# Detection and Analysis of Phishing Attacks

Qian Cui

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the Doctorate in Philosophy degree in Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

**Abstract**

The so-called "phishing attacks" are attacks in which a legitimate website is impersonated, in order to steal sensitive information from end-users. Phishing attacks represent one of the important threats to individuals and corporations in today's Internet. This problem has been actively researched by both academia and the industry over the past few years. Attempts to provide effective anti-phishing solutions have followed two main approaches: The first one is to identify a phishing attack by comparing its similarity to the target site. The second approach is to look at intrinsic characteristics of the attacks.

In this thesis, we first look at this problem from a new angle. Instead of using the intrinsic characteristics of an attack or of comparing the similarity between attacks and target sites, we go back to the source of the problem. We perform an in-depth analysis of how phishing attacks are being built by the attackers. We show that most phishing attacks are duplicates or quasi-duplicates of former attacks. Given that phishing attacks are not built from scratch, we propose two clustering-based methods to evaluate the similarity between attacks. When comparing a newly reported attack against our database of known ones, our method achieves an accuracy of at least 90%, with a false-positive rate of 0.65%.

We then explore the evolution of phishing attacks and track variations over time. Our aim is to better understand what attackers do change, and why, across iterations of the attack. We propose a graph-based model in order to monitor and analyze theses changes and their relations.

In addition to the detection and analysis of phishing attacks on the client-side, we also explore the server-side aspect of phishing. We conduct a static analysis of the source code of "phishing kits" and propose an approach to track stolen information. Since most phishing attacks use email as the means to exfiltrate stolen information, we propose a deep learning model to detect these messages in network traffic. This approach can be used to easily detect that a phishing attack is hosted inside a large network for example.

The third and final contribution of this thesis is a "blind" phishing scanning system, which is used to search for and identify unreported phishing attacks at large scale. The only input of that system is a very large list of domain names. In order to efficiently handle the list, we propose a ranking algorithm which combines natural language processing and machine learning techniques to prioritize the domains that are most likely to be harmful. We then mine our extensive, real-time phishing attack database to guess possible URLs of attacks on these domains and use our own detection algorithm for eventual detection.

**Acknowledgements**

First of all, I would like to thank my supervisor Professor Guy-Vincent Jourdan for his patient guidance and helpful suggestions during my study. My deepest gratitude also goes out to Professor Gregor V. Bochmann and Dr. Viorel Iosif Onut for their advice and feedback through my research.

Many thanks to all the people of my defense committee: Professor Babak Esfandiar, Professor Carlisle Adams, Professor Tet Yeap, and Professor Ali A. Ghorbani for giving valuable feedback on this thesis.

I would like to thank my colleges of Software Security Research Group (SSRG), Sophie Le Page and Emad Badawi. Thanks for your help in writing during my study and the happy time we spent together.

Thanks to the financial support from the IBM Center for Advanced Studies (CAS) and the Natural Sciences and Engineering Research Council of Canada (NSERC). Without these financial support, I can't start my study and life here.

Finally, I want to express my sincere gratitude to my parents and my wife, Zhen Yu for providing me with unfailing support and continuous encouragement.

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

(in alphabetical order)

| | |
|---|---|
| **DOM** | Document Object Model |
| **HTML** | HyperText Markup Language |
| **HTTP** | HyperText Transfer Protocol |
| **LSTM** | Long Short-Term Memory |
| **RNN** | Recurrent Neural Network |
| **SMTP** | Simple Mail Transfer Protocol |
| **SVM** | Support Vector Machines |
| **URL** | Uniform Resource Locator |

# Chapter 1

# Introduction

The Internet provides people and companies with a new way of connecting. This has caused a shift in people's lifestyles, where for example online shopping and online education have begun to replace stores and schools. This has also caused a shift in companies who now focus on creating new Internet products and services in order to dominate the market. However, this shift gives criminals the opportunity to launch new types of crime using computers and networks, known as **Cybercrime**.

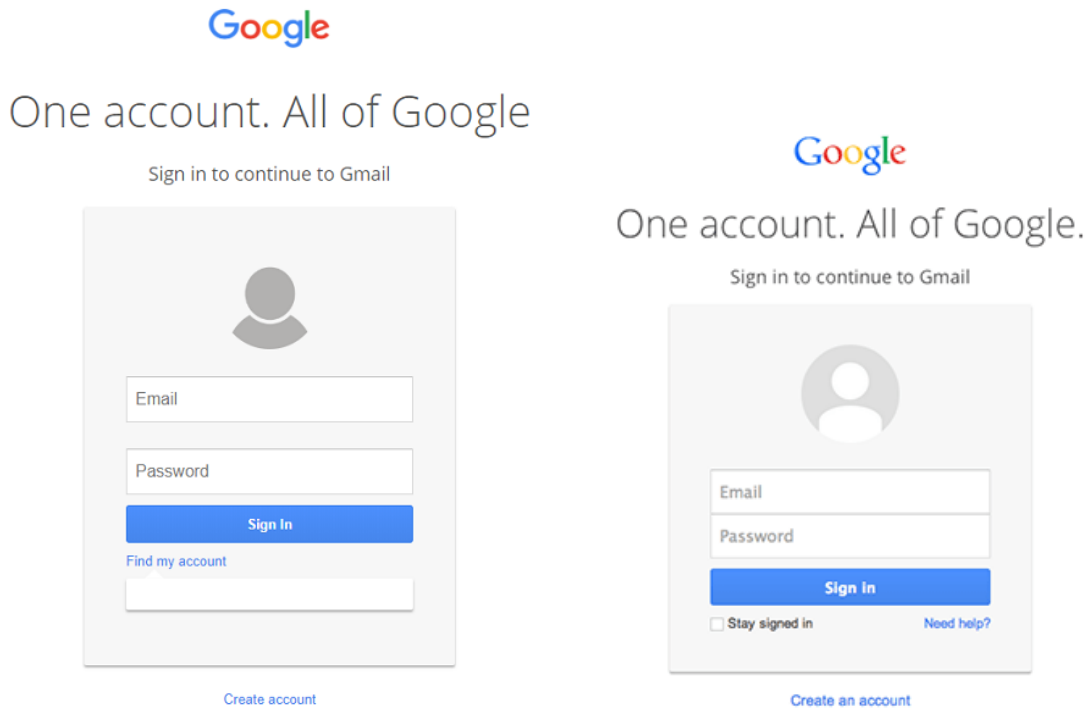
A phishing attack is a type of cybercrime in which people's sensitive information is stolen via a website that imitates another, legitimate website. Despite the tremendous efforts of academia and industry in recent years, anti-phishing research still faces several challenges. In this thesis, we look at the problem from new perspectives. Instead of just detecting and blocking phishing attacks, we focus on exploring the causes and motivations behind the attacks and effectively suppressing the growth of phishing attacks.

In this chapter, we first discuss the definition and risks of phishing attacks (in Section 1.1). We then give an overview of the phishing attack ecosystem, and talk about the business model of phishing attacks in the underground market (in Section 1.2). In Section 1.3, we discuss the lifecycle of phishing attacks, which is followed by the motivation of this research (in Section 1.4) and challenges of phishing detection (in Section 1.5). We finally summarize our contributions and the organization of the thesis (in Section 1.7).

## 1.1 What are Phishing Attacks?

So-called "phishing attacks" are attacks in which phishing sites are disguised as legitimate websites in order to steal sensitive information. It occurs when attackers send out SMS or emails with malicious URLs, tricking victims to open the malicious link, which directs

them into a fake website with highly similar appearance to a legitimate website. Once the victims are tricked into providing their sensitive information, the attacker can use this information to launch social engineering attacks or steal other information associated with the victim's account. Figure 1.1 shows a phishing attack which mimics Gmail login page. The attacker imitates every detail of the legitimate one, such as the logo, the favicon and even the external links, making it hard to be visually distinguished from the legitimate one.




(a) Phishing attack which mimics the Gmail login page

(b) Legitimate Gmail login page

Figure 1.1: An example of phishing attacks which targets Gmail

According to the target range of attacks, phishing attacks could be divided into two categories: spear phishing and clone phishing. The spear phishing is an attack that targets an individual. To increase the probability of success and credibility, the attackers usually gather and use the personal information of some of the members of the organization. For instance, an attacker pretending to be an administrator sends phishing emails to the members. Conversely, the clone phishing, as the main form of phishing attacks, is the attack without the specific selection of attack targets. In clone phishing, in order to make the attack more effective, an attacker continuously improves the obfuscation technology to increase the similarities between the fake site and its target site.

## 1.2 Ecosystem of Phishing Attacks

Sood et al. [58] indicate that in the cybercrime's underground economy, crimeware-as-a-service(CaaS) becomes a prominent model [58], which involves three roles: producers, advertisers and buyers, as shown in the Figure 1.2. First, the producer designs and builds a complete framework of crimeware and then notifies the advertiser. The advertiser then publishes the crimeware on underground forums and Internet Relay Chat (IRC) servers. Finally, a buyer orders the advertised crimeware and uses it to launch cyber attacks. Sometimes the producer and the advertiser may be the same person. In this case, the producer and the buyer trade directly.

In this thesis we will show that phishing attacks follow a similar business model. The producer designs crimeware, known as phishing kits, which makes it easy for people with little knowledge of web technology to launch phishing attacks. These phishing kits flow into the underground market through advertisements in forums and IRC, and are eventually purchased by buyers. In addition to the benefit gained from the buyer's payment, some phishers drop backdoors into the phishing kit in order to obtain stolen information. Specifically, the producer hides his own email address in the code. Once the purchased phishing site successfully deceives a victim to provide sensitive information, a copy of the stolen information sent to the buyer will also be sent to the producer [21, 42].

## 1.3 Lifecycle of Phishing Attacks

In general, the lifecycle of a phishing attack consists of three periods: the preparation period, the active period and the inactive period. A phishing attack starts from the preparation period and becomes active when the phishing link is published in the social networks or sent by email. Once the attack is detected or blocked, it moves to the inactive period. However this does not mean the attack is terminated. The attacker can relaunch the attack after making a series of modifications. We will detail period state in the following section.

### 1.3.1 Preparation Period

During the preparation of a phishing attack, an attacker needs to go through three phases: prepare the phishing server, deploy the phishing attack and publish the phishing attack, as shown in the Figure 1.3.

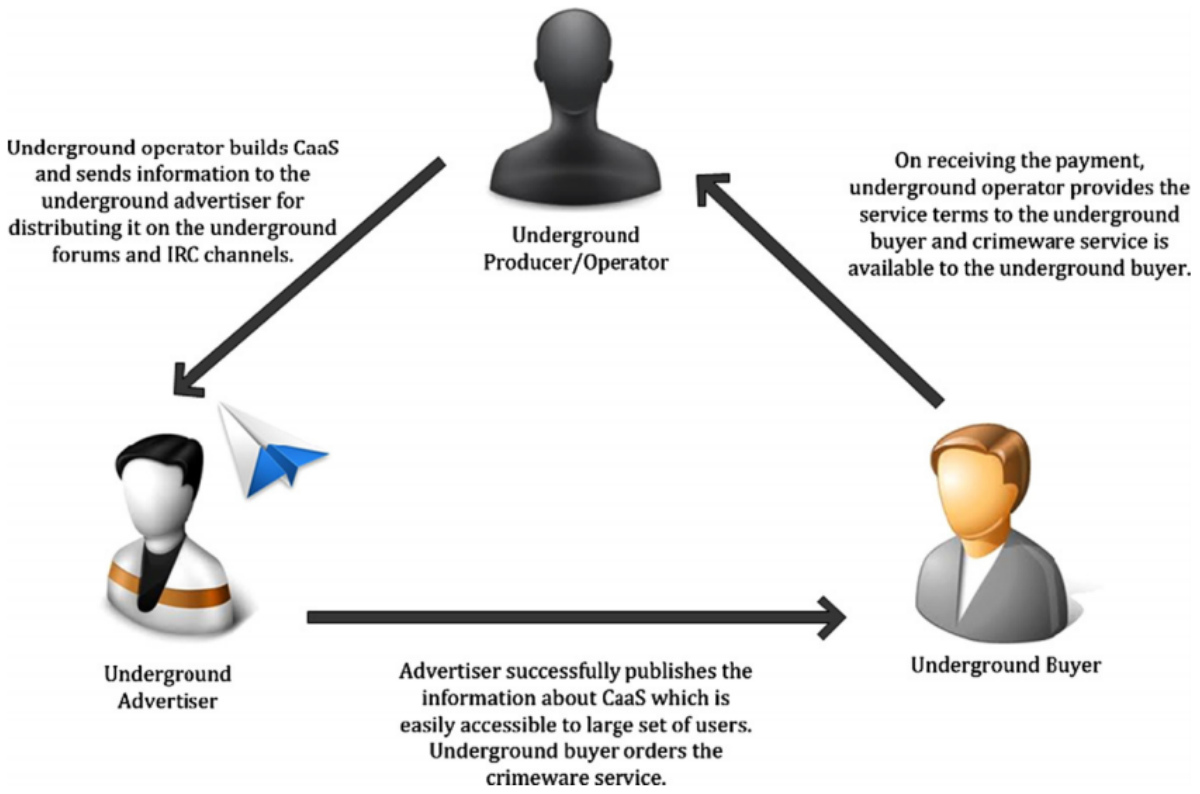



Figure 1.2: CaaS distribution model (taken from [58])

- **Prepare phishing server**. In this phase, an attacker either rents a cloud host (malicious server) or hacks a vulnerable server (compromised server) to use as the phishing server. Malicious servers usually come from cloud hosting providers that offer lower subscription fees or free trials. A compromised server is a legitimate host, such as a personal blog or a commercial website, but is vulnerable because it does not update patches in time or uses some older version of platform. To find vulnerable hosts, an attacker uses various tools to scan their prey on the Internet. One of the approaches called **Evil Searchings** is reported by Moore et al. [46], which query Internet search engines with well designed search terms called "Google Dorks". Google Dorks use advanced search operators of search engines, such as "inurl" or "intitle", to retrieve the vulnerable portals of websites. The attacker then hacks the hosts to gain control and access authority.

- **Deploy phishing attack**. During the phase of the attack deployment, an attacker uploads and deploys the phishing attack on the phishing server, and completes the corresponding configuration in preparation for the attack. For instance, the attacker starts web and SMTP services to enable web access and email delivery. For the at-

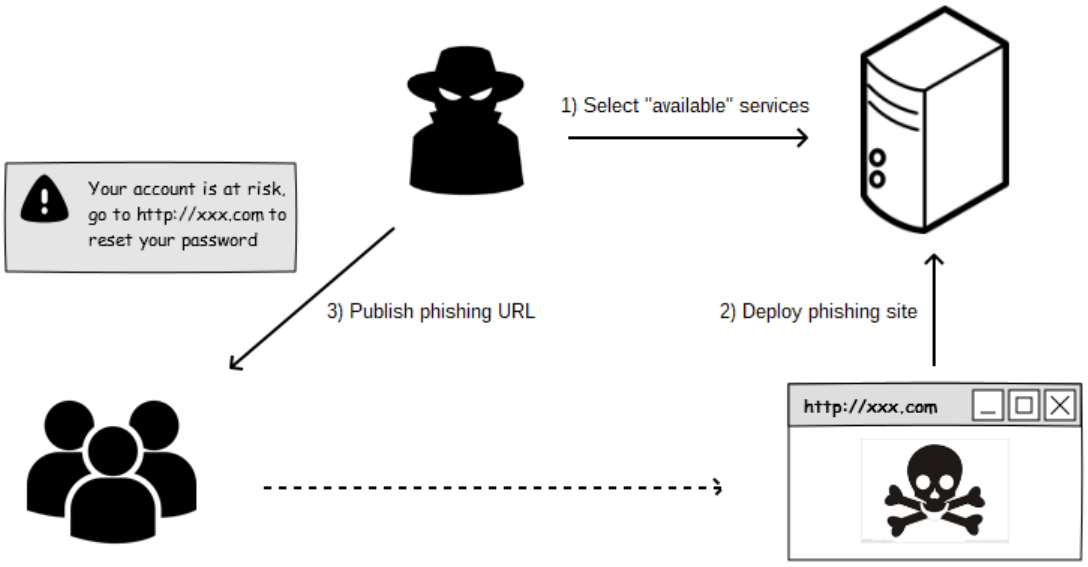



Figure 1.3: Steps to launch phishing attacks

tackers with little web knowledge, they can choose to pay for a specific underground organization, which manages large amounts of phishing servers, to help them complete this phase. The cost of service is based on the scale of deployed servers and the number of phishing attacks being launched.

- **Publish phishing attack**. In order to "publish" the attack, an attacker sends messages with phishing URLs to victims via emails or the social network and tricks them into opening the phishing link. In order to increase the credibility of the link, the attacker uses various obfuscation techniques to hide the actual phishing link. For instance, in a **homograph attack**, the attacker creates a domain name similar to a well-known legitimate domain name by replacing some characters with homographs (e.g. G00gle.com, App0le.com) [27]. URL shortening services, which provide users a shorter URL to represent the original one, is also abused by the attackers to hide the actual phishing link [37].

### 1.3.2 Active Period

At this stage, the victim, who has clicked the phishing links, is directed to the phishing site, and is induced to provide his personal information. Subsequently, the stolen information either becomes a commodity in the underground market or is directly used to steal the victim's property (e.g., stealing the bank account contents). It is worth noting that the

content of each visit to the phishing website may be different. In some advanced phishing attacks, the attacker uses techniques such as one-time access or multiple display pages. This allows the attacker to redirect security company's crawlers or victims who visit the site again to a legitimate site or a dummy page. Han et al. [32] claim that such a approach could help extend the lifespan of attacks.

### 1.3.3 Inactive Period

Once a phishing attack is identified by a vigilant victim or detected by anti-phishing products, its information is synchronized to the network security organization and added to their blacklist. The blacklist is then synchronized to anti-phishing products and browsers to curb the spread of the attack. At this point, the phishing attack has entered an inactive period. However, this does not mean the end of the attack; the attacker can still restart the attack by moving to a different host or changing the URL.

## 1.4 Motivations

Both industry and academia have put a lot of effort into the research of combating phishing and invented various anti-phishing solutions. For example, browsers such as Google Chrome, FireFox and Safari all use **Google Safe Browsing** to provide their users a built-in service to prevent phishing attacks[1]. Microsoft Internet Explorer and Edge browsers also include a similar built-in defence mechanism called **SmartScreen**. However, the growth of phishing attacks has never stopped or even slowed down. According to the report from **Anti Phishing Working Group (APWG)**[2], the number of reported phishing attacks increased by 340% from 2007 to 2017. This indicates that the problem of phishing attacks has not been completely solved. In this thesis, we provide suggestions and solutions for the following problems.

- **Few usable dataset** Although academia has proposed various anti-phishing studies, only a few have published their datasets used in the experiments. Even for those published datasets, only phishing URLs are provided, and these URLs are almost all no longer valid at the time of publication. This makes it difficult for subsequent studies to compare their methods with previous work. As a result, most anti-phishing studies report perfect results under their own dataset, making it hard to assess the effectiveness of these methods.

[1] `https://safebrowsing.google.com/`

[2] `https://www.antiphishing.org/resources/apwg-reports/`

In this thesis, we compile and publish a large phishing dataset, including around 100,000 phishing attacks, that are collected from public and commercial sources. In addition to the URL of phishing attacks, we also provide the DOM and screenshot of phishing pages, that allows other to compare their methods to ours.

- **New research direction needed in detecting phishing attacks** In recent anti-phishing studies, there is a trend that new studies either combine some new features with the features in previous work or mix various features from different work. However, they do not provide any evaluation about the impact of their new features or combination of features in detecting phishing attacks. Therefore, it is hard to evaluate the value of their work.

  In this thesis, we propose several new directions in detecting phishing attacks, including comparing phishing attacks with known attacks rather than target sites, using deep learning to monitor the server traffic, and applying semantic analysis to "blindly" scan early phishing. These new directions provide several opportunities and insights for the fight against phishing.

- **Lack of analysis on phishing evolution** None of the current studies have conducted a comprehensive analysis for the evolution of phishing attacks. The only related study is to analyze the evolution of phishing emails [35]. Understanding the law of phishing evolution plays an important role in the fight against phishing attacks. If we are able to understand the rules of phishing evolution, we can block a series of attacks, not just the attack instances of some specific version.

  In this thesis, we propose a graph-based method to track and analyze the phishing evolution over time. Our results show several interesting characteristics of phishing evolution, that help us better understand the motives and strategies of attackers creating attacks.

## 1.5 Challenges

The current anti-phishing studies are mainly in two directions. The first approach is to find similarities between a phishing site and the legitimate site that it mimics. The second approach is to try to find intrinsic characteristics of phishing sites for training a machine learning model to identify phishing sites and legitimate sites. However, these approaches face a number of challenges in gaining insight into phishing attacks.

- **Track phishing attacks**. Although some public databases, for instance, *PhishTank*[3] and *VirusTotal*[4], provide the long-term historical archive of phishing attacks, they only contain limited information, such as phishing URL and reported date. Moreover, the lifetime of phishing attacks is very short, less than 15 hours on average[5]. Therefore, most published phishing URLs are not active or reachable. For the above reasons, no framework or method is available for long-term monitoring of the evolution and modification of phishing attacks. The current analysis of phishing evolution focuses on the simple statistics, such as the number and the target of attacks, rather than the internal changes.

- **Stop the spread of phishing attacks**. Nowadays, many anti-phishing products use blacklists to curb the spread of phishing attacks. But even after the phishing attack is blocked, an attacker can still relaunch the same phishing attack on another host or with a different URL, which can simply crack the existing URL-based or domain-based blacklists.

- **Detect phishing attacks in advance**. The delay between the initiation and detection of an attack, even only a few hours, still gives attackers a chance to successfully deceive victims. A better solution is required to stop the attack as soon as the attack becomes active.

## 1.6 Research Method and Evaluation

In this thesis, we take the following steps to conduct research and assessments of the research.

- Defining the problem: In Chapter 1, we define the problem and discuss the limitations of current anti-phishing research and solutions. We also explain the current challenges of anti-phishing studies.

- Literature review: We then provide an overview of the existing anti-phishing research in Chapter 2. Through the comparison between different approaches, we perform a comprehensive analysis of advantages and disadvantages of each method and establish background knowledge.

[3] https://www.phishtank.com/

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

[5] https://webroot-cms-cdn.s3.amazonaws.com/7314/8070/2914/Webroot_Threat_Trends_December_2016.pdf

- Proposing approaches and tools: In our research, the goal is to propose new algorithms and tools to analyze the phishing attacks and effectively detect them. We discuss our approaches and tools used to analyze and detect client-side phishing attacks in Chapter 3 and 4. In Chapter 5, we discuss a phishing scanning system which is used to detect potential phishing attacks through "bind" scanning. In Chapter 6, we discuss a method to detect phishing activities by monitoring outgoing email on the server side.

- Experiments: In order to evaluate the performance and applicability of our method, in each chapter, we discuss the experimental results following the introduction of the methods.

## 1.7 Contributions and Organization of the Thesis

In this thesis, our goal is to study the nature of phishing attacks, analyze their internal connections, and then provide effective anti-phishing solutions. We have made the following contributions:

- **A new direction to detect phishing attacks**. The existing anti-phishing approach is either to compare the similarity between attacks and the targets that they imitate or to use specific characteristics of attacks learned from machine learning models. In this thesis, we present a clustering algorithm which groups attacks based on structural similarity. Our clustering result indicates that more than 90% of phishing attacks are not built from scratch. Instead, they are replicas or variations of other attacks, and can be flagged immediately by our approach. What is more, the level of false positives is very low, only 0.65%. This shows that the method presented here is an effective tool to add to a phishing defense strategy.

- **A graphic-based analysis model for monitoring phishing evolution and modifications**. We present a model called *Semi-Complete Linkage (SCL)* to look into the details of phishing evolution over time. Our result indicates that unlike usual software, phishing attacks tend to be derived from a small set of master versions with only a couple of successive iterations being deployed. We also show that different groups of phishing attacks tend to evolve independently from one another, without much cross-coordination. Our findings will further enhance the understanding of the phishing ecosystem and it will help with combating the problem more effectively.

- **A "blind" phishing scanning system**. To detect the potential or early attacks we propose a "blind" phishing scanning system which only uses a list of domains as input. So-called "blind" means the system scans the domain without knowing any accessible URL paths. We propose a domain ranking model which can figure out potential phishing domains by analyzing the semantics of a domain name. We show that such an approach could effectively detect early attacks which greatly narrows down the lifespan of attacks. We have implemented our proposed system in a real-time scanning tool for IBM's cloud service.

- **Phishing detection mechanisms on the server side**. By analyzing the source code of phishing kits, we propose a deep learning algorithm for detecting phishing emails which include the sensitive data of victims. The results show that our method can detect such phishing emails with more than 98% accuracy and with only 0.3% false positives.

### 1.7.1 Publications

We have published three papers out of this research:

- Cui, Qian, Guy-Vincent Jourdan, Gregor V. Bochmann, Russell Couturier, and Iosif-Viorel Onut. *Tracking phishing attacks over time*, in Proceedings of the 26th International Conference on World Wide Web (WWW'17), pages 667-676, 2017.

- Cui, Qian, Guy-Vincent Jourdan, Gregor V. Bochmann, Iosif-Viorel Onut, and Jason Flood. *Phishing Attacks Modifications and Evolutions*, in European Symposium on Research in Computer Security (ESORICS'18), pages 243-262, 2018.

- LePage, S., Qian Cui, Guy-Vincent Jourdan, Gregor V. Bochmann, Jason Flood, and Iosif-Viorel Onut. *Using AP-TED to Detect Phishing Attack Variations*, in Proceedings of the 16th conference on Privacy, Security and Trust (PST 2018), 6 pages, 2018 (short paper).

We have also finished two additional conference papers and a journal paper. We also have three patents that have been filed.

### 1.7.2 Organization

The rest of this thesis is structured as follows:

- In Chapter 2, we present an overview of the existing anti-phishing studies. We compare the similarities and differences between these methods and analyze the limitations and problems of these methods.

- In Chapter 3, we first introduce a clustering based algorithm to detect phishing variations. We then discuss several improvements to this algorithm. At the end of this chapter, we present the experimental results of our model's effectiveness in detecting phishing attacks and the performance comparison between the improved algorithm and the original algorithm.

- In Chapter 4, we discuss our graph-based model used to track and monitor phishing attack evolution and modifications. We also introduce several new metrics to evaluate the changes between attacks, followed by interesting findings regarding the phishing attacks.

- In Chapter 5, we propose a blind scanning system to detect phishing attacks at an early stage. The system uses domain names as input and prioritizes the domains which are most likely hosting phishing sites based on semantic features. To assess the effectiveness of our system in practice, we apply our system to large-scale domain logs. Our results are listed and discussed at the end of this chapter.

- In Chapter 6, we describe three machine learning-based models for detecting phishing emails sent to victims. In order to test the robustness of our model, we discuss two possible attacks to crack our model, and demonstrate the performance of our model against these attacks.

# Chapter 2

# Literature Review

## 2.1 Introduction

There is a significant body of academic work focusing on the automatic detection and analysis of phishing attacks, mainly in three directions: phishing sites detection, phishing email detection and analysis of phishing kits. Figure 2.1 shows the roadmap of the anti-phishing studies. In the following sections, we present an overview of these studies.

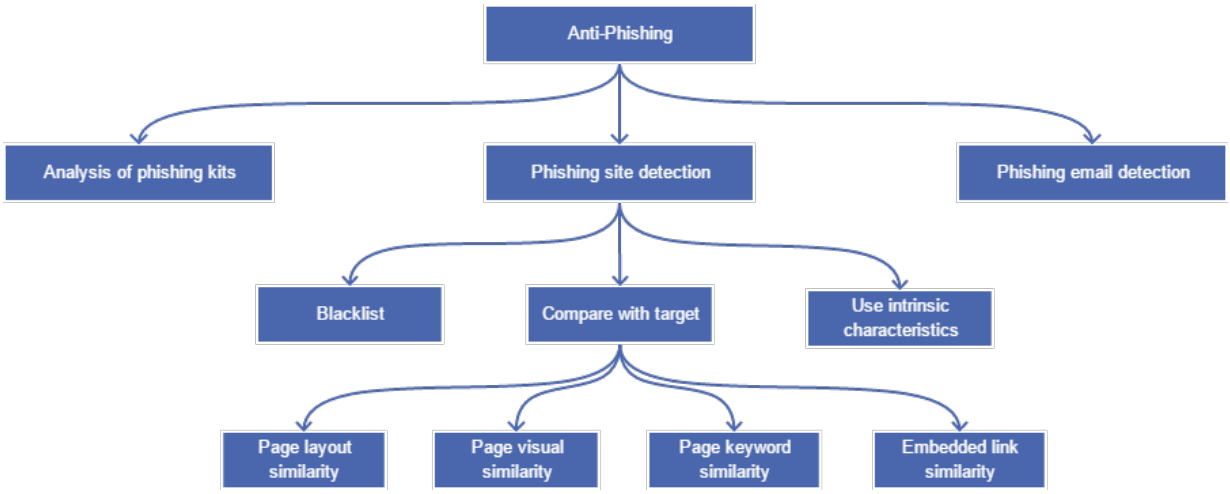



Figure 2.1: Roadmap of anti-phishing research

## 2.2 Phishing Site Detection

As discussed in Section 1.3, since a phishing attack only works when victims are induced to the phishing site, a common and intuitive defense strategy is to identify and block these

malicious sites. Most published literature in this direction falls into one of the following categories: the phishing detection by using blacklists, the phishing detection by comparing with targets, and the phishing detection by looking at intrinsic characteristics.

### 2.2.1 Phishing Detection by Using Blacklists

A blacklist is an access control mechanism which blocks the listed elements to access the protected resources. To prevent phishing attacks, major browsers have integrated dynamic blacklists. For instance, Firefox, Safari, and Chrome all have Google's blacklist, **Google Safe Browsing** to provide users with protection against phishing attacks. In this section, we discuss how phishing blacklists work and the problems blacklists face.

#### How do Phishing Blacklists Work?

Compiling and publishing a phishing blacklist requires three steps[1], as shown in Figure 2.2.

- **URL collection**. In this step, a blacklist provider collects phishing URLs from various data sources. It includes the detected phishing pages or emails from Internet security companies, as well as the attacks reported to community-based systems (e.g. PhishTank).
- **URL verification**. In order to reduce false positives, the collected URLs may need to be manually verified. Reviewers flag the collected URLs by checking additional information such as domain reputation and page archive[2].
- **Blacklist updating**. Once the verified URL is confirmed as phishing, it is added to the central database and then synced to client browsers and to the partners that have subscribed to the blacklist source.

#### Ineffectiveness of Phishing Blacklists

In practice, the issue with phishing blacklists is that they cannot prevent phishing attacks in real time. In other words, there is a delay between when the phishing attack is initiated and when it is added to the blacklist. Sheng et al. [56] report that 47%-83% of phishing attacks were blacklisted after 12 hours. The results that Han et al. [32] reported are even

[1] `http://phishtank.com/faq.php#whatisphishtank`
[2] `https://archive.org`

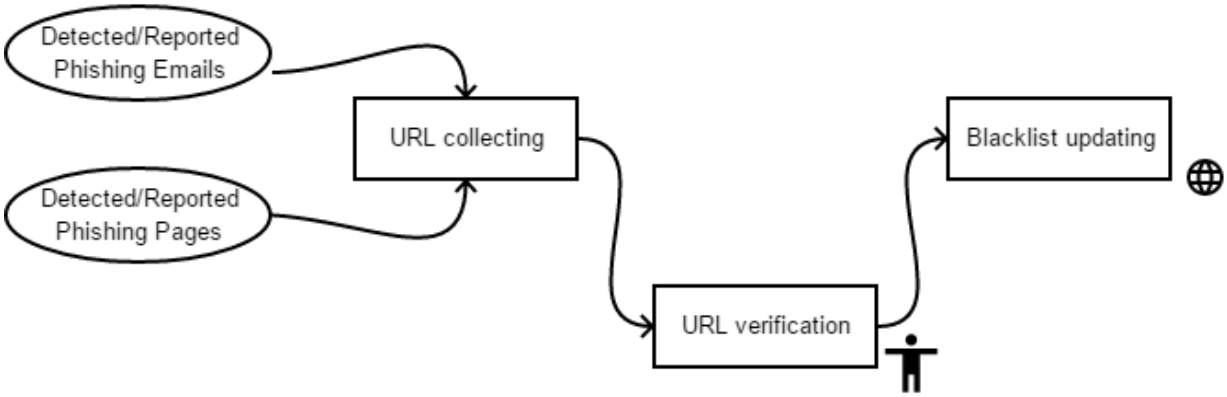



Figure 2.2: Steps to compile and publish a phishing blacklist

worse: most phishing attacks (62%) were blacklisted on average 20 days after installation. This delay is caused by each step in the compiling and publishing process, as shown in Figure 2.2. For instance, when the blacklist provider collects the phishing URL, the attack may have been active for a while. Since the verification and update processes usually take more time, the latency is even greater.

The delay in phishing blacklists gives attackers time to spoof victims and collect the stolen information. Therefore, blacklists cannot effectively prevent phishing attacks and instead can only block the attacks that have been active for a while.

### 2.2.2 Phishing Detection by Comparing with Target

The essence of phishing attacks is, through mimicking a legitimate site, attackers are able to trick the victims into entering their sensitive information. Therefore, if a site is similar to another legitimate site, it is likely to be a phishing site. From this perspective, some methods suggest comparing similarities between phishing sites and legitimate sites to identify phishing attacks. These studies focus on tackling two major challenges. One of the challenges is to identify the phishing target. Another challenge is to choose appropriate metrics to measure the similarity.

#### Page Layout Similarity

The DOM is a programming interface of a HTML page, which represents each component of the page as a node in a tree structure. The association between components is represented as the connection between the parent node and child node. Thus, some studies attempt to use DOM to analyze the similarity of the page layout.

Rosiello et al. [53] present a browser extension using the similarity of the DOM tree to detect phishing attacks. They first store a mapping between the user's sensitive information and the legitimate sites that use the information. When the same sensitive information is reused on a different site, the extension checks whether the DOM of the new site is similar to the associated one. If two DOMs are similar, the new site is reported as phishing. In their experiment, they only report the false positives and false negative under different thresholds, but they do not mention how many legitimate sites and phishing sites are used for testing, which makes it hard to assess the effectiveness of their method. Another problem in this study is that the detection condition "similar DOM" can be simply cracked by attackers in practice. For instance, as shown in Figure 2.3, some websites may have multiple entries of the login page. The left one is a common Facebook login page used by most users. On the right it is another login entry. Thus, the browser extension may only store the DOM of the common entry. Therefore if an attacker creates an attack that mimics the login entry on the right, the suggested method will fail. The attacker can also build phishing attacks that have a similar look but consists of a completely different DOM. For example, an attacker may store most components as images and embed them as the background of the phishing page.

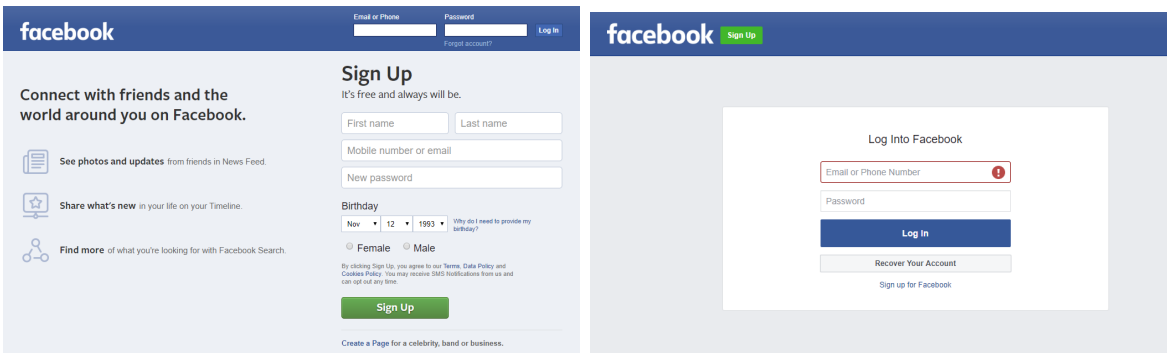



Figure 2.3: Multiple entries of Facebook

To overcome this problem, Zhang et al. [66] suggest using the spatial layout features to compare page layout similarity. They first split the page into several components and then extract the spatial features of each component such as position and size. These spatial features could help to catch the pages with partially similar components. To detect a phishing page, spatial features of suspicious page are compared with a "benign" dataset which consists of the spatial features fetched from legitimate sites. If the similarity of spatial features is beyond a predefined threshold, it is identified as phishing. They report a result of 93.3% precision and 91.9% recall on a dataset of 100 phishing sites and 100 legitimate sites. Although this approach enhances the metric used for page layout comparison and does not require user's sensitive information to retrieve the target site, it still suffers from

the issue similar to Rosiello's work: it is hard to detect the phishing page that uses most of the components of the target site as background images.

### Page Visual Similarity

Since using page layout similarities is not effective for phishing detection, some studies attempt to apply visual comparison technologies to find similarities between phishing attacks and the target sites.

Dhamija et al. [24] propose a scheme, *Dynamic Security Skins*, which prevents the attacker to mimic the site by embedding a visual identity in the page. The goal of this method is to set up a security certificate between the user and the trustable website, which is hard for any third party to mimic. In other words, the user and the trustable website share a secret as the proof of secure communication, which cannot be predicted or known by any third party. Specifically, the user first sends a security token to the trustable website, and then the server generates a hash based on this token using a hash function known to both sides. This hash value is later verified by comparing with the client-calculated hash value. To simplify the verification process, they suggest a technology called *visual hash*, a visual representation of a hash value. The page returned from the trusted server uses this visual hash as the background image, as shown in Figure 2.4. If the page being verified does not include this visual hash or contains an unmatched hash, it is identified as phishing.


(a) A generated visual hash

(b) Verified page tiled by visual hash

Figure 2.4: An example of webpage protected by visual hash [24]

One problem with this study is that the authors do not apply any formal experiment on their method. Instead, they only discuss a number of designs for future tests. Therefore, it is hard for others to evaluate their method. What is more, this method has several

limitations that make it difficult to apply in practice. First, it requires the additional server-side modification to support the verification. What is more, it requires several human interventions to complete the verification. The user is not only risking leaking the security token (consider if the user submits the security token to the phishing site), but is also responsible for identifying the security indicator (visual hash).

In order to reduce the user's intervention, some studies suggest using a predefined collection of benign websites [26, 17, 29] or identifying phishing target sites through analyzing the visual information [15]. These studies focus on looking for the proper visual features and similarity metrics, which can represent the similarity of the phishing site to the target site.

Fu et al. [26] suggest a method using Earth Movers Distance (EMD) and visual signatures of the low resolution page screenshot to evaluate visual similarities. They normalize the page screenshot into a fixed-size square image (10*10). A feature vector is proposed to represent the visual features of the scaled image, which consists of the centroid of the position distribution and the frequency of each color. To find an optimal EMD threshold to identify phishing attacks, they conduct experiments on a verification set and select the threshold that yields the best detection result. If the EMD distance between the suspicious page and any of the sites in the protected site collection is less than this optimal threshold, it is marked as phishing. They report a result of 99.87% accuracy on a very unbalanced dataset, which consists of 10,272 legitimate pages but only 9 phishing pages. However, by compressing the page screenshot to such a small thumbnail, most of the information on the page will be lost. Therefore, if the model is applied to a large phishing dataset, there may be many false positives.

Chen et al. [17] present a straightforward method to fetch the visual features of the page screenshot. In their work, they apply a common compressor such as gzip or bzip2 to compress the page screenshot. Normalized Compression Distance (NCD) is used to measure the similarity between the compressed images. A result of 95% accuracy and less than 1.7% false positives is reported on a dataset of 120 legitimate pages and 320 phishing sites. The problem with this approach is that the detection performance highly depends on the choice of the compressor, but the authors do not give clear answers about how to choose an "optimal" compressor.

Rather than using the compressed visual feature of the screenshot, some studies report good results by using partial visual information as the detection identifier, such as the favicon [29]. In their work, the authors first fetch the favicon from the DOM of the suspicious page. It is then compared with the favicon and the logo of the phishing target sites. To reduce false positives, the domain information and Page Rank are suggested

as additional indicators for identifying phishing attacks. They report a result of 99.5% true positives and 0.15% false positives on a dataset of 3,642 phishing pages and 19,585 legitimate pages. However, this approach only works for the attacks using favicons.

### Page Keyword Similarity

We can see that all the above methods require manual collection of potential phishing targets due to the lack of ability to automatically identify the phishing target. In order to overcome this limitation, some studies apply the search engine to retrieve the phishing targets. The goal of this research is to extract the keywords from the suspicious page, and query them in a search engine. If the domain of a suspicious page does not match any of the pages indexed by the search engine, it is identified as phishing.

Zhang et al. [67] propose a text-based method "Cantina" using the **Term Frequency and Inverse Document Frequency (TF-IDF)** to identify the page keywords. The keywords extracted by the TF-IDF algorithm are then submitted to the search engine to query pages containing similar keywords. They report 89% true positives and 1% false positives on a database of 100 phishing sites and 100 legitimate sites.

Xiang et al. [64] improve "Cantina" by combining natural language processing techniques. In their work, they suggest extracting the keywords from the page title and copyright text that are more representative of the target brand. To effectively extract the brand name, they define 11 regular expressions targeting different variants of copyright formats. To cover the specific case where the brand name is not included in the title or copyright but exists in other DOM objects, they apply *Named Entity Recognition* to locate the brand name in the text content. Named Entity Recognition is a classification task under the machine learning framework, which is seeking and grouping the entity names in unstructured text into some predefined categories, such as persons and organizations. In this work, a Stanford Named Entity Recognizer is used to identify website brand names. The proposed approach achieves a result of 90.06% true positives and 1.95% false positives over a dataset with 7,906 phishing pages and 3,543 legitimate pages.

Instead of extracting keywords from the text content, some studies use other features to fetch the page keywords. Chang et al. [15] present a solution using the site logo. They first use the image segmentation technique to extract the website logo from the page being checked. Then they use Google Image Search to retrieve keywords related to the site logo. They are able to obtain a true positive rate of 95.8% and a zero false positives rate using a dataset of 400 phishing pages and 50 legitimate pages. The disadvantage of this method is that the detection performance mainly depends on the image recognition ability of the

search engine and the image segmentation quality of the site logo. In their experiment, they show that the improper logo segmentation greatly reduces the detection accuracy to below 40%.

#### Embedded Link Similarity

Another direction to find similarity between phishing attacks and target sites is to check the hyperlinks embedded in the page.

Liu et al. [39] suggest a graph-based method using embedded links to identify the phishing target. Given a suspicious page, they first construct a set of associated pages called "parasitic community", which consists of all pages reached by the embedded links (direct links) and the pages with the similar keywords obtained by querying the search engine (indirect links). A crawler is then sent to build connections for the parasitic community. Specifically, the crawler examines the embedded links of each page in the parasitic community, and adds a directed edge if a page is reached from another page through the embedded link. Finally, a directed graph is created between the suspicious page and its parasitic community. The page with the strongest connection to the suspicious page is identified as the phishing target. In their work, 99.67% true positives and 0.5% false positives are reported on a database of 3,374 phishing sites and 1,200 legitimate sites.

Ramesh et al. [52] review the hyperlink connection between phishing sites and target sites from another perspective. Instead of retrieving the site with strongest connection to the suspicious page, they look at the intersection between direct and indirect links. Then, the IP addresses of the common links are compared with the suspicious page. If the suspicious page does not match any IP, it is flagged as phishing. They report an accuracy rate of 98.95% and a false positives rate of 1.2% on a database of 10,005 phishing sites and 1,000 legitimate sites.

### 2.2.3 Phishing Detection by Looking at Intrinsic Characteristics

With the widespread use of machine learning techniques, some studies attempt to use machine learning to discover the intrinsic characteristics of phishing attacks. The goal of these approaches is to train a binary classifier by learning the relations between data features and the ground truth (phishing or legitimate). The prediction performance depends on two factors: feature selection and model selection. In this section, we mainly discuss these two factors.

### Machine Learning Models Used for Phishing Page Detection

We first introduce two common errors that cause the poor performance of machine learning models: **overfitting** and **underfitting**. Overfitting is the error that the model fits too closely to the training data and fails on the new data. In other words, the model has the perfect performance on the training set but performs relatively poorly on the test set. Overfitting occurs when overly complex models are used to explain a dataset with simple relations among features. On the other hand, underfitting is the error that the model cannot fit the training set well due to using an excessively simple model without enough parameters to represent the data. Different machine learning algorithms have different sensitivity to these two errors.

Although there is a significant body of machine learning algorithms proposed in recent years, only seven algorithms are widely used in phishing detection. Since each algorithm has its own advantages and disadvantages, the performance of different data sets and feature sets is also different. In the rest of this section, we briefly discuss these algorithms.

- **Logistic regression**. It is a machine learning model that applies a logistic function (usually sigmoid function) to predict the probability of the linear combination of multiple variables in categories "1". A simple logistic regression model is defined by the following formula.

  $$y = sigmoid(Wx + B), \text{ where } sigmoid(Z) = \frac{1}{1 + e^{-z}}$$

  $W$ and $B$ are the parameter vectors that are learned from data, and $x$ and $y$ represent the input and the output. However logistic regression suffers from the *Complete Separation* problem when the target variable completely separates a predictor variable [4]. This will causes the model to do the estimation without using other predictor variables.

- **Decision tree**. It is a tree-like hierarchical structure model which learns the data features through calculating the contribution of each feature to the final decision. In a decision tree, as shown in Figure 2.5, each node (feature) and the connected edges (conditions) form a rule that subdivides the data into several subsets. This process loops recursively until the data ends in a specific category. Thus each path from root node to leaf node conducts a set of decision rules which classifies the data into one category. Compared with the other machine learning algorithms, decision trees are much easier to interpret, but they suffer from overfitting issues when there are too many branches in the tree.

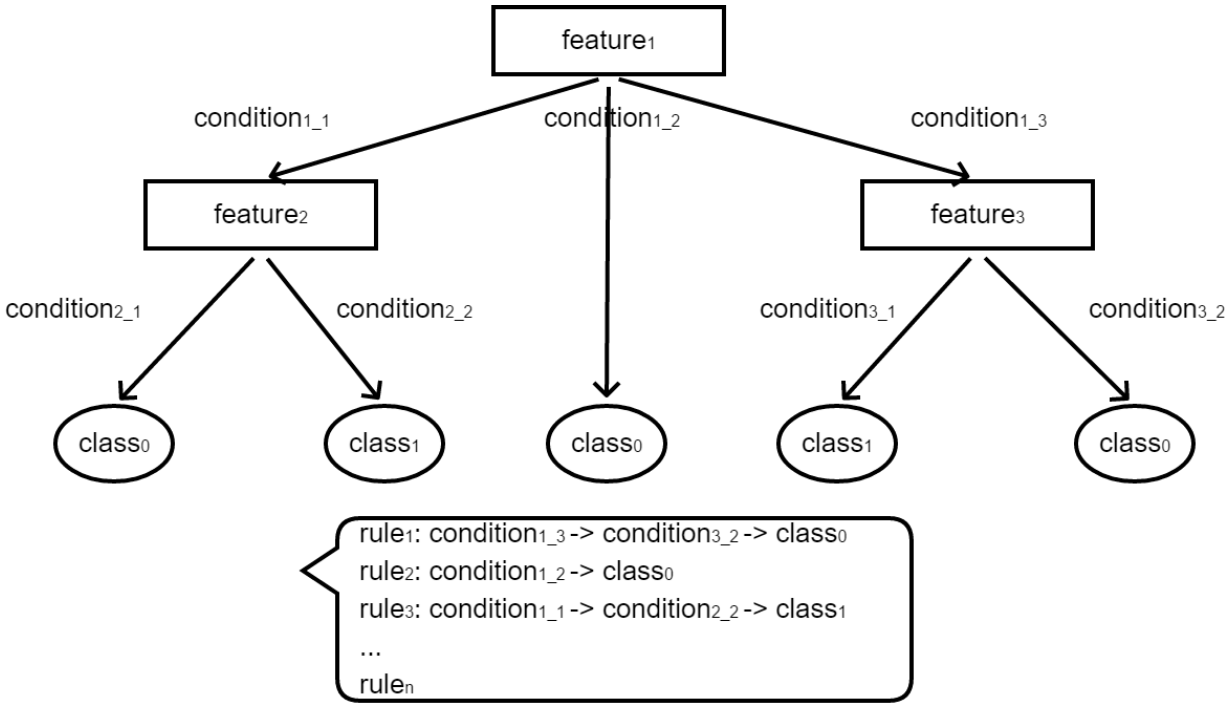



Figure 2.5: An example of decision tree

- **Random forest**. It is a model that constructs multiple decision trees during the training process. Each decision tree uses a random subset of features. The model output is the combined result of the individual trees such as the majority voting or the mean value. Random forest can directly handle various types of data such as categorical features and numerical features without data preprocessing. Since it uses a random sampling of features and multiple decision trees, random forest does not overfit as long as enough trees are applied in the forest. The problem with this algorithm is that the prediction process is slow when there are too many trees in the forest.

- **Support vector machine (SVM)**. The goal of the SVM algorithm is to find a hyperplane which is able to separate the data into two categories. Figure 2.6 shows how SVM works in binary classification: It first converts the input data into high dimensional space using a kernel function, and then searches the hyperplane with the maximum distance to the nearest points on each side. The problem with SVM is that the choice of a meaningful kernel function depends on rich domain knowledge, and there is no general kernel function that can work well for all problems.

- **Naive Bayes model**. It applies Bayes' theorem to predict the probability of an event occurring. Specifically, the model learns the conditional probability and inde-

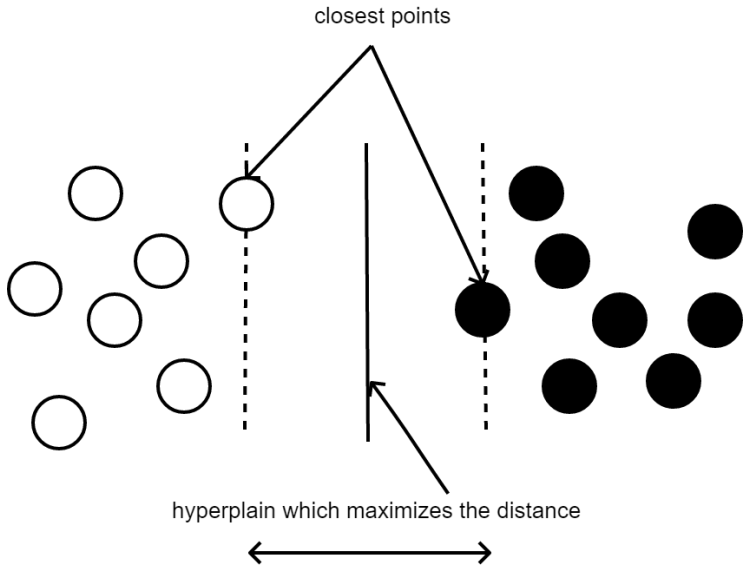



Figure 2.6: An example of SVM

pendent probability of each feature from the training set and uses them to predict the probability that a new event belongs to the category. The Naive Bayes model is fast for training and suitable for processing large data sets, but its premise is that all the features must be independent, which is difficult to meet in practice.

- **Artificial neural network (ANN)**. ANN can be deemed as a combination of multiple logistic regression models, which consists of three parts: the input, the output and the hidden layers, as shown in Figure 2.7. The core part of ANNs is the hidden layers, which include multiple layers of cells (sigmoid functions) with full connection to the nearby layers. The problem with ANNs is that adjusting hyperparameters (such as the number of hidden layers and the number of cells) is a cumbersome task. Currently, there are no good theoretical frameworks, and instead a thorough search of the hyperparameter space must be done.

- **AdaBoost**. AdaBoost is a machine learning framework, which combines the results of multiple weak learners (for example, decision trees) into a weighted output to improve the performance of the weak learners. The problem with this model is that, to achieve high classification accuracy, it needs to combine multiple weak learners, which makes it very expensive to compute in practice.

Table 2.1 shows a summary of the studies which apply machine learning algorithms to detect phishing pages. Works [30] and [63] have achieved the best two results by applying SVM and Bayesian network. However, it does not mean that these two algorithms

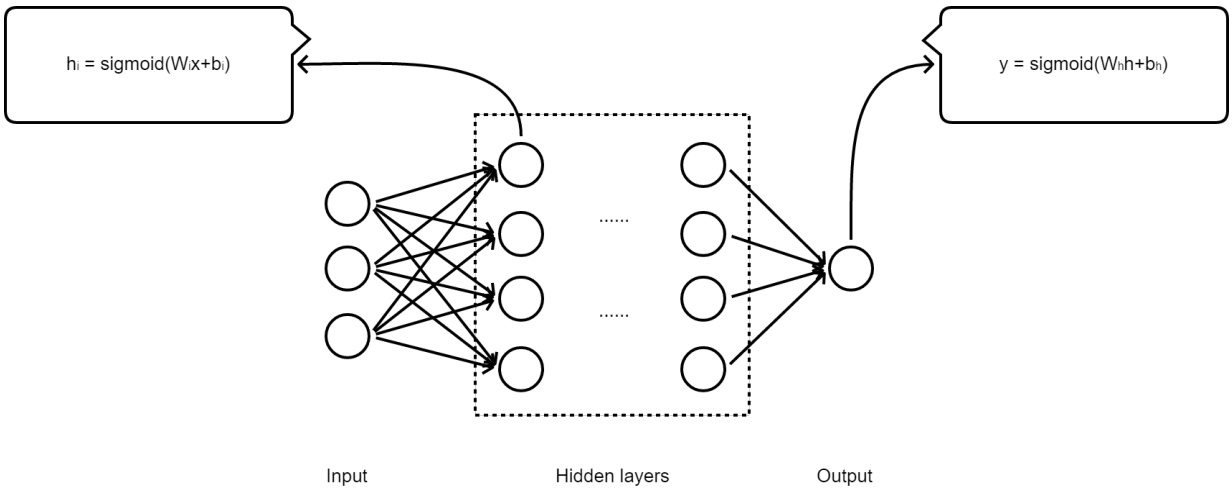



Figure 2.7: An example of ANN

are better than the others due to using different feature sets and datasets. Algorithm comparison under the same conditions are only performed in works [44, 63]; however these two studies give different conclusions. Xiang et al. [63] argue that the Bayesian network classifier defeats all the other classifiers in their experiments, while Miyamoto et al. [44] indicate that AdaBoost yields the best results. It can been seen that due to the lack of common datasets and feature sets, it is hard to conclude which machine learning algorithm is most suitable for phishing page detection. Therefore, algorithms should be compared in the model selection process, but most studies lack this necessary step.

| **Work** | **Machine learning algorithm** | **# of features** | **Result** | **Dataset** |
|---|---|---|---|---|
| [40] | Decision tree | 18 | 83.09% TP and 0.43% FP | 680 phishing pages and 4,149 legitimate pages |
| [63] | AdaBoost, SVM, Random forest, Bayesian network and ... (totally six algorithms) | 15 | 99.63% TP and 0.41% FP | 8,118 phishing pages and 4,883 legitimate pages |
| [30] | SVM | 15 | 99.65% TP and 0.42% FP | 1,764 phishing pages and 700 legitimate pages |
| [45] | ANN | 17 | 92.18% accuracy | 800 phishing pages and 600 legitimate pages |
| [49] | SVM | 7 | 84% accuracy | 279 phishing pages and 100 legitimate pages |
| [44] | AdaBoost, SVM, Random forest, ANN and ... (totally nine algorithms) | 9 | 86.14% TP and 13.83% FP | 1,500 phishing pages and 1,500 legitimate pages |

Table 2.1: Machine learning algorithms used for phishing page detection

### Features Used for Phishing Page Detection

Feature engineering also plays a crucial role in machine learning. Many studies attempt various features to improve the performance of phishing detection. In this section, we talk about the features used for machine learning based phishing page classifier. Table 2.2 lists a summary of these features.

| **Category** | **Feature** | **Work** | **Description** |
|---|---|---|---|
| URL based features | IP Host | [28, 63, 45, 44] | Check whether hostname is an IP address |
| | URL contains the brand name | [40, 63, 45, 44] | Check whether the URL contains the brand name |
| | Number of dots in URL | [63, 45, 44] | Count the number of . that appear in URL |
| | Suspicious symbols in domain | [63, 45] | Check whether the domain contains some suspicious symbols such as *at(@), dash(−)* |
| | Incorrect position of top level domain (TLD) or brand name | [63] | Check whether the TLD or brand name appears in the incorrect position |
| | URL contains sensitive words | [28, 63] | Check whether URL includes sensitive words (e.g. login and sign) |
| | Lexical property of URL | [36] | Count the frequency of the specific string token appearing in URL |
| | Length of URL | [45] | Count the number of characters in URL |
| HTML element based features | Web identity | [49, 45] | Check whether the specific page components match with the web identity claimed in the page |
| | Number of forms | [40] | Count the number of form elements in the DOM |
| | Number of embedded links | [40] | Count the number of embedded links in the DOM |
| | Number of input elements | [40] | Count the number of input fields, text fields and password fields. |
| | Number of Javascript | [40] | Count the number of Javascript tags/blocks |
| | Number of redirection | [45] | Count the times of redirection |
| | Bad embedded link | [49, 63, 45, 44] | Check whether embedded links are empty or point to a different domain |
| | Bad form handler | [49, 63, 45] | Check whether form submission URL is empty or points to a different domain |
| | Bad forms | [63] | Check whether the page contains harmful forms |
| | Pop-up windows | [45] | Check whether pop-up windows are used |
| | Hidden link | [45] | Check whether obfuscation techniques to hide embedded links are used |
| | Right click disabled | [45] | Check whether the mouse right click is disabled |
| Web based features | Page information | [28, 63, 45, 44] | Check the page related information (e.g the search results returned by search engine and page rank) |
| | Domain information | [28, 63, 45, 44] | Check the domain related information (e.g. age of domain, and domain rank, DNS record) |
| Security based feature | Incorrect use of SSL | [40, 45] | Check whether the page URL is enhanced by SSL or uses a trusted certificate |

Table 2.2: Features used for phishing page detection

- **URL based features**. Since the phishing attack tends to obfuscate the URL as the legitimate one, some studies attempt to explore the features in the URL to identify phishing attacks.

  Garera et al. [28] study the characteristics of phishing URLs. In their study, the authors analyze three obfuscation techniques commonly used in phishing URLs.

1. **Obfuscating the host with an IP address**. To hide the true phishing domain, some attacks use the IP address as the hostname (e.g. `http://37.110.23.110/login`).
2. **Obfuscating the host with another domain**. In this category, the attack is hosted on a domain that appears to be normal, but the URL path contains the target brand name, such as `http://2-mad.com/hsbc.co.uk/index.html`.
3. **Obfuscating with a long host name with target brand**. This type of attack uses the target domain as part of the domain, for instance, `http://www.volksbank.de.custsupportref1007.dllconf.info/`.

Their results indicate that most phishing attacks tend to use IP addresses or long host names embedded with legitimate brand names. In order to capture this behavior, many URL-based features are proposed in subsequent studies.

For instance, Xiang et al. [63] suggest six URL based features to detect the abnormal URL: 1)URL uses the IP address as the hostname. 2) URL contains the brand name. To detect brand names in URLs, they apply a regular expression to extract letters and numbers from the domain name, and then compare the obtained string to a list of pre-compiled brand names. 3) Number of dots in URLs. 4) Incorrect position of the top level domain (TLD) or the brand name. Some phishing URLs directly use the legitimate hostname as the subdomain. Consequently, more dots are used in these URLs than legitimate ones, and the TLD or the brand name appears in an abnormal location in the domain name (e.g. `http://netflix.com-securebillingupdate.sign.newandsupport.com`). 5) Suspicious symbols (e.g. @ and $-$) appear in the domain name. 6) URL contains sensitive words. Garera et al. [28] find that phishing attacks tend to build URLs using a limited set of words. They compile a list of sensitive words that typically appear in phishing URLs, such as login and signin. Xiang et al. then used the number of these sensitive words in the URL as the identifier of the phishing attack. In addition to the above six features, in work [45], the length of the URL is also suggested as a URL based feature for phishing detection since phishing URLs are usually longer than the legitimate one.

Khonji et al. [36] present a lexical feature of phishing URLs. In this work, they first extract the string token from the URL. A string token is defined as a case-sensitive string consisting of characters A-Z, a-z, 0-9, underscore (_) and hyphen (-). Any other characters are used as the token separators. They then count the number of each token that appears in phishing URLs and legitimate URLs, noted as $N_p$ and $N_l$. The phishing probability estimate for a string token is defined as $\frac{N_p}{N_p+N_l}$. Finally, the probability that the URL is detected as phishing is the probability average of all

tokens in the URL.

- **HTML element based features**. Another category of features for detecting phishing attacks is based on HTML elements in the page. Ludl et al. [40] conclude that phishing attacks tend to use more specific HTML elements, such as forms, embedded links, input elements, and Javascript, than legitimate websites. They count the number of these elements that appear in the DOM and use the frequency of each element as a numeric feature.

  Pan et al. [49] compare the HTML elements with the **Web Identity**, which is an abstraction of the page ownership. In order to fetch the web identity, they first extract words from page content and rank them according to their occurrence frequency. The word with the highest frequency is selected as the web identity. They then check if the web identity appears in the URL, form handler or cookie. Binary features are generated from each comparison and eventually combined together as a feature vector.

  Xiang et al. [63] look at the characteristics of the form element in the phishing page. First, they find that many phishing attacks submit forms to different domains. In addition, phishing attacks rarely use https to encrypt submission requests. Last but not least, the structure of the phishing form elements is very simple, only consisting of input elements and texts (e.g. username and password).

  Mohammad et al. [45] analyze the web technologies used in the phishing page. They report that a number of attacks use pop-ups to submit credentials. Multiple redirects or **onMouseOver** is abused to hide real phishing links. They also report that approximately 1.6% of phishing attacks disable the right-click feature to prevent users from viewing the DOM of phishing pages in the browser.

- **Web based features**. Web based features are the features that are collected from the Internet rather than the page itself, such as the age of the domain, the page rank and the domain rank. Since the lifespan of phishing attacks is very short, the age of the domain is far less than the legitimate one and the page rank is lower. Some studies also apply the method of retrieving related domain names by keywords (Section 2.2.2) to the machine learning based approaches [63, 45, 44]. Specifically, a binary feature is set to 1 if the domain being checked does not match any of the pages returned by the search engine with similar keywords, otherwise it is set to 0.

- **Security based feature**. Since most legitimate sites, specifically e-commerce websites, use SSL to enhance the website security, some studies [40, 45] suggest that if a

site does not use SSL or the SSL certificate is not issued by a trusted company, it is suspicious.

### 2.2.4 The Problem with Studies on Phishing Site Detection

One common problem with the above studies is that most studies do not publish the dataset used in their experiments, which makes it impossible for others to reproduce their results and compare them to their own methods. Although some studies conduct the comparison with previous work, comparisons under different datasets have no meaning for evaluating the validity of the model because the characteristics of attacks may change over time. Another problem is that these studies do not discuss how they preprocess their database. For instance, most studies use PhishTank's archives as the phishing database, while in our experiments we find multiple duplicates and false positives in PhishTank's archives. The authors should report these details and discuss how they clean up the database. Otherwise, it is hard for others to know the quality of the dataset and how effectively these models perform in practice.

## 2.3 Phishing Email Detection

Since phishing emails are the primary means of spreading phishing attacks, some studies focus on detecting phishing emails. Similar to the phishing page detection methods, the phishing email detection studies look at the intrinsic features of phishing emails and then apply machine learning algorithms to train binary classifiers to identify phishing emails. In this section, we discuss feature sets and machine learning algorithms applied in these studies.

### 2.3.1 Machine Learning Models Used for Phishing Email Detection

Table 2.3 lists the common machine learning algorithms used for phishing email detection and the results of the related studies. We can see that the machine learning algorithms used in phishing email detection are not very different from the phishing page detection. However, these studies have the same problem as the phishing page detection: there is very little work that discusses the choice of machine learning models. Instead, most studies concentrate on the discussion of feature sets. To the best of our knowledge, only one paper discusses the comparison of different algorithms under the same feature set and data set [1].

The authors compare six machine learning algorithms commonly used for phishing email detection and conclude that the random forests algorithm outperforms all other algorithms when the dataset is balanced. In their results, although the ANNs algorithm yields the worst performance in error rate, it has the best performance in the area under the ROC curve (AUC). It shows that the comparison of the algorithms is necessary because the performance is quite different under different data sets and feature sets. Unfortunately, most studies do not clearly state this part.

| **Work** | **Machine learning algorithm** | **# of features** | **Result** | **Dataset** |
|---|---|---|---|---|
| [14] | SVM | 25 | 88% accuracy | 100 phishing emails and 100 legitimate emails |
| [25] | Random forest | 10 | 96% TP and 0.1% FP | 860 phishing emails and 6,950 legitimate emails |
| [31] | Bayes Net | 7 | 92% accuracy | 4,559 phishing emails and 2,364 legitimate emails |
| [62] | Major voting of three binary classifiers | 10 | 97% accuracy | 2,000 phishing emails and 1,000 legitimate emails |
| [57] | Decision tree | 23 | 96.75% TP and 0.53% FP | 4,559 phishing emails (same as work [31]) and 4,559 legitimate emails |
| [47] | ANN | 5 | 92.2% accuracy | 14,370 phishing emails and 14,370 legitimate emails |
| [1] | ANN, SVM, Random forest and ... ( six machine learning algorithms in total) | 43 | 92.18% accuracy | 860 phishing emails and 6950 legitimate emails |

Table 2.3: Machine learning algorithms used for phishing email detection

### 2.3.2 Features Used for Phishing Email Detection

There is no doubt that in the past few years, academia and industry have achieved great success in detecting spam [19, 6, 23, 22]. It has inspired the following studies of research to detect phishing emails based on the existing spam detection technologies. Note that the feature set used for spam detection does not apply for phishing email detection. Therefore, studies in this area tend to explore specific features that can effectively distinguish phishing emails from regular emails. In total, there are thirty-three features divided into six categories, as shown in Table 2.4.

- **Semantic features**. The features in this category encode email texts as a series of numeric variables. For instance, Chandrasekaran et al [14] propose a feature set that consists of the statistical profiles of email content, such as the number of words and

| Category | Feature | Work | Description |
|---|---|---|---|
| Semantic features | Number of words | [14] | Count the number of words in the email |
| | Number of characters | [14] | Count the number of characters in the email |
| | Number of unique words | [14] | Count the number of unique words in the email |
| | Frequency of sensitive words | [14] | Count the frequency of each sensitive word, such as account, bank. |
| | Number of sensitive words | [14] | Count the total number of the sensitive words |
| | Suspicious email subject | [14, 31] | Check whether the email subject contains suspicious pattern |
| | Suspicious email greeting | [14] | Check whether the email uses suspicious greeting |
| | Suspicious words for link | [25] | Check whether the email uses suspicious words to describe the links |
| | Word vectors | [47] | Apply word2vec to convert the email content into a set of vectors |
| | Context score | [62] | Compare the similarity between email context and email text |
| | Text score | [62] | Compare root words of the email text with a predefined sensitive word set |
| Email components based features | Matched sender | [57, 31] | Check whether the sender domain is matched with message ID. |
| | Matched domains | [57] | Check whether the domains of embedded links match domain of sender |
| | Number of senders | [31] | Count the number of senders |
| | Number of sender domain | [31] | Count the number of unique sender domains |
| | Number of receivers | [57] | Count the number of receivers |
| | Number of attachments | [57, 47] | Count the number of attachments |
| | Message size | [57] | Count the size of message |
| URL based features | IP Host | [25, 31, 57] | Check whether hostname is an IP address |
| | Non-standard port | [57] | Check whether non-standard port used in URL |
| | Hidden link | [25] | Check whether use obfuscation techniques to hide links |
| | Number of dots in the URL | [25, 31, 57] | Count the number of . appear in the URL |
| | Suspicious symbols in the domain | [31, 57] | Check whether the domain contains some suspicious symbols such as *at(@)*, *dash(−)* |
| | Whether use redirection | [25] | Check whether URL uses redirection to reach the final page |
| HTML element based features | HTML email | [25, 57, 47] | Check whether the email organizes the content in HTML format |
| | Number of embedded links | [25, 47, 62] | Count the number of embedded links in the email |
| | Number of embedded links that contains suspicious symbols | [57] | Count the number of embedded links that contains suspicious symbols |
| | Number of domains | [25] | Count the number of domains in the embedded links |
| | Number of hyberlink image | [57] | Count the number of images that use hyberlink |
| | Contain Javascript | [25, 57, 47] | Check whether the email uses Javascript |
| Web based feature | Domain information | [25, 57, 62] | Check domain information by third party tool such as search engine and DNS query |
| Security based feature | Incorrect use of SSL | [57] | Check whether the embedded link in the email uses a trusted certificate |
| | DKIM signature | [62] | Check whether *Received From* section contains DKIM signature |

Table 2.4: Features used for phishing email detection

the number of characters. They also fetch features from the subject and the greeting by comparing them to a predefined corpus of 18 words that are commonly used in phishing emails. Similarly, Fette et al. [25] compile a sensitive term library for the text describing the phishing links in phishing emails. The authors claim that **link**, **click** and **here** are the most common words appearing in the text of embedded link. Verma et al. [62] and Moradpoor et al. [47] apply natural language processing (NLP) techniques to obtain a refined representation of email content. The former utilize WordNet[3] to replace the word in the email with the corresponding root word. For instance, the words "was" and "were" are eventually converted to the root word "to be". These root words are then compared to a predefined word library to calculate **Text Score**, which estimates the probability that the text is from a phishing email. Moradpoor et al. [47] use Word2Vec[4] to convert each word in the email into a vector that represents the relationship to nearby words. They then add all the vectors and use the average as the lexical representation of the email content.

- **Email components based features**. In addition to the body content, there are other components that can be used to identify phishing emails. Smadi et al. [57] and Hamid et al. [31] propose a set of features that check the characteristics of email components. They claim that the sender domain should be matched with the message ID, which is an identifier automatically generated by the sending mail system. However, in the phishing email, the attacker usually disguises its sender domain as a legitimate organization, but the email is sent out from a private host. Therefore, for the phishing email, the message ID usually cannot match the sender domain. What is more, they report that phishing emails often use multiple senders, recipients, and attachments, which is also not common in regular emails.

- **URL based features**. The goal of phishing emails is to induce victims to click the phishing URL and then direct them to the phishing page. Therefore, some studies apply the URL based features used in phishing page detection for phishing email detection. They combine these features with other categories of features to improve detection performance [25, 62, 57, 47].

- **HTML element based feature**. In the work [25, 62, 57, 47], the authors claim that the phisher tends to send HTML emails rather than plain text emails. They look at the features extracted from HTML elements that are commonly used in phishing page detection, and apply these features for phishing email detection.

[3] `https://wordnet.princeton.edu/`

[4] `http://deeplearning4j.org/word2vec`

- **Web based feature**. Some studies use the domain information applied in phishing page detection as the indicator of phishing emails [25, 57, 62]. In the work [25, 62], TF-IDF is applied to identify keywords of an email, and embedded links are compared to the search results by using the same keywords in a search engine. Sami Smadi et al. [57] check whether the domains of the embedded links exist in the valid reverse DNS records. A link without a corresponding reverse DNS entry is likely phishing.

- **Security based feature**. In the work [62, 57], some security-related features have been proposed to detect phishing emails. Verma et al. [62] suggest that if the embedded links are enhanced by SSL, the certificate should be checked to see whether it is signed by an authorized organization. Sami Smadi et al. [57] report that most phishing attacks do not have **DomainKeys Identified Mail (DKIM)** which is an identification used to verify whether the sender domain is indeed authorized by the domain it claims.

### 2.3.3 Analysis of Phishing Email Evolution

In order to understand the behavioral changes of phishing emails, there are some studies analyzing the evolution of phishing emails over time. Iran et al. [35] analyze more than 380,000 phishing emails with a time span of 15 months. They find that some attacks repeatedly use similar messages over a long period of time. To analyze the modification and the evolution of phishing emails, the authors extract 20 features, as shown in Table 2.5, and classify them into two groups based on the their lifespan and the number of instances. The features in the category **Transitory Feature** are the characteristics with a short lifespan. Conversely, the features in the **Transitory Feature** set are the features used in most attacks and constantly appearing for long periods of time. They report four **Transitory Features**: forgedMuaOutlook, msgidFromMtaHeader, htmlMessage, and mimeHtmlOnly, that indicate that most attacks utilize HTML to spread phishing emails and their majority target focuses on Microsoft Outlook.

### 2.3.4 Problem with Phishing Email Detection

Research on phishing email detection is superior to phishing page detection in dataset publishing. Some studies have conducted experiments under the same public dataset. However, these studies lack innovation in new models and new features. It can be seen that the models and feature sets for phishing email detection and phishing page detection are quite similar

| Feature | Name Description |
|---|---|
| forgedMuaOutlook | Forged mail pretending to be from MS Outlook. |
| hideWinStatus | Hide actual link by changing the displayed URL using onmouseover. |
| htmlMessage | Message contains HTML. |
| htmlMimeNoHtmlTag | HTML message, but no HTML tag. |
| htmlTagBalanceBody | HTML message has unbalanced body tags. |
| mimeBoundDigits | Spam tool pattern (contains digits only) in MIME boundary. |
| mimeHtmlOnly | Message only has text/html MIME parts. |
| mimeHtmlOnlyMulti | Multipart message has text/html MIME parts only. |
| missingMimeOle | Message has X-MSMail-Priority, but no X-MimeOLE. |
| msgidFromMtaHeader | Message-ID from MTA header. |
| msgidSpamCaps | Spam tool Message-ID (CAPS). |
| mpartAltDiff | HTML and text parts are different. |
| msoeMidWrongCase | MS Outlook Express Message-ID wrong case. |
| normalHttpToIp | Dotted-decimal IP address in URL. |
| partCidStock | Spammy image attachment (by stock Content-ID). |
| scriptInHtml | HTML contains script tag. |
| phSubjAccountsPost | Subject contains particular words (Activate, Verify, Restore, Flagged, Limited, . . . ). |
| phRec | Message has standard phishing phrase your account record. |
| urlHex | URL contains Hex characters. |
| weirdPort | Non-standard port number for HTTP. |

Table 2.5: 20 features used to analyze phishing email evolution

## 2.4 Phishing Kits

In order to better understand the phishers' behaviors and the technical details of phishing attacks, some researchers have conducted the analysis and studies on phishing kits. In this section, we talk about the methods used to analyze phishing kits and some interesting phenomena found in these studies.

### 2.4.1 Collection of Phishing Kits

In the studies on phishing page detection, the information of attack instances, such as the URL, the DOM and the screenshot, can be directly collected when attacks are detected or reported. However, it is not straightforward to collect the information of phishing kits since the phishing page is the rendered content returned by the server, where you cannot see the details of the source code and the backend design. To overcome this challenge, three approaches are proposed to collect the phishing kits.

- **Collecting phishing kits through underground forums**. As discussed in Section 1.2, the phishing kits are usually advertised through underground forums and

IRC channels. Therefore, it is possible to collect phishing kits by accessing these forums through searching the download entry [21] or directly trading with the kit producer [61] in the underground forums.

- **Collecting phishing kits through server crawling**. In some cases, after uploading and deploying the phishing kit on the web server, the attacker forgets to delete the original kit. Therefore, it is possible to fetch the phishing kit directly from the server holding the attack. Cova et al. [21] apply such a method to the phishing attacks reported by PhishTank and collect 151 phishing kits from the phishing hosts in two months.

- **Collecting phishing kits through deploying honeypot**. A honeypot is a network system which is used to monitor the behavior of a network attacker. In order to lure an attacker to detect the honeypot, the honeypot is deliberately configured with some known vulnerability. Once the attacker connects to the honeypot, all the operations are closely monitored under an isolated network. Han et al. [32] apply honeypots to collect and monitor the phishing kits. They have collected 643 unique phishing kits over a period of five months.

### 2.4.2 Analysis of Phishing Kits

The goal of phishing kit analysis is to better understand the backend design and behavior of phishing activities. In this section we outline some interesting findings reported in these studies.

- **Phishing target** Since the purpose of phishing attacks is to steal victims' sensitive information, it is not surprising that phishing targets are mainly the banks or e-commerce sites, as reported in the work [21]. Interestingly, with the popularity of social networks, the phishing target begins to gradually shift to social networking sites such as Facebook and Google [32, 61]. The main reason for this shift is that many e-commerce companies work with these leading social networks, and customers can use their social network accounts to complete registrations. As an example shown in Figure 2.8, a new customer can associate his Facebook or Google account with eBay. Table 2.6 shows the comparison of phishing targets in different periods.

- **Attacker behavior** Another analysis topic is to understand the behavior of attackers. Cova et al. [21] claim that 40% of phishing kits have a backdoor to collect stolen information. This means that the kit creator not only gains benefits through selling

| **Period** | April 2008 [21] | September 2015 - January 2016 [32] | March 2016 - March 2017 [61] |
|---|---|---|---|
| **Top 5 brand** | Bank of America | Paypal | Yahoo, Hotmail, Gmail |
| | eBay | Apple | Workspace Webmail |
| | Wachovia | Google | Dropbox |
| | HSBC | Facebook | Google Drive |
| | PayPal | French online tax payment system | Docusign |

Table 2.6: Comparison of phishing targets in different periods

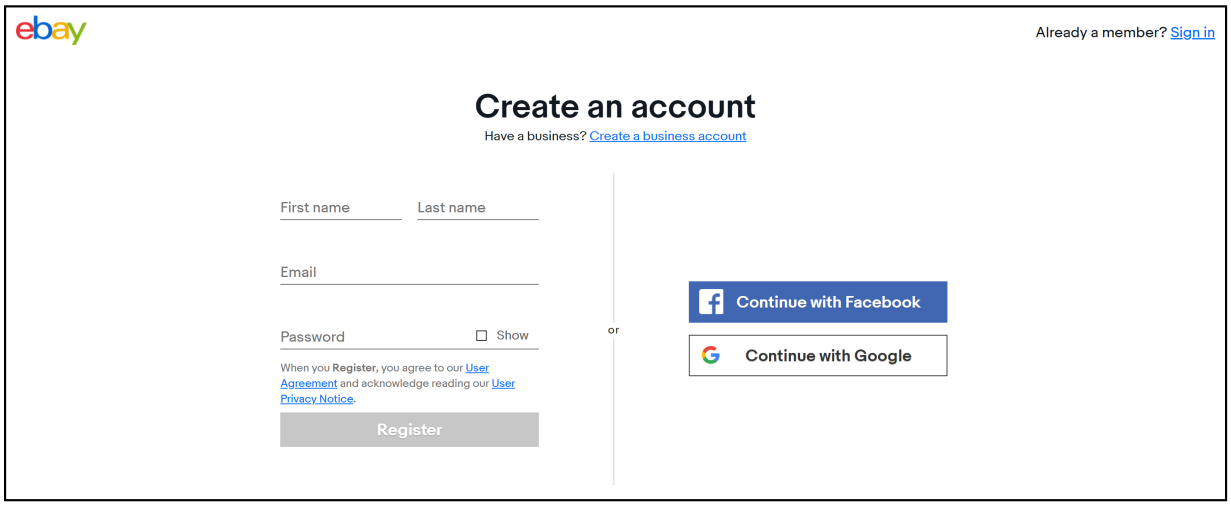



Figure 2.8: eBay registration page

the kit, but also gets the stolen information whenever the kit user successfully deceives the victim. Various obfuscation techniques are used to achieve this goal. For instance, a common method is to hide the recipient's email address by encoding it in base64 or in ASCII. They also find that some kits use custom encryption to encode the email address, for instance, replacing the character in the fixed location with a customized symbol. They report that a few kits send the stolen information to a different host rather than the server hosting the attack.

Han et al. [32] provide a deeper insight into phishing activities through monitoring the attackers' operations that have occurred on their honeypot. The authors look at the referer of the attacks. Their results show that 40% of attacks carry a facebook-related referer, which is followed by the Google search engine referer (29%). It indicates that either the attackers shift to using social networks for sharing information or intentionally forge the referer with a legitimate site to avoid being detected. They also observe that some kits generate a copy of the kit under multiple locations to avoid being blacklisted.

- **Effectiveness** To assess the effectiveness of current blacklists against phishing at-

tacks, Han et al. [32] look at two primary blacklists: PhishTank and Google Safe Browsing (GSB) [5]. They report that most phishing kits (62%) were blacklisted only after 75% of victims had accessed the corresponding phishing page, which took an average of 20 days after installation. It shows that the current blacklists are sluggish in response to phishing attacks and need to be overhauled.

## 2.5 Some New Directions of Phishing Detection

Recently, there are a number of new approaches and directions that have been proposed to detect phishing attacks. For instance, Corona et al. [20] propose a method to detect attacks hosted on compromised servers, which compares the page of the attack with the homepage of the host. Their assumption is that the phishing attacks often target the dominant legitimate site, so the phishing page is inconsistent with the homepage of the compromised host. They compare both the HTML similarity and the visual similarity between the homepage and the phishing page. A result of 97% true positive and 0.5% false positives is reported on a dataset of 1,012 phishing pages and 4,499 legitimate pages.

Some studies have shifted the protected devices from PCs to mobile devices. Bottazzi et al. [13] present a phishing detection framework for mobile devices, which combines the existing anti-phishing techniques, such as blacklists, with a machine learning detection model consisting of 53 features. They report a result of 89.2% true positive and 14.4% false positives on a dataset of 34,400 phishing pages and 51,600 legitimate pages. Aonzo et al.[9] discuss a phishing issue caused by two features of Android mobile: **mobile password managers** and **Instant Apps**. **Mobile password manager** is a type of mobile app that stores the user's login credentials locally and automatically populates them into the mobile app when the login action is detected. **Instant Apps** is a service provided by Google which allows the user to use Android apps without fully installing the app. The authors present several mechanisms which make it possible to use these two features to launch phishing attacks and design a set of APIs to prevent such attacks.

## 2.6 Conclusion

In this chapter, we overview the three major topics in anti-phishing studies: phishing page detection, phishing email detection and phishing kit analysis. We discuss these methods in detail and their limitations. Phishing blacklists have the issue that they cannot effectively

[5] `https://safebrowsing.google.com/`

prevent early attacks and only block the attack instances rather than the real attacks. Although studies on phishing email and phishing page detection can compensate for this deficiency, these studies face several problems. The studies on phishing page detection are poor at providing access to their dataset, which makes it impossible for others to reproduce their results and compare them to their own method. What is more, in recent studies on phishing page and phishing email detection, new research directions are needed to be explored. For studies on the analysis of the behavior behind the attack, very few studies outline a complete analysis of phishing attacks.

Our research goal is to fill these gaps. We compile and publish a large phishing dataset including around 100,000 phishing pages. In addition to phishing URLs, we also provide DOMs and screenshots of phishing pages, which allows others to compare our methods to their own approaches. In addition to our accessible dataset, our approaches on detecting phishing attacks have opened up several new directions, and have proven to be effective in detecting client-side and server-side attacks. We will discuss the details of these approaches in the rest of this thesis.

# Chapter 3

# Similarity Based Phishing Detection

## 3.1 Introduction

In this chapter, we look at the problem of phishing attacks from a new angle. Instead of comparing attacks with target sites, we propose phishing detection solutions based on comparing the similarities of known phishing attacks. Since we see that most new attacks are variations of known ones, it shows that detecting such replicas is a sensible strategic move. We start by introducing page fingerprint (Section 3.2.1). We then discuss the method of measuring the similarity of phishing attacks (Section 3.2.2), clustering algorithm (Section 3.2.3) and the related contents: the optimal threshold selection and the evaluation of clustering quality (Section 3.2.4). In Section 3.3, we discuss several improvements for the phishing attack clustering. We present the experimental results in Section 3.4 and then conclude at the end of this chapter.

## 3.2 Detecting Phishing Replicas and Variations

In this section, we introduce and discuss the various mathematical concepts that we have used in our detection algorithm.

### 3.2.1 Fingerprint of Phishing Page: Tag Vector

A web page is rendered as a tree data structure, called DOM. It is thus straightforward to measure similarity of pages by comparing the DOM. One common DOM metric is the *Tree Edit Distance (TED)*, which represents the minimal operation cost that transforms one

DOM into another. However, TED based comparison is time-costly, and the "state of the art" algorithm *AP-TED* [50] has the time complexity $O(n^3)$ and the memory complexity $O(m*n)$, where $n$ and $m$ are the sizes of the two DOMs being compared.

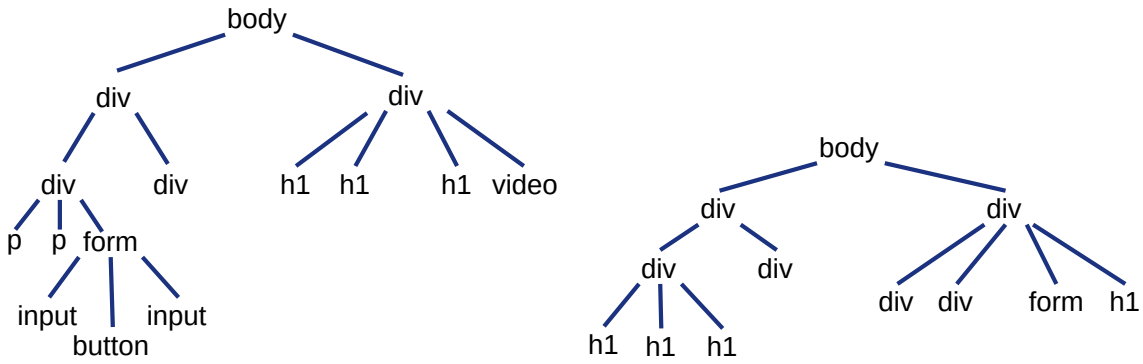
Some studies try to solve this problem by proposing a simplified page representation which is easier to compare. In [63], the authors explain that they have used a "Hash-based Near-duplicate Phish Detection" method that, in their case, finds that 72.67% of their phishing sites database are replicas from at least one other site in that same database. This "hash-based" method takes the HTML of the phishing page and removes all the spaces from it. It also removes all the default values in every INPUT field, replacing these values by empty strings. The resulting HTML is then hashed using SHA-1 [48]. Manku et al. [41] apply *SimHash* [16] to detect page duplicates for web crawling. The authors first convert each web page into a set of features, noted as feature vector, and then transform the feature vector into a 64-bit SimHash. The hamming distance between two SimHash is used to evaluate the similarity of two page (the smaller the value, the more similar the pages).

However, none of the above methods apply to the similarity comparison of phishing attacks. The hash-based method is very strict and fails to match pages that have very small variations of each other. The SimHash-based method only works on a page with lots of text, but phishing pages usually contain very little or even no text (e.g., only have images). Therefore, we need a page representation which is able to quickly compare the similarity of the DOMs.

Our idea is to compute what we have called the "tag vector" of the phishing page. We first define a corpus of HTML tags. We use the complete set of HTML elements provided by the World Wide Web Consortium[1], from which we have removed some of the more common tags such as <body>, <head> and <html>. We end up with a list of 107 tags, which can be fetched from our website [2]. We then assign an arbitrary but fixed ordering of the tags in the corpus, and define a "vector" of the size of the corpus. For each DOM, we compute the corresponding vector by counting how many times each tag of the corpus appears in the DOM. As an example, consider Figure 3.1, where two simple DOMs are provided. If the corpus consists of the html tags <form> <b> <p> <h1> <button> <video> <input> <iframe> and , in that order, then the tag vector for the page $p_1$ is <1, 0 ,2 ,3, 1, 1, 2, 0, 4>, while the tag vector for the page $p_2$ is <1, 0 ,0 ,4, 0, 0, 0, 0, 6>.

[1] `https://www.w3.org/TR/html-markup/elements.html`

[2] `http://ssrg.site.uottawa.ca/phishingdata`




(a) DOM of the page $p_1$

(b) DOM of the page $p_2$

Figure 3.1: Tag vectors

### 3.2.2 Similarity of Phishing Pages: Proportional Distance

There are two common metrics of measuring the similarity of vectors: *Euclidean Distance* and *Cosine Similarity*, but these metrics do not apply for tag vectors. The problem with *Euclidean Distance* is that it is not normalized and has the tendency to yield larger distances for vectors with more tags. Thus, attackers could easily increase the distance by adding many dummy tags (e.g. ) without affecting the appearance of the page. *Cosine Similarity* evaluates the angle between vectors, regardless of the value scale, which will cause many false positives in practice. To overcome the above limitations, we use what we call the **proportional distance** to measure the similarity between the tag vectors. Note that other metrics such as the **Normalized Squared Euclidean Distance** could have been used, probably with comparable results. We chose proportional distance because it is fast to compute and yields good results.

**Proportional distance** is a fairly simple difference metric, in which we essentially count the number of elements of the corpus that appear a different number of times in both pages (in other words, the number of indexes of the tag vectors that have a different value). This simple count would have the tendency to generate greater differences for pages that are relatively complex and that use a large variety of tags from the corpus, while pages that only use a few of the tags would have smaller differences. In order to alleviate this tendency, we divide the initial number by the number of corpus tags that appear in at least one of the two pages.

Let $t_1$ and $t_2$ be two non-null tag vectors over the same corpus of size $n$. Formally, we

define the **proportional distance**[3] $PD(t_1, t_2)$ between $t_1$ and $t_2$ as follows:

$$PD(t_1, t_2) = \frac{\sum_{i=1}^{n} D(t_1[i], t_2[i])}{\sum_{i=1}^{n} L(t_1[i], t_2[i])} \tag{3.1}$$

$$\text{where } D(x,y) = \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{otherwise} \end{cases} \text{ and } L(x,y) = \begin{cases} 1 & \text{if } x \neq 0 \text{ OR } y \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

As an example, consider the two vectors from Figure 3.1: seven of the nine possible tags of the corpus are used by at least one of the two vectors, and only the first one, <form>, appears the same number of times in both vectors, so the $PD(p_1, p_2) = 6/7 = 0.85$, which indicates a large difference between the two pages.

### 3.2.3 Clustering Algorithm

The goal of our clustering is to group together in the same cluster the pages that have relatively similar tag vectors, since they are deemed to be similar to one another. We believe that a given site can be slightly modified a great number of times and re-published after each modification. It is thus likely that we are probably looking at long "strings" of modifications, in which each new instance is fairly similar to the instance it is based upon, but after a while we may end up with instances that are fairly far apart. We thus avoid to use a centroid-based clustering algorithm such as k-means. Instead, we use a hierarchical clustering approach to group together vectors that have at least one other vector which is relatively similar in the cluster. This has the advantage of always producing the same output for the same input, regardless of the order in which the vectors are added to the model. This is a significant value-added in our context, since the database is going to evolve over time, as new phishing sites are being discovered. These new sites can be added to the model in a greedy fashion, without having to recompute the model. Note that when a new site is added to the set, two clusters can end up being merged together. A straightforward greedy algorithm has been used in our experiments, with an overall complexity of $O(n^2)$ for $n$ vectors, as shown in Algorithm 1. The algorithm first calculates the proportional distance of each pair of tag vectors and builds a set of the edge information $ES$ (lines 5-7). The edge set is then sorted in ascending order of distance (line 8). Finally, it starts to merge vectors from the smallest distance to the larger one until the given threshold is reached (lines 9-13).

[3]The proof that this is a mathematical distance can be found in Appendix A

**Algorithm 1** Clustering Algorithm

**Input:** A set of tag vectors being clustered $T = \{t_1, t_2, ..., t_n\}$ and a threshold $H$

**Output:** A set of clusters of the given tag vectors $C$

```
1: procedure CLUSTERING(Vecset T, Double H)
2:     ES ← {}
3:     n ← size of T
4:     C ← {{t_1}, {t_2}, ..., {t_n}}          ▷ In the beginning, each vector forms a cluster
5:     for i ← 0 to n − 2 do                    ▷ Initialize edge information
6:         for j ← i + 1 to n − 1 do
7:             Add [t_i, t_j, PD(t_i, t_j)] into ES
8:     SES ← Sort(ES)                           ▷ Sort edge set in ascending order of distance
9:     for t_i, t_j, pd ∈ SES do                ▷ Merge clusters until reach to the threshold
10:        if pd > H then
11:            break
12:        if isSameCluster(t_i, t_j) = false then
13:            MergeCluster(t_i, t_j, C)
       return C
```

### 3.2.4 Coupling of Clustering and Optimal Threshold

As is usual with clustering algorithms, our method is based on some threshold $H$. A common and intuitive definition of a "good" clustering threshold is a threshold that yields clusters that are both compact and far away from each other, so this is what we also aim for here. We define a **coupling of clustering** using the average proportional distance of vectors inside a cluster. We average this value, and divide it by the smallest proportional distance between two vectors in any two clusters. The numerator evaluates the compactness of the clusters (the more compact the clusters, the smaller the value) while the denominator evaluates how far apart clusters are: the further apart the clusters, the larger the value. Thus, a smaller value for our coupling of clustering is better.

Formally, given $C_k \subseteq C$ the subset of a set of clusters $C$ having more than one element, we define the coupling of clustering as follows: for a given cluster $C_i \in C_k$ having $n_i > 1$ elements, we define the compactness of the set of clusters $C_k$ as follows:

$$\begin{aligned} Comp(C_k) &= \frac{1}{|C_k|} \sum_{C_i \in C_k} (InterComp(C_i)) \\ \text{where } InterComp(C_i) &= \frac{2}{n_i(n_i-1)} \sum_{x \in C_i, y \in C_i, x \neq y} PD(x,y) \end{aligned} \tag{3.2}$$

We also define the minimal distance in a set of clusters $C$:

$$\begin{aligned} Min(C) &= min_{C_i,C_j \in C, C_i \neq C_j}\ InterMin(C_i, C_j) \\ \text{where } InterMin(C_i, C_j) &= min_{x \in C_i, y \in C_j}\ PD(x, y) \end{aligned} \tag{3.3}$$

Finally, the **coupling of clustering** of $C$ is defined as:

$$CC(C) = \frac{Comp(C_k)}{Min(C)} \tag{3.4}$$

To choose the optimal clustering threshold for a given vector set, we calculate **coupling of clustering** with various thresholds. The one which minimize the **coupling of clustering** is considered optimal.

### 3.2.5 Algorithm to Detect Phishing Variations

To identify whether a new tag vector is a phishing variation, the vector is compared against every known attack in our phishing database. If the distance is less than the optimal threshold, it is considered as phishing. The problem with this approach is that it is not easy to scale since as more and more phishing instances are added into our database, the comparison time will linearly increase.

To solve this problem, we take the advantage of our clustering result to identify the boundary of clusters. Given a cluster of vectors $C$, we define a new vector $pt$ called **prototype of cluster**, where each element is a set of values appearing in this position. For instance, consider a cluster of three vectors, <<0,1,34,2,4>, <2,1,36,3,4>, <2,1,37,3,4>>, its prototype is <<0,2>, <1>,<34, 36, 37>, <2,3> ,<4>>. In order to compare vector and prototype, we redefine the $PD$ in Formula 3.1 as follows:

$$\begin{aligned} PD_{proto}(t, pt) &\in \left[ \frac{\sum_{i=1}^{n} minD(t[i], pt[i])}{\sum_{i=1}^{n} maxL(t[i], pt[i])}, \frac{\sum_{i=1}^{n} maxD(t[i], pt[i])}{\sum_{i=1}^{n} minL(t[i], pt[i])} \right] \\ \text{where } minD(e, E) &= \begin{cases} 1 & \text{if } e \notin E \\ 0 & \text{otherwise} \end{cases} \text{ and } maxD(e, E) = \begin{cases} 1 & \text{if } E \setminus e \neq \varnothing \\ 0 & \text{otherwise} \end{cases} \\ minL(e, E) &= \begin{cases} 1 & \text{if } \exists z \in (E \cup e) \\ & \text{where } z \neq 0 \\ 0 & \text{otherwise} \end{cases} \text{ and } maxL(e, E) = \begin{cases} 1 & \text{if } (E \cup e) \setminus 0 \neq \varnothing \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.5}$$

Now we explain how to use the **prototype of cluster** to identify the boundary of a cluster. We first calculate the value range of $D$ and $L$. Consider that at a particular position $i$, if the value ($t[i]$) in vector $t$ does not appear in the value set ($pt[i]$) of **prototype of cluster**, the minimum of $D$, $minD$, is 1, otherwise it is 0. On the contrary, if $pt[i]$ contains any value other than $t[i]$, the maximum of $D$ is 1. Similarly, the minimum of $L$, $minL$, is 1 if there exists a non-zero value in the union of $pt[i]$ and $t[i]$; otherwise it is 0. The maximum ($maxL$) of $L$ is 1 if at least one of $pt[i]$ and $t[i]$ does not contain 0. Finally, we sum the values for all the positions and get the final value range of $PD_{proto}$. If the minimum of $PD_{proto}$ is greater than the threshold, the vector must be out of the cluster; if the maximum of $PD_{proto}$ is less than the threshold, the vector is definitely in the cluster. We only have to compare each vector in the cluster if it does not belong to the above two cases. The complete algorithm is shown in Algorithm 2.

**Algorithm 2** Detect Variations Algorithm

**Input:** A set of clusters of the known phishing attacks $C$, the optimal threshold $H$, and the vector being checked $t$

**Output:** Whether the vector is a variation of the known phishing clusters

```
1:  procedure isVariation(ClusterSet C, Double H, Vector t)
2:      PT ← fetchProtoType(C)                          ▷ Fetch cluster prototype
3:      for pt_i ∈ PT do                                ▷ Check each cluster
4:          minD, maxD, minL, maxL ← 0
5:          for e_j ∈ pt_i do                           ▷ Check each element (tag) of prototype
6:              if elem(t, j) ∉ e_j then                ▷ Update minD and maxD
7:                  minD++, maxD++
8:              else if len(e_j) ≠ 1 then
9:                  maxD++
10:             if 0 ∉ e_j or elem(t, j) ≠ 0 then       ▷ Update minL and maxL
11:                 minL++, maxL++
12:             else if len(e_j) ≠ 1 then
13:                 maxL++
14:         minPD ← minD/maxL
15:         maxPD ← maxD/minL
16:         if minPD > H then return false
17:         else if maxPD ≤ H then return true
18:         else if isInCluster(t, C_i) = true then return true
        return false
```

## 3.3 Improvement of Phishing Clustering

Although our clustering and detection algorithms are simple and efficient, there are some areas that need to be improved. In this section, we discuss some improvements of the content of the above algorithms.

### 3.3.1 Weighted Proportional Difference

In our experiments, we find that the proportional distance, $PD$ as defined in Section 3.2.2, does not emphasize the "amount" of differences between each HTML tag, and simply focuses on whether the number of tags is the same. For example, the vector $t_1 = \{1, 2, 5, 6\}$ and $t_2 = \{109, 2, 5, 6\}$ both have the same distance to the vector $t_3 = \{2, 2, 5, 6\}$, that is, $PD(t_1, t_3) = PD(t_2, t_3)$. For our study, we would like to capture the fact that $t_2$ is more different from $t_3$ than $t_1$ is. Therefore, we define a new distance, called the **Weighted Proportional Difference** ($WPD$) to compare the similarity of tag vectors. Instead of using the Hamming Distance as the numerator, we use the sum of the *Weighted Differences* ($WD$), defined by the following formula:

$$WD(t_1, t_2) = \sum_{i=1}^{n} \frac{|t_1[i] - t_2[i]|}{max(t_1[i], t_2[i])} \tag{3.6}$$

In $PD$, the value of $D$ for a given tag is boolean (0 or 1). For tags that are used in both vectors, $WD$ is in the range $[0, 1)$. The larger the difference between the number of tags, the larger $WD$ is.

We define $S$ as follows:

$$\begin{aligned} S(t_1, t_2) &= \sum_{i=1}^{n} EQU(t_1[i], t_2[i]) \\ \text{where } EQU(t_1[i], t_2[i]) &= \begin{cases} 1 & \text{if } t_1[i] = t_2[i] \text{ AND } t_1[i] \neq 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.7}$$

Finally, the *Weighted Proportional Difference* ($WPD$) is defined as follows:

$$WPD(t_1, t_2) = \frac{WD(t_1, t_2)}{WD(t_1, t_2) + S(t_1, t_2)} \tag{3.8}$$

Note that the *Weighted Proportional Difference* could raise the difficulty for attackers to increase the similarity. Attackers need to make more changes for multiple tags to avoid being detected as a variation of known attacks.

### 3.3.2 Algorithm to Detect Phishing Variations for $WPD$

Although the *Weighted Proportional Difference* $WPD$ is an improved version of *Proportional Distance* $PD$, we cannot directly use the detection algorithm proposed in Section 3.2.5 for $WPD$ since the definition of the numerator and denominator in the distance has been completely changed.

Our idea is to still use the prototype of cluster to find the boundary. Considering the definition of $WPD$, we can convert it into the following one:

$$WPD(t_1, t_2) = \frac{WD(t_1, t_2)}{WD(t_1, t_2) + S(t_1, t_2)} = \frac{1}{1 + \frac{S(t_1,t_2)}{WD(t_1,t_2)}} \tag{3.9}$$

Similarly, if we know the minimum and maximum of the $S$ and $WD$, then we can get the value range of $WPD(t, pt)$: $\left[\frac{1}{1+\frac{maxS(t,pt)}{minWD(t,pt)}}, \frac{1}{1+\frac{minS(t,pt)}{maxWD(t,pt)}}\right]$, where $t$ is the vector being checked and $pt$ is the prototype of the phishing cluster. In order to calculate the $minS(t, pt)$, $maxS(t, pt)$, $minWD(t, pt)$ and $maxWD(t, pt)$, we compute the related minimum and maximum in each position, and finally add them up. Formally,

$$WPD_{proto}(t, pt) \in \left[\frac{1}{1 + \frac{\sum_{i=1}^{n} maxS(t[i],pt[i])}{\sum_{i=1}^{n} minWD(t[i],pt[i])}}, \frac{1}{1 + \frac{\sum_{i=1}^{n} minS(t[i],pt[i])}{\sum_{i=1}^{n} maxWD(t[i],pt[i])}}\right] \text{ where}$$

$$minWD(e, E) = min\left(\frac{|e - r|}{max(e, r)} | \forall r \in E\right)$$

$$maxWD(e, E) = max\left(\frac{|e - r|}{max(e, r)} | \forall r \in E\right)$$

$$minS(e, E) = \begin{cases} 1 & \text{if } e \neq 0 \text{ and } E = \{e\} \\ 0 & \text{otherwise} \end{cases} \text{ and } maxS(e, E) = \begin{cases} 1 & \text{if } e \neq 0 \text{ and } e \in E \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Note that in order to calculate the value range of $WD(e, E)$, we have to know all the values at each position in the prototype. Therefore, we make the following changes to the prototype. We include all the values and sort them in ascending order, noted as $E = \{e_1, e_2, ..., e_n\}$ where $e_1 <= e_2 <= ... <= e_n$. The maximum of $WD(e, E)$ is the largest of $\frac{|e-e_1|}{max(e,e_1)}$ and $\frac{|e-e_n|}{max(e,e_n)}$, while the minimum is the smallest of $\frac{|e-e_l|}{max(e,e_l)}$ and $\frac{|e-e_r|}{max(e,e_r)}$ where $e_l$ and $e_r$ are the two closest to $e$, which can be done by applying binary search with a complexity $O(log(n))$. For $S(e, E)$, if value set $E$ only has the non-zero element $e$, its minimum $minS(e, E)$ is 1. Its maximum $max(e, E)$ is 1 when non-zero $e$ appears in the $E$. The complete algorithm used to detect phishing variations for $WPD$ is shown below.

**Algorithm 3** Detect Variations Algorithm for $WPD$

**Input:** A set of clusters of the known phishing attacks $C$, the optimal threshold $H$, and the vector being checked $t$

**Output:** Whether the vector is a variation of the known phishing clusters

1: **procedure** isVariation($ClusterSet\ C$, $Double\ H$, $Vector\ t$)
2: $PT \leftarrow fetchProtoType(C)$ ▷ Fetch cluster prototype
3: **for** $pt_i \in PT$ **do** ▷ Check each cluster
4: $minWD, maxWD, minS, maxS \leftarrow 0$
5: **for** $e_j \in pt_i$ **do** ▷ Check each element (tag) of prototype
6: $minWD_j \leftarrow 1$
7: $maxWD_j \leftarrow 0$
8: **for** $r \in e_j$ **do** ▷ Update $minWD$ and $maxWD$
9: $tmp \leftarrow \frac{|elem(t,j)-r|}{max(elem(t,j),r)}$
10: **if** $tmp > maxWD_j$ **then**
11: $maxWD_j \leftarrow tmp$
12: **if** $tmp < minWD_j$ **then**
13: $minWD_j \leftarrow tmp$
14: $minWD+ = minWD_j$, $maxWD+ = maxWD_j$
15: **if** $elem$(t,j) $\in e_j$ **and** elem($t$, $j$) $\neq 0$ **then** ▷ Update $minS$ and $maxS$
16: $maxS{+}{+}$
17: **if** $len(e_j) = 1$ **then**
18: $minS{+}{+}$
19: $minWPD \leftarrow \frac{1}{1+\frac{maxS}{minWD}}$
20: $maxWPD \leftarrow \frac{1}{1+\frac{minS}{maxWD}}$
21: **if** $minWPD > H$ **then return** $false$
22: **else if** $maxWPD \leq H$ **then return** $true$
23: **else if** isInCluster($t$, $C_i$) $= true$ **then return** $true$
**return** $false$

### 3.3.3 Intra-Cluster Vectors Connections

There are at least two common models that are widely used when it comes to intra-cluster connections: 1) Single-linkage, where each node inside the cluster is connected to at most one parent, creating a minimal spanning tree over the elements of the cluster, or 2) Complete-linkage, where a complete graph is created between all the elements of the clusters (we use this connection in Algorithm 1). However, we found that neither of these two models can accurately capture the evolution of the phishing vectors inside a cluster. A good model should keep a connection between the elements of a series of modifications done to a given attack (and some of these elements may end up being fairly far apart after a long series of modifications), but it should also capture the fact that some elements are at a very small distance from each other within the cluster. This idea is illustrated in Figure 3.2. Vectors $a$, $b$, $c$ and $d$ are close to one another, meaning that there is little variation between these four vectors. On the other hand, Vector $e$, while still part of the same cluster, is actually relatively "far" from these first four vectors, and is only linked to them through a long series of small variations.

To capture these series of modifications done to the phishing attacks inside a cluster, we propose to use a *Semi-Complete Linkage* ($SCL$) model. Specifically, for any pair of tag vectors $t_i$ and $t_j$ in the same cluster, where $i \neq j$, we have an edge $E(t_i, t_j) \in SCL$ if and only if $WPD(t_1, t_2) \leqslant OPT$, where $OPT$ is the optimal threshold for tag vector clusters defined in Section 3.2.4. A simple way to see this model is that inside a cluster, vectors that are "similar" are linked together. This model is an intermediate model between the spanning tree and the complete graph.

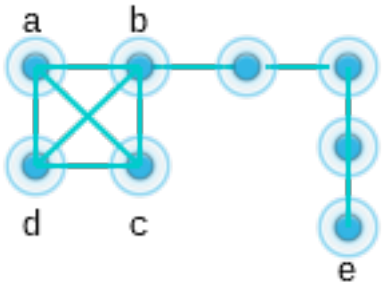



Figure 3.2: An illustration of a *Semi-Complete Linkage* graph.

### 3.3.4 Coupling of Semi-Complete Linkage Clustering

We now explain how we define the *coupling of clustering for the semi-complete linkage graph*. We define $Min(C_i, C_j)$ to be the minimal distance between two clusters, which is

defined as the minimum distance that can be found between two vectors, one in $C_i$ and one in $C_j$. That is :

$$Min(C_i, C_j) = min(\{WPD(x, y) | \forall x \in C_i, \forall y \in C_j\}) \tag{3.11}$$

As discussed in Section 3.3.3, we use the $SCL$ model to capture the connections inside tag vector clusters. Thus, we define the **coupling of $SCL$ clustering** with the following formula, which computes the average distance inside the SCL, and divides it by the distance between clusters. It is noted that we only include the clusters that have more than one element.

$$\frac{\frac{1}{k}\sum_{i=1}^{k}\frac{1}{|E_i|}\sum_{j=1}^{|E_i|}\{WPD(x, y) | E_j(x, y) \in SCL_i\}}{min\{Min(C_i, C_j) | i \neq j, 1 \leqslant i, j \leqslant k\}} \tag{3.12}$$

where $k$ is the number of clusters having more than one element, $E(x, y)$ is the edge between $x$ and $y$ in the $SCL$ graph, $C_i$ is the $i^{th}$ cluster with more than one element, $SCL_i$ is the $SCL$ for $C_i$ and $|E_i|$ is the number of edges in $SCL_i$.

## 3.4 Experiment

In this section, we report our results, starting with the description of our database. We then discuss the optimal threshold and clustering result. A performance comparison between the original clustering algorithm and the improved one is also conducted. Finally, we report the analysis of phishing clusters.

### 3.4.1 Database

We have used three sources to compile our database of phishing sites, which includes the community-driven portal PhishTank[4], the cloud-based threat intelligence sharing platform IBM X-Force[5], and the phishing intelligence analysis platform OpenPhish[6]. We have collected a list of 98,571 "verified" phishing sites by downloading archives daily between January 1st, 2016 and January 1st, 2018. We have also recorded the submission date of each URL. For each of these URLs, we automatically fetch the DOM by sending a Web crawler to the URL. We also gather the web server's IP address. Note that in some cases,

[4] `https://www.phishtank.com`

[5] `https://www.ibm.com/security/xforce`

[6] `https://openphish.com/`

the initial URL returns a redirection to another URL, in which case we recursively follow redirections until reaching the actual alleged phishing site. The data reported is the data collected after following these redirections.

In order to compare the result of our clustering algorithm on phishing sites and on legitimate sites, we have also gathered a database of non-phishing websites. For that purpose, we use the portal Alexa[7], which monitors web traffic and provides lists of the most popular websites. We have fetched 24,800 legitimate sites, including 9,737 URLs from a series of Alexa free "top 500" most popular site by countries [3] and another 15,063 URLs randomly selected from Alexa's top 100,000 to 460,697 web sites.

### 3.4.2 Clustering Results

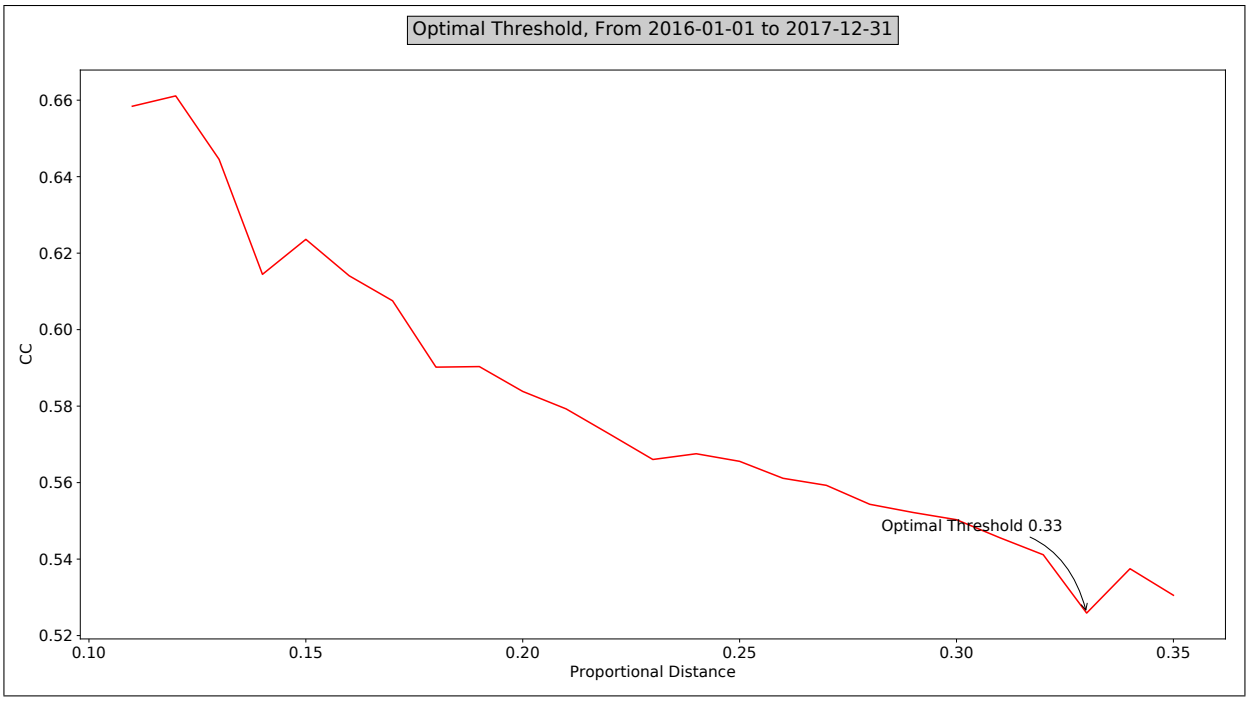
The results of the clustering model presented in Section 3.2 and Section 3.3 are shown in Table 3.1. The difference between the two models is that the latter improved model uses **Weighted Proportional Difference ($WPD$)** and **Semi-Complete Linkage ($SCL$)** to construct the clusters. We find that our dataset of 98,571 phishing instances only generates 22,785 different vectors, while 24,800 legitimate sites generates almost the same number of vectors. Using **Proportional Distance ($PD$)**, 90,551 phishing instances fall into 5,729 clusters that contain more than one instance (we note these clusters as "flagged" clusters in the table), which means that 91.86% of our entire phishing database is made of repeated attacks over our observation period. In the improved cluster model which uses $WPD$ and $SCL$, we catch more phishing variations (92.91%) but yield fewer flagged clusters.

| **# of phishing instances** | | 98,571 |
|---|---|---|
| **# of phishing vectors** | | 22,785 |
| **# of legitimate sites** | | 24,800 |
| **# of legitimate vectors** | | 24,661 |
| | **Model using PD** | **Model using WPD and SCL** |
| **Optimal threshold** | 0.33 | 0.26 |
| **# of flagged clusters** | 5,729 | 5,334 |
| **# of phishing instances in flagged clusters** | 90,551 (91.86%) | 91,580(92.91%) |
| **# of legitimate sites in phishing clusters** | 128 (0.52%) | 161 (0.65%) |

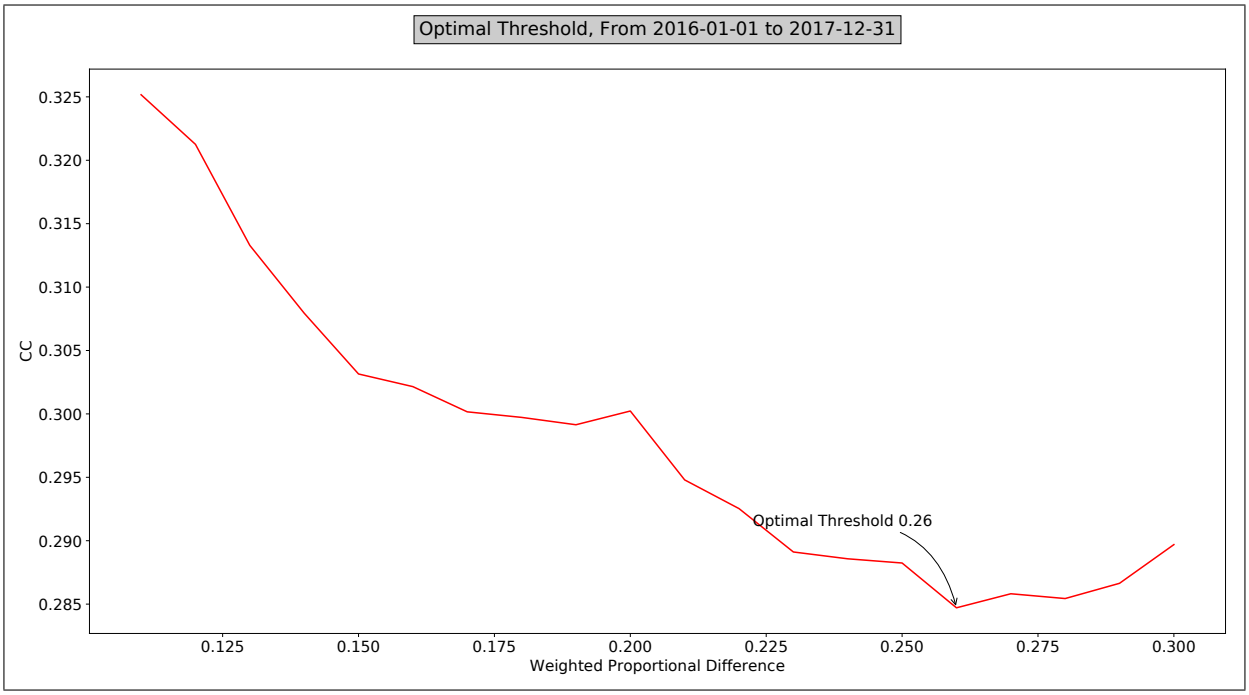
Table 3.1: Clustering results

Figure 3.3 shows how we choose the optimal threshold for both cluster models. We can see that both models return the optimal threshold at the elbow point with the lowest value of **coupling of clustering**. Since $WPD$ always yields a smaller value than $PD$, the improved clustering model has a lower optimal threshold 0.26.

[7] `http://www.alexa.com/`




(a) Optimal threshold of model using PD

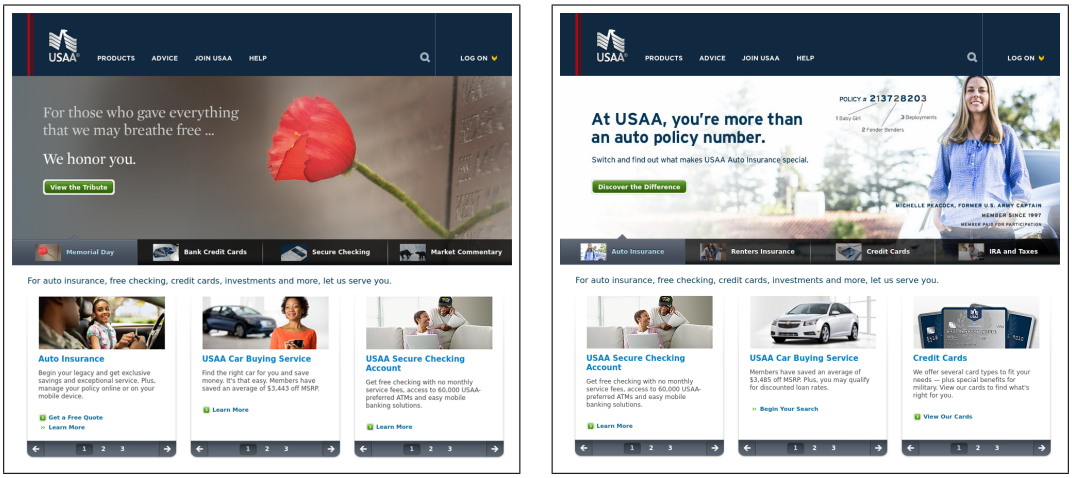



(b) Optimal threshold of model using WPD and SCL

Figure 3.3: Choosing an optimal threshold

In our database of 24,800 legitimate sites, only 128 of these sites end up in one of the phishing clusters if we use $PD$ as the similarity metric, a false positive rate of only 0.52%.

In the improved cluster model, the false positives is slightly increased but still remains very low at 0.65%. The main reason for these false positives is that some phishing sites are copies of the legitimate site (See for example Figure 3.4 (a) and (b)). It is actually remarkable how infrequently this situation occurs. It shows that phishers do not tend to copy directly the site they are attacking. The reason for this is probably because it makes it particularly easy to detect and filter the attacks [52]. In practice, we could easily reduce this false positive rate even further, e.g., by identifying the legitimate sites being targeted [39] and white-listing them.




(a) Legitimate site (b) Phishing site

Figure 3.4: Example of a false positive

### 3.4.3 Clustering Comparison

Our results show that the improved clustering model catches more phishing variations than the original model using $PD$. In order to better understand the details of these additional captured cases, in this section we discuss the differences of clustering result between two models.

| | Orignal model (using PD) | Improved model (using WPD and SCL) |
|---|---|---|
| **# of singleton attacks** | 8020 | 6991 |
| **# of attacks that the model cannot detect variants but can by the other model** | 1052 | 23 |
| **Distance to the nearest neighbor** | 0.27-0.32 | 0.26-0.30 |
| **Optimal threshold** | 0.33 | 0.26 |

Table 3.2: Clustering comparison

To simplify the description, we name the attacks not in the flagged clusters as "singleton attacks". Our comparison result is shown in Table 3.2. There are 1,052 singleton attacks

that the original model cannot detect variants but can by the improved model. Most of these singleton attacks change multiple tags, such as , <p>, which increases the $PD$ between vectors. But the difference of change is very small, which has little impact on the $WPD$, so they can still be identified by the improved model as phishing variations. There are 23 singleton attacks identified by the improved model but not the original model. For these instances, the $PD$ to the nearest neighbor ranges from 0.27 to 0.32, and the $WPD$ to the nearest neighbor ranges from 0.26 to 0.30. As can be seen, these instances refer to vectors that are close to the threshold boundary. Through manual checking, we find most of the instances are false positives yielded by the original model, and only 5 instances are the actual phishing variations.

### 3.4.4 Performance of Identifying Phishing Variations

Our clustering-based phishing detection approach requires a comparison between the given vector and every known phishing vector. To boost the performance, in Section 3.2 we propose the cluster prototype which takes advantage of clustering results to identify the cluster boundary. Our main results of performance comparison between using and not using the prototype are shown in Table 3.3.

| **# of test phishing vectors** | 2,282 | |
|---|---|---|
| **# of test legitimate vectors** | 2,282 | |
| **# of test instances** | 10,941 | |
| **# of phishing vectors in clusters** | 20,503 | |
| | **Model using PD** | **Model using WPD and SCL** |
| **# of phishing clusters** | 12,583 | 11,320 |
| **# of test vectors in phishing clusters** | 1,097 | 1,264 |
| **Time for building the prototype** | 2 s | |
| **Time for comparison without prototype** | 2,919 s | 4,884 s |
| **Time for comparison with prototype** | 1,461 s | 3,451 s |
| **Time per vector without prototype** | 640 ms | 1070 ms |
| **Time per vector with prototype** | 320 ms | 756 ms |
| **Time per instance without prototype** | 267 ms | 446 ms |
| **Time per instance with prototype** | 134 ms | 315 ms |

Table 3.3: Performance comparison of phishing variation detection

We first compile a balanced test set of 2,282 phishing vectors (10% of the entire phishing dataset) and legitimate vectors of the same size, that are randomly sampled from our phishing and legitimate dataset. The remaining 20,503 phishing vectors are used as the phishing clusters being compared. It would be interesting to know what this difference

in time is, since most vectors that will be compared in deployment will be legitimate vectors. In order to eliminate the measurement error, we perform 10 experiments for each comparison and report the average of the measurements.

In the comparison for the clustering model using $PD$, 1,097 test vectors are detected as phishing variations, which would take 2,919 seconds to complete the comparison without using the prototype. By using the prototype, the comparison time is reduced by half. It indicates that the cluster boundary yielded by the cluster prototypes greatly reduces the number of comparisons with the vectors. If we look at the improved clustering model using $WPD$ and $SCL$, 1,264 test vectors are identified as similar to the known attacks. This model takes more time than the model using $PD$ since computing $WPD$ is more complicated than $PD$. If we use the prototype, the comparison time is reduced to 71%, and it takes an average of 756 ms to check a vector (resp. 315 ms for one instance). It only takes around 2 seconds to build the cluster prototypes for more than 11,000 clusters, which shows that updating the prototype is very efficient when such a phishing detection approach is used with streaming input.

As we can see, through the comparison between our original clustering model and the improved clustering model, the latter model is more effective at identifying phishing variations. Therefore, we use the improved clustering model in the following analysis.

### 3.4.5 Phishing Targets

| Cluster index | Brand name of phishing target | # of pages |
|---|---|---|
| 1 | Gmail | 11391 |
| 2 | PayPal | 4493 |
| 3 | Google Docs | 2016 |
| 4 | MailBox | 1933 |
| 5 | DropBox | 1619 |
| 6 | Yahoo | 1536 |
| 7 | Multiple | 1421 |
| 8 | PayPal | 1332 |
| 9 | Free Mobile | 1229 |
| 10 | OneDrive/GoogleDrive | 1191 |

Table 3.4: Target brand of top 10 clusters

In Table 3.4, we show the targeted brands for the top 10 clusters ordered by # of attack instances. Except for the seventh cluster, all other clusters have well identified targets. The

seventh cluster targets most major social networks and email providers, such as Youtube and Gmail. One example of the attack in this cluster is shown in Figure 3.5(a). In the tenth cluster, we find an attack that uses the same template but targeting different brands, as shown in Figure 3.5(b) and Figure 3.5(c). The only change between the two attacks is the text and images in the page: their tag vectors are identical. The reason clusters are formed with attacks targeting the same brand is because different templates are used to target each brand.

### 3.4.6 Lifespan of Phishing Sites

One cluster represents one attack, which is made of several attack instances (actual mailing campaigns trying to drive victims to the site). We investigate here the lifespan of these attacks in our database. The lifespan is defined as the duration between the first and the last observed attack instance[8]. Since the singleton clusters only contain one instance, our analysis focuses on the flagged clusters. The main results are shown Figure 3.6: around 40% of the flagged clusters are active for less than one month. The long-lasting attacks, however, can stay active anywhere between 2 months to the entire 24 months of our observation timeframe. In general, the average lifespan of a cluster in our database is 128 days. If we look at the actual attack instances on the other hand, we have a bimodal distribution: the 40% of clusters that are active for less than one month only account for about 6% of the attack instances (it is noted that here we consider the instance percentage of the entire phishing dataset, not just the instances in the flagged clusters), while more than 40% of the attack instances are on the other end of the spectrum, in clusters lasting 22 months and more. This shows that, contrary to what was previously assumed, actual attacks can last for a long time through many short-lived attack instances (attack instances are blocked after about 10 hours according to [7]), and the majority of the instances are part of long-lasting attacks. Figure 3.7 shows how often attack instances are observed in the 5,334 flagged clusters in our database. We see that almost half of the flagged clusters average more than one attack instance per two weeks, containing almost 80% of the instances. There is also a small number of clusters with long relaunching period, some even spanning more than one year. It indicates that most attacks stay active for a long time through frequently relaunching new instances, and some attacks are relaunched again after a long period of incubation.

In order to understand how these attacks evolve and survive such a long time, we

[8]Keep in mind that we cannot really tell when the attack observed at the beginning of our observation period actually started, and similarly at the end of the 24 months.

look at the distribution of vectors, IP addresses and second-level domain names[9] over time. Figure 3.8 shows this for two sample clusters: one of the longest lasting ones in our database (711 days) and the one with the most attack instances (4,493 phishing attacks). In this figure, numbers are cumulative, and reuse of a vector/domain/IP is indicated by another dot at the level at which that vector/domain/IP was first observed. As can be seen here, attackers do tend to change domains most often, with IP addresses a close second, but the actual vectors do not change nearly as much. The cluster with the longest lifespan gives us an interesting case that the attack stays active for a long time, accompanied by several incubation periods. If we look at the complete set of flagged clusters, the average ratio IPs/vectors among clusters is 1.91 (maximum is 103) and the average ratio domains/vectors is 2.05 (maximum is 105). This proves that what attackers do when they relaunch an attack is to find a different domain for the attack, and very often a different host for it. But they seldom invest time to modify the site itself. This suggests that it is costlier for them to modify the attack, let alone to create a new one, when compared to using a new domain or a new host.

One of the salient points of this analysis is that a given phishing attack can have a long lifespan, through thousands of "instances" of this attack. In this condition, it is not surprising that trying to protect against actual instances of an attack is not a good, effective strategy. It is a losing game since attackers can simply launch new instances at very low cost. What might be costlier for the attackers is to create new attacks. It is thus unfortunate that most of the prevention techniques in use today are really based on instance-detection. It is also what creates the illusion that the prevention is effective, whereas it is only effective at removing ephemeral attack instances. A much more effective strategy is to target the actual attack itself. In that context, our clustering method is a step in the right direction: once an instance is detected, a cluster is created for the actual attack, and new forthcoming instances will be detected. This suggests that Web browsers should be enhanced with solutions such as ours to be more effective in phishing prevention.

## 3.5 Phishing Sites Detection Tool

To understand why our tool will play an important role in a phishing protection strategy despite the fact that it does not detect new phishing sites but rather new iterations of known ones, one has to consider how a typical realistic corporate in-depth anti-phishing strategy works. In such a context, a steady stream of tens of thousands, if not millions, of

[9] When a country-code domain name is used, we look at the third-level domain names, so `www.example.com` is "example" while `www.CountryExample.co.uk` is "CountryExample".

daily URLs has to be processed to assess and blacklist phishing attacks in near-real-time. This is normally done in two steps: a first level of fast filtering is applied to remove the vast majority of the URLs, which are benign. The resulting, much-smaller set of potential phishing sites is then analyzed using time-consuming methods such as the ones reviewed in Chapter 2. This second step is slow, often requires an actual in-browser rendering of the web sites, and sometimes even requires human intervention. It takes seconds to minutes to process a site at that stage. The number of sites reaching this second step is thus of importance: for the defense to be effective, that number must be as low as possible.

The results of Section 3.4.2 show that today, about at least 90% of the attacks are repeats of previous attacks. It means that at the second step above, the vast majority of the time is spent re-confirming a known attack, whereas a technique such as ours can be used to automatically find many, if not all, of these repeats, with almost no false positives. What is more, our clustering method can assess a site quickly: with non-optimized code, on our database of almost 100,000 URLs, we process a site in an average of 134 milliseconds (resp. 332 milliseconds if we use $WPD$).

Our tool is thus an effective and important intermediary step in the defense strategy: the list of potential attacks that have been flagged by the first step is then analyzed by our method. Based on our results, as it stands about 90% of the actual attacks in that list will automatically be removed (and the corresponding vectors immediately added to our clusters). A much reduced list is then passed to the next and final step, where new phishing attacks are evaluated using slower methods. When a new attack is confirmed at this last step, this information is immediately fed back to our system, and the corresponding cluster is created. From that point on, all the incoming variations of that new attack will be automatically removed and will not reach the last step. Adding that intermediary step can be the difference between a system that can cope with the kind of flow of incoming attacks that we see today and a system that cannot do it.

Attackers can attempt to counter our tool by making it more difficult to detect re-publishing of an attack. Our current approach is not difficult to defeat in that regard, although it currently works well. However, the actual method used to detect resubmission can be adapted, and detecting similarities between DOMs is an active field of research. Even if the percentage of caught repeat attacks decreases, the ones detected will still be a net gain for the defense, and this detection step should therefore be part of the strategy as long as attackers repeat attacks. The ultimate goal of that step is to prevent attackers from relying on existing attacks, and force them to create new ones every time, driving up their cost and reducing their efficiency.

## 3.6 Limitation

Although our results show that the approach we proposed in this chapter performs well at detecting phishing variations, there exist some limitations that could be improved in future work.

First, an attacker might crack our approach by injecting many dummy tags (e.g.,  and <p>) to reduce the similarity between attacks. A solution for this potential attack is to set a weight for each tag. For the dummy tags, we could set a lower weight, therefore, the impact of the change on this tag has little impact to the final similarity metrics. Of course, the challenge of this solution is to find "good" weights for each tag. We can train a machine learning model to set these weights by using instances of the potential attacks.

Another limitation of our approach is that it cannot detect the "new" attacks that have not been seen by the model. One solution for this is, as mentioned in Section 3.5, to add another detection model which looks at characteristics of phishing attacks for a further check. Once a "new" attack is detected by the additional model, it is added into our database, therefore a series of variations of this attack will be blocked by our clustering-based approach

## 3.7 Conclusion

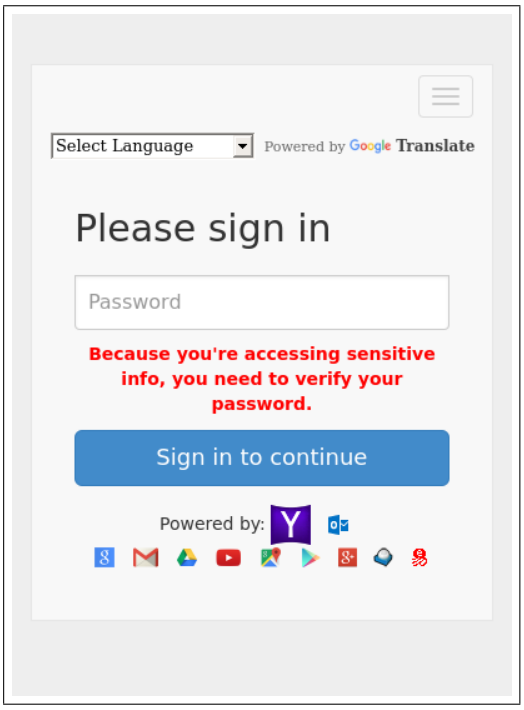
In this chapter, we have demonstrated that a common behavior of attackers using phishing sites is to repeatedly re-publish their attacks, using different domain names, different hosts, and sometimes some minor modifications, probably as a way to get around the very short average lifetime of a given phishing site (about 10 hours according to [7]). This provides an opportunity to detect these "replicas" by comparing new attack instances with previously known ones. We have introduced a clustering based method which computes a vector counting the number times HTML tags are used in the DOM of the sites. We have defined a distance and its improved version between these vectors, and used these distances to cluster together sites with a distance lower than a given threshold. Our clustering method puts together in the same cluster vectors that have at least one other vector from the cluster that is closer than the threshold. These clusters are thus not centroid based, and accommodate long strings of successive modifications. Our method is deterministic, which means that the clusters can be iteratively built over time.

Our tests show that a very large number of newly discovered phishing attack instances today would be caught by our method: about 90% of reported phishing sites are in fact replicas of already known phishing sites, and can be flagged immediately by our tool.

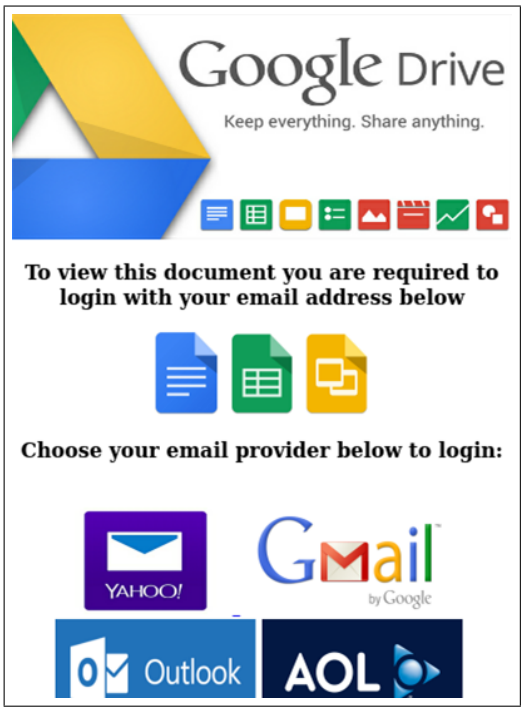
What is more, the level of false positive is very low at 0.65%. This shows that the method presented here is an effective tool to add to a phishing defense strategy, as a preprocessing step to weed-out a large number of new phishing attack instances, dramatically reducing the load on the time-consuming back-end tools in charge of finding and confirming new attacks. This tool is meant as an addition to such a defense mechanism, and not a replacement of the existing tools. It is, however, a necessary improvement to prevent the time-consuming step of new attack detection from being overwhelmed by the sheer volume of new attack instances recorded today. Although our clustering method is today quite effective, there is no doubt that as it gets deployed in corporate environments, attackers will attempt to defeat it by modifying their attack instances more thoroughly. The overall defense scheme will simply have to be adapted with more intelligent ways of detecting different instances of the same attack. The real novelty here is to compare an attack with other known attacks, instead of with the targeted site.

Since our clustering based phishing detection approach requires comparisons with every known attack, the detection time will be linearly increased as more and more attack instances are added to the database. It makes our approach hard to apply to a large scale database. To overcome the above limitation, we propose the cluster prototype which takes the advantage of clustering results to identify the cluster boundary. Our result shows that using the cluster prototype is able to decrease the processing time by up to half.

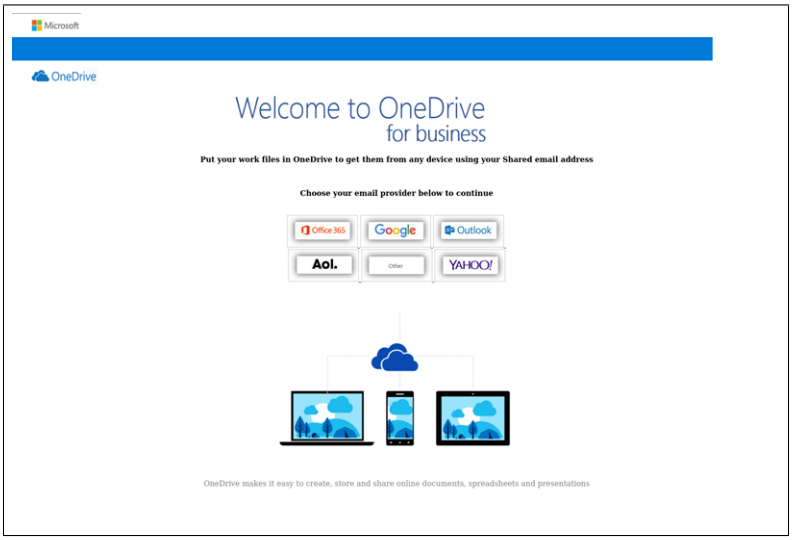
Finally, we have shown that long-lasting attacks survive current prevention methods by re-launching very similar attacks on new domains and on new servers, possibly after some minor variation of the DOM. Current prevention methods aim at identifying and black-listing attack instances, instead of the attacks themselves. This is a losing strategy, easily evaded by the attackers, and a strategy that provides a false sense of efficiency. Instead, moving forward, anti-phishing defense strategies should focus on the actual attacks behind the attack instances, forcing attackers to invest more between each iteration of their attacks. The method presented here is one such strategy. Once an attack instance is identified, the entire attack can be neutralized.




(a) One example of the attack in cluster 7




(b) The attack in cluster 10 targeting Google Drive




(c) The attack in cluster 10 targeting OneDrive

Figure 3.5: Examples in top 10 clusters

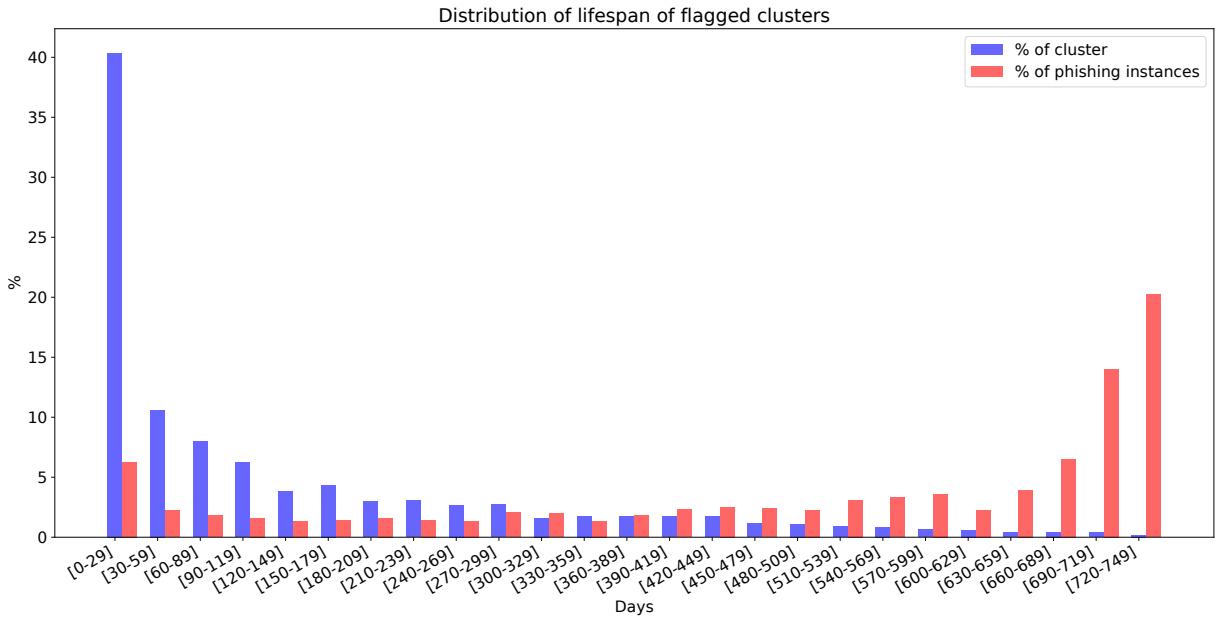



Figure 3.6: Clusters lifespan.

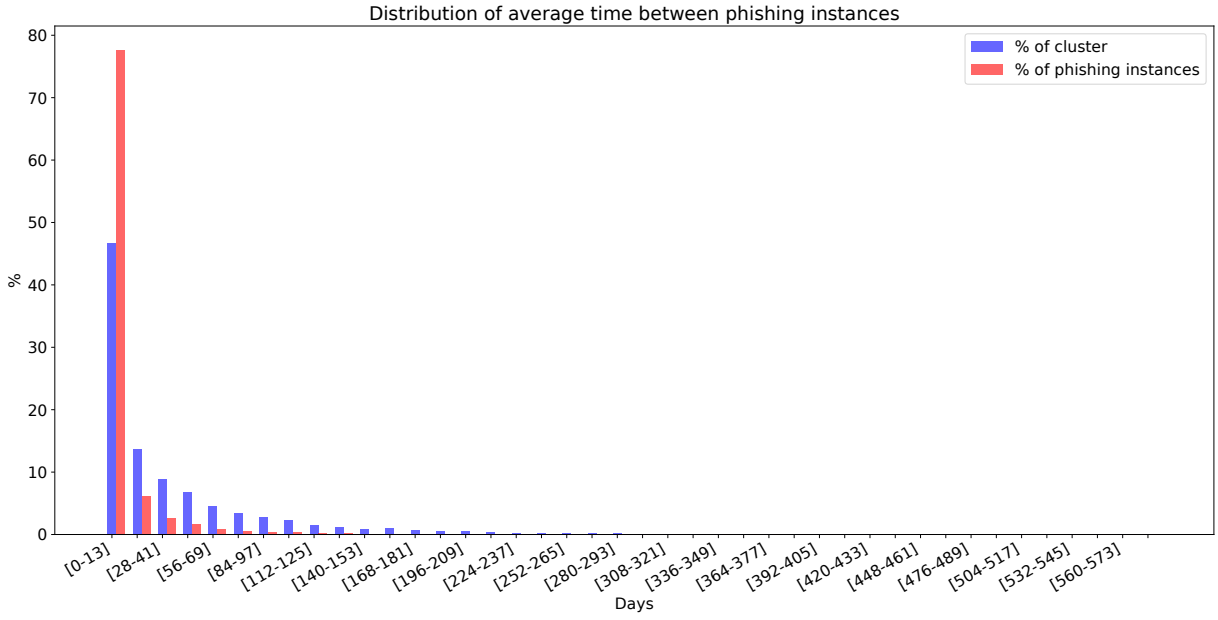
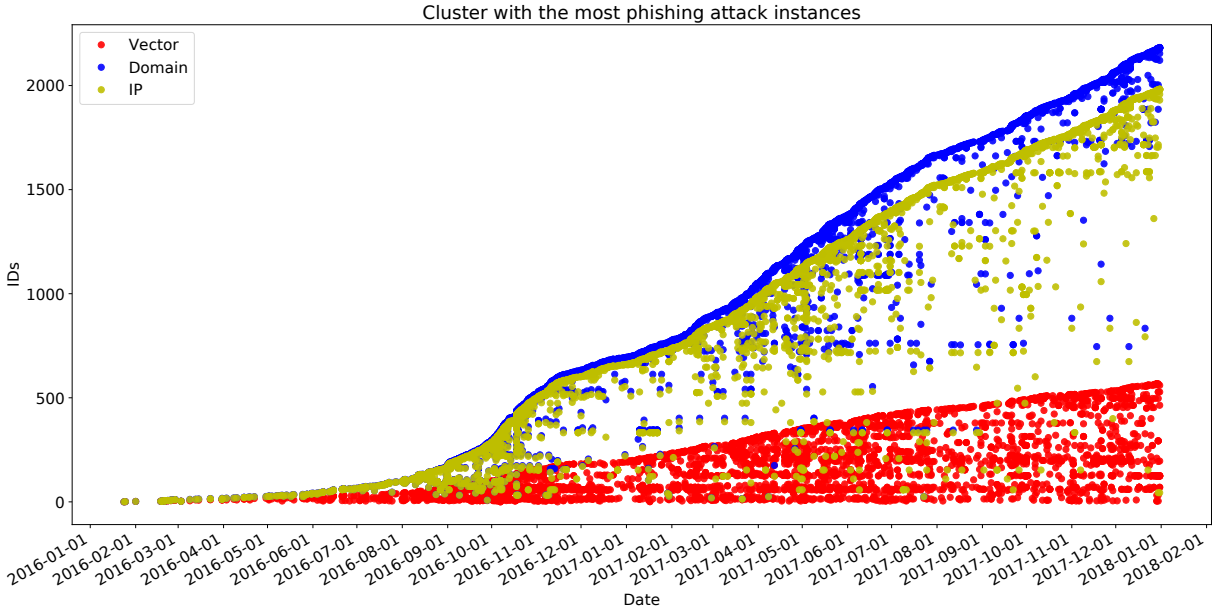
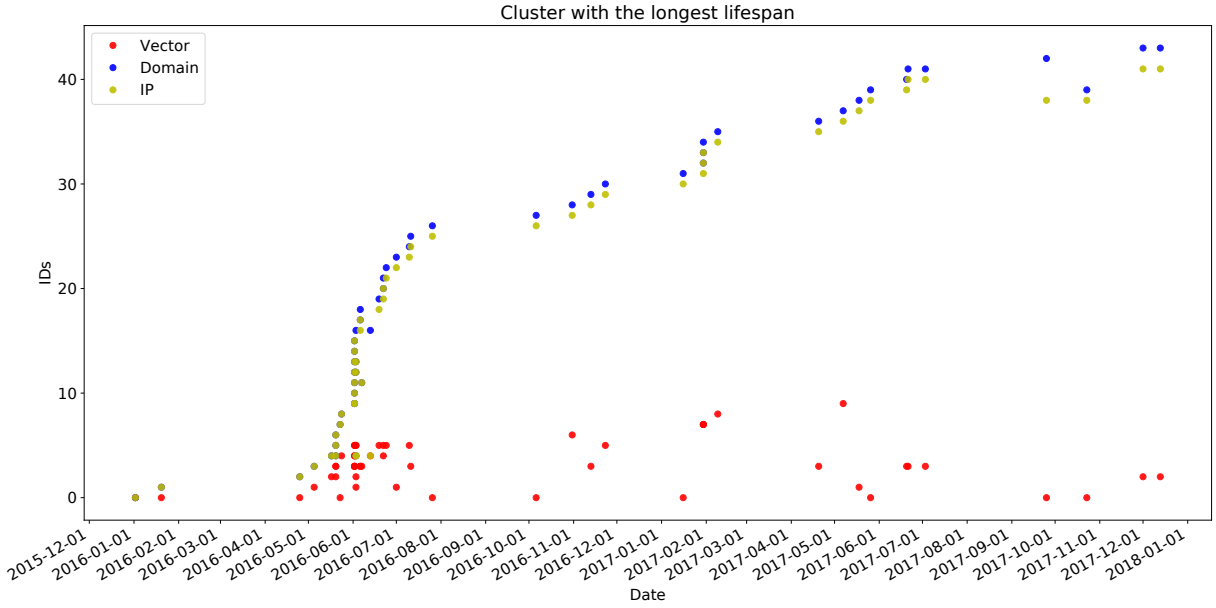



Figure 3.7: Average time between attack instances.




(a) Cluster with the most phishing attack instances




(b) Cluster with longest lifespan

Figure 3.8: Number of vectors, IP addresses and domains used in sample clusters over time.

# Chapter 4

# Analysis of Phishing Modifications and Evolution

## 4.1 Introduction

In Chapter 3, we show that most phishing attacks are repeatedly relaunched after some modifications. In this chapter, we look into the details of these modifications and their evolution over time. We propose a graph based model used to track and analyze the phishing modifications and evolution. We start with a discussion about our proposed graph based model. We then introduce the various mathematical concepts that we have used in our analysis: master vector, evolution path and modified tags. We conclude this chapter by reporting our findings of phishing modifications and evolution.

## 4.2 Phishing Attack Modification Graph

To analyze the evolution of phishing attacks, we first build the $SCL$ model proposed in Section 3.3.3 for each tag vector cluster. We then attach additional information used to analyze the phishing modifications and compose a **phishing attack modification graph**, as illustrated in Figure 4.1. Each node represents a unique tag vector, and the node's label shows the number of phishing attack instances using this vector. The directed edge $E(x, y)$ captures an evolution from vector $x$ to vector $y$, that is, a modification made to the corresponding attack, which transforms the original attack (which has vector $x$) into a slightly different attack (which has vector $y$). The text on the edge provides the details of the modifications. For example, an edge with the label "div:+2, input:-3" should be interpreted as meaning that two *div* tags were added to the attack and three *input* tags

were removed in the creation of the new variation of the attack. The direction of the edge is determined by the reported date of the two connected vectors; the edge flows from the earlier attack to the later one. Since one vector may have multiple attack instances, we consider that the "reported date" of a vector is the date at which we learned of the first attack that produced this vector.

As a consequence of this definition, a source node of the graph, that is, a node that has an in-degree of zero, is the earliest reported attack instance in our data source from this series of modifications. We color these nodes in green. Node that the variations of previously reported attacks have a positive in-degree and are shown in blue in our graph.

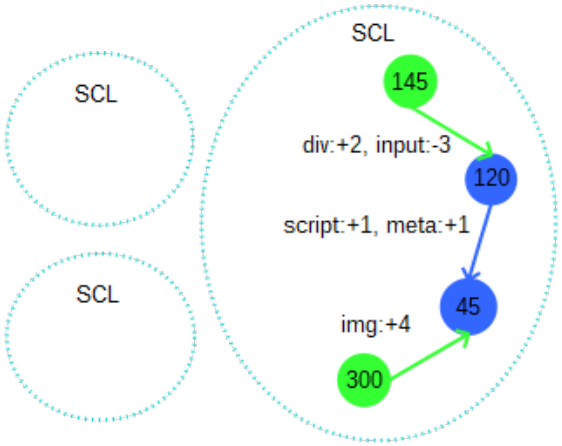
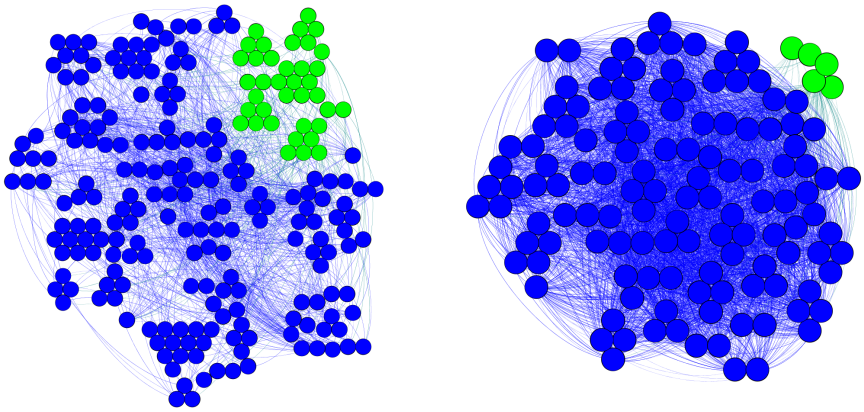



Figure 4.1: Example of phishing attack modification graph

### 4.2.1 Master Vectors

As explained in Section 4.2, the orientation of the edges in the phishing attack modification graph is determined by the reported date of the DOMs creating the tag vectors, from the earlier one to the later one. We call a tag vector of in-degree zero in the phishing attack modification graph a **Master Vector**. A master vector represents one of the initial versions of the attack in our database. Of course, each cluster contains at least one master vector (the earliest reported vector in that cluster), but they can have several ones when the distance between the vectors is too large for them to be connected in the phishing attack modification graph. Having several master vectors in a cluster means that some attacks have been modified at once, or that we are missing intermediate steps in our database. Each non-master vector can be reached from at least one of the master vectors in the cluster. Those master vectors provide a view of the initial attacks and the non-master vectors give a view of how they evolved over time. Figure 4.2 shows the phishing attacks modifications

graphs of the two largest clusters in our database (master vectors in green, non-master vectors in blue). We can see that there are far fewer master vectors than non-master ones, indicating that the majority of attacks in these clusters evolved from the original vectors.



(a) Modification graph of cluster 0 (b) Modification graph of cluster 1

Figure 4.2: Phishing attack modification graphs of two largest clusters

### 4.2.2 Evolution Path and Modified Tags

As explained before, every non-master vector $v$ has at least one directed path in the phishing attack modification graph from a master vector to $v$. We define the **Evolution Path of** $v$ ($EP_v$) as the directed path from a master vector to $v$ for which the sum of weighted proportional distances of the edges along the path is minimal. In other words, $EP_v$ is the directed path from one of the master vectors to $v$ for which the amount of transformation is the smallest.

For a non-master vector $v$ and its evolution path $EP_v = [t_0, t_1, \ldots, t_{k-1}, t_k = v]$ which consists of $k$ vectors, we have the following definitions:

1. The *Path Distance* ($PathDis_v$) is the sum of the weighted proportional distances of the edges along the evolution path $EP_v$. It represents an evaluation of the "amount" of difference between $v$ and its master vector.

$$PathDis_v = \sum_{i=0}^{k-1} (WPD(t_i, t_{i+1})) \tag{4.1}$$

2. The *Evolution Distance* ($ED_v$) is the average weighted proportional distance of the edges along the evolution path $EP_v$. It represents the average "amount" of difference

in each modification. Formally,

$$ED_v = WPD_v/k \tag{4.2}$$

3. The *Variation Lifespan* ($VL_v$) is the time difference between the reported date of $v$ and the reported date of its master vector. It represents the complete length of time during which this attack has been actively modified. If $T_{report}(t_i)$ is the reporting date of vector $t_i$, we have

$$VL_v = T_{report}(t_k) - T_{report}(t_0) \tag{4.3}$$

4. The *Update Interval* ($UI_v$), is the average of the time difference between consecutive vectors along the evolution path $EP_v$. It represents how often modifications are being deployed. Formally,

$$UI_v = VL_v/k \tag{4.4}$$

The evolution path provides a tool to track the modifications between the variation and its "source" attack. In order to analyze the modifications found on the evolution paths, we have the following definitions:

1. The *Modified Tags* ($MT$) is the set of tags used anywhere on an edge of the set of the *Evolution Paths*. These are the tags that have been added or removed to modify attacks.

2. The *Modification Tags Subsets* ($MTS$) are all the subsets of the set of tags used on at least one edge of the set of the *Evolution Paths*. We exclude singletons from $MTS$, so we only consider subsets of at least two tags.

For example, if a phishing attack modification graph has only two edges, one labeled with {div:+1, a:+6} and the other one labeled with {input:+3, a:+5, h2:+1}, the set $MT$ is { , <a>, <input>, <h2>} and we have five subsets in $MTS$, namely { , <a>}, {<input>, <a>, <h2>}, {<input>, <a>}, {<input>, <h2>}, and {<a>, <h2>}.

## 4.3 Analysis Results

### 4.3.1 Phishing Sites Database

We have compiled our phishing database by collecting the URLs of phishing-attack instances from the community-driven portal PhishTank[1] and the enterprise security analysis

[1] `https://www.phishtank.com/`

platform IBM X-Force[2]. A total of 54,575 "verified" phishing sites were collected by fetching the daily archive from PhishTank between January 1st, 2016 and October 31st, 2017 and from IBM X-Force between June 12th, 2017 and October 31st, 2017. For each phishing site, we fetched the DOM, the first URL (the reported one), the final URL (which is different from the first URL only when redirection has been used by the attacker), and a screenshot of the final page. It is noted that the dataset used in this section is a subset of the dataset in Chapter 3 because we only have the information of home pages on phishing servers for this subset dataset. We will discuss the details in Section 4.3.3.

### 4.3.2 Vectors and Clustering Results

To compute the set of tag vectors, we use the same corpus of 107 unique tags as was used in Section 3.2.1. We then count the number of occurrences of each tag in each DOM and use these numbers to create integer vectors of 107 features. We obtain 8,400 unique tag vectors out of the DOMs of our 54,575 phishing attack instances.

| | |
|---|---|
| **Optimal threshold** | 0.24 |
| **# of vectors** | 8,400 |
| **# of multiple-element clusters (flagged)** | 941 |
| **# of single-element clusters** | 3,869 |
| **# of phishing sites in flagged clusters** | 50,706 (92.9%) |

Table 4.1: Vector and clustering results

We apply the improved clustering model, as described in Section 3.3, to build phishing clusters. As shown in Table 4.1, we get the 0.24 optimal threshold and end with 941 flagged clusters which covers 50,706 (92.9%) phishing instances.

### 4.3.3 Who Made Modifications, Phishers or Hosts?

One possible explanation for the modifications we see on different instances of the same attack is that the attack is not actually modified by the attacker, but by the hosting server, which is automatically injecting some HTML into the pages, e.g. some Google Analytics tracking links, some WordPress plugins or some other Javascript libraries. Since a given attack will be hosted on a range of servers, these modifications would be misinterpreted as modifications to the attack itself.

[2]`https://exchange.xforce.ibmcloud.com/`

To verify this, we compare the DOM of the phishing attacks to the DOM of homepages of the server hosting these attacks. We remove all the "blanks" (including \t \r \n \f \v) from both DOMs, and we then extract the content that is common between the two DOMs. This content could have been coming from the hosting server, and not from the attack itself. We do this for all the attack instances in our database for which the host homepage could be reached and has a different tag vector from the attack[3]

We are able to collect the DOMs of 14,584 such homepages[4]. Of these, 2,566 have some common content with the hosted attacks. A closer look at the tags involved in these common contents shows that the tag <**meta**> is involved in 2,280 of these cases, which is not surprising since <**meta**> is used for information such as encoding, page size etc., information usually set by the hosting server. The tag <**script**> is a very distant second present in only 96 cases. This shows that the tag <**meta**> is the only tag for which the hosting server can really impact our results. Therefore, we decided to remove that tag altogether from our tag vectors. Redoing the experiment of Section 4.3.2 without that tag, we find the same optimal threshold (0.24), and end up with 8,290 tag vectors distributed across 913 flagged clusters (cluster with at least 2 vectors) and 3,912 single-vector clusters. Out of an abundance of caution, we use that updated model in the analysis presented in the following sections.

### 4.3.4 Clusters Sample Selection

We apply the phishing attack modification graph discussed in Section 4.2 to our 913 flagged clusters. We observe that there are several clusters with very few edges in their graph, meaning that for these clusters, our database does not contain many variations of the corresponding attacks. Table 4.2 shows a detailed distribution of sizes of the phishing attack modification graphs. Only 46.88% of the clusters have a phishing attack modification graph with two or more edges, but they contain more than 75% of the phishing attack instances. For our study, we select the clusters having a phishing attack modification graph with 30 or more edges because they capture the majority of the phishing attack instances (52%) and they contain the enough variations of attacks for our analysis on their evolution over time.

[3]This excludes attacks that are located right at the homepage of the hosting server.

[4]Many hosting servers were not reachable anymore by the time we did this experiment.

| # of edges in the cluster | # of clusters covered (%-tage of total) | # of pages covered (%-tage of total) | # of edges covered |
|---|---|---|---|
| at least 2 | 428 (46.88%) | 41,229 (75.55%) | 18,636 |
| at least 3 | 394 (43.15%) | 40,579 (74.35%) | 18,568 |
| at least 4 | 258 (28.26%) | 38,059 (69.74%) | 18,160 |
| at least 5 | 243 (26.62%) | 37,321 (68.38%) | 18,100 |
| at least 10 | 150 (16.43%) | 34,539 (63.29%) | 17,504 |
| at least 15 | 107 (11.72%) | 31,638 (57.97%) | 17,043 |
| at least 20 | 88 (9.64%) | 30,797 (56.43%) | 16,732 |
| **at least 30** | 62 (6.79%) | 28,801 (52.77%) | 16,113 |
| at least 40 | 47 (5.15%) | 26,298 (48.19%) | 15,591 |
| at least 50 | 42 (4.60%) | 25,306 (46.37%) | 15,381 |

Table 4.2: Number of edges and pages distribution among phishing attack modification graphs

### 4.3.5 Analysis of Master Vectors

Table 4.3 provides an overview of the master/non-master vectors for all 62 clusters: overall, there are 190 (10.47%) master vectors, covering around 35% of the attack instances. This shows that the master vectors are often reused to relaunch the attacks. Moreover, 34 clusters (54.84%) have two or more master vectors, suggesting several initial versions of the attack which were later merged through a series of updates.

| | |
|---|---|
| **# of clusters** | 62 |
| **# of vectors** | 1814 |
| **# of attack instances** | 28,455 |
| **# of master vectors** | 190 (10.47%) |
| **# of attack instances in master vectors** | 9,855 (34.22%) |
| **# of clusters with two or more master vectors** | 34 (54.84%) |
| **# of clusters with only one master vector** | 28 (45.16%) |

Table 4.3: Overview of master/non-master vectors in the 62 largest clusters.

By manually inspecting the DOMs of master vectors, we found that master vectors can be grouped into three categories: 1) Different initial versions of the attack by attackers, with enough changes to push the distance beyond the threshold. It could be the case that the target is modified or that several new features are released at once. Figure 4.3(a) shows such an example. 2) Different steps of the same attack. Some attacks go through several

steps as they attempt to gather additional information from the victim. For example, in Figure 4.3(b), a first step is used to capture login information, and if it is provided, a second step follows in which credit card details are requested. These different steps are recognized as belonging to the same attack, but the difference between them is too large for the threshold and there is no directed path between them in the $SCL$ graph. 3) Copies of different versions of the target site. As shown in Figure 4.3(c), sometimes the master vectors are essentially copies of the target sites taken at different times. The target site was modified, so the corresponding attack instances do not initially match. It is also possible that in some cases our database is missing an even earlier version of the attack that would yield an initial and sole master vector.

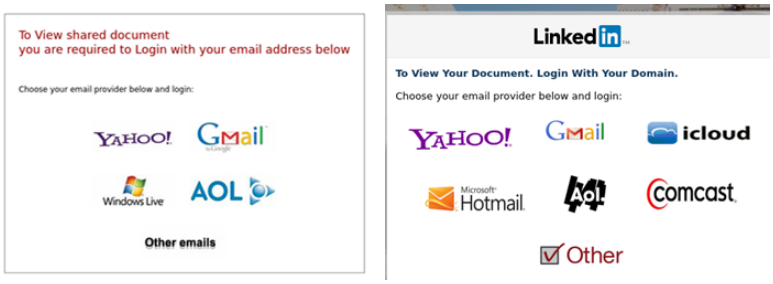
### 4.3.6 Analysis of Variation History

Table 4.4 provides the average values of the attributes discussed in Section 4.2.2 for all evolution paths in the selected 62 clusters. To compute these values, we have not included evolution paths that are included in other, longer evolution paths. The results show that, in general, the attacks are only modified once every six months (186 days) and that the modifications are usually not drastic (the average $WPD$ between these modifications is 0.111). We also see that average path distance is low, only 0.1719. Consequently, the average length of the evolution paths is only $0.1719/0.111 < 2$, less than two edges. This indicates that attackers usually do not maintain long evolution paths to create lots of variations over time. Instead, they tend to re-create new variations from the same master vectors over and over. We also find that each variation tends to stay active for a long time, around nine months (267 days).

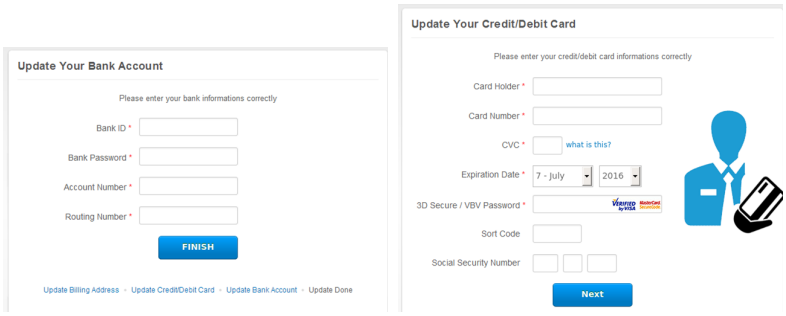
| **# of evolution paths** | 1,230 |
|---|---|
| **Average *Path Distance*** | 0.1719 |
| **Average *Evolution Distance*** | 0.111 |
| **Average *Variation Lifespan*** | 267 days |
| **Average *Update Interval*** | 186 days |

Table 4.4: Analysis of the evolution paths in our database.

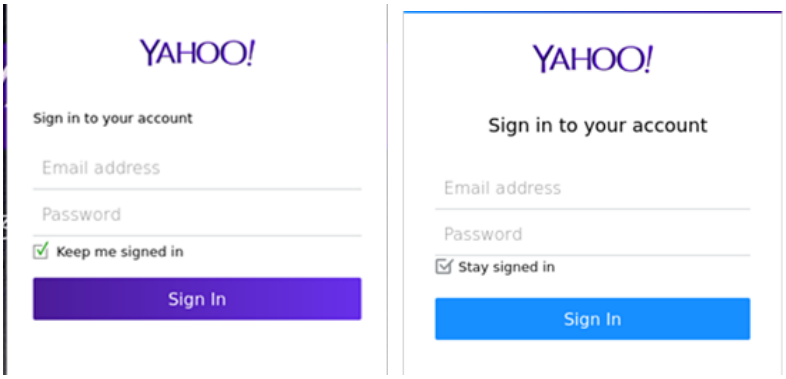
In conclusion, we see that most phishing attack modifications are derived from a small set of master versions. Each of these modifications tend to be reused *as is* for an extended period of time. This behavior matches the "crimeware-as-a-service" model proposed by Sood [58]: The underground producers build the crimeware and sell them to underground buyers who are the ones launching cyber-attacks.




(a) Different versions developed by phishers




(b) Different steps of the same attacks




(c) Different versions copied from legitimate sites (Yahoo login page, circa 2015 and 2016)

Figure 4.3: Examples of master vectors

### 4.3.7 Types of Modifications Seen in Phishing Attacks

In order to understand the modifications that occur on the evolution path, we first analyze the common modifications among clusters. The top 10 most common *Modified Tags* (*MT*), and the number of clusters in which they appear, are <script> (57),  (53), <img> (52), <a> (51), <input> (50), <br> (48), <link> (47),  (47), <p> (41), and <style> (40). The top 10 most common *Modification Tags Subsets* (*MTS*) among the selected 62 clusters are shown in Table 4.5. We found that beside the tags ,  and <br> that are used for spacing or containers, and the functional tags <script>, and

<link> that are used for adding scripts and resources, phishers only use three tags in the top 10 $MTS$: <a>, <img> and <input>. Figure 4.4 shows two examples of (visual) modifications that have only one tag that is actually updated.

| $MTS$ | **# of clusters** | **%** | **# of edges** | **%** |
|---|---|---|---|---|
| {a, div} | 45 | 72.58% | 403 | 24.82% |
| {div, img} | 44 | 70.97% | 286 | 17.61% |
| {div, script} | 44 | 70.97% | 403 | 24.82% |
| {div, span} | 40 | 64.52% | 264 | 16.26% |
| {br, div} | 39 | 62.90% | 215 | 13.24% |
| {img, script} | 39 | 62.90% | 199 | 12.25% |
| {a, img} | 37 | 59.68% | 235 | 14.47% |
| {link, script} | 37 | 59.68% | 215 | 13.24% |
| {script, span} | 35 | 56.45% | 174 | 10.71% |
| {input, span} | 35 | 56.45% | 161 | 9.91% |

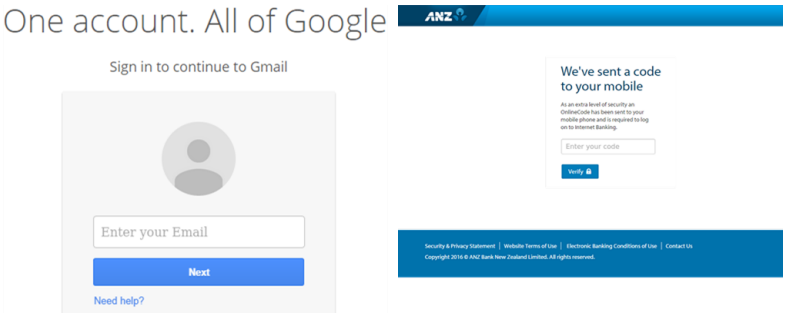
Table 4.5: The top 10 most common $MTS$ in our database.

In Figure 4.4(a), one <img> tag is added to change the target. In Figure 4.4(b), an email credential phishing attack is converted into a tax return phishing page by changing the background images and adding 31 <input> tags.

We also note that despite the very small number of tags used to perform these modifications, none of the top $MTS$ are used by more than 25% of the edges. In order to better understand how common or uncommon each combination of $MTS$ is, we compute the *Jaccard Index*: for each pair of clusters, we computed the number of top 10 $MTS$ (resp. top 10 $MT$) common to both clusters, divided by the number of top 10 $MTS$ (resp. top 10 $MT$) included in either cluster. Figure 4.5 shows the distribution of the values thus obtained.

As shown in Figure 4.5, the distribution of Jaccard Indexes for the pairs of top 10 $MT$ covers a relatively wide range, from 0.1 to 0.7. This indicates that different clusters do use the same tags to create the variations, for example  or <input>. The distribution of Jaccard Indexes for the pairs of $MTS$ on the other hand is very different: most indexes are less than 0.3 and the vast majority (almost 80%) are less than 0.1.

These results show that even though very few tags are actually used when the attacks are modified, the combination of tags used tends to be unique to the attack. In other words, attacks are evolving independently from one another, and the modifications are made for reasons that are specific to each attack, and not as some sort of global update made across a range of attacks.




(a) One <img> tag is added between the left and the right attack


(b) Between the left and the right, 31 <input> tags are added, and the background image is changed

Figure 4.4: Modification of attacks by changing one tag

## 4.4 Threat to Validity

The model we introduced in this chapter is based on our phishing dataset. Therefore, our results may have biases that deviate from true phishing attacks. For instance, our finding of the major vectors may not be the case for real phishing attacks. Also, in actual phishing attacks, the ranking of $MTS$ may vary. What we want to show here is the validity of our model in analysing phishing evolution, and others can apply our model to better databases to track modifications in phishing attacks over time.

## 4.5 Conclusion

In this chapter, we have proposed a graph based model, the **phishing attack modification graph**, to analyze similar phishing attack instances. This model gives us an opportunity to track the evolution of these attacks over time. We discover that the two main reasons for attackers to update their attacks are for aiming at a new target and for

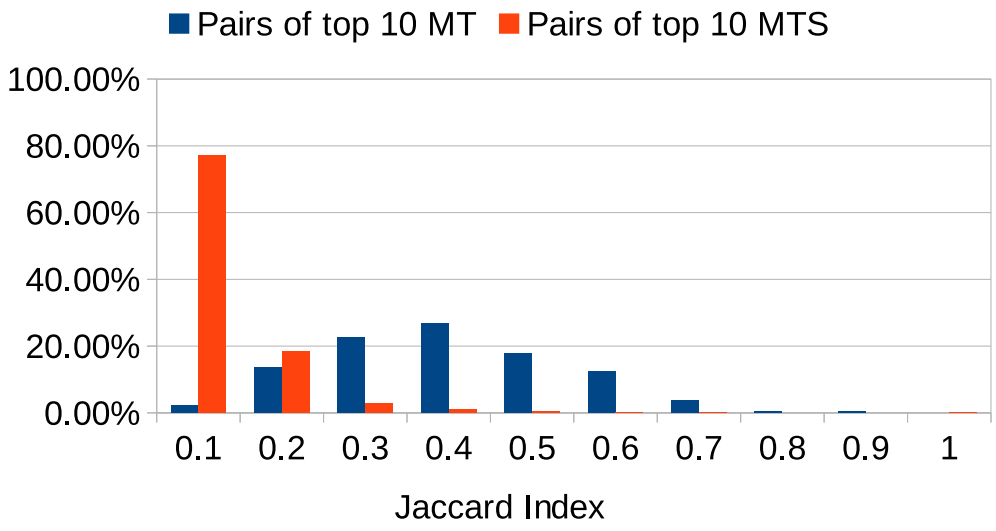



Figure 4.5: Histogram of Jaccard index for top 10 $MT$ and $MTS$.

adding new features, e.g., collecting additional information or improving the interface.

Our analysis shows that most attack instances are derived from a small set of "master" attacks, with only a couple of successive versions being deployed. This shows that attackers do not tend to update and improve a baseline of their attacks, and instead keep reworking from the same base version. This suggests that the phishing ecosystem follows a producers-buyers economic model: the producers build and adapt crimeware and sell them to buyers who launch cyber-attacks but barely update them.

Finally, we have also shown that each attack tends to be modified on its own, independently from other attacks; each cluster of attacks uses its own page template and is improved without a general plan across attacks. This could be because a different attacker is behind each attack, or more likely because attackers follow poor software engineering standards.

# Chapter 5

# BlindPhish: A Blind Scanning System for Phishing Detection

## 5.1 Introduction

Another problem with the current anti-phishing solutions is that they mainly focus on "late" detection, which means there is a delay between an attacker launching an attack and the attack being detected. In other words, an attack can only be detected when the phishing URL is known.

In this chapter, we explore the possibility to use the semantics of domain names, and the semantic differences between domains used to launch phishing attacks and legitimate domain names, to detect sites that are worth monitoring. Note that phishing attacks can be hosted by the attacker on a domain that they own, which is the situation of interest to us here, or they can be hosted on a compromised server, in which case the domain name is not related to the attack. According to [38, 8], a growing number of phishing attacks are hosted by the attacker, and currently represent 38 to 49% of the attacks. To avoid confusion, in the following we refer to a phishing domain owned by the attacker as a malicious domain.

We look at the following research questions:

- **RQ1**: Is there a noticeable difference in the semantics of the malicious domains when compared to legitimate ones?
- **RQ2**: If the answer to RQ1 is true, then can we train a system to distinguish malicious and legitimate domains with a reasonable accuracy?

- **RQ3**: If the answer to RQ2 is true, then can such a system be used to effectively detect some phishing attacks significantly earlier than they are today, in particular before there is any evidence that the attack itself is active?

To answer RQ1, we propose a model which creates vectors based on the semantics of the words used in malicious and legitimate domains, and show that these vectors can be separated by hyperplanes in the vector space.

To answer RQ2, we train a machine-learning model using the vectorization of RQ1 and achieve an accuracy of almost 84% despite working with a noisy dataset.

To answer RQ3, we propose *BlindPhish*, a system that prioritizes the domains that are most likely malicious in order to monitor them. We show that *BlindPhish* can indeed detect attacks without being provided any URL. We also compare our detection results with Google's live update blacklist service: Safe Browsing. Our results show that around 70% of the attacks detected by *BlindPhish* are still not reported by Google Safe Browsing 5 days after *BlindPhish*'s detection.

## 5.2 Domain Words Model (RQ1)

We first answer RQ1: *Is there a noticeable difference in the strings used to create malicious domains when compared to legitimate ones*?

In order to convince a victim to respond to a phishing attack, an attacker not only creates phishing pages that mimic legitimate pages, but often also uses a URL that appears to be legitimate. When the phisher owns the domain name hosting the attack, then that domain can also be used to convey some information about what the site is supposed to be about. When this is the case, attackers usually use one of two strategies when choosing malicious domain names: they can be related to the target of the attack, or they can be related to the content of the attack. The target-related domain uses similar words or homographs of the target domain, e.g., "apple-id.xyz" or "äpple.com". When it comes to the content-related domain name, we can rely on the fact that phishing attacks usually target social network and e-commerce sites. Therefore, words used in phishing domains that are meant to convey the content of the site are biased towards a specific corpus of words which is different from regular websites. This provides an opportunity to identify these phishing domains through the semantics of the words used to construct these domain names.

Our approach is to combine natural language processing and a model created by machine learning to build a word model which is able to "understand" the semantics

of domain names. We proceed in three steps: the domain canonicalization, the words parser, and finally the words vectorization. Figure 5.1 illustrates this with the domain *Updateϵ YourÀccθunt.ga*.

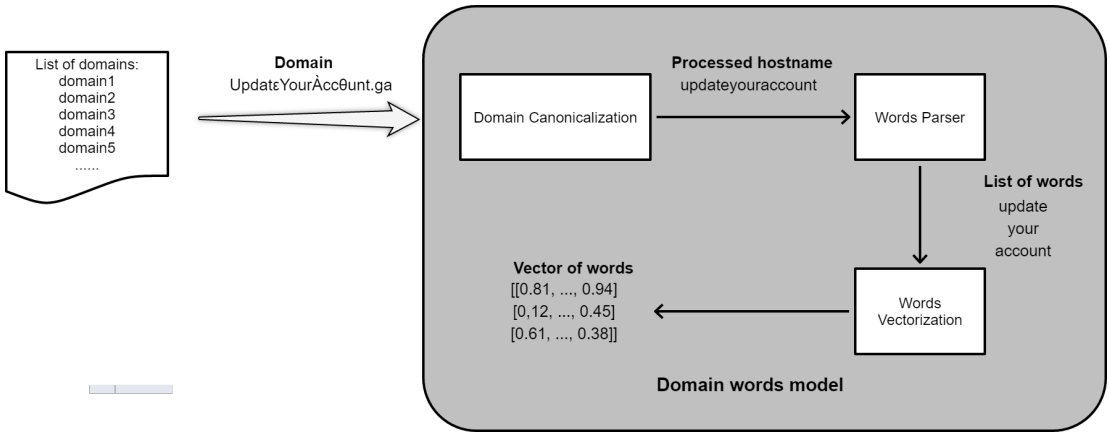



Figure 5.1: Structure of the domain words model

### 5.2.1 Domain Canonicalization

The first step is the *domain canonicalization*. The so-called *homograph attacks* [27] use visually confusing text rendering to create domains that can be easily misread by the victim and confused for something else. In our case, we use the Unicode Technical Report #36 from Unicode Technical Standard[1] to create a character replacement table and obtain a domain name with only digits and English letters. For example, the domain *Updateϵ YourÀccθunt.ga* would be changed to *updateyouraccount.ga*.

We then remove the domain TLD as well as some common sub-domain names, such as www, mail, cpanel, webmail and webdisk, that do not carry meaningful semantics for the analysis. Since some domains are subdomains of common host providers such as myshopify, sharepoint or wordpress, we also remove several of these well-known second level domains. The complete list can be found on our website `http://ssrg.site.uottawa.ca/blindphish`). The resulting text is the input to the **Words Parser**.

### 5.2.2 Words Parser

A straightforward way to extract words from a domain name would be to use a predefined list of separators such as "-" and ".". However, such a method is too restrictive, and would

[1] `http://www.unicode.org/reports/tr36`

fail if the domain name consists of multiple consecutive words. In order to effectively split a string into a list of words, we apply Zipf's law [68] to infer the words position. Specifically, Zipf's law found that occurrence frequency of a word times its rank in a given frequency table is equal to a constant which is linearly related to # of words in the frequency table. Formally, $freq(w) * r_w = aN$, where $r_w$ is the rank of the word $w$ in the frequency table, and $N$ is the number of words in the dictionary. The frequency table is compiled by counting the frequency of words appearing in a large corpus of text and sorting the words in descending order by frequency. As an example, Table 5.1 shows a 10-word frequency table for a dictionary of 10 words ($a = 1$). The higher the rank of a word (lower index) in the table, the higher the occurrence frequency of that word. Once the frequency table is compiled, we use dynamic programming to infer word position. The resulting word list is the one that maximizes the frequency of each individual word.

| **Word** | **Frequency rank** | **Occurrence probability** |
|---|---|---|
| apple | 1 | 10.0 |
| security | 2 | 5 |
| paypal | 3 | 3.3 |
| update | 4 | 2.5 |
| account | 5 | 2.0 |
| your | 6 | 1.6 |
| user | 7 | 1.4 |
| service | 8 | 1.3 |
| verify | 9 | 1.1 |
| lock | 10 | 1.0 |

Table 5.1: Example of a frequency table

We first split the domain name into multiple text segments by using the separator dot (“.”). We then remove non-alphabetic characters in each text segment and apply the above words parser to the extract words. The implementation of our words parser is based on the Python library *WordNinja*[2], which uses a frequency table of over 126,000 “words” created from English Wikipedia. The list of words extracted from each segment is combined as the input to the **Words Vectorization** step.

[2]`https://github.com/keredson/wordninja`

### 5.2.3 Words Vectorization

**Word Embedding** is a technique that converts a word into a high-dimensional vector. In our context, we are interested in the semantic similarity of words. In general, words with similar meanings will be close to each other in the vector space. For instance, the words "car" and "truck" have two similar vectors in the word vector space because they are instances of the same category. Training such a word vector model requires very powerful computing resources and a large collection of texts. We use a pre-trained vector model provided by the Stanford Natural Language Processing Group[3], which has been trained using a large number of texts from the Internet, including 2.2 million words. We use the intersection between the list provided by Stanford and the frequency table of *WordNinja* to convert domain names into a list of word vectors.

Another useful feature of word vectorization in our context is that we can create a vector for a string consisting of multiple words by adding the vectors of each word of that string. This provides us a tool to measure the semantic similarity of domain names by comparing the vectors that we obtain for these domains: vectors that are similar tend to be made from domains that use semantically close words.

### 5.2.4 Analysis of the Model

#### Dataset

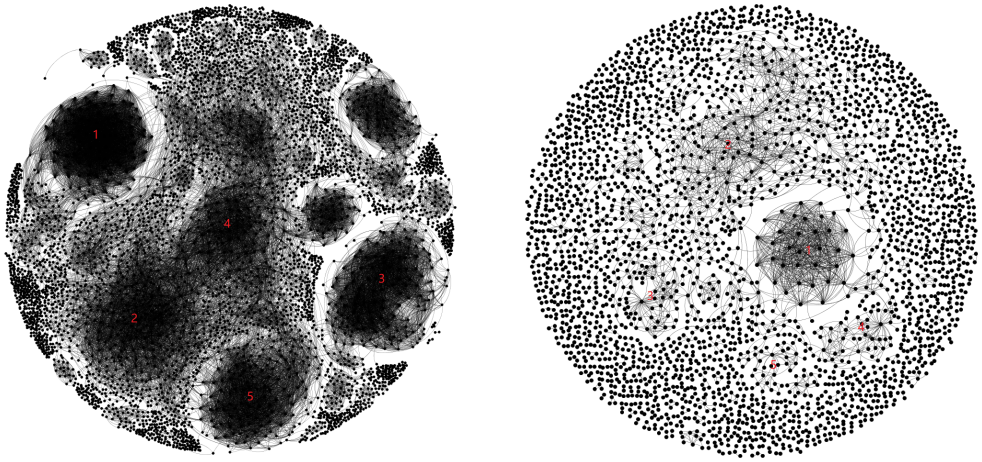
In order to test our model, we need two datasets: a set of malicious domains and a set of legitimate ones. For the list of malicious domains, we use a dataset privately communicated by PhishLabs[4] in the context of [38]. This dataset contains almost 10,000 domain names that have been manually cured by our industry partner and is supposed to contain only malicious domains used in phishing attacks (see [38] for additional details about the construction and cleaning of this dataset). Note that even after our cleaning efforts, this dataset of malicious domains still has some noise and contains a number of legitimate domains.

We created the list of legitimate domain names by randomly selecting domains ranked between 10,000 to 1 million in Alexa Top 1M[5]. After removing duplicates with the same hostname, we ended up with 9,768 malicious domains and 19,976 legitimate ones as our ground truth data. In the following sections, we use this dataset for various experiments.

[3] `https://nlp.stanford.edu/projects/glove/`

[4] `https://www.phishlabs.com/`

[5] `http://s3.amazonaws.com/alexa-static/top-1m.csv.zip`



(a) Malicious domain connection graph (b) Legitimate domain connection graph

Figure 5.2: Domain connection graphs (generated by Gephi using layout ForceAtlas 2)

### Semantic Similarity Analysis

In order to analyze the semantic similarity between domains, we build what we call the **domain connection graph**. Specifically, we use the cosine similarity as the metric to assess the similarity between word vectors. If the absolute value of the cosine similarity between two domains is more than the empirical threshold of 0.8, an edge is built in the graph. We apply this strategy to both of our datasets. If a domain has no connection to any other domain in the dataset, it is dropped from the graph.

The domain connection graphs of both domain datasets are shown in Figure 5.2. The malicious domain connection graph consists of 5,136 domains (52.58% of the malicious dataset), whereas the legitimate domain connection graph only includes 2,992 domains (14.98% of the legitimate dataset). This indicates that it is much more common for malicious domains to have some level of semantic similarity with other malicious domains than it is for legitimate domains. The figure also shows clearly that there are more clusters in the malicious domain connection graph than in the legitimate domain one, which suggests that these domains can be detected by looking for clusters of semantically related malicious domains.

In Table 5.2, we provide a random sampling of domain names for the top five largest clusters in both datasets. It can be seen that the domains in the cluster of the malicious graph have similar semantics (e.g. domain names related to security for the first cluster), even though these domains only share few common words. On the other hand, the largest clusters of the legitimate domain graph are only connected because the domains either use

| Cluster label | Malicious | Legitimate |
|---|---|---|
| 1 | a)service-appteamsupport.support<br>b)macsoftwareinternalstorageappleerrorcodesecurewaringalert.xyz<br>c)securesoftwarestorageinternalwaringalertcode0978.xyz | a)2007l04.com<br>b)l25.ir<br>c)l495b9.com |
| 2 | a)verifyaccount-unlockedsid.tk<br>b)manageaccount.ga<br>c)resolvemyaccount-locked.com | a)mygobe.com<br>b)gowesgo.com<br>c)letgo.cz |
| 3 | a)wellsfargo-43043l33.com<br>b)wells-fargo-profile-l430l023.com<br>c)wellsfargocards.net | a)online4.love<br>b)sudonline.sn<br>c)onlinevsem.ru |
| 4 | a)securitycentre-appleid.com<br>b)applehomesecure.com<br>c)appleid-fraud-operations.com | a)pro.com<br>b)date-pro.com<br>c)tspro.com.br |
| 5 | a)service-account-billing-information.com<br>b)recoveryidinformation.com<br>c)security-informationpayment-apple.com | a)itsfree.club<br>b)gofreeapp.net<br>c)freeexe.net |

Table 5.2: Example domains in labeled clusters

exactly the same word (cluster 2 to 5) or, in the case of the top cluster, because they use only one letter between digits (the letter "l" in the case of that cluster), which causes our vectorization algorithm to group them together.

### Domain Words Analysis

In the above analysis, our results indicate that some of the malicious domains are created based on a set of topics, such as "software security" or "account update". But it is still unclear whether the words used by malicious domains are limited to a narrow range. In order to answer this question, we build a **word-domain** graph to analyze the correlation between words and domains. Specifically, we first sort words based on the number of domain names they cover in decreasing order. During this process, we ignore single characters and stop words. We then draw a cumulative graph to show the correlation between # of words and # of domains covered. The result is shown Figure 5.3. 50% of the malicious domains are covered by only 0.51% of the words in that dataset, while 3.33% of the words are required in the legitimate dataset. The 80% and 90% range is reached after 6.9% and 17.65% of the words in the malicious dataset, while 17.31% and 36.54% of the words are required in the legitimate dataset. If we look at the ratio between the number of domains and the number of the used words, it is 1.85:1 for the malicious dataset, and 1.46:1 for the legitimate dataset.

The results indicate that malicious domains tend to use fewer but more semantically similar words than legitimate domains, and thus the answer to RQ1 is **yes**. It is however still possible that this model can only capture words that have been seen in the training set;

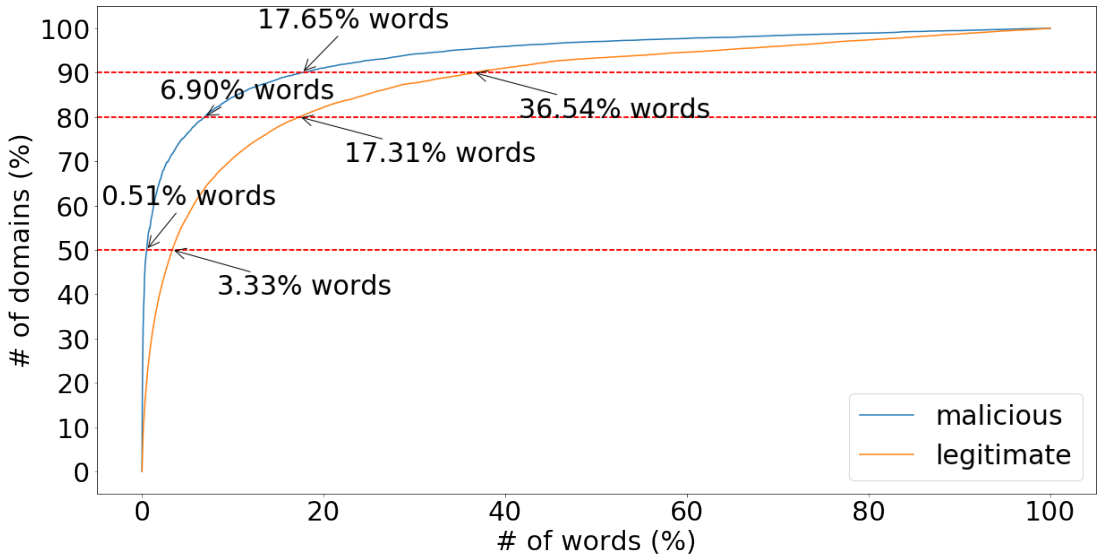



Figure 5.3: Word-domain Graph

that is, words that happen to appear in our ground truth dataset. It is also not clear that even though the two dataset do not have the same behaviour when it comes to semantic similarity, that difference can be used to separate the two sets apart. The latter is the topic of RQ2, which we address next. We will then be able to address the former, using the classifier to create a completely new dataset of malicious domains and show that the set of words in the newly created malicious dataset is quite different from the set of words in the ground truth malicious dataset.

## 5.3 Machine Learning Classifier (RQ2)

We now focus on RQ2: *If the answer to RQ1 is true, then can we train a system to distinguish malicious and legitimate domains with a reasonable accuracy?*

In order to answer this, we have tested three machine learning algorithms that are often used for binary classification: AdaBoost (ADA), Random Forest (RF), and Support Vector Machine (SVM). AdaBoost is a machine learning framework, which combines the results of multiple weak learners (for example, decision trees) into a weighted output, therefore improving the performance of the weak learners. The Random Forest algorithm uses a similar idea: it constructs multiple decision trees during the training process. Each decision tree uses a random subset of features. The model output is a combination of the results of the individual trees (e.g. majority voting, mean value etc.). The goal of the SVM algorithm is to search for the hyperplane that "best" separates the two classes.

Specifically, it first uses a kernel function to project the input data into a high-dimensional space, and then searches for the hyperplane with the maximum distance to the nearest points on each side. Since the output of standard SVMs is a distance to the hyperplane rather than a classification probability, we use **Platt calibration** [51] to transform the distance into a probability over classes. All these machine learning models are implemented by scikit-learn[6] with the default parameters.

- AdaBoost: base_estimator=DecisionTree, n_estimators=50, learning_rate=1, algorithm=SAMME.R
- Random Forest: n_estimators=10, criterion=gini, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0
- SVM: C=1.0, kernel=rbf, degree=3, gamma=auto_deprecated, coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200

We compare the performance of these three models, and list our results in the following section.

### 5.3.1 Classifier Comparison

We first compare the results between the three classifiers. We apply a 4-fold cross validation, and report the average of the results in Table 5.3. It can be seen that the SVM classifier yields the best accuracy (83.96%), and the RF classifier yields the best false positives (1.15%).

| **Model** | **Accuracy** | **False Positive** |
|---|---|---|
| **AdaBoost (ADA)** | 78.01% | 12.66% |
| **Random Forest (RF)** | 81.32% | **1.15%** |
| **Support Vector Machine (SVM)** | **83.96%** | 3.05% |

Table 5.3: Comparison between three classifiers

In order to further compare the performance between these three classifiers, we draw the Receiver Operating Characteristic (ROC) curve, which shows the relation between

[6] `https://scikit-learn.org/stable/index.html`

true positives against the false positives at various threshold, and use the Area Under The Curve (AUC) to evaluate the performance: the greater, the better. The result is shown in Figure 5.4. We can see that the SVM classifier performs better than the other two models, with a relatively high AUC (0.88). In addition to performing better, SVM also provides a more natural interpretation of the score. In our case, for malicious domains, the further away from the hyperlane the sample is, the higher the score. The opposite is true for legitimate domains: the further away from the hyperplane, the lower the score. Therefore, in the following experiments, we use SVM as our classifier.

Note that the results are far from perfect, the best accuracy being only around 84%. This is in fact to be expected: what we are trying to detect is some distinguishing features in the semantics of the words used to create malicious domain names. What we show is that the vast majority of these domains do exhibit such detectable characteristics. Of course, not **all** malicious domains will have these characteristics, and therefore we cannot expect such a classifier to find all malicious domains.

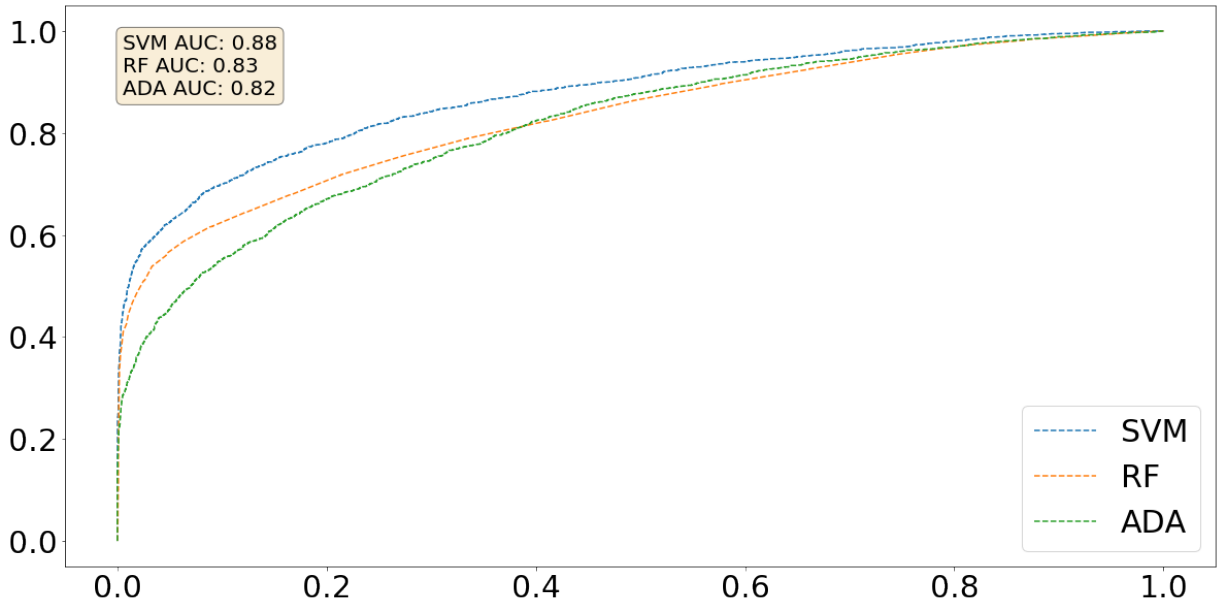



Figure 5.4: ROC of three classifiers

### 5.3.2 Flexibility of the Domain Words Model

As mentioned in Section 5.2.4, it is possible that our model can detect domains that use words appearing in our training set, but fail to recognize words with similar semantics which it has not seen before. In order to find out, we use the dataset in Section 3.4.1

including a list of 98,571 "verified" phishing sites where we end up with 47,876 unique phishing domains.

We applied the SVM classifier obtained from our initial dataset as described in Section 5.3.1 on these 47,876 domains. 9,659 of these domains obtained a score greater than 0.5 and are flagged malicious.

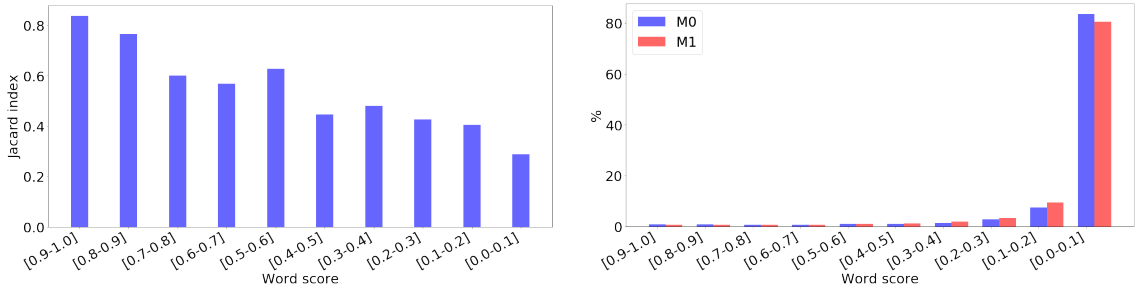
To avoid confusion, we call $M_0$ the set of 9,768 malicious domains used for training (Section 5.3.1) and $M_1$ the set of 9,659 domain obtained here.

We want to see how similar $M_0$ and $M_1$ are. We use two similarity metrics for this: the Jaccard index and the histogram intersection. The Jaccard index is defined as the number of items in the intersection between two sets divided by number of items in the union of the same two sets. The histogram intersection calculates the similarity between two histograms. In our case, it is used for the comparison between the word score distribution graphs of $M_0$ and $M_1$. The word score distribution graph is a histogram where the bar in each bin represents the percentage of words in a range of scores. Our results are shown in Table 5.4 and Figure 5.5.

| **Comparison entry** | $M_0$ | $M_1$ |
|---|---|---|
| **# of domains** | 9,768 | 9,730 |
| **# of words** | 5,274 | 5,717 |
| **Jaccard index of word sets** | 0.32 | |
| **Histogram intersection of word score distribution graph** | 0.97 | |
| **Histogram intersection of word score distribution graph (excluding the range 0-0.1)** | 0.96 | |
| **Histogram intersection of word score distribution graph (excluding the range 0-0.2)** | 0.95 | |

Table 5.4: Comparison result between $M_0$ and $M_1$

Both sets end up with very similar results in terms of number of domains and number of words, but the similarity between the two word sets is quite different, and the Jaccard index is only 0.32. Figure 5.5 (a) shows the details of the Jaccard index comparison. We can see that the two word sets have higher similarity for the words in the high score bin than the words in the low score bin. Specifically, the sets of words with scores greater than 0.8 have a Jaccard index greater than 0.7. This indicates that our semantic-based word model can detect malicious domains that use different words but have similar semantics.




(a) Jaccard index across word score between $M_0$ and $M_1$

(b) Word score distribution of $M_0$ and $M_1$

Figure 5.5: Similarity comparison between $M_0$ and $M_1$

This would not be possible with a keyword-based word model using a predefined list of keywords.

## 5.4 Blind Scanning System: *BlindPhish* (RQ3)

We now turn our attention to RQ3: *If the answer to RQ2 is true, then can such a system be used to effectively detect some phishing attacks significantly earlier than they are today, in particular, before there is any evidence that the attack itself is active?*

### 5.4.1 System Design of *BlindPhish*

To answer RQ3, we propose *BlindPhish*, a system for monitoring and detecting phishing attacks at an early stage. The architecture of *BlindPhish* is shown in Figure 5.6. In addition to the domain words model and machine learning classifier discussed in the previous sections, there are three other components, responsible for crawling, detecting and verifying.

- **Crawling Scheduler**. As discussed in Section 5.3, the output of the classifier is a score between 0 and 1. In order to efficiently scan a large number of domains, we divide the score range into 10 intervals of length 0.1, corresponding to 10 buckets. We distribute the domains into these buckets based on their score. The priority of the bucket is based on the range of scores it covers. The crawling scheduler assigns crawlers to scan each bucket, starting from the bucket with the highest priority. The crawlers simply scan all the domains in the buckets, and move to the next bucket down when finished. Each time a higher priority bucket is updated (new domains are

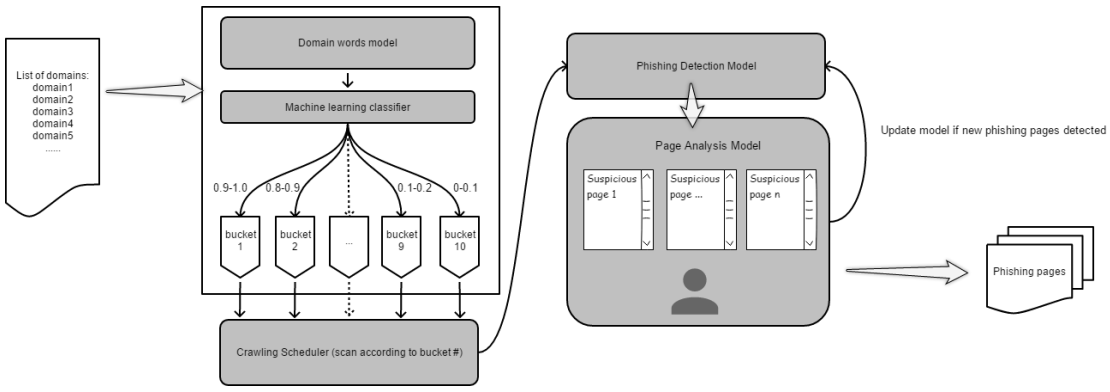



Figure 5.6: Architecture of *BlindPhish*

added), the crawling scheduler sends the crawlers back to that buckets, and the new domains are scanned. The crawlers then resume scanning the lower priority bucket.

For each domain scanned, the crawler attempts to reach the domain's root path using the http and https protocols, and stores the DOM as well as the page screenshot when one is returned. If there are redirects between the root path and the final URL, the crawler logs the redirect path. Finally, the stored page information is then passed to the **Phishing Detection Model**.

- **Phishing Detection Model**. The goal of the phishing detection model is to identify phishing attacks using any existing phishing attack detection approach. In our experiments, we used the approach discussed in Chapter 3 as the detection model. Specifically, we first extract what we call the tag vector. The tag vector simply counts the number of occurrences of each possible HTML tag in the DOM. We then compare the similarity between that vector and a set of similarly constructed vectors obtained from known phishing attacks. If the distance is less than a specific threshold, the page is flagged as phishing. It should be noted that many different detection models can be used instead. The one presented here is chosen because it is fast to compute and easy to deploy.

- **Page Analysis Model** Since the phishing detection model may miss zero-day attacks or yield false positives, we include a manual verification step to our model. In the page analysis model, we list the detailed information of the page that has been scanned, for manual analysis and validation. In order to reduce the duplicates, we group the pages that have the same tag vector. The phishing pages that are validated are output, as marked as confirmed phishing attacks. The phishing detection model is updated if necessary. This manual verification step is optional and can be adapted to the level of trust one has in the phishing detection model. We have used it

systematically here to ensure that the numbers reported in this chapter are accurate.

In *BlindPhish*, there are two models that need training before being used for detection: the machine learning classifier and the phishing detection model. The classifier is trained as explained in Section 5.2.4. The training data for the phishing detection model is the dataset we have used in Section 3.4.1.

### 5.4.2 Evaluation of *BlindPhish*

The goal of *BlindPhish* is to detect phishing attacks early. In order to evaluate the detection efficiency of *BlindPhish* in practice, we apply *BlindPhish* on a real-time stream of domain logs. It is worth noting that the ideal input for *BlindPhish* is a list of the domains that have recently become active, which allows *BlindPhish* to start monitoring and detecting as early as possible. However, we do not have access to such a source. We adopt an alternative and use the certificate transparency logs (CT logs) provided by CertStream[7]. Certificate transparency is an open framework that monitors and audits TLS/SSL certificates in real time[8], and is used by many domain providers. The idea of certificate transparency is that each time a domain updates its certificate, a log about the transaction will be submitted to the certificate transparency network. That network is accessible to domain owners, CAs and domain users. Therefore, the domains that are in CT logs are domains for which a certificate has been issued, which includes domains that have had their certificate renewed, and well as new domains that have an TLS/SSL certificate issued for the first time.

#### Efficiency of Filtering Malicious Domains

One advantage of *BlindPhish* is that it is able to initially filter out potential malicious domains only using domain names rather than collecting other information which would be much slower. To assess the efficiency of filtering malicious domain names, we used *BlindPhish* to track CT logs between January 29, 2019 and February 6, 2019. Since the size of stream CT logs, roughly 1.8 million domains per day at the time, was too large for our server to handle, we randomly sampled around 1% of the logs per hour.

Our results are shown Table 5.5. In total, we have scanned 109,694 domains, and only a few domains fall in high score bins, e.g., 0.23% in 0.9-1.0 range and 0.78% in 0.8-0.9 range. This is completely expected since the vast majority of the domains in CT logs are regular, benign sites and should not have a high score in our domain words model. 66,295 of the

[7] `https://certstream.calidog.io`

[8] `http://www.certificate-transparency.org/`

109,694 domains were reachable by our crawlers at that time. Again, that is not really surprising, since site owners might acquire certificates for their domains before launching their site. Of these, we have 11 verified malicious domains detected, 4 of which fall in the score range 0.8-1.0, which is 2.51% of the domains in that range. The phishing domains with low scores either do not use English words (e.g., scure.anhaengerkalberer.ch or tebcepte.com), or use misspelled words (e.g., welocmweincre.com or deveryservices.gb.net).

This indicates that our *BlindPhish* is able to effectively filter out malicious domains from large-scale domain logs. What is more, it only takes on average 8 milliseconds on our machine to compute the score of a domain. A system such as *BlindPhish* can thus easily be integrated into existing detection systems.

| **Score range** | **# of scanned domains (%)** | **# of reachable domains** | **# of domains verified as phishing (% of reachable domains)** |
|---|---|---|---|
| **0.9 - 1** | 257(0.23%) | 135 | 3 (2.22%) |
| **0.8 - 0.9** | 855(0.78%) | 339 | 1 (0.29%) |
| **0.7 - 0.8** | 2,150(1.96%) | 749 | 0 |
| **0.6 - 0.7** | 3,822(3.48%) | 1,389 | 2(0.14%) |
| **0.5 - 0.6** | 5,828(5.31%) | 2,673 | 1(0.04%) |
| **0.4 - 0.5** | 8,297(7.56%) | 4,050 | 2(0.05%) |
| **0.3 - 0.4** | 12,525(11.42%) | 7,034 | 0 |
| **0.2 - 0.3** | 18,675(17.02%) | 11,106 | 2(0.01%) |
| **0.1 - 0.2** | 26,276(23.95%) | 17,094 | 0 |
| **0 - 0.1** | 31,009(28.27%) | 21,726 | 0 |

Table 5.5: Domain score distribution of sampled CT logs

### Efficiency of Detecting Early Phishing Attacks

Our previous experiment shows that *BlindPhish* is efficient at filtering malicious domains. In this section, we assess the performance of *BlindPhish* in detecting phishing attacks early.

We again tracked the stream of CT logs between May 30, 2019 and June 5, 2019. We only kept the domains with a score of 0.9 or higher, that is, highly suspicious domains. We re-scanned these domains every 6 hours for 5 days in order to detect any change in their status. When changes were detected (e.g. new DOM and new landing URL), we updated our database.

We choose Google Safe Browsing (GSB)[9] as the baseline for our comparison because it is widely used by major browsers, such as Chrome, Firefox and Safari. Since the list of phishing sites provided by GSB is updated dynamically, we verified the list of our domains daily with GSB and collected the domains that were flagged, as well as when they were flagged.

[9] `https://safebrowsing.google.com/`

Our results are shown in Table 5.6. In total, we have scanned 21,131 domains, and 14,546 (68.84%) of them are reachable. Of all the scanned domains, 766 domains are detected as malicious by either GSB or *BlindPhish* or both. *BlindPhish* detects 361 of these domains, **250 of which are not still not flagged by GSB 5 days after *BlindPhish*'s detection**.

When looking at these 250 domains, we find that many of them belong to a series of related phishing domains, and GSB only flags a few of the domains in the series. For instance, we had 72 domains following the pattern webmail-client*xx*.dns05.com, where *xx* is a number between 0 and 100. Only 4 of these 72 domains were flagged by GSB.

We also found series of malicious domains that use the same hostname but different TLD. Again, only a few of them were flagged by GSB. These results indicate that the attacker tends to deploy phishing attacks on multiple domains using similar names. *BlindPhish* is able to flag such phishing attacks at an early stage: whenever the phishing domain appears in the monitoring log, the attack starts being monitored and can be flagged as soon as the phishing website is activated.

For the 516 domains reported to be malicious by GSB, 182 domains were never reachable by our crawlers throughout the experiment and we were thus unable to confirm or refute GSB's verdict. Another 223 domains were not confirmed as malicious by *BlindPhish*: these domains either host blank pages or are parked with advertisement pages. We cannot tell if these are incorrect verdicts from GSB, or if these domains were indeed phishing attacks that were already taken down by the time we reached them (e.g. some malicious domains may have already been detected or blocked before the certificate was issued for example). The remaining 111 domains are identified as malicious domains by both GSB and *BlindPhish*. 31 of them are flagged by GSB one day or more after being first detected by *BlindPhish*. Our results show that *BlindPhish* can indeed detect phishing attacks days before they are detected by GSB, at a very early stage. Adding a system such as *BlindPhish* to the series of detection tools being used can greatly reduce the lifespan of phishing attacks.

## 5.5 Limitation

Although our results show that *BlindPhish* has good performance at detecting phishing attacks at early stage, there are several limitations that could be improved in future work.

- As we have shown in Section 5.4.2, *BlindPhish* cannot effectively detect the domain names created using non-English words. This problem can be solved by using a word corpus in other languages.

| | |
|---|---|
| **# scanned domains** | 21,131 |
| **...# reachable domains** | 14,546 |
| **# total malicious domains** | 766 |
| **Malicious domains detected by *BlindPhish*** | |
| **# detected domains (manually confirmed)** | 361 |
| **...# detected malicious domains not reported by GSB** | 250 |
| **Malicious domains detected by GSB** | |
| **# detected malicious domains** | 516 |
| **...# unreachable domains** | 182 |
| **...# domains not confirmed by *BlindPhish*** | 223 |
| **...# domains flagged by both *BlindPhish* and GSB** | 111 |
| **...# domains detected at least 24h before GSB by *BlindPhish*** | 31 |

Table 5.6: Detection result comparison between *BlindPhish* and GSB

- Our current approach only works for the phishing domains that are created by the attackers rather than the compromised ones. It is a limitation of our domain ranking model but not the blind scanning system. Other methods that are used to detect compromised domains [38] can be integrated into our system system to fill the gap.
- *BlindPhish* currently only explores the root path of each suspicious domains. This prevents our model from detecting attacks that are hosted somewhere else. In future work, we plan to attempt scanning other paths, using known common locations for other attacks.
- A better domain source can help further improve system performance. Instead of using CT logs, we could use real-time domain logs from DNS servers, or even from registrars.
- In our experiments, we choose a 5-day observation period, and find that some suspicious domains have a longer incubation period, and become active after this observation period. Therefore, a longer monitoring period may help capture more attacks.

## 5.6 Conclusion

In this chapter, we have proposed a blind scanning system, *BlindPhish*, to detect phishing attacks at an early stage. This system gives us an opportunity to track and monitor phishing activities before an attacker launches an attack. In order to effectively identify suspicious hosts that likely host phishing attacks, we have proposed a domain ranking

model by analyzing the semantics of the domain name. Since this ranking model uses only a list of domain names as input, and does not require some time-consuming processing, such as crawling pages and collecting domain information, it allows our system to be easily integrated into existing systems.

Our results show that compared with legitimate domains, domains created for phishing attacks typically use smaller but more semantically similar word sets. This is due to the fact that the goal of these phishing domains is to deceive victims by imitating the domain name of a well-known website or using words that express warnings. What's more, we have observed that attackers often use multiple similar domain names to launch the same attack. Our model can effectively block these phishing domains.

Finally, through comparing with Google Safe Browsing (GSB), we have also demonstrated that *BlindPhish* can monitor and detect phishing attacks at an early stage. Specifically, around 70% of the attacks identified by *BlindPhish* are not reported by GSB within 5 days. Even for the remaining phishing domains that are detected by GSB and *BlindPhish*, 28% are reported by GSB more than one day after being detected by *BlindPhish*. Our dataset used in this chapter can be found on our website `http://ssrg.site.uottawa.ca/blindphish`.

# Chapter 6

# DeepPK: a Deep-Learning Approach for Detection of Phishing Kit Traffic

## 6.1 Introduction

As discussed in Chapter 2, most of the literature on anti-phishing focuses on either detecting phishing web pages or on detecting phishing emails sent to 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. This is the topic of this chapter, and more specifically the channel through which the attacker collects the stolen information from the phishing site. 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 chapter, we look at detecting exfiltrating emails using a deep-learning approach, which is called **DeepPK**, and compare it with two different machine learning techniques that are provided by IBM. We compare these models, and test their robustness against potential attacks showing that detecting these messages is possible and effective. Our results show that **DeepPK** is the one that is the overall best since it 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.

In order to train and test our models on a realistic database, we used a synthetic exfiltration email dataset produced by our industry partner. This dataset contains almost 65,000 messages generated from real phishing kits. For the regular email dataset, we compile our dataset from the **Enron email dataset**[1], which contains about 0.5 million messages coming from 150 users.

## 6.2 Methodology

### 6.2.1 Structure-Based Detection Model

Exfiltrating emails tend to follow a specific format that is rarely used in regular emails. For instance, they are often organized following the format: <header> + <field name> + <delimiter> + <value>. 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[18, 10, 60]; therefore we attempt a deep-learning classifier to distinguish exfiltrating emails from regular emails. We first preprocess the messages using a **structure token** explained below.

#### 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 "C2SC6LSC4SC2SC4S2C6SC4SC2SC4S" (where single instances of a category where the number 1 is omitted are 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

[1] `http://www.cs.cmu.edu/~enron/`

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 6.1 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.

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[2] 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 index of one of the symbol 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]].

### 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 6.2 shows one such example.

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 [55]. Formally, given a string $S$

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

| | |
|---|---|
| ———=F3dreport 2018=———— | ———=F3dreport 2018=———— |
| Em@il: victim1@gmail.com | Em@il: victim2@hotmal.com |
| pass: victim1pass | pass: victim2test11 |
| pass2: victim1pass2 | pass2: victim2test11 |
| ———=IP Address & Date=——— | ———=IP Address & Date=——— |
| IP Address: 123.123.123.222 | IP Address: 123.123.12.12 |
| Country: Unknown | Country: Unknown |
| Date: 2018-12-14 04:16:11 | Date: 2018-12-12 01:23:19 |

Figure 6.1: Two instances of the exfiltrating emails generated from the same template

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:

Figure 6.2: One example of false positives

consisting of $n$ characters $\{c_1, c_2, ..., c_n\}$ that are generated by a corpus of $k$ unique symbols, the entropy of $S$, $ent(S) = -\sum_{i=1}^{m} p(s_i) * log_2(p(s_i))$, 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 $S$. As an example, if the word corpus contains only "a", which thus as a probability of occurrence of 1, then the entropy of the string "aaa" is 0. On the other hand, if the word corpus contains only "a", "b" and "c", each with a probability of occurrence of 1/3, then the entropy of the string "aaa" is 1.58. 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]: $ent_{normal}(s) = -\sum_{i=1}^{m} \frac{p(s_i)*log_2(p(s_i))}{log_2(m)}$.

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 we 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 Figure 6.1 are 0.7065 (left) and 0.7097 (right), while the text proportion of the regular email in Figure 6.2 is 0.7703, higher than Figure 6.1.

### Long Short-term Memory Model

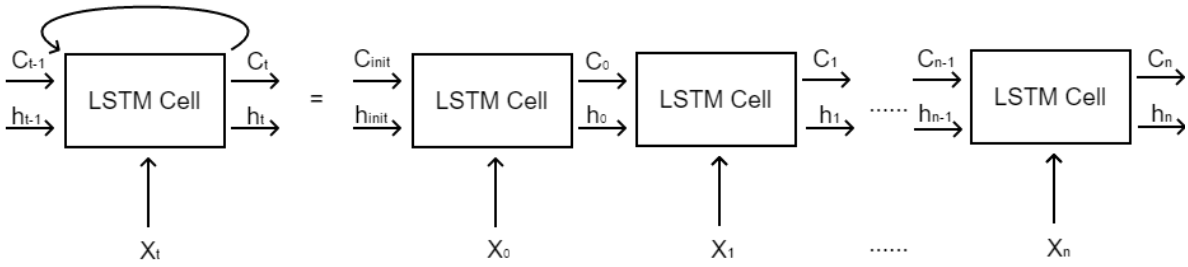



Figure 6.3: LSTM cell and its unrolled form

A Recurrent Neural Network (RNN) is a neural network where cells are connected in a round-robin fashion. 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 [12, 43]. The Long Short-Term Memory (LSTM) model is a type of RNN which has been proved to perform well in dealing with complex patterns and long sequences [33,

59]. As shown in Figure 6.3, a 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 come from an exfiltrating email.

### Detection Model

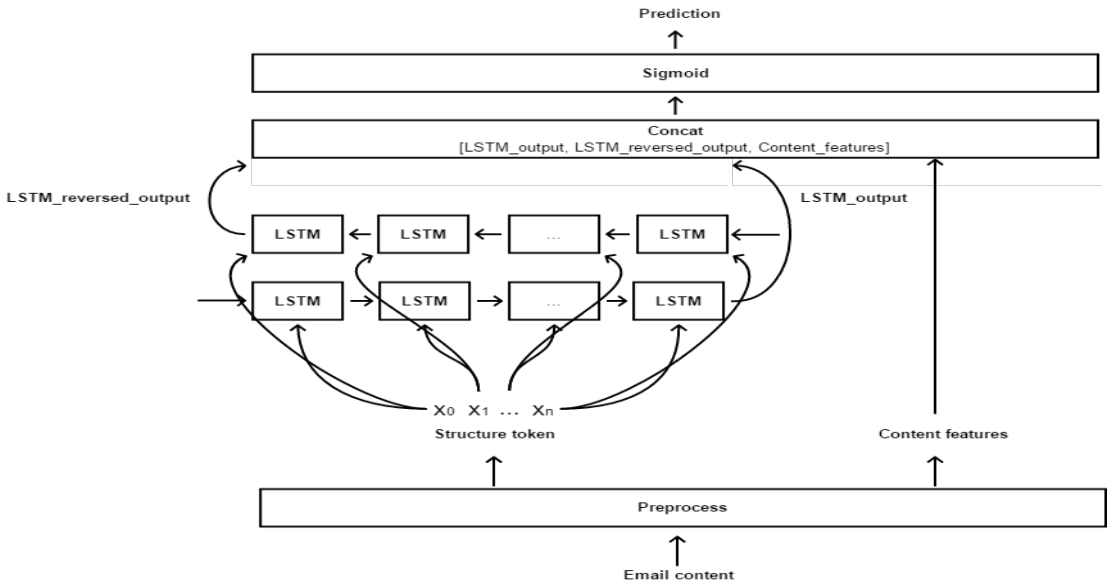



Figure 6.4: System design of DeepPK

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 about 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 [54]. Specifically, the additional LSTM could help the model to extract certain useful from the beginning of the input sequence. We call this detection model **DeepPK**. The complete overview is shown in Figure 6.4.

- **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 an 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, the 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 Section6.3.3). 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 tokens 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 to 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 joined together with the semantic features to form a new feature vector, which is later used as the input to 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 an exfiltrating email if and only if one of its token segments is detected as such.

#### Injection on Training Set

As discussed in Section 6.2.1, 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 begining. 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 fragments that are sampled from the regular training set.

### 6.2.2 Baseline Models

In order to assess the performance of **DeepPK**, we also include two baseline models provided by IBM as our classifiers, and use them to compare with **DeepPK**. In this section, we give a brief description about these baseline models.

#### Word-Based Detection Model

Naive Bayes approaches have been shown to be very successful in the text classification task [65, 11]. Therefore, we have 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, the model first extracts consecutive alphanumeric characters using the regexp [0-9A-Za-z] to get a "word" list. It then applies 1-gram and 2-gram to create word tokens. The corpus of the model is built using the 5,000 most frequent tokens. It applies 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 $1 + \log(\#$ of occurrences of $w_i$ in the document $d_j)$. It then applies tf-idf using the scaled tf to vectorize the tokens. For vector normalization, it applies an "L2" normalization: the sum of squares of vector elements is 1. Finally, for each document (email), the model ends 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 $p(C_k)\prod_{i=1}^{n} p(x_i|C_k)$. The model applies 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 $\sigma_{C_k}$ and $\mu_{C_k}$ are learned 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**.

### Pattern-Based Detection Model

In addition to using a different set of words (when compared to regular emails), exfiltrating emails tend to use some specific patterns that are rarely used in regular emails. Therefore, we also include a model to look for patterns. This model first encodes 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. The model then computes all n-grams of length 10 to 16 on the encoded email sets, exfiltrating emails and regular emails, and it keeps 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. The model derives 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**.

## 6.3 Experiment

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

### 6.3.1 Experiment Environment

We have developed DeepPK using Keras[3] with Tensorflow as the backend. For HTML email preprocessing, we use Beautifulsoup[4] to extract the text from the HTML emails. The baseline 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 3.5Ghz and 16GB RAM. DeepPK is trained and tested on an NVIDIA Geforce GTX 1060 with 6GB RAM. Our source code can be found on our website `http://ssrg.site.uottawa.ca/phishing_kit/`.

### 6.3.2 Exfiltration Email and Regular Email Database

We obtained our regular email database from the **Enron email dataset**, which 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, is provided by IBM (10 instances per template).

We split our 64,480 exfiltration emails into two sets at a ratio of 4:1: 51,580 instances are used for training, and the remaining 12,900 instances are used for testing. For the regular email database, we create a balanced training set by randomly sampling 51,580

[3] `https://keras.io/`

[4] `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`

[5] `https://scikit-learn.org/stable/`

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 an attempt to have a more realistic scenario in our tests, since exfiltration emails would be a fraction of the mail traffic in reality. Of course, in real life the ratio would be much smaller, so the situation is still not representative of reality.

As described in Section 6.2.1, 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 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 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. The breakdown of our dataset is given Table 6.1.

Table 6.1: Experiment dataset

| **Data type** | **Training** | | **Testing** |
|---|---|---|---|
| | **Training set** | **Validation set** | **Test set** |
| **# of exfiltration emails** | 41,260 | 10,320 | 12,900 |
| **# of regular emails** | 41,260 | 10,320 | 64,500 |

### 6.3.3 Analysis of Structure Token Length

As discussed in Section 6.2.1, 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 Figure 6.5.

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

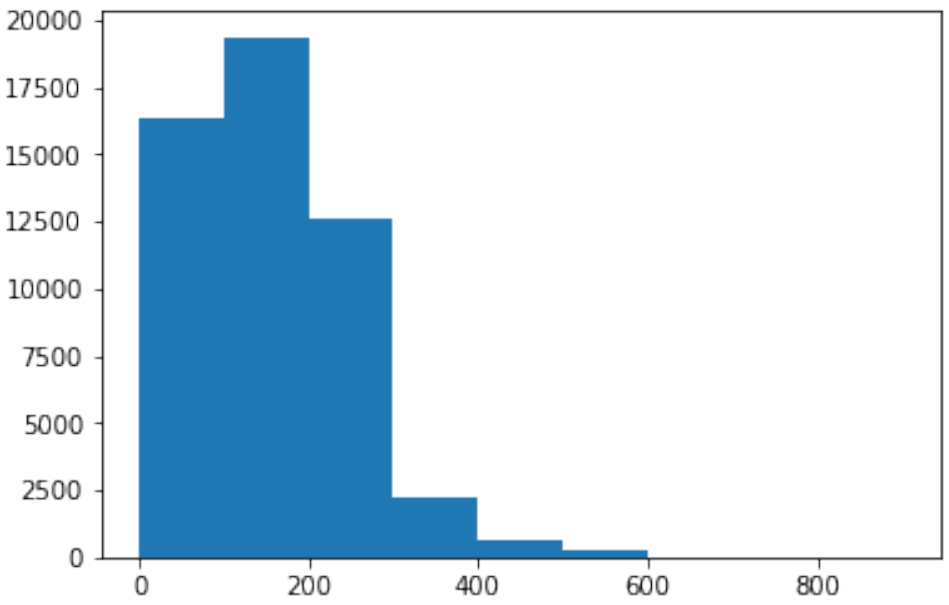



Figure 6.5: Distribution of structure token length in the phishing database (x-axis: token length, y-axis: # of exfiltrating emails)

shown in Figure 6.6. 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 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 instances that exceed this length, the cropped part is often a repeat of the previous part.

### 6.3.4 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 posi-

[6]In the actual exfiltration email, the data is where the "**" are in the figure.

```
========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]:**[13]:**[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]:**
============================
```

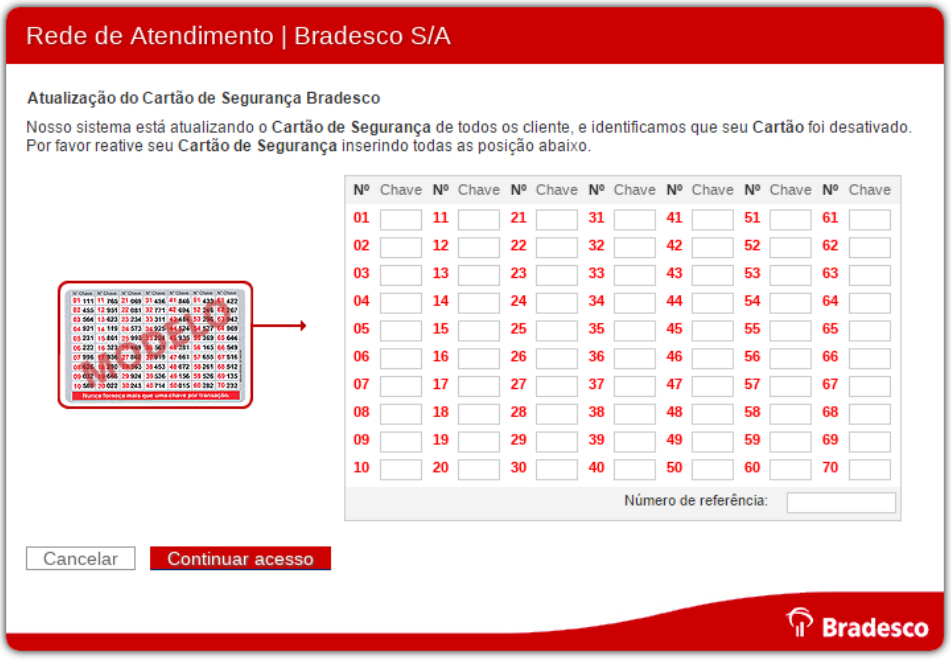



Figure 6.6: Email template with long structure tag and its screenshot[6]

tive[7] (FP), false negative (FN), precision=$\frac{TP}{TP+FP}$ (TP stands for true positive), recall=$\frac{TP}{TP+FN}$ and f-score=$\frac{2*precision*recall}{precision+recall}$. The results are shown Table 6.2.

For the NB model, we note that using a tumbling window improves the false negative rate but at the expense of the false positive rate. For DeepPK, the model that only uses a single LSTM yields the best false negative rate 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

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

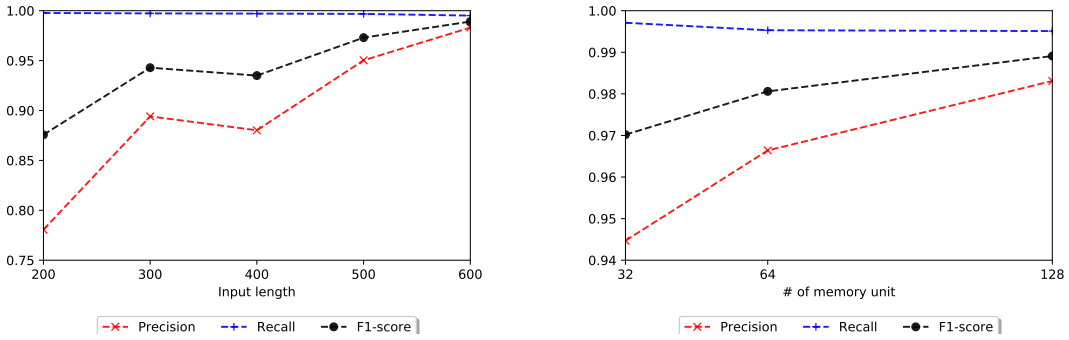
Table 6.2: Performance comparison between models

| 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%** |

emails with a structure similar to that of the exfiltrating emails (e.g. the case shown in Figure 6.2). In general, the model which uses bidirectional LSTM and semantic features yields the best false positive rate (0.34%) and the best F1 score (98.91%) across all models.

### 6.3.5 Analysis of DeepPK

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




(a) Input length with 128 memory units (b) # of memory units with 600 input length

Figure 6.7: DeepPK performance with different parameters

Our results are shown Figure 6.7. 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.

## 6.4 Model Robustness

Our results in the above section 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. 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 injections 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 different structure token 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 Section 6.3.2. 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 6.3 shows a summary of the attack test sets.

Table 6.3: 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 words |
| **Replace_non_word** | Replace non-words with randomly generated non-words |
| **Replace_all** | Word and non-word replacement |

The comparison of model performance against injection attacks is shown in Figure 6.8. 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 a 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.

The comparison of model performance against replacement attacks is shown in Figure 6.9. It shows that the word replacement attack has almost no effect on 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%.

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 they are received would also be much harder than the current situation. Therefore, this attack, however effective, seems of limited practicality.

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 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 default to an "exfiltrating emails" 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 words.

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 performance.

## 6.5 Limitations and Conclusion

Of course, our detection method can be defeated by encrypting the content of the messages. While not terribly difficult, doing so will still be a step up from a technical viewpoint, and would impact the ability of the phisher to easily collect victim's data while sorting through the vast amount of useless data which are typically pushed through phishing sites. 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 up their game and start encrypting their messages. When that time comes, a new solution will have to be found, e.g. several approaches have been proposed to work on encrypted traffic by comparing the traffic pattern going to the same destination [5, 34, 2].

Another possible criticism of 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 their 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.

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 combat phishing campaigns by focusing on exfiltrating emails instead of on the original phishing email or the detection of the phishing site. This method has the advantage of working very well in our experiments, and would be extremely effective at stopping an attack immediately, and completely prevent harvesting of any data by the attacker. We also introduce a new "structure token" which proves to be very effective when combined with our deep learning algorithm. The combination is shown to be resistant to several evasion techniques.

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.

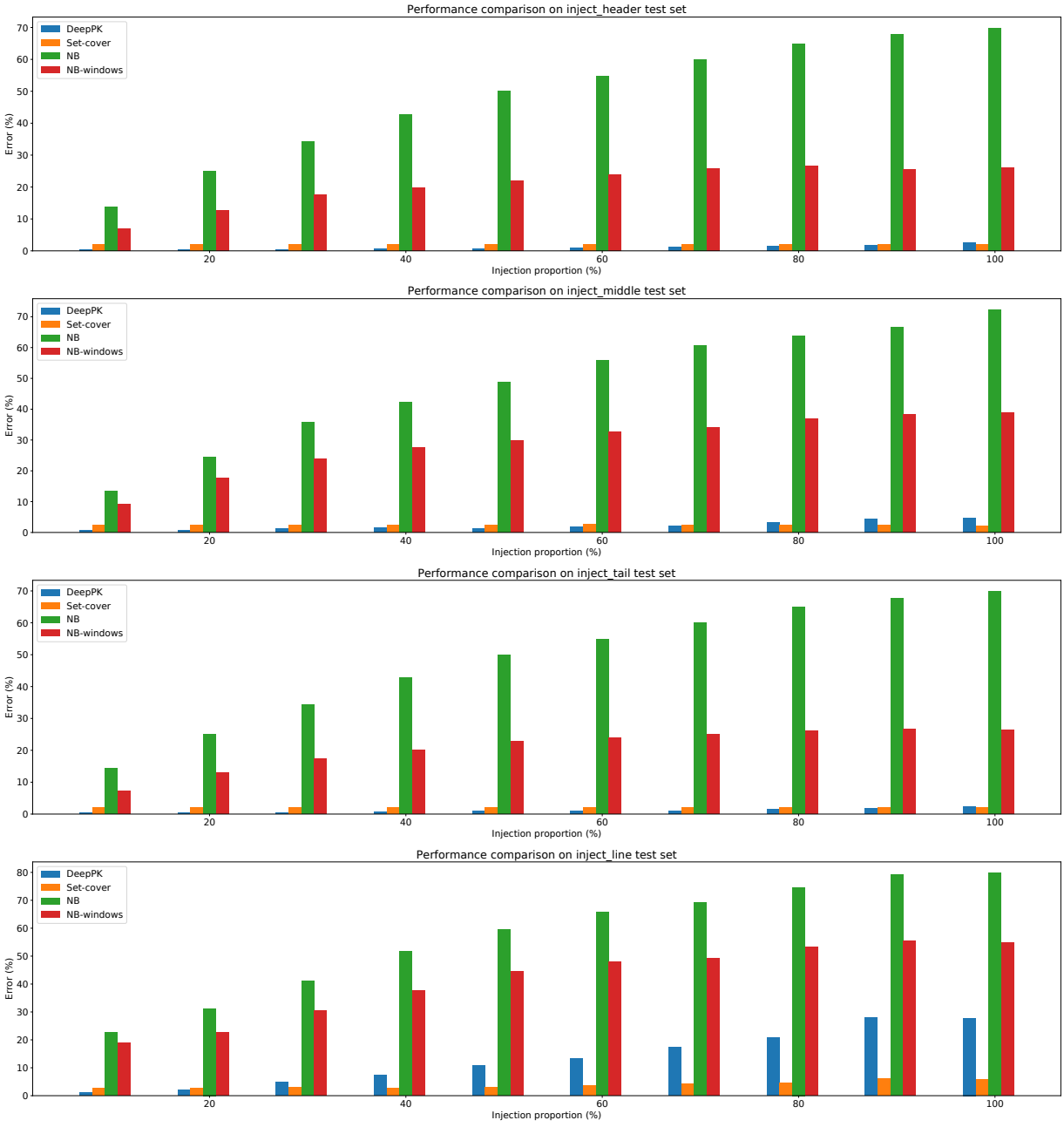



Figure 6.8: Performance comparison on injection attack test sets

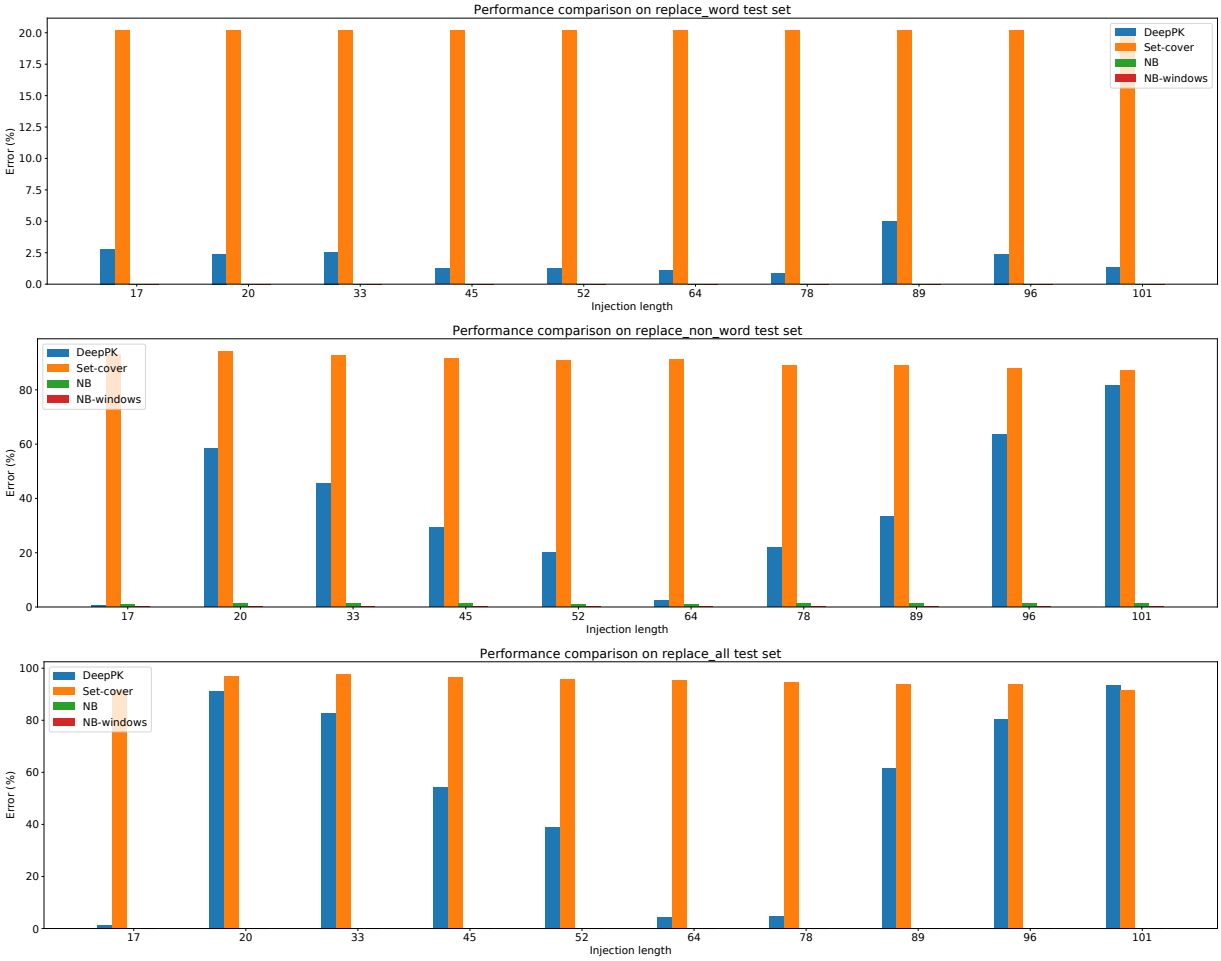



Figure 6.9: Performance comparison on replacement attack test sets

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

In this thesis, we have proposed several methods to detect and to analyze phishing attacks. In general, the work in this thesis can be divided into two categories: the client-side of phishing detection and the server-side of phishing detection.

For the client-side of phishing detection, we have proposed a clustering-based method to detect phishing variations. Our results show that a common behaviour of phishing attackers is to repeatedly republish their attacks, using different domain names and different hosts with minor modifications. Our clustering method can effectively detect phishing variations by comparing the site being checked with a set of known phishing attacks. In order to improve detection performance, we have proposed a new similarity metric and clustering connection model, that can better represent modifications between phishing attacks. What's more, we have proposed a new comparison algorithm which uses clustering results to reduce the comparison time. In order to better understand the details of how phishing variations evolve over time, we have proposed a graph-based tool for monitoring phishing evolution.

In order to detect phishing attacks at an early stage, we have proposed a blind scanning system, which only uses a list of domain names as input. The system is used to screen out candidate hosts that may host phishing attacks by analyzing the semantics of domain names. Our results show that compared with existing detection techniques, such as Google Safe Browsing, our system can effectively detect early phishing attacks once the phishing host becomes active.

For the server-side of phishing detection, we have proposed a method to detect phishing attacks by monitoring outgoing network traffic. To this end, our proposed method is to

train a machine learning classifier that is able to identify the specific patterns of outgoing traffic that is phishing. Our results show that our method can effectively identify phishing attacks and is robust to several potential evasion techniques.

## 7.2 Future Work

### 7.2.1 More Similarity Metrics for Detecting Phishing Variations

In this thesis, we have proposed two metrics based on the tag frequency in the DOM to compare the similarity between phishing attacks. Although these two methods can detect phishing variations very well at present, the attacker may attempt to crack them in the future, such as injecting multiple hidden tags. Therefore, proposing metrics with better robustness can be considered as a future direction of this research.

### 7.2.2 Larger Dataset for Comparison

The experiments conducted in this thesis are based on a dataset we have collected from multiple sources, such as PhishTank and IBM X-Force. One limitation of this dataset is that there may exist a bias in these phishing attack sources specific to regions or targets, which may result in our conclusion not representing phishing attacks globally. A comparison with our experiments on a larger dataset which covers a wider range of phishing attacks will provide a better understanding of phishing attacks.

### 7.2.3 Comparison with Other Machine Learning Models

In the study of detecting exfiltration emails, we have proposed three machine learning models: the Naive Bayes model, the set-cover model and the LSTM model, and compared their performance and robustness. Although three proposed methods perform well on the test set, their performance will be greatly reduced for some potential attacks. Using other machine learning algorithms that have been proven successful in text identifying tasks is a future direction of this work.

### 7.2.4 Improvements of Phishing Blind Scanning System

There are several improvements that can be done for the research of blindly scanning phishing attacks. First, a better domain source can help further improve system perfor-

mance. In our experiments, we have used a source of real-time CT logs as the input to our system. However, this source keeps updates with the domains that recently update certificates rather than the newly registered domains. An ideal source for our system is real-time domain logs from DNS servers, which allows our system to start monitoring at an earlier stage. Second, our current model cannot effectively detect domain names created using non-English words. In future work, we plan to train and evaluate our model using a word corpus in other languages. Third, our current model only detects the root path of each suspicious domain. We plan to scan other paths using common locations of known attacks. Last but not least, in our experiments, we choose a 5-day observation period, and find that some suspicious domains have a longer incubation period, and become active after this observation period. Therefore, a longer monitoring period may help capture more potential attacks.

# Appendix A

# Proof the "proportional distance" is a Mathematical Distance

*Proof.* Recall that we have for all non-zero vectors $t_1$ and $t_2$ of size $n$

$$PD(t_1, t_2) = \frac{D(t_1, t_2)}{L(t_1, t_2)} = \frac{\sum_{i=1}^{n} D(t_1[i], t_2[i])}{\sum_{i=1}^{n} L(t_1[i], t_2[i])}$$

where, for all integers $x$ and $y$, we have

$$D(x, y) = \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{if } x = y \end{cases}$$

and

$$L(x, y) = \begin{cases} 1 & \text{if } x \neq 0 \text{ OR } y \neq 0 \\ 0 & \text{if } x = 0 \text{ AND } y = 0 \end{cases}$$

We need to show that the following properties hold:

1. $\forall t, PD(t, t) = 0$
2. $\forall t_1, t_2, PD(t_1, t_2) = PD(t_2, t_1) \geq 0$
3. $\forall t_1, t_2, t_3, PD(t_1, t_2) + PD(t_2, t_3) \geq PD(t_1, t_3)$

The first two properties are immediate consequences of the definition. Also, it can be easily shown that $D$ is a mathematical distance (in fact, it is the well known "Hamming Distance"), thus $\forall t_1, t_2, t_3, D(t_1, t_2) + D(t_2, t_3) \geq D(t_1, t_3)$.

Note that if two vectors $t_1$ and $t_2$ have common zeros ($\exists i$ such that $t_1[i] = t_2[i] = 0$), we can remove these indexes from the vector and obtain two vectors $t_1^*$ and $t_2^*$ such that $PD(t_1, t_2) = PD(t_1^*, t_2^*)$. Thus, in the following, we can assume that $t_1, t_2$ and $t_3$ do not have common zeros, otherwise we simply remove the corresponding indexes from all three vectors.

Let's first assume that $t_1$ and $t_3$ have no common zeros, that is, $L(t_1, t_3) = |t_1| = n$. We have $\forall t_i, t_j, L(t_i, t_j) \leq |t_i| = n$, thus $\frac{D(t_1,t_2)}{L(t_1,t_2)} + \frac{D(t_2,t_3)}{L(t_2,t_3)} \geq \frac{D(t_1,t_2)}{n} + \frac{D(t_2,t_3)}{n}$. Moreover, $D(t_1, t_2) + D(t_2, t_3) \geq D(t_1, t_3) \Rightarrow \frac{D(t_1,t_2)}{n} + \frac{D(t_2,t_3)}{n} \geq \frac{D(t_1,t_3)}{n}$, and thus $\frac{D(t_1,t_2)}{L(t_1,t_2)} + \frac{D(t_2,t_3)}{L(t_2,t_3)} \geq \frac{D(t_1,t_3)}{n} = \frac{D(t_1,t_3)}{L(t_1,t_3)}$.

Let's now assume that $t_1$ and $t_3$ have $k$ common zeros, that is, $L(t_1, t_3) = n - k$ and $0 < D(t_1, t_3) \leq n - k$. Let $t_i'$ be the vector obtained from $t_i$ where the $k$ indexes corresponding to the common zeros between $t_1$ and $t_3$ have been removed.

If $t_1$ and $t_2$ have no common zeros, and $t_2$ and $t_3$ have no common zeros, that is, $L(t_1, t_2) = L(t_2, t_3) = n$, we have $D(t_1, t_2) = D(t_1', t_2') + k$, $D(t_2, t_3) = D(t_2', t_3') + k$ and $D(t_1, t_3) = D(t_1', t_3')$. Since $D(t_1', t_2') + D(t_2', t_3') \geq D(t_1', t_3')$, we have $D(t_1, t_2) + D(t_2, t_3) \geq D(t_1, t_3) + 2k$, and thus $PD(t_1, t_2) + PD(t_2, t_3) = \frac{D(t_1,t_2)}{n} + \frac{D(t_2,t_3)}{n} \geq \frac{D(t_1,t_3)+2k}{n}$. Assume $\frac{D(t_1,t_3)+2k}{n} < PD(t_1, t_3) = \frac{D(t_1,t_3)}{n-k}$, then $(n - k)(D(t_1, t_3) + 2k) < nD(t_1, t_3)$, and thus $2nk - 2k^2 < kD(t_1, t_3)$, that is $2(n - k) < D(t_1, t_3)$, which contradicts $0 < D(t_1, t_3) \leq n - k$. Thus, if $t_1$ and $t_2$ have no common zeros, and $t_2$ and $t_3$ have no common zeros, $PD(t_1, t_2) + PD(t_2, t_3) \geq \frac{D(t_1,t_3)+2k}{n} \geq PD(t_1, t_3)$.

If $t_1$ and $t_2$ do have common zeros and/or $t_2$ and $t_3$ do have common zeros, let $v$ be a random positive integer that does not appear in $t_1, t_2$ or $t_3$, and let's replace every common zero between $t_1$ and $t_2$ and every common zero between $t_2$ and $t_3$ by $v$, thus defining three new vectors $t_1'', t_2''$ and $t_3''$. Since $t_1''$ and $t_2''$ have no common zeros, and $t_2''$ and $t_3''$ have no common zeros, we have shown previously that $PD(t_1'', t_2'') + PD(t_2'', t_3'') \geq PD(t_1'', t_3'')$. We have $D(t_1, t_2) = D(t_1'', t_2'')$, $D(t_2, t_3) = D(t_2'', t_3'')$ and $D(t_1, t_3) = D(t_1'', t_3'')$, and we have $L(t_1, t_2) \leq L(t_1'', t_2'')$, $L(t_2, t_3) \leq L(t_2'', t_3'')$ and since $t_1, t_2$ and $t_3$ do not have common zeros, the replaced common zeros between $t_1$ and $t_2$ and $t_2$ and $t_3$ do not impact the common zeros between $t_1$ and $t_3$, and thus $L(t_1, t_3) = L(t_1'', t_3'')$, and $PD(t_1, t_2) + PD(t_2, t_3) \geq PD(t_1'', t_2'') + PD(t_2'', t_3'') \geq PD(t_1'', t_3'') = PD(t_1, t_3)$. □

# References


[1] Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. A comparison of machine learning techniques for phishing detection. In *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 60–69. ACM, 2007.

[2] F. Al-Obeidat and E-SM El-Alfy. Hybrid multicriteria fuzzy classification of network traffic patterns, anomalies, and protocols. *Personal and Ubiquitous Computing*, pages 1–15, 2017.

[3] Alexa. Top 500 Sites in Each Country. `http://www.alexa.com/topsites/countries`.

[4] Paul D. Allison. Convergence failures in logistic regression. In *SAS Global Forum*, volume 360, pages 1–11, 2008.

[5] Riyad Alshammari and A. Nur Zincir-Heywood. Machine learning based encrypted traffic classification: Identifying ssh and skype. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–8. IEEE, 2009.

[6] Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D. Spyropoulos, and Panagiotis Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *arXiv preprint cs/0009009*, 2000.

[7] Anti-Phishing Working Group. Global Phishing Report 2H 2014. `http://docs.apwg.org/reports/APWG_Global_Phishing_Report_2H_2014.pdf`.

[8] Anti-Phishing Working Group. Global Phishing Survey: Trends and Domain Name Use in 2016. `http://docs.apwg.org/reports/APWG_Global_Phishing_Report_2015-2016.pdf`, 2017.

[9] Simone Aonzo, Alessio Merlo, Giulio Tavella, and Yanick Fratantonio. Phishing attacks on modern android. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18. ACM, 2018.

[10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[11] Mohammad Behdad, Luigi Barone, Mohammed Bennamoun, and Tim French. Nature-inspired techniques in the context of fraud detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1273–1290, 2012.

[12] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[13] Giovanni Bottazzi, Emiliano Casalicchio, Davide Cingolani, Fabio Marturana, and Marco Piu. Mp-shield: a framework for phishing detection in mobile devices. In *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, pages 1977–1983. IEEE, 2015.

[14] Madhusudhanan Chandrasekaran, Krishnan Narayanan, and Shambhu Upadhyaya. Phishing email detection based on structural properties. In *NYS cyber security conference*, volume 3. Albany, New York, 2006.

[15] Ee Hung Chang, Kang Leng Chiew, San Nah Sze, and Wei King Tiong. Phishing detection via identification of website identity. In *2013 International Conference on IT Convergence and Security, ICITCS 2013*, pages 1–4. IEEE, 2013.

[16] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[17] Teh-Chung Chen, Scott Dick, and James Miller. Detecting visually similar web pages: Application to phishing detection. *ACM Trans. Internet Technol.*, 10(2):5:1–5:38, June 2010.

[18] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[19] William W. Cohen. Learning rules that classify e-mail. In *AAAI spring symposium on machine learning in information access*, volume 18, page 25. California, 1996.

[20] Igino Corona, Battista Biggio, Matteo Contini, Luca Piras, Roberto Corda, Mauro Mereu, Guido Mureddu, Davide Ariu, and Fabio Roli. Deltaphish: Detecting phishing webpages in compromised websites. In *European Symposium on Research in Computer Security*, pages 370–388. Springer, 2017.

[21] Marco Cova, Christopher Kruegel, and Giovanni Vigna. There Is No Free Phish: An Analysis of "Free" and Live Phishing Kits. In *2nd Conference on USENIX Workshop on Offensive Technologies (WOOT)*, volume 8, pages 1–8, San Jose, CA, 2008.

[22] Elisabeth Crawford, Irena Koprinska, and Jon Patrick. Phrases and feature selection in e-mail classification. In *ADCS*, pages 59–62, 2004.

[23] Bin Cui, Anirban Mondal, Jialie Shen, Gao Cong, and Kian-Lee Tan. On effective e-mail classification via neural networks. In *International Conference on Database and Expert Systems Applications*, pages 85–94. Springer, 2005.

[24] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 symposium on Usable privacy and security*, pages 77–88. ACM, 2005.

[25] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web*, pages 649–656. ACM, 2007.

[26] Anthony Y. Fu, Liu Wenyin, and Xiaotie Deng. Detecting phishing web pages with visual similarity assessment based on earth mover's distance (emd). *IEEE transactions on dependable and secure computing*, 3(4):301–311, 2006.

[27] Evgeniy Gabrilovich and Alex Gontmakher. The homograph attack. *Communications of the ACM*, 45(2):128, 2002.

[28] Sujata Garera, Niels Provos, Monica Chew, and Aviel D. Rubin. A framework for detection and measurement of phishing attacks. In *Proceedings of the 2007 ACM workshop on Recurring Malcode - WORM '07*, pages 1–8, New York, NY, 2007. ACM.

[29] Guang Gang Geng, Xiao Dong Lee, Wei Wang, and Shian Shyong Tseng. Favicon - a clue to phishing sites detection. In *eCrime Researchers Summit (eCRS), 2013*, pages 1–10, Sept 2013.

[30] R. Gowtham and Ilango Krishnamurthi. A comprehensive and efficacious architecture for detecting phishing webpages. *Computers & Security*, 40:23–37, 2014.

[31] Isredza Rahmi A. Hamid and Jemal Abawajy. Hybrid feature selection for phishing email detection. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 266–275. Springer, 2011.

[32] Xiao Han, Nizar Kheir, and Davide Balzarotti. Phisheye: Live monitoring of sandboxed phishing kits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1402–1413. ACM, 2016.

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

[34] Martin Husák, Milan Čermák, Tomáš Jirsík, and Pavel Čeleda. Https traffic analysis and client identification using passive ssl/tls fingerprinting. *EURASIP Journal on Information Security*, 2016(1):6, 2016.

[35] Danesh Irani, Steve Webb, Jonathon Giffin, and Calton Pu. Evolutionary study of phishing. In *ECrime Researchers Summit, 2008*, pages 1–10. IEEE, 2008.

[36] Mahmoud Khonji, Youssef Iraqi, and Andrew Jones. Lexical url analysis for discriminating phishing and legitimate websites. In *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference*, pages 109–115. ACM, 2011.

[37] Sophie Le Page, Guy-Vincent Jourdan, Gregor V. Bochmann, Jason Flood, and Iosif-Viorel Onut. Using url shorteners to compare phishing and malware attacks. In *APWG Symposium on Electronic Crime Research (eCrime), 2018*, pages 1–13. IEEE, 2018.

[38] Sophie Le Page, Guy-Vincent Jourdan, Gregor V Bochmann, Iosif-Viorel Onut, and Jason Flood. Domain classifier: Compromised machines versus malicious registrations. In *International Conference on Web Engineering*, pages 265–279. Springer, 2019.

[39] Wenyin Liu, Gang Liu, Bite Qiu, and Xiaojun Quan. Antiphishing through Phishing Target Discovery. *IEEE Internet Computing*, 16(2):52–61, 2012.

[40] Christian Ludl, Sean McAllister, Engin Kirda, and Christopher Kruegel. On the effectiveness of techniques to detect phishing sites. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 20–39. Springer, 2007.

[41] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150. ACM, 2007.

[42] Heather Mccalley, Brad Wardman, and Gary Warner. Analysis of back-doored phishing kits. In *IFIP International Conference on Digital Forensics*, pages 155–168. Springer, 2011.

[43] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[44] Daisuke Miyamoto, Hiroaki Hazeyama, and Youki Kadobayashi. An evaluation of machine learning-based methods for detection of phishing sites. In *International Conference on Neural Information Processing*, pages 539–546. Springer, 2008.

[45] Rami M. Mohammad, Fadi Thabtah, and Lee McCluskey. Intelligent rule-based phishing websites classification. *Neural Computing and Applications*, 25(2):443–458, 2014.

[46] Tyler Moore and Richard Clayton. Evil searching: Compromise and recompromise of internet hosts for phishing. In *Financial Cryptography*, pages 256–272. Springer, 2009.

[47] Naghmeh Moradpoor, Benjamin Clavie, and Bill Buchanan. Employing machine learning techniques for detection and classification of phishing emails. In *Computing Conference, 2017*, pages 149–156. IEEE, 2017.

[48] National Institute of Standards and Technology (NIST). Secure Hash Standard. Federal Information Processing Standards Publication 180-1., 1995.

[49] Ying Pan and Xuhua Ding. Anomaly based web phishing page detection. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 381–392. IEEE, 2006.

[50] Mateusz Pawlik and Nikolaus Augsten. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173, 2016.

[51] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[52] Gowtham Ramesh, Ilango Krishnamurthi, and K. Sampath Sree Kumar. An efficacious method for detecting phishing webpages through target domain identification. *Decision Support Systems*, 61(1):12–22, 2014.

[53] Angelo P.E. Rosiello, Engin Kirda, Fabrizio Ferrandi, et al. A layout-similarity-based approach for detecting phishing pages. In *2007 Third International Conference on*

*Security and Privacy in Communications Networks and the Workshops-SecureComm 2007*, pages 454–463. IEEE, 2007.

[54] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[55] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[56] Steve Sheng, Brad Wardman, Gary Warner, Lorrie Cranor, Jason Hong, and Chengshan Zhang. An empirical analysis of phishing blacklists. In *Sixth conference on email and anti-spam (CEAS)*. California, USA, 2009.

[57] Sami Smadi, Nauman Aslam, Li Zhang, Rafe Alasem, and M. A. Hossain. Detection of phishing emails using data mining algorithms. In *Software, Knowledge, Information Management and Applications (SKIMA), 2015 9th International Conference on*, pages 1–8. IEEE, 2015.

[58] Aditya K. Sood and Richard J. Enbody. Crimeware-as-a-servicea survey of commoditized crimeware in the underground market. *International Journal of Critical Infrastructure Protection*, 6(1):28–38, 2013.

[59] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[60] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[61] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, and Angelika Moscicki. Data breaches, phishing, or malware?: Understanding the risks of stolen credentials. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1421–1434. ACM, 2017.

[62] Rakesh Verma, Narasimha Shashidhar, and Nabil Hossain. Detecting phishing emails the natural language way. In *European Symposium on Research in Computer Security*, pages 824–841. Springer, 2012.

[63] Guang Xiang, Jason Hong, Carolyn P. Rose, and Lorrie Cranor. Cantina+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Trans. Inf. Syst. Secur.*, 14(2):21:1–21:28, September 2011.

[64] Guang Xiang and Jason I. Hong. A hybrid phish detection approach by identity discovery and keywords retrieval. In *Proceedings of the 18th international conference on World wide web*, pages 571–580. ACM, 2009.

[65] Haiyi Zhang and Di Li. Naïve bayes text classifier. In *2007 IEEE International Conference on Granular Computing (GRC 2007)*, pages 708–708. IEEE, 2007.

[66] Weifeng Zhang, Hua Lu, Baowen Xu, and Hongji Yang. Web phishing detection based on page spatial layout similarity. *Informatica*, 37(3), 2013.

[67] Yue Zhang, Jason Hong, and Cranor Lorrie. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th International Conference on World Wide Web*, pages 639–648, Banff, AB, 2007.

[68] George K. Zipf. Human behavior and the principle of least effort, 1950.