and validate the three different variants of our model (HTMLPhish-Character, HTMLPhish-Word, and HTMLPhish-Full). From 10 January 2019 to 17 January 2019, testing data set **D2** consisting of HTML document from 24,000 legitimate URLs and 2,400 phishing URLs were generated.

Note that $\mathbf{D1} \cap \mathbf{D2} = \emptyset$. Also, our testing dataset **D2**, is slightly larger than our training dataset **D1**. This is because learning with fewer data, and having decent tests on a broader test data means that the detection technique is generalised. This ensures that the features and model of classification include specific features from legitimate and phishing web pages and that the approach can be applied to the vast number of online Web pages. In total, our corpus was made up of 47,000 legitimate HTML documents and 4,700 phishing HTML documents, as shown in Table I.

The legitimate URLs were drawn from Alexa.com's top 500,000 domains, while the phishing URLs were gathered from continuously monitoring Phishtank.com. The web pages in our dataset were written in different languages. Therefore, this does not limit our model to only detecting English web pages. We manually sanitised our corpus to ensure no replicas or web pages that are pointing to empty content. Alexa.com offers a top list of working websites that internet users frequently visit, so it is an excellent source to be used for our aim.

VI. Evaluation of HTMLPhish Variants

A. Experimental Setup

Table II details the selected parameters we found gave the best performance on our dataset bearing in mind the unavoidable hardware limitation for our proposed HTMLPhish variants:

- HTMLPhish-Character
- HTMLPhish-Word
- HTMLPhish-Full

The three CNN models were implemented in Python 3.5 on a Tensorflow backend and a learning rate of 0.0015 in the Adam optimizer [22]. The batch size for training and testing the model were adjusted to 20.

All HTMLPhish and baseline experiments were conducted on an HP desktop with Intel(R) Core CPU, Nvidia Quadro P600 GPU, and CUDA 9.0 toolkit installed.

B. Evaluation Metrics

Because of the severely imbalanced nature of our dataset, we evaluated the performance of our models in terms of the Area under the ROC Curve (AUC). We also used the receiver operating characteristic (ROC) curve in our evaluation. The ROC curve is a probability curve, while the AUC depicts how much the model can distinguish between two classes, which for our model is - legitimate or phishing. The higher the AUC value, the better the performance of the model. The ROC curve is plotted with the true positive rate (TPR) against the false positive rate (FPR) where $TPR = \frac{(TP)}{(TP+FN)}$ and $FPR = \frac{(FP)}{(TN+FP)}$. Where TP, FP, TN, and FN stand for the numbers of True Positives, False Positives, True Negatives, and False Negatives, respectively.

Additionally, we employed the precision, True Positive Rate, and F-1 score metrics to evaluate the performance of HTMLPhish and the baseline models. The True Positive Rate computes the ratio of phishing HTML documents that are detected by the models. In contrast, the precision metrics compute the ratio of detected phishing HTML documents that are actual phishes to the total number of detected phishing HTML documents.

C. Overall Result

To record the performance of HTMLPhish-Full and the baseline models on the D1 dataset, we split the dataset into 80% for training, 10% for validation, and 10% for testing. Also, taking cognizance of how our data is severely imbalanced, we ensured we manually shuffled the datasets before training.

The ROC curves of HTMLPhish and its variants are shown in Figure 4. From the result detailed in Table III, in general, HTMLPhish-Full significantly outperforms the other two variants: HTMLPhish-Character, and HTMLPhish-Word. While HTMLPhish-Character and HTMLPhish-Word have similar performances, HTMLPhish-Full takes advantage of the strengths of both and produces more consistently better results. Also, HTMLPhish-Full offered a significant jump in AUC over the other variants, while HTMLPhish-Word performs slightly worse amongst the three.

On the D1 dataset, HTMLPhish-Full provided a 98% accuracy and 2% False Positive Rate. The minimal False Positive Rates indicates the ratio of legitimate web pages, which are incorrectly identified as a phish. This is helpful when the model will be deployed in real-world scenarios as users will

TABLE III: Result of HTMLPhish and Baseline Evaluations on the D1 dataset

Models	Accuracy	Precision	True Positive Rates	F-1 Score	AUC	Training time
HTMLPhish-Full	**0.98**	**0.97**	**0.98**	**0.97**	**0.93**	**6.75 mins**
HTMLPhish-Word	0.94	0.93	0.94	0.93	0.88	10 mins
HTMLPhish-Character	0.95	0.92	0.95	0.94	0.90	3.5 mins
[28]	0.97	0.96	0.97	0.96	0.93	5.25 mins
[20]	0.95	0.94	0.95	0.94	0.91	18 mins

TABLE IV: Result of HTMLPhish and Baseline Evaluations on the D2 dataset

Models	Accuracy	Precision	True Positive Rates	F-1 Score	AUC	Testing time
HTMLPhish-Full	**0.93**	**0.92**	**0.93**	**0.91**	**0.88**	**9 seconds**
HTMLPhish-Word	0.90	0.87	0.91	0.88	0.73	107 seconds
HTMLPhish-Character	0.91	0.89	0.91	0.89	0.77	7 seconds
[28]	0.91	0.84	0.91	0.87	0.73	15 seconds
[20]	0.90	0.90	0.92	0.90	0.78	112 seconds

not be inappropriately blocked from accessing legitimate web pages.

Considering the computational complexity of HTMLPhish-Full, it can be seen that on a dataset of over 25,000 HTML documents, HTMLPhish-Full can be speedily trained within 7 minutes. Once trained, HTMLPhish-Full can evaluate an HTML document in 1.4 seconds.

D. Comparison with State-Of-The-Art Techniques

We compared HTMLPhish-Full with the methodology, speed, and performance of existing state-of-the-art models in [20] and [28]. [28] is a Deep Neural Network with multiple layers of CNNs that takes as input word tokens from a URL to determine the maliciousness of the associated web page. On the other hand, [20] takes as input the character sequence of a URL and models its sequential dependencies using Long short-term memory (LSTM) neural networks to classify a URL as phishing or benign. We applied these techniques to the HTML documents in the D1 dataset and also tested them on the D2 dataset.

From the result detailed in Table III and Table IV, HTMLPhish-Full provides better precision, recall and comparable accuracy against the existing state-of-the-art models. The performance of HTMLPhish-Word and [28] can be attributed to the fact that it is trained on a definite dictionary of words from the training data. Therefore it will be unable to obtain useful embeddings for new words in the test data. HTMLPhish-Character and [20] perform better with respect to the AUC metric because the individual character embedding CNN can learn structural patterns in the HTML document and can also obtain feature representations for new words. This makes it easy to be applied to the test data. In addition, due to the limited number of characters, the scale of the CNN model using the individual embedding character remains fixed when compared to word-based model sizes. However, CNN models built with individual character embeddings cannot exploit structural information available in long sequences in the HTML document. It also disregards word borders and makes it challenging to differentiate special characters in the data.

Furthermore, CNN's using only character level embedding struggles to differentiate information for scenarios where phishing HTML documents try to imitate benign HTML documents through small modifications to one or few words in the HTML document[29]. This is because the Convolutional filters will likely yield similar output from a sequence of characters with a similar spelling. Therefore, CNNs using only character embeddings are not enough to obtain structural information from the HTML document in detail. That is the reason word embeddings must be taken into account. Consequently, HTMLPhish-Full takes advantage of both word and character embedding matrices to accommodate unseen words in the test data, and therefore yield a better result than the other variants and baseline models.

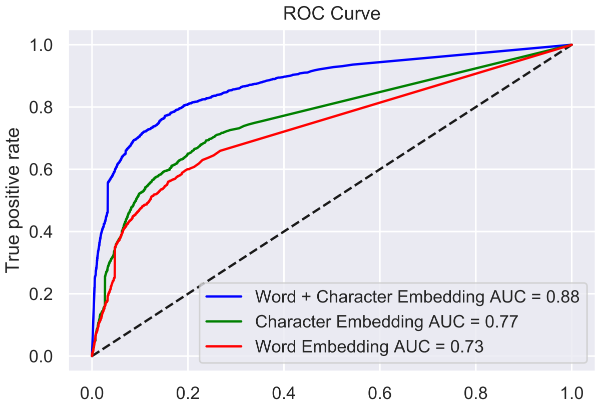

Fig. 4: The ROC Curve of HTMLPhish Variants

E. Temporal Resilience

The techniques for implementing a phishing web page is continuously evolving due to emerging technology applications for designing phishing web pages. The evaluation of the resilience of this evolution is paramount for a phishing web page detection technique. In this paper, we applied the longitudinal study [30] by evaluating the accuracy of the HTMLPhish-Full using freshly collected data. This study enabled us to infer a maximum retraining period, for which the accuracy of the system does not reduce. For a security supplier deploying HTMLPhish-Full in the wild, the retraining time frame can provide an approximate cost of maintenance.

Using the evaluation metrics detailed above, we compared the accuracy of HTMLPhish variants and baseline models on the training data **D1** with its accuracy when applied to the test data **D2** without retraining the model. From the results in Table IV, HTMLPhish-Full provided a 98% accuracy on the training dataset while yielding a 93% accuracy on the test dataset. The result of our longitudinal study demonstrates the readiness of HTMLPhish-Full for real-world deployment. HTMLPhish-Full will remain temporally robust, and will not need retraining within at least two months.

VII. Conclusion

In this paper, we proposed HTMLPhish, a deep learning based data-driven end-to-end automatic phishing web page classification approach. HTMLPhish receives the HTML content of a web page as input and applies CNNs to learn the semantic dependencies in both the characters and words in the HTML document in a jointly optimized network. Furthermore, we applied convolutions on a concatenation of the matrix of character and word embeddings in order to ensure the effective embedding of new words in the test HTML documents. Our approach can learn context features from HTML documents without requiring extensive manual feature engineering.

We evaluated our model using a comprehensive dataset of HTML contents presented in a real-world distribution. HTMLPhish provided a high precision rate, showing a temporally stable result even when it was trained two months before being applied to a test dataset.

The future work is to compare our model to feature engineering-based models that extract features only from the HTML document. Also, we intend to implement our model as a browser extension. This will enable HTMLPhish to recognise phishing websites in real-time.

Acknowledgment

The authors acknowledge the Petroleum Technology Development Fund (PTDF), Nigeria for the funding and support provided for this work.

References

[1] J. Lopez and J. E. Rubio, "Access control for cyber-physical systems interconnected to the cloud," *Computer Networks*, vol. 134, pp. 46–54, 2018.

[2] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from urls," *Expert Systems with Applications*, vol. 117, pp. 345–357, 2019.

[3] APWG, "Phishing activity trends report, 1st quarter 2018," Tech. Rep., 2018.

[4] D. Chattaraj, M. Sarma, and A. K. Das, "A new two-server authentication and key agreement protocol for accessing secure cloud services," *Computer Networks*, vol. 131, pp. 144–164, 2018.

[5] T. Acar, M. Belenkiy, and A. Küpçü, "Single password authentication," *Computer Networks*, vol. 57, no. 13, pp. 2597–2614, 2013.

[6] "Google safe browsing," http://code.google.com/apis/safebrowsing/, accessed: 2019-09-30.

[7] C. Amrutkar, Y. S. Kim, and P. Traynor, "Detecting mobile malicious webpages in real time," *IEEE Transactions on Mobile Computing*, no. 8, pp. 2184–2197, 2017.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[9] C. N. Gutierrez, T. Kim, R. Della Corte, J. Avery, D. Goldwasser, M. Cinque, and S. Bagchi, "Learning from the ones that got away: Detecting new forms of phishing attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 988–1001, 2018.

[10] E. Buber, B. Dırı, and O. K. Sahingoz, "Detecting phishing attacks from url by using nlp techniques," in *International Conference on Computer Science and Engineering (UBMK), 2017*. IEEE, 2017, pp. 337–342.

[11] G. Varshney, M. Misra, and P. K. Atrey, "A phish detector using lightweight search features," *Computers & Security*, vol. 62, pp. 213–228, 2016.

[12] S. Smadi, N. Aslam, and L. Zhang, "Detection of online phishing email using dynamic evolving neural network based on reinforcement learning," *Decision Support Systems*, vol. 107, pp. 88–102, 2018.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[16] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[17] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *arXiv preprint arXiv:1702.01923*, 2017.

[20] A. C. Bahnsen, E. C. Bohorquez, S. Villegas, J. Vargas, and F. A. González, "Classifying phishing urls using recurrent neural networks," in *Electronic Crime Research (eCrime), 2017 APWG Symposium on*. IEEE, 2017, pp. 1–8.

[21] H. Le, Q. Pham, D. Sahoo, and S. C. Hoi, "Urlnet: Learning a url representation with deep learning for malicious url detection," *arXiv preprint arXiv:1802.03162*, 2018.

[22] D. Kingma and J. Ba, "Adam: a method for stochastic optimization (2014)," *arXiv preprint arXiv:1412.6980*, vol. 15, 2015.

[23] L. Richardson, *Beautiful Soup*, 4th ed., 2017. [Online]. Available: https://www.crummy.com/software/BeautifulSoup.

[24] S. Marchal and N. Asokan, "On designing and evaluating phishing webpage detection techniques for the real world," in *11th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 18)*, 2018.

[25] C. Whittaker, B. Ryner, and M. Nazif, "Large-scale automatic classification of phishing pages." in *NDSS*, vol. 10, 2010, p. 2010.

[26] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: a content-based approach to detecting phishing web sites," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 639–648.

[27] E. Borgida and N. Brekke, "The base rate fallacy in attribution and prediction," *New directions in attribution research*, vol. 3, pp. 63–95, 1981.

[28] B. Wei, R. A. Hamad, L. Yang, X. He, H. Wang, B. Gao, and W. L. Woo, "A deep-learning-driven light-weight phishing detection sensor," *Sensors*, vol. 19, no. 19, p. 4258, 2019.

[29] W. Chu, B. B. Zhu, F. Xue, X. Guan, and Z. Cai, "Protect sensitive sites from phishing attacks using features extractable from inaccessible phishing urls," in *2013 IEEE International Conference on Communications (ICC)*. IEEE, 2013, pp. 1990–1994.

[30] S. Marchal, G. Armano, T. Gröndahl, K. Saari, N. Singh, and N. Asokan, "Off-the-hook: An efficient and usable client-side phishing prevention application," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1717–1733, 2017.