Proactive Detection of Phishing Kit Traffic

Qian Cui[1,2](✉), Guy-Vincent Jourdan[1,2](✉), Gregor V. Bochmann[1](✉), and Iosif-Viorel Onut[2](✉)

[1] University of Ottawa, Ottawa, Canada
gjourdan@uottawa.ca, bochmann@uottawa.ca
[2] IBM Centre for Advanced Studies, Ottawa, Canada
vioonut@ca.ibm.com

Abstract. Current anti-phishing studies mainly focus on either detecting phishing pages or on identifying phishing emails sent to victims. In this paper, we propose instead to detect live attacks through the messages sent by the phishing site back to the attacker. Most phishing attacks exfiltrate the information gathered from the victim by sending an email to a "drop", throwaway email address. We call these messages **exfiltrating emails**. Detecting and blocking exfiltrating emails is a new tool to protect networks in which a number of largely unmonitored websites are hosted (universities, web hosting companies etc.) and where phishing sites may be created, either directly or by compromising existing legitimate sites. Moreover, unlike most traditional antiphishing techniques which require a delay between the attack and its detection, this method is able to block the attack as soon as it starts collecting data.

It is also useful for email providers who can detect the presence of drop mailbox in their service and prevent access to it. Gmail deployed a simple rule-based detection system and detected over 12 million exfiltrating emails sent to more than 19,000 drop Gmail addresses in one year [52].

In this work, we look at this problem from a new perspective: we use a Recurrent Neural Network to learn the structure of exfiltrating emails instead of their content. We compare our implementation, called **DeepPK**, against word-based and pattern-based methods, and tested their robustness against evasion techniques. Although all three models are shown to be very effective at detecting unmodified messages, **DeepPK** is the overall more resistant and remains quite effective even when the messages are altered to avoid detection. With DeepPK, we also introduce a new message encoding technique which facilitates scaling of the classifier and makes detection evasion harder.

Keywords: Phishing kit · Exfiltrating emails · Network traffic detection

K. Sako and N. O. Tippenhauer (Eds.): ACNS 2021, LNCS 12727, pp. 257–286, 2021.
https://doi.org/10.1007/978-3-030-78375-4_11

1 Introduction

A so-called “phishing attack” is a cyber crime in which an attacker (the *phisher*) deploys a website that mimics another site in order to induce victims to provide sensitive information. Although significant efforts to the defend against phishing attacks have been made both in academia and in industry, the fight between attackers and defenders keeps going on. The Anti-Phishing Working Group (APWG) reports having detected 266,387 phishing sites during the third quarter of 2019, the highest number in three years, with more than two third using SSL, the highest percentage seen since this is being tracked [5]. Phishers keep improving their techniques to avoid detection, for example using SSL or adding multiple redirections [6].

Most of the literature on anti-phishing focuses either on detecting phishing emails sent to the victims (e.g. [11,22,26,36,49,53] and many more) or on detecting phishing web pages(e.g. [2,12,13,18,23,32,44,45,59] and many more). These solutions are centered around the victims, the goal is to protect the victims from the attacks, that is, to cut off the channel between victims and attacks. However, very little work has been done focusing on the channel between the attackers and the attacks. However, that channel is as critical as the other ones: breaking it defeats the attack. This is the topic of this paper, and more specifically the channel through which the attacker collects the stolen information from the phishing site. This approach helps web hosting provider and network owner to combat phishing by detecting immediately that an attack is being deployed on their network. It is a new idea that is not centered around the actual victims (most victims have no connection to the network on which the attack is being deployed) and thus this is a new tool which can work in combination with existing ones.

In most phishing attacks, the stolen information is exfiltrated back to the phisher by email: the code of the phishing site simply sends an email to a “drop address” each time someone submits something to the phishing site. Each email contains the data submitted by one victim. In the case of a multi-page phishing site, it is even often the case that several emails are sent for a single victim. Therefore, detecting and blocking these emails is a different and complementary means to combat phishing attacks. In the following, we call these emails sent by the phishing site to the phisher “**exfiltrating emails**”.

In this paper, we evaluate three different machine learning technique to detect exfiltrating emails: word-based, pattern-based and structure-based detection. We test the robustness of our three models against potential attacks. Although all three models are shown to be very effective at detecting these messages, the model using a deep-learning approach, which is called **DeepPK**, is the one that is the overall best since it remains quite effective even when the messages are altered to avoid detection. The key idea of **DeepPK** is to deal with the email structure as a sequence of components that follows specific grammar rules. The key component of **DeepPK** is a bidirectional Long Short-Term Memory (LSTM) network [46]. It allows **DeepPK** to automatically learn the difference between the structural grammar rules of exfiltrating emails and regular emails. In order to effectively represent email structure, we propose a new encoding method, the **structure token**, which uses a small corpus containing only 14 symbols.

We train and test our models on a realistic database of exfiltrating emails. These emails are built from the combination of two real datasets: a database of exfiltrating emails generated by actual "phishing kits", which gives us the patterns of the exfiltration emails, but not the data provided by the victims, and a database of values that have been submitted to real phishing sites. By meshing up these two databases, we end up with a system that can generate a large number of exfiltrating emails. In this paper, we use almost 65,000 such messages to train and test our models.

The solution described in this paper has a key advantage over most of the existing ones: it does not require the attack to be first reported, or to be somehow actively discovered. Instead, it is the attack's own network traffic that is being detected and stopped. Therefore, this technique can be used to stop a phishing attack immediately, preventing a single delivery of stolen information to the phisher.

The paper is organized as follows: In Sect. 2 we explain what detecting exfiltrating email achieves that current phishing detection method do not. In Sect. 3, we present our exfiltrating emails database. Then in Sect. 4, we introduce our machine learning-based approaches. In Sect. 5, we present the evaluation of our models, which is followed by the robustness test in Sect. B. We provide an overview of the literature in Sect. 6 before concluding in Sect. 7. All of the source code and some of the non-sensitive data used in this paper will be made available after the anonymous review.

2 Motivations

As already mentioned, most of the antiphishing efforts are directed at protecting victims, either by preventing the attacker's message from reaching its target, or by detecting that a site is not genuine. However, not every potential victim uses these mechanisms, and even when they do, these mechanisms are not perfect: for instance, Hu et al. [29] have shown that even email providers that do apply anti-spoofing detection techniques fail to always prevent forged emails from reaching the victims. It is therefore also important to help network administrators to proactively detect that a phishing site has been deployed on their network, without being notified of the URL fist. Such a phishing site can be deployed on a network because the attacker has compromised one of the servers, or because the attacker as a legitimate right to deploy a website there. Very little work has been done in this area. Of course, one could use any phishing site detection method and scan the network to look for such sites, but this can be difficult due to the network's size and the lack of control over what is being deployed there. More importantly, scanning would probably yield limited success without first somehow knowing the actual URL of the phishing site on the server. Waiting to be notified about the attack has the obvious disadvantage of being out of the control of the network's administrator, and of opening a window of time during which the attack is live on the network. Closing that gap is necessary to prevent victims from providing data to the phishing site, and to preserve the reputation of the network, which may otherwise end up being blacklisted.

Detecting exfiltrating emails is a new tool which provides a web hosting providers a new way to learn that a phishing attack is hosted on their network and stop it immediately, by monitoring outgoing emails instead of scanning their own network. It addresses the problem of having to find out or to be informed of the exact URL of the phishing attack, since that is instead the network traffic of the phishing site itself that triggers the detection. Another considerable advantage of such a system is that it can detect and block the attack as soon as someone submits data to the site, preventing the attackers from collecting any information. In addition, drop email addresses are uncovered and can be reported to the email providers and suitable authorities.

This tool will be useful to web hosting companies, but also to any entities managing a large and relatively open network, such as a university for example.

As already demonstrated in [52], our work can also be useful to email providers: it is possible to reliably and rapidly detect that one of the mailboxes is the recipient of such exfiltrating emails and block its access immediately, also completely preventing the attacker from accessing the data. This is of particular interest to free email providers, which are used extensively by phishers to create dedicated drop email addresses.

In this research, we focus on phishing attacks that are using clear-text drop emails as exfiltration techniques. It is always surprising to the academic community that such a basic and vulnerable exfiltration technique could be used in practice. A vast array of other techniques are of course possible to exfiltrate the data, including but not limited to simple email encryption, pushing the data out using another protocol such as http(s) or (s)ftp, more covert methods such as DNS-based exfiltration [38], or storing the data on the server and switching from a push-based model to a pull-based model. Several easy-to-find implementations of phishing-kit proof-of-concepts do in fact provide alternate ways of exfiltering data. In practice, the almost exclusive reliance on plain-text emails is well documented by practitioners [20,31,34,39,42,52,57]. Most recently, in [52] it is reported that all of the 10,000 kits analyzed in that study use the PHP *mail()* command to exfiltrate data. In [31] an analysis of 1,000 Phishing Kits done in 2018 found that "the vast majority of kits (98%) used email to exfiltrate stolen data to attackers". Another study from 2018, [39], does not mention any other mean of data exfiltration. This is certainly also our empirical evidence having worked with well over 10,000 live attacks over the past couple of years: attackers today use almost exclusively clear-text drop emails for data exfiltration. Even when the phishing kit offers other alternative (usually some level of encryption), these alternatives are almost never enabled in live attacks. One explanation for this is that phishing attacks are very low-skill attacks, and any complication would negatively impact the model (see Sect. 7 for some more discussion about this). It is also possible that only some of the attacks are using clear-text drop emails, and for some reasons these are the attacks that we discover.[1] Even if that is the case, it remains that a large number of attacks are using clear-text drop emails as exfiltration techniques and stopping these ones is a step in the right direction.

Our tool is not meant to replace existing ones. Detecting classical phishing emails is still necessary but serves a different purpose: it prevents email users of the domain from being victimized by phishing sites that are usually hosted somewhere else. Our tool is as a new and effective mechanism to secure networks against hosting phishing attacks themselves. Classical phishing email detection does not provide any direct protection against that.

We believe that there are two main contributions in this paper: first, we provide a new direction for detecting exfiltrating emails using neural networks trained on the structural information of the message. We introduce a encoding method which effectively extracts that structural information with only 14 characters. Second, we identify a missing piece in the fight against phishing. The hosting mechanism and the data exfiltration techniques are an essential and somewhat overlooked part of the equation. The detection models that we present here do work very effectively on current phishing attacks. Other detection models might be as effective, and attackers will certainly take countermeasure to prevent detection in the future. Nevertheless, it remains that web hosting providers must now be included in the defense against phishing, and that proactive techniques such as the one presented here must be developed and maintained as the situation evolves.

3 Exfiltrating Emails Generations

One difficulty with this research is to access to exfiltrating emails to train and test the models. We are not aware of any such database prior to this work. Some prior work could have indirectly access to some exfiltrating emails (e.g. using honeypots [27]) but in limited quantity.

In this work, the starting point for the generation of exfiltrating emails is two datasets that the forensic teams of our industry partners have collected from real attacks:

1. A set of 3,162 distinct **Phishing Kits** which are actual phishing websites written in the PHP language,
2. and a collection of 370 files containing various amount of data collected by real phishing sites.

The generation process involves three stages: Phishing Kit Deployment, Files Parsing, and Email Generation described in the next subsections.

[1] Maybe because these are low-skill attacks, and some higher-skill attacks are evading our detection.

3.1 Phishing Kit Deployment

Each phishing kit is deployed in a custom sandbox environment. By redefining functions and certain global objects of the standard library (PHP language) used by the phishing kits, it is arranged that the calls requesting values for HTTP GET/POST request variables and cookies will return special placeholder values which we can later use to identify the value which is requested e.g. a POST request variable named "username".

Any email messages sent are captured. These messages are parsed, identifying all special placeholder values in addition to a small number of special patterns including IP address, date/time and user agent, with the end result being a sequence of static strings and dynamic value specifications termed an **email template**. A sample is shown in Fig. 1. A total of 6,448 unique email templates acceptable for use in subsequent steps are generated from the data. As previously noted, phishing kits often send more than one message, either because the attack is done in several steps and each step triggers a separated message, or because the phishing kit contains more than one phishing sites.

```
———=F3dreport 2018=———
Em@il: <EMAIL>
pass: <PASSWORD>
pass2: <PASSWORD>
———=IP Address & Date=———
IP Address: <IP>
Country: <COUNTRY>
Date: <DATE>
```

Fig. 1. Sample exfiltrating email template extracted from a phishing kit (manually modified for obfuscation)

3.2 Data File Parsing

Our data files contain sets of values that have been collected during phishing attacks and recovered by forensics teams. These values correspond to what the victims provide to the phishing site (and thus what is then exfiltrated in the emails). The type of data found in this dataset is what one expects from a phishing site: mostly credentials for websites and other systems, but also credit cards information and other personal information. In addition, the IP address of the victim, time of access, type of browser etc. is often collected by phishers.

It is worth noting that in a typical phishing attack, the majority of the values submitted to the site are not genuine. Instead, the majority of the inputs seem to come from users attempting to "get back" at the phishers by submitting a flurry of random data, insults and denial-of-service attempts. Nevertheless, these are the values that a typical phishing attack will receive and exfiltrate, and thus all of these values are valid and indeed necessary for our purpose.

We did parse all of our data files to extract the individual values and match them to the values requested by the phishing kits. The end result of this process is the population of an **Exfiltration Database** with data for 115,713 entries comprising 332,224 values.

3.3 Email Generation

The general idea is to generate emails from each email template by filling in placeholders using data from the exfiltration database. For each of the 6,448 email templates, we generate 10 email messages randomly filling in placeholders using data from the exfiltration database. When doing so, we require that all template values are populated, although we do not insist that the data all belongs to a single entry or even to data from the same file. This resulted in 64,480 exfiltration emails, two examples are provided in Fig. 2.[2] To ensure that our models are trained and tested on different datasets, email messages coming from the same template are either all used for training or all used for testing.

```
———=F3dreport 2018=———
Em@il: victim1@gmail.com
pass: victim1pass
pass2: victim1pass2
———=IP Address & Date=——
—
IP Address: 123.123.123.222
Country: Unknown
Date: 2018-12-14 04:16:11
```

```
———=F3dreport 2018=———
Em@il: victim2@hotmal.com
pass: victim2test11
pass2: victim2test11
———=IP Address & Date=——
—
IP Address: 123.123.12.12
Country: Unknown
Date: 2018-12-12 01:23:19
```

Fig. 2. Two instances of exfiltrating emails generated from the template of Fig. 1, values manually obfuscated.

4 Methodology

We have trained three different models to recognize exfiltrating emails. In this Section, we first introduce two approaches that are commonly used in email classification: word-based and pattern-based detection model. We then introduce our structure-based model.

[2] Because these files do contain some sensitive data, we cannot publish this database as is. We will however make available the encoded version of the emails on which our deep learning algorithm works upon request and after verification.

4.1 Word-Based Detection Model

Naive Bayes approaches have been shown to be very successful in text classification task [8,58]. Therefore, we included one such implementation in our exfiltrating email classifiers.

Specifically, the model learns the conditional probability and the independent probability of each word from the training set, and uses these probabilities to predict the probability that a new text belongs to a certain category. Formally, we work from a set of documents consisting of n unique word tokens $[w_1, w_2, \ldots, w_n]$. These documents are classified into p categories $[C_1, C_2, \ldots, C_p]$. Each document can be represented as a vector $x = (x_1, ..., x_n)$, where x_i represents the relative weight of w_i in that document.

In our case, to effectively represent word features, we first extract consecutive alphanumeric characters using the regexp [0-9A-Za-z] to get a "word" list. We then apply 1-gram and 2-gram to create word tokens. The corpus of the model is built using the 5,000 most frequent tokens. We apply a "scaled term frequency" to calculate the frequency of the token. Formally, the scaled term frequency of the word token w_i in the document d_j is

$$1 + \log(\#\text{ of occurrences of } w_i \text{ in the document } d_j).$$

We then apply tf-idf using the scaled tf to vectorize the tokens. For vector normalization, we apply an "L2" normalization: the sum of squares of vector elements is 1. Finally, for each document (email), we end up with a 5,000-dimension vector.

The probability that a document of vector $(x_1, ..., x_n)$ belongs to the category C_k is $p(C_k|x_1, ..., x_n) = \frac{p(C_k)\prod_{i=1}^{n} p(x_i|C_k)}{p(x_1,...,x_n)}$.

Note that x_i is a TF-IDF value of the word token w_i, which is only related to the set of documents. In other words, given a set of documents, $p(x_1, ..., x_n)$ is a constant for each category C_k. Therefore, $p(C_k|x_1, ..., x_n)$ is proportional to $p(C_k)\prod_{i=1}^{n} p(x_i|C_k)$. We apply the Gaussian Naive Bayes algorithm to estimate the likelihood of features, $p(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_{C_k}{}^2}}\exp\left(-\frac{(x_i-\mu_{C_k})^2}{2\sigma_{C_k}{}^2}\right)$, where the parameters σ_{C_k} and μ_{C_k} are learnt by the model during training. $p(C_k)$ is also a learnable parameter, which is equal to

$$\frac{\#\text{ of documents in } k^{th} \text{ category}}{\#\text{ of documents}}$$

Once the model is trained, it is used to assign a new document of vector $x'_1, ..., x'_n$ to the category C_i which maximizes $p(C_k|x'_1, ..., x'_n)$. In the following sections, we name this model **NB**.

4.2 Pattern-Based Detection Model

In addition to using different set of words (when compared to regular emails), exfiltrating emails also tend to use singular patterns. For instance, they are often organized following the format: <header> + <field name> + <delimiter> +

<value>. Therefore, we also trained a classifier to look for patterns. We first encode the content of the messages using only five character classes: letters (L), digits (D), punctuation (P), newline (N) and whitespace other than newline (W). Each email is first encoded using these five classes. We then compute all n-grams of lengths 10 to 16 on the encoded email sets, exfiltrating emails and regular emails, and we keep only the n-grams that appear only in one of the two sets, that is, n-grams that are found at least once in the exfiltrating (resp. regular) training set but never appear in the regular (resp. exfiltrating) training set. A greedy set cover algorithm is applied to obtain a token cover set, which only covers the same set of documents. We derive a classifier using only the token cover set for the class of exfiltrating emails which classifies a document as exfiltrating if and only if its token set contains one of the tokens in the exfiltrating token cover set.

Formally, let t be a tokenizer function and A and B be email classes. Let $D(A, B) = \bigcup t(A) \setminus \bigcup t(B)$ and similarly let $D(B, A) = \bigcup t(B) \setminus \bigcup t(A)$. Finally, using a set cover algorithm, select a small subset $C(A, B) \subseteq D(A, B)$ such that $\{m \in A \mid t(m) \cap D(A, B) \neq \emptyset\} = \{m \in A \mid t(m) \cap C(A, B) \neq \emptyset\}$ and similarly for $C(B, A)$. Let C_0 be the set of clean messages, and C_1 be the exfiltrating emails. Define a classifier c by

$$c(M) = \begin{cases} 1 & \text{if } t(M) \cap C(C_1, C_0) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

In the following sections, we name this model **Set-cover**.

4.3 Structure-Based Detection Model

As discussed in Sect. 4.2, exfiltrating emails tend to follow a specific format that is rarely used in regular emails. If we look at the structure of the document as a grammar, exfiltrating emails and regular emails follow two different grammars. Deep learning algorithms are known to be effective at learning underlying grammars of text documents [7,14,51], therefore we also include a deep learning-based classifier.

As we did in Sect. 4.2, we first encode the message using using a new **structure token** using 14 symbols. The details of that encoding is provided in Appendix A.1. In addition to the structure token, our model also include two "semantic" features: the **content entropy** and the **text proportion**, which are detailed in Appendix A.2.

Recurrent Neural Networks (RNN) are often used for problems with sequential information as input and have been shown to be effective in a variety of natural language processing problems [9,35]. For this model, we use a Long Short-Term Memory (LSTM) RNN, which has been proved to perform well in dealing with complex patterns and long sequences [28,50]. The details of our use of LSTM, which we call **DeepPK**, are provided in Appendix A.3.

5 Experiment

We now report our basic results, starting with a description of our experiment environment.

5.1 Experiment Environment

We have developed DeepPK using Keras[3] with Tensorflow as the back end. For HTML emails preprocessing, we use Beautifulsoup[4] to extract the text from the HTML emails. Our models NB and Set-cover are implemented using Scikit-learn[5]. Our experiments are performed on a Windows-based system with an Intel i5 CPU at 3.5 Ghz and 16 GB RAM. DeepPK is trained and tested on a NVIDIA Geforce GTX 1060 with 6 GB RAM. Our source code can be found on our website, http://ssrg.site.uottawa.ca/phishing_kit/.

5.2 Exfiltration Email and Regular Email Database

We obtained our regular emails database from the **Enron email dataset**[6], which was collected and prepared by a third party organization, and contains about 0.5 million messages coming from 150 users. Our exfiltrating emails database, which consists of 64,480 messages from 6,448 unique exfiltration email templates are generated by the approach discussed in Sect. 3.

To ensure that training and testing data is separated, we first split our 6,448 unique exfiltration email templates into two sets at a ratio of 4:1: 5,158 templates are randomly selected for training, and the remaining 1,290 are used for testing. This yield 51,580 email instances for training, and 12,900 email instances are for testing. For the regular email database, we create a balanced training set by randomly sampling 51,580 messages from the Enron email dataset. For the regular email test set, we use 5 times the number of test exfiltrating emails, for a total of 64,500 regular emails. This unbalance is to mimic a real-life scenario in our tests, since exfiltration emails would be a fraction of the mail traffic in reality.

As described in Appendix A.3, we inject into some of the (encoded) exfiltration emails some length of tokens taken from regular emails in order to avoid learning only the prefix of these messages. Specifically, we inject into 8 of the 10 instances generated from the each template a token segment randomly sampled from the regular training set. The size of the segment is randomly selected between 1 and 50 characters.

In order to avoid overfitting during training, we further split our training set: 80% is used for the actual training, while 20% is used for validation. Accordingly, we end up with 41,260 messages in each exfiltration email set and regular email

[3] https://keras.io/.
[4] https://www.crummy.com/software/BeautifulSoup/bs4/doc/.
[5] https://scikit-learn.org/stable/.
[6] http://www.cs.cmu.edu/~enron/.

set used for training, and 10,320 messages in each set used for validation. During training, we store the model which yields the best performance on the validation set and then evaluate it on the test set.

5.3 Model Evaluation

In order to evaluate the effectiveness of our models, we compared them on similar experiments and report here the results. By default, we use the following parameters for DeepPK: input length is 600 and number of memory units is 128. Since DeepPK uses tumbling windows to process the data, to ensure a fair comparison, we also test the NB model with tumbling windows (that model is noted **NB-Window** below). We tried window sizes of 5 to 10 lines and report only the one with the best performance.

We apply five standard metrics to evaluate the performance of the models: false positive[7] (FP), false negative (FN), precision (pre)=$\frac{\text{TP}}{\text{TP+FP}}$ (TP stands for true positive), recall (rec)=$\frac{\text{TP}}{\text{TP+FN}}$ and f-score=$\frac{2*\text{pre}*\text{rec}}{\text{pre+rec}}$. The results are shown Table 1.

Table 1. Performance comparison between models

Performance comparison					
Model	# false positive (%)	# false negative (%)	Precision	Recall	F1 score
NB	728 (1.13%)	115 (0.89%)	94.61%	99.11%	96.81%
NB-window	2,596 (4.02%)	99 (0.77%)	83.14%	99.23%	90.48%
Set-cover	261 (0.40%)	285 (2.21%)	97.97%	97.79%	97.88%
Single LSTM	626 (0.97%)	**37 (0.29%)**	95.36%	**99.71%**	97.49%
Bidirectional LSTM w/o content feature	343 (0.53%)	65 (0.50%)	97.40%	99.50%	98.44%
Bidirectional LSTM with content feature	**221 (0.34%)**	63 (0.49%)	**98.31%**	99.51%	**98.91%**

For the NB model, we note that using a tumbling window improves the false negatives rate but at the expense of the false positives rate. For DeepPK, the model that only uses a single LSTM yields the best false negative rate (0.29%) but the worst false positive rate (0.97%). Through manual inspection of these false positives, we found that most of them are very short regular emails. The model that uses bidirectional LSTM fixes this issue thanks to the additional information provided by the backward direction. The performance is further improved by using our semantic features, which help the model correctly classify regular emails with a structure similar to that of the exfiltrating emails (e.g. the case shown in Fig. 3). In general, the model which uses bidirectional LSTM and semantic feature yields the best false positive rate (0.34%) and the best F1 score (98.91%) across all models.

[7] Here, a "positive" classification means that the message is flagged as exfiltrating email.

5.4 Model Robustness

Our results in the Sect. 5.3 show that all three proposed models perform well in detecting exfiltrating emails. In this section, we discuss several possible ways an attacker could modify exfiltrating emails to evade detection, and we evaluate how resilient the models are to these modifications. When looking at these potential detection evasion techniques, we specifically focus on solutions that would be relatively easy to implement for the attacker and would modify the exfiltrating emails without preventing automatic processing at the receiving end. More advanced evasion techniques are of course possible, but they would likely impact negatively the "business model" of phishing by requiring more advanced technical skills from attackers (see Sect. 7). Here, we consider two potential attacks:

- **Injection attack**. In this attack, the phisher injects additional noise into the exfiltrating email, which is otherwise unchanged. In practice, the injected text can be random strings, or pieces of text extracted from regular emails. The latter is a more effective attack because it introduces "negative" noise (segments possibly matching what the model has learned from the regular emails), which is more likely to result in misclassification. In our study, we consider a worst case scenario and use actual text segments from our regular email database to increase the chances of defeating the models. We test four different ways of injecting "negative" noise : injecting at the top of the message, at the bottom of the message, in the middle of the message, and finally scattering the injected text throughout the exfiltrating email.
  We run several experiments. When injecting top, middle or bottom of the message, we injected a size of text ranging from 10% to 100% of the original exfiltration email, measured by the length of the resulting structure token. So in the worst case, 50% of the resulting structure token comes from injected text. When scattering the injection throughout the text, the injection is measured in terms of number of lines in the original text. In our experiment, we increase the number of injected lines, going from one line randomly inserted in the original text to one line inserted between each line of the original text.
- **Replacement attack**. In this attack, the phisher replaces the text of the structure of the exfiltrating emails with strings that the model has rarely or never seen. The purpose of the attack is to eliminate "positive" indicators. An easy way to perform such an attack is to systematically replace existing field names with other strings. Note that because DeepPK detects exfiltrating emails based on our structure token and not on the message itself, this model is not impacted by this attack if the strings used for replacement are of the same length as the strings they replace (since it would yield the same structure token). In order to have an effective attack against our model, we apply what we have called "incremental injection", where the size of the injected stings is gradually increased.
  We run several experiments with this attack as well. First, as mentioned we change the length of consecutive tokens, trying various increments from 17 to 101. This ensures that each experiment produces a different structure token

Table 2. Attack test sets

Injection attack (injection proportion: from 0.1 to 1.0 by steps of 0.1)	
Label	Description
Inject_header	Injection of "negative" noise at the top
Inject_middle	Injection of "negative" noise in the middle
Inject_tail	Injection of "negative" noise at the bottom
Inject_line	Injection of "negative" noise scattered throughout the message
Replacement attack (incremental injection length: [17, 20, 33, 45, 52, 64, 78, 89, 96, 101])	
Replace_word	Replace words (continuous alphabetical characters) with randomly generated ones
Replace_non_word	Replace non-words with randomly generated non-words
Replace_all	Word and non-word replacement

fragments. For each length, we try three different types of replacements: we try to replace only "words" (that is, sequences encoded as C in the structure token). We then try to replace only "non-words" (that is, sequences encoded as N, L or S in the structure token), and finally, we try to replace everything.

In these experiments we use the model trained on the original database, so the modified exfiltration messages have never been seen by the models before. We do not report the results on the regular emails again, since these would not be impacted by these experiments. We use the test set discussed in Sect. 5.2. Instead of using 10 instances per template, we randomly choose one instance from each template, and end up with 1,290 exfiltrating emails that we modify for the experiments. As explained, the injected text segments are randomly sampled from the regular test set. In order to facilitate the comparison, we use the same random seed for all our experiments (Table 2).

When faced with injection attacks, in general, DeepPK performs well, with an error rate of at most 5%, except with the test set inject_line. On that test, the error rate increases with the proportion of injected text, to reach 28% at the top. This is because, as expected, this injection destroys the sequence of structure tokens, eliminating some key tokens. The Set-cover model is stable in the injection test, with an error rate of at most 6%. This is not surprising since the Set-cover model only looks for learned "bad" token in the message. Injecting noise does not impact the presence of these tokens and the noise is just ignored by this model. Still, except for the test set inject_line, the Set-cover model performs worse than DeepPK even with a relatively high proportion of injected text (up to 70 to 90% of the original message depending on the test). The NB model does not perform well in the injection test. The model breaks down significantly as more "negative" content is injected. The use of tumbling windows does help, but the performance is still worse than the other two models. More details are available in Fig. 9 of Appendix B.

The word replacement attacks has almost no effect to the performance of DeepPK, with an error rate peaking at 5%. On the other hand, the performance of DeepPK on the test sets replace_non_word and replace_all is quite inconsistent: it sometimes performs very well with an error rate of less than 2%, but in some cases the error rate goes above 80% (Fig. 10 of Appendix B). To better analyze this phenomenon, we have conducted a complete set of tests on the test set replace_all, ranging the injection proportion from 1 to 100, step by step. Out of these 100 tests, the error rate is below 10% 42 times, and below 5% 31 times. The explanation might be that "non-words" in the template are important indicators of exfiltrating email for DeepPK. However, to successfully conduct such an attack, the attacker needs to successfully break up the part of the structure that happens to have been learned by DeepPK, which is quite challenging and a process of trial and error. Generating such exfiltrating emails would be significantly more difficult than what is currently done. What is more, interpreting these emails once that are received would also be orders of magnitude harder than the current situation. Therefore, this attack, however effective, seems of limited practicality. Set-cover and NB are basically defeated by this attack, see Appendix B for more details.

6 Related Works

Most of studies on phishing attack detection focus on identifying phishing pages and phishing emails that are used to spread phishing links.

Most proposed phishing sites detection techniques look for some intrinsic characteristics of the attack. For instance, [16,24,33,37,40,54,55] use an array of machine learning models to train a binary classifier. Some work has also been done to compare these approaches [1,36]. But as mentioned in Sect. 2, detecting that a site is a phishing site does not address the needs of a network administrator if, as is the case in these papers, the site's exact URL is needed for the detection.

The main general approach for detecting phishing emails is to apply machine learning techniques to detect the characteristics of a content that is designed to deceive the victim. Fette et al. [22] propose such as method. The feature of their model mainly focus on the phishing link embedded in the email, such as the number of dots and the number of domains in URL, rather than the email content. They report a 99.5% accuracy and 0.13% false positive on a dataset of 860 phishing emails and 6,950 regular emails. In [48,53], the authors suggest to combine natural language processing techniques and contextual information to identify phishing emails. In [53], the authors report a 98% true positive rate and 0.7% false positive rate on a dataset of 2,000 phishing emails and 1,000 regular emails. In [48], the authors report an accuracy of 92.2% and a 4.9% false positive rate on a dataset of 14,370 phishing emails and 14,370 regular emails. Some researchers suggest to also use delivery information to detect phishing emails. In [11,26], sets of features such as the consistency between sender domain and the embedded link are used. Stringhini et al. [49] propose a detection model for spear phishing attacks by profiling the email sender: writing habits, composing habits, and interaction habits. Such behavioral-based detection would not be directly suitable for our purpose, since in our case no impersonation is taking place. However, none of these techniques would probably be very effective at detecting exfiltrating emails because exfiltrating emails do not contain URLs or deceptive text, and are sent to the attacker's drop email address and from the header's viewpoint are not different from regular emails.

The work most related to ours is [52], in which a large scale analysis on credential theft is conducted. The author work on a source of about 10,000 kits, and propose a method to extract phishing templates by parsing the kits source code. They then look for instances of these templates on Gmail, using Gmail's built-in anti-abuse detection system. They detect over 12 millions exfiltration emails between March 2016 and March 2017. This works confirms that most phishing attackers (who use Gmail 75% of the time for drop email address) simply use plain-text when exfiltrating emails and thus detecting and blocking these messages at the hosting site would currently be extremely effective. The detection method that they use is however based on text matching; as we have shown in the Sect. 5.4, attacker could simply evade detection merely by using different keywords. Our method is more resistant and is aimed primarily at hosting provider.

One general problem with the above methods is that the attacks need to be first discovered and reported, and this means some delay between the attack and its detection (about 10 h according to the report from APWG [25]). Our method can identify a phishing attack as soon as it starts to collect information. It basically prevents the attack to succeed at all if exfiltrating emails are scanned in real time, at the source or at the receiving end. In [17], two "zero delay" phishing attack detection methods are presented: one uses domain names to infer that a site will host an attack, and the other does proactive "blind" scanning of the network. By contrast, the method proposed here works regardless of the domain name used (in particular it works even when the domain name is not related to the attack) and will work without knowing nor guessing the URL of the attack.

The main difference between our work and all the above methods is that the goal of these methods is to protect a victim from an attack. Although it could indirectly help network administrators to detect a phishing site on their network, it usually requires the URL of the attack to be known, which usually means that someone needs to first report the attack to the administrator. In contrast, the goal of our work is to directly help the administrator to detect a phishing site on their network, and it does it automatically and without delay. In [27], a system is presented in which honeypots are safely deployed and phishing kit are monitored. This is probably the closest work to ours, but the aim is quite different. That system does not provide a way to detect an attack being deployed on a live network. It is however one possible way to learn new email exfiltration patterns and thus it can work and combination to our system. In [43,56], the authors propose to monitor spam botnets and infer regular expressions matching the messages sent by these botnets. A similar approach may also achieve good performance in our context. However, as explained before, in our case the attacker controls the entire channel, from the message creation to the message consumption, and thus simple rule-based systems would be easier to be defeated by simply changing the messages body, as we did in Sect. B. As we showed, the models that we propose, in particular our deep learning-base model, can be quite resistant to simple pattern modifications of the messages.

In addition to phishing detection, there is a significant body of academic work focusing on email classification for several purposes, such as spam detection. For instances, Blanzieri et al. [10] present a survey of supervised machine learning algorithms for spam detection in 2008. These methods treat the email content either as a set of word tokens, or as a text in natural language. A binary classifier is then trained based on the extracted features to identify spam. Some methods also combine other information, such as attachments, headers and embedded images to improve the performance. Elssied et al. [21] apply a k-means clustering technique to identify spam. Not all solutions rely on machine learning-based classifier, e.g. Pérez-Díaz et al. [41] propose a method using a set of rules.

In general, all these spam detection methods mainly focus on email content and use semantic features to build classifiers. To the best of our knowledge, we are the first to propose a machine learning method which uses structural features of the messages to classify emails.

7 Limitations and Conclusion

One clear limitation of our empirical evaluation is that the attacker does control the entire exfiltration system and therefore can in theory very easily change it to avoid detection. As previously mentioned, using simple email encryption or switching to a completely different exfiltration technique would defeat the detection methods evaluated in this paper. Such a switch would not be terribly difficult to achieve from the attacker viewpoint. We argue that forcing phisher to step-up their game and implement more advanced exfiltration techniques is a good thing that will hurt the business of phishing attacks. The main reason for this is that phishing attacks are very low-skill attacks. In [15], 15 phishing attack "vendors" are surveyed. In general, these individuals have very low technical skills, and are only claiming the most basic web-programming abilities. Their clients, who are the actual attackers, presumably have even lower technical skills. Empirically, we can confirm that the code that we have seen in thousands of phishing kits is of very low quality and does not suggest any kind of programming understanding. In [19], an analysis of the evolution of phishing attacks over time also shows that only the most basic updates are performed on live attacks by the attackers. Raising the technical bar even slightly will likely exclude many of the current players. Another reason is the low return that phishing attacks yield, and the poor quality of the data collected. In [15], it is reported that the cost of a tailor-made phishing site ranges from 15$ to 250$. As mentioned in Sect. 3.2, in our experience the vast majority of the data sent to a phishing site is bogus[8] and thus processing the data to identify usable information is a time consuming process. Adding a decryption step, or using less structured exfiltration format

[8] Anecdotally, the more advanced technical steps that we regularly see in phishing kits are techniques to prevent returning visitors from submitting data again, presumably in an attempt to limit the amount of fake data submission.

will complicate data processing further and reduce profitability even more. We can report that in practice, we have almost never seen an attack in which the phisher bothered encrypting the content of the exfiltration emails.

Of course, if our system or a similar one becomes widely adopted, this will force attackers to step up their game and e.g. start encrypting their messages. As explained, we think that this will hurt their business. Nevertheless, when that time comes, new detection techniques will have to be found, depending on the new exfiltration trends. For example, several approaches have been proposed to work on encrypted traffic by comparing the traffic pattern ending to the same destination [3,4,30]. If the main exfiltration technique remains email-based, then some protection could be expected from a wide adoption of standards such as SPF[9] and DKIM[10], which will limit the ability to successfully send email from hacked servers that are not meant to send emails.

Another possible criticism to our work is that we will not be able to detect exfiltrating emails that follow a completely different pattern. This criticism is mitigated by the fact that this new pattern can simply be added to our training set once known, and that we see much fewer patterns than there are attacks, suggesting a vast amount of code-sharing among phishers. It is in practice likely that our current model would catch many actual exfiltrating emails sent in North America and Europe at the time of writing. System such as the one described in [27] could also be used to discover new patterns as they are introduced.

We also acknowledge that our database is heavily biased toward North-American and European attacks. This is not a limitation of our method but a limitation of our database. Training our model on a larger database should address this issue.

The solution proposed here is, as far as we know, the first one that suggests to detect exfiltrating emails using structural information. This method has the advantage of working very well in our experiments, and being robust against evasion techniques trying to avoid detection by modifying the email content. We also introduce a new "structure token" which proves to be very effective when combined with our deep learning algorithm. Our work is also the first one to our knowledge to be tested on synthetic but realistic exfiltration emails, using a combination of two real datasets.

Unlike usual solutions that can be deployed at the end-user end, our solution needs to be deployed by host providers, where the phishing sites are being deployed, or by email providers, where the exfilrating emails are being received. This can be seen as a limitation, but also as a strength, since a handful of very large scale players could deploy our system and have a significant and immediate impact on phishing activities.

[9] https://tools.ietf.org/html/rfc7208.
[10] https://tools.ietf.org/html/rfc6376.

A Details About DeepPK

A.1 Structure Tokens

In order to compare the “structure” of the body of emails, we introduce what we call the **structure token**, which is a symbolic representation of that email structure. Formally, we encode the text of the message using four categories: letters ([a-zA-Z]), encoded as C, digits ([0–9]), encoded as N, line breaks ([\n\r]), encoded as L, and finally any character that does not belong to the previous categories, encoded as S. In addition, we count consecutive occurrences of characters in the same category and append the number of occurrences to the category symbol. For compactness, we do not append that number if it is 1. For instance, the text “Hi Yvonne\n This is John, please call me back.” is represented as the structure token “C2$\underline{\text{S}}$C6$\underline{\text{LS}}$C4$\underline{\text{S}}$C2$\underline{\text{S}}$C4S2C6$\underline{\text{S}}$C4$\underline{\text{S}}$C2$\underline{\text{S}}$C4$\underline{\text{S}}$” (where single instances of a category where the number 1 is omitted are $\underline{\text{underlined}}$).

There are several advantages to using such a structure token. First, it does not capture the actual text (the words) used in the message, and instead captures the structure of the content. For instance, in the example above, if some words are changed (e.g., greetings or names are modified), we still get a similar structure token. The number of consecutive occurrences of a particular category might change a little bit when a word is changed, but the sequence of categories will remain relatively stable. This adds significant value in our context because in exfiltrating emails, what will change between messages is the part containing the victim’s data. The remaining content is the **template**, which doesn’t change across messages sent by the same phishing attack. Figure 2 shows two instances of the same template. The “template” part (separators, fields name, line breaks) remains identical in both messages, and the corresponding structure tokens will match. In addition, it is often the case that the structure token will still be quite similar across messages in the parts containing victim’s data. For instance, all IP addresses end up with the structure token “NXSNXSNXSNX” where X $\in$ [‘’, 2, 3]. It is also true that using a structure token makes is more difficult for the attacker to evade detection, since it is not enough to modify the text of the template. A new template needs to be introduced to significantly change the structure token. Finally, last but not least, using a structure token insures that model learns patterns from one-way encoded inputs rather than directly from data containing sensitive information. This protects users data privacy both during training and at run time, since actual email content is never sent to the system.

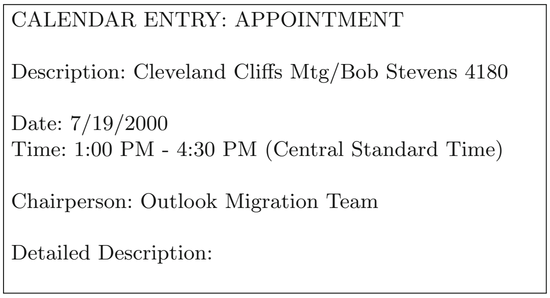
But a very important practical consequence of using structure token instead of traditional encoding methods, such as using words as encoding units, is that our method uses a very small corpus containing only 14 symbols[11] which allows our tokens to be applied to large datasets. In order to vectorize structure tokens, we apply the so-called “one-hot encoding”, which is a vector of bits of the same size as the encoding corpus, 14 bits in our case. Each bit corresponds to the

[11] Our four categories, C, N, L and S, and the 10 digits, 0 to 9.

index of one of the symbols in the corpus, and each character is being encoded with a vector in which only one bit is set to 1. As an example, given a corpus {a,b,c}, ‘a’ could be encoded [1, 0, 0], ‘b’ encoded [0, 1, 0] and ‘c’ encoded [0,0,1]. The one-hot encoding string of the text “aacb” would then be [[1, 0, 0], [1, 0, 0], [0, 0, 1], [0, 1, 0]].

A.2 Semantic Feature of Email

Our initial intent was to use only structure tokens to identify exfiltrating emails. However, we noticed that this resulted in a handful of false positives in the odd cases where regular emails follow a structure similar to exfiltrating emails. Figure 3 shows one such example.


CALENDAR ENTRY: APPOINTMENT

Description: Cleveland Cliffs Mtg/Bob Stevens 4180

Date: 7/19/2000
Time: 1:00 PM - 4:30 PM (Central Standard Time)

Chairperson: Outlook Migration Team

Detailed Description:

Fig. 3. One example of false positives

In order to correctly classify these messages, we enhance our method by introducing two “semantic” features: the **content entropy** and the **text proportion**.

Entropy is a commonly used metric in information theory. It measures the uncertainty of a piece of information produced by a source of data [47]. Formally, given a string S consisting of n characters $\{c_1, c_2, ..., c_n\}$ that are generated by a corpus of k unique symbols, the entropy of S, $ent(S) = -\sum_{i=1}^{m} p(s_i) * log(p(s_i))$, where m is the number of symbols used in the string S, and $p(s_i)$ is the probability of symbol s_i appearing in S. The higher the value of entropy, the more disordered or uncertain the string generated by the corpus. However, entropy has a tendency to generate greater values for the string that uses a large variety of symbols. In order to alleviate this tendency, we divide the initial number by the logarithm of the number of symbols in the string. Finally, we end up with a normalized entropy in the range [0,1]: $ent_{normal}(s) = -\sum_{i=1}^{m} \frac{p(s_i)*log(p(s_i))}{log(m)}$.

In our case, we use the above normalized entropy and a corpus of 26 English letters ([a–z]) and 10 digits ([0–9]) to build what call the **content entropy**. Specifically, we first convert email text into lowercase. We then calculate the normalized entropy for the processed content and get the content entropy. Since a regular email is mainly composed of English words, which has a higher certainty than the content of an exfiltrating email (e.g. username and password), it yields a lower content entropy.

Another difference between exfiltrating emails and regular emails is that exfiltrating emails tend to use a greater proportion of non-numeric and non-letter symbols. In order to quantify this difference, we propose another context feature, the **text proportion**. Formally, given a string S consisting of n characters $\{c_1, c_2, ..., c_n\}$, the **text proportion** $TP(S)$ is defined with the following formula:

$$TP(S) = \frac{\sum_{i=1}^{n} LN(c_i)}{n}$$

$$\text{where } LN(c) = \begin{cases} 1 & \text{if } c \in \text{[a-zA-Z0-9]} \\ 0 & \text{otherwise} \end{cases}$$

As an example, the text proportions of the exfiltrating emails in Fig. 2 are 0.7065 (left) and 0.7097 (right), while the text proportion of the regular email in Fig. 3 is 0.7703, higher than Fig. 2.

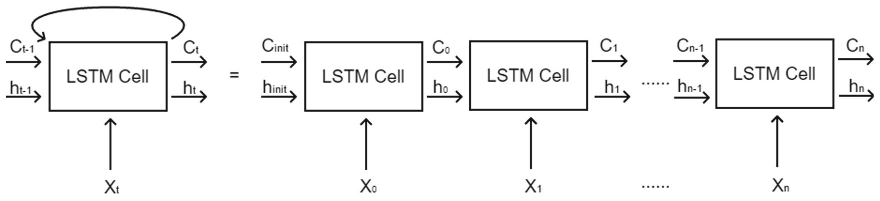


Fig. 4. LSTM cell and its unrolled form

A.3 Long Short-Term Memory Model

A Recurrent Neural Network (RNN) is a neural network where cells are connected in a round-robin fashion. Long Short-Term Memory (LSTM) is a type if RNN. As shown in Fig. 4, an LSTM cell has three inputs: X_t, C_{t-1} and h_{t-1}. X_t is the t^{th} character in the input sequence X. C_{t-1} is the state passed from the previous step, which stores the "memory" of what has been learned from the previous sequence. h_{t-1} is the output of the LSTM cell in the previous step, representing the latest prediction based on the previous sequence. The LSTM cell uses these values to calculate outputs, which are taken as the input in the next step.

Formally, $C_t = f_t * C_{t-1} + i_t * \tilde{C_t}$, where $f_t = \text{sigmoid}(W_f \cdot [h_{t-1}, x_t] + b_f)$, $i_t = \text{sigmoid}(W_i \cdot [h_{t-1}, x_t] + b_i)$ and $\tilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$. It can be seen that the new cell state C_t is equal to the partial previous status C_{t-1} plus the scaled update candidate $\tilde{C_t}$, and controlled by two gating components

f_t and i_t, that are the functions of the current element x_t and the output in the previous step h_{t-1}. In our context, these two gating components control the memory focus of the model during training: it keeps the memory of the key sequence and ignores the parts that do not contribute meaningful indicators for the model.

The output of the LSTM cell h_t is a function of the new cell state C_t. Formally, $h_t = o_t * \tanh(C_t)$, where $o_t = \text{sigmoid}(W_o \cdot [h_{t-1}, x_t] + b_o)$. The gating component o_t controls the output scale of the cell status. In our context, h_t is a vector indicator that identifies whether the currently processed token comes from an exfiltrating email.

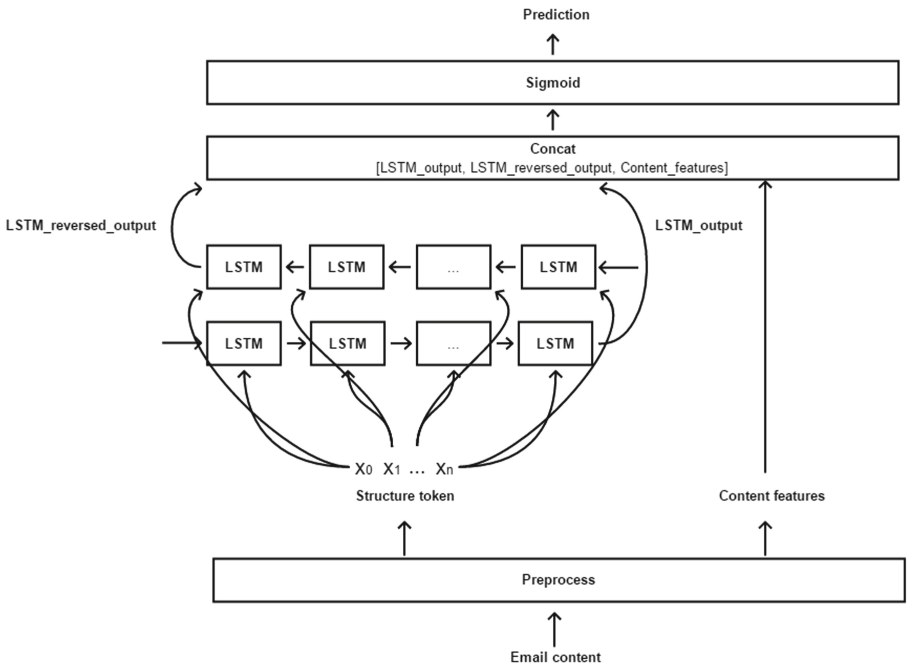


Fig. 5. System design of DeepPK

Detection Model. In order to construct our detection model, we pass the structure token through the LSTM cell and combine the LSTM output in the final step with the content features to yield the final prediction. A problem with using a single LSTM cell is that the output of the LSTM cell in the final step may not provide complete information of email structure. To overcome this issue, we apply a variant of LSTM: the **Bidirectional LSTM**, which uses a reversed copy of the input sequence to train an additional LSTM cell. Therefore, the model is able to know the complete information of the input in both directions [46]. We call this detection model **DeepPK**. The complete overview is shown Fig. 5. Additional information about **DeepPK**'s parameters are provided Appendix A.4.

- **Preprocessing Model**. When an email is classified, the first step is the preprocessing model. In this model, we first parse the text of the email body. If it is a HTML email, we scan all HTML tags and extract the text from each tag. We then generate the structure token and the semantic features based on the text content. Different message bodies yield structure tokens of different lengths. However, LSTM cell requires fixed-length input. By trial and error, we have selected a "reasonable" size as the input length (the details of the selection of the input length is discussed in the Appendix A.5). For the structure tokens that are longer than this input length, we use a tumbling window of the input length to create several non-overlapping token segments for that message. For the structure token that are shorter than the input length (or for the last token segment when several are created), we simply pad them with placeholders. Finally, the token segments are encoded into one-hot vectors and used as the input of our LSTM model.
- **Bidirectional LSTM**. A Bidirectional LSTM model consists of two LSTM cells. The output of the forward LSTM cell (LSTM_output) and the backward LSTM cell (LSTM_reversed_output) are joint together with the semantic features to form a new feature vector, which is later used as the input of the sigmoid output layer to yield the final prediction. The output of Sigmoid indicates the probability that the given email is an exfiltrating email.

Training Stage and Testing Stage. As mentioned above, we use a tumbling window of the input length to split each message into multiple non-overlapping token segments, and pad the last one. During training, each token segment is treated as an individual ground-truth sample. In other words, the model only knows if the token segments are from exfiltrating emails and cannot link segments of the same message back together. On the test set, multiple token segments from the same message are treated as a complete identifier. A message is classified as exfiltrating email if and only if one of its token segments is detected as such.

Injection on Training Set. As discussed in Sect. A.3, the function of the LSTM cell is to extract and learn key structure tokens from exfiltrating emails. However, when the training set is not sufficiently diverse, the model may fail to learn useful token sequences and instead may only remember some sequence or symbols at a specific position. For instance, exfiltrating emails often contain some series of dashes at the beginning. As a consequence, the structure token of these exfiltrating emails starts with the symbol S. In contrast, regular emails normally start with greetings, so the structure token of most regular emails starts with C. If such a training set is used to train the model, it causes the model to only use the first symbol as a strong indicator of exfiltrating emails and ignore the subsequent sequence. It causes the model to be very vulnerable in practice because an attacker can easily fool it, e.g. by embedding the exfiltrating email into a regular email.

In order to solve this issue, we randomly inject structure token fragments of different lengths, that are sampled from regular emails. To prevent learning these injected fragments, we inject the fragments that are sampled from the regular training set.

A.4 Analysis of DeepPK

In this section, we discuss the impact of various parameters in DeepPK's performance.

Our results are shown Fig. 6. In general, we can see that the precision increases but the recall decreases with the number of memory cells and the size of the input. The recall is still quite stable and stays above 99% across the board. The input length plays an important role: a shorter input allows the model to recognize more exfiltrating emails (higher recall), but increases the false positive rate. This indicates that the model requires enough structural information to accurately classify the messages.

The model is less sensitive to the number of memory units (the precision remains above 94% across the board). The model with 128 memory units and an input length of 600 yields the highest F1 score.

A.5 Analysis of Structure Token Length

As discussed in Sect. A.3, we needed to select a "reasonable" length for the structure token, since the LSTM cell requires fixed-length input. A reasonable length is the length that is able to cover "enough" context for the model to learn the required information from the structure token. To determine that, we first look at the length distribution of the structure token length in the exfiltrating email database, as shown in the Fig. 7.

We can see that save a few instances that end up with a very long structure token, most exfiltrating tokens have fewer than 600 characters. Through manual inspection, we find that these instances with long structure tokens can be divided into two categories: one category comes from instances produced by a specific template that collects 70 fields, as shown in Fig. 8. It comes from a phishing attack targeting a Brazilian bank https://www.bradescoseguranca.com.br. The other category are instances of exfiltrating emails that are coming from end users that have attacked back the phishing site: in these messages, the fields are populated with extremely long dummy strings. We thus chose 600 as the input length for DeepPK, since this length can cover most exfiltrating emails. In fact, even for the instance that exceeds this length, the cropped part is often a repeat of the previous part.

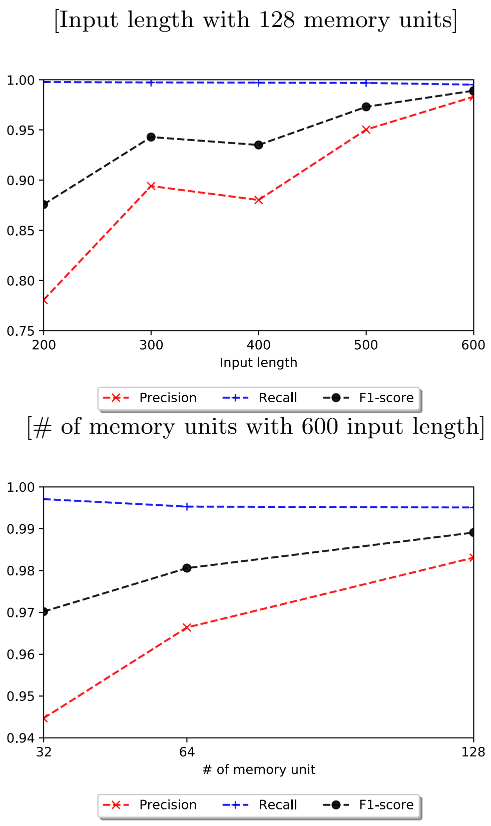


Fig. 6. DeepPK performance with different parameters

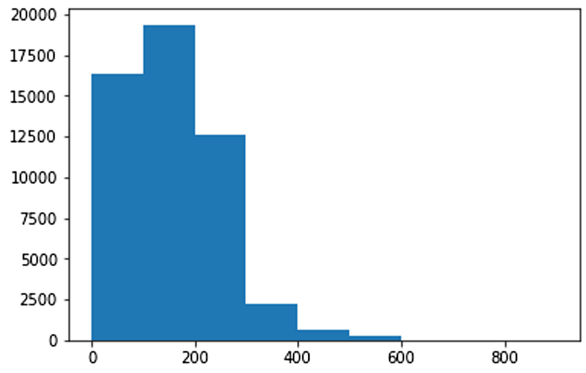
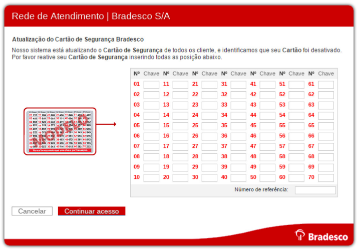


Fig. 7. Distribution of structure token length in the phishing database

```
========—B.r.a.d.e.s.c.o—========[XX]:**[
XX]:**[XX]:**[XX]:**[XX]:**[01]:**[02]:**[03]:**[04]:*
*[05]:**[06]:**[07]:**[08]:**[09]:**[10]:**[11]:**[12]:**[1
3]:**[14]:**[15]:**[16]:**[17]:**[18]:**[19]:**[20]:**[21]:
**[22]:**[23]:**[24]:**[25]:**[26]:**[27]:**[28]:**[29]:**[
30]:**[31]:**[32]:**[33]:**[34]:**[35]:**[36]:**[37]:**[38]
:**[39]:**[40]:**[41]:**[42]:**[43]:**[44]:**[45]:**[46]:**
[47]:**[48]:**[49]:**[50]:**[51]:**[52]:**[53]:**[54]:**[55
]:**[56]:**[57]:**[58]:**[59]:**[60]:**[61]:**[62]:**[63]:*
*[64]:**[65]:**[66]:**[67]:**[68]:**[69]:**[70]:**[00]:**=
==========
```



Rede de Atendimento | Bradesco S/A

Atualização do Cartão de Segurança Bradesco

Nosso sistema está atualizando o **Cartão de Segurança** de todos os cliente, e identificamos que seu **Cartão** foi desativado. Por favor reative seu **Cartão de Segurança** inserindo todas as posição abaixo.

Nº	Chave	Nº	Chave	Nº	Chave	Nº	Chave	Nº	Chave	Nº	Chave	Nº	Chave
01		11		21		31		41		51		61	
02		12		22		32		42		52		62	
03		13		23		33		43		53		63	
04		14		24		34		44		54		64	
05		15		25		35		45		55		65	
06		16		26		36		46		56		66	
07		17		27		37		47		57		67	
08		18		28		38		48		58		68	
09		19		29		39		49		59		69	
10		20		30		40		50		60		70	

Número de referência:

Fig. 8. Email template with long structure tag and its screenshot (In the actual exfiltration email, the data is where the "**" are in the figure.)

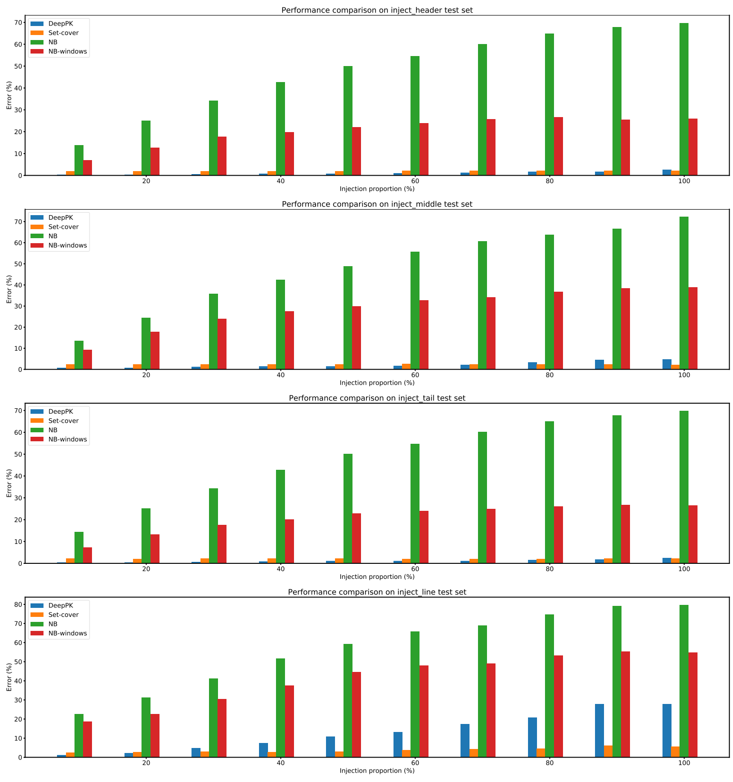


Fig. 9. Performance comparison on injection attack test sets

B Model Robustness

Set-cover does not fare well at all against replacement attacks, because this attack removes the information that these models have learned.

The apparent success of the model NB and NB-windows against replacement attack is misleading. It is because in these attacks, the model does not recognize anything at all and ends up with a zero vector. Since the model can only provide 2 outputs (exfiltrating email or non exfiltrating emails), this simply indicates that our model happens to defaults to an "exfiltrating email" output when the input is completely unknown. It also indicates that this model would flag as "exfiltrating emails" any message for which it knows none of the word.

It is noted that the replacement attack test we conduct is very strict: each structure token fragment in the attack instance is totally different from the original one, which may rarely occur in practice. Our results show that even under this extreme test, DeepPK can still provide reasonable performances.

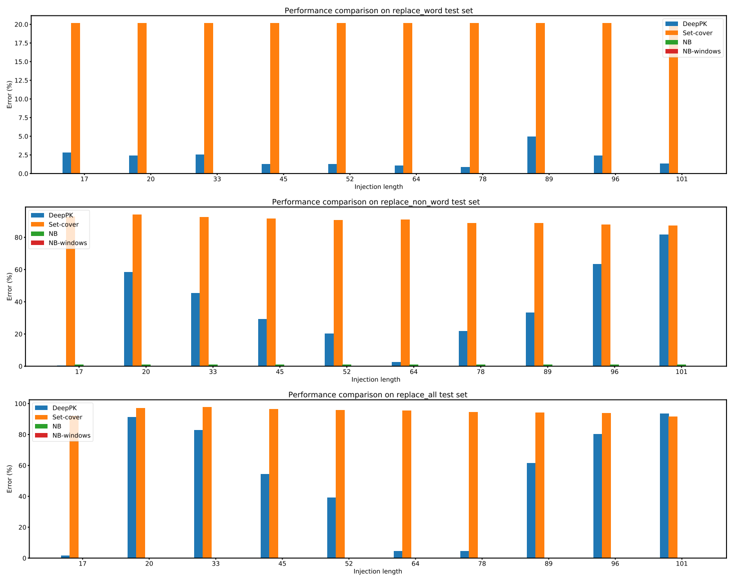


Fig. 10. Performance comparison on replacement attack test sets

References

1. Abu-Nimeh, S., Nappa, D., Wang, X., Nair, S.: A comparison of machine learning techniques for phishing detection. In: Proceedings of the Anti-phishing Working Groups 2nd Annual eCrime Researchers Summit, pp. 60–69. ACM (2007)

2. Afroz, S., Greenstadt, R.: Phishzoo: detecting phishing websites by looking at them. In: 2011 Fifth IEEE International Conference on Semantic Computing (ICSC), pp. 368–375. IEEE (2011)
3. Al-Obeidat, F., El-Alfy, E.S.: Hybrid multicriteria fuzzy classification of network traffic patterns, anomalies, and protocols. Personal and Ubiquitous Computing, pp. 1–15 (2017)
4. Alshammari, R., Zincir-Heywood, A.N.: Machine learning based encrypted traffic classification: identifying SSH and skype. In: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, pp. 1–8. IEEE (2009)
5. Anti-Phishing Working Group: Phishing Activity Trends Report 3rd Quarter in 2019. `docs.apwg.org/reports/apwg_trends_report_q3_2019.pdf`
6. Anti-Phishing Working Group: Phishing Activity Trends Report 4th Quarter in 2018. https://docs.apwg.org//reports/apwg_trends_report_q4_2018.pdf
7. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)
8. Behdad, M., Barone, L., Bennamoun, M., French, T.: Nature-inspired techniques in the context of fraud detection. IEEE Trans. Syst. Man Cybernet. Part C (Applications and Reviews) **42**(6), 1273–1290 (2012)
9. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A neural probabilistic language model. J. Mach. Learn. Res. **3**, 1137–1155 (2003)
10. Blanzieri, E., Bryl, A.: A survey of learning-based techniques of email spam filtering. Artif. Intell. Rev. **29**(1), 63–92 (2008)
11. Chandrasekaran, M., Narayanan, K., Upadhyaya, S.: Phishing email detection based on structural properties. In: NYS Cyber Security Conference, vol. 3. Albany, New York (2006)
12. Chang, E.H., Chiew, K.L., Sze, S.N., Tiong, W.K.: Phishing detection via identification of website identity. In: 2013 International Conference on IT Convergence and Security, ICITCS 2013, pp. 1–4. IEEE (2013)
13. Chen, T.C., Dick, S., Miller, J.: Detecting visually similar web pages: application to phishing detection. ACM Trans. Internet Technol. **10**(2), 5:1–5:38 (2010)
14. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
15. ClearSky Cyber Security: The Economy Behind the Phishing Websites Creation. https://www.clearskysec.com/wp-content/uploads/2017/08/The_Economy_behind_the_phishing_websites_-_White.pdf (2017)
16. Corona, I., et al.: DeltaPhish: detecting phishing webpages in compromised websites. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10492, pp. 370–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66402-6_22
17. Cui, Q.: Detection and Analysis of PhishingAttacks. Ph.D. thesis, University of Ottawa (2019)
18. Cui, Q., Jourdan, G.V., Bochmann, G.V., Couturier, R., Onut, I.V.: Tracking phishing attacks over time. In: Proceedings of the 26th International Conference on World Wide Web, pp. 667–676. International World Wide Web Conferences Steering Committee (2017)
19. Cui, Q., Jourdan, G.-V., Bochmann, G.V., Onut, I.-V., Flood, J.: Phishing attacks modifications and evolutions. In: Lopez, J., Zhou, J., Soriano, M. (eds.) ESORICS 2018. LNCS, vol. 11098, pp. 243–262. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99073-6_12
20. EC-Council: How Strong is your Anti-Phishing Strategy? (2018). https://blog.eccouncil.org/how-strong-is-your-anti-phishing-strategy/

21. Elssied, N.O.F., Ibrahim, O., Abu-Ulbeh, W.: An improved of spam e-mail classification mechanism using k-means clustering. J. Theoret. Appl. Inf. Technol **60**(3), 568–580 (2014)
22. Fette, I., Sadeh, N., Tomasic, A.: Learning to detect phishing emails. In: Proceedings of the 16th international conference on World Wide Web, pp. 649–656. ACM (2007)
23. Geng, G.G., Lee, X.D., Wang, W., Tseng, S.S.: Favicon - a clue to phishing sites detection. In: eCrime Researchers Summit (eCRS), pp. 1–10, September 2013
24. Gowtham, R., Krishnamurthi, I.: A comprehensive and efficacious architecture for detecting phishing webpages. Comput. Secur **40**, 23–37 (2014)
25. Group, A.P.W.: Global Phishing Report 2H 2014 (2014). http://docs.apwg.org/reports/APWG_Global_Phishing_Report_2H_2014.pdf
26. A. Hamid, I.R., Abawajy, J.: Hybrid feature selection for phishing email detection. In: Xiang, Y., Cuzzocrea, A., Hobbs, M., Zhou, W. (eds.) ICA3PP 2011. LNCS, vol. 7017, pp. 266–275. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24669-2_26
27. Han, X., Kheir, N., Balzarotti, D.: Phisheye: Live monitoring of sandboxed phishing kits. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1402–1413. ACM (2016)
28. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
29. Hu, H., Wang, G.: End-to-end measurements of email spoofing attacks. In: 27th {USENIX} Security Symposium ({USENIX} Security 2018), pp. 1095–1112 (2018)
30. Husák, M., Čermák, M., Jirsík, T., Čeleda, P.: Https traffic analysis and client identification using passive SSL/TLS fingerprinting. EURASIP J. Inf. Secur. **2016**(1), 6 (2016)
31. Imperva: Our Analysis of 1,019 Phishing Kits (2018). https://www.imperva.com/blog/our-analysis-of-1019-phishing-kits/
32. Liu, W., Liu, G., Qiu, B., Quan, X.: Antiphishing through phishing target discovery. IEEE Internet Comput. **16**(2), 52–61 (2012)
33. Ludl, C., McAllister, S., Kirda, E., Kruegel, C.: On the effectiveness of techniques to detect phishing sites. In: M. Hämmerli, B., Sommer, R. (eds.) DIMVA 2007. LNCS, vol. 4579, pp. 20–39. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73614-1_2
34. McCalley, H., Wardman, B., Warner, G.: Analysis of back-doored phishing kits. In: Peterson, G., Shenoi, S. (eds.) DigitalForensics 2011. IAICT, vol. 361, pp. 155–168. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24212-0_12
35. Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S.: Recurrent neural network based language model. In: Eleventh Annual Conference of the International Speech Communication Association (2010)
36. Miyamoto, D., Hazeyama, H., Kadobayashi, Y.: An evaluation of machine learning-based methods for detection of phishing sites. In: Köppen, M., Kasabov, N., Coghill, G. (eds.) ICONIP 2008. LNCS, vol. 5506, pp. 539–546. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02490-0_66
37. Mohammad, R.M., Thabtah, F., McCluskey, L.: mohammad2014. Neural Computi. Appl **25**(2), 443–458 (2014)
38. Nadler, A., Aminov, A., Shabtai, A.: Detection of malicious and low throughput data exfiltration over the DNS protocol. Comput. Secur. **80**, 36–53 (2019)

39. Oest, A., Safei, Y., Doupé, A., Ahn, G., Wardman, B., Warner, G.: Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis. In: 2018 APWG Symposium on Electronic Crime Research (eCrime), pp. 1–12, May 2018. https://doi.org/10.1109/ECRIME.2018.8376206
40. Pan, Y., Ding, X.: Anomaly based web phishing page detection. In: null. pp. 381–392. IEEE (2006)
41. Pérez-Díaz, N., Ruano-Ordas, D., Mendez, J.R., Galvez, J.F., Fdez-Riverola, F.: Rough sets for spam filtering: Selecting appropriate decision rules for boundary e-mail classification. Appl. Soft Comput. **12**(11), 3671–3682 (2012)
42. PhishLabs: How to Fight Back against Phishing (2013). https://info.phishlabs.com/hs-fs/hub/326665/file-558105945-pdf/White_Papers/How_to_Fight_Back_Against_Phishing_-_White_Paper.pdf
43. Pitsillidis, A., et al.: Botnet judo: Fighting spam with itself. In: NDSS (2010)
44. Ramesh, G., Krishnamurthi, I., Kumar, K.S.S.: An efficacious method for detecting phishing webpages through target domain identification. Decis. Support Syst. **61**(1), 12–22 (2014)
45. Rosiello, A.P.E., Kirda, E., Kruegel, C., Ferrandi, F.: A layout-similarity-based approach for detecting phishing pages. In: Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks, SecureComm, pp. 454–463. Nice (2007)
46. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Trans. Signal Process. **45**(11), 2673–2681 (1997)
47. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948)
48. Smadi, S., Aslam, N., Zhang, L., Alasem, R., Hossain, M.: Detection of phishing emails using data mining algorithms. In: 2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA), pp. 1–8. IEEE (2015)
49. Stringhini, G., Thonnard, O.: That ain't you: blocking spearphishing through behavioral modelling. In: Almgren, M., Gulisano, V., Maggi, F. (eds.) DIMVA 2015. LNCS, vol. 9148, pp. 78–97. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20550-2_5
50. Sundermeyer, M., Schlüter, R., Ney, H.: LSTM neural networks for language modeling. In: Thirteenth Annual Conference of the International Speech Communication Association (2012)
51. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014)
52. Thomas, K., et al.: Data breaches, phishing, or malware?: understanding the risks of stolen credentials. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1421–1434. ACM (2017)
53. Verma, R., Shashidhar, N., Hossain, N.: Detecting phishing emails the natural language way. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 824–841. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33167-1_47
54. Whittaker, C., Ryner, B., Nazif, M.: Large-scale automatic classification of phishing pages. In: In Proceedings of the Network & Distributed System Security Symposium (NDSS 2010), San Diego, CA, pp. 1–14 (2010)
55. Xiang, G., Hong, J., Rose, C.P., Cranor, L.: Cantina+: a feature-rich machine learning framework for detecting phishing web sites. ACM Trans. Inf. Syst. Secur. **14**(2), 21:1–21:28 (2011)

56. Xie, Y., Yu, F., Achan, K., Panigrahy, R., Hulten, G., Osipkov, I.: Spamming botnets: signatures and characteristics. ACM SIGCOMM Comput. Commun. Rev. **38**(4), 171–182 (2008)
57. Zawoad, S., Dutta, A.K., Sprague, A., Hasan, R., Britt, J., Warner, G.: Phish-net: investigating phish clusters using drop email addresses. In: 2013 APWG eCrime Researchers Summit, pp. 1–13, September 2013. https://doi.org/10.1109/eCRS.2013.6805777
58. Zhang, H., Li, D.: Naïve Bayes text classifier. In: 2007 IEEE International Conference on Granular Computing (GRC 2007), p. 708. IEEE (2007)
59. Zhang, Y., Hong, J., Lorrie, C.: Cantina: a content-based approach to detecting phishing web sites. In: Proceedings of the 16th International Conference on World Wide Web, Banff, AB, pp. 639–648 (2007)