Toward a Phish Free World: A Feature-type-aware Cascaded Learning Framework for Phish Detection

Guang Xiang

CMU-LTI-13-001

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:
Jason Hong (Chair)
Carolyn Rose (Co-chair)
Alexander Hauptmann
Christos Faloutsos
Markus Jakobsson

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
In Language and Information Technologies

Keywords: Anti-phishing, machine learning, image processing, natural language processing, information retrieval

This dissertation is dedicated to my parents whose endless love, support and encouragement have accompanied me throughout my life, especially during the pursuit of my dream toward a doctoral degree.

Acknowledgments

Toward the completion of this dissertation, I ponder in retrospect the journey to my doctoral degree at Carnegie Mellon and find myself immersed in a basketful of fruitfulness, gratification and gratitude over the course of the past five and a half years. My great appreciation first goes to my principle advisor, Prof. Jason Hong, for his insightful research supervision, generous financial support and profound advice on my career.

I also thank my thesis committee members for their precious direction and encouragement on my research and thesis work. Prof. Carolyn Rose, my co-advisor, constantly inspired me to conduct thorough and in-depth analysis on the research result, which honed my analytical skills significantly. Dr. Alexander Hauptmann often instructed me to generalize my idea to a broader scope of applications, which cultivated my abstraction and summarization ability as a researcher. Prof. Christos Faloutsos, who offered incisive thoughts from the perspective of graph mining, enriched my knowledge base and skill set tremendously. Dr. Markus Jakobsson, a senior researcher and industry veteran, provided additional ideas about related cutting-edge technology and thus broadened my vision.

Carnegie Mellon is a great place for studying and growing. I feel fortunate to have the chance to know so many friends, who made my life here full of laughter. With top-notched universities and amiable ambience in Pittsburgh, I have grown from a young person who did not stand quite poised for the unpredictables to a mature man who is well prepared for all kinds of challenges in the future.

Above all, I am deeply grateful for my parents for their endless love, understanding, support and encouragement, which has accompanied me throughout my life, especially amid the critical years on my pursuit of a doctoral degree.

List of Figures

List of Tables

Contents

Abstract

In the real world, many fields have highly skewed class distributions and features that vary dramatically in terms of their classification and runtime performance. With a huge volume of data on the web, such fields typically require machine learning (ML) techniques with low latency and high performance. Anti-phishing is one of those fields, which requires a very low False Positive Rate (FP), a reasonably high True Positive Rate (TP) and a fast response time.

In those great number of areas including anti-phishing, however, almost all existing ML-based approaches simply focused on designing features, and building a monolithic model using them all at once. A fast response time is of paramount importance to the user experience in a live scenario, and naively extracting values for all features upfront is often an overkill.

In our previous work, we proposed a number of anti-phishing approaches that either extend existing URL blacklists in a probabilistic fashion or enhance feature-based anti-phishing methods with novel features, and in this thesis, we build on our previous experience with anti-phishing and propose a feature-type-aware cascaded learning framework for the a variety of domains with skewed class distribution and features with various classification and runtime performance in an effort to achieve a good balance between the three desiderata of TP, FP and latency. By utilizing lightweight features in early stages of the cascade and postponing prohibitive ones to later stages, our approach achieves a superior runtime performance in general, and can be further improved via parallelization in the distributed computing environment. Moreover, our approach is scalable with more features, and can be optimized in favor of FP or TP based on the specific domains. In the context of anti-phishing, our cascaded approach achieves 55.7% reduction in runtime on average over traditional single-stage models, with a low FP of 0.65% and a TP of 83.34%, and thus provides a fast and reliable solution for live detection scenarios.

Chapter 1

Overview

In the past decade, machine learning has found its way into a great variety of areas that involve classification tasks in our everyday life such as email analysis (spam and virus email filtering, etc.), web page analysis (anti-phishing [92], etc.), social network analysis (swearing language filtering in Twitter [87], etc.), image analysis (face recognition [89], etc.) and so on. All those areas require effective and fast or even realtime techniques to enhance user experience, especially given the gigantic amount of data on the web. However, most of the existing solutions in those fields simply build all features into a single classifier without considering the inherent difference among them, and mostly rely on the cheap commodity hardware for speedup.

We argue that the machine learning (ML) techniques in those domains can be improved fundamentally by re-factoring this monolithic model into a cascade of classifiers that employs increasingly expensive features along the cascaded pipeline to balance the runtime and classification performance. By cascade, we mean a model composed of multiple stages, each of which is a regular ML classifier. Using features with different costs and detection rates and designed to handle a vast number of pages on the web, phish detection is a perfect example of such fields, and in the rest of this thesis, we will focus on our proposed technique and its application in the context of anti-phishing.

Phishing is a form of identity theft, in which criminals build replicas of target websites and lure unsuspecting victims to disclose sensitive information like passwords, etc. Phishing has caused huge economic losses in the past a few years. Although there is a wide range of values for damage [13, 41], the number of attacks is substantial. Recently, the Anti-phishing Working Groups (APWG) estimated that the number of unique phishing sites detected in a month reached an all time high of $56,859$ in February, 2012 [16]. In terms of monetary losses, the statistics of the first half of 2012 released by RSA in July [46] show that the estimated worldwide losses from phishing attacks alone amounted to over $687 million US dollars, which marks a 32% increase compared with the first half of 2011. In the past a few years, phishing scams have evolved to target users on the web through social networking sites such as Facebook [48], and phishing has also been increasing in other areas of the world like China [67].

Two example phishing sites impersonating ebay.com and paypal.com are shown in Fig 1.1, which have high visual resemblance to the genuine web sites.

The exact definition of phish varies from paper to paper. Here, we define phish to be a web page satisfying the following criteria:

1. It impersonates a well-known website by replicating the whole or part of the target site, showing high visual similarity to its target.

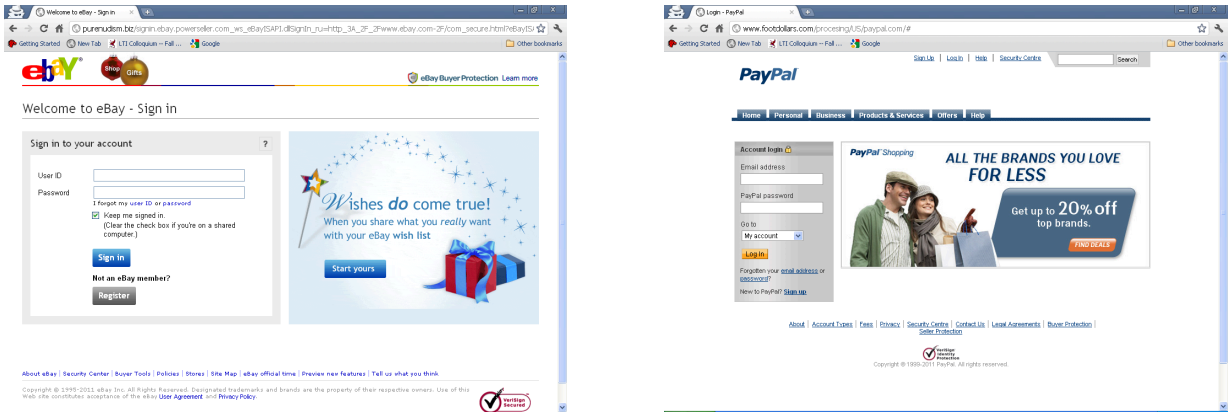

Figure 1.1: Example phishing sites targeting eBay (left) and PayPal (right).

2. It has a login form requesting sensitive information such as a password.

The anti-phishing arena is an ecosystem, covering a wide spectrum of areas including phishing label acquisition, feature and algorithm design for phish detection, system optimization, etc. Existing techniques have typically focused on the detection algorithms, which generally fall into two categories, i.e., those that perform URL matching based on the server-side human-verified blacklists and those that make use of features via machine learning techniques.

To alleviate the problems and improve the performance of the existing methods, we proposed a slew of techniques in our previous work, all of which achieved reasonably good performance.

1. To improve human effort in labeling phishing attacks, we explored novel techniques using computational approaches and designed a system that trains humans in identifying potential phish and enhances phish labeling by taking advantage of individual contributions.
2. To augment the rigid human-verified blacklists, we proposed a hierarchical blacklist-enhanced method for phish detection, which leverages existing human-verified blacklists and applies fuzzy matching techniques to detect phish in a probabilistic fashion with very high accuracy.
3. To exploit the brand name of a web site, which was never investigated in the literature before, we designed a technique that detects phish by discovering the inconsistency between the claimed brand and the genuine brand of a web page.
4. To capture more novel phish and partially alleviate the problem of high FP in the feature-based techniques, we proposed a layered anti-phishing solution that exploits the expressiveness of a rich set of features with machine learning to achieve a high TP on novel phish, and limits the FP to a low level via a hash-based near-duplicate phish filter and a login form detector.
5. To explore the strong look-and-feel signal on a web page, i.e., the brand logo image, we proposed a method that utilizes novel features with ML algorithms to identify the logo image, and then classifies a web page by inspecting the inconsistent identities via near-duplicate image matching techniques.

In spite of the techniques we pioneered as listed above, there are still a few missing pieces to the whole anti-phishing picture as well as other domains with similar characteristics, which calls for further attention and research effort. First, current blacklisting methods, usually provided by industry giants like Google [82] on their massive distributed infrastructure, cannot cover novel phishing trends in a timely fashion, because crawling a large number of URLs to update those

blacklists frequently remains a challenging task. According to [82], the classifier that updates Google's URL blacklist is retrained in an offline process daily. Second, the distribution of web pages is highly skewed in favor of the legitimate cases, and a majority of the legitimate pages are simple cases which do not need all the expensive features to classify, thus rendering the process of extracting all feature values in most existing anti-phishing methods inefficient. In the real world, low latency is of paramount importance to the end user experience, and the slew of features proposed in the literature are often an overkill. In fact, some fast features turn out to be sufficient in identifying a good number of the legitimate cases and a high percent of phishing attacks. Third, liability for false positives has been a major concern in industry. However, the existing ML-based techniques, all of which utilize the whole feature set in a monolithic classifier, either fail to deliver a low FP, or offer FP reduction via extra layers of filtering [86, 88].

To address those issues and further improve the state of the art, we take a holistic view of the phishing problem in this thesis, and propose a feature-type-aware cascaded learning framework for phish detection and other domains with similar characteristics. This dissertation makes the following major research contributions:

1. We propose a feature-type-aware cascaded learning framework for classification tasks in a variety of domains.
2. Our cascaded learning framework yields a parameterizable approach composed of classifiers in a cascaded structure for achieving a good tradeoff between the runtime cost and classification performance according to the requirements of the specific domains
3. Our cascaded learning framework produces a highly scalable and extensible system that can integrate future features easily.

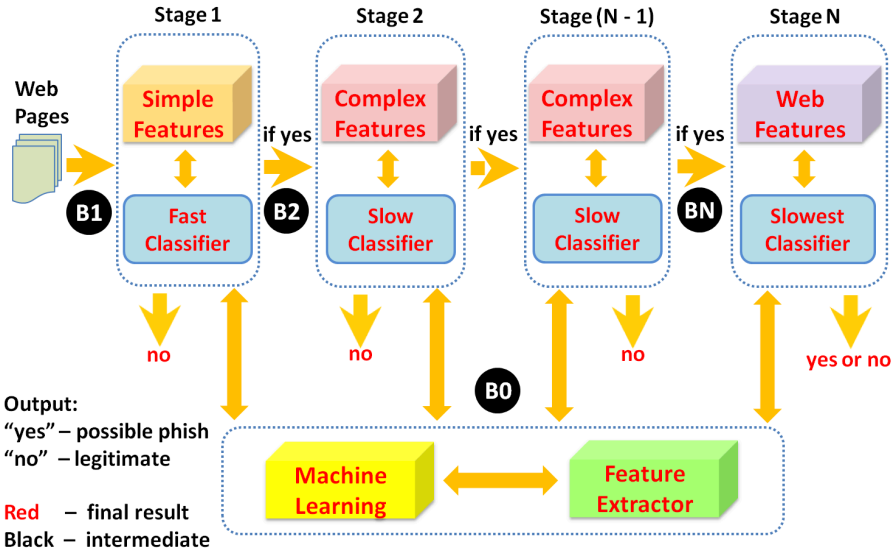

Figure 1.2: An example system diagram of the cascaded learning framework for phish detection. Each stage (B_1 to B_N) extracts a subset of features from the web pages arriving at that stage via the feature extraction component (B_0) and make classifications with a pre-trained model.

Specifically, our feature-type-aware cascaded learning framework exploits the distributional skewness of the web and builds different types of features into multiple stages of a single cascade. By utilizing lightweight features in early stages of the cascade and postponing slower ones to

later stages, our approach achieves a superior runtime performance in general, and can be further improved by the parallel computing infrastructure. In the context of anti-phishing, our approach achieves 55.7% reduction in runtime on average over stage-of-the-art single-stage models, with a low FP of 0.65% and a TP of 83.34%, and thus provides a fast and reliable solution for live detection scenarios. An example system diagram of the cascade structure for fields that emphasize FP more such as anti-phishing is shown in Fig 1.2.

In the following chapters, we will first introduce some representative existing work in the literature, then elaborate on each of our techniques as listed above, and finally focus on the cascaded learning framework for phish detection and other areas with similar characteristics. To make the readers better understand the structure of this thesis, we list in Table 1.1 all the techniques that we proposed in this thesis together with their core ideas and benefits.

Table 1.1: Summary of the techniques we proposed in this thesis. Each approach aims at improving the weaknesses from one perspective of anti-phishing domain. Particularly, the feature-type-aware cascaded learning framework can be generalized and applied to other fields with similar characteristics as anti-phishing.

Technique Idea	Technique Type	Benefits	Chapter
Improving human verification via computational techniques		Facilitate phish labeling	3
Enhancing URL blacklists with adaptive probabilistic techniques	Blacklist-based	Improve the TP of blacklist-based methods	4
Detecting phish by brand name discovery in the HTML text	Feature-based	Design features based on the site brand name	5
A Feature-rich ML framework for phish detection	Feature-based	Design novel features for phish detection	6
Detecting phish with logo images via near-duplicate matching	Feature-based	Exploit clues about phish via logo images	7
A feature-type-aware cascaded learning framework	Feature-based	Balance runtime/classification for fields such as anti-phishing	8

Chapter 2

Background and Related Work

Since we proposed a number of anti-phishing techniques and a general cascaded learning framework for phish detection and other fields in this thesis, we would like to structure this chapter in a similar fashion by first introducing our previous anti-phishing work that aimed at improving existing blacklist-based and feature-based solutions and then focusing on our cascade learning framework. In each of the following chapters from Chapter 3 to Chapter 7, we will elaborate on one anti-phishing approach that we proposed previously. We then introduce our cascaded learning framework for phish detection and other tasks with similar characteristics in Chapter 8.

2.1 Existing Anti-phishing Solutions

To make the readers understand the whole slew of existing anti-phishing techniques better, we categorize them into four major categories: improving the design of user interfaces to help end-users make better decisions, improving end-user training, leveraging wisdom of crowds, and making the problem invisible to end users via automated phish detection techniques. Since the anti-phishing approaches we proposed in this thesis all fall in the last category, i.e., the automatic phish detection techniques, we would like to focus on that in this section. Moreover, the literature review in this chapter only covers anti-phishing approaches, and we refer the readers to Hong's paper [42] for additional details such as an anatomy of an attack and why people fall for phishing attacks.

2.1.1 Improving the User Interface

One strategy for combating phish is to improve user interfaces and help users make better decisions. Examples of past work here include Dhamija et al's work in dynamic security skins [27], Wu et al's Web Wallet [84], and Egelman et al's study on browser anti-phishing warnings.

2.1.2 Training End Users

Another primary strategy for anti-phishing is to train end users to enhance their awareness of the phishing attacks. Examples of past work here include Anti-Phishing Phil, a game designed to engage the participant while progressively exposing them to more sophisticated phish-identification training [73], and PhishGuru, which uses simulated phishing attacks to train end-users [50]. One of our proposed techniques, i.e., Aquarium, made use of Anti-Phishing Phil to train participants, and focused on using minimally trained participants to help identify phishing web sites.

2.1.3 Leveraging Wisdom of Crowds

There has been a substantial amount of work looking at how to organize people online in an effective manner. In particular, in recent years, there has been rapid growth in research investigating how to build systems that leverage human effort for tasks that are too difficult for computers to do today.

Some research has examined specific domains, for example using games for image labeling tasks [78] or tagging shared documents [38, 60]. Other research has investigated how to improve people's contribution to a group, for example by assigning work to users in a way that makes the user believe their work is uniquely matched to his or her capabilities [44]. SuggestBot generated suggestions for articles to edit in Wikipedia based on machine learning techniques, to increase participation [25].

One technique in our thesis, Aquarium (see chapter 3), does not examine motivation. Instead, our work looks at how to improve the wisdom of crowds for a computer security task, to improve the results of human effort by applying computational techniques. Our work focuses on effective use of participants rather than increasing participation, by applying computational techniques such as clustering and vote weight.

There have also been several papers that have either used or have examined the use of Mechanical Turk for user studies. For example, Heer and Bostock [39] showed that MTurk was effective for crowdsourcing evaluations of visualizations. Kittur et al [49] used MTurk to collect ratings on the quality of Wikipedia articles, and offered guidelines for improving worker performance. Mason and Watts [58] investigated the effects of compensation for simple tasks, finding that increasing compensation increased the quantity of responses but not quality. Ipeirotis [43] examined the distribution of compensation for tasks, completion rates of tasks on different days, and the distribution of time to complete tasks. Relevant to our work, Ipeirotis found that the distribution of completion times follows a power law, where most tasks are finished quickly but a few tasks take very long. Partly for this reason, we posted new tasks every day in our experiment for Aquarium. Our work Aquarium in this thesis looks at how to apply crowdsourcing techniques to a security task, in this case, phishing.

2.1.4 Detecting Phish via Automatic Techniques

General Anti-phishing Methodology

In general, anti-phishing techniques fall into two major camps, i.e., blacklist-based methods, which leverage human-verified phishing URLs in an effort to control the FP, and feature-based approaches, which exploit the power of machine learning techniques to detect novel phish.

Though somewhat effective, existing solutions have apparent weaknesses. While human-verified blacklists are widely adopted in industry due to the very low FP, they do not generalize well to new attacks. For example, Sheng et al [74] found that zero hour protection offered by major blacklist-based toolbars has a TP between 15% and 40%. Furthermore, human-verified blacklists can be slow to respond to new phishing attacks, and updating blacklists usually involves enormous human effort. For example, PhishTank [65], a popular community site where people can submit, view and confirm phishing URLs, posted statistics in March 2009 showing that it took on average 10 hours to verify a submitted URL [66]. Moreover, Sheng et al found that blacklists were updated at different speeds, and an estimated 47% - 83% of phish appeared on blacklists about 12 hours after they were launched [74].

On the other hand, while feature-based approaches offer a more general mechanism to detect novel attacks, this type of methods tend to have a relatively higher FP. Concerns over liability for false positives have been a major barrier to deploying these technologies [72]. To underscore this point, Sheng et al [74] evaluated eight popular toolbars, all of which employ human verified blacklists to achieve an extremely low FP in spite of the amount of human labor required.

Since feature-based techniques account for a vast majority of the automatic phish detection approaches, we will focus on this line of prior research in this section.

Anti-phishing Methods Monitoring Passwords

Passwords are the key ingredient to any service with authentication, and researchers have also proposed approaches that guard users against phishing attacks by monitoring the information flow, mostly the password. Kirda et al implemented a system called AntiPhish [47], which watches the password field of the HTML form and searches the domain of the site being visited among a list of previous logins when identical password is found. AntiPhish warns users of potential attacks if a domain match is not found. One problem with this tool is that manual labor is usually involved, and also false positives could be raised if the same password is used on multiple sites, which is what users typically do. Moreover, no formal evaluation of the tool was conducted. Another work is password hashing (PwdHash) by Ross et al in [70]. PwdHash sends a hash value computed from the user's password and the website domain, rather than the plain text password, to the server for authentication, rendering password stealing at the fake phishing site futile. In a recent study, Yue et al [90] designed a client-side tool called BogusBiter, which sends a large number of bogus credentials to suspected phishing sites, hiding the real credential among the bogus ones. However, BogusBiter relies on web browsers' built-in components or third-party toolbars to detect phish and thus is more of a reaction rather than detection technique.

Text-based Phish Detection Methods

In the URL blacklist camp, Xiang et al. [88] proposed a content-based probabilistic approach, which leverages existing human-verified blacklists and applies the shingling technique, a popular near-duplicate detection algorithm used by search engines, to detect phish.

Since feature-based techniques dominate in the anti-phishing arena, we will focus on this body of work in the rest of this section. Among them, one area of work uses URL features to detect phishing web pages. Garera et al [36] categorized phishing URLs into four groups, each capturing a phishing pattern, and used a set of fine-grained features from the phishing URLs together with other features to detect phish. Applying a logistic regression model yielded an average TP of 95.8% and FP of 1.2% over a repository of 2508 URLs. Though interesting, this method has unstable performance in that URLs could be manipulated with little cost, causing the features to fail. In PhishDef [51], Le et al utilized lexical features and external features based on the URLs with online learning algorithms to detect phish, achieving an accuracy of over 90%. However, this work suffers from the same problems as [36].

Techniques that use features based on the textual information extracted from the HTML DOM are also available. In [69], Rosiello et al came up with a layout-similarity-based approach, trying to overcome the problem in [47] that the same password for multiple sites tend to raise false alerts by AntiPhish. They added one extra layer of examination, checking the similarity of the HTML DOM between the web page currently being visited and the one visited before with the same password. This technique, however, is brittle in that the HTML DOM is very easy to

manipulate without changing the layout of the web page. In [55], Ludl et al applied a J48 decision tree algorithm on 18 features solely based on the HTML and URL, achieving a TP of 83.09% and a FP of 0.43% over a corpus with 4149 good pages and 680 phishing pages. However, features purely based on HTML DOM and URL are rather limited and may fail in capturing artfully designed phishing attacks. Another feature-based work exploring the HTML DOM is CANTINA [92], in which Zhang et al proposed a content-based method using a simple linear classifier on top of eight features, achieving an 89% TP and a 1% FP on 100 phishing URLs and 100 legitimate URLs. Recently, Xiang et al. [86] proposed designed 15 features in CANTINA+ utilizing the HTML DOM, search engines and third party services with machine learning techniques to detect phish, achieving a TP of 92% with an FP of 0.4%.

Aside from the methods introduced above that examine the degree of unusualness of a webpage via features, a couple of techniques have been proposed that detect phish by directly discovering the target brand being phished. Pan et al [64] proposed a method that extracts the webpage identity from key parts of the HTML via the χ^2 test, and compiled a list of features based on the extracted identity. Their method achieved an average FP of about 12%. However, the assumption that the distribution of identity-related words usually deviates from that of other words is questionable, which is indicated by their high FP. Even in DOM objects, the most frequent term often does not coincide with the web identity. Xiang et al [85] proposed a hybrid detection model that recognizes phish by discovering the inconsistency between a webpage's true identity and its claimed identity via search engine and information extraction techniques. Their full system achieved a TP of 90.06% and a FP of 1.95%. In another work, Liu et al. [81] proposed an approach to automatically discover the phishing target of a given suspicious web page by first finding its parasitic community and then its target with which the given web page has the strongest parasitic relationship. Particularly, the parasitic community of a given web page is built by finding all web pages with direct or indirect association relationship with it.

Anti-phishing Techniques Exploring Visual Elements

To exploit visual similarity between webpages, Liu et al [54] proposed a method using three similarity metrics, i.e., block level similarity, layout similarity and overall style similarity, based upon webpage segmentation. A page is reported as phishing if any metric has a value higher than a threshold. Though interesting, this work suffers from the following weaknesses: a high FP of 1.25% and potential instability due to the high flexibility of the layout and style elements in the HTML documents. In [27], Dhamija et al proposed a new scheme named dynamic security skins, which authenticates the server by users' visual verification of the expected image and an image or "skin" generated by the server. Though interesting, this paper has no formal evaluation.

SpoofGuard [23], a technique proposed by Chou et al., utilizes image check as one feature, examining the domain name and the existence of popular target site logos on a web page. However, SpoofGuard simply checks all images on a page and cannot handle images with modifications, which is easy to beat and inefficient. Another work that exploits visual elements is [59], in which Medvet et al compute a signature using the visible text, visible images and overall visual look-and-feel to compare the suspected pages with their legitimate counterparts. They conducted pairwise comparison along each dimension, which is costly even with some optimization. Recently, Chen et al [22] took a holistic view of the visual similarity between web pages, and applied compression algorithms on the pages as indivisible entities to detect phish. One problem of their approach is that it cannot handle novel attacks well. In [79], Wang et al. extracted local features from images and matched the features to a set of reference logo images for phish detection. However,

they considered all img tags on a web page, which is inefficient, and their 1% FP is much higher than our 0%. In [17], Bannur et al. combined a great variety of features using URLs, HTML DOM, page links, web search, topic models, visual elements, etc., achieving a precision of 97.6% and a recall of 96.6%.

Fast Anti-phishing Techniques

In addition to the techniques that designed features to detect phish, another line of work emphasizes fast solutions via either online learning algorithms to update the classification model efficiently or distributed computing infrastructure to expedite feature extraction.

For the former, the fast nature of those approaches is actually only for processing new training data in an offline mode, and prohibitive web access is still needed to get information such as the WHOIS record for live scenarios. In [56], Ma et al adopted a series of online learning algorithms with lexical features and host-based features from the URLs, including perceptron, logistic regression with stochastic gradient descent, passive-aggressive (PA) algorithm, confidence-weighted (CW) algorithm, and achieved a classification accuracy of up to 99%. In another work [82], Whittaker et al also adopted an online gradient descent logistic regression learning algorithm to constantly train the detection model with new data, processing the majority of tasks quickly with a median of 76 seconds. They used a proprietary implementation of the logistic regression algorithm using Google's hardware infrastructure. Similarly, Thomas et al [75] used the online logistic regression algorithm with L1-regularization with a large set of features including URLs, HTML content, page links, etc. on the Amazon Web Services (AWS) cloud infrastructure, which offers high parallelism.

For the latter, feature extraction cannot be parallelized entirely for many tasks such as anti-phishing because a lot of features based on the HTML DOM have to wait until the browsers finish parsing the HTML, which renders the parallel framework more or less a sequential process. In addition, costly features still need to be computed, thus posing a bottleneck to the efficiency of those approaches. One such example is [75], in which Thomas et al. utilized the Hadoop Distributed File System (HDFS) on a 50-node cluster of Amazon EC2 Double-Extra Large instances to train their models and ran feature collection in parallel on 20 EC2 High-CPU instances for efficient spam filtering. Our technique in this thesis, however, provides a fast solution by attacking the inefficiency of previous techniques from the fundamental level, i.e., avoiding and postponing the expensive features as much as possible, Naturally, our approach can also take advantage of the distributed cloud environment to achieve further speedup.

2.2 Previous Research in Cascaded Classifiers

Learning a cascade of simple classifiers is not a brand new idea, and has been successfully applied to areas such as object detection in computer vision. In [77], a seminal work about object detection, Viola and Jones proposed a method for integrating increasingly more complex classifiers in a cascade which quickly discards background regions of an image while spending more computation on promising object-like regions. A face detection system based on their idea ran approximately 15 times faster than previous approaches. This classic paper spawned a great number of follow-up work, most of which were developed in the context of building rapid real-time object detection systems where there are a large volume of features of the same type. For instance, the features used in [77] are of one type only, i.e., rectangle features, and yet come with a large volume, e.g., over $180,000$ given a detector with 24×24 base resolution. In reality, however, most tasks have

a set of heterogeneous features with different classification performance and runtime cost, which make the classification process tougher and trickier.

Motivated by [77], quite a few ideas were proposed to tune the cascade learning process, such as [20] in which a maximum FP and a minimum detection rate are specified for each stage when building the cascade structure, or [57] which uses cost-sensitive learning to penalize different types of errors, etc. Although that work achieved a certain amount of improvement over [77], they did not take into consideration the cost of different features during the training process.

To the best of our knowledge, only a few previous research approaches attempted to integrate the cost of feature extraction into the classifier training process. In [68], Raykar et al. estimated the cost of computing each feature based on the training set and then incorporated the constraint regarding the expected cost into the optimization function of the maximum a posteriori (MAP) estimation. In their experiment, however, they only considered simple features such as text features whose costs typically do not have significant variance, and for cases where feature extraction sometimes leads to unexpectedly high overhead like features using web resources, their approach may not work well. In another work [52], Ling et al. investigated encoding the cost of feature extraction into decision tree learning, which yields a decision tree that minimizes the sum of misclassification and feature extraction costs. However, it is non-trivial to define the overhead of misclassification and runtime on the same cost scale, and the decision tree learned that way may not be desirable.

Our proposed approach extends the previous work and builds a set of heterogeneous features with different costs into stage classifiers by considering constraints on the runtime and detection performance jointly in a single cascaded learning framework. It is capable of achieving a nice tradeoff between the runtime and classification performance, and is more reasonable, robust and extensible with existing and novel features.

Chapter 3

Improving Human Verification via Computational Techniques

Phish labeling is a critical stage of the anti-phishing ecosystem, and in this chapter, we introduce our approach that explores novel techniques for combating the phishing problem using computational techniques to improve human effort on labeling potential phishing attacks [1].

Confirmed phishing attacks, typically from major URL blacklists, are source of ground truth, and are of particular importance to any ML-based anti-phishing technique. However, the way major blacklists obtain labels typically involves considerable human efforts which is inefficient and could be improved by computational techniques.

Using tasks posted to the Amazon Mechanical Turk human effort market, we measure the accuracy of minimally trained humans in identifying potential phish, and consider methods for best taking advantage of individual contributions. In particular, we use clustering techniques to facilitate phishing labeling by aggregating similar phish in terms of textual content, and exploit difference among individual users via a vote weighting mechanism to improve the results of human effort in fighting phishing. We found that these techniques could increase coverage over and were significantly faster than existing blacklists used today.

3.1 Introduction

Nowadays, many problems still require human intelligence to solve. Some require human intelligence as an intrinsic part of the process, such as in a democratic election. Others have no known technical solutions which match human performance, such as image labeling [78]. In the case of certain kinds of computer security tasks, it has been suggested that it is too risky to take the human entirely out of the loop [30]. The anti-phishing domain is no exception, where the ground truth of phishing data is typically obtained by human effort.

In reality, labeled phish are usually provided by human-verified blacklists, which contain URLs of sites that have been manually verified as phish and are also widely used in industry to detect phishing attacks. For example, three well-known phishing blacklists are operated by Microsoft, Google, and PhishTank. The main advantage of blacklists is that there are very few, if any, false positives, thus reducing the liability risk of incorrectly labeling a legitimate site as a phishing attack. However, human verification is inherently more labor intensive and can be much slower

[1] Parts of this chapter were previously published in the Proceedings of the 7th Symposium On Usable Privacy and Security (SOUPS'11) [53]

in detecting attacks. Moreover, human verification can also be overwhelmed by simply generating more phishing sites and/or URLs for phishing sites, as has been done with automated phishing attack toolkits and "fast flux" techniques that hide a phishing site behind a large number of compromised hosts to make detection more difficult [80].

Among the major URL blacklists, of particular interest to us is the one maintained by PhishTank, which uses a wisdom of crowds approach relying on web users to identify phish. According to PhishTank's own statistics [66], out of 1.1M URL submissions from volunteers, there were 4.3M votes, resulting in about 646K identified phish between October 2006 and February 2011.

Table 3.1 shows some basic statistics regarding PhishTank's efficiency in validating real phish. From Jan 2010 to Jan 2011, the median time to identify a phish has dropped from 12 hours to about 2.4 hours. The percentage of valid phish identified has also increased, going from 5,751 out of 18,836 (30.5%) in January 2010 to 12,789 out of 16,019 (79.8%).

Two observations come naturally from Table 3.1. First, for January 2011, there are still 2,681 URLs not identified as phish or legitimate. Most of these URLs represent "wasted" votes which did not reach the required number of votes for verification. Optimally, with 4 votes required to identify a phish, 69,648 votes could have identified a maximum of 17,412 labels rather than the 12,789 phish and 549 legitimate sites actually identified. Second, a median delay of 2.4 hours still represents a significant gap in protection, as most victims of a phishing scam fall for it within 8 hours of the start of the attack [50]. Furthermore, 2.4 hours only represents the delay from when the URL was first submitted to PhishTank, meaning that the phish was in the wild longer. Lastly, 2.4 hours represents the median, with past work suggesting that there is a power-law distribution in identifying and taking down phish [62].

Table 3.1: PhishTank self-reported statistics. Submissions require a minimum of 4 votes before labeling, with at least 70% agreement (some votes weighted differently). Median time has improved significantly.

	January 2010	January 2011
Submissions	18,836	16,019
Total votes	54,847	69,648
Valid phish	5,751	12,789
Invalid phish	518	549
Median time	12 hours 10 minutes	2 hours 23 minutes

We believe a promising solution is to improve the wisdom of crowds by combining manually verified blacklists with computational techniques, to keep false positives extremely low while also reducing the time to verify attacks. Such an approach would benefit not only sites like PhishTank, but also other manually-verified blacklists such as Google and Microsoft. A hybrid approach could also help with forensic analysis (such as identifying trends in phishing attacks and attacked brands), as well as help reduce the labor in maintaining the many databases that store data about current and past phishing attacks.

In this chapter, we present the results of a study that we conducted with Aquarium, an experimental system we developed on top of Amazon's Mechanical Turk system for gathering human-verified labels on potential phishing sites. From a broad perspective, this technique looks at how to apply crowdsourcing techniques to a security task, and how to use computational techniques to improve the performance of a crowd. More specifically, the approach in this chapter makes the following research contributions:

1. We present the design of Aquarium, a novel phish detection approach that makes use of two points in this design space, namely (a) clustering similar phish together and having minimally trained participants vote on clusters rather than individual phish, and (b) developing a vote weighting mechanism based on a participant's historical performance.
2. We present an evaluation of our two approaches, examining time to label a URL, accuracy, coverage, and monetary cost. Through a two-week study of verification of suspicious URLs, we show that our approach achieves a TP of 95.4% with a FP of 0%, with a median time to label of 0.7 hours.
3. We present our voteweight formula and the results of our parameter tuning, which can reduce the median time to label a URLs down to 0.5 hours.

3.2 Improving Human Effort with Computational Techniques

3.2.1 Improving Human Effort

In this section, we outline a design space for improving human effort in phish identification. This design space is not comprehensive, but rather sketches out some of the opportunities at hand.

One area for improvement is modifying the order in which suspicious URLs are shown to participants. For example, one could show a submission that is closest to completion, newest submissions, oldest submissions, or even random. One could also tailor what phish a participants sees based on their presumed knowledge of that brand or past votes. PhishTank's ordering has not been formally published; however, it does not seem to be by recency only. With Aquarium, we order submissions first by closest to completion and then by newest.

Another area for improvement is modifying how submissions are shown, for example showing them one-by-one or showing similar submissions together. In Aquarium, we compare the effectiveness of both of these approaches. We believe showing groups of suspicious URLs should help in two ways. First, one can apply a vote to multiple suspicious URLs simultaneously rather than going through them individually, mitigating the effect of attackers trying to overwhelm the people verifying these phishing sites. Second, for unfamiliar brands, seeing multiple copies of the same page, each of which have unusual URLs, can help participants in inferring whether or not the cluster is a phish.

A third possible intervention is to adjust the threshold for when a submission is labeled. PhishTank's threshold has not been formally published, but appears to require at least 4 votes minimum and at least 70% agreement between voters (with some votes weighted more than others). One could imagine many variants of this, including for example changing the minimum number of votes, changing the level of agreement needed (e.g. from 70% to 80%), changing how votes are weighted, and even having an automated algorithm provide a vote. Changing this threshold could affect accuracy, the time it takes to successfully label a submission, and breadth of coverage.

A fourth kind of intervention is to find better ways of motivating people to submit more votes or more accurate votes. As we noted in the related work section, there have been several papers looking at how to motivate people to contribute more work and higher-quality work. In the domain of phishing, some possibilities include showing specific brands to people who either care a lot or know a lot about that brand, having competitions, organizing people into teams of voters with specific goals, and virtual rewards such as achievements or leaderboards. We do not investigate these issues in this work, and instead use MTurk's payment system.

3.2.2 Aquarium System Architecture

Our system architecture is shown in Fig 3.1. We first crawl the web pages of URLs submitted to PhishTank that have not yet been verified as phish. These URLs may or may not have any votes on them. PhishTank's API and web page do not show how many people have voted on unverified URLs. We submit these URLs as tasks to Amazon's Mechanical Turk, where qualified participants are paid to label them as phish or legitimate. Aquarium then clusters web pages by similarity before they are presented to users. We currently use DBSCAN and shingling, a common algorithm often used by search engines for detecting duplicate pages. To be qualified on Mechanical Turk, we required participants to achieve a certain score on the Anti-Phishing Phil micro game [73]. As participants cast votes, we weight those votes based on their history of votes.

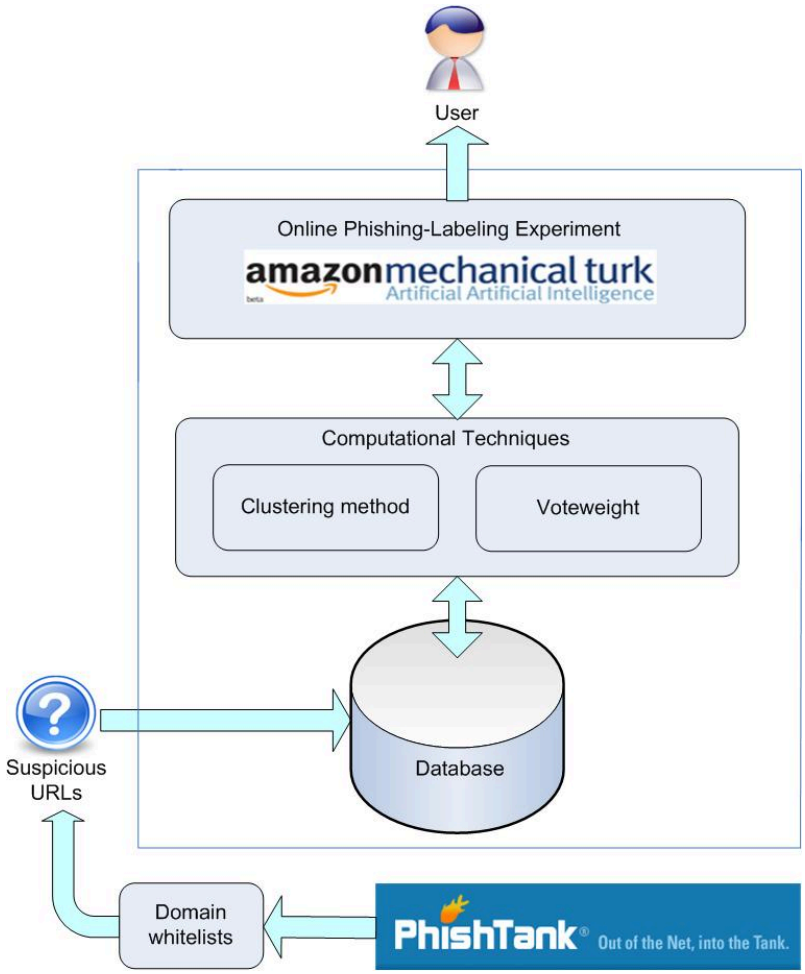

Figure 3.1: Aquarium system architecture. We first crawl unverified URLs from PhishTank and check them against a whitelist. We download the web pages of URLs not on the whitelist. We use DBSCAN and shingling to cluster similar pages. We submit these clusters to Amazon's Mechanical Turk for verification by participants. Finally, each participant's vote weight is adjusted based on past performance.

In the first step, we collect URLs submitted to PhishTank as our test dataset. We use a small whitelist to filter legitimate web pages, to reduce effort by users. In February 2011, we collected

2,784 domains to whitelist from Google safe browsing [1] and 424 from millersmiles [2]. In our past research, we found that this combination of whitelist works reasonably well with minimal false positives [85, 88].

Next, our system clusters similar phish together. We set the shingling similarity threshold to 0.65, a figure that worked well in our past work [88]. To demonstrate the potential of clustering, using all of the data crawled from PhishTank, we found 3,180 out of 3,973 web pages could be grouped into 392 clusters, with cluster size ranging from 2 to 153 URLs. Note that these clusters do not take into account time. For Aquarium, we cluster similar URLs currently available at that time. We also made the maximum size of clusters 25, high enough that clusters would be useful but low enough so that mistakes (or malicious votes) would have limited damage. The distribution of clusters after capping at 25 is shown in Fig 3.2.

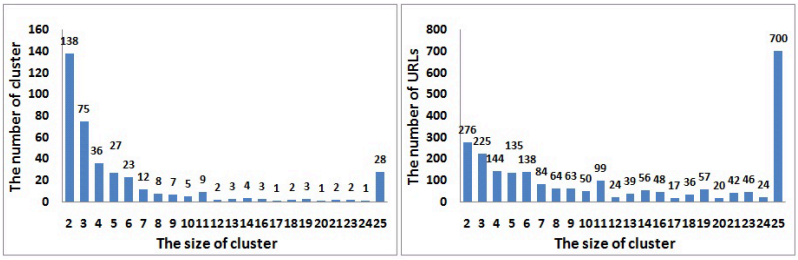

Figure 3.2: The distribution of clusters in our time-based approach to grouping. The top figure (a) shows that there are many small clusters of size 2 which quickly tail off. The bottom (b) shows the total number of URLs in different size of clusters. For example, we have 28 clusters of size 25, meaning that these clusters represent $28 \times 25 = 700$ URLs.

Submissions are then submitted to Amazon's Mechanical Turk (MTurk) system as Human Intelligence Tasks (HITs) for verification. We submit two kinds of HITs. The first lets participants verify submissions one-by-one. The second one lets participants verify clusters of phish (see Fig 3.3). Participants saw a given URL at most once regardless of HIT condition.

Ideally, as participants vote on submissions, we can apply our voteweight model to modify the impact of a user's vote. Currently, we do not do this, and only examined the effects of voteweight after the fact. In the voteweight model, we consider two factors, namely a user's performance on verification and the time when a user casts a vote. Briefly, people who vote early and have a high accuracy in voting correctly are weighted more. We factor in time because an old vote does not tell us as much about a user's current performance as a more recent vote. The exact formula used is described in Section 6.1.

Like PhishTank, Aquarium requires a minimum of 4 votes. If the majority of votes for that URL identify it as phish, then we label that URL as phish (this mimics PhishTank's threshold of 70% with 4 votes). The same is true with legitimate URLs. However, if a URL has equal votes both for phish and legitimate, we label it as unidentified. In the Control Condition, there were 153 URLs (3.9%) not labeled due to tie votes. In the Cluster Condition, there were 127 (3.2%). Unlike PhishTank, we do not continue to gather more votes from people. This is primarily due to limitations with MTurk, which make it very difficult to have a variable number of workers per HIT. Although this does place caveats on our results, we argue that our results are very strong and should still generalize despite this weakness.

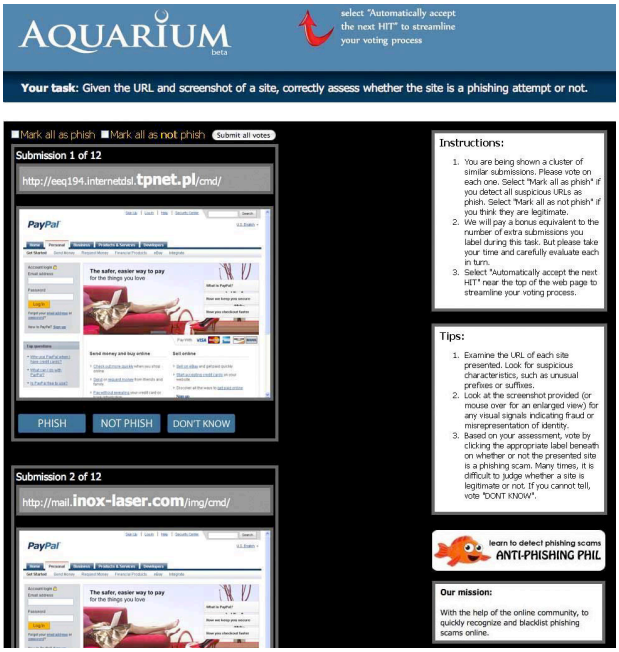

Figure 3.3: A sample task on Aquarium. Users can see the URL and screenshot of a suspicious web page and then label it as phish, not phish, or don't know. Users in the Cluster Condition (as shown above) could see up to 25 similar sites all at once. Participants in the Cluster Condition could "mark all as phish" or "mark all as not phish".

3.2.3 Measuring Page Similarity with Shingling

To cluster effectively, we need a way of measuring similarity. We could easily do exact page comparisons or use hash functions. Given that many phishing web pages are created using toolkits, this simple approach should work reasonably well today. In fact, in our early evaluations, we found that hash codes worked reasonably well for clustering. However, exact matching is very brittle, in that changing a single byte would lead to a non-match.

As such, we opted to use an approximate matching algorithm. Shingling is a popular page duplication algorithm invented for search engines. The core idea behind shingling is to break up web pages into n-grams and then compare how many n-grams two pages have in common. Here, n-grams are a term from natural language processing, and are subsequences of n contiguous tokens from the text. For example, sample text "shop without exposing your financial information" has the following 3-grams: shop without exposing, without exposing your, exposing your financial, your financial information.

Shingling employs a metric called resemblance to calculate the percent of common n-grams between two web pages. Let $S(p)$ denote the set of unique n-grams in page p and the similarity metric resemblance $r(q, d)$ for pages q and d is then defined as:

$$r(q,d) = \frac{|S(q) \cap S(d)|}{|S(q) \cup S(d)|} \tag{3.1}$$

This approximate matching approach first breaks each page into a set of unique n-grams, and saves them in memory to speedup runtime performance. After excluding good pages whose domains appear in the whitelist, we compute resemblance $r(q, d)$ for a query page q, and fire an

alarm whenever $r(q, d)$ exceeds a threshold t. We used the same threshold as in our past work [88], namely 0.65. The average time cost of calculating similarity of two web pages on a laptop with 2GHz dual core CPU with 1 GB of RAM is 0.063 microseconds (SD = 0.05).

3.2.4 Clustering Algorithm

Shingling is a similarity metric used over the shingles or n-grams in our context, and we still need a way of clustering similar pages together. In Aquarium, we used the well-known density-based DBSCAN algorithm. We chose this clustering method for two reasons. First, it can select any data point as the start point for clustering. Second, the algorithm only needs one scan of the database to finish clustering. The concepts used in our clustering approach are described below.

1. **Eps**: Minimum similarity of the neighborhood of the cluster
2. **MinPts**: Minimum number of points in an Eps-neighborhood of that point
3. **Core point (CO)** : Point is in the interior of a density-based cluster
4. **Border point** : A border point is not a core point, but falls within the neighborhood of a core point
5. **Directly-density-reachable (DDR)** : If point x is CO, point y is in x's Eps-neighborhood
6. **Density-reachable** : There exists a chain of DDR objects from point x to point y

Based on the concepts above, we present the algorithm that clusters phishing web pages in Algorithm 1.

Algorithm 1 PhishClustering

Require: A set of phishing web pages S
Ensure: Clusters for S
1: $CC \leftarrow \phi$
2: Pick a phish P, quantify the similarity from P to each phish in S through shingling
3: Select P as the start point and retrieve all points density-reachable from P with respect to Eps and MinPts
4: If P is a core point, a cluster C is formed, $CC \leftarrow CC \cup C$
5: If P is a border point, no points are density-reachable from P, visits the next phish
6: Continue until all phishing web pages in S have been processed
7: **return** CC

We tested on our data with different values of Eps from 0.6 to 1 by steps of 0.5 and MinPts of 2. With Eps at 0.60, the accuracy is 98.8% (we visually scanned all of the generated clusters). However, accuracy was 100% with Eps from 0.65 to 1. Hence, for our clustering, we chose Eps=0.65 and MinPts=2. The time cost of clustering over all 3,973 pages was about 1 second.

3.2.5 Incremental Update of the Data

Since there is a stream of suspicious URLs, the clusters discovered by our method need to be periodically updated. Clustering can be expensive in terms of time. However, it is not necessary to re-cluster the whole database each time. We use following method to assign a new URL to a cluster. We first compare the content similarity of each new submission with those of the dataset.

- If there is no similar web page, we create a new cluster for the new submission.

- If the similarity is above the given threshold and all similar web pages are in the same cluster, we assign the new submission to this cluster (unless the cluster is at its maximum size).
- If there are many similar web pages in different clusters, we choose the largest cluster that is not at its maximum size.

When a new submission is grouped in a cluster, it has zero votes and does not inherit the votes of any other submissions in the same cluster. It is simply presented with other available similar submissions of the cluster for verification.

3.3 Exploiting Inter-personal Difference via Vote Weight

The core idea behind voteweight is that participants who are more helpful in terms of time and accuracy are weighted more than other participants. Weighting votes more accurately should also help reduce the time it takes to label a submission. Our notion of voteweight is similar to the concept of mavens in the Acumen system [37], though we examine a different domain (phishing vs cookies) and leverage time in our model. Our voteweights are also continuous, whereas mavens were chosen to be the top 20% of users in Acumen.

Intuitively, a correct vote should be rewarded and a wrong one should be penalized. In addition, recent behavior should be weighted more than past behavior, as it gives us a better sense as to a participant's current abilities (or level of malice).

Towards this end, we propose a metric called voteweight in Eq 3.2 that combines these factors in one summary statistic. In our model, we use $y \in \{t, +\infty\} \cup y \in \{-\infty, -t\}$ to label the status of a URL, where y is the sum of voteweight of a given URL, t is the threshold of voteweight, and $y \geq t$ means a URL has been voted as a phishing URL and $y \leq -t$ means voted as legitimate.

$$v_i^{'} = \frac{v_i}{\sum_{k=1}^{M} v_k} \tag{3.2}$$

where $v_i^{'}$ is the normalized voteweight of user i, with a value between $[0, 1]$, v_i is the raw voteweight, and M is the number of users. Note that because our normalized value is less than 1 we also have to adjust our threshold accordingly.

$$v_i = \begin{cases} RV_i & \text{if } RV_i \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

$$RV_i = R_i - \alpha \cdot P_i \tag{3.4}$$

Equations 3.2 and 3.3 show the formulas for raw voteweight (RV). Conceptually, raw voteweight for a user i is the sum of the rewards for correct votes R_i, minus a weighting parameter times the penalization for incorrect votes P_i. Adjusting the parameter allows us to weight the penalty relative to the reward. For example, an value greater than 1 means that participants are penalized more heavily for wrong votes than they are rewarded for correct votes.

$$R_i = \sum_{j=1}^{N} \frac{T_j - T_0 + 1}{T - T_0} \times I_{C_{ij}=L_j}(j) \tag{3.5}$$

$$P_i = \sum_{j=1}^{N} \frac{T_j - T_0 + 1}{T - T_0} \times I_{C_{ij}\neq L_j}(j) \tag{3.6}$$

Equations 3.5 and 3.6 show our reward and penalty formulas. The first part of both equations show the weight we give to time. Here, T_0 is the timestamp of user i's first vote ever; T_j is the timestamp of user i's vote on phish candidate j; and T is the current time when computing user i's voteweight based on the historical vote information. Thus, if T_j is recent and close to current time T, this first part is close to 1. If T_j is very old, then the first part becomes smaller, meaning that older votes have less weight. In our study, we calculated the interval of time in hours.

There are alternative variations for weighting time that also could have worked, for example, having a sliding window of the last N days of votes, taking only the last N votes, and so on. We wanted to explore how well any voteweight feature worked first before trying the many alternatives. As such, we chose one that worked well with the two weeks of data we had.

The right half of equations 3.5 and 3.6 are an indicator function with a value of 0 or 1. Essentially, for the reward formula, we want the indicator to be 1 if they voted correctly and 0 otherwise. For the penalty formula, the opposite is true. More formally, C_{ij} is the label that user i assigns to phish candidate j; L_j is the ground truth label for phish candidate j; N is the number of phish candidates that user i has voted on; $I_A(x)$ is an indicator function defined as:

$$I_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

With our voteweight, we determine the label for a candidate phish t based on users' votes by

$$l_t = \sum_{i=1}^{K} v_i^{'} \times C_{it} \tag{3.8}$$

where the label of phish candidate t is a weighted average of the votes by K users and the value of a vote is defined as

$$C_{it} = \begin{cases} 1 & \text{if voted as phish} \\ -1 & \text{otherwise} \end{cases} \tag{3.9}$$

3.4 Experiment Setup

In evaluating this technique, we wanted to (a) assess how well clustering worked versus labeling each submission individually, (b) determine the effectiveness of various approaches for weighting votes, and (c) compare the effectiveness of Aquarium to existing blacklists in terms of time, accuracy, and coverage.

In an early pilot test of this work before clustering was implemented, we found that people often did not know certain brands and had a hard time labeling a site as phish or legitimate the first time they saw that brand. However, we also saw that people realized a site was phish after seeing the same site for the third or fourth time. Furthermore, we saw a large number of visually duplicate sites in our pool of URLs. This insight led us to add clustering as a possible way of improving accuracy as well as reducing time and overall effort.

3.4.1 Gathering Data with Mechanical Turk

Since PhishTank does not make its raw voting data easily available, and since we could not directly modify the PhishTank site, we created Aquarium to mimic the functionality of PhishTank. We used PhishTank's API to sample live data from the stream of sites being submitted. We then submitted both individual submissions as well as clusters of submissions as Human Intelligence

Tasks (HITs) to Amazon's Mechanical Turk (MTurk), an online service designed to allow work requesters to quickly hire web-based workers by posting tasks for a set price. Workers were paid $0.01 for each HIT.

Normally, having MTurkers simply label data does not require an IRB at our university. However, since we had designed an intervention which was the subject of an experiment, we submitted an IRB, which was approved as a minimal risk study.

To compare Aquarium with PhishTank, we first collected unverified URLs submitted to PhishTank from Jan. 1, 2011 to Jan. 14, 2011. Unverified URLs are those that do not have enough votes to be verified as legitimate or phish. We also captured a screenshot of each submission when they were alive. We replayed this data as HITs over a different period of 14 days from Feb. 11 to Feb. 24 and mapped them to the submissions we downloaded from PhishTank from Jan. 1 to Jan. 14. Tasks were presented to users for verification only after the same time corresponding to when they were previously submitted to PhishTank. For example, suppose a suspicious URL was submitted to PhishTank at 2:51 am, Jan. 3, 2011. In our study, the task of such URL could be viewed by our participants only after 2:51 am, Feb. 13, 2011.

The Control Condition and the Cluster Condition were listed as separate HITs. Both conditions had the same data. Participants could move back and forth between conditions. However, a participant only saw a given URL at most once. We chose this experimental design primarily because Mechanical Turk offers no facilities for enforcing between-subjects designs. Furthermore, we felt that there would be minimal learning effects if people switched between conditions.

To avoid having few votes at the beginning of the HIT and too many rushed votes at the end (which we saw in an earlier iteration of the experiment), we added a new HIT each day rather than having a single HIT last two weeks.

Since our task is one of identifying intentionally misleading websites, sites which criminals have deliberately built to deceive, we required our participants to complete a short training task using the first two rounds of Anti-Phishing Phil, which has been shown to increase phishing recognition in those who play it [73].

Though our model site, PhishTank, does not explicitly train users, we assume that users who participate there are more likely to be familiar with how to identify phish than our users recruited through MTurk, because of the selection bias of a volunteer opting to donate time to participate. We also chose to train users to decrease the likelihood of low identification performance that lower participation users exhibit on PhishTank [62]. Once users completed both rounds of Anti-Phishing Phil, they were then eligible to complete our HITs. Participants who completed the game spent an average of 5.2 minutes (SD = 6.5 minutes).

3.4.2 Task Design for Mechanical Turk

Fig 3.3 shows the Aquarium user interface that was presented to MTurk users. Our interface was modeled to be functionally similar to PhishTank's site, with the primary difference being that our interface did not display any voting progress indicators, unlike PhishTank which displays a breakdown of the voting percentages after a user has voted.

To complete a HIT, a participant chooses one among "Phish", "Not Phish", or "Don't Know". In the Cluster Condition, participants can also select "Mark All as Phish" and "Mark All as Not Phish".

We sampled data from the PhishTank site on an ongoing basis, extracting newly-submitted potential phish, typically within minutes of being submitted to PhishTank. We crawled new submissions using an automated tool that we created that collects a screenshot as well as the raw

content used in an actual browser rendering the suspicious site. This process was run in a virtual machine to protect against any content or malware attacks, and virtual machines were reset to a clean state approximately every 10 minutes. This data collection process allowed us to protect our participants from any possible malware, as well as provide a more uniform experience robust to some kinds of fast flux where phishing sites temporarily shut themselves down to interfere with detection (which we attempt to overcome by regularly re-checking sites which were previously down), network problems, or differences between participants' browsers and security settings.

The raw content extracted from each site by our tools represents all of the data that is required by a web browser to render the web page, including the final internal document representation that is used to render the web page on a normal user's screen, and is the content upon which we cluster submissions.

Once we had all the information for a submitted site, we added it to our live study site, where users were given tasks based on (a) closest to 4 minimum votes, and then (b) newest submission.

3.5 Experimental Result

3.5.1 Summary of Participation Data

During the 2 weeks of this study, we had 267 users visit Aquarium, with 239 users participating. Of these 239, 174 cast votes in both conditions (as stated earlier, participants only saw a given URL at most once), 26 in the Control Condition only, and 39 users in the Cluster Condition only.

A total of 33,781 votes were placed, with 16,308 in the Control Condition, and 11,463 votes on clusters (yielding an equivalent of 17,473 votes on URLs without clustering) in the Cluster Condition. We paid $277.71 for the users for completed and approved HITs, and $198.96 to Amazon for approved HITs and bonus rewards, yielding a total cost of $476.67.

Because we only presented tasks to our participants if we were able to generate a thumbnail and download the site's content, our feed of submitted phish was not as large as PhishTank's, having 3,973 of the 5,686 submissions available from PhishTank. There were 1,713 submissions not used in the experiment since we could not obtain their screenshots.

We compared our results to four different resources. The first is the labels from PhishTank. We periodically checked the status of a given URL on PhishTank. If PhishTank updated their information, we would update our database accordingly. The second is the Google Safe Browsing API, which checks a given URL against their blacklist. We periodically checked the status of given URL using this API. The third is the SmartScreen Filter used by Microsoft Internet Explorer. We created a program that instantiated the MSIE browser in a virtual machine, visited the suspicious URL, and then analyzed the response of the IE browser to verify whether it is phishing or not. Fourth, when we could not obtain the status of a suspicious URL from above methods, we manually checked it. We use a queue to store the unverified URLs and repeatedly checked them following FCFS (First Come First Served) service discipline until they are verified. At worst, these URLs are checked every 10 minutes. We manually checked those URLs unverified by the first three methods during a given time (i.e. two weeks). In our study, we only manually checked 137 URLs, the majority of which were checking sites labeled as not phish. We also did not see any disagreement in the blacklists during the study period.

Using the above methods, we identified 3,877 as phishing URLs and 96 as legitimate. Table 3.2 shows the comparison of PhishTank, Google Safe Browsing, and Microsoft's SmartScreen Filter during the study period. The difference between the average time and median time is huge, due to a power law distribution, which has also been reported in past work [62].

Table 3.2: Comparison of the coverage and time of 3,973 URLs among PhishTank, Google Safe Browsing, and Microsoft's SmartScreen Filter. Given past work in this area, we assume that the false positive rates of Google's and Microsoft's techniques are 0%.

	Coverage rate	Average time (hours)	Median time (hours)
PhishTank	89.2%	16.4 (SD = 25.3)	3.98
Google safe browsing	65.7%	10.1 (SD = 10.1)	8.47
SmartScreen Filter of MSIE	40.4%	24.5 (SD = 24.0)	15.01

Also note that our reported coverage rates are different than from those in our past work [74]. This is primarily due to our source of phish, which is drawn from PhishTank rather than the UAB feed which is more comprehensive and has fresher phish. These previous results should still be considered more representative of blacklist behavior. Our results here should be viewed as a relative comparison of anti-phishing techniques on a sample of phishing attacks, rather than an absolute comparison.

Table 3.3: Comparison of true positive rate and false positive rate of all votes in the two conditions, as well as all labeled URLs based on those votes. The TP of votes from the Control Condition to the Cluster Condition improved by 6.4%, which was statistically significant (p = 0.026). There were no other differences, however.

	TP	FP
All Votes in Control Condition	83.0%	2.6%
All Labeled URLs in Control Condition	94.8%	0.0%
All Votes on Clusters in Cluster Condition	89.4%	0.05%
All Labeled URLs in Cluster Condition	95.4%	0.0%

Table 3.4: Comparison of URLs labeled in the two conditions. Due to tie votes, there are 96.1% URLs labeled in the Control Condition and 96.8% URLs in the Cluster Condition. FP was reduced to zero in both conditions.

	Coverage rate	Average time (hours)	Median time (hours)
All Labeled URLs in Control Condition	96.1%	11.8 (SD = 22.6)	3.8
All Labeled URLs in Cluster Condition	96.8%	1.8 (SD = 2.6)	0.7

Table 3.3 and 3.4 show the results of our two conditions. In Table 3.3, the first row "All Votes in Control Condition" shows the TP and FP of all 16,308 votes cast in that condition. Note that we saw a FP rate of 2.6%, which is fairly high. The second row "All Labeled URLs in Control Condition" shows the results of our labels when compared to our four resources (i.e. PhishTank, Google Safe Browsing, Microsoft's SmartScreen Filter, and manual checks). Note that the labels for URLs have a reasonably good TP rate (94.8%), which is higher than individual votes (83.0%). Aggregating people's votes also led to 0% FP in our experiment. We saw no systematic errors in false positive votes.

The third row of Table 3.3, "All Votes on Clusters in the Cluster Condition", shows the TP and FP of all 11,463 votes on clusters. The fourth row, "All Labeled URLs in Cluster Condition", shows a comparable TP and FP rate to the Control Condition.

Table 3.4 shows Aquarium's performance with respect to coverage and time. Here, Aquarium does quite well compared to PhishTank, Google, and Microsoft. The coverage rate of our Control and Cluster Conditions (96.1% and 96.8% respectively) is higher than the other blacklists (89.2%, 65.7%, and 40.4%). However, it should be noted that our recorded coverage rates for PhishTank, Google, and Microsoft do not take into account phishing pages that are taken down, since these blacklists may not bother labeling a phish that no longer exists.

To a large extent, this problem of not labeling a page that no longer exists would be less of a problem if blacklists could label pages faster, which would also provide better protection for people in the first few hours of an attack when people are most vulnerable [50]. Table 3.4 shows the average and median time to label a page in our two conditions. In particular, the clustered condition offers the best average time (1.8 hours, SD = 2.6 hours) as well as median time (0.7 hours), outperforming all other blacklists by a wide margin.

3.5.2 Individual Human Accuracy

Earlier, we had hypothesized that clustering could help people identify phish better. For example, a given participant might not recognize a single instance of phish on an unknown brand (for example, a single instance of the Tibia phish shown in Fig 3.4), but seeing four instances of the same site but with different URLs would suggest that it is suspicious.

We calculated each individual's accuracy (true positives plus true negatives over all votes) based on votes in both conditions. Individual performance varied, with a mean accuracy of 82.7% (SD = 23.3) in the Control Condition, and a mean accuracy of 86.7% (SD = 18.5) in the Cluster Condition. In our experiment, 174 of these 239 users cast votes in both conditions. We compared their performance between two conditions using a paired t-test with one-tailed distribution. There was a statistically significant effect for clustering, $t(173) = 2.78$, $p < 0.05 (p = 0.006)$, with users' performance in the Cluster Condition obtaining higher accuracy than that in Control Condition. As such, our results support our hypothesis that clustering helps people identify phish better. We also examined if the size of a cluster helped with accuracy. There was marginal improvement, but not statistically significant.

Figure 3.5 shows the overall performance of all participants sorted by performance and organized into deciles. The top 50% of participants performed very well in both conditions. However, there is a large dropoff in performance in the Control Condition, with the bottom 10% of MTurkers in the Control Condition performing under 30%. We suspect this is due to lazy workers.

3.5.3 Reducing Effort Using Task Clustering

To determine if the Cluster Condition is more effective in determining if a submission is a phishing attack, we looked for a difference in the performance of users in evaluating submissions. Time to label is an important metric here, as it measures both how quickly a user was able to identify an attack, and is a coarse representation of the effort required to complete the task.

Participants in the Control Condition took 11.8 (SD = 22.6) hours on average to label a site, whereas participants in the Cluster Condition took 1.8 (SD = 2.6) hours. By comparing two conditions with one-tailed paired t-Test, there was a significant main effect on clustering, $t = 23.63, p < 0.001$, with much less time used in identifying a URL in Cluster Condition than that in Control Condition. Comparing median times (3.8 hours to 0.7 hours) yields a similar result.

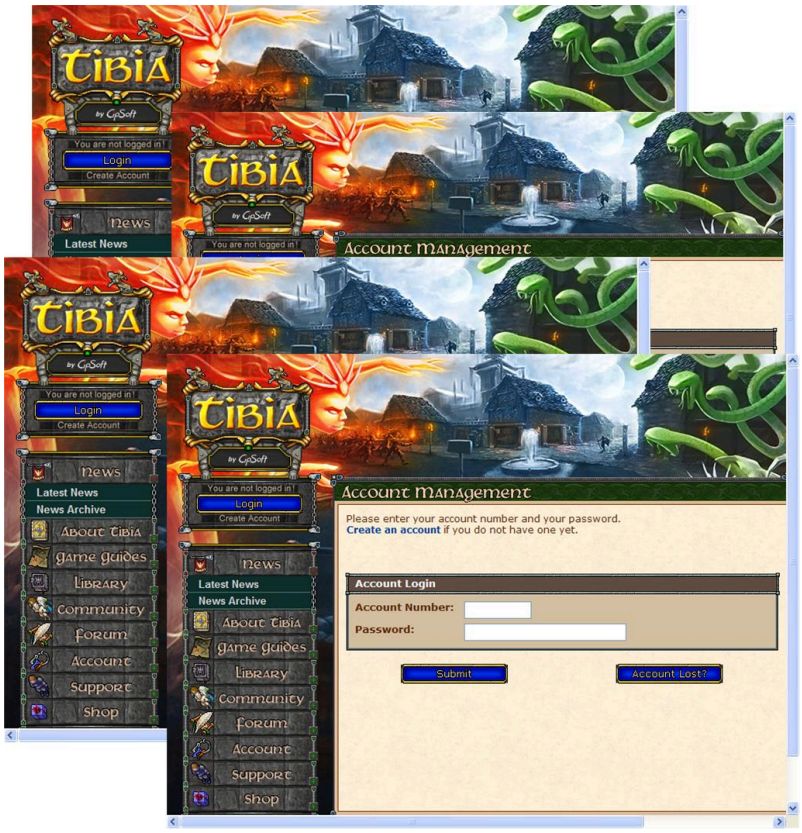

Figure 3.4: Four phishing examples from a cluster collected during our study. In this case, all 42 submissions in the cluster were nearly or completely visually identical. In this case, it should not be hard to identify the phishing attack, as the sites are identical, but do not share the same primary domain name.

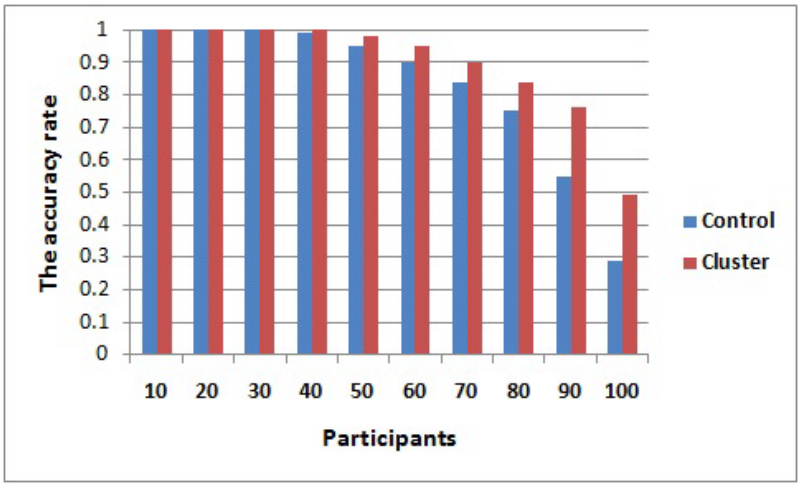

Figure 3.5: Average accuracy for each decile of users, sorted by accuracy. For example, the average accuracy of the top 10% of users in both conditions was 100%, whereas the average accuracy of the bottom 10% was under 30% for the Control Condition and under 50% in the Cluster Condition.

3.5.4 Tuning the Voteweight Parameters

In this experiment, we tuned the parameter required in Eq 3.4 to optimize the accuracy rate and time cost in labeling URLs.

We tested on 16,308 votes from the Control Condition on 3,973 URLs, and 11,463 votes on clusters from the Cluster Condition on 3,973 URLs using different values of ranging from 0.5 to 9 in increments of 0.5. Again, a higher here means that incorrect votes incur a higher penalty relative to the reward. We calculated the accuracy and time cost when a URL was identified based on different values of voteweight.

We also tested the threshold of voteweight from 0.01 to 1 by steps of 0.01. Figure 6 only shows the results under the threshold from 0.01 to 0.20, since there was no improvement above 0.20. Again, our label for a URL was determined based on the voteweight it obtained beyond the threshold. For this tuning, we recalculated voteweight after each hour.

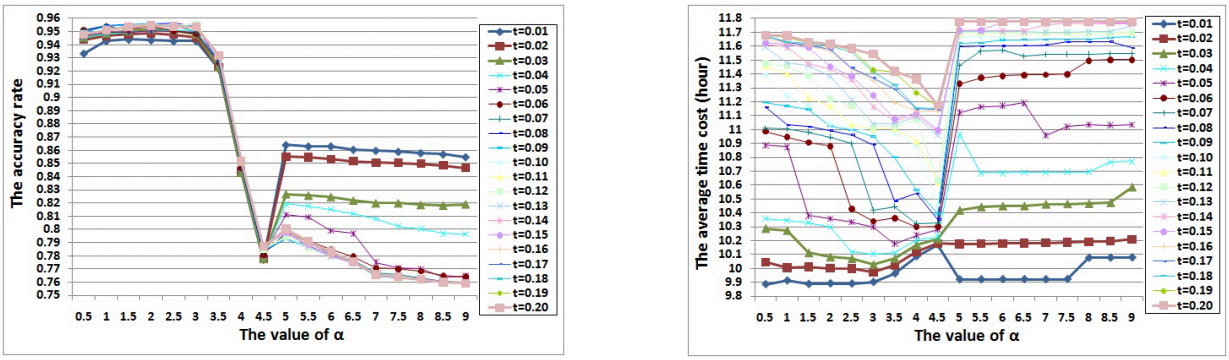

Figure 3.6: Voteweight parameter tuning in the control condition (left:accuracy, right:average time cost). t is the threshold of votepower, α is the weight of penalty for wrong verification of URLs. As α increases, the accuracy first increases a little and then drops down quickly while the average time cost increases in a small range. When t increases, the average time cost increases accordingly. Voteweight achieves its best accuracy with $t = 0.08$ and $\alpha = 2.5$.

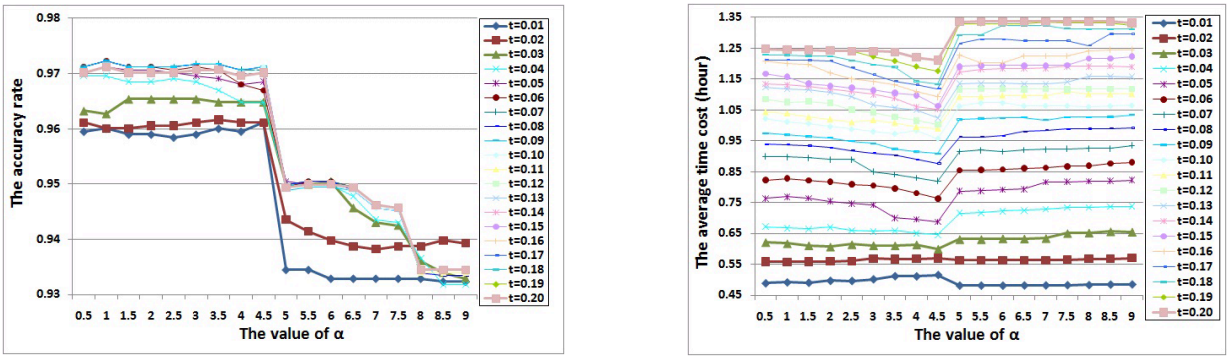

Figure 3.7: Voteweight parameter tuning in the cluster condition (left:accuracy, right:average time cost). Voteweight achieves its best accuracy with $t = 0.06$ and $\alpha = 1$.

Figure 3.6 and 3.7 show our results for both the Control Condition and Cluster Condition. The left plot in Fig 3.6 shows how accuracy varies as the value of α increases. Accuracy increases incrementally for a while and then plummets dramatically when α approaches 4.5 regardless of

the threshold. This finding suggests that an appropriate penalty can offer a small benefit in terms of distinguishing between skilled and unskilled participants. However, excessive punishment on occasional errors of users dramatically decreases performance. The right plot in Fig 3.6 shows that as the threshold t is increased, the time cost of identifying a URL also increases, as one might expect.

Overall, in the Control Condition, voteweight obtained its highest accuracy (true positives and true negatives over all votes) of 95.6% when the threshold t is 0.08 and α is 2.5. At these values, the average time cost was 11.0 hours and median time cost was 2.3 hours.

Given this tuning, how much improvement does voteweight offer over not having it? In the Control Condition (without voteweight), Table 3.3 shows that the TP was 94.8% and FP was 0%, with an average time cost of 11.8 hours (SD = 22.6) and a median of 3.8 hours. By using voteweight, we can achieve a comparable accuracy, and reduce the average time by 0.8 hours and reduce the median time by 1.5 hours.

In the Cluster Condition, we obtain the highest accuracy rate at 97.2% when threshold t is 0.06 and α is 1. With this accuracy rate, the average time is 0.8 hours and median time is 0.5 hours. In the Cluster Condition without voteweight, the true positive rate is 95.4%, with an average time of 1.8 hours (SD = 2.6) and median time of 0.7 hours. By using voteweight, we can reduce the average time cost by about 1 hour and reduce the median time cost by about 0.2 hours, while achieving comparable accuracy.

Chapter 4

Enhancing URL Blacklists with Adaptive Probabilistic Techniques

To augment the rigid human-verified blacklists, we proposed in this chapter a hierarchical blacklist-enhanced method [1] for phish detection, which leverages existing blacklists and applies fuzzy matching techniques to detect phish in a probabilistic fashion with very high accuracy.

In combating phish, industry relies heavily on manual verification to achieve a low FP, which, however, tends to be slow in responding to the huge volume of unique phishing URLs created by toolkits. Naturally, one potential improvement is to combine the best aspects of human verified blacklists and feature-based methods, i.e., the low FP of the former, and the broad and fast coverage of the latter.

To this end, we present the design and evaluation of a hierarchical blacklist-enhanced phish detection framework. The key insight behind this detection algorithm is to leverage existing human-verified blacklists and apply the shingling technique, a popular near-duplicate detection algorithm used by search engines, to detect phish in a probabilistic fashion with very high accuracy. To achieve an extremely low FP, we use a filtering module in our layered system, harnessing the power of search engines via information retrieval techniques to correct false positives.

4.1 Introduction

A significant proportion of the losses due to phishing attacks were caused by one particularly infamous group, known as the "rock phish gang", which uses phish toolkits to create a large number of unique phishing URLs [61], putting additional pressure on the timeliness and accuracy of blacklist-based anti-phishing techniques.

Generally, blacklist-based anti-phishing techniques have a very low FP. However, human-verified blacklists do not generalize well to novel phishing attacks and can be slow to respond to new phishing attacks. Moreover, updating blacklists usually involves enormous human effort, and human-verified blacklists can be easily overwhelmed by automatically generated URLs. On the other hand, feature-based approaches enjoy the flexibility of being able to recognize new phish, but often lead to a relatively higher false positive rate.

The goal of our work in this chapter is to combine the best aspects of human verified blacklists and heuristics-based methods, and develop a reliable and robust method that is able to adaptively

[1] Parts of this chapter were previously published in the Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS'10) [88]

generalize to new attacks with reasonable true positive rate (TP) while maintaining a close to zero FP. Our approach exploits the fact that a large number of current phishing attacks are created with toolkits, which tend to have a high similarity in terms of content. Our detection engine analyzes the content of phishing web pages on manually-verified URL blacklists via n-grams, and employs the shingling technique to identify near-duplicate phish in a probabilistic fashion. We also use a filtering module, which uses information retrieval (IR) techniques querying search engines to further scrutinize the legitimacy of a potential phish in an effort to control false positives. Our whole system is constantly updated by a sliding window upon the arrival of new phishing data, and is thus capable of adapting quickly to new phishing variants, while still maintaining a reasonable level of runtime performance. Under the optimal experimental setup, our method achieves a TP of 67.15% with 0% FP using search oriented filtering, and a TP of 73.53% and a FP of 0.03% without the filtering module, much better than blacklist-based methods in TP while comparable in FP. For applications like anti-phishing where FP is of paramount importance, a slightly lower TP is acceptable. Furthermore, we do not expect our approach to be used alone, but rather reside in the first part of a pipeline augmenting the existing system such as the commercial blacklists, thus fabricating a superior integrated solution.

We do not claim that our approach will solve the phishing problem. Rather, our specific claim is that we can augment existing blacklists in a very conservative manner using probabilistic techniques, with a very low FP, if not zero, and a reasonably good TP. Capable of identifying a fair amount of phishing attacks with no sacrifice on FP and considerably reducing the human effort involved in manual verification, our approach significantly complements the prevalent blacklist-based methods, leveraging the manual labor that is already being used in verifying phishing sites. The major contributions of our technique in this chapter are three fold.

1. We present the design of a novel hierarchical, content-based approach that leverages existing human-verified blacklists, by making use of shingling and information retrieval techniques to detect phish.
2. We demonstrate that with incremental model building via a sliding window mechanism, our approach is able to adapt quickly to the constantly evolving zero-hour phish attacks. Also, we only need the most recent 30 days' worth of data to achieve the same TP as using two months' worth of data, thus balancing accuracy with runtime efficiency.
3. By harnessing URL blacklists in a probabilistic fashion, we are able to leverage our approach to improve the coverage and timeliness of human-verified blacklists using considerably less human effort than existing techniques, without having to sacrifice the false positive rate. With only two weeks' worth of phish, our method achieves a TP of 65.02% with 0% FP using search oriented-filtering, and a TP of 71.23% and a FP of 0.03% without filtering.

4.2 Toolkits for Creating Phishing Sites

In recent years, an increasingly large number of phishing web pages were automatically created by toolkits, which substantially increases the scale of attacks that criminals can attempt, while also countering current human-verified blacklists. For example, Cova et al [26] identified 584 phishing kits during a period of two months starting in April 2008, all of which were written in PHP. An analysis of such phishing sites, usually called rock phish, by Moore et al [61] from February to April in 2007 reveals that 52.6% of all Phishtank reports were rock phish. One key observation of the rock phish is that their content is highly similar due to the way they are created, which

is the property that our framework is based on. It is possible that criminals may modify their toolkits to include randomization to circumvent our detection mechanisms, and we discuss this issue towards the end of this chapter.

4.3 A Multi-layered Phish Detection Algorithm

4.3.1 System Architecture

The overall architecture of our framework is shown in Fig 4.1. The first stage of processing involves filtering using domain whitelists, directly passing known benign web pages. The detection engine employs a technique based on shingling to classify the remaining web pages, forwarding potential phish to the FP filter for further examination, which interacts with search engines to correct false positives. New phish from blacklists are added into our training set via a sliding window to update the detection model with the latest phishing patterns.

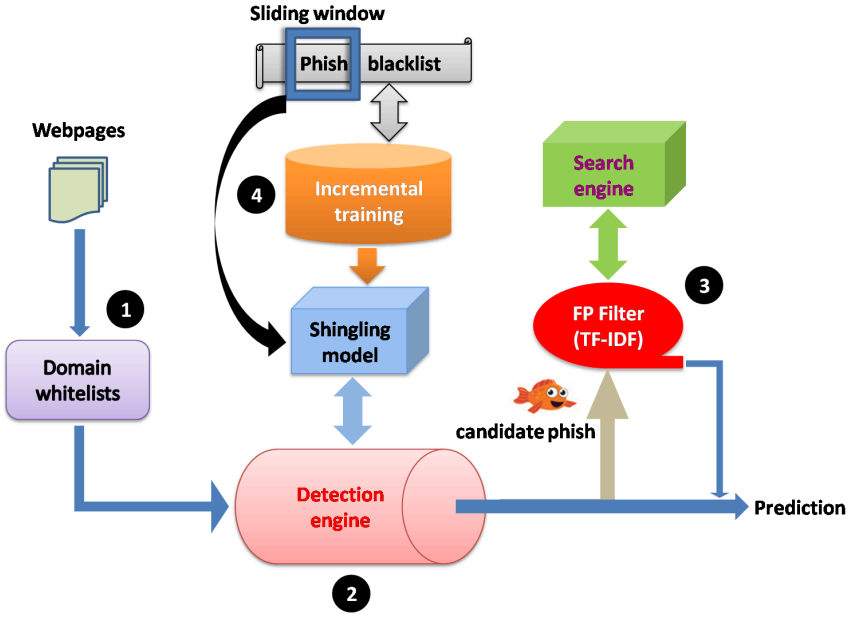

Figure 4.1: System architecture for our adaptive probabilistic blacklist. An incoming web page is first checked against a small domain whitelist (1). If a match is not found, our detection engine (2) compares the content of the web page against the content of existing phish using the shingling algorithm. If a page is flagged as a potential phish, we check for false positives, resorting to search engines (3) if needed for additional verification. We use a sliding window (4) in the back-end to incrementally updating the machine learning model as new phishing signatures arrive.

4.3.2 Shingling-based Probabilistic Matching

The essence of our detection algorithm is to do "soft" matching of a given web page against known phishing pages. The intuition here is that many phishing web pages are created by toolkits, and thus have many semantic similarities in terms of page content. Our detection method manipulates this semantic uniformity via soft matching, which allows more flexibility

than the rigid URL matching adopted by major blacklist-based methods. Our early evaluations using exact matching with hash codes of page content turned out to be reasonably effective, but also brittle and easy to defeat. As such, we want to make our system robust to simple changes, thus raising the bar for criminals.

Shingling [19], a technique for identifying duplicate documents, examines the web page content on a finer-grained level via the notion of n-gram, and measures the inter-page similarity based on these basic units. N-grams are subsequences of n contiguous tokens. For example, suppose we have sample text ***connect with the eBay community***. This text has 3-grams {***connect with the, with the eBay, the eBay community***}. Shingling employs a metric named *resemblance* to calculate the percent of common n-grams between two web pages. More formally, let q and d represent a web page being examined and a phishing page in the blacklist respectively. Let D represent the set of all training phish, and $S(p)$ denote the set of unique n-grams in p. The similarity metric *resemblance* $r(q, d)$ is then defined as $r(q, d) = |S(q) \cap S(d)|/|S(q) \cup S(d)|$. Our soft matching approach first generates the set of n-grams for each $d \in D$. We then compute $r(q, d)$ $\forall d \in D$ for a query page q, and fire an alarm whenever $r(q, d)$ exceeds a threshold t. We choose the optimal t via cross validation.

4.3.3 Search Engine Enhanced Filtering

As we will show later on in the evaluation, shingling is effective in comparing a given web page against known phish. However, a potential problem is with false positives. More specifically, phishing web pages usually imitate legitimate web pages, which means that if there are no safeguards in place, shingling by itself is likely to label those target legitimate cases as phish as well. To solve this problem, we propose a filtering algorithm leveraging the power of search engines via information retrieval techniques. This module, based on one heuristic in CANTINA [92], compensates for the incompleteness of domain whitelists, and is able to minimize FP even for less popular phishing target sites.

Our filtering module is triggered when the detection engine recognizes a candidate phish, and works by executing in Google queries composed of K top keywords chosen from the page content plus the web page domain keyword [2] and examining the presence of the page domain in the top N search results. The final prediction is restored to "legitimate" if the top N entries subsume the page domain, and thus we no longer incorrectly label such sites as phish. The validity of this filtering algorithm is partially attributed to the fact that legitimate websites are very likely to be indexed by major search engines, while phishing sites are not, due to their short-lived nature and few in-coming links.

We currently use $K = 5$, $N = 30$ according to the tuning result in [92][85]. Candidate query terms on the page are ranked by the TF-IDF scoring function widely used in IR, which selects the terms that are most representative of the web page content. The rationale is that search engines use TF-IDF when they match queries to documents in such a way that terms with high TF-IDF scores are the ones that have more influence over retrieval and ranking of documents.

4.3.4 Incremental Model Building via Sliding Window

To incorporate the latest phishing signatures into our database and to improve the runtime performance of our whole system, we utilize a sliding window of the most recent phish from phish

[2]The domain keyword is the segment in the domain representing the brand name, which is usually the non-country code second-level domain or the third-level domain.

blacklists and incrementally build the detection model with those phishing web sites. In our evaluation, we show that discarding older data as the sliding window moves actually has little impact on accuracy.

Furthermore, a positive side effect of using a sliding window is that the time complexity of shingling is reduced from $O(|D|)$ to $O(|D_{win}|)$, where D_{win} represents all phishing data covered by the current sliding window win. Asymptotically, $|D_{win}|$ can be deemed as a large constant, and in light of this shrunk magnitude, we refrain from trading accuracy in exchange of speed via approximated algorithms as used in many applications [40]. For example, this sliding window could reduce a year's worth of phish to just a month's worth, achieving $\times 12$ runtime speedup without significantly sacrificing detection performance.

4.4 Experiment Setup

4.4.1 Domain Whitelists

An enormous percentage of phishing frauds target well-known financial entities like eBay, Paypal, etc., by imitating their sites, and it is of practical value to pass those legitimate websites without feeding them to our detection engine. To reduce false positives and improve runtime performance, we quickly eliminate these known good sites through a whitelist. specifically, we collected known good domains from two sources. Google safe browsing provides a publicly-available database [3] with legitimate domains, and we obtained a total of 2758 unique domains from this whitelist after duplicate removal. Millersmiles [2] maintains an archive of the most common spam targets such as ebay, and we extracted 428 unique domains out of 732 entries after mapping organization names to domains and removing duplicates. In total, we had 3069 unique domains in our whitelist.

4.4.2 Web Page Corpus

Phishing sites are usually ephemeral, and most pages do not last more than a few days typically because they are taken down by the attackers themselves to avoid tracking, or taken down by legitimate authorities [74]. To study our approach over a larger corpus, we downloaded phishing pages when they were still alive and ran experiment offline. Our downloader employed Internet Explorer to render the web pages and execute Javascript, so that the DOM of the downloaded copy truly corresponds to the page content and thus gets around phishing obfuscations.

Our collection consists of phishing cases from PhishTank, and good web pages from seven sources. To eliminate the influence of language heterogeneity on our content-based methods, we only downloaded English web pages.

For phishing instances, we used the verified phishing URLs from the phish feed of Phishtank [65], a large community-based anti-phishing service with $59,884$ active accounts and $1,038,001$ verified phish [66] by the middle of Feburary 2013. For this work, we started downloading the feed in late February of 2009 and collected a total of 1175 phishing web pages from February 27, 2009 to April 2, 2009. All seven legitimate corpora were downloaded after April 2, the details of which are given in Table 4.1. Note that the open directory project is the most comprehensive human-edited directory of the Web maintained by a vast community of volunteers, and by using this corpus, we want to verify that our algorithm achieves a very low FP on the low-profile and less popular sites.

Table 4.1: Legitimate collection with a total of 3336 web pages.

Source	Size	Crawling Method
Top 100 English sites from Alexa.com	958	Crawling homepages to a limited depth
Misc login pages	831	Using Google's "inurl" operator and searching for keywords like "signin"
3Sharp [15]	87	Downloading good web pages that still existed at the time of downloading
Generic bank category [4] on Yahoo directory	878	Crawling the bank homepages for a varying number of steps within the same domains
Other categories of Yahoo directory	330	Same as the generic bank category
The most common phishing targets	69	Saving login pages of those sites
The open directory project [5]	183	Downloading "least popular" pages with zero pagerank

4.4.3 Test Methodology

For notational convenience, we define in Table 4.2 the free variables in our context. Our experiment here focused on tuning these variables to optimize our results. To simulate a more realistic scenario, we processed data in chronological order in all of our experiments. In assessing TP, we move the sliding window of length L step by step along the time line and apply our detection algorithm to the web pages at each time point T_i using a shingling model built on the phishing data with time labels falling in window $[T_{i-L}, T_{i-1}]$. The FP is tested in a slightly different manner. In [35], Fetterly et al discovered through large-scale web crawling that web page content was fairly stable over time, and based on that finding, we did not download the same set of legitimate pages at each time point but rather downloaded only once the whole set at a time later than all the phishing timestamps. Sliding windows of different sizes L are used similarly. Under all cases, four whitelist combinations are exercised with our detection algorithm, i.e., Millersmiles, Google, none, and both whitelists.

Table 4.2: Definition of symbols.

Variable	Explanation	Variable	Explanation
G	granularity of time	L	sliding window length
W	whitelist	n	n-gram
r	resemblance	t	resemblance threshold

4.5 Experimental Result

4.5.1 Shingling Parameter Tuning

Figure 4.2 shows the validation performance under different values for n and t. For all n-grams in the evaluation, the TP monotonically decreased as we raised the resemblance bar higher. With a resemblance of 65%, shingling achieved over 66% TP under all shingle lengths, manifesting the considerable similarity in content among phish due to rock phish. Although FP worsens as t and n decrease, we still stick to $n = 3, t = 0.65$ in the remaining evaluation of our experiment, hoping for the best TP performance and counting on the TF-IDF filtering algorithm to control false positives. The tuning results under all other configurations of G, L and W exhibit the same pattern, and we do not report them here.

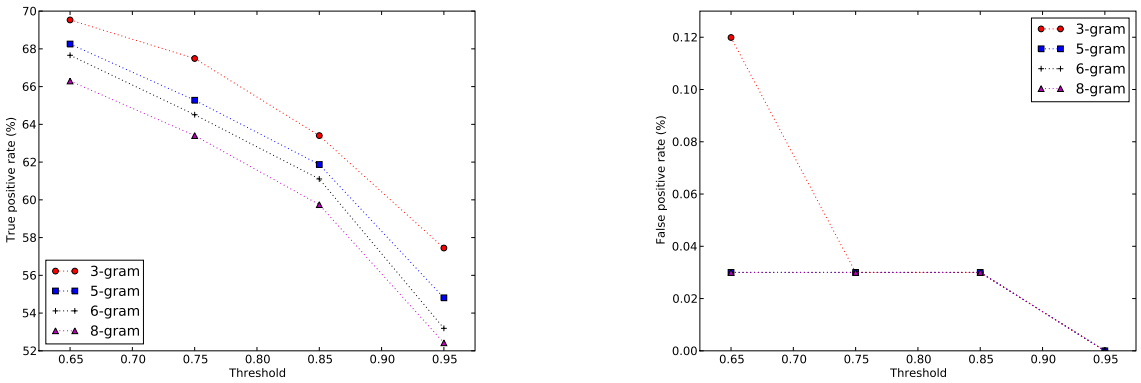

Figure 4.2: Shingling parameter tuning ($L = 60$, $G =$ day, $W =$ Millersmiles, no TF-IDF FP filtering). A tradeoff between TF (left) and FP (right) is an important factor in choosing t in the final model. As t is increased, the rate of detection drops and FP picks up. TP tops at 69.53% and FP reaches a culmination of 0.1199% under $n = 3, t = 0.65$. The other three FP curves $n = 5, 6, 8$ perfectly coincide.

4.5.2 Evaluation of True Positive Rate

Figure 4.3 suggests that even with only one day's worth of training phish, our algorithm is able to detect around 45% phishing attacks, demonstrating the efficacy of our method and also proving the conjecture that mainstream phishing attacks are created by toolkits.

Another finding is that when using search engines to filter false positives (the right plot in Fig 4.3 and Fig 4.4), TP dropped as a side effect. An explanation is that some phishing URLs (2%/1% with a 1-day/1-hour sliding window) are actually returned among the top 30 entries when querying TF-IDF ranked terms plus the domain keyword on Google and are thus mistakenly filtered as legitimate.

Real-time application of our algorithm does not suffer from this false filtering problem as much as observed in our offline experiment. A semi-formal explanation for this finding has two main points. First, when a new phish just comes out of attackers' workshop, few, if any, links point to that phishing site. As such, search engines are unlikely to return its domain as a top result; second, search engines might index the phish as time progresses when more links out in the web begin referring to it, however, the phish may have already become unavailable due to the short-lived nature of phishing activity and no harm will be done to the users even if it is incorrectly passed as a good page. The usefulness of this FP filtering module will become more evident when we embark on the analysis of FP in the following section.

Figure 4.3 and 4.4 suggest that the TPs under Millersmiles whitelist are universally better than those under Google whitelist. Examining both whitelists reveals that Millersmiles only contains a core group of the most spammed domains while the Google whitelist has many more less popular domains. None of the phishing domains in our corpus appear in the Millersmiles whitelist, however, some do show up in the Google whitelist, among which is "free.fr", occurring 6 times in our phishing set. Those phish were thus erroneously filtered, lowering the TP inadvertently. This observation delivers a message about the use of domain whitelists, i.e., the quality of whitelists does impact TP and the optimal usage is to adopt a small core whitelist covering a group of popular spam target sites. Our detection method performed convincingly better with respect to

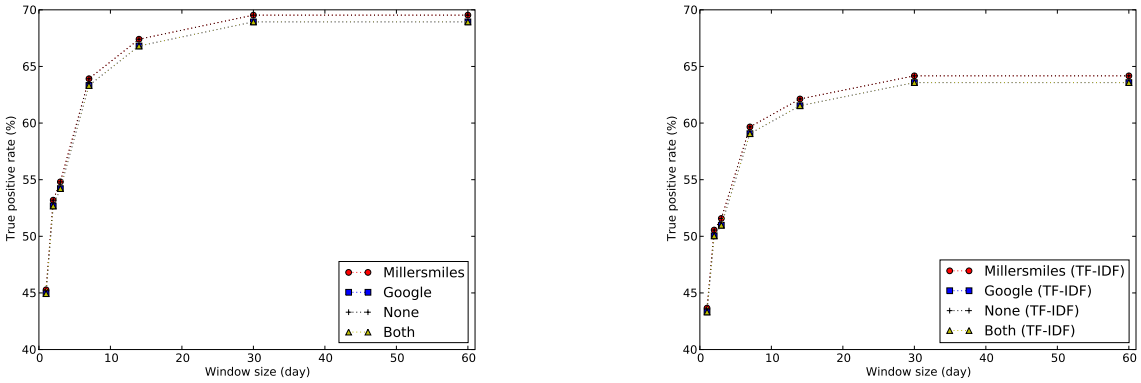

Figure 4.3: TP under various L (G = day) and W (left:without TF-IDF filter, right:with TF-IDF filter). Our approach achieves about 45% (no FP filtering) and 43% (with FP filtering) TP in all cases with only 1 day's worth of training phish, and around 69% (no FP filtering) and 64% TP (with FP filtering) with a 60-day window. FP filtering hurts TP, and a whitelist with only popular phishing targets beats a more comprehensive whitelist.

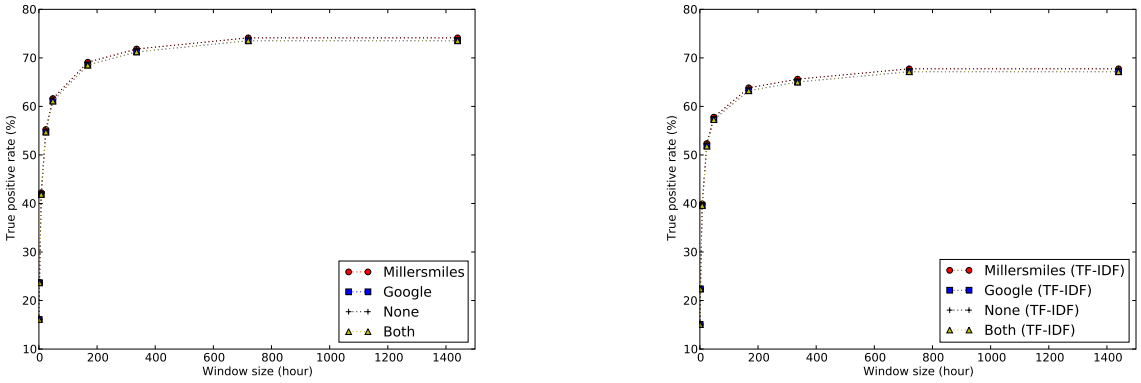

Figure 4.4: TP under various L (G = hour) and W (left:without TF-IDF filter, right:with TF-IDF filter). Under all whitelists, TP bottoms around 16% in all cases with a 1-hour window and peaks around 74% with a 1440-hour window without FP filtering; with FP filtering, TP bottoms around 15% with a 1-hour window and peaks around 67% with a 1440-hour window.

TP when the model is iteratively built on a hourly basis.

4.5.3 Evaluation of False Positive Rate

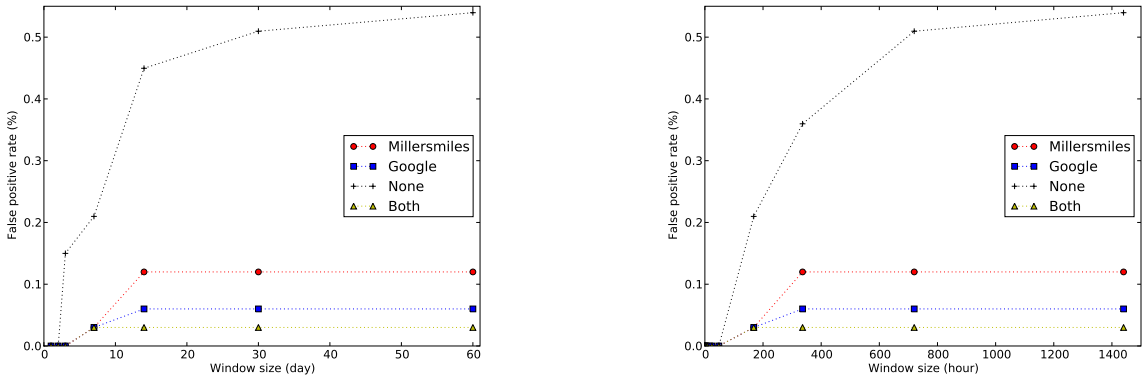

Figure 4.5: FP under various L and W with no TF-IDF filtering (left:window size by days, right:window size by hours). Under all whitelists, FP escalates with the growth of the sliding window size. FPs are zero when using TF-IDF to filter false positives under all settings and are not plotted here.

Figure 4.5 shows the FPs under different sliding window sizes and whitelists with no TF-IDF filtering. All four curves in both plots start with zero FPs, when L is minimum, and gradually escalate as more training phish are added to the model. Domain whitelists prove to be effective in suppressing false positives, with FPs of $0.1199\%, 0.06\%$, 0.5396%, 0.03% for Millersmiles, Google, none and both whitelists under both a 60-day window (left) and a 1440-hour window (right). With TF-IDF filtering, FPs are all zero under all circumstances, and we do not explicitly show the plots here.

4.5.4 Granularity of Time Unit for Window Size

A comparison of TPs with day and hour based L (Table 4.3 in the appendix) shows that under sliding windows of identical time length, hour-level incremental model building outperformed day-level building, indicating the superior responsiveness of hourly updating. The largest gaps occurred at a window length of 1 day (24 hours), amounting to TPs of 9.95%, 9.78%, 9.95%, 9.78% with no FP filtering and 8.68%, 8.51%, 8.68%, 8.51% with FP filtering under four whitelist configurations. This disparity gradually diminished as L increased, which is reasonable in that as more and more phish are absorbed into the training set by the growing window, the tiny amount of shift in time relative to the window size no longer has as large of an impact as before. Surprisingly, simply with a 24-hour window, our algorithm was able to achieve over 50% TP under all whitelists and filtering setups.

As expected, the FPs under two time units in Table 4.4 in the appendix are identical except for one cell, since all legitimate pages in our web collection were downloaded after the phishing ones and regardless of time measurement (day or hour), the sliding window with the same length in terms of time actually covered roughly the same set of training phish. Interestingly, the FP filtering module successfully removed all the false positives, leading to zero FP under all

experiment settings, at the cost of slight degradation on TP. Note that the evaluation of FP in our experiment is sound and thorough partially in that our legitimate corpus contains a diverse variety of data including those categories that are the worst case scenarios for phish detection. As a result, the experimental result offers conservative statistics that are more meaningful to the actual adoption and deployment of our system. As suggested by the statistics in Table 4.3 and Table 4.4, another feature of our system is that it offers an adjustable range of performance depending on a user or provider's willingness to accept false positives.

Table 4.3: TP (%) under day/hour-measured sliding window. Under all settings, shingling with hour-level incremental model building is more responsive to phishing attacks, attaining higher TPs under all L values. Our approach achieved almost optimal TP with only 1 month's worth of training phish.

	No TF-IDF filtering											
	Window size (day)						Window size (hour)					
Whitelist	1	2	7	14	30	60	24	48	168	336	720	1440
Millersmiles	45.28	53.19	63.91	67.4	**69.53**	**69.53**	55.23	61.62	69.11	71.83	**74.13**	**74.13**
Google	44.94	52.68	63.32	66.81	**68.94**	**68.94**	54.72	61.11	68.51	71.23	**73.53**	**73.53**
None	45.28	53.19	63.91	67.4	**69.53**	**69.53**	55.23	61.62	69.11	71.83	**74.13**	**74.13**
Both	44.94	52.68	63.32	66.81	**68.94**	**68.94**	54.72	61.11	68.51	71.23	**73.53**	**73.53**
	With TF-IDF filtering											
	Window size (day)						Window size (hour)					
Whitelist	1	2	7	14	30	60	24	48	168	336	720	1440
Millersmiles	43.66	50.55	59.66	62.13	**64.17**	**64.17**	52.34	57.79	63.83	65.62	**67.74**	**67.74**
Google	43.32	50.04	59.06	61.53	**63.57**	**63.57**	51.83	57.28	63.23	65.02	**67.15**	**67.15**
None	43.66	50.55	59.66	62.13	**64.17**	**64.17**	52.34	57.79	63.83	65.62	**67.74**	**67.74**
Both	43.32	50.04	59.06	61.53	**63.57**	**63.57**	51.83	57.28	63.23	65.02	**67.15**	**67.15**

Table 4.4: FP (%) under day/hour-measured sliding window. Whitelists lessen the FPs, reaching 0.1199%, 0.06%, 0.5396%, 0.03% respectively with the Millersmiles, Google, none and both whitelists at $L = 60$ days or $L = 1440$ hours. The search engine oriented filtering step significantly significantly improves the FPs , downsizing FP values in all settings to zero.

	No TF-IDF filtering											
	Window size (day)						Window size (hour)					
Whitelist	1	2	7	14	30	60	24	48	168	336	720	1440
Millersmiles	0.00	0.00	0.03	0.1199	0.1199	0.1199	0.00	0.00	0.03	0.1199	0.1199	0.1199
Google	0.00	0.00	0.03	0.06	0.06	0.06	0.00	0.00	0.03	0.06	0.06	0.06
None	0.00	0.00	0.2098	0.4496	0.5096	0.5396	0.00	0.00	0.2098	0.3597	0.5096	0.5396
Both	0.00	0.00	0.03	0.03	0.03	0.03	0.00	0.00	0.03	0.03	0.03	0.03
	With TF-IDF filtering											
	Window size (day)						Window size (hour)					
Whitelist	1	2	7	14	30	60	24	48	168	336	720	1440
Millersmiles	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Google	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
None	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Both	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Chapter 5

Detecting Phish via Textual Identity Discovery and Keywords Retrieval

In this chapter, we introduce our technique that detects phish by discovering the inconsistency between the claimed brand and the genuine brand of a web page [1].

Among all the information on a phishing web page, the identity-bearing elements are essential for the phish detection task, such as the brand name text, brand logo images, etc. However, none of the previous research has explored this key piece of information, and in this chapter, we propose a technique for exploiting the brand name in the textual content of a web page for phish detection. In particular, our method has two components. While the identity-based component detects phishing web pages by directly discovering the inconsistency between their identity and the identity they are imitating, the keywords-retrieval component utilizes information retrieval (IR) algorithms exploiting the power of search engines to identify phish. Our method requires no training data, no prior knowledge of phishing signatures and specific implementations, and thus is able to adapt quickly to constantly appearing new phishing patterns.

5.1 Introduction

Phishing patterns evolve constantly, and it is usually hard for a detection method to achieve a high true positive rate (TP) while maintaining a low false positive rate (FP) simultaneously. In this chapter, we propose a novel hybrid detection method based on information extraction (IE) and information retrieval (IR) techniques in an attempt to achieve a good balance between TP and FP. The identity-based detection component of our framework utilizes IR techniques to recognize the identity a web page claims and captures phish by examining the discrepancy between the claimed identity and its own identity. Named entity recognition (NER) algorithms are used to reduce false positives. This identity-oriented component is aided by a keywords-retrieval component that employs search engines to detect potential phish via searching keywords of significant importance with respect to IR. For instance, a phishing site in Fig 5.1 claims to be eBay, while actually its true identity is a phishing domain "ovmu98yn1xcy13281mz1.com". Our approach exploits this discrepancy as well as keywords of IR significance from the page ("ebay bid account password forgot") to catch it. To control false positives, we use a domain whitelist and a login form detector to filter good web pages. Experiments over a diverse spectrum of data

[1] Parts of this chapter were previously published in the Proceedings of the 18th International Conference on World Wide Web (WWW'09) [85]

sources with 11,449 pages showed that our approach achieved a true positive rate of 90.06% with a false positive rate of 1.95%.

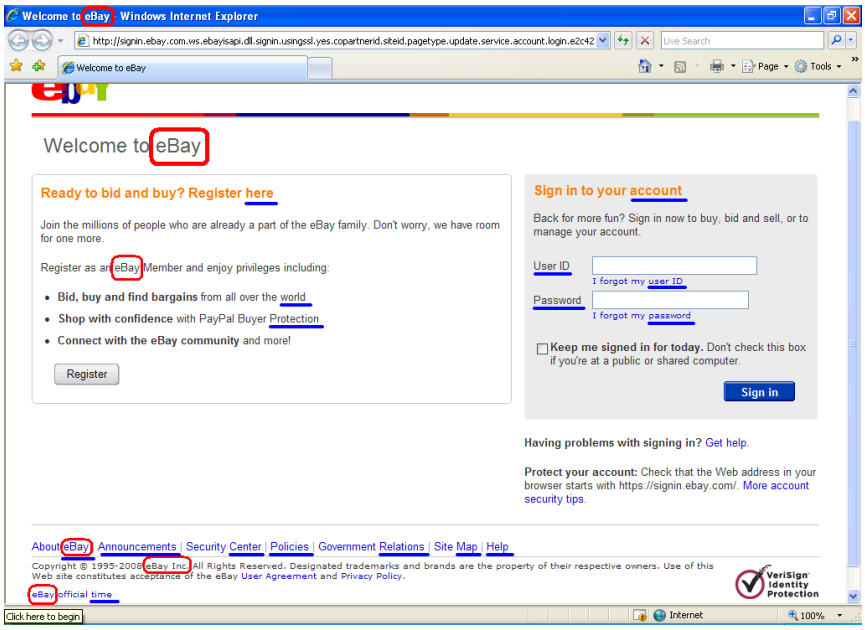

Figure 5.1: A phishing page targeting eBay with brand name circled in red and words missing ensuing punctuations underlined in blue. The identity-based detection component executes on search engines query *site:ebay.com "ovmu98yn1xcy13281mz1.com"* and detects the phish by zero search result.

One major advantage of our method is that it requires no training data, no prior knowledge of phishing signatures and specific implementations, and thus is able to adapt quickly to the constantly appearing new phishing patterns. Traditional blacklist-based method demands an up-to-date phish database to learn machine learning classifiers with high coverage, and thus is slow in responding to new phishing attacks. Another property of our approach is that it attacks the TP/FP dilemma by investigating two subcomponents both with a low FP and a reasonable TP yet focusing on different phishing patterns, and boosting the detection performance via an integrated system.

5.2 System Overview

Our hybrid detection approach exploits a few properties and common practices of website design:

1. Website brand names usually appear in a certain parts of a web page such as title, copyright field, etc, which renders the website identity searchable and recognizable. For example, term "eBay" appears in many places of its login page (Fig 5.1), as highlighted by red circles.
2. The universal practice of synchronizing the brand with a domain name lends legitimacy to the strategy of matching textual brand name with domain keyword to determine if a domain truly points to the website the brand refers to. The domain keyword is the segment in the domain representing the brand name, which is usually the non-country code second-level domain such as "Paypal" for "paypal.com" or third-level domain.

3. Phishing web pages are much less likely to be crawled and indexed by major search engines than their legitimate counterparts due to their short-lived nature and few in-coming links.
4. A phishing site usually provides a login form to request sensitive user information, which alone could serve as a feature in classifying web pages.

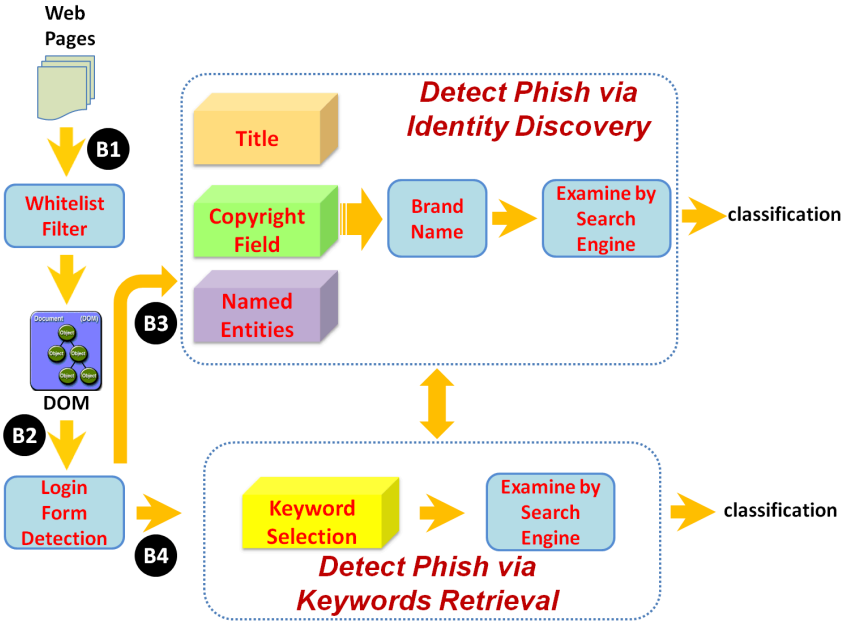

Figure 5.2: System architecture of our identity-based anti-phishing approach. Our system has an identity-based detection component and a keywords-retrieval detection component.

Our hybrid approach consists of an identity-based detection component and a keywords-retrieval detection component (Fig 5.2), both manipulating the DOM after the web page has been rendered in Internet Explorer to get around intentional obfuscations. The former relies on identity recognition to find the domain of the page's declared identity, and examines the legitimacy of the web page by comparing this extracted domain with its own domain via executing query of the form *site:declared_brand_domain "page_domain"* in search engines. The two domains in the query are deemed as pointing to the same identity if searching returns results. We do not directly match two domain strings in that some closely related domains (e.g., company affiliations) are literally different (such as "blogger.com" and "blogspot.com"). Using the "site" operator thus reduces such false positives. For the phishing example in Fig.1, our retrieval-based identity recognition module finds the brand domain "ebay.com" based on the brand name "eBay" in the title and copyright field. It then executes on search engines the query *site:ebay.com "ovmu98yn1xcy13281mz1.com"*, and finds no result, indicating the web page under examination is probably a phish. The NE identity recognition module augments the retrieval-based one in cases where brand names are absent in title and copyright field to control false positives.

Leveraging property 3 above, the keywords-retrieval component, a variant of CANTINA, first identifies from the page content and meta keywords/description tags a set of top ranking keywords using the well-known TF-IDF scoring function, and searches in search engines a query composed of top keywords plus the page domain keyword. Though not all search engines support meta tags, their content still sometimes subsumes valuable information such as the web site's identity.

A web page is regarded as a good page if the page domain appears in the top N search results. This augments the identity-based method in the scenario where web site identities are missing, thus leading to an improved true positive rate. This keywords-retrieval method is based on the work [92], which also explored the TF-IDF metric.

Domain whitelists and a login form detector sit in the front end of the system, filtering safe pages from further examination. Though this strategy tends to ignore the first a few pages with no login forms of a multi-page attack, it is still able to catch the phish as long as login forms appear eventually, and no harm has been done to the users so far.

5.3 Login Form Detection

In this section, we present a heuristics-based algorithm using the HTML DOM to identify login forms. Typically, the presence of login forms on a page is characterized by three properties, i.e., FORM tags, INPUT tags and login keywords such as password, PIN, etc. INPUT fields are usually to hold user input and login keywords guarantee that we are actually facing a login form rather than other types of forms such as the common search form. We compiled 42 login keywords to allow flexibility in detecting various patterns such as "passcode", "customer number", etc.

Due to phishing and other inadvertent behaviors, a login form does not always satisfy all three properties above, and to cope with such variations, we designed the following algorithm to declare the existence of a login form.

1. We first handle the regular case in which form tags, input tags and login keywords all appear in the DOM. Login keywords are searched in the text nodes as well as the alt and title attributes of element nodes of the subtree rooted at the form node. Return **true** if all three are found.
2. We then handle the case where form and input tags are found, but login-related keywords exist outside the subtree rooted at the form node f. First, examine whether the form f is a search form by searching keyword "search" in the same scope as in step 1. If f is not a search form, traverse the DOM tree up for 2 levels starting from f to ancestor node n, and search login keywords under subtree rooted at n in the same scope as in step 1. Return **true** if a match is found.
3. This branch captures the phishing pattern in which forms and inputs are detected, but phishers put login keywords in images and refrain from using text to avoid being detected. Check the subtree rooted at f for text and images, and return **true** if no text is found and only images exist.
4. This branch handles the case where phishers only use input fields and leave out form tags on purpose. Search login keywords and image patterns in a similar fashion as above, but in the scope of the whole DOM tree r, and return proper results.

The heuristics here may flag a form as a login form when it actually is not. However, this slightly larger coverage on one hand helps prevent falsely filtering a phishing page prior to the content analysis stage, and on the other still removes a vast majority of pages with no login forms from consideration, thus reducing false positives and accelerating the detection process.

5.4 Identity-based Phish Detection

Two algorithms exploiting website brand identities are given in this section. The basic idea is to first locate entity names in DOM text nodes or attributes of element nodes that are most likely to represent the site brand name, then find domains for those names via searching, and compare the matching domains with the page domain to find identity inconsistency via a strategy defined in section 5.2. As long as one matching domain is found to be related to the page domain, we classify the web page as "good". If no matching domains are found, the classification defaults to "good". Both attempt to reduce false positives.

5.4.1 Retrieval-based Identity Recognition

Our phish detection algorithm in this section solves identity name recognition and name-to-domain translation together in one step via heuristics-aided search, using a page's title and copyright field since they usually contain the site brand name (either its own or the one it is impersonating).

Before delving into details, we give some notational conventions first. Let q denote a search query, w denote a word from q, W_s denote a set of stopwords[2], d denote a domain keyword, ac denote an acronym (defined below) from q and L is a specified minimum length[3].

In light of the first two properties introduced in the beginning of section 5.2, we find candidate brand domains by searching on major search engines important page fields like title and copyright field, hoping domains corresponding to the brand name in the title/copyright to be returned. We compare the domain of each search result URL with the terms in the search query to find a match[4]. Our algorithm defines four heuristics to evaluate a domain-query match

1. $\exists w \in q, \neg w \in W_s, |w| \geq L, w$ is a substring of d
2. $\exists w \in q, \neg w \in W_s, |d| \geq L, d$ is a substring of w
3. $\exists ac \in q, |ac| \geq L, ac$ is a substring of d
4. $\exists ac \in q, |d| \geq L, d$ is a substring of ac

An acronym is the concatenation of initial letters of a segment in title/copyright, handling the cases where a domain keyword is the combination of initial letters of the brand name[5]. Web page titles sometimes manifest a certain patterns like "subcategory delimiter category delimiter brand name"[6][7], and to extract acronyms, we define a four-tiered delimiter [8] and split the title iteratively to segments. The copyright field on a web page typically shows some patterns like "Copyright © 1995-2008 eBay Inc. All Rights Reserved." in Fig 5.1, and we defined 11 regular expressions targeting different variants to extract the brand name. Some page has more than one copyright field, and we prefer the one with word overlap with the page domain, with keywords like "Inc.", "Ltd.", or simply the last copyright field. Note that sometimes we extract more words than necessary from a copyright field, but still have the brand name in them.

[2]Words that occur very often yet bear little actual meaning such as "the", "of", etc.

[3]The minimum length is defined to be 3 characters here.

[4]In this context, the term "match" means a match between terms in the query and the domain in the query search result according to the four heuristics, not the match where search engines return results for a query.

[5]This usage is not unusual. An example is the brand name "nebraska university federal credit union" whose domain is "nufcu.org", with domain keyword "nufcu".

[6]Two such examples are "Music > Alternative - Mininova" and "Tony Stewart - NASCAR - Yahoo! Sports".

[7]More punctuations may exist in subcategory, category or brand name.

[8]First tier has "|","·",">","/"; second has "–"; third has "," and "."; and fourth consists of spaces.

To accurately map a brand name to its domain, we employ Google and Yahoo, two popular commercial search engines, and define two strategies in selecting domains when domain-query matches occur.

- **Strategy I:** *We evaluate domain-query matches among the top* 5 *search results*[9] *of Google and Yahoo. If both search engines have such matches and the domain of the No.1 match from each side coincides, we take it as a candidate domain of the brand corresponding to the query.* If only one search engine has matches, we take the No.1 domain as a candidate brand domain. *Joining the candidate brand domain set also are the first ranked results of both search engines if their domains are identical, regardless of domain-query matches.*
- **Strategy II:** Just take the two branches corresponding to the italicized part of strategy I.

The goal of using two strategies is to investigate whether we have to have both search engines return results with domain-query matches. According to the strategies, at most two candidate brand domains are returned (thus at most two queries), and for each of them, we conduct search using query of the form defined in section 5.2 with the site operator, and flag a "good" label if either search engine yields results for any query. Otherwise, a phish alarm is fired.

5.4.2 Named Entity Enhanced Identity Recognition

Web site brands often manifest themselves as textual names in places in the page other than title and copyright field, and we can apply NER to identify these names to facilitate phish detection. The focus of this component is mainly to reduce false positives especially in cases where target brand names are absent in title and copyright but are present in other DOM objects. This component first matches recognized entity names with the domain keyword to extract a single name most likely to be the web site brand name, uses the search-based mapping algorithm in section 5.4.1 to obtain its domain and then executes query to identify good sites.

Named entity recognition (NER) is the task of identifying various types of entity names in free text, such as persons, organizations, etc. NER is usually cast as a classification problem [71] and often explores linguistic features such as part-of-speech tags, affixes (n-grams), etc. In this work, we used the Stanford Named Entity Recognizer to identify website brand names. Stanford NER, a 3-class (organization, person, location) named entity recognizer for English, is a CRF-based information extraction system augmented by Gibbs sampling.

DOM-based End Punctuation Insertion

Formatting tricks via HTML tags ease web page reviewing, but sometimes omit sentence ending punctuations while not affecting reading, which are of significant importance to the NER task. An example is shown in Fig 5.1 in which words that should have been followed by an end punctuation are highlighted with thick blue lines. Extracting page content as is tends to produce noisy NER result, and we propose a novel method to attack that problem in this section.

The intuition of our punctuation insertion algorithm is that though various formatting tags are used, non-end punctuations are still necessary to keep the web page readable, while end punctuations are sometimes omitted. In Fig 5.1, underlined words miss "." afterwards, while non-end punctuations like "," are all punctuated well. Moreover, a sentence usually ends at the rightmost text node of a DOM subtree. Though occasionally such rightmost text node points

[9]This is a tunable parameter, and we remove noisy URLs like "www.phishtank.com" and "www.millersmiles.co.uk" from the returned list beforehand.

to the middle of a sentence, adding a period here does not have a big influence on the following NER step.

Our algorithm exploits the DOM tree and adds a period to the end of either the rightmost text node of a basic block, a text node preceding a BR node, or each text node of a link list structure, when end punctuation is missing. In this context, a basic block is defined to be a subtree composed entirely of anchor nodes and text nodes (except the subtree root), and a link list is a subtree with only anchor nodes or anchor nodes separated by text separator ("|" or "–") such as the 9 anchors on the bottom of Fig 5.1 starting with "About eBay". Note that both definitions only apply to the DOM tree after processing because otherwise there could be many formatting tags in a subtree like DIV. Link lists are important because there is often a link list on the bottom of a web page followed by a copyright field, where website brand name appears.

In our algorithm, we first prune the DOM tree by removing non-informative nodes including empty text nodes, SCRIPT nodes, NOSCRIPT nodes, SELECT nodes, STYLE nodes, nodes whose children are all removed, all but the first of contiguous sibling BR nodes and other non-text leaf nodes. We then add a period if there is none to the text node prior to a BR node. Next, we add a period to the end of the page title if necessary and collapse the DOM tree by removing non-text leaf nodes and non-anchor internal nodes that are the only child of their parents. Note that this collapsing step will remove the BR nodes that survive the pruning stage. Collapsing the DOM tree will significantly cut the tree size, and facilitate punctuation addition via basic blocks. In the end, we add a period to the proper positions of a basic block and link list.

The major part of our punctuation insertion method is described formally in Algorithm 2 and 3. The procedures of adding periods to link lists and basic blocks and collapsing the DOM tree are omitted. Note that correcting punctuations perfectly is a hard problem, and our approximate algorithm tends to add more punctuations, which is desirable since such redundancy reduces unwanted named entities.

Algorithm 2 AddPunctuationMain

Require: Raw DOM tree r
Ensure: Punctuation-added DOM tree
1: Remove non-informative nodes from r
2: Add "." to text nodes preceding BR nodes if necessary
3: Add "." to title if necessary
4: CollapseTree(r)
5: AddPunctuations(r)

Dual-source NE-based Identity Recognition

Our dual-source identity recognition algorithm proceeds by first identifying via NER a list of organization names from the visible content (set 1) and invisible DOM objects including the alt and title attributes of element nodes and the content attribute of meta description tags (set 2), and then applying heuristics to find a single name (or none) that is most likely to be the brand identity from the two sets. Each candidate name in the two sets is split into terms, and its count of matches with the page domain keyword is recorded. A match is found if a term is not a stopword, satisfies a minimum length, and either is a substring of the domain keyword or the other way around. If the acronym of an organization name is identical to the domain keyword, it is also counted as a match. Our heuristics prefer entity names 1) with higher match count, 2)

Algorithm 3 AddPunctuations

Require: a subtree root r of the processed DOM tree

```
1: if r is text leaf then
2:     if (r is the last child) && (r not end with punctuation) then
3:         text_r ← text_r + "."
4: else
5:     if r is link list then
6:         AddPunctuations2LinkList(r)
7:     else if r is basic block then
8:         AddPunctuations2BasicBlock(r)
9:     else
10:        for all child n of r do
11:            AddPunctuations(n)
```

recognized from the page content, 3) with shorter length. This procedure is shown in Algorithm 4. After getting a final name from the two sources, we extract its domain and classify the web page using the query algorithm introduced in section 5.4.1.

Algorithm 4 FindORGIdentityName

Require: domain keyword d, web page p, DOM tree r

```
1: N_1 ← NERFromContent(p)
2: N_2 ← NERFromInvisibleDOMObjects(r)
3: ∀n ∈ N_1, N_2, break n into terms
4: ∀n ∈ N_1, C_1 ← compute n's terms match count with d
5: ∀n ∈ N_2, C_2 ← compute n's terms match count with d
6: if non-zero count exists in C_1, C_2 then
7:     if max(C_1) ≠ max(C_2) then
8:         return the name with highest count, breaking count-ties by choosing the shortest
           name
9:     else
10:        if names with highest match count with d from N_1, N_2 intersect then
11:            return a name in the intersection from N_1 preferring shorter ones
12:        else
13:            return a name with highest count from N_1, breaking count-ties by choosing
               the shortest one
14: else
15:    return the name with acronym match with d from N_1, or N_2 if N_1 yields none, or none
```

5.5 Keywords Retrieval for Phish Detection

Motivated by the property that phishing web pages are much less likely to be crawled and indexed by major search engines due to their short-lived nature and few in-coming links, we present in this section a method utilizing search engines to detect phish.

In light of the fact that all search engines employ scoring functions to rank matching documents, we should intuitively feed search engines those keywords that are more likely to push

intended web pages to top positions in the result list. Toward that end, we adopted the classic TF-IDF metric in ranking candidate query words

$$TF\text{-}IDF(w) = TF(w) \cdot IDF(w)$$

where term frequency TF(w) denotes the number of occurrences of w in the page, and inverse document frequency IDF(w) measures the general importance of w in the whole collection. We used Google as the collection corpus, and estimated the document frequency of a term w by the number of search results on the upper right corner of the result page when searching w in Google. To increase TP and reduce FP, we also put the page domain keyword in the query.

In this algorithm, we also use two search engines, Google and Yahoo, and report a page as phish if neither has the page domain in the top 30 results. The full-blown model integrating the identity-based and retrieval-based detection methods is described in Algorithm 5.

Algorithm 5 DetectPhish

Require: Web page p, page domain keyword d, page domain n, white domain list D_w
Ensure: **true** – phish; **false** – good

```
Parse p
r ← DOM
FilterByWhiteDomain(n, D_w)
DetectLoginForm(r)
if (n in D_w) || (no login form found) then
    return false
else
    t ← GetTitle(r)
    cp ← GetCopyright(r)
    terms ← GetTopTFIDFTerms(r)
    AddPunctuationMain(r)
    id ← FindORGIdentityName(d, p, r)
    pred ← DetectByIdentity(d, t, cp, id)
    pred ← pred || DetectByIFIDF(d, terms)
    return pred
```

5.6 Experiment Setup

White domains are good domains verified by authorities, and serve as an effective way in reducing false positives and speeding up detection. We collected such domains from three sources. First, Google safe browsing provided a whitelist of [3] 2770 domains by mid September of 2008, and we obtained a total of 2682 unique domains after removing duplicates. Second, millersmiles [2] maintains an archive of common spam targets like Paypal, and we extracted 424 unique domains out of a total of 732 entries after mapping organization names to domains and removing duplicates. Moreover, we also utilized an online white domain service [6], which performs DNS lookup to determine if a query domain is on the whitelist. Like any other whitelists, this online database's coverage is rather limited, and out of all the 3543 good URLs we have, only 480 appear on it.

Our web page collection consists of phishing cases from one source, and good web pages from six sources. To eliminate the influence of language heterogeneity on our text-oriented methods, we only downloaded English web pages.

For phishing instances, we used the XML feed of Phishtank [65]. We started downloading the feed in early May of 2008 and manually examined the downloaded web pages to remove legitimate cases, 404 errors, and other types of noisy pages, collecting a total of 7,906 phishing web pages during a five-month period.

Good pages came from the following six sources. Alexa.com maintains a top 100 website list for a variety of languages, and we crawled the homepages of the top 100 English sites to a limited depth, collecting 1039 good web pages in this category. To introduce web pages with login forms into our data set, we downloaded 961 login pages, utilizing Google's inurl operator and searching for pages with keywords such as "signin" and "login" in the URL. Although not every page in this category contains an actual login form, there is guarantee that all of these URLs point to legitimate websites. 3Sharp [15] released a public report on anti-phishing toolbar evaluation in 2006, and we downloaded 101 good English pages out of the 500 provided in the report that still existed at the time of downloading. Moreover, we went to Yahoo directory's bank category [4], crawling the bank homepages for a varying number of steps within the same domains and collecting 988 bank pages. Likewise, we conducted crawling on other categories [7, 8, 9, 10, 11, 12] of Yahoo directory including US bank, credit union, online escrow services, travel agencies, real estates and financial services, and gathered 371 web pages. We name this data set "*Yahoo misc pages*" for reference convenience. To test the robustness of our methods, we manually chose 83 login pages of popular phishing target sites, such as eBay, etc. We call this data set "*prominent pages*". Note that none of the other five categories has overlap with URLs in this set, rendering this category independent of others. In our evaluation, we applied our algorithms to the whole corpus and reported the result statistics.

5.7 Experimental Result

5.7.1 Detecting Login Forms

As shown in Table 5.1, we successfully detected 99.82% phishing pages with login forms, and filtered a significant percent of good pages from other categories. For the remaining 0.18% (14 in absolute number) phishing pages, they either do not have a login form (very rare in our phish corpus), use login keywords not in our list such as "serial key", or organize the form/input tags in a way our method misses. Note that a lot of web pages in the login category do not have login forms. Pages with keywords like "login" in URLs do not necessarily have login forms.

Table 5.1: Statistics of login form detection. 99.82% phishing pages with login forms were successfully detected.

Corpus	Phishtank	Alexa	Login pages	3Sharp	Banks	Yahoo misc	Prominent
#total pages	7906	1039	961	101	988	371	83
#pages detected with login forms	7892	318	639	35	234	98	76

5.7.2 Phish Detection by Keywords-Retrieval

In this section, we report the performance of the keywords-retrieval component, varying the number of top keywords.

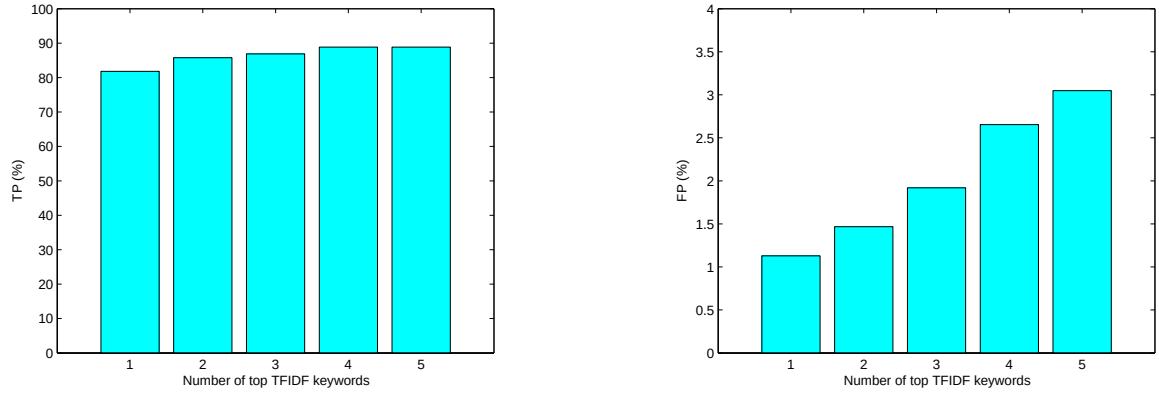

Figure 5.3: Performance of the keywords-retrieval detection method. Both TP (left) and FP (right) increase monotonically as the number of TF-IDF-ranked keywords grows from 1 to 5. TP on the left has a minimum of 81.80% and tops at 88.86%. FP on the right ranges from 1.13% to 3.05%. The priority of FP suggests using fewer TF-IDF-ranked keywords for this detection algorithm.

Examining both graphs in Fig.5.3 reveals that throwing more words with top TF-IDF scores in the query may bring up irrelevant result pages that on one hand increase TP while on the other hurt FP. This is an interesting observation contradictive to the thought that more relevant query words will help find the intended web pages more effectively. The secret sauce of Google and Yahoo has not been published, and considering the fact that false positive is usually weighed more heavily in industry, we took only the No.1 TF-IDF word with the domain keyword in building queries in other experiments of this work.

5.7.3 Identity-based Detection under Strategy I

The effectiveness of each individual module and their combination is interesting to explore, and in this section, we experimented with five approaches, i.e., detection by 1) title, 2) copyright, 3) TF-IDF, 4) title + copyright + NE, and 5) a full-blown method with a combination of the four. Among them, four approaches except the pure TF-IDF one used strategy I (section 5.1) in matching domains of query search result URLs with the query. The TF-IDF-based method does not perform domain-query match. Notice that our NE-based detection algorithm (section 5.4.2) was only used as an auxiliary module to the identity-based component to reduce false positives, and thus was not tested individually.

A quick glance at the result in Table 5.2 reveals that all individual detection algorithms have low FP ($< 1.5\%$).

Also shown in Table 5.2, the identity-retrieval method using title and copyright captured 78.03% and 54.33% phish respectively. About 21.97% and 45.67% phish were missed mainly in cases where either no identity name was found (predict "good" by default) or the page domain was on the whitelist. The latter was caused by phisher hacking into legal domains and uploading

Table 5.2: Performance of all methods under strategy I. The use of a whitelist degrades the TP of detection-by-title method by 1.15%, due to the activity of planting phishing pages into legal domains. Login form filtering significantly reduces the FP of the detection-by-title method from 3.81% to 1.38%. The combination of Title+Copyright+NE+TF-IDF boosts the TP to 93.31% with a FP of 2.26%. The keywords-retrieval detection method uses the word with No.1 TF-IDF score plus domain keyword.

	Title	Copyright	TF-IDF	Title+Copyright+NE	Title+Copyright+NE+TF-IDF
TP(%)	78.03	54.33	81.80	82.90	93.31
FP(%)	1.38	1.41	1.13	1.38	2.26
	Title only with no domain whitelist			Title only with no login form detection	
TP(%)	79.18			78.09	
FP(%)	1.92			3.81	

phishing sites. Moreover, the 1.38% and 1.41% FP were mostly due to the absence of brand names in the title and copyright field, leading to false domain-query matches and zero result when executing query of the form *site:declared_brand_domain "page_domain"*.

Another cause of false positives was that some extracted domain did not truly represent the intended brand, even if the true brand name appeared in the title/copyright field and a domain-query match was found. An example page of this with URL `http://www.fbandt.com/atmSearch.php` has title "firstbank & trust", while matching the title search results with the title returns domain "firstbank.com", pointing to a different entity.

To examine the effectiveness of domain whitelist and login form filtering, we tested the identity retrieval based detection method using title only in two experiments, with no whitelist and login form detection respectively. Result in Table 5.2 suggests that though these two filtering steps have no dramatic impact on the TP, applying whitelist does improve the FP slightly (from 1.92% to 1.38%) and skipping login form filtering hurts FP significantly, which plummeted from 1.38% to 3.81%.

Another observation is that title seems to be more effective than the copyright field in directly delivering brand-related information. One reason is that copyright field sometimes gives the name of a parent organization offering a variety of services or products, and the page domain points to one of them, leading to possible false positive when the parent organization domain does not refer on their site to the service or product domain name.

Keywords-retrieval detection outperformed identity-based detection with title and copyright in both metrics, demonstrating the power of commercial search engines in their crawling breadth and document ranking capability.

Enhancing title/copyright with identity NER pushed the TP up to 82.90%. Interestingly, this enhancement kept FP the same as detection by title alone (lower than the 1.41% FP of copyright-based detection), suggesting that the NE-based detection algorithm correctly removed a certain false positives caused by copyright-based detection.

The synthesis of all four algorithms boosts the TP to 93.31%, with a low FP of 2.26%, which suggests that the phish captured by the four methods do not entirely overlap. Though the types of false positives each individual module suffers from are hard to perfectly specified by pure analysis, the stacking strategy will lead to a combined model with low FP as long as each component has reasonably low FP.

5.7.4 Identity-based Detection across Strategies

Table 5.3: Performance of all methods across strategies. An integrated Title + Copyright + NE + TF-IDF boosts the TP significantly under both strategies. The keywords-retrieval detection method uses the term with No.1 TF-IDF score plus domain keyword.

	Title	Copyright	TF-IDF	Title+Copyright+NE	Title+Copyright+NE+TF-IDF
TP(%), strategy I	78.03	54.33	81.80	82.90	93.31
TP(%), strategy II	57.24	54.15	81.80	68.23	90.06
	Title	Copyright	TF-IDF	Title+Copyright+NE	Title+Copyright+NE+TF-IDF
FP(%), strategy I	1.38	1.41	1.13	1.38	2.26
FP(%), strategy II	0.40	0.90	1.13	0.93	1.95

Besides the individual detection modules, the efficacy of the strategies in selecting candidate brand domains upon the occurrence of domain-query match is also worth exploring, and we report the evaluation for that purpose in this section. Table 5.3 shows the experiment result of all detection methods under two strategies. Note that the keywords-retrieval detection method does not involve domain selection for the website brand name and thus shows the same performance under both strategies.

Across strategies, the TPs of title-based detection method were tremendously different, with 78.03% under strategy I and 57.24% under strategy II, and the corresponding FPs also dropped from 1.38% to 0.40%. Considering the different sizes of the phish (7906) and legitimate (3543) corpus in our experiment, these statistics suggest that even if only a single search engine returns top domains with term match with the query, it is still beneficial to take those domains as corresponding to the true brand name since they were able to catch a significant number of phish (over 20% or 1580 pages) at the cost of limited degradation on FP (around 1% or 35 pages). The performance of the full identity-based detection method (Title+Copyright+NE) also confirms this by lifting the TP from 68.23% to 82.90%, with 0.45% decline in FP, suggesting the effectiveness of search engines in discovering brand domains. Another insight is that Google and Yahoo may use different ranking and crawling algorithms, and it is desirable to adopt both for phish detection.

The TPs of the detection-by-copyright approach almost remained identical across two strategies (54.33% vs 54.15%), delivering the message that copyright field is usually more stable for website identity extraction, which makes perfect sense since the purpose of copyright field is to show website brand names while the tile could express any information and thus is much noisier.

Similar to the experiment in the previous section, a stacked hybrid model of four algorithms achieved the highest TP at 90.06% under strategy II, significantly better than each of the individual method, with a low FP of 1.95%.

5.7.5 Evaluation with Other TF-IDF Approaches

We evaluated our proposed approach against CANTINA on the same corpus and report the result in this section.

Table 5.4 shows that the TPs of our algorithms were comparable to CANTINA, while the FPs were much better (2.26%/ 1.95% vs 5.98%). Four hypothesis tests were conducted comparing the TP/FP of our methods under each strategy with CANTINA, all with the null hypothesis hypothesizing equal performance while the alternative hypothesis favoring our method. Table 5.4 reveals that all but one case are statistically significant (marked by $*$) with strong evidence in favor of our detection algorithms.

Table 5.4: Performance of the full-blown model (Title+Copyright+NE+TF-IDF) under two strategies vs CANTINA. Our algorithms perform comparably with CANTINA in terms of TP, while far outperform it on FP (2.26%/1.95% vs 5.98%). Hypothesis tests compare our methods against CANTINA for each metric under each strategy, with statistically significant results marked by $*$.

	Strategy I	Strategy II	CANTINA
TP(%)	93.31	90.06	91.40
FP(%)	2.26	1.95	5.98
p-value (TP%)	$< 1.0e$-5 $(*)$	0.998	
p-value (FP%)	$\ll 1.0e$-5 $(*)$	$\ll 1.0e$-5 $(*)$	

Although phishing signatures constantly evolve, the conclusion from [92] still carries and our experiment results suggest that our proposed algorithms are at least as good as, if not better than, the state-of-the-art anti-phishing toolbars.

Chapter 6

A Feature-rich Machine Learning Framework for Phish Detection

As explained in the previous chapters, URL blacklists are frail in terms of new phish, and feature-based methods need more effective features. To alleviate those problems, we proposed a layered anti-phishing solution in this chapter that aims at 1) exploiting the expressiveness of a rich set of features with machine learning to achieve a high TP on novel phish, and 2) limiting the FP to a low level via filtering algorithms.

Specifically, we proposed CANTINA+ [1], the most comprehensive feature-based approach in the literature so far, including eight novel features, which exploits the HTML Document Object Model (DOM), search engines and third party services with machine learning techniques to detect phish. Moreover, we designed two filters to help reduce FP and achieve runtime speedup. The first is a near-duplicate phish detector that uses hashing to catch highly similar phish. The second is a login form filter, which directly classifies web pages with no identified login form as legitimate.

6.1 Introduction

An ideal anti-phishing solution needs to have reasonable TP against new attacks with very low FP while involving minimum manual labor. The key to achieve a high TP is to design new features that are characteristic of phishing patterns, and the core ingredient leading to a very low FP is filtering via heuristics. With those in mind, we set as our goal in the technique in this chapter contributing to the literature by addressing the weaknesses of both blacklists and feature-based methods in a unified framework. Specifically, we propose novel features to improve the TP and design filtering algorithms absent in the literature to reduce FP and human effort.

We name such a layered system CANTINA+, which exploits the generalization power of machine learning techniques and the expressiveness of a rich set of web page features to detect phish variants. Our pipeline consists of three major modules. The first leverages the high similarity among phishing web pages due to the prevalent use of phishing toolkits, and examines a web page's similarity to known phishing attacks via hashing to filter highly similar phish. The second exploits the property that phishing attacks usually utilize login forms to request sensitive information, and employs heuristics to filter web pages with no login forms prior to the classification phase. The third module, the core of our framework, utilizes 15 highly expressive features with

[1] Parts of this chapter were previously published in the ACM Transactions on Information and System Security (TISSEC) [86]

machine learning algorithms to classify web pages. This module adopts the idea of extracting website ownership from our previous work [85] in building two features, and significantly extends our past work with CANTINA [92] by eight novel discriminative features.

The work in this chapter makes the following three research contributions. First, we propose eight novel features capturing the intrinsic characteristics of phishing attacks using a wide spectrum of resources, including the HTML DOM, search engines, and third party services, obtaining superior classification performance. Second, our approach ameliorates the typical weakness of high FP of feature-based approaches by using a layered structure with login form filtering. Note that login form detection is quite nontrivial due to the flexibility of the HTML DOM, which will be explained later in this chapter. Third, the diversity of web pages in our corpus and the comprehensiveness of our evaluation methods all exceed the techniques in the literature.

In our experiment, we evaluate our approach on a rich corpus with web pages from six categories, and conducted a thorough experiment with randomized and time-based methodologies to inspect the generality of our method as well as its real-world performance. In the randomized evaluation, CANTINA+ achieved an over 92% TP on unique testing phish, an over 99% TP on near-duplicate testing phish, and an about 0.4% FP with 10% phish in the training set with login form filtering. In the time-based evaluation, our method achieved an over 92% TP on unique testing phish, an over 99% TP on near-duplicate testing phish, and an about 1.4% FP with 20% phish in the training data with a two-week sliding window. Those phishing attacks whose timestamps fall in the sliding window will be used to train machine learning models, and by using such a length-adjustable moving window, we are able to incorporate the latest phishing variants into our training data and also achieve runtime speedup. There has not been any experimental evaluation as to what is acceptable for end users in the literature, and we have among the lowest FP rate of any feature-based detection techniques out there. It is possible to get even lower FP using extremely conservative features, though this would significantly impact true positives.

6.2 System Architecture

Figure 6.1 shows the overall flow of CANTINA+. The feature extractor, shared by the training and testing phases, is the core of our hybrid framework, in which the values of the 15 features are extracted. Specifically, the goal of the training phase is to obtain the feature values for each instance of the training corpus, which is then used by the machine learning engine to build classifiers. The goal of the testing phase is to label real web pages as phish or not.

In the testing phase, we first apply two filters to web pages to reduce false positives and speed up runtime performance. The first filter is a hash-based filter that compares a web page against known phish. The second filter checks a given web page for a login field. We will describe the details of these two filters in the next two sections. If the web page is not detected as a near-duplicate of the existing phish and a login form is found in the HTML, we move on to extract the 15 features from the web page using the URL, HTML DOM and other resources, and apply a pre-trained model to classify its identity. In real-world scenarios, we can use a sliding window to include the most recent phishing attacks in the training data.

6.3 Hash-based Near-duplicate Page Removal

The growing use of toolkits [26] to create phish produces a massive volume of phishing web pages that are very similar or even identical to each other in terms of HTML. This observation led us

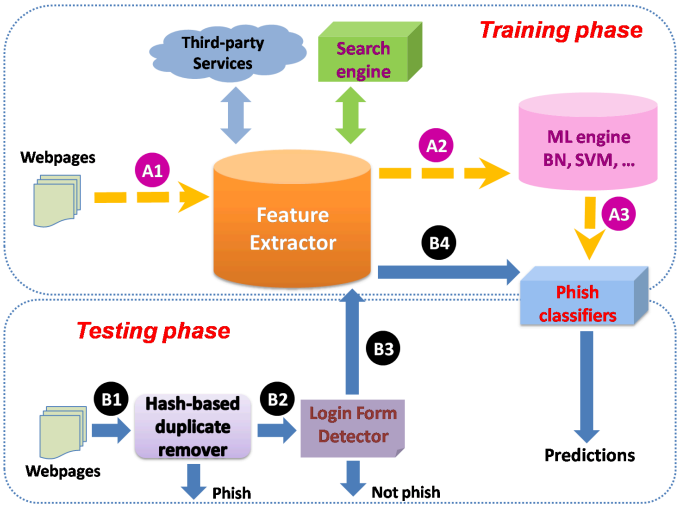

Figure 6.1: CANTINA+ system architecture. In the training stage, A1) 15 feature values are extracted from each instance in the training corpus; A2) the feature values are organized in proper format and forwarded to the machine learning engine; A3) classifiers are built. In the testing stage, B1) the hash-based filter examines whether or not the incoming page is a near-duplicate of known phish based on comparing SHA1 hashes; B2) if no hash match is found, the login form detector is called, which directly classifies the web page as legitimate if no login form is identified; B3) the web page is sent to the feature extractor when a login form is detected; B4) the pre-trained learning models run on the features and predict a class label for the web page.

to adopt page duplicate detection algorithms to identify pages that are extremely likely to be phish, by comparing a given page against known phish. In our previous work [88], we proposed an adaptive probabilistic anti-phishing algorithm based on URL blacklists exploiting the high similarity among phishing attacks. In this work, we simply design a more rigorous hash-based filter to quickly recognize identical phish while mainly rely on the machine learning engine to detect other variants.

To detect duplicate pages, we used the SHA1 hash algorithm, a popular method for checking if two pieces of digital content are the same [63]. SHA1 is a fast and secure procedure that produces 160-bit hash values and is applicable to content of any length with a low likelihood of collisions. To use SHA1, we first remove all spaces in the HTML. We also remove all default values in HTML input fields and replace them with empty strings. Our rationale here is that we have seen some phishing sites that insert random email addresses into such fields. We then compute a SHA1 hash on the processed HTML, which is then compared against a pool of hash values of known phishing web pages. Currently, we use PhishTank's verified blacklist as our known list of phishing sites.

We acknowledge that this hashing-based filter is easy to beat. However, it is highly effective against existing phish today, is fast in terms of runtime performance, and cheap to implement. Also, we only use it as a filtering step to remove near-duplicate phish, and mainly rely on machine learning approaches with our feature set for phish detection.

6.4 Login Form Detection

Almost all phishing attacks try to trick people into sharing their information through a fake login form. In this section, we present our algorithm using the HTML DOM to filter pages with no login forms prior to the classification step. On the surface, this sounds simple, yet finding a login form in practice is actually by no means trivial. Typically, a login form is characterized by three properties, i.e., 1) FORM tags, 2) INPUT tags, and 3) login keywords such as password, PIN, etc. INPUT fields are usually used to hold user input. Login keywords guarantee that we are actually facing a login form rather than other types of forms such as the common search form. We compiled 42 login keywords to allow flexibility in detecting various patterns such as "passcode", "customer number", etc.

Due to phishing and other unconventional design patterns, a login form does not always satisfy all three properties above, and to cope with such variations, we designed the following algorithm to declare the existence of a login form.

1. We first handle the regular case in which form tags, input tags and login keywords all appear in the DOM. Login keywords are searched in the text nodes as well as the alt and title attributes of element nodes of the subtree rooted at the form node. Return **true** if all three are found.
2. We then handle the case where form and input tags are found, but login keywords exist outside the subtree rooted at the form node f. First, we examine whether the form f is a search form by searching for keyword "search" in the same scope as in step 1. If f is not a search form, we traverse the DOM tree up for K levels[2] to ancestor node n, and search login keywords under the subtree rooted at n in the same scope as in step 1. Return **true** if a match is found.
3. We then capture the phishing pattern in which forms and inputs are detected, but phishers put login keywords in images and refrain from using text to avoid being detected. Check the subtree rooted at f for text and images, and return **true** if no text is found and only images exist.
4. Finally, we handle the case where phishers only use input fields and leave out form tags on purpose. Search login keywords and image patterns in a similar fashion, but in the scope of the whole DOM tree r, and return proper results.

This algorithm covers most of the login form variants, with a 98.06% TP on our phishing corpus as shown in section 8.2. The features in this algorithm may flag a form as a login form when it actually is not. However, this slightly larger coverage on the one hand helps prevent falsely filtering a phishing page prior to the content analysis stage, and on the other still removes the vast majority of pages with no login forms from consideration, thus reducing false positives and significantly accelerating the detection process.

6.5 A Feature-rich Machine Learning Framework for Anti-phishing

When the hash-based filter finds no matches with existing phish and the login form filter detects a login form in the HTML, our approach relies on the machine learning engine to capture phishing variants. The set of high-level features is the major contribution of our technique in this chapter, and in this section we will elaborate on the design and rationale of our feature set, as well as

[2]We took $K = 2$ in our current implementation.

the machine learning algorithms. In particular, features 2, 3, 4, 12 are taken from CANTINA, feature 5 and 13 are taken from [36], feature 11 is a variant of one feature in CANTINA, and features 1, 6, 7, 8, 9, 10, 14, 15 are the novel ones we proposed.

It is necessary to point out that false positives or false negatives might be caused by each feature. However, the combination of all the features will make up for the inadequacy of individual features, and will yield better performance.

6.5.1 High-level Web page Features

We have organized our features into three categories. The first (features 1 through 6) deals with the URL of the web page. The second (features 7 through 10) inspects the HTML content of the web page. The third (features 11 through 15) involves searching the web for information about that web page. Specifically, feature 1, 6, 7, 8, 9, 10, 14, 15 are novel ones proposed by us in CANTINA+.

URL-based Features

1. ***Embedded domain***. This feature examines the presence of dot separated domain/hostname patterns such as "www.ebay.com" in the path part of the web page URL. Phishers sometimes add their target's domain/hostname in the path segment to trick users into trusting their phishing sites. We avoid hard-coded domains and instead search in the path segment of the URL with a regular expression that seeks dot-separated string segments[3].
2. ***IP address***. This feature checks if a page's domain name is an IP address.
3. ***Number of dots in URL***. This feature counts the number of dots in the URL. Phishing pages tend to use more dots in their URLs than the legitimate sites.
4. ***Suspicious URL***. This feature checks if a page's URL contains an "at" (@) or the domain name has a dash (-). An @ symbol in a URL causes the string to the left to be disregarded, with the string on the right treated as the actual URL for retrieving the page. Dashes are also not often used by legitimate sites.
5. ***Number of sensitive words in URL***. In [36], Garera et al summarized a set of eight sensitive words[4] that frequently appear in phishing URLs, and we create this feature counting the number of the eight sensitive words that are found in a page URL. This is a numeric feature with a range of 0 to 8.
6. ***Out-of-position top level domain (TLD)***. This feature checks if a TLD appears in an unusual position in the URL. An example is `http://cgi.ebay.com.ebaymotors.732issapidll.private99dll.qqmotorsqq.ebmdata.com`, in which we see the TLD "com" in a position usually not for TLDs in the hostname.

HTML-based Features

7. ***Bad forms***. Phishing attacks are usually accomplished through HTML forms. This feature checks if a page contains potentially harmful HTML forms. To satisfy our definition of harmful, a web page is required to have all of the following: 1) an HTML form, 2) an <input> tag in the form, 3) keywords related to sensitive information like "password" and

[3]Three constraints must be met for a dot-separated string to be eligible for an embedded domain. First, at least three segments must exist. Second, each segment must have two or more characters. Third, each segment is composed of letters, numbers and underscores only.

[4]The sensitive words include "secure", "account", "webscr", "login", "ebayisapi", "signin", "banking", "confirm".

"credit card number" or no text at all but images only within the scope of the HTML form, 4) a non-https scheme in the URL in the action field or in the web page URL when the action field is empty.

Login form recognition is realized by the SAX parser, a sequential access parser API to read data from XML documents, rather than the DOM-based process as in the login form filter in section 5. Specifically, we defined 39 login-related keywords to narrow down our focus to forms that truly request user private information.

8. ***Bad action fields***. From an engineering point of view, placing the authentication scripts of the whole website in one location facilitates the development and maintenance of the code, and legitimate websites tend to adopt this practice. Accordingly, the authentication methods in the script on legitimate websites are usually called via absolute URLs in the action field of the HTML form. However, phishing sites are usually ephemeral and the design principle is often to make everything as simple as possible, as a result of which the authentication code is usually placed in the current directory and the action field of the HTML form is typically a file name without directory hierarchy. As such, this feature is set to 1 if the action field is empty or a simple file name, or points to a domain different from the web page domain.
9. ***Non-matching URLs***. This feature examines all the links in the HTML, and checks if the most frequent domain coincides with the page domain. The rationale behind this feature is that links on phishing sites are usually meaningless and thus noisy values such as "#", "index.html", URLs of the target legitimate sites, etc., are often seen especially when the attacks are automatically created by toolkits, leading to inconsistency between the page domain and the most frequent domain in the links. Sometimes, the links on a phishing page point back to various parts of the phishing site, however, phishers do not very often use a different absolute URL for each such link but rather stick to similar URLs. To catch that pattern, we count the percentage of highly-similar links[5] in the HTML, and set the value of this feature to 1 if any single pattern occurs more often than a threshold. We also count the percentage of empty or ill-formed links in the HTML, and apply thresholding to set corresponding feature values. We set those thresholds manually after examining a number of HTML files.
10. ***Out-of-position brand name***. The vast majority of companies put their brand name into their domain name. Typically, the brand name appears in the domain string as the second-level or third-level domain. Phishing sites, however, are always hosted on compromised or newly registered domains. To make these sites look trustworthy, attackers sometimes include brand names or domain names of the victim sites in their phishing URLs, causing an out-of-position brand name. The example in the 6th feature above still applies here, in which ebay, the target brand, appears in an unusual position in the hostname.

    However, since we have no a priori knowledge about the brand name of a web page, we follow the analysis in the 9th feature above and use the most frequent domain keyword in the HTML links as the website brand name. With this estimated brand name, we remove the page domain keyword as well as the string to its right from the URL, and search in the remaining portion for the brand name. If a match is found, the page under investigation is suspicious and the feature value is set to 1.

[5]Highly-similar links are defined to be those that are either identical or differ only in the fragment part of the URL.

Web-based Features

11. ***Age of domain***. This feature checks the age of the web page domain name via WHOIS lookups. Many phishing sites are hosted on recently registered domains, and as such have a relatively young age.

12. ***Page in top search results***. This feature was originally used in CANTINA [92]. Specifically, we extract the top K words from the page content ranked by the term frequency and inverse document frequency (TF-IDF) metric, and search those top terms plus the web page domain keyword[6] in Google. The web page is deemed legitimate if the page domain matches the domain name of any of the top N search results; otherwise, it is regarded as being phishy. The intuition behind this feature is that search engines are more likely to index legitimate websites, while phishing sites have much less chance of being crawled. According to the experimental findings in [85, 92], we took $K = 5, N = 30$ in this work.

13. ***PageRank***. PageRank is a link analysis algorithm first used by Google, in which each document on the web is assigned a numerical weight from 0 to 10, with 0 indicating least popular and 10 meaning most popular. An intuitive rationale behind this feature is that phishing web pages usually have very low PageRank scores due to their ephemeral nature and few incoming links pointing to them, while legitimate cases tend to have higher PageRank values.

14. ***Page in top results when searching copyright company name and domain***. This and the following feature complement the 12th feature by directly seeking the web page on the web without analyzing the terms in the page content. Generally, the TF-IDF feature above may not work well for two cases. First, some terms with high TF-IDF scores may not be relevant in searching the intended web page, and as a result, the query may not return the expected web page domain in top N entries. Second, due to company affiliations, two closely related domains are sometimes literally different such as "blogger.com" and "blogspot.com", which renders straightforward string matching inadequate.

    This feature uses as query phrase the page domain plus the copyright company name that is usually found on the bottom of a web page showing a website's brand name, and treats a web page as suspicious if its domain is absent from the top N search results ($N = 10$) and legitimate otherwise. This brand recognition idea was taken from our previous work [85]. In this technique, we only employed the copyright field to extract brand names instead of searching the page title and using the whole page content via named entity recognition.

    There are many advantages to this feature. First, search engines are more likely to have entries in their index for legitimate sites, and searching the page domain and the website brand name directly has a higher chance of returning the intended page in top positions, thus remedying the first problem of the 12th feature. Second, this feature alleviates the second weakness of the 12th feature as discussed above in that related domains tend to be all returned when searching the copyright brand name without other irrelevant query terms thanks to the broad coverage of modern search engines. Third, copyright fields may not show up in every page, and once they are missing, we simply query the page domain in search engines. Again, the argument in the first benefit of this feature explained above applies here, and we eschew false positives even if we misclassify a phish under this scenario.

[6]The domain keyword is the segment in the domain representing the brand name, which is usually the non-country code second-level domain such as "Paypal" for "paypal.com" or the third-level domain such as "ebay" in http://www.ebay.com.au/.

15. ***Page in top results when searching copyright company name and hostname***. This feature is identical to the 14th feature except that we use the hostname instead of the domain name in the query, which is useful especially when the domain name is too short and introduces noisy results in top result entries.

6.5.2 Machine Learning Algorithms

We compare six learning algorithms in training the phish detector, including Support Vector Machines (SVM) [21], Logistic Regression (LR), Bayesian Network (BN) (a probabilistic graphical model that makes inferences via a directed acyclic graph), J48 Decision Tree, Random Forest (RF) and Adaboost, with the primary goal of evaluating the effectiveness of our feature set. All the ML algorithm implementations were taken from the Weka package [83]. We found through extensive experiment that BN performed among the best algorithms consistently. This is mainly due to BN's nonlinear and probabilistic nature, which is important for features with interactions and to bias the algorithm towards one class or another. Therefore, we only report the performance of BN here.

6.6 Experiment Setup

6.6.1 Evaluation Metrics

In addition to the standard TP and FP measures, we also used the F1 measure, which integrates both TP and FP with equal weights into one summary statistic. In tuning the machine learning models, we adopted the concept of Receiver Operating Characteristics (ROC) curves [32] and employed the area under the ROC curve (AUC) [24] metric, which, as a standard approach to evaluate binary classification performance, portrays the trade-off between TP and FP. Statistically, the AUC equals the probability that given a randomly generated positive instance and negative instance, a classifier will rank the positive one higher than the negative one, and thus is a good summary statistic for model comparison. Other standard measures such as precision and recall can be easily inferred from TP and FP.

6.6.2 Web Page Corpus

Our web page collection consists of phishing cases from PhishTank, and legitimate web pages from five sources. To eliminate the impact of language heterogeneity on our content-based method, we only included English web pages in our corpus. Our legitimate collection mainly focuses on popular sites, commonly spammed sites, common phishing target sites, etc. Although our corpus is not representative of what users would experience in their every day browsing, by evaluating CANTINA+ on these hard cases, we actually provide pessimistic performance statistics in terms of FP, which is more beneficial for an objective evaluation of our method and its real-life application that follows.

To fully study our approach over a larger corpus, we downloaded the phishing web pages when they were still live and conducted our experiment in an offline mode.

For phishing pages, we used the phish feed of Phishtank [65]. The phish corpus in our experiment was collected over two periods. Phish set 1 was collected starting in early May of 2008, and 6,943 phishing web pages were downloaded during a five-month period. Phish set 2 was initiated in late February of 2009 and a total of 1,175 phish were garnered from February 27,

2009 to April 2, 2009. The purpose of two-separate web crawls is to roughly examine whether or not phishers tend to reuse phishing sites built a while ago.

To thoroughly test the FP, we collected the same five sets of legitimate web pages in two separate crawlings, the details of which are given in Table 6.1, with legitimate corpus 1 for randomized evaluation and legitimate corpus 2 for time-based evaluation. The missing pages in legitimate corpus 2 compared with the legitimate corpus 1 were due to broken links. Fetterly et al discovered through large-scale web crawling that web page content was fairly stable over time [35], and based on that finding, we did not download legitimate corpus 2 at each time point but rather downloaded only once the whole set at a time later than all the phishing timestamps in phish set 2.

Table 6.1: Legitimate collections from 5 sources. The size column marked by "1" gives the corpus sizes for randomized evaluation, while that marked by "2" gives the corpus sizes for time-based evaluation. We use legitimate corpus 1 and legitimate corpus 2 to refer to the two collections. Legitimate corpus 2 was downloaded on April 2, 2009, and was a subset of legitimate corpus 1.

Source	Size (1)	Size (2)	Crawling Method
Top 100 English sites from Alexa.com	1,023	958	Crawling homepages to a limited depth
3Sharp [15]	101	87	Downloading web pages that still existed at the time of downloading
Generic banks on Yahoo directory	985	878	Crawling the homepages for a varying number of steps within the same domains
Other categories of Yahoo directory	371	330	Same as the generic bank category
The most common phishing targets [74]	81	69	Saving login pages of those sites

6.6.3 Evaluation Methodology

For anti-phishing algorithms, two typical evaluation methodologies exist, i.e., *randomized evaluation* and *time-based evaluation*. The former is mainly to inspect the overall performance on all the available data, while the latter is to examine the performance under more real-world scenarios, training models on the past data and applying the models to future cases. To fully evaluate our approach, we adopted both methodologies in our experiment.

In light of a significant percent of near-duplicate phish with high similarity in terms of content due to the use of toolkits, it is necessary to see how our approach performs on the testing data with unique phish and with near-duplicates of the training phish respectively. Ideally, the TP on the testing set with unique phish should be reasonably high, and the TP on the testing set with near-duplicate phish of the training set should be even higher, if not 100%. Accordingly, we conducted two series of experiments under each evaluation methodology, using unique testing phish and near-duplicate testing phish respectively. This unique and near-duplicate dichotomy is important also because learning models with repetitive patterns in the training data tends to decrease the effectiveness of our machine learning approach.

In the randomized evaluation, we utilized both phish set 1 and phish set 2, and legitimate corpus 1 in this evaluation. We adopted the standard train, validation and test methodology, which is a common practice in machine learning. All the train/test splits were performed randomly. We reserved 70% percent of the legitimate set as the testing set, and used the remaining

30% for model training and tuning. To see the impact of the percentage of phish p in the training data on the detection performance, we built a series of randomly selected training sets varying p from 10% to 70%. In optimizing the algorithm parameters, those training sets with different p were further divided via stratified sampling into a training portion and a validation portion. Stratification ensures that the class distribution is preserved between the training and validation parts. In performing the final tests with the optimal model parameters, the whole training sets were used to train the classifiers. To reduce random variation and avoid lucky train/test splits, we used the average statistics over 10 runs in all our experiments.

In the time-based strategy specifically, we utilized the timestamps of the phishing attacks and simulated real-life scenarios by training our models on the past data and testing them on future data via a sliding window mechanism. We used legitimate corpus 2 and phish set 2 in this experiment. Specifically, we moved a sliding window of length L (in terms of days) step by step along the time line and applied our detection algorithm to the web pages with timestamp T_i (day in our current evaluation) using models learned on a training set composed of the phishing data with time labels falling in window $[T_{i-L}, T_{i-1}]$ and a subset of randomly selected legitimate web pages from legitimate corpus 2 in Table 6.1. Varying the percentage of phish p in the training set from 20% to 70% controls how many legitimate cases to be chosen for each sliding window. After a subset of legitimate pages are randomly selected into the training set for each sliding window, the rest of the legitimate corpus together with the phishing instances on time point T_i make their way into the testing set for the evaluation of TP and FP on time point T_i. The reported TP and FP are the mean of the TP_i and FP_i at all time points.

6.7 Experimental Result

6.7.1 Hash-based Near-duplicate Phish Detection

Our goal is to see the extent to which phishing toolkits are employed to produce phishing site replicas, and thus we compute the hash values for all phish in our corpus using the algorithm given in section 4 and explicitly keep only one copy among the web pages with identical SHA1. Table 6.2 shows that 72.67% phish are replicas according to our hash-based filtering algorithm, suggesting the effectiveness of this hashing-based filter in capturing near-duplicate phish and their simple variants.

Table 6.2: Statistics of near-duplicate phish detection. In detecting near-duplicate phish in set 1, we only examined the phish set 1 for hash matches, while for phish set 2, we scanned through both phishing sets. There is only one common duplicate phish between two sets, suggesting that phishers do not replicate phishing sites created long ago for future attacks.

	Phish set 1	Phish set 2
Download time	May 2008 – Sep 2008	Feb 27, 2009 – Apr 2, 2009
Total size	6943	1175
#unique web pages	1595	624

6.7.2 Login Form Detection

As shown in Table 6.3, we detected 98.06% phishing pages with login forms, and filtered a significant percentage of good pages from other categories. For the remaining 1.94% phishing

pages, they either do not have a login form (very rare in our phish corpus), use login keywords not in our list such as "serial key", or organize the form/input tags in a way our method misses.

This step contributes to the reduction of FP in that a certain portion of legitimate pages as shown in Table 6.3 are removed from being processed by the feature extraction and classification modules. Note that we actually did not particularly train the models on pages without forms in the whole layered system. Some legitimate pages for training purposes happen to contain no login forms. If the training set consists solely of pages with login forms, the TP and FP will both be higher.

Table 6.3: Statistics of login form detection in the evaluation of CANTINA+. 98.06% phishing pages with login forms were successfully detected.

Corpus	Phishtank	Alexa	3Sharp	Banks	Yahoo	Prominent
#total pages	2219	1023	101	985	371	81
#detected with login forms	2176	263	31	229	77	73
% good pages filtered		74.29	69.31	76.75	79.25	9.88

6.7.3 Randomized Evaluation

The main goal of randomized evaluation is to inspect thoroughly the overall performance of CANTINA+ on all our data via stratification and multiple run averaging, which is a standard practice in machine learning. In this section, we show the performance of our layered method under the smallest percentage of training phish, i.e., 10%, mainly due to two reasons. First, this setting is more realistic because in the real-world scenario, the volume of legitimate cases typically far outnumbers phishing attacks. Second, our approach did not manifest drastic difference under various percentages of training phish. For the same reason, we only give the experimental result under 20% training phish in the time-based evaluation in section 8.4. This number is different from the 10% in the randomized evaluation in that we only used legitimate corpus 2 in the time-based evaluation and for some sliding window, the volume of the training phish is large enough such that we do not have 90% legitimate pages for training.

Machine Learning Model Tuning

Machine learning algorithms use different strategies to regulate the learning process. We tuned our models on the validation set and used the optimal parameter values on the testing set.

For each algorithm, we found the optimal parameter with the best AUC [7] via 10-run tuning. Specifically, the optimal settings always improved the algorithms over the default parameters, mostly with less than 2% improvement in AUC. Typically, when the amount of training phish is insufficient, the AUC on the validation set is undesirable due to the low TP, and as the proportion of phish in the training set increases, AUC gradually amplifies and then possibly declines at the point where the degradation on FP outweighs the improvement on TP. In terms of the model complexity parameter, we see that mostly the classifiers achieved optimality when the amount of regularization is just appropriate. The major reason is that we are tuning the classifiers on a separate validation set, and overly small penalization leads to overfitting, while the other extreme yields undertrained models.

[7] Area under the ROC curve, introduced in section 6.6.1

In the testing phase of randomized evaluation, we assigned the models with the optimal parameters, and tested them on a separate testing set. We felt that this was a reasonable and feasible approach, since in a real deployment, one could tune the models offline and then employ the optimal setup for online scenarios.

Testing on the Holdout Data with Unique Phish

Multiple run averaging is a standard practice in machine learning, and the goal of this experiment is to examine the performance of our feature set trained on randomly selected phish and tested on the remaining unique phish. Specifically, we used all the good URLs in legitimate corpus 1 and all the unique phishing web pages in our collection, i.e., 1595 from phish set 1 and 624 from phish set 2. In Table 6.4, we show the performance of CANTINA+ in this experiment with 10% phish in the training data.

Table 6.4: Performance (10-run average) of CANTINA+ using Bayesian Network and CANTINA under randomized evaluation. For all cases in the table, CANTINA+ was trained with 10% phish in the training set. The legitimate testing sets are the same for the evaluations on both unique testing phish and near-duplicate testing phish, and therefore, the FPs remain the same. Overall, CANTINA+ significantly outperforms CANTINA. CANTINA has no explicit training phase, and is not influenced by the percentage of phish in the training set.

		Type of phish in the testing set					
		Unique			Near-duplicate		
Algorithm	Login filtering	TP (%)	FP (%)	F1	TP (%)	FP (%)	F1
CANTINA+ (with BN)	Yes	92.54	0.407	0.9592	99.63	0.407	0.9961
	No	93.47	0.608	0.9632	99.64	0.608	0.9952
CANTINA	Yes	71.47	0.335	0.8320	93.17	0.335	0.9630
	No	72.15	0.714	0.8348	93.19	0.714	0.9612

As shown in Table 6.4, CANTINA+ achieved a high TP of 92.54% and 93.47% with a low FP of 0.407% and 0.608% with and without login form filtering respectively. The filtering step makes the TP significantly worse. For all cases including CANTINA, FP filtering via login form detection significantly improves FP because a certain number of legitimate pages (Table 6.3) are detected with no login forms.

Testing on the Holdout Data with Near-duplicate Phish

The goal of this experiment is to show that learning with our feature set performs very well, if not perfectly, on the near-duplicate of the phish in the training set under the randomized evaluation setting. This is critical in demonstrating the power of our machine learning approach since we might as well directly use the simpler hash-based filtering if our proposed approach performs poorly on near-duplicate testing phish. In this experiment, we used a subset of good URLs in legitimate corpus 1 and unique phishing URLs from phish set 1 and phish set 2 for training, and tested our models on the near-duplicate phish from phish set 1 and phish set 2, i.e., a total of 5899 near-duplicate phish. In Table 6.4, we show the TP and FP of CANTINA+ in this experiment with 10% phish in the training data.

With a training set containing only 10% phish, CANTINA+ was able to achieve a very high TP of 99.63% and 99.64% with and without FP filtering respectively due to the strong similarity

between the phish in the training and testing data, manifesting the power of our feature set in capturing the characteristics of phishing attacks. The experiment result also shows no significant effect of the FP filter. Since our legitimate corpus contains no replicated instances, choosing unique or near-duplicate testing phish only influences TP and the FP remains the same, as shown in the table.

6.7.4 Time-based Evaluation

The randomized experiment in the previous section aims at evaluating CANTINA+ using the standard machine learning practice in which the training and testing data are randomly selected with multiple runs to reduce variance. The main goal of time-based evaluation, however, is to inspect the performance of CANTINA+ in real-world scenarios, in which we train our models using historical data and apply the models to future data. Since the overhead of stepwise parameter tuning for each sliding window is prohibitive, we did not explicitly tune the model parameters in this experiment and simply used the default values. In addition, because the original CANTINA has no training process, its performance should remain identical and we therefore did not compare our approach with CANTINA again in this evaluation. The experimental result of this time-based evaluation using a two-week sliding window with 20% training phish in each sliding window is shown in Table 6.5.

Table 6.5: Performance of CANTINA+ using Bayesian Network under time-based evaluation with a two-week sliding window. For all cases in the table, CANTINA+ was trained with 20% phish in the training set. The legitimate testing sets are the same for the evaluations on both unique testing phish and near-duplicate testing phish, and therefore, the FPs remain the same. CANTINA+ achieves a high F1 of over 0.95 under all cases.

		Type of phish in the testing set					
		Unique			Near-duplicate		
Algorithm	Login filtering	TP (%)	FP (%)	F1	TP (%)	FP (%)	F1
CANTINA+ (with BN)	Yes	92.25	1.375	0.9529	99.25	1.375	0.9894
	No	94.24	1.948	0.9607	99.64	1.948	0.9886

Result on Unique Testing Phish

The goal of this experiment is to evaluate our approach on real-world phish stream with models trained on historic phishing attacks that have no overlap with the phish in the testing set. In this experiment, the positive corpus comes from the 624 unique phishing URLs from phish set 2, and the negative data set uses all the pages in legitimate corpus 2.

Table 6.5 shows that CANTINA+ achieved a high TP of 92.25% and 94.24% with and without FP filtering respectively with only two weeks' worth of phish in the training set for each sliding window. Login form filtering makes the TP significantly worse yet benefits the FP significantly.

We see from Table 6.4 and Table 6.5 that the TPs under the two evaluation strategies are comparable. However, the number of training phish with different sliding windows in the time-based evaluation varies significantly, from a minimum of 19 to a maximum of 398, causing considerable variations in the resultant TPs across days, which is confirmed by the result that the maximum TP did not always occur under the setting with 70% training phish for each algorithm.

Table 6.5 also shows that with 20% phish in the training set in each sliding window, the FP of CANTINA+ with no login form filtering is 1.948%, which drops to 1.375% with login form filtering, both worse than the counterpart in the randomized experiment. The gap in FPs under the two evaluation methodologies can be attributed to the following observations. First, we conducted 10 experiments and averaged the resultant statistics for each setting in the randomized evaluation, which helped reduce random variations in the performance. On the other hand, we could not perform 10-run averaging in the time-based evaluation due to the nature of this experimental strategy, since the training phish in the sliding window prior to the current time point are fixed and could not be randomized. This one-time random selection of legitimate pages might cause unlucky train/test split, leading to variable FPs. Second, learning models are optimized in the randomized experiment, while default parameter values are used in the time-based analysis.

A breakdown on the performance by days shows that the FP of our approach on the second day in our corpus is significantly worse than the other days, and after that the FP stays relatively stable. This is caused by the fact that we only have one day's worth of training phish in evaluating our model on the web pages of the second day, leading to a very small training set and therefore the undesirable FP.

Result on Near-duplicate Testing Phish

The goal of the experiment in this section is to demonstrate the effectiveness of our feature set on real-world phish stream with near-duplicate attacks of the training phish. Specifically, we utilized part of the 624 unique phishing URLs from phish set 2 and part of legitimate corpus 2 for training, and evaluated the models on the 550 near-duplicate phish from phish set 2 and the remaining URLs of legitimate corpus 2.

Overall, the pattern in the result of this experiment in Table 6.5 is similar to that of the experiment on unique testing phish in section 8.3.2. Particularly, the TP of CANTINA+ is over 99%, higher than the statistics in the experiment on unique testing phish, which is what we expected since the testing phish here highly resemble the training phish. Moreover, the TPs on near-duplicate testing phish under the time-based evaluation are also on the same level as those under the randomized evaluation.

6.7.5 Result under Various Percents of Training Phish

In deploying our system for real-world applications, we need to build a training set and train our approach in advance, for which the percentage of phish in the training data is a key parameter. To examine the impact of the ratio of the two types of web pages in the training data on the performance of our approach, we varied its value and evaluated the performance of CANTINA+ under both the randomized and time-based evaluation. The experimental result is given in Fig 6.2 and Fig 6.3.

In Fig 6.2, a significant positive correlation is seen between TP and the percent of phish in the training data in all cases but the time-based evaluation on near-duplicate phish. This trend is self-evident in that machine learning models are able to detect more phish with more phishing patterns in the training set. Fig 6.3 illustrates a similar story in terms of FP, with a significant negative correlation between FP and the percent of phish in the training data. Although the FP of CANTINA+ deteriorated as more training phish were added, the ratio between opposite classes in the training data, among other parameters, is of our choosing, and we can always

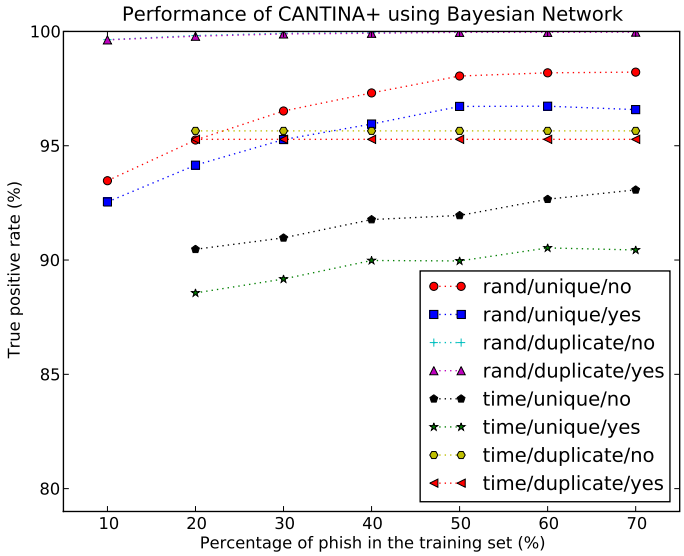

Figure 6.2: TPs of CANTINA+ using Bayesian Network. Eight curves are shown, corresponding to all the combinations of evaluation methodology, type of testing phish and the use of login form filtering. A significant positive correlation is seen between TP and the percent of training phish. With other settings being identical, CANTINA+ always performs better on near-duplicate testing phish than on unique testing phish, and login form filtering makes the TP significantly worse. The two curves corresponding to the randomized evaluation on near-duplicate testing phish with and without login form filtering almost coincide.

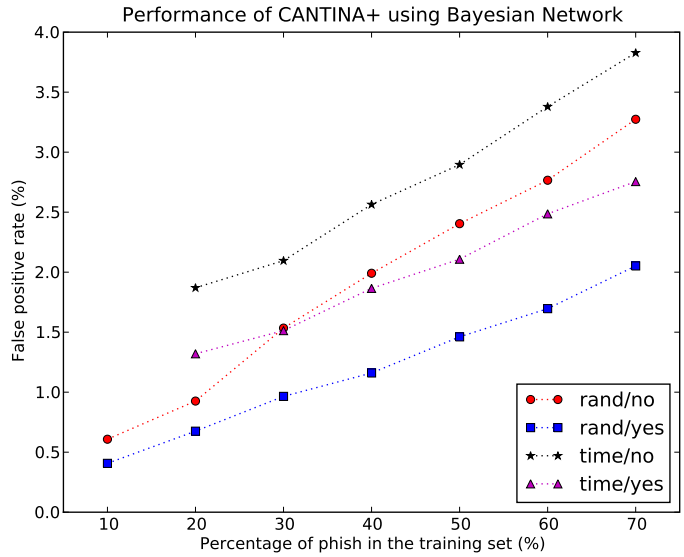

Figure 6.3: FPs of CANTINA+ using Bayesian Network. The four curves correspond to the performance of our approach on unique testing phish. FP rises as the percentage of phish in the training data increases. Login forming filtering makes the FP significantly better.

optimize our approach to guarantee the best generalized performance. For example, we can adopt cost-sensitive techniques to learn models [91].

6.7.6 Comparing CANTINA+ vs CANTINA

Zhang et al proposed CANTINA, a content-based method, which performed competitively in their experiment against two state-of-the-art toolbars, SpoofGuard and Netcraft [92]. We implemented an offline version of CANTINA, and evaluated our hierarchical CANTINA+ with CANTINA on the same testing sets in the randomized evaluation. Table 6.4 shows that our CANTINA+ always outperformed CANTINA by a huge margin in terms of TP, with a comparable FP.

In the experiment on unique testing phish, CANTINA+ outperformed CANTINA with a huge margin of over 0.12 in terms of F1, a statistically significant result indicating the superiority of CANTINA+. In particular, CANTINA+ gained an over 20% improvement over CANTINA on TP. Notably, our experiment shows a TP of about 71% for CANTINA in Table 6.4, drastically different from the TP of 89% in our original CANTINA paper [92]. Three factors mainly caused this discrepancy. First, one feature in CANTINA assumes phishing mostly focuses on nine target sites and examines the inconsistency between the nine logos and the page domain. As phishing attacks evolve, however, the distribution of the most phished brands changes, and this feature in CANTINA often fails. Second, CANTINA learns its feature weights on a rather limited 200 URLs, leading to a significantly undertrained model. For instance, the "IP address" feature in CANTINA almost never fires alarms, but has a non-trivial weight. Third, we evaluated CANTINA in this work on a much larger collection with harder testing cases, compared with the 200 testing URLs from three categories in [92].

Particularly, CANTINA has no explicit training stage, which explains the phenomenon that the TPs of CANTINA only show slight fluctuations under an increasing value of the percentage of training phish. Furthermore, the testing legitimate sets remain the same under various ratios of training phish, and thus the FPs of CANTINA are thus constant.

In another experiment comparing CANTINA+ with CANTINA, we evaluated both methods on the 5899 near-duplicate phish from phish set 1 and phish set 2, and report the result in Table 6.4. The statistics indicate that CANTINA+ still outperformed CANTINA on this data set with a F1 of around 0.99 versus 0.96. Particularly, CANTINA+ beat CANTINA by far in terms of TP, with a roughly 6% margin. We observe that the TP of CANTINA is about 93% in this experiment, significantly better than its 71% TP on the unique testing phish. The cause of this is that we utilized a much larger testing set with substantial near-duplicate of the training phish in this experiment.

6.7.7 Learning with Individual Features

The statistics in the previous sections were obtained by using the whole feature set (15 features in total), and in this section, we evaluate the contribution of each single feature to the overall performance. We refrain from using TP/FP and instead stick to the summary statistic AUC in measuring the performance of each individual feature, because the separability of opposite classes in the 1-dimensional input space is prone to the impact of the quality of training data, and TP/FP may exhibit high variance since they only capture a single aspect of the big picture.

Across the learning algorithms, BN, Adaboost, RF and LR perform comparably, with all significantly better than J48, which in turn significantly outperforms SVM. We find through our experiment that the correlation between the AUC of each feature and the percent of phish in the

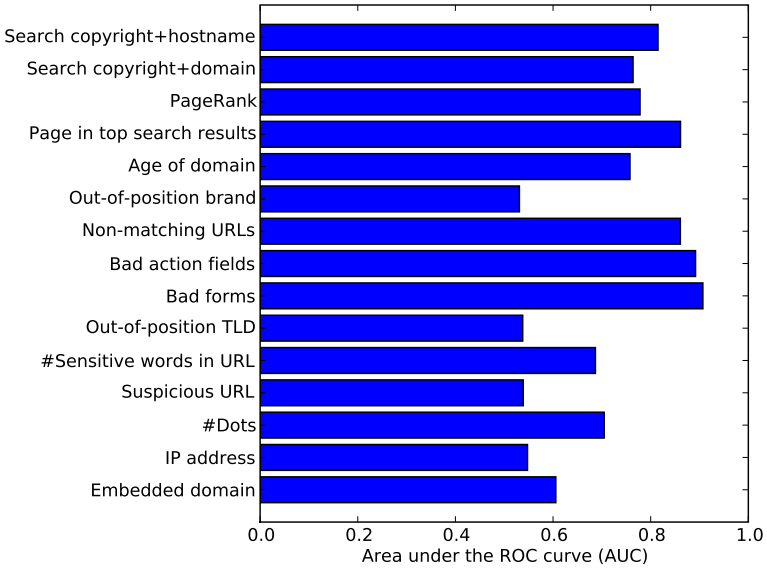

Figure 6.4: Area under the ROC curve (AUC) of BN with each single feature under 10% training phish. Top-performing features include bad forms, bad action fields, non-matching URLs, and page in top search results, with over 0.85 AUC.

training data is almost zero, and therefore, we only report the result with BN under 10% training phish in Fig 6.4.

Fig 6.4 shows that a few features clearly stand out from the others with over 0.85 AUC, including "bad forms", "bad action fields", "non-matching URLs", and "page in top search results". Besides, "age of domain", "search copyright company name plus domain", "search copyright company name plus hostname" and "PageRank" also perform fairly with over 0.75 AUC, though less stellar than the above four superstars. The remaining features are apparently inferior, with under 0.7 AUC or even close to 0.5 AUC, almost amounting to random guessing.

Chapter 7

Detecting Phish via Logo Images

The brand logo image is a key element in representing the identity as well as the look-and-feel of a web site, and a robust technique that exploits this key element on a web page provides a potentially effective anti-phishing approach. However, previous research simply adopted the rigid pixel-wise matching strategy in exploring the images for phish detection, and is thus easy to beat.

In this chapter, we propose an approach that takes on phishing attacks by examining the inconsistency between the claimed identity (e.g., an eBay logo) and the genuine identity (e.g., a rapidshare.com domain) of a web page via this strong signal. Our approach utilizes novel features with machine learning (ML) algorithms to identify the logo image, and then classifies a web page as phish or not by inspecting the inconsistent identities via near-duplicate image matching techniques.

7.1 Introduction

One major reason that people fall for phishing attacks is that the look and feel of the fake web site is so similar, if not identical, to the genuine web site [28, 29]. Among the various components in the HTML DOM of a web page, perhaps the greatest influence on the perceived trustworthiness of a web page is the brand logo image, which is a graphic mark or emblem commonly used by commercial enterprises, organizations and even individuals to aid and promote instant public recognition. To boost the look and feel of the phishing web sites, phishers very often use on the phishing web pages replicas or near duplicates of the official logos from their target sites. If one considers what lures users into a phishing site in the first place, the look and feel of the web page is much more salient in their experience than the text, the URL and other features. In the technique proposed in this chapter, we attack the phishing problem by leveraging the key component of the look and feel of a web page, namely the brand logo image.

We propose a layered approach that takes on phish based on a highly predictive and representationally rich signal, i.e., the discrepancy between the claimed identity (e.g., an eBay logo) and the genuine identity (e.g., a rapidshare.com domain) of a web page, by first identifying the brand logo image via machine learning algorithms and then examining such identity contradiction via near-duplicate image matching techniques. Specifically, the former step extracts features for an image and assigns a probability of it being the brand logo via machine learning techniques; the latter seeks a match for images with a high probability (typically a probability of larger than 0.5) among a database of official logos for popular phishing targets, and compares the domain name of the web page and that of the matching official logo. Using near-duplicate image matching

techniques based on our logo identification algorithm is robust and capable of finding matching regions between images even after various transformations, leading to a good TP. On the other hand, our approach is designed to have very low FP because logos are meant to be visually distinct and legitimate web sites are extremely unlikely to take another web site's logos.

Our contributions to the literature are three fold.

1. Our work is the first in designing features for images on a web page and computing the probability of each image being the brand logo via machine learning to facilitate phish detection. With those statistics, the following phish classification step can simply use the images with a probability of over a threshold[1] to match against a set of official logos, achieving significant runtime speedup over other techniques that process all images in a brute force fashion.
2. Our work is the first attempt to detect phish by matching the identified candidate logos against official logos for popular phishing target sites via near-duplicate image matching algorithms [93]. Near-duplicate image matching techniques compare two images' content by aligning their local interest points (LIP), which is much more robust than the simple techniques based on pixel-wise comparison adopted in existing works.
3. Our work provides a highly reliable and easily extensible anti-phishing technique that focuses on most phished brands. It has an FP of 0% and a TP of 87.63% on a holdout testing set consisting of 257 legitimate web pages and 283 phish, without even using a domain whitelist filter, which is the best FP and a competitive TP against the techniques in the literature.

7.2 Algorithmic Details

Figure 7.1 shows the architecture of our anti-phishing solution. The logo image bears the maximum amount of information about the brand name of the corresponding web page, and our research in this chapter centers around this key element in the HTML DOM to detect phish.

In our system, we explicitly train the logo classifier on a training set of images from phish and legitimate web pages. In evaluating our approach on a stand-alone data set, we apply the logo classifier on the images extracted from the testing web pages, and then use an off-the-shelf algorithm for near-duplicate image matching [93] on the candidate logos to detect phish. In the rest of this section, we will talk about the detailed algorithm for each component.

7.2.1 Clustering-based Near-duplicate Phish Removal

Given the fact that a great volume of phishing attacks are similar in terms of their textual content due to the wide adoption of toolkit in creating phish automatically in batch [61], we designed a near-duplicate removal algorithm based on the well-known DBSCAN clustering algorithm [31] to retain the phish corresponding to cluster centroids for the following training and testing purposes. This gives a more realistic assessment of the performance, helping to ensure that results are not inadvertently inflated due to duplicates.

A clustering algorithm requires a metric to gauge the similarity between two objects, and in our work, we used the shingling technique to meet this purpose. Shingling is a well-known technique for identifying similar documents in information retrieval (IR), which has also been

[1]This threshold needs to be consistent with the cutoff value used by the machine learning algorithms in classifying logos, which is typically 0.5.

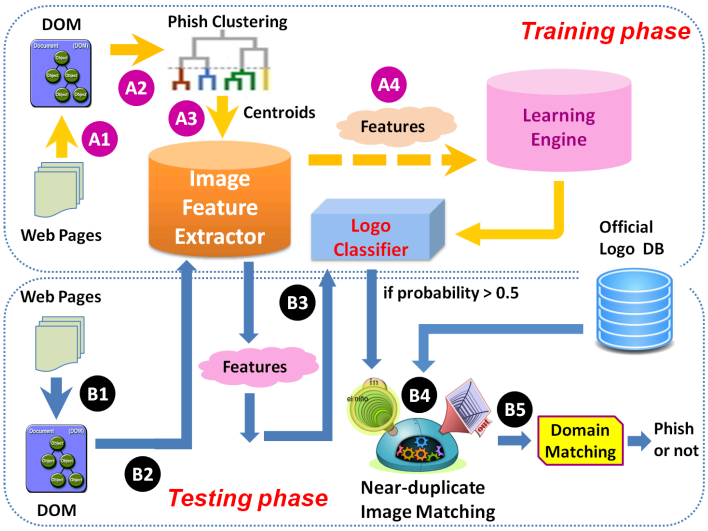

Figure 7.1: Key steps in the training stage, A2) training phish are clustered with the density-based DBSCAN algorithm; A3) features are extracted from all the images on the phishing web pages corresponding to the cluster centroids, as well as all legitimate web pages in the training set. Key steps in the testing stage, B3) a probability of each image being the logo is computed by the logo classifier; B4) images with a probability of over 0.5 is compared with the official logo images in a pre-compiled logo database via near-duplicate image matching; B5) once a matching logo is found, the domain name of the web page is examined the against the domain name for the official logo, and the web page is classified as phish if the two domains do not match.

used in anti-phishing [53, 86] with success. The core idea is to break up the content of web pages into n-grams and then examine how many n-grams they have in common.

In particular, shingling [19] employs a similarity metric named *resemblance* to calculate the percent of common n-grams between two web pages. Formally, let p and q represent two web pages, and *resemblance* $r(p, q)$ is then defined as

$$r(p, q) = \frac{|S(p) \cap S(q)|}{|S(p) \cup S(q)|} \tag{7.1}$$

where $S(p)$ denote the set of unique n-grams in p.

7.2.2 Logo Classification via Machine Learning

The central idea of this component is to classify which image on a web page is likely to be the brand logo via machine learning algorithms. A web page typically contains many images of various sizes, and when inspecting the legitimacy of a web page via the brand logo, it is desirable to be able to identify with a high probability which image is the logo, and inefficient to process all images in a brute force fashion.

Fortunately, the brand logo image on a web page tends to have some special properties compared with other images, and we summarized them into five features.

1. ***Position of Image on the Web Page***. From a visual design perspective, the brand logo is always displayed in the most distinct position, which usually means the top-left region of a web page. To make use of such information, we parse the HTML DOM, rank the images

by their positions first from top to bottom and left to right of the web page, and then use the order as the value for this feature.

2. ***Width of the Image***. In general, the logo image needs to meet the visual needs of the reviewers, and therefore, its pixel size typically falls in a certain range. Hence we take the image width as a feature.
3. ***Height of the Image***. As explained in the previous feature, we take the image height as a feature.
4. ***Presence of Logo-related Keywords***. A common practice in creating a web page is to employ an alternative text attribute in cases when images and other elements are not supported or cannot be properly displayed by users' browsers. Such "alt" attribute provides extra information about the HTML elements, and in this feature, we retrieve keywords such as "logo" from the "alt" attribute of the image tag (¡img¿) and assign binary values to this feature according to the presence of such keywords.
5. ***Similarity between the Textual Description of Image Tags and that of the Web Page Title***. Title is the best summarization of the content of a web page, which also usually contains the brand name of the web site, and in this feature, we exploit this property and utilize the similarity between the title and the textual descriptions of an image to distinguish logos and other images. Specifically, we extract words from the following places in the HTML for each image tag: the src, alt and ID attribute of the image tag, as well as the image caption which is usually the piece of text following the image in the HTML. We then compute a score based on a similarity model proposed by psychologist Tversky [76], which measures the similarity between objects in terms of their common and distinctive features as defined in the following formula

$$\text{Similarity}(k) = \frac{|W_T \cap W_I(k)|}{|W_T|} \tag{7.2}$$

where $W_I(k)$ is the set of words extracted for the k-th image tag and W_T is the set of words extracted from the web page title.

With these features, we represent each image by a feature vector, and the logo classification problem simply boils down to a machine learning task like this. Given n images I_i ($i \in \{1, 2, \ldots, n\}$) each represented by $< \overline{f}_i, y_i >$ where $\overline{f}_i$ denotes the m-dimensional feature vector [2] $\{f_{i1}, f_{i2}, \ldots, f_{im}\}$ as defined above and $y_i \in \{0, 1\}$ is the class label for I_i (i.e., $y_i = 1$ if I_i is the brand logo), we build a machine learning model M over $I_1, \ldots, I_n$, and for a new image NI_j with feature vector $\{f_{j1}, f_{j2}, \ldots, f_{jm}\}$ whose y_j is unknown, we will predict its $\widehat{y}_j$ with a probability of p_j using M. Typically, NI_j is regarded as a logo if $p_j > 0.5$.

It is likely that phishers may attempt to spoof our logo classification, for example, by changing the size and position of the logo image, however, machine learning algorithms are robust against a certain amount of noise in the features, and moreover, there is a limit on how far phishers can go before web users start feeling suspicious about the legitimacy of the web pages.

The truly time consuming operation in our approach is comparing images via near-duplicate image matching, not computing p_j with machine learning over the features. By selecting the images with $p > 0.5$ as candidate logos to match against a pool of official logos for popular phishing targets, we achieved tremendous runtime speedup because there are typically at most 2 images with $p > 0.5$ out of the many.

[2] $m = 5$ in this context

7.2.3 Phish Detection via Near-duplicate Image Matching

In this step, we take all candidate logos (those with $p > 0.5$ computed by the logo classifier), and seek a match for each of them in a set of official logos for popular phishing targets via near-duplicate image matching techniques. Once a match is found, we examine the domain of the web page and that of the site the matching official logo belongs to. The details are shown in Algorithm 6. The rationale behind our algorithm is that, although a phishing web page uses exact copies or variants of logos from legitimate web sites, the phishing web site's domain will differ from that of the target's. This algorithm scales well, because we can easily add more logos to our set of official logos and moreover, near-duplicate matching can be distributed across multiple machines leading to an ideal $O(N)$ complexity, where N is the number of candidate logos (those with $p > 0.5$) to check.

A key ingredient in this phish detection step is near-duplicate image matching, which is capable of finding regions with similar content in two images even after the images have gone through a serial of geometric and photometric transformations. This has significant advantage over existing anti-phishing works, all of which simply compare the pixel-wise similarity and thus are trivial to defeat. In particular, we adopted the existing technique in [93], which identifies near-duplicate image pairs by extracting the local interest points (LIP) from images and matching the LIP sets from two images efficiently via a one-to-one symmetric matching (OOS) algorithm with a special index structure to facilitate nearest neighbor search for LIPs. Particularly, [93] adopts the PCA-SIFT descriptor [45] to represent LIPs, which has been shown to be highly distinctive, invariant to image scale and rotation, robust to color and photometric changes, and thus suits the anti-phishing task very well.

Algorithm 6 DetectPhish

Require: logo classifier M, official logos from legitimate sites L_{DB}, probability threshold P, web page t

Ensure: predicted class label C for t, i.e., $C = 1$ if t is a phish, and $C = 0$ if not

1: $D \leftarrow$ extract DOM from t
2: $\overline{f}_i \leftarrow$ extract features for each image I_i from D
3: $p_i \leftarrow$ apply M to compute the probability of each $I_i = \overline{f}_i$ being a logo
4: **for** each image $I_k \in D$ with $p_k > P$ **do**
5: search L_{DB} for a match via the near-duplicate matching algorithm
6: **if** (logo L_j in L_{DB} matches I_k) && !match(L_j's domain, t's domain) **then**
7: **return** 1
8: **return** 0

7.3 Experiment Setup

7.3.1 Web Page Corpus and Official Logos from Phishing Target Sites

To ensure consistency in our evaluation, we downloaded the phishing web pages when they were still alive. In particular, we used the phish feed of Phishtank, and collected 11,056 phishing web pages, which targeted at 154 well-known brands. After clustering all these phish with our shingling-enhanced DBSCAN algorithm, we had a total of 970 phishing web pages in the cluster centroids, and we used this reduced phish set for training and testing.

As to the legitimate corpus, we collected 513 web pages from Alexa's catalog of top 500 sites on the web [14]. Note that a lot of these are actually common phishing targets. Although we did not include those less popular legitimate web sites in our evaluation, by evaluating our approach on the heavily-phished web sites, we actually provide pessimistic performance statistics especially in terms of FP, which is more beneficial for an objective evaluation of our method and its real-life application that follows.

For the official logos used for near-duplicate image matching, we collected 550 logo images from 360 legitimate web sites in Alexa's top sites, covering most of the highly phished brands. One web site may have multiple different logos, such as bank of America, and we manually downloaded all the variants.

7.3.2 Machine Learning Algorithms for Logo Classification

We compare 6 ML algorithms in training the brand logo classifier, including Bayesian network (BN) (a probabilistic graphical model that makes inferences via a directed acyclic graph), J48 decision tree, Support Vector Machines (SVM) [21], logistic regression (LR), random forest (RF) and Adaboost. All the ML algorithm implementations were taken from the Weka package [83].

7.3.3 Evaluation Methodology

To fairly evaluate our anti-phishing approach, we adopted the common practice in machine learning to randomly split our whole data set into a training portion, on which parameter tuning and model building was performed, and a testing portion, on which the model learned on the training set was evaluated.

In creating our training set, we used all the images from 70% of our phishing web pages randomly chosen from the cluster centroids obtained in the clustering process, and all the images from 50% randomly selected legitimate web pages. We used all the remaining images for testing.

Moreover, we manually label all the images on the web pages in our corpus, so that we have ground truth in evaluating our logo classification algorithm. The breakdown on the specific number of images in our training and testing sets is shown in Table 7.1.

Table 7.1: Statistics about our training and testing sets. The quantities inside and outside the parentheses denote the number of images and web pages respectively in the corresponding cell.

Source	Training Set	Testing Set
Legitimate web pages	8182 (256)	7384 (257)
Phishing web pages	3249 (687)	1327 (283)

7.4 Experimental Result

7.4.1 Model Tuning for Logo Classification

Model tuning refers to the process in which we optimize the settings of our learning algorithms on the training data for the following testing stage. The only model that we need to explicitly train in our system is the logo classifier, and to obtain the optimal configurations for the logo classifiers, we tuned the machine learning algorithms on the training data with 10-fold cross-validation (cv), which is a standard evaluation strategy in machine learning to reduce the variance of the resulting

estimates. We found through a series of experiments that discretizing the continuous features yielded the largest improvement in performance, while adjusting parameter values for the learning algorithms did not produce much benefit. Therefore, we applied a classic feature discretization algorithm [33], which utilized the minimum description length principle (MDL) to determine the partitioning of feature intervals, on both the training and testing sets prior to the model building and testing phases. Table 7.2 shows the result of this step.

As suggested by the table, our feature set performed well in classifying images into logos under all machine learning algorithms, especially BN, both of which yielded a TP of 89% with a FP of 0.8%. In particular, all algorithms had a high F1 value of close to 0.9, manifesting the superiority of our proposed feature set in identifying brand logos among other images. Note that these numbers were achieved without using domain whitelist filtering [88].

To gain further insight about the efficacy of each feature in discriminating logos from other images, we analyzed the cross-validation result and found that the absence of logo-related terms is a good indicator of non-logos, and a higher position in the HTML DOM (top-left part of the web page) usually implies the corresponding image is the brand logo.

Table 7.2: Comparing different machine learning algorithms in classifying logos in the training data (10-fold cross-validation), including Bayesian networks (BN), J48 decision tree, Support Vector Machines (SVM), logistic regression (LR), random forest (RF) and Adaboost. All algorithms achieved a high F1 value of close to 0.9, suggesting the efficacy of our features in recognizing logos among other images. The best F values are marked by $*$.

	BN	J48	SVM	LR	RF	Adaboost
TP (%)	89.0	87.4	86.9	87.5	87.3	84.8
FP (%)	0.8	1.0	0.8	0.7	0.8	0.7
F-measure	0.899 ($*$)	0.881	0.89	0.897	0.889	0.879

7.4.2 Classifying Web Pages with Our Approach

After tweaking the models, we adopted the best configuration and embarked on the evaluation of our proposed approach in detecting phish in the holdout testing set. As introduced previously, detecting phish has two main steps, i.e., identifying logos and classifying web pages, and we will report the performance of each in this section.

Logo Classification on the Testing Set

In this section, we report the result in identifying logos on the holdout testing data, with the logo classifier trained using our proposed feature set on the whole training data.

The result is shown in Table 7.3. Again, all algorithms had a high F1 value of about 0.9. Consistent with the cross-validation result reported in the last section, BN outperformed other algorithms with a high TP of nearly 90% with a low FP of 0.7%. In particular, the statistics reported on the testing set in Table 7.3 are better than those from the cross-validation result, and our explanation is that the size of the data we actually used in training the logo classifiers is bigger in the former than in the latter.

Since this testing data set was never touched in the previous model tuning stage, the superior performance indicates that our proposed features are truly capable of identifying brand logos, which builds a solid groundwork for the phish detection step in the next stage of the whole system.

Table 7.3: Performance of logo classification on the holdout testing data. All algorithms achieved a low FP of below 0.7%. BN reached a high TP of almost 90%. The best F values are marked by $*$.

	BN	J48	SVM	LR	RF	Adaboost
TP (%)	89.1	87.8	86.5	86.7	85.8	82.5
FP (%)	0.7	0.6	0.6	0.5	0.6	0.5
F-measure	0.892 ($*$)	0.891	0.886	0.892 ($*$)	0.885	0.871

Phish Detection on the Testing Set

Since Bayesian Networks (BN) is shown to perform among the best in identifying logos in the previous section, we picked the images with the probability of being the logo (specifically those with $p > 0.5$) output by BN, and report in this section the performance on phish detection via near-duplicate image matching. Our experiments showed that our phish detection algorithm achieved a TP of 87.63% and an FP of 0% on the holdout testing data, as shown in Table 7.4.

Table 7.4: Performance of our logo-based technique in detecting phish on the holdout testing set with 257 legitimate web pages and 283 phish. Based on our logo classifier trained with Bayesian Networks over images from 256 legitimate web pages and 687 phish, our approach has a competitive TP of 87.63% and the best FP of 0% in the literature.

	TP	FP
Our approach	87.63%	0%

Since a web page is classified as phish only when the image that our logo classifier identifies as the brand logo matches one of the official logos in our logo database, and its domain does not match the domain of the corresponding official logo, the 87.63% TP indicates that our proposed algorithm which exploits the discrepancy between the double identities of a web page is effective in detecting phish.

Although we did not explicitly evaluate our approach against other techniques in the literature on the same data set, the sheer performance statistics still provide some insight into a comparison among all. Typically, the TPs of previous works range from 73% to 96%, and FPs lie between 0% and 12%, as introduced in the related works section. On our side, the 87.63% TP of our approach in this work is comparable to existing methods, and the 0% FP is the best, which is a very good sign as to the potential of our proposed approach to be an effective and practical anti-phishing solution, given the concern in industry over liability issues due to high FPs.

The TP in this section is somewhat lower than that in the logo classification step in the previous section, which comes as no surprise since our phish detection algorithm is based on the candidate logos identified by the logo classifier and thus the final TP is bounded from above by the TP of logo identification. Moreover, there are 12 phish in our testing set for which our crawler failed to download the logo images, which is another factor causing this difference because the TP of phish detection is computed over the number of phishing web pages and that of logo classification is calculated on the number of logos.

Runtime Performance

The most time-intensive phase in our pipeline is web page classification via pairwise near-duplicate image matching, which involves considerable computations in terms of feature extraction from local interest points (LIP), LIP alignment via bipartite graph matching, and nearest neighbor search in LIP set matching. Compared with a naive approach that explores all images on a web page to detect phish, our approach has significant advantages in terms of runtime performance mainly due to the logo classification mechanism that recognizes the brand logo with a high accuracy among the many images, such that the web page classification step via image matching can focus on the a few images highly likely to be the logo.

To verify the hypothesis of our runtime performance, we compared the web page classification step (image matching) of our approach with the traditional strategy that blindly iterates over all images to match against official logos. Table 7.5 shows that our phish detector using near-duplicate image matching based on logo classification achieved substantial reduction in running time on a machine using Windows 7 with Duo CPU (3.16GHz and 3.17GHz) and 4GB RAM.

Interestingly, the average running time on legitimate web pages is higher than that on phish, especially with the brute-force search strategy. We found that this was caused by the observation that there were many more images (28.73 per legitimate web page) on a legitimate web page than on a phishing web page (4.67 per phish) in average in our testing data, as shown in Table 7.5. Actually, this phenomenon can be generalized to the whole web, since phishing attacks always target well-known financial institutions, whose web sites typically do not have many pictures, while for a lot of the popular legitimate sites like news portal, social networks, micro-blogs, photo sharing sites, etc., a large number of images are quite normal. Given the fact that the vast majority of web pages people view in their everyday browsing fall into the latter camp, our phish detection approach with logo classification has potential to be a practical and effective anti-phishing solution.

Table 7.5: Average running time (seconds) in classifying a web page in the holdout testing set between our approach based on logo classification against the traditional method, which compares all images on a web page with the protected logos. Our approach outperformed the brute-force method on all three data sets (i.e., phishing web pages, legitimate web pages, and both) by a great margin, with all results being statistically significant (marked by $*$).

	Phish	Nonphish	Both
Our approach (s)	2.55 ($*$)	2.78 ($*$)	2.66 ($*$)
Brute-force (s)	11.38	74.68	41.38

Chapter 8

A Feature-type-aware Cascaded Learning Framework

As shown in Chapter 3 to 7, we have proposed a few novel anti-phishing approaches, each of which aimed at improving existing techniques from a different angle and has been shown to be effective in our experimental evaluation.

In the real world, a lot of other fields share similar characteristics with anti-phishing, i.e., they all require a solution with low latency and reliable classification performance and they all have a heterogeneous set of features with varying cost and detection rate. Such fields, to name a few, include email analysis (spam and virus email filtering, etc.), web page analysis (anti-phishing [92], etc.), social network analysis (swearing language filtering in Twitter [87], etc.), image analysis (face recognition [89], etc.). Due to its capability in classifying future cases based on historical data, machine learning (ML) has been the de facto technique in those areas, and however, almost all state-of-the-art ML-based solutions in those fields utilize a monolithic model that computes all available features at once, which often yields no classification improvement and yet dramatically incurs unnecessary overhead.

In the context of anti-phishing, for instance, there are also still a few missing pieces to the whole picture. First, current server-side blacklisting methods cannot catch novel phishing patterns reliably unless URL blacklists are updated in realtime, which is infeasible in practice. Second, the distribution of web pages is highly skewed in favor of the legitimate cases, and a majority of the legitimate pages are simple cases which do not need all the expensive features to classify, thus rendering the process of extracting all feature values in most existing anti-phishing methods less desirable. In fact, some fast features turn out to be sufficient in identifying a good number of the legitimate cases and a high percent of phishing attacks. One such feature is the presence of login forms on a web page. This is based on the observation that the vast majority of the legitimate web pages have no login forms, and a simple feature checking login forms will do an excellent job in removing those web pages from further examination. Third, liability for false positives has been a major concern in industry. However, the existing ML-based techniques, all of which utilize the whole feature set in a monolithic classifier, either fail to deliver a low FP, or offer FP reduction via extra layers of filtering [86, 88]. In our thesis, we embed FP control in the stage classifiers of a single principled cascaded learning framework.

In an effort to offer more control over the three desiderata of FP, TP and runtime through an automated solution, we propose a feature-type-aware cascaded learning framework that exploits the distributional skewness of the web and builds different types of features into multiple stages

of a single cascade. By utilizing lightweight features in early stages of the cascade and postponing slower and more discriminatory ones to later stages, our approach achieves a superior runtime performance in general, and is independent of hardware infrastructure. In the context of anti-phishing, our approach achieves 55.7% reduction in runtime on average over stage-of-the-art single-stage models, with a low FP of 0.65% and a TP of 83.34%, and thus provides a fast and reliable solution for live detection scenarios. Moreover, our approach is scalable with more features and can be adjusted to emphasize TP or FP according to specific application domains.

The main research contributions of our proposed approach are two fold.

1. Our work is the first in providing a feature-type-aware cascaded learning framework that exploits the highly skewed distribution of the web and builds features of various types (such as text, structural, web, etc.) into a cascade consisting of classifiers in multiple stages. Our learning algorithm also enables practitioners to adjust the preference over FP or TP based on specific applications.
2. Our work is the first to provide a practically efficient solution for domains like anti-phishing, which is scalable with more features and can be further improved by distributed hardware infrastructure.

8.1 Feature-type-aware Cascade Learning

In reality, real-world domains typically require either a high TP with a reasonable FP or a very low FP with an acceptable TP, with both cases preferring low latency of the technique. Our proposed feature-type-aware cascaded learning framework can be customized to meet the demands of both scenarios. The principle domain of our cascaded learning technique in this thesis, i.e., phish detection, together with other domains like virus detection, belongs to the latter which has a much more strict requirement on the FP. For both camps, the essence of our proposed approach is to automatically build a cascade of classifiers using fast features in early stages and features involving expensive operations in later stages. In the rest of this section, we will briefly introduce the system architecture of the cascade that emphasizes FP more, and then elaborate on our feature set and the details of our cascaded learning framework such as structure learning in the context of anti-phishing.

8.1.1 Architecture Overview

The architectural diagram of our cascaded approach in classifying web pages is shown in Fig. 8.1. Each stage of the whole pipeline consists of an ML classifier using a different subset of features. The intended design is such that a majority of legitimate web pages will be classified quickly and correctly in the first stage. Similarly, most phish will make their way to the final stage, where the slowest web features are extracted. The early exiting nature of our design significantly reduces the computation spent in feature extraction, leading to a superior runtime performance over a diverse categories of web pages that represent in some sense a miniature of the web, which is characterized by a highly skewed distribution of web pages of two classes.

8.1.2 Feature Space

Features are the cornerstone of any ML-related technique, and before embarking on explaining the cascade structure learning algorithm, which involves selecting a subset of features for each

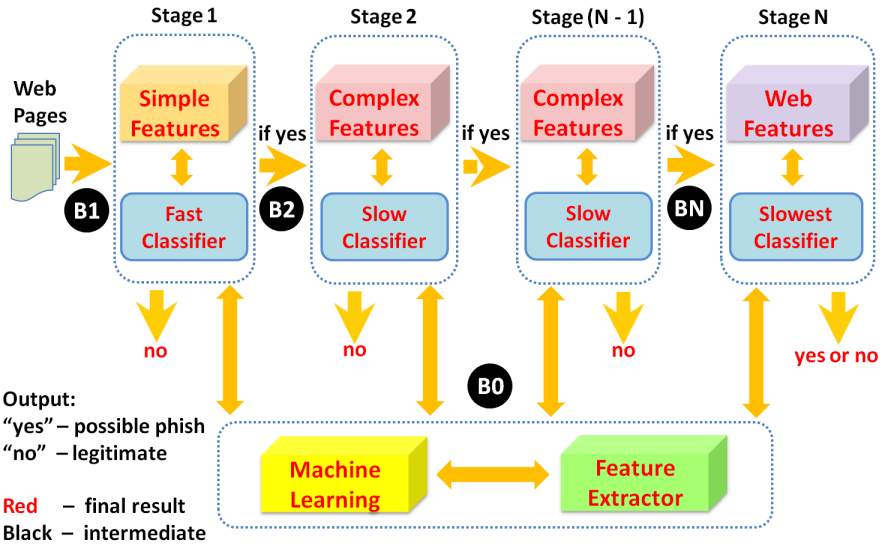

Figure 8.1: The system architecture of our approach for phish detection. B_1) the first-stage classifier examines the web page via fast features with high TP; B_2)—B_{N-1}) the web page passes the cascade as long as all stages emit a "phish" prediction; B_N) the last-stage classifier inspects the web page via web-based features. On a high level, each stage has a decent TP, leading to a reasonably high overall TP. The overall FP is low because each stage reduces the FP significantly. Particularly, the first stage guarantees the vast majority of genuine phish to be forwarded to the following stages, while rejecting a portion of legitimate web pages from entering the next stage.

stage of the cascade, we would like to introduce the whole feature set in this section. Specifically, we choose a feature set of moderate scale and decent performance (13 features in total) in this approach based on our previous result [86]. A comprehensive feature set utilizing URL, HTML DOM, web resources, topic models, visual appearance might boost the TP by some margin, as shown in [17]. However, the prohibitive computation associated with feature extraction renders user experience much less desirable in realtime scenarios.

We have organized our features into three types in a cost-ascending order. Generally, type 3 features run slowest and dominate the runtime in feature extraction. Specifically, our previous work [86] shows that the runtime of feature extraction is typically on the order of 10μs, $10^3\mu$s and $10^5\mu$s for URL, HTML and web features respectively.

URL-based Features

1. ***Embedded domain***. This feature examines the presence of dot separated domain/hostname patterns such as "www.ebay.com" in the path part of the web page URL. Phishers sometimes add their target's domain or hostname in the path segment to trick users into trusting their phishing sites.
2. ***IP address***. This feature checks if a page's domain name is an IP address.
3. ***Number of dots in URL***. This feature counts the number of dots in the URL. Phishing pages tend to use more dots in their URLs than the legitimate sites.
4. ***Suspicious URL***. This feature checks if a page's URL contains an "at" (@) or the domain name has a dash (-).
5. ***Number of sensitive words in URL***. In [36], Garera et al summarized a set of eight

sensitive words that frequently appear in phishing URLs, and we create this feature counting the number of the eight sensitive words that are found in a page URL.

6. ***Out-of-position top level domain (TLD)***. This feature checks if a TLD appears in an unusual position in the URL.

HTML-based Features

7. ***Login forms***. Phishing attacks are usually accomplished through HTML forms, and this feature checks if a page contains HTML forms for login purposes. To satisfy our definition of a login form, a web page is required to have all of the following: 1) an HTML form, 2) an input tag in the form, 3) keywords related to sensitive information like "password" or images within the scope of the HTML form. We defined 41 login-related keywords to narrow down our focus to forms that truly request user private information.
8. ***Bad login forms***. This feature examines if a page contains potentially harmful HTML forms by conducting one more check in addition to the three listed in feature 7 above, i.e., a non-https scheme in the URL in the action field or in the web page URL when the action field is empty.
9. ***Bad action fields***. This feature checks if the action field is empty or a simple file name, or points to a domain different from the web page domain.
10. ***Non-matching URLs***. This feature examines all the links in the HTML, and checks if the most frequent domain coincides with the page domain.
11. ***Domain keyword in page text***. This feature searches the domain keyword (such as "ebay" in "www.ebay.com") in the web page text content. Due to the common practice of synchronizing brand names with web page domains, the text content of a legitimate web page is much more likely to contain the domain keyword. Specifically, we examine the unigrams, bigrams and trigrams in the page content to cover glued words in domain keywords such as "bankofamerica".

Web-based Features

12. ***Age of domain***. This feature checks the age of the web page domain name via WHOIS lookups. Many phishing sites are hosted on recently registered domains, and as such have a relatively young age.
13. ***Page in top search results***. This feature was originally used in CANTINA [92]. Specifically, we extract the top K words from the page content ranked by the TF-IDF metric, and query those top terms plus the web page domain in Google. The feature indicates legitimacy if the page domain matches the domain name of any of the top N search results.

Table 8.1: Summary of the parameters and heuristics in learning the cascade structure with a low FP and a decent TP. Cascades that prefer a high TP and a reasonable FP can be learned with some minor adjustment to our learning algorithm.

Symbol	Interpretation	Symbol	Interpretation
N	#stages automatically learned	"URL" features	Used in stages $1 \dots N-1$
$TP_{overall}$	Overall TP for the cascade	"DOM" features	Used in stages $1 \dots N-1$
TP_1	A very high threshold for 1st stage	"Web" features	Used in the last stage
TP_i	High thresholds for stages $2 \dots N-1$	Forward selection	Used in 1st stage
pr	Ratio of phish in each training set	Backward selection	Used in stages $2 \dots N$

8.1.3 Cascade Structure Learning

One pronounced difference of our approach from previous cascaded learning techniques is the use of a heterogeneous set of features especially the web-based features, which have a rather ad-hoc runtime performance due to the unpredictable web and usually dominate all other overhead combined. By exploiting the distributional skewness of two types of web pages, our cascaded learning framework radically reduces the computation time in classifying web pages. The backbone of our proposed technique includes cascade learning, stage classifier learning and cascade evaluation, which are sketched out in Algorithms 7, 8, 9 respectively. The stopping criterion in line 10 of Algorithm 7 and the feature selection step in line 2 of Algorithm 8 control the preference over a high TP with reasonably low FP or a very low FP with a decent TP from the learned cascade.

Algorithm 7 TrainCascade

Require: training corpus T, validation corpus V, candidate feature set F, stage learning constraints H_S, cascade performance constraints H_C, percentage of phish in training set pr
Ensure: cascaded classifier C
1: cascaded classifier $C \leftarrow \phi$
2: stage classifier $C_{stage} \leftarrow \phi$
3: stage feature set $F_{stage} \leftarrow \phi$
4: $stage = 0$
5: **repeat**
6: $stage = stage + 1$
7: C_{stage}, $F_{stage} \leftarrow$ LearnStageClassifier(T, V, F, H_S, pr, $stage$)
8: $F \leftarrow F - F_{stage}$
9: append C_{stage} to C
10: **until** evaluate(C, V) $\geq H_C$ $\|$ allFeaturesUsed(F)
11: **return** C

Algorithm 8 LearnStageClassifier

Require: training corpus T, validation corpus V, candidate feature set F, stage learning constraints H_S, percentage of phish in training set pr, $stage$
Ensure: stage classifier C_{stage}, stage feature set F_{stage}
1: $F_c \leftarrow$ chooseViaConstraints(F, H_S)
2: $F_{stage} \leftarrow$ chooseBestVia10Fold-CV(F_c, T, V)
3: training set $TS \leftarrow$ bootstrap(T, pr)
4: $C_{stage} \leftarrow$ trainClassifier(TS, F_{stage})
5: **return** C_{stage}, F_{stage}

Our work in this chapter aims at providing a well-performing and practically usable technique, rather than offering a series of theoretically sound math equations. To optimize both the accuracy and efficiency, we designed a novel heuristic-guided algorithm that considers both the runtime and detection rate of the features in learning the cascade structure. Before delving into the details, we summarize the list of parameters and heuristics in our learning algorithm in Table 8.1. Specifically, we assign a type to each feature out of three possible values "URL", "DOM", "Web", with each indicating the rough amount of overhead needed for feature extraction. In learning the classifiers of the stages other than the last one, we only take "URL" and "DOM" features,

Algorithm 9 ApplyCascade

Require: cascaded classifier C, testing web page p
Ensure: classification result r

```
1: r = "phish"
2: stage = 1
3: MAX = #stages in C
4: while stage ≤ MAX do
5:     C_stage ← getStageClassifier(C, stage)
6:     F_stage ← getStageFeatureSet(C, stage)
7:     f_stage ← extractFeatureValues(p, F_stage)
8:     pred ← apply C_stage to f_stage
9:     if pred ≡ "legitimate" then
10:        r = pred
11:        break
12:    stage = stage + 1
13: return r
```

while for the final stage (i.e., when all other features have been used already), we use a candidate feature set composed of "Web" features entirely. Moreover, we set a very high threshold on TP (0.95 in our experiment) for the first stage and a relatively high threshold on TP (0.7 in the current setting) for the 2nd to $(N-1)$-th stage. As a matter of fact, the TP of each stage is often high enough, often over 0.85 for example, and the main purpose of the thresholds for the 2nd to $(N-1)$-th stage is to filter out spurious stage classifiers especially those built by only one feature that is not selected in previous stages.

For cascaded models that prefer a very low overall FP with a decent TP, each stage of the cascade structure needs to have a reasonably high TP and a relatively low FP. To obtain a lightweight classifier with high coverage in the front of our pipeline, we conduct forward feature selection (Line 2 in Algorithm 8) in the first-stage classifier until the stage TP requirement is satisfied. For classifiers in later stages, we apply backward feature selection (Line 2 in Algorithm 8) until dropping a feature hurts the corresponding stage TP. We keep adding stages until the pre-specified overall detection performance is met or all features have been used. Using the trained cascaded detector to classify web pages simply involves walking a web page along the stages, extracting necessary features, emitting a final "legitimate" prediction if any stage classifier says so and a "phish" one otherwise. To learn a cascade model with a high overall TP and a relatively low FP, alternatively, we can apply backward feature selection in building the classifiers in every stage, because backward selection tends to choose a larger feature set and thus leads to a cascade with a small number of stages.

In some previous work like [20], researchers specified a constraint on the overall performance and constructed each stage to meet a feasible point in the ROC curve along the way to a final acceptable cascade. In our approach here, we can adjust on which variables to set thresholds as well as the values for those thresholds according to the specific applications. For anti-phishing, we set thresholds only on the TP, and partially rely on the diminishing effect of the multiplicative stages to control FP. This is actually feasible in practice because the FP of the first stage in our cascade is usually below 40% and that of the other stages is typically well below 10%. Taking these values and assuming a maximum of 3 stages, our cascade has an overall FP of approximately $0.4 \times 0.1 \times 0.1 = 0.4\%$, which is among the lowest in anti-phishing literature. The FP will be even

lower if there are more stages. Moreover, our previous work [86] showed that some web-based features such as "page in top search results" are quite effective in correcting false positives, and by using those features in the last stage, we further reduce the FP of our cascaded detector. The overall TP does not go through the same level of decay, however, because each stage classifier has a reasonably high TP, and our compact yet effective feature space with backward feature selection bounds the number of stages to a small number.

Similar to decision trees, subsequent classifiers are trained with instances passing through all previous stages, leading to fewer legitimate training examples as stages are grown. We adopt the bootstrapping strategy, as shown in Algorithm 8, to keep the ratio between instances of two classes in the training set of each stage to be at a certain level. Moreover, we have a wide range of options in choosing a specific machine learning algorithm to train each stage model, and we decide to use Bayesian Networks (BN) in accordance with our previous finding [86] that BN is among the best learning algorithms for the anti-phishing task. The structure of BN is automatically learned by the K2 algorithm.

8.2 Experiment Settings

8.2.1 Web Page Corpus

To thoroughly evaluate the performance of our approach over time, we used two phishing corpus for our evaluation, one collected a few years ago which we name "classic corpus", and the other collected from July to September in 2012 called "new phish corpus". In this section, we give a brief summary of our classic corpus and new corpus in Table 8.2.

Phishing sites are usually ephemeral, and most pages will not last more than a few days typically because they are taken down by attackers to avoid tracking. Therefore, we downloaded phish when they were still alive. In particular, we used the phish feed of Phishtank, a large community-based anti-phishing service where people can view and confirm phishing sites. The growing use of toolkits [26] to create phish produces a massive volume of phishing web pages that are very similar or even identical to each other, and we adopted the hash-based filtering algorithm [86] to retain unique phish only in our evaluation.

Legitimate patterns do not exhibit drastic changes over time, and therefore, we chose to use the legitimate collection in the classic corpus, which was crawled at roughly the same time as the phishing corpus in the classic corpus. The legitimate web pages in the classic corpus came from 5 sources, the details of which are given in Table 8.2. Specifically, Alexa.com maintains a top 100 website list for a variety of languages, and we crawled the homepages of the top 100 English sites to a limited depth, collecting 1022 pages in this category. 3Sharp [15] released a public report on anti-phishing toolbar evaluation in 2006, and we downloaded 101 good English pages out of the 500 provided in the report that still existed at the time of downloading. Moreover, we went to Yahoo directory's bank category [4], crawling the bank homepages for a varying number of steps within the same domains and collecting 983 bank pages. Likewise, we conducted crawling on other categories [7, 8, 9, 10, 11, 12] of Yahoo directory including US bank, credit union, real estates and financial services, etc., and gathered 371 web pages in the "Yahoo Dir" category. To test the robustness of our methods, we manually chose 81 login pages of popular phishing target sites, such as eBay, etc. in the "Famous" category.

Table 8.2: Summary of the classic corpus and new phish corpus in our experiment, which were collected in mid 2008 and from July to September in 2012 respectively. Since legitimate patterns do not exhibit drastic changes over the time, we did not crawl legitimate pages in the new corpus. Each corpus contain unique phishing attacks only. Data in each cell were partitioned into three (70%/20%/10%) parts for training, validation and testing respectively in a 10-fold CV setting.

	Categories					
Classic Corpus	Phish	Alexa	3Sharp	Bank	Yahoo Dir	Famous
Size	2,219	1,022	101	983	371	81
New Phish Corpus	Phish					
Size	7,152					

8.2.2 Evaluation Methodology

Phishing attacks are constantly evolving, and to inspect the power of our approach and its capability to detect novel phish, we conduct two rounds of evaluations, one on the classic corpus that we crawled in our previous work [85][86], and the other on the new phishing corpus as introduced in the previous section. To demonstrate the efficacy of our technique, we compare it with a single-stage baseline BN, which preserves the essential ingredients of CANTINA+ [86] though with fewer features. Taking the standard practice in ML, we randomly split our classic corpus into a 70% portion for training, 20% for validation, and the remaining 10% for testing. The new phishing corpus is used solely for testing.

A key parameter in our cascade learner is the ratio of phishing web pages in the bootstrapping process to prepare training examples in learning each stage classifier. We tune this parameter on the 20% validation data set, and employ the optimal value to test our cascade detector and baseline model. To reduce random variation and avoid lucky train/test splits, we average the results via 10-fold cross validation (CV).

In our experiment, we adopted the True Positive Rate (TP) and False Positive Rate (FP), which are the standard metrics in evaluating many binary classification tasks such as anti-phishing. We also used the F1 measure, which integrates both TP and FP with equal weights into one summary statistic.

8.3 Experimental Result

8.3.1 Parameter Tuning

Due to the early rejecting property of the cascaded learning framework, we adopted a bootstrapping algorithm that randomly samples negative instances to keep the ratio of phishing pages at a certain level in each training set. This parameter is indicated by *pr* in Table 8.1. We varied its value and reported the result using the training and validation partitions of the classic corpus in Table 8.3. This parameter is critical mainly for two reasons. First, it governs the composition of the training data in learning a stage model, which determines the performance of our ML classifier to a great extent. Second, adjusting the values of *pr* essentially amounts to assigning variable penalties to the two types of errors in a cost-sensitive learning paradigm, a procedure of great importance to tasks where different types of errors do not bear the same level of consequence like anti-phishing. Other parameters such as the thresholds on the TPs shown in Table 8.1 can be tuned in a similar fashion. However, since there is a rough range of TP that is deemed as good

Table 8.3: Parameter tuning result over the validation set in the classic corpus. Bootstrapping is employed to maintain the ratio of phishing web pages in the training data. Our cascaded detector has a low FP of 0.4% to 0.7%, about 1/10 to 1/8 of the FPs by the single-stage baseline model. The TPs of our cascaded detector are also reasonably high.

	Cascade Detector			Baseline		
Phish Ratio in Training Data	TP (%)	FP (%)	F1	TP (%)	FP (%)	F1
0.3	78.60	0.43	0.88	93.72	3.41	0.95
0.4	78.26	0.47	0.88	94.49	4.25	0.95
0.5	78.94	0.53	0.88	94.72	4.75	0.95
0.6	83.34	0.65	0.91	94.72	4.75	0.95
0.7	81.49	0.69	0.89	94.72	4.75	0.95

Table 8.4: Testing result over the holdout testing set in the classic corpus and new phish corpus. The ratio of phish in the training data is taken to be 0.6 based on the tuning result in Table 8.3. Both methods yield a performance of the same level as on the validation set given in Table 8.3. After evolving for a few years, current phishing trends have more target web sites, and the statistics here show that our approach is able to generalize to novel phishing patterns. Since there are no legitimate pages in the new phish corpus, we use N/A as a placeholder in some cells.

	Cascade Detector			Baseline		
	TP (%)	FP (%)	F1	TP (%)	FP (%)	F1
Classic Corpus	83.32	0.81	0.90	94.26	4.23	0.95
New Phish Corpus	81.19	N/A	N/A	96.41	N/A	N/A

enough according to the anti-phishing techniques in the literature, we empirically set the thresholds for those TPs in Table 8.1. Specifically, we chose $TP_{overall} = 0.9$, $TP_1 = 0.95$ and $TP_i = 0.7$ ($i \in \{2, \ldots, N-1\}$) where N is the number of stages in our cascade learned by algorithm 7.

The result in Table 8.3 delivers the following messages. First, our cascaded detector has a low FP of 0.4% to 0.7%, about 1/10 to 1/8 of the FPs by the baseline approach. Second, the TP of our approach is reasonably high, reaching 83.34% with a high F1 of 0.91 when using 60% phish in the training data. Third, both the TP and FP ascend most of the times as the percentage of phish rises in the training data. Fourth, the TP of the baseline is much higher than our approach, however, its FP is also way higher, rendering the baseline an insufficient anti-phishing approach when used alone.

8.3.2 Testing on the Holdout Data Set

To provide an objective and comprehensive evaluation, we test our cascaded detector on the holdout testing set from the classic corpus and the new phish corpus, and report the result in Table 8.4. Since Table 8.3 suggests using 60% phish in the training data yields the best result, we equip our cascade learner with this value from now on.

On the holdout testing set from the classic corpus, both methods have a TP and FP comparable with the counterparts from the parameter tuning result on the validation set. Particularly, our cascaded detector presents a low FP of 0.81% with a decent TP of 83.32%, landing with a high F1 of 0.9. Again, our approach outperforms the baseline significantly on FP (0.81% vs 4.23%).

Our classic corpus contain phishing attacks collected a few years ago, and to examine our proposed technique on fresh phishing patterns, we conducted experiment on the new phish corpus with $7,152$ unique phishing web pages. Table 8.4 shows that our cascaded detector still achieves a fairly high TP of 81.19%, manifesting the power of our approach to generalize to novel phishing attacks. Similarly, the single-stage baseline model also exhibits a good detection rate on the new phish corpus, which confirms our hypothesis that a feature set of moderate scale often suffices and thus lends legitimacy to the way we chose our feature set. Although there is an about 2% decrease on TP from the result on the classic corpus, the key advantage that our cascading approach offers is a more explicit way for developers to make tradeoffs between TP, FP, and runtime.

Moreover, since the testing set was chosen randomly in a 10-fold CV setting and remains intact during the parameter tuning process, the superior result in this section demonstrates the genuine efficacy of our cascaded approach in detecting phish.

We notice that the FP of the single-stage baseline model is much higher than that of CANTINA [92], the work that proposed the "page in top search results" feature. One explanation is that our baseline here uses more features and has gone through more intensive model training, while CANTINA is essentially a very conservative linear model that has no formal training process.

8.3.3 Runtime Evaluation

Latency is an important measure to any network-based service, which determines to a great extent the quality of experience of the end users. Previous studies [18] show that a 500-millisecond increase in service latency reduces the traffic by 20% for Google.com, and a 100-millisecond increase in service latency causes a 1% reduction in sales for Amazon.com. For domains such as anti-phishing, virus detection and so on, a low latency is also of significant importance mainly in that a slow classification would put users under great risk. The maximum acceptable service latency is closely related to end user experience, and in reality, a successful response from a web application that takes longer than the maximum acceptable service latency will often be regarded as unsuccessful by users, presumably because they have abandoned the request prematurely. Users' satisfaction with service latency is application dependent, and a rule of thumb is that the maximum acceptable service latency is perhaps $10 - 20$ times the 50th percentile service latency requirement [18]. Given the 200 milliseconds for the typical 50th percentile service latency [18], the range of the maximum acceptable service latency is approximately $2 - 4$ seconds.

One major advantage of our proposed approach over existing techniques is its average runtime performance. To prove this, we collect the runtime statistics from our experiment in Section 8.3.2 and report the result in Table 8.5. To further expedite our technique, we cache web fetching results for the two "web" features, i.e., "age of domain" and "page in top search results". This simple caching strategy benefits the "age of domain" feature tremendously, since users typically visit multiple web pages from a single domain.

The prominent finding from the result is that our cascaded detector achieves a 55.7% reduction on average runtime in classifying a web page compared with the traditional one-stage approaches (672.63ms vs 1519.56ms) thanks to its early exiting strategy along the chain of classifiers in the cascade. Out of the 260 legitimate pages in the holdout testing set from the classic corpus in each fold of the CV, a staggering 95.73% are classified without touching the prohibitive "web" features, leading to a far less average time of 98.56ms on a legitimate page by our approach than the $1,536.36$ms by the single-stage baseline. This brings tremendous benefits to our cascaded approach in a live detection scenario, which tends to have a much shorter latency than existing ML-based single-stage filters, because the web is characterized by a highly skewed distribution of

Table 8.5: Average runtime (milliseconds) via 10-fold cross validation (CV) over the holdout testing set in the classic corpus and new phish corpus. The holdout testing set in the classic corpus has 223 phish and 260 legitimate cases, while the new phish corpus has 7,152 phish and 0 legitimate pages. Our cascaded detector achieves a much superior runtime than the baseline model using a monolithic classifier in one stage. Among the 260 legitimate pages in each fold of the 10-fold CV, an average of 95.73% are classified without having to extract the expensive "web" features, leading to a low runtime of 672.63ms per web page compared with the 1,519.56ms of the single-stage baseline. Particularly, the average overall runtime here is biased and provides a pessimistic estimation of the true value because our holdout testing set has roughly 50% phish and 50% legitimate pages, whereas in real life the percentage of the latter will be much higher.

	Cascade Detector		Baseline	
	Classic Corpus	New Corpus	Classic Corpus	New Corpus
On phishing pages (ms)	1,341.95	1,118.77	1,499.97	1,471.59
On legitimate pages (ms)	98.56	N/A	1,536.36	N/A
Overall Runtime (ms)	672.63	1,118.77	1,519.56	1,471.59
#Early exit on phish	25.9	932.8	0	0
#Early exit on legit pages	248.9	N/A	0	N/A

phishing and legitimate web pages.

Moreover, the average overall runtime in Table 8.5 is biased because our holdout testing set has roughly 50% phish and 50% legitimate pages, whereas in real life the percentage of the latter will be much higher. However, by giving a pessimistic estimation of the true average runtime value, our approach provides an upper bound on the runtime performance over the whole web and is actually more reasonable and beneficial for further evaluations against other techniques.

To provide more details in addition to the average runtime performance, we show in Fig 8.2 a box plot of the runtime of our cascaded detector and the single-stage baseline on each instance of the holdout testing set with 223 phishing and 260 legitimate web pages. The plot provides five summary statistics in one graph, i.e., the minimum, lower quartile, median, upper quartile, and the maximum, and depicts the rough distribution of the runtime on the testing corpus. As shown in the graph, the phish detector based on our cascaded learning technique runs much faster than the traditional single-stage model.

Although the average overall runtime in Table 8.5 is higher than the 200ms and 400ms corresponding to the 50th and 95th percentile latency of a typical web service [18], we found that a significant percent of the computation of our cascaded approach was often spent on building the HTML DOM tree from the HTML string. This was in turn caused by the inefficient off-the-shelf HTML parser we used in our technique for feature extraction, and is more of an implementation issue than a design one. We believe that given a state-of-the-art HTML parser such as the internal one used by Google's Chrome browser or our approach is integrated into the Chrome browser natively, our cascaded detector will achieve significant runtime speedup.

Since user satisfaction is directly impacted by the service latency, we further investigate the runtime performance of our approach in the context of user satisfaction. Specifically, we propose a technique that models user satisfaction as a function of service latency given by

$$S = \exp(-K \cdot t)$$

where S and t denote the user satisfaction and service latency respectively, and K is an coefficient

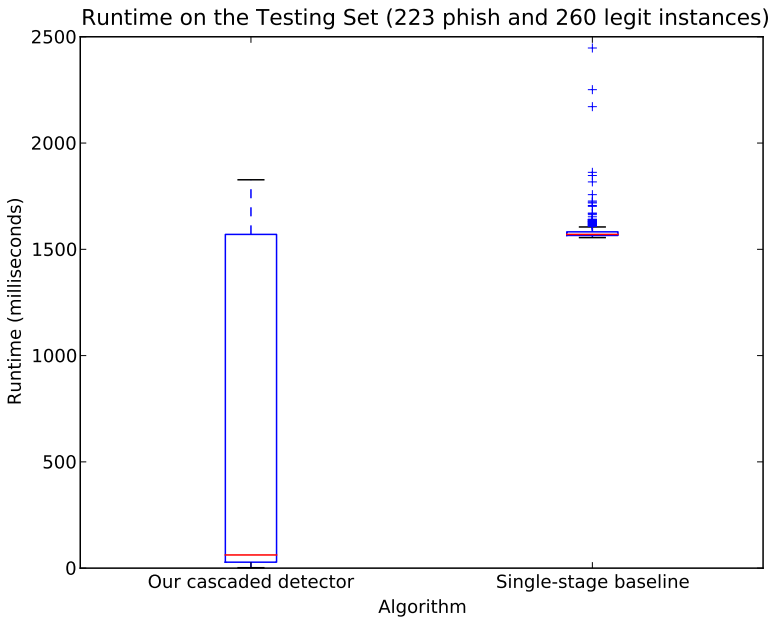

Figure 8.2: A box plot showing the five-number summaries (minimum, lower quartile, median, upper quartile, and maximum) of the runtime of our cascaded detector and the single-stage baseline on the holdout testing set. Our cascaded learning framework is significantly better than the single-stage baseline in terms of the runtime performance. The box for the baseline indicates that the web-based features dominate all the cost associated with classifying a web page.

governing the shape of the curve. This is not an empirically validated approach in the literature, but is very reasonable given the relationship between user satisfaction and service latency. Assuming a low user satisfaction of 0.1 at a latency of 2 seconds (a quantity 10 times the 50th percentile service latency as explained previously in this section), we arrive at a value of 1.1513 for K. With this assignment, the user satisfaction is 0.01 with a latency of 4 seconds (20 times the 50th percentile service latency), which is very reasonable based on the previous finding on the relationship between the maximum acceptable latency and the 50th percentile service latency.

We draw the user satisfaction curve as a function of service latency and plot a few key points at the same time, as shown in Fig 8.3. Specifically, our approach achieves an average runtime of 672.63 milliseconds per web page according to Table 8.5, which corresponds to a user satisfaction value of 0.461. The counterpart average runtime and user satisfaction for the single-stage baseline model, however, are $1,519.56$ milliseconds and 0.174. That means an almost $3X$ increase in user satisfaction with our proposed approach.

8.3.4 Cascade Structure

Having obtained superior performance on accuracy and efficiency, we inspect the structure of our cascaded classifier that is automatically learned by our learning algorithm 7 in section 8.1.3.

Among the 50 cascaded classifiers we learned with Algorithm 7 (10 fold each with 5 values for pr), 22 end up with 3 stages in total, another 20 have 4 stages, with the remaining 8 composed of 5 stages. This conforms to our design principle that a small cascade helps preserve TP, which

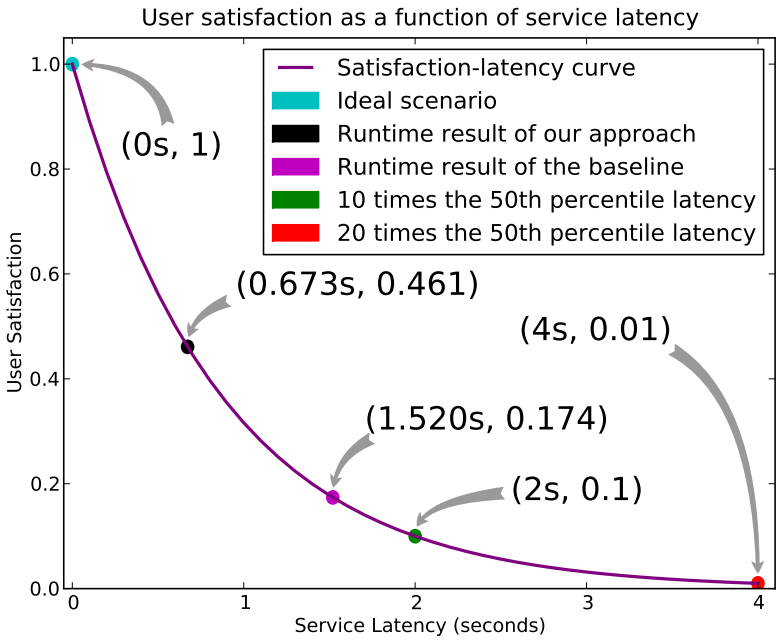

Figure 8.3: With an average runtime of 672.63 milliseconds per web page on the classic corpus, our proposed approach has a user satisfaction value of 0.461, almost 3 times of the 0.174 achieved by the baseline model that builds all features into a single-stage classifier.

typically deteriorates somewhat after each stage. Moreover, some cascaded classifiers ended up with a different number of stages and feature usage in the stages mainly due to the random split of training and testing sets, and the bootstrapping strategy in sampling negative instances in learning the cascade structure in each fold.

All 50 cascades choose a single feature "login forms" after forward selection in the first-stage classifier. This makes perfect sense since this feature alone has a TP over 96% and is also cheaper to extract than the web-based features. Intuitively, all phishing attacks have login forms to hold user input such as password. Moreover, the second-stage classifier uses most of the remaining non-web features, which forwards the majority of the phish passing through the first stage to the next stage while rejecting a portion of the legitimate cases making their way to this stage. The final stage further reduces the FP via the slow web-based features. The intuition of this is that search engines are more likely to index legitimate web sites, while phishing sites have much less chance of being crawled.

In Table 8.6, we show one cascaded detector that is automatically learned in one of the 10-fold CV, as well as its performance on the corresponding validation data set. As indicated in the table, the first stage utilizes the "login forms" feature only, which runs fast and has a high TP of 96.84%. As the cascade is being built, both the TP and FP decline somewhat. The learning process terminates with a final cascade composed of 3 stages. This structure looks straightforward, though it was algorithmically created. However, we only used 13 features in our current framework, and as the dimension of the feature space grows, the benefit of automatically learning a cascade structure via our approach will be more prominent.

Table 8.6: Structure of a cascaded detector in one fold of the 10-fold CV ($pr = 0.6$). The upper part of the table lists the set of features automatically selected in each stage of the final cascade. The lower part presents the overall performance of the cascade as each stage is added. Both TP and FP decline as the cascade grows, reaching a TP/FP/F1 of 85.1%/0.39%/0.917 when the learning process terminates with a total of 3 stages.

Stage ID	Final Feature Set		
1	Feature 7: login forms		
2	All except for feature 7 and the web features		
3	Feature 12: age of domains		
	Feature 13: page in top search results		
	Overall Cascade Performance		
#Stages	TP	FP	F1
1	96.84%	34.31%	0.819
2	88.71%	3.53%	0.920
3	85.10%	0.39%	0.917

8.3.5 Error Analysis

We examined the errors our cascaded detector made in classifying web pages, and present our error analysis in this section. Congruous with our previous finding in [86], most of the "HTML" and "Web" features are more powerful (and slower) in catching phish and filtering legitimate pages than URL-based features, and once misclassification occurs, it is usually the case that one or more of the "HTML" or "Web" features behave abnormally.

On the holdout testing set from the classic corpus, most of the false positives are a mixed effect of legitimate pages accidentally manifesting suspicious login forms or action fields, and web features failing to retrieve the intended feature values. There is not much we can do to influence the way people design web sites, but we can definitely resort to better repositories and finer algorithms to fetch web-based features. For example, quite a few domains in our corpus do not have entries in the WHOIS service we are using currently, leading to unnecessary noise in our feature values. As a remedy, we can turn to other premium services to get a complete list of WHOIS records for our URLs. In addition, the "page in top search results" feature sometimes cannot return the intended domain among top search result entries due to two reasons. First, some terms with high TF-IDF scores may not be relevant for searching purposes; second, due to company affiliations, two closely related domains are sometimes literally different such as "blogger.com" and "blogspot.com", which renders straightforward string matching inadequate. To augment this feature, we can use other web-based features like "page in top results when searching copyright company name and domain", as defined in [86], which in turn will cause some extra overhead.

The false negatives of our approach are mainly caused by the classifiers from the second to the last stage, which split the most effective features among them, thus inadvertently decreasing the detection power of each classifier. This is not difficult to fix, and we can simply introduce a few more highly discriminatory features, such as those exploiting visual elements on a web page.

An interesting observation on the new phish corpus is that 71 out of the 7,152 phishing web pages were hosted on "blogspot.com", with most of them incorrectly classified. Phishing hosted on compromised legal domains has always an arduous problem. However, we still have potential

countermeasures for it, given the fact that social sites such as "blogspot.com" are more likely to be exploited for phishing attacks. Specifically, we can collect a list of such most compromised domains, and conduct extra checking for URLs hosted on those domains.

8.3.6 Potential Adversarial Attacks

There are a certain phishing variants that our current cascaded detector, as well as almost all existing anti-phishing solutions, cannot deal with properly, which in turn makes our work and further research effort worthwhile.

One difficult scenario is that attackers compromise legitimate domains and host phishing attacks on those servers, such as blogspot.com which is seen often in our new phish corpus. In addition to the extra checking mentioned in Section 8.3.5, we can ameliorate the impact of this exploit by adding more features to our cascaded learning framework at the expense of elevated overhead. Our approach makes it easier to incorporate new features into the suitable stages of the cascaded model automatically, and the only thing we need to do is to provide those new features as normal input to our learning algorithm.

Attackers sometimes build phishing web pages purely made up of images, leaving our algorithm no text for analysis. Although text-based technique is infeasible here, we can integrate features that exploit visual elements into our cascaded framework, taking advantage of the high extensibility of our approach again.

8.4 A Phish Detector Prototype based on Our Cascaded Learning Framework

To better demonstrate the usage and performance of our proposed technique, we built an online phish detector prototype based on our proposed technique. Essentially, our online cascaded phish detector is composed of a client-side component and a server-side component. The client side is implemented as a Chrome extension, which injects content script to web pages and extracts the corresponding HTML DOMs. The server side is implemented as a Java web application that runs in the Java Servlet Environment provided by the Google App Engine (GAE).

8.4.1 System Architecture

Fig 8.4 shows the system diagram of the prototype online phish detector. Basically, there are four major steps in classifying a web page, among which the first step extracts the HTML DOM via a chrome extension and the third step handles the classification task in the backend server-side code. The client-side Chrome extension can be found at `http://bit.ly/VlUSRZ`, and installing it on the Chrome browser simply takes a mouse click.

8.4.2 Experiment

To evaluate our prototype detector against popular phishing filters, we need a fresh phishing corpus that is not incorporated in major blacklists (such as PhishTank) which are typically used by most phishing filters. Due to the labor-intensive nature of this experiment, we chose to focus on a small set of URLs. To that end, we hired a student in our school, who built 21 password-protected phishing attacks so that those industry blacklists cannot crawl them. One phishing attack targeting Amazon.com is shown in Fig 8.5, which has high visual similarity to the

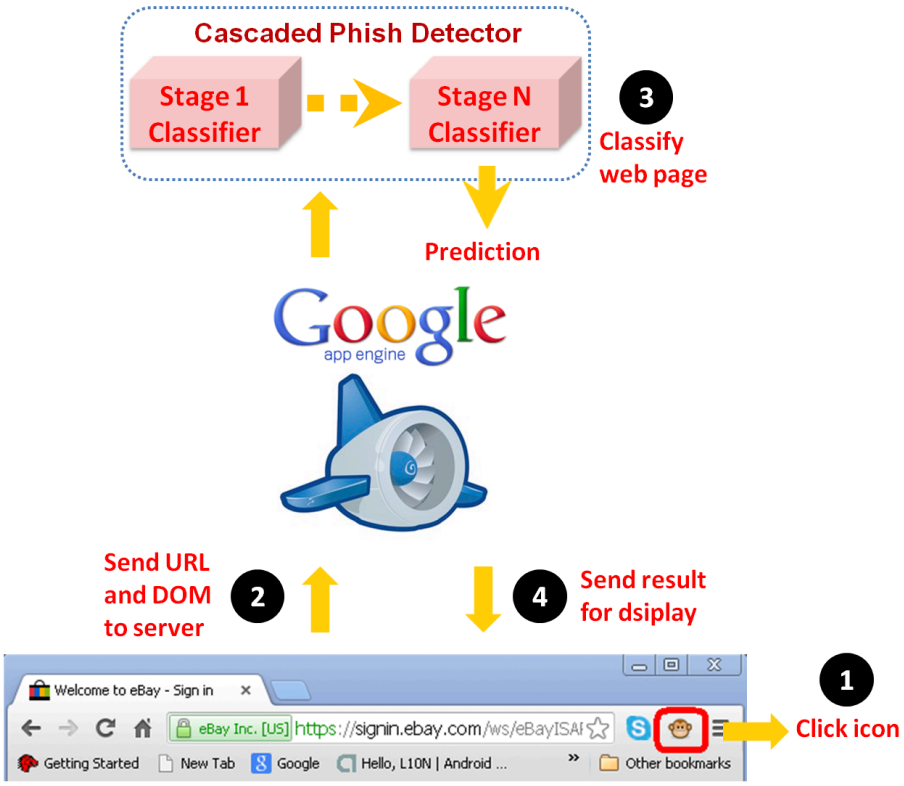

Figure 8.4: The system architecture of an online phish detector prototype based on our cascaded learning framework. The key steps in classifying a web page are as follows. Step 1) The user opens a web page in the Chrome browser and clicks the icon of our extension; Step 2) The HTML DOM is extracted by the dynamically injected content script in our chrome extension, and sent together with the URL to our server-side code hosted on GAE; Step 3) Our cascaded phish detector classifies the web page; Step 4) The classification result as well as some diagnostic statistics are sent back and displayed in the browser on the client side.

genuine counterpart. In accessing the performance of our prototype on legitimate web pages, we randomly selected 20 pages from Yahoo directory's bank category, as introduced in section 8.2.1, and another 20 pages from the popular phishing target sites. For comparison, we used the Google Safe Browsing filter embedded in the Chrome browser.

The goal of this experiment is to demonstrate the usability and effectiveness of our cascaded detector in identifying phish against well-known filters. We do not offer a comparison on runtime with those filters due to the different underlying hardware infrastructure used for the implementations. The experiment result shows that the prototype detector using our cascaded learning technique outperforms the Google Safe Browsing filter in catching novel phish, while maintains the FP at a comparable level. Specifically, out of the 21 phish we built, our prototype detector successfully detected 15, leading to a TP of 71.43%, while the Google Safe Browsing blacklist detected 0, with a TP of 0%. The false negatives for our prototype were mainly caused by the high-performing features such as "bad login forms", "bad action fields", "page in top search results", which failed to retrieve intended values for those cases. Moreover, the experimental result indicates a 0% FP for both our prototype and the Google Safe Browsing filter embedded in the

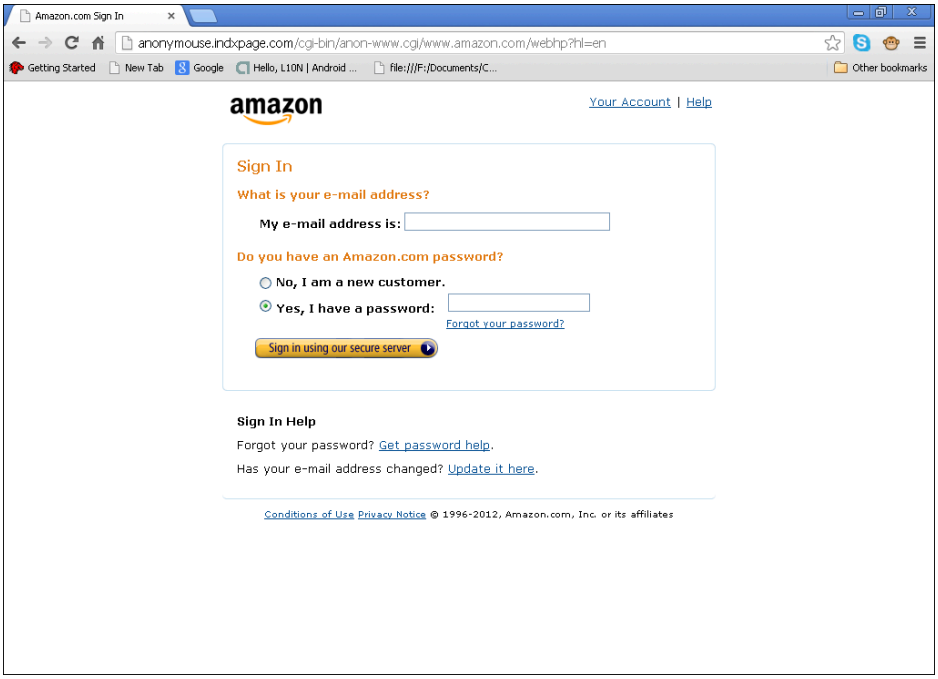

Figure 8.5: A phishing attack targeting Amazon.com that we built manually. The phishing web page has high visual similarity with the genuine Amazon sign-in page.

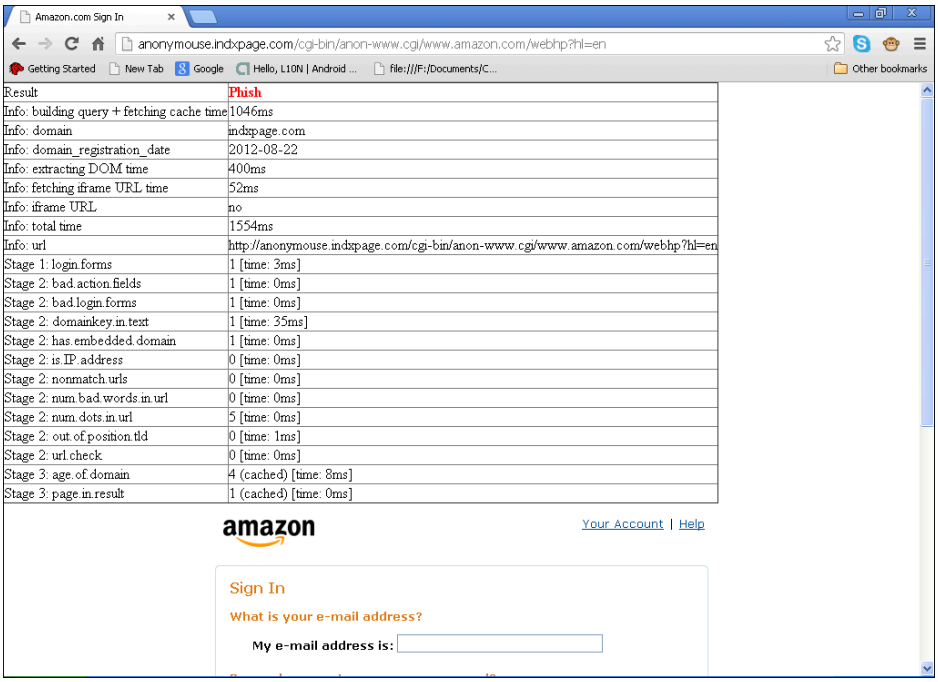

Figure 8.6: The result of applying our phish detector to classify a phishing attack targeting Amazon.com. The detection result is overlaid upon the phishing web page. Some diagnosis information such as the feature values and some runtime statistics are shown along with the classification result.

Chrome browser.

The usage of our prototype is self-explanatory. After the Chrome extension is installed, we simply need to open a web page in the Chrome browser and then click the extension icon. The detection result will then be overlaid upon the original web page. Fig 8.6 gives the screenshot of an example phish that targets Amazon.com and the detection result using our prototype system.

8.4.3 Further Improvement

Although the runtime of our prototype detector is usually on the order of 1 to 2 seconds for most phishing attacks and some legitimate web pages in our experiment, a level higher than the 200 milliseconds for the typical 50th percentile service latency, we observed that a significant percent of the computation was often spent on extracting the HTML DOM from the web page content string on the server side. Currently, we use JTidy as the DOM parser on the server side, which is not very efficient and presumably causes unnecessary overhead.

8.5 Discussion

8.5.1 Tradeoff between TP, FP and Runtime

For any domain involving classification tasks like anti-phishing, it is really hard to achieve superior TP, FP and runtime simultaneously. For real-world applications, responsiveness is usually a critical dimension to the usefulness of a technique. In terms of TP and FP, however, it depends on the specific characteristics and requirements of the domain. For instance, FP is given more emphasis in fields where concerns over liability issues exist, such as anti-phishing, virus detection, spam email filtering, and so on. For other areas where this is not the case, practitioners can balance the tradeoff according to their needs, such as cussing language detection.

For anti-phishing, a fast response time can enhance user experience in live scenarios, while a low FP renders the corresponding solution usable in practice. The TP should be as high as possible too, but there are no strict constraints on it as there are for FP. After all, the TP can always be improved with additional layers of checking, such as the URL blacklists embedded in major browsers that sit in the front of the whole anti-phishing pipeline. Moreover, humans' awareness of phishing attacks can be increased with some training.

Moreover, runtime is actually just one example of the cost, and we could generalize our ideas for other kinds of costs too, e.g., financial cost in terms of US dollars and so on.

8.5.2 Scalability

Being able to build a heterogeneous set of features into separate ML classifiers in a single cascade, our proposed approach is scalable with respect to more features of different types. This is a critical property of our approach, since phishing attacks are constantly evolving, and novel features may be needed in order to cope with brand new phishing patterns in the future. When new features come around, we simply need to assess their general runtime performance analytically and assign a type to each of them, such as "HTML", "Web", etc. Subsequently, they will be treated as normal input to our cascaded learning algorithm as other features are, and will be built automatically into the classifiers in the most suitable stages. Moreover, we can further speedup our technique by utilizing the cloud environment to distribute the computation of feature extraction into multiple nodes, as some existing work did [75].

8.5.3 Deployment of Our Cascaded Approach for Phish Detection

There are multiple choices in deploying our technique for phish detection in live scenarios, and we will briefly discuss the benefits and weaknesses of each in this section.

First, we can deploy our solution in a client-server model such as our phish detector prototype in section 8.4. One merit of this paradigm is its great convenience due to the multiple public cloud computing platforms such as Amazon Web Service (AWS) and Google App Engine (GAE). Another advantage is that the frontend extension or plug-in on the client machine handles part of the computation and thus reduces the work load on the server-side backend code, which is usually the bottleneck as the number of concurrent clients scales to a certain volume. One potential problem of this strategy concerns the security issues depending on the sophistication of the web servers. Web criminals may compromise the cloud computing services and figure out our anti-phishing algorithm via reverse engineering techniques.

Second, another deployment choice is to put our approach as a black box on the servers of the Internet Service Providers (ISP). One benefit of this strategy is its broad coverage in that it examines all the traffic passing the ISP machines and thus protects a larger scope of audience. However, the detection model in this deployment needs to have a very high performance because its classification directly impacts all the web users associated with the corresponding ISPs. Moreover, the ISP servers in this scenario have a much higher work load than the web servers in the client-server model and may cause interruptions and significant latencies.

8.5.4 Cost and Performance of Features in Different Domains

In the context of phish detection, our approach learns a cascade structure by optimizing each stage separately, and some of the high-performing features turn out to be fast enough to enable a local optimization. For other domains where the best features are the most expensive ones or using a really costly feature first will boost the overall performance on TP/FP, some constraints in our learning algorithm need to be relaxed to build an effective cascade model. However, the model learned this way will sacrifice the runtime performance to a certain extent in exchange of some improvement on the classification result. Whether this is a preferred solution will depend on the priority of speed and classification performance in specific domains. A better alternative is to design more features such that the high-performing features and the most computationally prohibitive ones do not coincide, such as in the field of phish detection.

8.5.5 Class Distributions in Various Fields

Our cascaded learning framework achieves a good balance between the classification and runtime performance by exploiting the skewed class distributions of a certain domains such as anti-phishing. For other areas where the majority/minority class is not as well-defined, our approach is still able to achieve some speedup on the runtime due to its multiple-staged nature and early exiting strategy, as evidenced in the anti-phishing domain where a number of phishing attacks were classified without inspecting the expensive web features (Table 8.5). However, the reduction on the average runtime over the traditional single-stage models in those fields will not be as pronounced as in the domains where the class distributions are highly skewed.

Moreover, our technique can be extended to multi-class classification. One way to realize this is to select the target class that we want to optimize and treat the instances of the remaining classes as negative examples in a standard binary classification scheme. Alternatively, metrics other than TP and FP can be employed to evaluate the model, and as long as not all classes

are treated equally, we could choose to optimize the most important class and build the cascade accordingly.

8.5.6 Adapting Our Cascaded Approach to Other Domains

Essentially, any application that emphasizes a fast responsive time and low FP (or high TP) will benefit from our cascaded learning framework. We simply need to list a set of features that will be built into the cascade, as well as some constraints particular to the specific applications, and we will obtain a cascaded model with ease.

Spam email filtering is one such area, which also relies on ML techniques to classify emails in our mailboxes. Similar to the anti-phishing arena, features typically used in this domain can also be categorized into types like "HTML" and "Web", such as "age of linked-to domain names", "non-matching URLs" (examining the inconsistency between the HREF field of a link and the anchor text), "number of links", "page in top search results" and so on. In [34], Fette et al. designed PILFER, a ML approach to detect phishing emails with 10 features including the 4 listed above. They did not report the runtime statistics in [34], however, PILFER can be accelerated by a factorization of the monolithic classifier into multiple ones with the slow web features (age of linked-to domain names, etc.) extracted in later stages. This process will be taken care of automatically by our cascade learning algorithm in Table 7, with the output being a cascade of classifiers using the same 10 features.

Chapter 9

Conclusions and Future Work

9.1 Summary of Main Results and Contributions

The primary contribution of this thesis is to propose a feature-type-aware cascaded learning framework for the a variety of domains with skewed class distribution and features with various classification and runtime performance in an effort to achieve a good balance between the three desiderata of true positive rate (TP), false positive rate (FP) and latency. We conducted rigorous experiment evaluating our idea in the context of anti-phishing and achieved good result. In addition, we also make a few other contributions in this thesis to the literature by proposing five anti-phishing techniques, each of which aims at improving the state-of-the-art solutions from one perspective. In this section, we would like to give a recapitulation of the results and contributions of each of our proposed methods in this thesis.

9.1.1 A Feature-type-aware Cascaded Learning Framework

This is the leading work of our thesis, with the goal of balancing the desire to minimize the FP, maximize the TP and operate efficiently for a variety of domains that require solutions with low latency and high performance. Built upon the understanding that the distribution of the web is highly skewed and various features have different costs and perform differently, our approach integrates a heterogeneous set of features and learns a cascade of classifiers with increasing complexity automatically. By utilizing lightweight features in early stages and postponing prohibitive features to later stages of the cascade, our approach achieves a superior runtime performance while maintaining a good classification rate. In the context of anti-phishing, we conducted a comprehensive evaluation of our cascaded approach on a classic corpus and a new phishing corpus. In particular, our approach achieves a 55.7% reduction in runtime on average over traditional single-stage models, with a low FP of 0.65% and a TP of 83.34%.

9.1.2 Improving Human Verification via Computational Techniques

To improve human effort in labeling phishing attacks, we explored novel techniques using computational approaches and designed a system that trains humans in identifying potential phish and enhances phish labeling by taking advantage of individual contributions. Using tasks posted to the Amazon Mechanical Turk human effort market, we measure the accuracy of minimally trained humans in identifying potential phish, and consider methods for best taking advantage of individual contributions. In particular, we use clustering techniques to facilitate phishing labeling

by aggregating similar phish in terms of textual content, and exploit difference among individual users via a vote weighting mechanism to improve the results of human effort in fighting phishing. We found that these techniques could increase coverage over and were significantly faster than existing blacklists used today.

9.1.3 Enhancing URL Blacklists with Adaptive Probabilistic Techniques

To augment human verified blacklists, we proposed a hierarchical blacklist-enhanced phish detection method, which leverages existing human-verified blacklists and applies the shingling technique, a popular near-duplicate detection algorithm used by search engines, to detect phish in a probabilistic fashion with very high accuracy. Our goal here is to combine the best aspects of human verified blacklists and heuristic-based methods, i.e., the low false positive rate of the former and the broad and fast coverage of the latter. Comprehensive experiments over a diverse spectrum of data sources show that our method achieves 0% FP with a TP of 67.15% using search-oriented filtering, and 0.03% FP and 73.53% TP without the filtering module. With incremental model building capability via a sliding window mechanism, our approach is able to adapt quickly to new phishing variants, and is thus more responsive to the evolving attacks.

9.1.4 Detecting Phish via Textual Identity Discovery and Keywords Retrieval

To directly exploit the inconsistency between the claimed identity and the genuine identity of a web page, we designed an anti-phishing technique with an identity-based component that discovers the discrepant dual identities, and a keywords-retrieval component that utilizes the power of search engines to identify phish. This method requires no training data, no prior knowledge of phishing signatures and specific implementations, and thus is able to adapt quickly to constantly appearing new phishing patterns. Comprehensive experiments over a diverse spectrum of data sources show that both components have a low FP and the stacked approach achieves a TP of 90.06% with an FP of 1.95%.

9.1.5 A Feature-rich Machine Learning Framework for Phish Detection

To capture more novel phish and partially alleviate the problem of high FP in the feature-based techniques, we proposed a layered anti-phishing solution called CANTINA+ that exploits the expressiveness of a rich set of features with machine learning to achieve a high TP on novel phish, and limits the FP to a low level via a hash-based near-duplicate phish filter and a login form detector. We extensively evaluated CANTINA+ with two methods on a diverse spectrum of corpora with $8,118$ phish and $4,883$ legitimate webpages. In the randomized evaluation, CANTINA+ achieved over 92% TP on unique testing phish and over 99% TP on near-duplicate testing phish, and about 0.4% FP with 10% training phish. In the time-based evaluation, CANTINA+ also achieved over 92% TP on unique testing phish, over 99% TP on near-duplicate testing phish, and about 1.4% FP under 20% training phish with a two-week sliding window.

9.1.6 Detecting Phish via Logo Images

In this work, we proposed an approach that takes on phishing attacks by examining the inconsistency between the claimed identity (e.g., an eBay logo) and the genuine identity (e.g., a rapidshare.com domain) of a web page via a very strong signal on a web page, i.e., the brand logo image. Our approach utilizes novel features with machine learning (ML) algorithms to identify

the logo image, and then classifies a web page as phish or nonphish by inspecting the inconsistent identities via near-duplicate image matching techniques. Our experiments show that our approach has a TP of 89.1% and an FP of 0.7% in identifying the brand logo among all images on a web page. Based on this result, our approach achieves a TP of 87.63% and an FP of 0% in detecting phish. In particular, our approach runs 38.72 seconds faster on average than traditional methods that process all images on a web page via pixel-wise comparisons, and is also robust against various image manipulation tricks.

9.2 Future Work

Although we evaluated the slew of our proposed approaches in the context of anti-phishing in this thesis, our feature-type-aware cascaded learning technique can be generalized to a variety of other domains. In this section, we will devote our concentration to the potential improvements and further extensions of our thesis work.

9.2.1 Adapting Our Technique to Other Domains

One property of our cascaded learning framework is to give practitioners flexibility to prioritize either the TP or FP in training the cascaded model. A plethora of fields emphasize a very low FP such as anti-phishing, spam email filtering, virus detection and so on, and other domains especially those with no concerns of liability issues favor a high TP such as bad language detection.

We have demonstrated the effectiveness of our cascaded learning technique for the former, and yet have not got a chance proving our conjecture for the latter. Toward that end, we need to go through a whole slew of steps as we did in our previous work. Specifically, we need to select a domain, perform data collection, learn cascaded models in the training set, and finally conduct evaluation on the holdout testing set.

9.2.2 Large-scale Live Evaluation on Fresh Phish

In section 8.4.2, we refrained from utilizing major public phish feeds such as PhishTank and chose to evaluate our prototype phish detector on a small set of phishing URLs that we managed to build ourselves mainly due to two reasons. First, popular phish repositories are typically linked to leading phishing filters like Google Safe Browsing, and therefore, using those feeds in the comparative experiment tends to yield results that are biased against our approach. Second, monitoring phish feeds constantly and performing evaluation before their phishing URLs are incorporated into the database of leading phish filters is a nontrivial labor-intensive task, while building phishing attacks is sufficiently challenging in its own way.

The timeliness of live evaluation against industry phish filters puts a strict requirement on the freshness of the phishing web pages, and to guarantee the scale of such experiments, we need to either build a large set of phishing attacks ourselves or conduct incremental evaluation during an extended period of time with each day examining a certain number of phishing URLs in primary phish feeds before they are crawled by major filter vendors.

9.2.3 Clustering of Phish as a Feature for Phish Detection

Features are the key ingredient to any machine learning task, and to further enhance the performance of our anti-phishing techniques, it is necessary to come up with more novel and effective

features.

Given the fact that an increasingly large number of phishing web pages were automatically created by toolkits [26][61] in recent years, which substantially increases the scale of attacks and generates a fair amount of similar HTML content, it is thus natural and reasonable to exploit this high similarity to design features for phish detection. Specifically, one idea along this line is to cluster the phishing web pages in the training corpus into clusters and save the cluster centroids as reference points for distance computation. When extracting features for a web page, we calculate the similarity between that page and each centroid, and choose the maximum as the value for this "clustering" feature. The rationale of this feature is that the vast majority of phishing attacks only target a small number of web sites, and if the clustering process could align most target brands with the resultant clusters with sufficient accuracy, this clustering feature has a good chance of distinguishing phish from legitimate web pages. The use of this feature will in turn incur additional work on choosing a reasonable clustering algorithm, deciding the number of clusters if necessary, and so on.

Bibliography

[1] http://sb.google.com/safebrowsing/update?version=goog-white-domain:1:1. 3.2.2

[2] http://www.millersmiles.co.uk/scams.php. 3.2.2, 4.4.1, 5.6

[3] http://sb.google.com/safebrowsing/update?version=goog-white-domain:1:1. 4.4.1, 5.6

[4] http://bit.ly/trQd6J. 4.1, 5.6, 8.2.1

[5] http://rdf.dmoz.org/. 4.1

[6] http://www.uribl.com. 5.6

[7] http://bit.ly/YVSrvu. 5.6, 8.2.1

[8] http://bit.ly/Vn0CxR. 5.6, 8.2.1

[9] http://bit.ly/TM1jm2. 5.6, 8.2.1

[10] http://bit.ly/GH9mrw. 5.6, 8.2.1

[11] http://bit.ly/WatW7n. 5.6, 8.2.1

[12] http://bit.ly/GHjJeM. 5.6, 8.2.1

[13] Boom time for cybercrime - the economy and online social networks are the latest fodder for scams. 2011. http://bit.ly/xV2Oe. 1

[14] Top 500 sites on the web, 2011. http://www.alexa.com/topsites. 7.3.1

[15] 3sharp report. Gone phishing: Evaluating anti-phishing tools for windows. 2006. http://www.3sharp.com/projects/antiphishing/gone-phishing.pdf. 4.1, 5.6, 6.1, 8.2.1

[16] G. Aaron. Apwg phishing activity trends report in 1st quarter 2012. 2012. http://www.antiphishing.org/reports/apwg_trends_report_q1_2012.pdf. 1

[17] S. N. Bannur, L. K. Saul, and S. Savage. Judging a site by its content: Learning the textual, structural, and visual features of malicious web pages. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (AISec'11)*, pages 1–10, 2011. 2.1.4, 8.1.2

[18] E. Bauer and R. Adams. *Reliability and Availability of Cloud Computing*. Wiley-IEEE Press, first edition, 2012. 8.3.3, 8.3.3

[19] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *Proceedings of the 6th international conference on World Wide Web*, pages 1157–1166, 1997. 4.3.2, 7.2.1

[20] S. C. Brubaker, J. Wu, J. Sun, M. D. Mullin, and J. M. Rehg. On the design of cascades

of boosted ensembles for face detection. *International Journal of Computer Vision*, 77(1-3):65–86, 2008. 2.2, 8.1.3

[21] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998. 6.5.2, 7.3.2

[22] T.-C. Chen, S. Dick, and J. Miller. Detecting visually similar web pages: Application to phishing detection. *ACM Transactions on Internet Technology (TOIT)*, 10(2), 2010. 2.1.4

[23] N. Chou, R. Ledesma, Y. Teraguchi, and J. C. Mitchell. Client-side defense against web-based identity theft. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04)*, 2004. 2.1.4

[24] C. Cortes and M. Mohri. Auc optimization vs. error rate minimization. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS'2003)*, 2003. 6.6.1

[25] D. Cosley, D. Frankowski, L. Terveen, and J. Riedl. Suggestbot: Using intelligent task routing to help people find work in wikipedia. In *Proceedings of the 12th International Conference on Intelligent User Interfaces (IUI'07)*, pages 32–41, 2007. 2.1.3

[26] M. Cova, C. Kruegel, and G. Vigna. There is no free phish: An analysis of 'free' and live phishing kits. In *Proceedings of the 2nd USENIX Workshop on Offensive Technologies (WOOT'08)*, 2008. 4.2, 6.3, 8.2.1, 9.2.3

[27] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 symposium on Usable privacy and security (SOUPS'2005)*, pages 77–88, 2005. 2.1.1, 2.1.4

[28] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'2006)*, pages 581–590, 2006. 7.1

[29] J. S. Downs, M. B. Holbrook, and L. F. Cranor. Decision strategies and susceptibility to phishing. In *Proceedings of the 2nd Symposium on Usable Privacy and Security (SOUPS'2006)*, pages 79–90, 2006. 7.1

[30] W. K. Edwards, E. S. Poole, and J. Stoll. Security automation considered harmful. In *Proceedings of the 2007 Workshop on New Security Paradigms (NSPW'07)*, pages 33–42, 2007. 3.1

[31] M. Ester, H. peter Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996. 7.2.1

[32] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 2006. 6.6.1

[33] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Articial Intelligence*, pages 1022–1027, 1993. 7.4.1

[34] I. Fette, N. Sadeh, and A. Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web (WWW'07)*, pages 649–656, 2007. 8.5.6

[35] D. Fetterly, M. Manasse, and M. Najork. On the evolution of clusters of near-duplicate web pages. In *Proceedings of the 1st Conference on Latin American Web Congress (LA-WEB'03)*, pages 37–45, 2003. 4.4.3, 6.6.2

[36] S. Garera, N. Provos, M. Chew, and A. D. Rubin. A framework for detection and measure-

ment of phishing attacks. In *Proceedings of the 2007 ACM Workshop on Recurring Malcode (WORM'07)*, pages 1–8, 2007. 2.1.4, 6.5, 5, 5

[37] J. Goecks and E. D. Mynatt. Supporting privacy management via community experience and expertise. In *Proceedings of 2005 Conference on Communities and Technology*, pages 397–418, 2005. 3.3

[38] S. A. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, 2006. 2.1.3

[39] J. Heer and M. Bostock. Crowdsourcing graphical perception: using mechanical turk to assess visualization design. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI'10)*, pages 203–212, 2010. 2.1.3

[40] M. Henzinger. Combinatorial algorithms for web search engines: three success stories. In *Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms*, pages 1022–1026, 2007. 4.3.4

[41] C. Herley and D. Florencio. A profitless endeavor: Phishing as tragedy of the commons. In *Proceedings of the 2008 workshop on New security paradigms*, pages 59–70, 2008. 1

[42] J. Hong. The state of phishing attacks. *Communications of the ACM*, 55:74–81, 2012. 2.1

[43] P. G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS: Crossroads, The ACM Magazine for Students*, 17(2), 2010. 2.1.3

[44] S. J. Karau and K. D. Williams. Social loafing: A meta-analytic review and theoretical integration. *Journal of Personality and Social Psychology*, 65(4):681–706, 1993. 2.1.3

[45] Y. Ke and R. Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer vision and Pattern Recognition (CVPR'04)*, pages 506–513, 2004. 7.2.3

[46] L. Kessem. Phishing in season: A look at online fraud in 2012. 2012. `http://bit.ly/O48Rfa`. 1

[47] E. Kirda and C. Kruegel. Protecting users against phishing attacks with antiphish. In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, volume 1, pages 517–524, 2005. 2.1.4, 2.1.4

[48] M. Kirkland. Facebook phishing scam costs victims thousands of dollars. 2011. `http://bit.ly/uIP1qi`. 1

[49] A. Kittur, E. H. Chi, and B. Suh. Crowdsourcing user studies with mechanical turk. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI'08)*, pages 453–456, 2008. 2.1.3

[50] P. Kumaraguru, J. Cranshaw, A. Acquisti, L. Cranor, J. Hong, M. A. Blair, and T. Pham. School of phish: A real-word evaluation of anti-phishing training. In *Proceedings of the 5th Symposium on Usable Privacy and Security (SOUPS'09)*, 2009. 2.1.2, 3.1, 3.5.1

[51] A. Le, A. Markopoulou, and M. Faloutsos. Phishdef: Url names say it all. *CoRR*, abs/1009.2275, 2010. 2.1.4

[52] C. X. Ling, Q. Yang, J. Wang, and S. Zhang. Decision trees with minimal costs. In *Proceedings of the International Conference on Machine Learning (ICML'04)*, pages 4–8, 2004. 2.2

[53] G. Liu, G. Xiang, B. A. Pendleton, J. I. Hong, and W. Liu. Smartening the crowds: Computational techniques for improving human verification to fight phishing scams. In *Proceedings of the 7th Symposium On Usable Privacy and Security (SOUPS'11)*, 2011. 1, 7.2.1

[54] W. Liu, X. Deng, G. Huang, and Y. Fu. An antiphishing strategy based on visual similarity assessment. *IEEE Internet Computing*, 10(2):58–65, 2006. 2.1.4

[55] C. Ludl, S. McAllister, E. Kirda, and C. Kruegel. On the effectiveness of techniques to detect phishing sites. *Lecture Notes in Computer Science (LNCS)*, 4579:20–39, 2007. 2.1.4

[56] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *Proceedings of the International Conference on Machine Learning (ICML'09)*, pages 681–688, 2009. 2.1.4

[57] H. Masnadi-shirazi and N. Vasconcelos. High detection-rate cascades for real-time object detection. In *Proceedings of the 11th IEEE International Conference on Computer Vision (ICCV'07)*, pages 1–6, 2007. 2.2

[58] W. Mason and D. J. Watts. Financial incentives and the "performance of crowds". In *Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP'09)*, pages 77–85, 2009. 2.1.3

[59] E. Medvet, E. K. Eurecom, and C. Kruegel. Visual-similarity-based phishing detection. In *Proceedings of the 4th international conference on Security and privacy in communication networks (SecureComm'08)*, pages 30–36, 2008. 2.1.4

[60] D. Millen, M. Yang, S. Whittaker, and J. Feinberg. Social bookmarking and exploratory search. In *Proceedings of the 2007 Tenth European Conference on Computer-Supported Cooperative Work (ECSCW07)*, pages 21–40, 2007. 2.1.3

[61] T. Moore and R. Clayton. Examining the impact of website take-down on phishing. In *Proceedings of the Anti-phishing Working Groups (APWG) 2nd annual eCrime Researchers Summit*, pages 1–13, 2007. 4.1, 4.2, 7.2.1, 9.2.3

[62] T. Moore and R. Clayton. Evaluating the wisdom of crowds in assessing phishing websites. In *Proceedings of the 12th International Financial Cryptography and Data Security Conference (FC 2008)*, 2008. 3.1, 3.4.1, 3.5.1

[63] NIST. Secure hash standard. *Federal Information Processing Standards Publication 180-1*, 1995. National Institute of Standards and Technology (NIST). 6.3

[64] Y. Pan and X. Ding. Anomaly based web phishing page detection. In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 381–392, 2006. 2.1.4

[65] PhishTank. `http://data.phishtank.com/data/online-valid/`. 2.1.4, 4.4.2, 5.6, 6.6.2

[66] PhishTank. Statistics about phishing activity and phishtank usage. `http://www.phishtank.com/stats.php`. 2.1.4, 3.1, 4.4.2

[67] L. Qi. Phishing report from the anti-phishing alliance of china. 2011. `http://www.apac.org.cn/gzdt/201110/P020111011570515875992.pdf`. 1

[68] V. C. Raykar, B. Krishnapuram, and S. Yu. Designing efficient cascaded classifiers: Tradeoff between accuracy and cost. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 853–860, 2010. 2.2

[69] A. P. E. Rosiello, E. Kirda, C. Kruegel, and F. Ferrandi. A layout-similarity-based approach

for detecting phishing pages. In *Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks (SecureComm'07)*, pages 454–463, 2007. 2.1.4

[70] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *Proceedings of the 14th conference on USENIX Security Symposium*, pages 17–32, 2005. 2.1.4

[71] E. F. T. K. Sang and F. D. Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Computational Natural Language Learning (CoNLL-2003)*, pages 142–147, 2003. 5.4.2

[72] S. Sheng, P. Kumaraguru, A. Acquisti, L. Cranor, and J. Hong. Improving phishing countermeasures: An analysis of expert interviews. In *Proceedings of the 4th APWG eCrime Researchers Summit*, 2009. 2.1.4

[73] S. Sheng, B. Magnien, P. Kumaraguru, A. Acquisti, L. F. Cranor, J. Hong, and E. Nunge. Anti-phishing phil: the design and evaluation of a game that teaches people not to fall for phish. In *Proceedings of the 3rd symposium on Usable privacy and security (SOUPS'07)*, pages 88–89, 2007. 2.1.2, 3.2.2, 3.4.1

[74] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong, and C. Zhang. An empirical analysis of phishing blacklists. In *Proceedings of the 6th Conference on Email and Anti-Spam (CEAS'09)*, 2009. 2.1.4, 3.5.1, 4.4.2, 6.1

[75] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time url spam filtering service. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy (S&P'11)*, pages 447–462, 2011. 2.1.4, 8.5.2

[76] A. Tversky. Features of similarity. *Psychological Review*, 84(2):327–352, 1977. 5

[77] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'01)*, pages 511–518, 2001. 2.2

[78] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI'04)*, pages 319–326, 2004. 2.1.3, 3.1

[79] G. Wang, H. Liu, S. Becerra, and K. Wang. Verilogo: Proactive phishing detection via logo recognition. Technical Report CS2011-0969, UNIVERSITY OF CALIFORNIA, SAN DIEGO, 2011. 2.1.4

[80] R. Weaver and M. P. Collins. Fishing for phishes: Applying capture-recapture methods to estimate phishing populations. In *Proceedings of the Anti-Phishing Working Groups 2nd Annual Ecrime Researchers Summit (eCrime'07)*, pages 14–25, 2007. 3.1

[81] L. Wenyin, G. Liu, B. Qiu, and X. Quan. Anti-phishing by discovering phishing target. *IEEE Internet Computing*, 2011. 2.1.4

[82] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification of phishing pages. In *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS'10)*, 2010. 1, 2.1.4

[83] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, second edition, 2005. 6.5.2, 7.3.2

[84] M. Wu, R. C. Miller, and G. Little. Web wallet: preventing phishing attacks by reveal-

ing user intentions. In *Proceedings of the 2nd symposium on Usable privacy and security (SOUPS'2006)*, pages 102–113, 2006. 2.1.1

[85] G. Xiang and J. Hong. A hybrid phish detection approach by identity discovery and keywords retrieval. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 571–580, 2009. 2.1.4, 3.2.2, 4.3.3, 1, 6.1, 12, 14, 8.2.2

[86] G. XIANG, J. HONG, C. P. ROSE, and L. CRANOR. Cantina+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)*, 14(2), 2011. 1, 2.1.4, 1, 7.2.1, 8, 8.1.2, 8.1.3, 8.2.1, 8.2.2, 8.3.5

[87] G. Xiang, G. Liu, J. Hong, and C. Rose. Using brand logos as a highly effective and robust technique for detecting phishing web sites. In *Submitted to the 18th International Conference on World Wide Web (WWW'12)*, 2012. 1, 8

[88] G. Xiang, B. A. Pendleton, J. I. Hong, and C. P. Rose. A hierarchical adaptive probabilistic approach for zero hour phish detection. In *Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS'10)*, pages 268–285, 2010. 1, 2.1.4, 3.2.2, 3.2.3, 1, 6.3, 7.4.1, 8

[89] M.-H. Yang, D. J. Kriegman, and N. Ahuja. Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 24(1):34–58, 2002. 1, 8

[90] C. Yue and H. Wang. Bogusbiter: A transparent protection against phishing attacks. *ACM Transactions on Internet Technology (TOIT)*, 10(2), 2010. 2.1.4

[91] B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the Third IEEE International Conference on Data Mining*, pages 435–442, 2003. 6.7.5

[92] Y. Zhang, J. Hong, and L. Cranor. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*, pages 639–648, 2007. 1, 2.1.4, 4.3.3, 5.2, 5.7.5, 6.1, 12, 6.7.6, 8, 13, 8.3.2

[93] W.-L. Zhao, C.-W. Ngo, H.-K. Tan, and X. Wu. Near-duplicate keyframe identification with interest point matching and pattern learning. *IEEE Transactions on Multimedia*, 9(5):1037–1048, 2007. 2, 7.2, 7.2.3