University of Trento

Effective Analysis, Characterization, and Detection of Malicious Activities on the Web

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

by

Birhanu Mekuria Eshete

December 2013

Acknowledgments

The well-spent years in my PhD were incredibly adventurous and transforming. Of all the lessons I learned, solving problems that matter to humanity is what always echoes in my head. Equally captivating to me is the often ignored truth that research should be fun to do and should not be a rote technical gymnastic to comply with academic routine. As much as the "what" and the "how" of doing research matter, so does the "why". During my PhD, I was lucky to have the support of many great people and I, therefore, owe them a heartfelt gratitude.

Most of all, I thank my supervisor, Prof. Adolfo Villafiorita, not only for his paramount mentorship but also for his kindness to give me the freedom to think independently and to think outside the box.

Next, I am thankful to members of my examination committee: Prof. Alessandro Armando, Prof. Luca Vigano, Prof. Venkat Venkatakrishnan, and Prof. Paolo Tonella, who dedicated their time to provide me with priceless comments to improve this dissertation.

I am particularly grateful to Prof. Venkat Venkatakrishnan for hosting me in his research group at the University of Illinois at Chicago and for the fruitful collaboration that ended up to be a chapter in this dissertation. I am also grateful to Prof. Mohammad Zulkernine from Queen's University in Canada for the great collaboration on a part of this dissertation.

From the days I was hesitant about pursuing a PhD up until the day I completed this dissertation, Komminist Weldemariam is not only a great friend but also a motivator and contributor to this dissertation.

I am indebted to Federico Maggi for the initial discussions that ignited my interest to work on this fascinating research topic.

I highly acknowledge the Bruno Kessler Foundation for funding my research work.

Biniyam Asfaw, Fitsum Meshesha, and Surafel Lemma are simply the greatest friends one can have. Their countless assistance and moral support makes me feel lucky to have them.

With my friend Biruk Haileye, I had the privilege of exploring fundamental topics in life as much as it takes, no matter what. I still miss those eye-opening conversations with him.

Itzel Morales, my special friend in Trento, with whom I shared a lot of constructive ideas and memorable moments is a great friend that I will always remember. Thank you Itzel for your great friendship.

Many friends and colleagues have left living memories in the course of this adventurous journey. So, I thank (in no particular order): Andrea Mattioli, Giordano Adami, Andrea Manica, Frencesca Longo, Aaron Ciaghi, Pietro Molini, Andrea Nodari, Lorenzo Rigato, Roberto Zen, Giulia Petronella, Andrea Bontempelli, Ali Alshammari, Ilse Grau, Andrea Avancini, Mariano Ceccatto, Chiara Di Francescomarino, Cu D. Nguyen, Mirko Morandini, Kalpana Gondi, Rigel Gjomemo, Phu Phung, Karen Heart, Maliheh Monshizadeh.

My family has always been my source of inspiration and courage. I am thankful of their love and encouragement. In particular, I am grateful to my mother, my sister (Mimmi), and my brothers (Tizazu and Tekabe).

Last, but certainly not least, Helen deserves a special gratitude for her unconditional love, support, and patience.

Birhanu M. Eshete
December 2013
Trento, Italy

Abstract

The Web has evolved from a handful of static web pages to billions of dynamic and interactive web pages. This evolution has positively transformed the paradigm of communication, trading, and collaboration for the benefit of humanity. However, these invaluable benefits of the Web are shadowed by cyber-criminals who use the Web as a medium to perform malicious activities motivated by illegitimate benefits. Cyber-criminals often lure victims to visit malicious web pages, exploit vulnerabilities on victims' devices, and then launch attacks that could lead to: stealing invaluable credentials of victims, downloading and installation of malware on victims' devices, or complete compromise of victims' devices to mount future attacks.

While the current state-of-the-art is to detect malicious web pages is promising, it is yet limited in addressing the following three problems. First, for the sake of focused detection of certain class of malicious web pages, existing techniques are limited to partial analysis and characterization of attack payloads. Secondly, attacker-motivated and benign evolution of web page artifacts have challenged the resilience of existing detection techniques. The third problem is the prevalence and evolution of Exploit Kits used in spreading web-borne malware. In this dissertation, we present the approaches and the tools we developed to address these problems.

To address the partial analysis and characterization of attack payloads, we propose a holistic and lightweight approach that combines static analysis and minimalistic emulation to analyze and detect malicious web pages. This approach leverages features from URL structure, HTML content, JavaScript executed on the client, and reputation of URLs on social networking websites to train multiple models, which are then used in confidence-weighted majority vote classifier to detect unknown web pages. Evaluation of the approach on a large corpus of web pages shows that the approach not only is precise enough in detecting malicious web pages with *very low false signals but also does detection with a minimal performance penalty.*

To address the evolution of web page artifacts, we propose an evolution-aware approach that tunes detection models inline with the evolution of web page artifacts. Our approach takes advantage of evolutionary searching and optimization using Genetic Algorithm to decide the best combination of features and learning algorithms, i.e., models, as a function of detection accuracy and false signals. Evaluation of our approach suggests that it reduces false negatives by about 10% on a fairly large testing corpus of web pages.

To tackle the prevalence of Exploit Kits on the Web, we first analyze source code and runtime behavior of several Exploit Kits in a contained setting. In addition, we analyze the behavior of live Exploit Kits on the Web in a contained environment. Combining the analysis results, we characterize Exploit Kits pertinent to their attack-centric and self-defense behaviors. Based on these behaviors, we draw distinguishing features to train classifiers used to detect URLs that are hosted by Exploit Kits. The evaluation of our classifiers on independent testing dataset shows that our approach is effective in precisely detecting malicious URLs linked with Exploit Kits with very low false positives.

Keywords

malicious web pages, web-borne malware, holistic characterization, evolution-aware detection, exploit kits.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

When Tim Berners-Lee invented the Web in 1991, it was a collection of a handful of static web pages used by a small number of scientists at CERN [6]. Now, the Web has evolved into a ubiquitous global platform that hosts well over a trillion of web pages backed by hundreds of millions of complex web applications used by billions of people from all over the globe [3, 45, 67]. The ubiquity of the Web has positively transformed the way people do business, spend their spare time, exchange ideas, and socialize. With new online businesses and communities emerging every day, the web will continue to evolve in features and content it hosts, and humanity will also continue to increasingly depend on the Web.

Unfortunately, the Web is also where evil-doers carry-out criminal activities to: steal invaluable credentials (e.g., online banking credentials) from unsuspecting victims, infect victims' devices with malware (e.g., keylogger), or even compromise and remotely command-and-control victims' devices as part of a Botnet [13, 22, 72, 83]. On the Web, it just takes a mere visit of a web page to be a victim of these kinds of attacks. In a typical daily online activity, it has become customary to visit a myriad of web pages for activities such as searching information, keeping in-touch with other people, checking-out news, or watching a video.

Taking advantage of the sheer number of users of the Web and the lack of awareness of users, one common strategy used by cyber-criminals is to lure unsuspecting victims into visiting a web page that is either purposely crafted to launch attacks or a vulnerable legitimate web page that is under the control of the cyber-criminal. Most attacks on the Web happen when victims visit malicious websites. They lure them to give away sensitive information (e.g., phishing sites) or exploit vulnerabilities in the web browser, its extensions and plugins (e.g., drive-by-downloads, malicious advertisements) to drop a malware binary on the victim's computer [16, 18, 19]. To maximize the success rate of malicious activities, cyber-criminals employ several traffic attraction mechanisms such as spam email, black-hat Search Engine Optimization (SEO), pay-per-install services, web blogs, and social networking websites.

In the past, orchestrating such malicious activities required an experienced attacker to craft malicious web pages. The motivation was often fame and curiosity. Now, the motivation behind cyber-crime is largely illegal financial gain. Inexperienced attackers can purchase attack toolkits, called Exploit Kits, to craft malicious web pages from which they can mount attacks. The Exploit Kits are developed and marketed by experienced attackers in the underground market [86]. Akin to legitimate software, Exploit Kits are periodically upgraded with novel attack payloads and evasion techniques to challenge detection mechanisms [37].

In order to defend users from malicious activities on the Web, the research community first responded with *blacklisting* of known or suspected malicious URLs, domains, and IP addresses. As blacklisting does not cope with the explosive growth of the number of web pages and exhaustive blacklisting is infeasible, *heuristics-based* countermeasures emerged. Heuristics-based methods are effective only to identify known attack patterns for which signatures are stored a priori. In addition, the rate at which malicious activities evolve on the Web is by far faster than the rate at which signature databases are updated —rendering such countermeasures ineffective.

To address the inadequacy of blacklisting and heuristics-based detection of malicious activities on the Web, two complementary approaches have been proposed. These techniques are based on static analysis and dynamic analysis of web pages. *Static analysis* techniques quickly extract features that characterize malicious activities on a web page without rendering the page in a browser [13, 14, 54, 55, 79, 89]. By contrast, *dynamic analysis* techniques capture behavioral artifacts that characterize malicious activities when the page is executed by the browser [11, 18, 43, 65, 72, 75, 78].

Despite existing countermeasures, the Web has become more and more susceptible to an increasing number of malicious activities [87]. With the aim of shedding light on the state of malware on the Web, Provos et al. [70] conducted a measurement study on a corpus of about 4.5 million web pages indexed by Google. They found out that one in ten web pages may contain malicious code. In another study, Provos et al. [71] reported that Google identified over 3 million URLs that launched attacks against their visitors. An even more troubling finding is that about 1.3% of the search queries submitted from users of Google search were served with at least one malicious URL. In a recent study, Canali and Balzarotti [12] deployed a network of 500 honeypot websites with several services over a period of 100 days to analyze what attackers do during and after compromising a website. Their findings indicate that, among other 10 malicious activities, phishing, drive-by-downloads, and defacement of websites are prevalent on the Web.

Similarly to these large-scale measurement studies, security firms also report that recent years have witnessed a significant prevalence of *malicious activities* embedded in web pages. The prevalence reports show two categories of malicious activities, i.e., those embedded in the purposely-crafted malicious web pages and those injected into legitimate websites compromised by cyber-criminals.

With regards to prevalence of malicious web pages, in the period 2009-2010, Websense estimated an increase of about 111% in the number of malicious web pages [101]. In the same period, Symantec estimated about $310,000$ malicious domains and a monthly average of about 4.4 million malicious web pages [85]. A recent report from Symantec on threats on the Internet indicates an increase in the number of: malware URLs sent via email, phishing attacks, and malicious domains [87].

The prevalence reports also show a trend in using compromised legitimate websites for malicious activities. For instance, Symantec reported that in 2010, about 70 of the top 100 reputable websites were either hosting malicious content or were injected with an iframe that redirects to other malicious web pages. In the same year, Websense [101] reported that about 80% of the Websites with malicious code were compromised legitimate sites.

The insights from these studies are attributed to three challenges in the state of the art detection techniques. Firstly, the focus on specific classes of malicious activities such as drive-by-download attacks. Secondly, the evolution of the threat landscape which challenges the resilience of existing countermeasures. Thirdly, the prevalence of Exploit Kits that played a key role in the proliferation of malicious activities on the Web.

In this dissertation, we present approaches that improve the current state of the art with the aim of effective analysis, characterization, and detection of malicious activities on the Web. As the aforementioned studies found out, malicious web pages are one of the major vectors to carry-around malicious activities on the Web. Hence, the problem domain of this dissertation is in the context of *malicious activities* embedded in malicious web pages pertinent to phishing sites, drive-by-download sites, malware distribution sites, malicious advertisements, and URLs linked with Exploit Kits.

1.1 Research Problems

Given the alarming prevalence of malicious activities on the Web and the constantly evolving tactics of cyber-criminals to spawn new variants of attacks [85, 86, 87], existing approaches, which are pretty effective at detecting one prominent attack (such as drive-by-downloads), are limited to partial and course-grained analysis and characterization of attack payloads [27, 63]. Moreover, on top of the prevalence of Exploit Kits on the Web, the constant evolution of attack payloads in malicious web pages and the healthy evolution of artifacts of benign web pages challenges the effectiveness and efficiency of existing defenses against malicious activities on the Web. In the following, we briefly pose the specific research problems that this dissertation addresses.

1.1.1 Partial Analysis and Characterization of Attacks

Existing approaches base the intuition of their detection techniques on specific attacks (e.g., phishing, drive-by-downloads). However, cyber-criminals are often one step ahead in crafting virtually any possible combination of existing attacks or blending existing attack payloads with newly spawned threats and embed it into web pages. As a result, the majority of the techniques, as they are based on partial analysis of web page artifacts, overlook a different type of attack. This happens due to loose characterization of malicious activities. Consequently, malicious web pages escape detection techniques.

1.1.2 Evolution of Web Page Artifacts

Artifacts of web pages, on which existing analysis and detection techniques are based, are under evolution [69]. Old artifacts become less relevant over time while new ones emerge due to evolution of artifacts. For benign web pages, the evolution of artifacts is attributed to: hosting infrastructure, web page source, functionality, web protocols, browser components and its extensions, and usage policies. Evolution regarding artifacts of malicious web pages is due to emerging attack vectors that exploit vulnerabilities. For instance, the transition from HTML4 to HTML5 is a relevant example for changes in some artifacts which can be extracted from HTML content of a web page (e.g., inline multimedia inclusion, local storage) [103]. Evolution of web pages results in evolution of the training dataset used to generate the detection models. In effect, the performance of the detection model eventually changes and the change might mean reduction in accuracy which leads to having more false signals. Hence, the challenge for existing analysis and detection techniques is how to cope with such evolution in artifacts of web pages without compromising the precision of detection.

1.1.3 Prevalence of Exploit Kits

Since the advent of the first Exploit Kit in 2006, Exploit Kits have become prevalent mean of attack on the Web [51]. To this end, it is a natural question to ask whether a given URL points to an Exploit Kit. This is a question that has significant implications for the safety of Web users, given the proliferation of criminal activities in recent years and the change in the “business model” of the underground market from selling crimeware to providing it as a service akin to software-as-a-service [37, 82].

1.2 Overview of Proposed Approaches

To address the problems discussed in the previous section, we propose and evaluate three approaches with a shared goal of improving the effectiveness of the analysis, characterization, and detection of malicious activities on the Web.

1.2.1 Holistic Detection

To address the *partial analysis and characterization of attacks* in malicious web pages, we propose and evaluate a *holistic* approach called BINSPECT. Our approach aims at ensuring the right balance between the fast-and-imprecise static analysis and the slow-and-precise dynamic analysis techniques. To achieve this balance, BINSPECT leverages a combination of static analysis and minimalistic emulation and uses supervised learning techniques to detect malicious web pages. The targeted attacks are pertinent to drive-by-download, phishing, injection, and malware distribution.

While we start from previous work [13, 14, 18, 54], we introduce novel features and enhance existing features to more effectively put apart malicious and benign web pages. Our approach is lightweight while capturing a comprehensive snapshot of the artifacts we extract from web pages. This is ensured by using an optimal set of features and a minimalistic emulation to render web pages. When provided with an unknown web page, instead of relying on one best model, BINSPECT uses confidence-weighted majority vote of multiple models to classify web pages as benign or malicious.

Evaluation of BINSPECT on a large-scale corpus of web pages achieved detection accuracy above 97% with low false signals and an average performance overhead of 5 seconds to analyze and detect a single web page.

1.2.2 Evolution-Aware Detection

To address *evolution of web page artifacts*, we propose and evaluate an approach called Einspect, that leverages evolutionary searching and optimization to align learning-based detection models with the evolution of web page artifacts.

Einspect starts with an initial population of candidate models trained using standard learning algorithms based on discriminative features extracted from: URL string, HTML content, JavaScript code, and reputation metadata of web pages on social networking websites. It then uses a Genetic Algorithm to automatically search and optimize the *best* combination of features and learning algorithms to obtain what we call the *fittest model*. Using the *fittest model*, it detects unknown web pages to flag them as malicious or benign. The key idea of our approach is that, upon a new dataset, instead of re-training multiple models and pick the model(s) with the best performance, Einspect exhaustively evaluates the *best* combination of features and learning algorithms using a Genetic Algorithm.

Evaluation results of Einspect show that, on a fairly large-scale dataset, it is possible to significantly reduce false negatives (up to 10% in our dataset) of a detection model using evolutionary model searching and optimization in order to make learning-based techniques evolution-friendly.

1.2.3 Detection of Exploit Kits

To address the *prevalence of Exploit Kits*, we tackle the problem with an approach formulated as a machine learning based technique for the detection of malicious URLs hosted by Exploit Kits. At the core of our approach is that, based on contained analysis of workflows of Exploit Kits, we leverage their *attack-centric* and *self-defense* behaviors to design distinguishing features based on which we train precise classifiers to detect malicious URLs.

Our approach resembles techniques that combine honeyclients and learning to analyze the side-effects of malicious activities. These techniques (e.g., [65, 99]) inspect the pre-execution and post-execution snapshots of a honeyclient system properties (e.g., processes, memory access). However, in our approach, the focus of the characterization of malicious activities is on what happens *during execution* instead of analyzing the side-effects. In effect, the goal of the analysis is shifted from examining side-effects to analyzing firsthand execution dynamics to reveal malicious activities. Moreover, we avoid the overhead of taking system snapshots before and after execution.

We implemented our approach in a tool called WebWinnow. Moreover, we built a fast pre-filtering front-end to make WebWinnow usable in a resource-constrained environment. Evaluation of WebWinnow with real world malicious URLs suggests that it is effective in the detection of malicious URLs hosted by Exploit Kits with very low false positives.

1.3 Contributions

The following are the main contributions of this dissertation in the area of Web Security:

- We propose and evaluate an approach for *holistic* analysis, characterization, and detection of malicious web pages by leveraging static aspects, dynamic aspects, and metadata in order to capture fine-grained artifacts web pages. In addition, we reduce the analysis cost using light-weight emulation.

- We propose and evaluate an approach that exploits *evolutionary searching* and *optimization* to improve the precision of detection models using Genetic Algorithm. By doing so, our approach aligns detection techniques with the evolution of the underlying web page artifacts.

- We propose and evaluate an approach that leverages the *attack-centric* and *self-defense* behavior of Exploit Kits to detect malicious URLs that are linked with Exploit Kits.
- We introduce *novel features* to enhance learning-based detection techniques in the characterization of malicious activities on the Web.

1.4 Dissertation Structure

The rest of this dissertation is organized into the following six chapters.

In Chapter 2, we present an illustrated discussion of the predominant malicious activities on the Web.

In Chapter 3, we present the approach-level details and experimental evaluation of an approach proposed to address the problem of partial analysis and characterization of attack payloads in malicious web pages.

In Chapter 4, we present an approach to address the evolution of artifacts of both malicious and benign web pages.

In Chapter 5, we present an approach that leverages the workflow of Exploit Kits to analyze and detect malicious URLs that are linked with Exploit Kits.

In Chapter 6, we present a detailed discussion of the related work focusing on approaches proposed to detect malicious activities on the Web.

In Chapter 7, we present a summary of the problems addressed in this dissertation, the approaches we proposed to overcome the problems, limitations of our approaches, and finally some directions for future work.

Chapter 2

Malicious Activities on the Web

In this chapter, we present a discussion of malicious activities relevant to the problems addressed in this dissertation.

The malicious landscape on the Web is characterized by several malicious activities. It ranges from phishing sites [16] to rogue Anti-Virus campaigns [19]; from organized spam campaigns to malware distribution centers; from drive-by-downloads to malicious advertisements, and from pay-per-install deceptions to Exploit Kits [37].

Central to the malicious landscape on the Web are malicious web pages. A malicious web page is a web page which exploits one or more vulnerabilities of the browsing environment[1] to launch an attack upon a visit by an unsuspecting victim [72]. Typical ways through which malicious web pages carry attacks include: obfuscated malicious JavaScript, redirection to other malicious destinations (e.g., using HTTP or JavaScript redirection), victim luring (e.g., social engineering tricks), and victim-takeover (e.g., installing malware).

In a recent trend, cyber-criminals combine social engineering, spam email, black-hat SEO, and compromised benign websites to make the infection chain more complex to analyze and detect [46]. For instance, an

[1]Browsing environment refers to a combination of the client operating system, the web browser, and plugins and extensions of the browser.

attacker, after crafting a malicious web page, creates a legitimate-looking profile on a social network website and sends a friendship request to an unsuspecting victim. The victim accepts the invitation and confirms the attacker as a friend. For the first few interactions, the attacker shares legitimate links to build trust.

After a while, the attacker starts to share a link to a website that hosts malware. The victim, as it has already trusted the attacker, checks out the link. When the page of the shared link is served by the victim's browser, malware is downloaded to the victim's machine without the victim noticing it. In fact, the attack may not be limited to just installing malware. Depending on the vulnerability of the environment (e.g., the browser cache), the attacker may steal session information and impersonate the victim on the social network site [25, 94].

Another example that combines spam, black-hat SEO, and compromised benign websites starts with the attacker sending spam email that contains a link the victim has to visit. Let us suppose that the victim is suspicious of the legitimacy of the link and makes a safety check by searching it on a search engine. Since the attacker has already implemented an SEO technique to boost ranking, the link shows up in the top search results. However, the victim is still suspicious of clicking on the URL in the search result and does an eyeball inspection of the search results. Tricky enough, the attacker might have already compromised a benign website and injected a redirection script to it. At this point, the victim is likely to stop speculation and deem the website as safe-to-visit. Unfortunately, the moment the URL is rendered by the victim's browser, the actual exploit happens whereby a vulnerability in one of the browser plugins allows not only an automatic download but also execution of malware binary on the victim's machine.

As a foundation for our discussions in the subsequent chapters of this

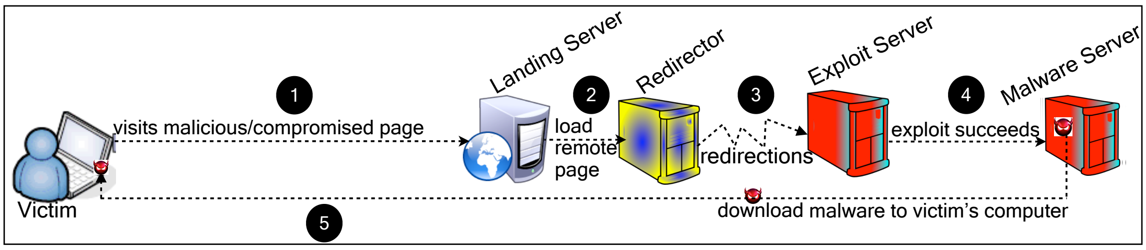


Figure 2.1: A typical drive-by-download attack chain.

dissertation, in the rest of this chapter, we focus on five malicious activities we believe are worth-watching in order to effectively analyze, characterize, and detect malicious activities on the Web. These are: drive-by-downloads, phishing sites, malware distribution centers, malicious advertisements, and Exploit Kits.

2.1 Drive-by-Downloads

In a typical drive-by-download attack (see Figure 2.1), a victim with a vulnerable browser visits a malicious (compromised) page. The page automatically redirects to a remote page that, after a series of redirections, lands on a page with the actual exploit. Then the victim's environment (e.g., the browser, browser plugins) is fingerprinted and inspected for known vulnerabilities based on which a presumably effective exploit is crafted. Finally, the exploit binary is automatically downloaded and executed on the realm of the victim's environment [24]. All this happens without the victim noticing any suspicious activity. In the following, we discuss the attack chain and implications of a real drive-by-download attack.

On September 26, 2011, when users visited `http://www.mysql.com`, the file at `http://mysql.com/common/js/s_code_remote.js?ver=20091011` was infected by a heavily obfuscated malicious JavaScript code (the de-obfuscated version of the code is shown in Listing 2.1). The malicious

JavaScript code embeds an iframe that points to the malicious domain: `http://falosfax.in/info/in.cgi?5`. Notice from Listing 2.1 that the small size of the iframe (10x10 pixels) and its hidden visibility make it difficult for a user to visually notice the difference on the page. Upon landing on this malicious domain, the browser is served with an HTTP 302 redirection[2]. This redirection leads to the exploit domain[3].

This exploit domain hosts the infamous `BlackHole` exploit pack which, upon discovering a vulnerable browsing environment (Java plugin vulnerability in this case), leads the browser to download a malware binary to the user's machine. All this happens without the user's knowledge and it happens just by visiting `www.mysql.com`. In this attack, the actual payload is an exploitation of Java runtime vulnerability in Internet Explorer 6. The final mission of the attack chain is to download and execute a malware binary. The duty of the malware binary is to steal and send to the attacker FTP client passwords from the user's machine. At first glance, the attack described before sounds specific to a compromised legitimate website, i.e., `http://www.mysql.com`. However, detailed examination of the attack chain provides a number of interesting insights.

Listing 2.1: De-obfuscated JavaScript exploit code of the attack in [5]

```
if (document.getElementsByTagName('body')[0]){
iframer();
}else{
document.write(<iframe src='http://falosfax.in/info/in.cgi?5' width='10'
    height='10'
    style='visibility:hidden;position:absolute;left:0;top:0;'></iframe>);
}
function iframer(){
 var f=document.createElement('iframe');
 f.setAttribute('src', 'http://falosfax.in/info/in.cgi?5');
```

[2]`http://falosfax.in/info/in.cgi?5&ab_iframe=1&ab_badtraffic=1&antibot_hash=1255098964&ur=1&HTTP_REFERER=http://mysql.com/`

[3]`http://truruhfhqnviaosdpruejeslsuy.cx.cc/main.php`

```
 f.style.visibility='hidden';
 f.style.position='absolute';
 f.style.left='0';
 f.style.top='0';
 f.setAttribute('width', '10');
 f.setAttribute('height', '10');
 document.getElementsByTagName('body')[0].appendChild(f);
}
```

First, the attacker targeted a high-profile website with solid user-base and large traffic. Secondly, she identified and then exploited a vulnerable spot on the website (to inject malicious code) and abused HTTP redirection to lead the browser to where the actual exploit is hosted. Then after, she exploited a vulnerability of the browser extension to trick the browser into downloading a malware binary.

Even though the target in this attack is the Java plugin, in principle this could have been any one of the vulnerable browser-components (e.g., HTML Parser) or the vulnerable browser-extensions (e.g., PDF Renderer, Flash Player) since the malware, once on the user's machine, runs with the privilege of the current user. Similarly, the downloaded malware binary could as dangerous as a key-logger that steals and submits passwords and credit card details to a remote server controlled by the attacker. Or even worse, it could be a malware that compromises the victim's machine to remotely control it as a member of Botnet[4] to use it in future criminal activities (e.g., spam campaigns, click fraud). Generally, the vulnerability of the browsing environment could be of different risks depending on the actual combination of the type and version of: the operating system, browser, and browser extensions. An essential part of the attack chain is the fingerprinting of the environment which provides clues to vulnerable

[4]Botnet: a network of compromised machines that are under the control of an attacker, often called the Bot Master [83].

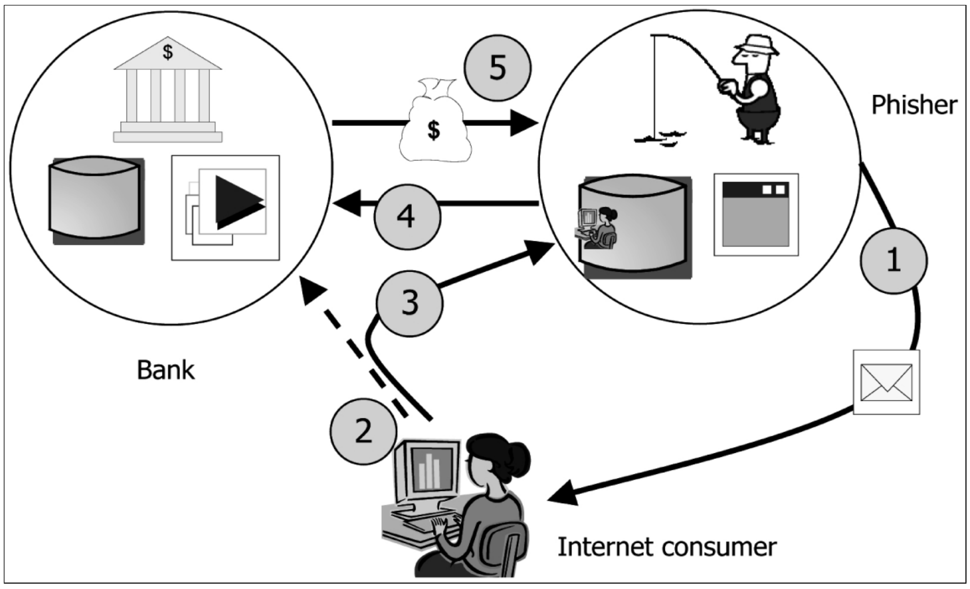


Figure 2.2: A typical phishing attack chain [15]. ①: Phisher sends a legitimate-looking email posing as a Bank. ②: User assumes email is legitimate and submits bank credentials. ③: In reality, bank credentials submitted to Phisher's server. ④: Phisher logs into the bank with user's credentials. ⑤: Phisher steals user's money.

spots based on which the next steps of the attack chain are subsequently orchestrated.

2.2 Phishing Sites

Phishing is a form of social engineering whereby cyber-criminals (Phishers in this sense) obtain confidential information from unsuspecting victims by exploiting their trust [16, 33]. A common way of phishing is using fraudulent pages that mimic trusted websites [105]. A typical phishing attack scenario is depicted in Figure 2.2. When designing these fraudulent pages, Phishers imitate the look-and-feel of a legitimate website (e.g., login page of victim's favorite bank) to lure the victim (e.g., via a link embedded in a tempting email) to give away sensitive credentials (e.g., online banking credentials, passwords).

In addition to crafting a misleading UI, Phishers deceptively manipulate the URL strings of the mimicked page to reduce the level of suspicion when victims see a slightly misspelled URL. For instance, it is likely that Phishers exploit the lack of attention by users who might not recognize the subtle difference in spelling of `https://www.unicreditbanca.it` (the legitimate URL) with respect to `https://www.unicerditbanca.it` (the fake URL).

2.3 Malware Distribution Networks

A Malware Distribution Network (MDN) refers to a collection of landing pages, malware repository servers, and intermediate redirection pages [98]. The goal of an MDN is to redirect the victim from a landing page to a malware repository sever through intermediate landing pages. Traffic is directed to landing pages in MDNs via a range of techniques such as rogue software download link, search engine referrers, links from compromised legitimate sites, or pay-per-install services. The downloading of a malware binary to the victim's machine may be automatically done by the exploit code on the malware repository server or it may involve the victim clicking on a download link (e.g., in case of rogue Anti-Virus alert).

A typical attack scenario in MDNs happens when a victim issues a search query (e.g., to download an audio player application) to a search engine. The attacker exploits SEO to boost the rank of a landing page so that the landing page shows up at the top of the search result. If the search query leads to the crafted landing page, the victim will visit it as the URL of the crafted page would show up in the top search results. When the victim visits the landing page assuming that it is where the audio player application is hosted, the MDN redirects the victim through a chain of intermediate landing pages and eventually to the malware repository server. Then from the malware repository server, a binary executable, that looks like the audio player application requested by the victim, is downloaded to the victim's machine. Then, the victim goes ahead and installs it. Once installed, the executable binary could do a myriad of attacks from key-logging to spamming other victims. The main challenge in detecting MDNs is cloaking, whereby the malware repositories and the intermediate landing pages change dynamically even for two successive visits from the same victim.

2.4 Malicious Advertisements

Online advertisement has become a multi-billion industry [10]. It is, therefore, no surprise that cyber-criminals target this lucrative industry. Malicious advertisements (malvertisements) are advertisements on the Web that infect the viewer's machine with malware. The malware makes the compromised machine a member of a Botnet, which is then used to orchestrate a more organized cyber-crime (e.g., spam campaign) [52].

Malvertisements are placed on a website via legitimate advertisements or pop-up ads which deliver the malware (e.g., in a form of Scareware[5]) as soon as the advertisement shows up on the viewer's screen. In some cases, the malware executes when the user clicks the close button on the pop-up window. Scripting languages such as ActionScript allow embedding additional logic into the advertisement [32]. This enables cyber-criminals to embed malicious code that executes in the victim's browser in parallel with the execution of the advertisement, or to redirect the victim's browser to other malicious destinations (e.g., Exploit Kit sites). To make instant detection difficult, the cyber-criminal may schedule the malicious code to run some time after the advertisement is displayed (see Listing 2.2 for an

[5]A type of malware designed to trick victims into purchasing and downloading useless and potentially dangerous software.

example in JavaScript).

Listing 2.2: A time bomb to skip execution of a malicious code on first encounter.

```
var now =new Date();
var future=Date(2013,12,31);
if now.getTime()<future.getTime(){
// keep quiet
}
```

Among the well-known incidents of malicious advertisements are the fake virus scanner alerts on the New York Times [95], eWeek [100], and FoxNews [21] that asked visitors to pay for the "virus scanner" which claimed to remove infections from their computer via a rogue Anti-Virus software.

2.5 Exploit Kits

An Exploit Kit is an of off-the-shelf software that can be purchased from the underground market. When installed and configured on a web server, it carries out a malicious campaign targeting innocent victims [37]. One of the vectors for the significant proliferation of cyber-crime on the Web in recent times is the advent of criminal, for-profit, software infrastructure for conducting attacks on endusers. In this infrastructure, Exploit Kits occupy a central role as they facilitate the infection of users through browser compromises. Examples of attacks that are launched through Exploit Kits include drive-by-downloads, spam and denial-of-service. A website hosting an Exploit Kit is usually advertised through URLs disseminated through spam links, search campaigns, social network sites, blogs, or sites hijacked by cyber-criminals. Innocent victims that click on these URLs have their systems compromised through drive-by-download attacks, and the infected hosts are subsequently used for staging further criminal activities.

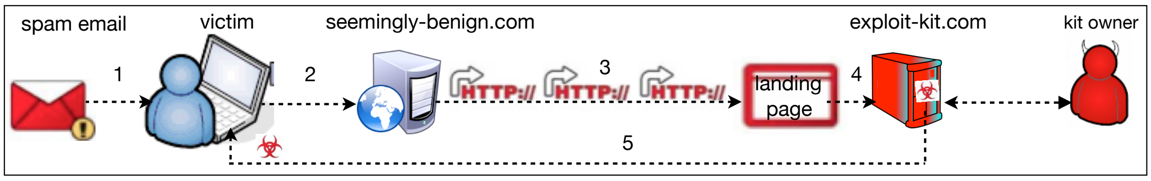


Figure 2.3: Typical workflow in Exploit Kits.

A typical workflow of an Exploit Kit is shown in Figure 2.3 where numbers highlight the major steps. It usually starts with a victim being lured to a URL (e.g., by clicking a link in a spam email as in Step 1) to visit a seemingly benign web page (Step 2). After a series of redirections (Step 3), the victim reaches a landing page(Step 4). The kit then gathers identifying information of the victim in pursuit of vulnerabilities to exploit. At this stage, a kit analyzes the User-Agent information of the victim to identify the type and version number of the operating system, browser, and third-party plugins. If the exploit succeeds, a malicious payload (malware) is silently downloaded and executed on the victim's machine (Step 5). In addition to delivering the exploit payload to the victim, the Exploit Kit also updates the infection statistics accessible to the kit owner (administrator).

In fact, the workflow of Exploit Kits is similar to a drive-by-download attack we already discussed. The difference, in the case of Exploit Kits, however, is that there is a complex and well-organized "business model" that is run by a network of cyber-criminals in the underground marketplace. The fact that most Exploit Kits provide functionalities on details of infection statistics to an Exploit Kit administrator is an indication that the malicious activity is well-planned and infection strategies evolve based on feedback from operational experience in the wild.

2.6 Summary

In fact, the infection chains of different malicious activities we discussed in this chapter have notable overlaps, which entails how difficult it is to detect these aforementioned malicious activities. A typical instance of malicious activity on the Web, for instance, may be initiated when a victim receives a spam email that lures him to give away his bank credentials via a phishing page that mimics the login page of the bank. In another occasion, a victim may be tricked to click on a rogue antivirus update alert where the ultimate landing page is a malware hosting server. A victim may click on a web advertisement banner that could be carrying a malicious code itself or redirects to an Exploit Kit site that fingerprints the client and downloads malware on the victim's machine. The bottom line is that a mix of these attack scenarios happen, or are at least initiated, when an unsuspecting victim *visits a web page.*

Chapter 3

Holistic Detection

In this chapter, we present an approach that addresses the partial analysis and characterization of attack payloads in detecting malicious web pages.

3.1 Overview

Studies show that attacks are getting more and more complex whereby attackers use blended techniques to evade existing countermeasures [48, 73]. More importantly, using static or dynamic analysis approaches in a complementary manner is limited to capturing partial snapshot of a malicious web page.

To this end, we present a *holistic* approach called BINSPECT to address *partial analysis and characterization of attack payloads* in malicious web pages. A key intuition of our approach is a holistic analysis and characterization of malicious payloads in web pages by ensuring the right balance between the fast-and-imprecise static analysis and the slow-and-precise dynamic analysis techniques. To achieve this balance, BINSPECT leverages a combination of static analysis and minimalistic emulation to use supervised learning techniques for detecting malicious web pages pertinent to drive-by-download, phishing, injection, and malware distribution.

While we reuse effective features from previous work (such as from [13],

[14], [18], [54]), we also introduce novel features and enhance existing features to more effectively put apart malicious and benign web pages. In the course of analyzing and characterizing web pages to capture a comprehensive snapshot of malicious web pages, we also ensure that the analysis remains lightweight in terms of its responsiveness and resource consumption.

The following are the contributions of the approach presented in this chapter:

- we developed an approach[1] that combines static analysis and minimalistic emulation to analyze and detect malicious web pages with low performance overhead.

- we introduced 10 novel features and enhanced existing ones to improve their discriminative power in the characterization of malicious and benign web pages.

- we designed, implemented, and evaluated our approach over a large dataset of malicious and benign web pages and demonstrated that our approach is effective in practice.

The rest of this chapter is organized as follows. In Section 3.2, we present details of how we characterize web pages holistically focusing on features. Section 3.3 discusses the details of our approach. A discussion of the implementation and experimental setup is presented in Section 3.4. Experimental evaluation and discussion of results is presented in Section 3.5. We present the summary of this chapter in Section 3.6.

[1]An earlier version and part of this approach has appeared in [28] and [26] respectively.

3.2 Holistic Characterization

Given an unknown web page, BINSPECT analyses and classifies the web page as malicious or benign. To do the analysis and the classification, BINSPECT extracts features from the page under inspection and applies a number of models that evaluate the features extracted from the page. The models are derived from training on a known mix of benign and malicious web pages. The corpus of malicious web pages used in training BINSPECT comprise web pages that launch drive-by-download, phishing, injection, and malware delivery attacks.

In BINSPECT, we have three classes of features used for the statistical characterization of web pages. These feature classes are: *URL* features, *Page-Source* features (HTML and JavaScript), and *Social-Reputation* features. The underlying assumption in using these classes of features is based on the premise that the statistical distribution of feature values of malicious web pages are different from that of benign web pages [13]. In fact, there are some exceptions to some features (e.g., the use of some JavaScript functions like `setTimeout()`) that might appear in both malicious and benign samples. Nonetheless, practice shows that it is highly unlikely to get similar distribution of feature values for the combination of all the features we use in this work. In the rest of this section, we describe the 39 features we extract to build the models we use to classify unknown web pages in BINSPECT.

3.2.1 URL Features

The use of the lexical elements of a URL string has been proved to be effective in identifying benign and malicious URLs, specially for fast detection of spam and phishing URLs [54]. In BINSPECT, we rely on 11 URL features among which we reuse 8 features from prior work ([13], [54]) and we introduce 3 novel features. The URL features we reuse are: length of URL string, length of host name, number of dots (‘.’), number of hyphens (‘-’), number of underscores (‘_’), number of forward slashes (‘/’), number of equal signs (‘=’), and presence of the `client` and/or `server` words in the URL.

By evaluating the F-Score [102] measure of candidate URL features, we found the 3 novel features to be of significant relevance as a high F-score value of a feature indicates a higher potential of the feature to split benign and malicious web pages. These novel features are: *length of the path* in the URL string, *length of the query* in the URL string, and *length of the file-path* in the URL string. Apart from the F-Score, manual inspection also shows that most malicious URLs have abnormally long path and query as compared to benign URLs. In Section 3.5, we will show the experimental evaluation as to the effectiveness of these novel URL features in practice.

In the following, we give context on the statistical variation of URL features based on a measurement study[2] we conducted on a corpus of: benign URLs from Alexa top sites (100, 000 URLs), phishing pages from the PhishTank database (7, 896 URLs), malware-delivery URLs from multiple blacklists (6, 801 URLs), and URLs received via spam emails (119, 833 URLs). For the sake of the following analysis of features, we randomly selected 500 URLs from: Alexa top sites (*Alexa-Set*), PhishTank database (*Phish-Set*), multiple blacklists (*Blacklist-Set*), and Spam Email (*Spam-Set*).

URL Length. This feature is relevant as malicious URLs tend to be unusually long as compared to benign URLs. For example, the average URL length of the *Alexa-Set* and the *Phish-Set* is 15 and 47 respectively with a ratio of 1:3.

Host Name Length. The host name part of malicious URLs is usually

[2]`http://disi.unitn.it/~eshete/pdfs/SVM_MULTI_CLASS_URLs.pdf`

quite complex and longer than its counterpart in legitimate URLs. For instance, the average host name length: in the *Spam-Set* is 15, in the *Phish-Set* is 32, and in the *Blacklist-Set* is 10. On the other hand, the average host name length of the *Alexa-Set* is 8.

Number of Dots. Due to attempts to have URLs hidden within a domain or file paths within a long path, malicious URLs have often times large number of dots. From our measurement, the average number of dots for the *Alexa-Set* is just 1 as compared to an average of 4 and 2 dots for the *Phish-Set* and the *Blacklist-Set* respectively.

Path Length. Although there are legitimate URLs with long paths, it is more common to encounter abnormally long paths in the URL string of malicious web pages. For instance, as opposed to the average path length of 1 in the *Alexa-Set*, the average path length of URLs in the *Phish-Set* is 29 and that of URLs in the *Blacklist-Set* is 12.

File Path Length. Malicious URLs exhibit long and obscurely-generated (sometimes random-looking) file paths because files leading to malicious payloads are stored under complex paths to trick human eyes. The average file-path length of the *Phish-Set* URLs is 46 and that of the *Blacklist-Set* URLs is 19. This average is in contrast to the zero average file-path length of the *Alexa-Set* URLs.

Query Length. The query string of malicious URLs is usually complex and long as compared to query strings in benign URLs. The average query length of the *Phish-Set* URLs is 17 and that of the *Blacklist-Set* URLs is 9 with respect to the zero average query length of the *Alexa-Set* URLs.

Number of Hyphens. This feature is relevant as a significant number of phishing URLs have larger number of hyphens as compared to benign URLs. For instance, of the *Phish-Set* URLs, on average there are 2 hyphens in each URL while it is 1 in the *Alexa-Set* URLs.

Number of Underscores. The occurrence of underscores in malicious

URLs is more frequent than in benign URLs. In the *Alexa-Set* URLs, underscores occurred only 4 times. While in the dataset of the *Phish-Set* URLs and the *Blacklist-Set* URLs, underscores occurred 121 and 42 times respectively.

Number of Forward Slashes. On average, the *Phish-Set* URLs and the *Blacklist-Set* URLs respectively have 5 and 4 forward slashes as opposed to 2 forward slashes in the *Alexa-Set* URLs.

Number of Equal Signs. This feature is correlated with length of query in a URL string. The longer the query length, the higher the number of equal signs to pass query parameters. For instance, the average number of equal signs in the *Alexa-Set* URLs is zero while in the *Phish-Set* URLs and the *Blacklist-Set* URLs is 2 and 1 respectively.

Presence of "client" or "server" Words in URL. words "client" and "server" often appear in URL strings of malicious (specially spam URLs) more often than in benign ones. In the *Spam-Set* URLs, these keywords appeared 5 times while they did not appear at all in the *Alexa-Set* URLs.

3.2.2 Page-Source Features

While previous work (e.g., [13, 42, 54]) extracts HTML and JavaScript features statically, we use an emulated browser to visit the URL, parse and render the HTML, and execute JavaScript on page-load to capture what is manifested by JavaScript code. In this sense, the granularity of HTML features used in BINSPECT is high because the side-effects of the JavaScript code that is executed on page-load enriches the HTML features. Another reason to use an emulated browser is to capture the side-effects of obfuscated JavaScript code that is executed when the page loads because malicious JavaScript is often 'shipped' with a strong shell of obfuscation.

In total, we extract 25 Page-Source features. These are: document

Table 3.1: Summary of features used in BINSPECT.

URL Features		
Feature Name	**Feature Description**	**Remark**
URL Length	character count of a URL string	used in [13], [54]
HostName Length	character count of the host name part of a URL	used in [13], [54]
Number of Dots	count of ’.’ in a URL string	used in [13], [54]
Path Length	length of path in a URL string	novel
File Path Length	length of only file path in the path of a URL string	novel
Query Length	length of query appended in a URL string	novel
Number of Hyphens	count of ’-’ in a URL string	used in [13], [54]
Number of Forward Slashes	count of ’/’ in a URL string	used in [13], [54]
Number of Equal Signs	count of ’=’ in a URL string	used in [13], [54]
Number of Underscores	count of ’_’ in a URL string	used in [13], [54]
Number of Client-Server Words	count of “client” and “server” words in a URL string	used in [13]
Page-Source Features		
Feature Name	**Description**	**Remark**
Document Length	character count of the whole HTML page	used in [13], [42]
Number of Words	count of words in a page	used in [13], [42]
Number of Lines	count of lines in a page	used in [13], [42]
Number of Blank Spaces	count of blank spaces in a page	used in [13], [42]
Number of Blank lines	count of blank lines in a page	used in [13], [42]
Average Length of Words	average length of words in a page	used in [13], [42]
Number of Links	count of href links on a page	used in [13], [42]
Number of Executable Remote Links	count of links pointing to remote executables	refactored
Number of Same-Origin Links	count of links to same origin	refactored
Number of Remote-Origin Links	count of links of remote origin	refactored
Number of Remote JavaScript Files	count of remote JavaScript inclusions	refactored
Number of Hidden Elements	count of all hidden elements	used in [13], [42]
Number of Iframes	count of iframes	used in [13], [42]
Number of Suspicious JavaScript Functions	count of JavaScript functions linked with malicious activities	used in [13], [42]
Number of subString	count of the subString() function	used in [13], [42]
Number of fromCharCode	count of the fromCharCode() function	used in [13], [42]
Number of eval	count of the eval() function	used in [13], [42]
Number of setTimeout	count of the setTemeout() function	used in [13], [42]
Number of document.write	count of the document.write() function	used in [13], [42]
Number of createElement	count of the createElement() function	used in [13], [42]
Number of escape	count of the escape() function	used in [13], [42]
Number of unescape	count of the unescape() function	used in [13], [42]
Number of link	count of the link() function	used in [13], [42]
Number of exec	count of the exec() function	used in [42]
Number of search	count of the search() function	used in [13], [42]
Social Reputation Features		
Feature Name	**Description**	**Remark**
Facebook Share Count	count of unique public shares of a URL on Facebook	novel
Twitter Share Count	count of unique public shares of a URL on Twitter	novel
Google Plus Share Count	count of unique public shares of a URL on Google+	novel

length, number of words, number of lines, number of blank spaces, average length of words, number of links, number of same-origin links, number of different-origin links, number of external JavaScript files, number of hidden elements, number of iframes, and number of suspicious JavaScript functions. Moreover, we include the count of the individual suspicious functions including: `subString()`, `fromCharCode()`, `eval()`, `setTimeout()`, `document.write()`, `createElement()`, `unescape()`, `escape()`, `link()`, `exec()`, and `search()`.

While the Page-Source features we use are mostly from prior work, the way in which we extract these features, i.e., when the emulated browser finishes loading the page, enriches the values with artifacts. Moreover, we refactor existing features for a fine-grained characterization of web pages. For instance, apart from extracting the total number of links on the page, we split links to: number of *same-origin links*, number of *remote-origin links*, number of *remote-origin links to executables*, and number of *external-JavaScript files*. We refactored link features because manual analysis shows that malicious web pages link to remote origins and malicious JavaScript is often downloaded from external domains, for which an aggregate count of links may reduce the discriminative power of the feature.

3.2.3 Social-Reputation Features

The widespread use of social networking websites, such as Facebook, Twitter, and Google Plus, is continuously changing the landscape of online social interaction and reputation building about what is shared online. For instance, search engines rely on social network reputation of URLs to enrich their ranking algorithms because of human intervention in rating URLs [80]. To evaluate if these social-reputation indicators are of use in the characterization of malicious and benign URLs, we examined the statistical distribution of URL-Sharing on Facebook and Twitter as these

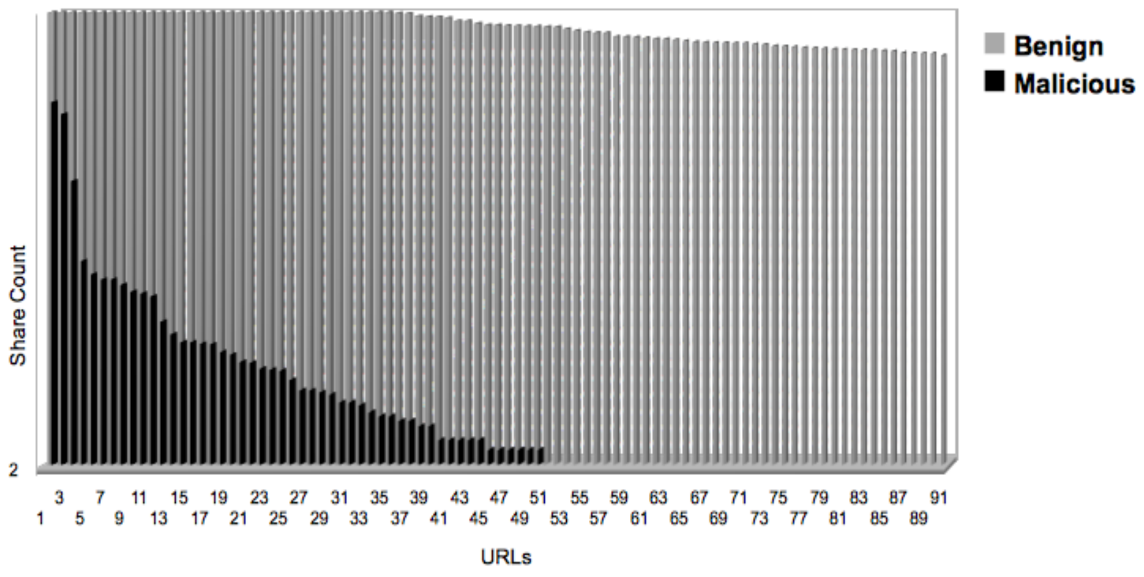


Figure 3.1: Distribution of the top 100 Twitter share-counts for benign and malicious URLs on the training set.

platforms keep track of the public share-count of URLs.

Figure 3.1 shows a statistical separation in distribution of public share-counts for benign and malicious URLs on Twitter over a part of the training set we used for this work. Based on this statistical separation, we introduce three novel features namley: *Facebook Share Count*, *Twitter Share Count*, and *GooglePlus Share Count*, which tell the number of times a URL is publicly shared on Facebook, Twitter, and Google Plus, respectively.

An attentive reader may argue that these features may contribute to false negatives in the case where an attacker publicly shares a malicious URL on a social network and accumulates large share-count within a short period of time. We too recognize this as a legitimate concern. However, as time passes by, the tendency that a malicious URL is circulated across the social networking website reduces or the share-count of the URL is unlikely to increase because of built-in URL analysis and detection techniques in the social networking websites which will flag it as malicious. For instance, Facebook uses the Link Shim[3] system to protect its users from malicious URLs.

3.3 Approach

With the aim of addressing partial analysis and characterization of attack payloads in malicious web pages, Binspect combines static analysis and minimalistic emulation to analyze and characterize web pages using proven existing features and novel features we introduce so as to train multiple models. When provided with an unknown web page, instead of relying on one best model, Binspect uses confidence-weighted majority vote by multiple models to classify web pages as benign or malicious. In a nutshell, Binspect has three major components: *feature extraction*, *multi-model training*, and *confidence-weighted majority vote classification*, as shown in Figure 3.2. In the following, we present a high-level discussion of the components of Binspect.

3.3.1 Feature Extraction

As shown in the upper block of Figure 3.2, we use a labelled dataset of benign and malicious samples (described in Section 3.4) to extract the necessary features that characterize malicious and benign web pages. The URL features are extracted by lexical scanning of the URL string. The Page-Source features are collected by visiting the page via a lightweight emulated browser so as to capture the details of what is rendered (HTML) and executed (JavaScript). For the purpose of collecting Page-Source features, we customized the HTMLUnit [81] headless browser for the emulation and used it with two User-Agent personalities (Internet Explorer 6 and Mozilla Firefox 3). For each URL we visit for feature extraction,

[3]`http://www.facebook.com/note.php?note_id=10150492832835766`

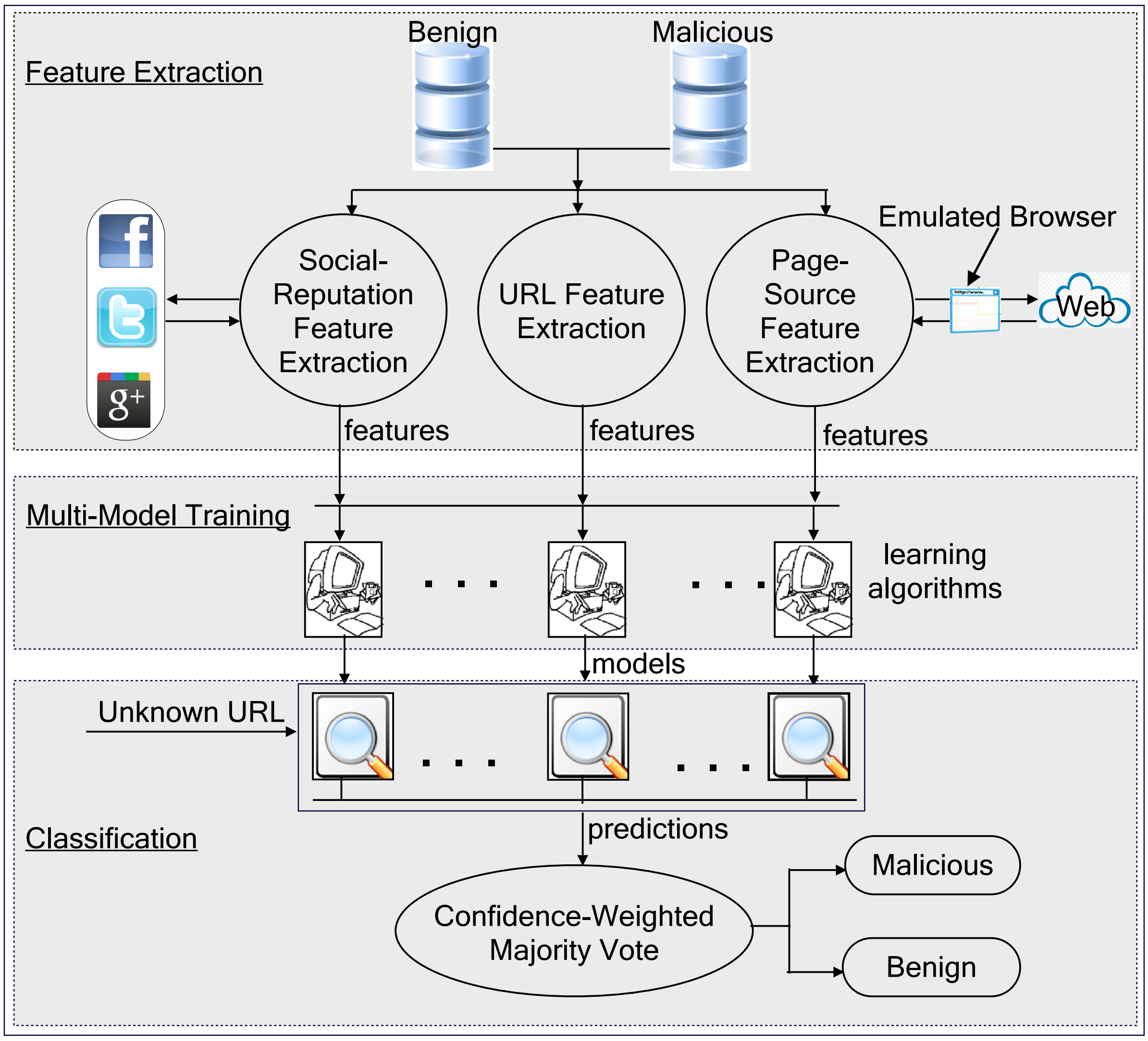


Figure 3.2: BINSPECT System Overview.

a fresh instance of the emulated browser is created to ensure a unique session for each URL. To extract the Social-Reputation features, we used the Facebook Graph API [30], the Twitter URLs API [92], and a custom[4] script for Google Plus. Finally, features extracted from each web page are represented as a vector of the form $[v_1^{(i)}, v_2^{(i)}, ..., v_{n-1}^{(i)}, v_n^{(i)}, class^{(i)}]$ where the $v_k^{(i)}$'s are feature values $(k = 1, .., n)$, n is the number of features, and

[4]A standard API for Google Plus was not available at the time of the experiment for this work.

$class^{(i)} \in \{\texttt{benign}, \texttt{malicious}\}$ is the class label of the ith URL.

3.3.2 Multi-Model Training

In machine learning, supervised learning is the task of inferring a function from a labeled training data [7]. The training data consist of a set of training examples. Each example is a pair consisting of an input object (typically a vector) and a desired output (target) value (also called a class label). A supervised learning algorithm analyzes the training examples and produces an inferred function, which is called a classifier —for discrete output or a regression function —for continuous output. In formal terms, given a set of training examples of the form (x_1, y_1), ..., (x_n, y_n), a learning algorithm seeks a function $h : X \to Y$, where X is the input space and Y is the output space. The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to unseen instances in a way that avoids over-fitting [7].

In BINSPECT, using the extracted features, we train seven supervised learning algorithms namely J48 Decision Tree, Random Tree, Random Forest, Naive Bayes, Bayes Network, Support Vector Machine, and Logistic Regression. At the end of the training, one model for each classifier is maintained as shown in the middle block of Figure 3.2. The mathematical formalism of the supervised learning algorithms we use in this dissertation is discussed in Appendix A.

3.3.3 Confidence-Weighted Majority Vote Classification

For classification of an unknown web page using the learned models, we use the confidence-weighted majority vote algorithm described in [34] that we customized for the purpose of this work (see Algorithm 1) to decide the class of an unknown web page. To deem a page as either malicious or benign, instead of taking the class label that obtains the highest number of votes (the most frequent vote), in our approach, the vote count of the class label is multiplied with the sum of confidence values, strictly probability values, with which the predictions are made by each model (Lines 17, 20, and 23 in Algorithm 1).

Algorithm 1 Confidence-Weighted Majority Vote Classification.

```
1:  $Conf_{benign} \leftarrow 0$
2:  $Conf_{malicious} \leftarrow 0$
3:  $Vote_{benign} \leftarrow 0$
4:  $Vote_{malicious} \leftarrow 0$
5:  for $i = 1 \rightarrow numModels$ do
6:      $features \leftarrow extractFeatures(URL)$
7:      $Vote_i, Conf_i \leftarrow getPredictionWithConfidence(features, Model_i)$
8:      if $Vote_i = benign$ then
9:          $Vote_{benign} \leftarrow Vote_{benign} + 1$
10:         $Conf_{benign} \leftarrow Conf_{benign} + Conf_i$
11:     end if
12:     if $Vote_i = malicious$ then
13:         $Vote_{malicious} \leftarrow Vote_{malicious} + 1$
14:         $Conf_{malicious} \leftarrow Conf_{malicious} + Conf_i$
15:     end if
16: end for
17: if $(Vote_{malicious} \times Conf_{malicious}) > (Vote_{benign} \times Conf_{benign})$ then
18:     $Prediction \leftarrow malicious$
19: end if
20: if $(Vote_{malicious} \times Conf_{malicious}) < (Vote_{benign} \times Conf_{benign})$ then
21:     $Prediction \leftarrow benign$
22: end if
23: if $(Vote_{malicious} \times Conf_{malicious}) = (Vote_{benign} \times Conf_{benign})$ then
24:     $Prediction \leftarrow suspicious$
25: end if
```

The advantage of confidence-weighted majority vote is twofold. First, it minimizes the bias of relying on a single model to do classification as some classifiers perform differently depending on the statistical distributions of an unknown sample. Secondly, it allows comparison of different models and makes the overall result more resistant to evasion attempts by attackers.

3.4 Dataset and Setup

Next, we describe the data collection, dataset preparation, and the experimental procedure we used to evaluate BINSPECT.

3.4.1 Implementation Overview

The URL feature extraction engine is implemented based on the Java URL class. The Page-Source feature extraction engine which interacts with the emulated browser is also implemented in Java as the emulated browser itself, HTMLUnit, is also Java-based. The training and classification are automated using scripts to invoke the respective JAR files for the classifiers in the WEKA machine learning suite, which is also Java-based.

3.4.2 Dataset Source and Dataset

We collected samples from multiple sources for both malicious and benign web pages and divided the dataset into a training and a testing set as shown in Table 3.2. For the malicious dataset, we collected 71,919 URLs from the malware and phishing blacklist of Google Safe Browsing Service [35], the Phishtank database of collaboratively verified phishing pages [68], and the malware and injection attack blacklist of MalwareURL [60]. A dataset of 414,000 benign URLs is also drawn from three popular sources. These are the Alexa Top sites [1], the Yahoo random URL generation service [44], and the DMOZ directory [23].

Table 3.2: Dataset for training and testing BINSPECT.

Purpose	**Benign**	**Malicious**	**Total**
Training	300,000	50,000	350,000
Testing	114,465	21,919	136,384

3.4.3 Experimental Procedure

Using the training set, we extracted the 39 features shown in Table 3.1 of which 3 are Social-Reputation features, 11 are URL features, and the remaining 25 are Page-Source features. When extracting the Page-Source features, we configured the emulated browser to manifest two different browser personalities (Internet Explorer 6 and Mozilla Firefox 3) and we used only the core components of the browser, i.e., the Necko HTML Engine and Rhino JavaScript Engine in order to make the analysis lightweight. We used the WEKA [38] machine learning toolbox to train seven standard classifiers with 10-fold cross validation. These classifiers are J48 Decision Tree, Random Tree, Random Forest, Naive Bayes, Bayes Net, Support Vector Machine, and Logistic Regression. As a sanity check of the dataset, we removed from the training set, i.e., all URLs that were unreachable when visited from the emulated browser. Using the testing set, we run the confidence-weighted majority vote to classify the URLs as benign or malicious.

3.5 Evaluation

We now evaluate BINSPECT from the standpoint of its accuracy, significance of the features we introduced, its performance overhead, and its immunity to possible evasion.

3.5.1 Metrics

To evaluate the effectiveness of detection models, we use three established metrics [7]. These are: Detection (classification) Rate (DR), False Positive Rate (FPR), and False Negative Rate (FNR). Suppose that a dataset contains a total of M_t examples of malicious URLs and a total of B_t benign URLs. Moreover, let us assume that M_c denotes the correctly classified malicious URLs out of M_t (M_c is called True Positive). Similarly, let B_c denote the correctly classified benign URLs out of B_t (B_c is called True Negative). Equations 3.1, 3.2, and 3.3 are used to compute DR, FPR, and FNR respectively.

$$DR = \frac{M_c + B_c}{M_t + B_t} \tag{3.1}$$

$$FPR = \frac{M_t - M_c}{M_t} \tag{3.2}$$

$$FNR = \frac{B_t - B_c}{B_t} \tag{3.3}$$

An effective detection model is characterized by high DR, low FPR, and low FNR. In this chapter and throughout this dissertation, unless explicitly specified, the meanings of DR, FPR, and FNR correspond to Equations 3.1, 3.2, and 3.3 respectively.

3.5.2 Analysis of Models

To decide the best combination of classifiers in Binspect, we evaluated the 7 classifiers in terms of accuracy, False Positive Rate (FPR), and False Negative Rate (FNR). Figures 3.3, 3.4, 3.5, and 3.6 show performance evaluation of the classifiers over the training set across the four classes of features, i.e., all features, URL features, Page-Source features, and Social-Reputation features respectively. As shown in Figure 3.3, training on all

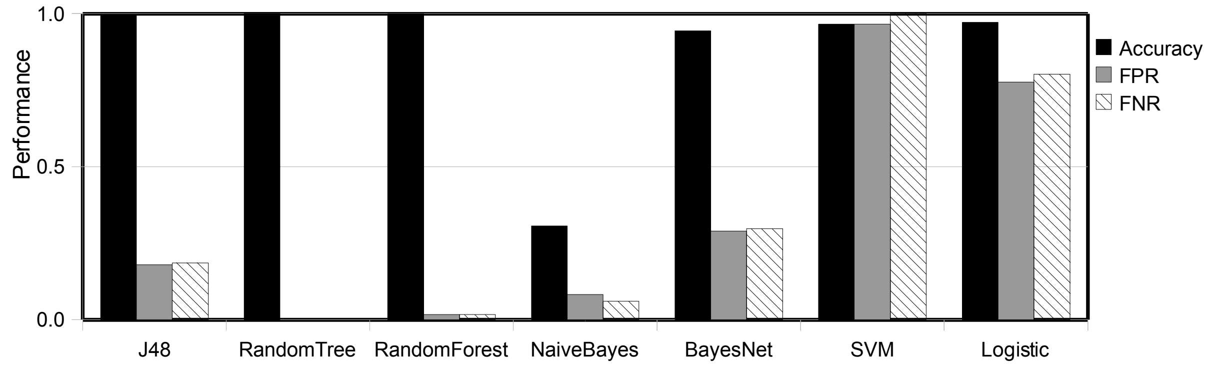


Figure 3.3: Binspect: Evaluation of classifiers with all features.

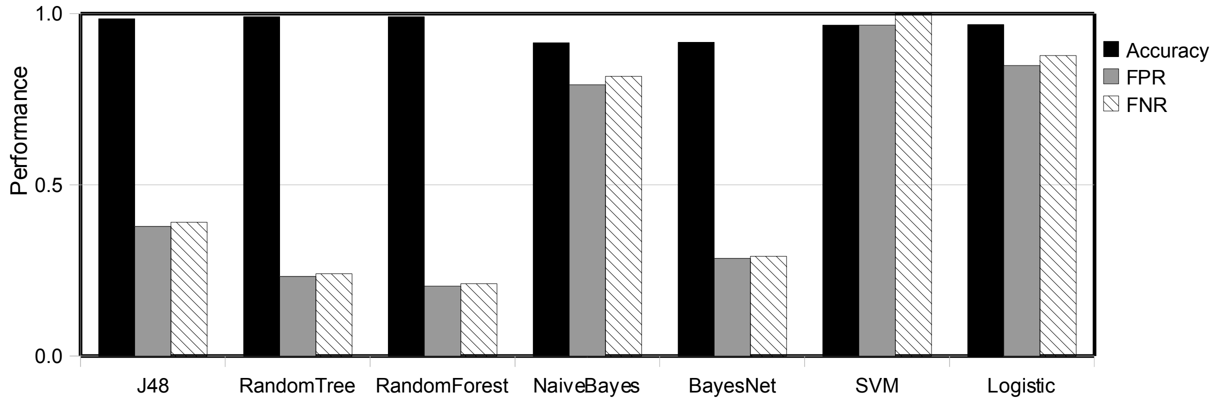


Figure 3.4: Binspect: Evaluation of classifiers with URL features.

the features suggests that tree-based classifiers outperformed the other classifiers. In particular, the Random Tree classifier achieved 100% accuracy, 0% FPR, and 0% FNR.

We also evaluated how the classifiers perform on individual feature classes and the results suggest that some classifiers perform way better than the union of the features. For instance, accuracy of Naive Bayes increased by 30% (Figure 3.4) on URL features probably because the URL features have a statistical distribution that fits into the high degree of independence assumed in the algorithm.

Another interesting insight from Figure 3.6 is the high FNR of all the classifiers on social-reputation features which is attributed to the fact that

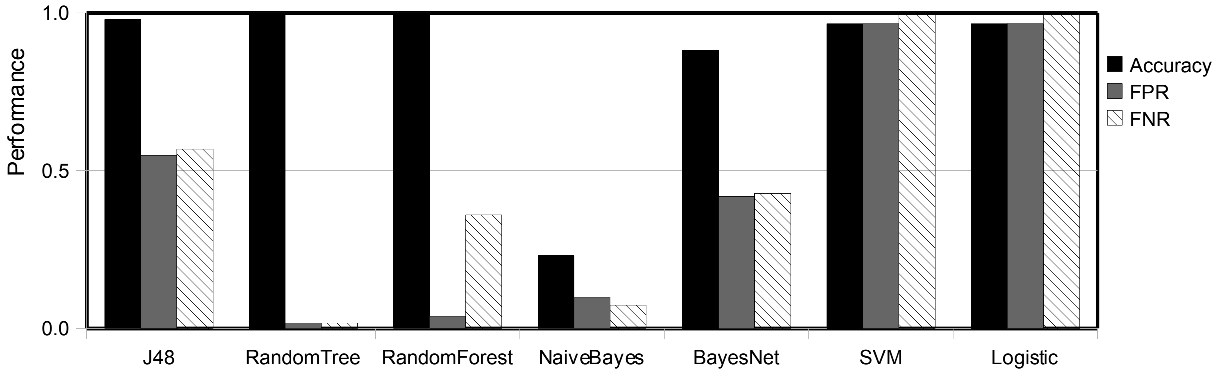


Figure 3.5: BINSPECT: Evaluation of classifiers with Page-Source features.

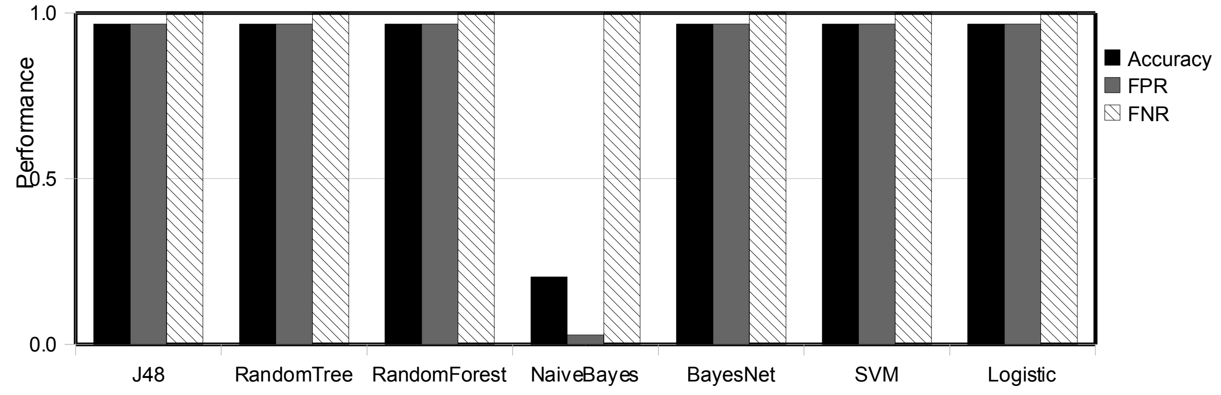


Figure 3.6: BINSPECT: Evaluation of classifiers with Social-Reputation features.

malicious URLs which have higher share-count are likely to be misclassified as benign, suggesting that it is more effective to combine social-reputation features with other features to increase their predictive power. In general, the overall classification performance is better on all the features than the individual feature classes with the exception of Naive Bayes, which did not perform well in most cases (see Figures 3.3, 3.5, and 3.6).

3.5.3 Significance of New Features

To verify whether the new features are of predictive importance in enhancing the accuracy of detecting malicious web pages, we compared the classification accuracy, FPR, and FNR of the classifiers with and without our newly introduced (enhanced) features on the training set. As shown in Table 3.4, the new features, particularly the new URL features, improved the overall performance of 5 of the 7 classifiers (J48, Random Forest, Naive Bayes, Bayes Net, and Logistic Regression) shown with ($\uparrow$) for accuracy and with ($\downarrow$) for FPR and FNR. The new Page-Source features improved the overall performance of only 2 classifiers (Random Forest and Logistic Regression). Social-Reputation features have also improved the overall classification accuracy of Random Forest, Bayes Net, and Logistic Regression classifiers. Not surprisingly, the performance of Naive Bayes has not improved much with the new features as its overall performance is also very low.

In addition to the individual contribution of the new features, we also measured the overall improvement in accuracy of the classifiers as a result of the new features. This evaluation is summarized in Table 3.3. The new features improved the accuracy of 4 of the 7 classifiers with improvements in the range 0.21% to 3.08%. Among the remaining 3 classifiers, on 2 (Random Forest and Support Vector Machine), the new features seem to have no contribution on accuracy. The Random Tree classifier is an exception in this case as its accuracy was 100% even without the new features. Out of curiosity, we measured its accuracy with the new features and it remained the same, which most probably implies that this is the best classifier given the feature set and the dataset we used for training.

Table 3.3: BINSPECT: Overall Contribution of new features on the accuracy of classifiers.

Classifier	**Without new (%)**	**With new(%)**	**Change(%)**
J48 Decision Tree	98.97	99.27	↑ 0.30
Random Tree	100.0	100.0	–
Random Forest	99.94	99.94	–
Naive Bayes	28.16	30.62	↑ 2.46
Bayes Net	91.28	94.36	↑ 3.08
SVM	96.62	96.62	–
Logistic Regression	96.94	97.15	↑ 0.21

3.5.4 Classification Accuracy

For testing, we used all the classifiers except Naive Bayes due to its poor performance on the training set. Table 3.5 shows the overall classification accuracy of BINSPECT over the testing set. We submitted the same testing set to WEPAWET [93] to compare BINSPECT with a publicly deployed analysis and detection service. As can be seen from Table 3.5, BINSPECT correctly classified 97.81% of the test set with a FPR of 0.189 and FNR of 0.011. On the other hand, WEPAWET achieved a classification acuracy of 61.62% on the same testing set. The only speculation behind the low performance of WEPAWET in our opinion is the difference in the class of features we use in BINSPECT which span URL, HTML, JavaScript, and social reputation scores while WEPAWET uses emulation to dynamically analyze web pages.

The high accuracy of BINSPECT and its very low FNR on the testing set

Table 3.4: BINSPECT: Performance of classifiers with and without new features on the training set.

Classifier	**Accuracy(%)**	**False Positive Rate**	**False Negative Rate**
Without new features			
J48 Decision Tree	98.97	0.260	0.268
Random Tree	100.00	0.000	0.000
Random Forest	99.94	0.017	0.017
Naive Bayes	28.16	0.122	0.100
Bayes Net	91.28	0.381	0.391
Support Vector Machine	96.62	0.966	1.000
Logistic Regression	96.94	0.845	0.874
With new URL features			
J48 Decision Tree	98.98(↑)	0.254(↓)	0.262(↓)
Random Tree	100.00	0.000	0.000
Random Forest	99.95(↑)	0.014(↓)	0.014(↓)
Naive Bayes	46.45(↑)	0.184(↑)	0.171(↑)
Bayes Net	93.32(↑)	0.350(↓)	0.360(↓)
Support Vector Machine	96.62	0.966	1.000(↑)
Logistic Regression	97.05(↑)	0.798(↓)	0.825(↓)
With new Page-Source features			
J48 Decision Tree	98.93(↓)	0.260	0.268 ↑)
Random Tree	100.00	0.000	0.000
Random Forest	99.95(↑)	0.014(↓)	0.014(↓)
Naive Bayes	28.08(↓)	0.119(↑)	0.095(↓)
Bayes Net	90.85(↓)	0.381(↓)	0.391(↓)
Support Vector Machine	96.62	0.966	1.000
Logistic Regression	96.96(↑)	0.0842(↓)	0.871(↓)
With new Social-Reputation features			
J48 Decision Tree	98.99(↑)	0.265(↑)	0.274(↑)
Random Tree	100.00	0.000	0.000
Random Forest	99.95(↑)	0.014(↓)	0.014(↓)
Naive Bayes	26.69(↓)	0.075(↓)	0.051(↓)
Bayes Net	93.29(↑)	0.353(↓)	0.362(↓)
Support Vector Machine	96.62	0.966	1.000
Logistic Regression	97.06(↑)	0.806(↓)	0.834(↓)

proves that our approach is effective at analyzing and detecting malicious web pages in a holistic manner with low performance overhead while covering malicious web pages leading to drive-by-download, phishing, injection, and malware delivery.

Table 3.5: Performance of BINSPECT in comparison with a public malicious web page analysis and detection service on the testing set.

Measure	BINSPECT	WEPAWET [93]
Classification Accuracy	97.81%	61.62%
False Positive Rate	0.189	0.983
False Negative Rate	0.011	0.073

3.5.5 Performance Overhead

The experimental infrastructure we used is an Intel dual-core 2.66GHz CPU and 64-bit MacOSX operating system with 8GB of memory. Under this computational resource, the average time it takes to train a classifier is only 1.51 seconds. BINSPECT took between 3 to 5 seconds (under variable system load) to analyze and detect a single page, which is an acceptable overhead given the fact that part of the analysis requires rendering the page in an emulated browser. Unfortunately, we could not compare performance overhead of BINSPECT with WEPAWET due to the long delay it took to get back the results from WEPAWET server which uses queueing to process batch requests for analysis.

3.5.6 Resilience to Evasion

Given the holistic nature of our approach, we claim that BINSPECT is not easily evadable. However, by closely inspecting the features we use, there are a few things an attentive attacker could try to evade our analysis and detection technique. One method an attacker might use is to craft a benign-looking URL so as to imitate lexical aspects of benign URLs, which makes the URL features less useful in discriminating benign URLs from malicious ones.

Another approach is for the attacker to use a highly obfuscated client-side code (e.g., JavaScript) that is executed after page-load. In such a case, Binspect is likely to be partly tricked because we only consider side-effects of obfuscation (if any) on page-load.

With regards to the Social-Reputation features, the major risk is that the attacker might lure users on social networks to publicly share a link to a malicious URL in order to collect reputation scores that could mislead Binspect. Even in this case, the luring would not last long because the built-in URL scanning facility of the social networking platform would most likely discover the maliciousness of the URL.

In general, it requires a great deal of effort from the attacker's side to completely bypass Binspect as it is quite difficult for the attacker to take control of the three complementary classes of features used in our approach and due to the nature of the classification that relies on weighted-confidence of each classifier.

3.6 Conclusions

Existing techniques for detecting malicious web pages are effective at detecting specific attack types. However, they are limited to partial snapshot of a malicious payload which limits their ability to cope up with the blended and complex threats posed by malicious web pages.

In this chapter, we presented Binspect, a holistic approach to defend users against malicious web pages by leveraging static analysis and lightweight emulation combined with supervised learning. We have shown through large scale evaluation that BINSPECT is effective at precisely detecting malicious web pages with very low false signals. Moreover, the new features we introduced are relevant enough in improving the performance of the analysis and detection of malicious web pages. Our experiments suggest that BINSPECT incurs acceptable overhead cost to analyze web pages in a realistic scenario due to effective features reused from prior work and due to novel features introduced in this work.

Chapter 4

Evolution-Aware Detection

In this chapter, we address a problem that challenges the resilience of learning-based analysis and detection of malicious web pages, i.e., evolution of web page artifacts.

4.1 Overview

To derive detection models for malicious web pages, distinguishing artifacts of benign and malicious web pages are analyzed. Nevertheless, these artifacts are under constant evolution [31, 48, 73]. The evolution is typically two-sided. On the one hand, cyber-criminals constantly revamp their strategies to craft attack payloads in malicious web pages not only aimed at making attacks more complex but also to evade existing countermeasures [48, 73]. On the other hand, benign web pages evolve because of new content, new functionalities, or changes to the underlying technologies used in building the web pages [31]. Both types of the evolution impact the precision of the detection techniques, rendering detection models out-of-date, in turn, resulting in malicious web pages that escape detection.

To this end, we present an approach called EINSPECT, that leverages evolutionary searching and optimization to align learning-based detection models with evolution of web page artifacts with the goal of more precise analysis and detection of malicious web pages. To achieve this goal, EINSPECT starts with an initial population of candidate models trained using standard learning algorithms based on discriminative features extracted from: URL string, HTML content, JavaScript code, and reputation metadata of web pages on social networking websites. It then uses a Genetic Algorithm (GA) to automatically search and optimize the *best* combination of features and learning algorithms. We call this best combination the *fittest model*, which embraces the evolution of web page artifacts into the analysis and detection task. Using the *fittest model*, it detects unknown web pages to flag them as malicious or benign.

The key idea of our approach is that instead of training multiple classifiers on a given feature set and pick the classifier(s) with the best performance, as in most learning-based approaches (such as [13], [18], [54], [55], and [14]), in EINSPECT we exhaustively evaluate the *best* combination of features and learning algorithms using a GA.

The contributions of the approach presented in this chapter are the following:

- an evolution-aware approach[1] to embrace the constant evolution of the underlying artifacts of both benign and malicious web pages into the learning-based analysis and detection task using a GA.
- design, implementation, and evaluation of the approach on a fairly large-scale dataset.

The remainder of this chapter is structured as follows. In Section 4.2, we give an experimental evidence to support our claim that web pages are under constant evolution and this evolution impacts the effectiveness of detection models. Section 4.3 presents the key intuition and technical

[1]An earlier version of the work in this chapter and part of it has been presented in [29] and [26] respectively.

details of our approach. The implementation, dataset description, and experimental protocol are discussed in Section 4.4. Section 4.5 discusses the evaluation results of EINSPECT. Finally, we present a summary of this chapter in Section 4.6.

4.2 A Case Study

The vast majority of existing approaches which use machine learning (e.g., [54], [18], [42], [55], [75], [105], [14], [89]) report the best performance of a detection model, let us say M_t, at time t with dataset D_t on feature set F_t with feature values V_t. The question, however, is *"What will be $D_{t'}$, $F_{t'}$ and $V_{t'}$ at a later time t', and what will be the implication on the efficacy of M_t?"*. In response to this question, we present some insights from a measurement study to demonstrate the impact of the evolution of web page artifacts on the precision of detection models using real-life data. Using the insights in this section as a springboard, we formulate our approach in the next section.

4.2.1 Context

To proof-check the research problem, we measured the classifiers' accuracy and feature-value trend on real-world samples for both benign and malicious web pages. For benign web pages, we used the top 100 Alexa sites on July 17, 2012. For the malicious ones, we used 54 malicious web pages from the Google phishing and malware blacklist. Before using the samples in this experiment, we validated the malicious web pages using McAfee Site Advisor[62]. It is worth noting that even though we started with about 200 malicious web pages on the first day, only 54 web pages remained active for the duration of the study, for which we could collect features on a daily basis in parallel with the benign ones.

Using the dataset, during July 17-31, 2012 (for 15 consecutive days), we configured an emulated headless browser based on HtmlUnit [81] that disguised itself as Mozilla Firefox 6 to render each web page and extract a total of 25 (HTML and JavaScript) features we presented in Chapter 3. Using the features collected daily, we trained the Decision Tree and Naive Bayes classifiers to examine their daily performance by measuring their classification accuracy and false positive rate with 10-fold cross-validation on the training sample. In addition, we also measured the mean feature value trend over the training dataset to examine the evolution pertinent to each feature.

4.2.2 Insights

Decision Tree: As shown in Figure 4.1, within the 15 days period the classification accuracy fluctuated between 94.1% (on the 13^{th} day) and 98.02% (for most of the days). The false positive rate of the classifier is even more variable between 0.191 and 0.950. On the 10^{th} day, the highest false positive rate (0.191) is encountered. Manual analysis of the web pages shows that, about 19 of the 54 malicious samples redirected the emulated browser to a benign web page. In 2 of the 19 cases, the redirection was to the home page of Google (`http://www.google.com`).

Naive Bayes: As compared to the Decision Tree classifier, the Naive Bayes classifier has relatively lower classification accuracy (with a maximum of 79.2%). However, its false positive rate is way lower than that of Decision Tree (see Figure 4.2). For instance, for Decision Tree, the average false positive rate is 0.379 while for Naive Bayes it is 0.027.

Discussion. As can be seen from Figure 4.1 and Figure 4.2, the variation in classification accuracy and false positive rate across different classifiers and within the same classifier over an extended period of time proves that different learning algorithms respond differently to the underlying changes in features of the web pages. The evolution in model performance is not only about quantitative changes in accuracy but also of qualitative changes in the discriminative significance of the underlying features used in building the models. Infact, the features also evolve when web pages change for the good or the evil purpose. To verify this issue, we examined the average feature values over the period of the case study.

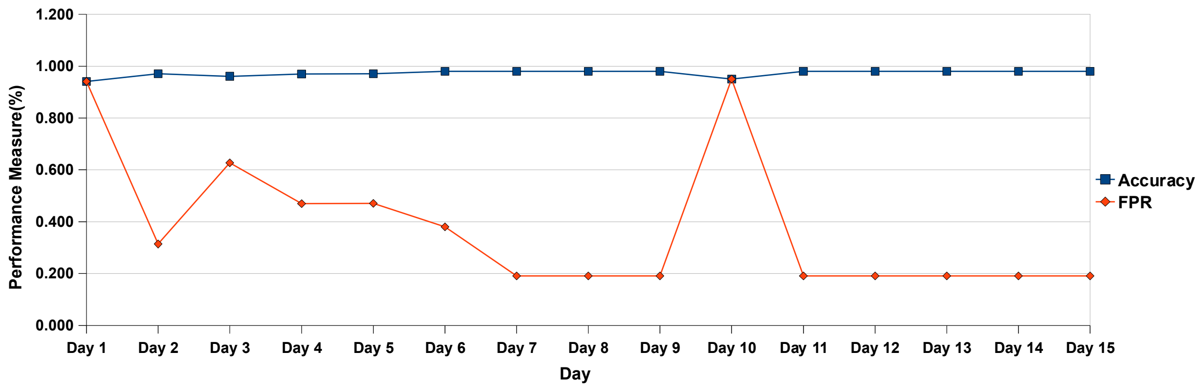


Figure 4.1: Decision Tree classifier performance evolution over a period of 15 days with daily feature extraction and classifier training.

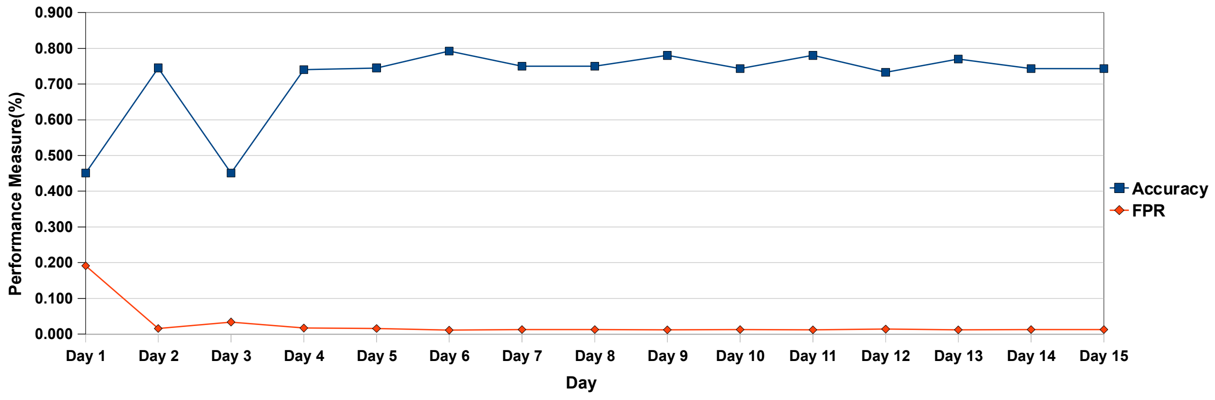


Figure 4.2: Naive Bayes classifier performance evolution over a period of 15 days with daily feature extraction and classifier training.

Figures 4.3 and 4.4 show the mean value trend of HTML features and Figure 4.5 shows the mean value trend of JavaScript features. While changes in some feature values (e.g., document length, number of words)

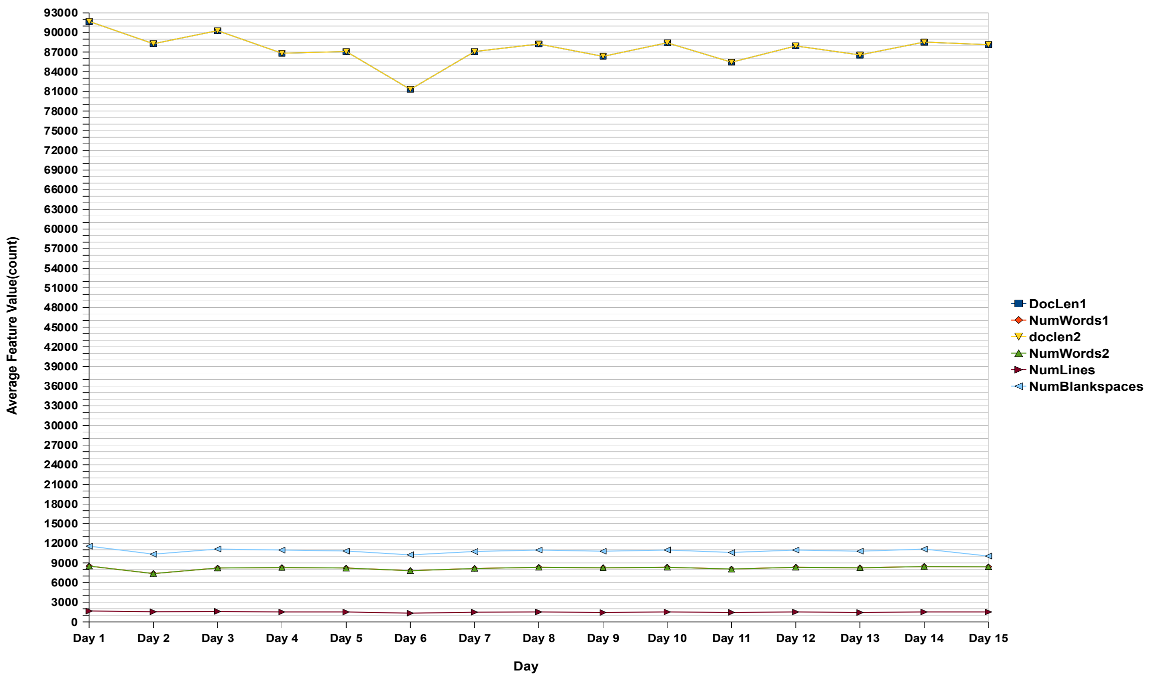


Figure 4.3: Feature value evolution of HTML features over a period of 15 days. These features are separately plotted for the sake of clarity.

might happen for benign purposes (e.g., new text content on a web page), there are specifically suspicious changes in feature values which are linked to malicious activities. For example, apart from the trends shown in Figures 4.3 and 4.4, and Figure 4.5, manual analysis of the feature values reveals that dangerous JavaScript functions mostly linked with attacks involving obfuscated malicious JavaScript code (e.g., `eval()`, `escape()`, `unescape()`), time-bomb attacks (e.g., `setTimeout()`), and shell-code execution (e.g., `exec()`) are manifested more frequently in malicious web pages than in benign ones.

In summary, the observations show that it is not sufficient to just retrain a detection model with new dataset. To cope with evolving web page artifacts, it requires a more systematic and *evolution-aware* method.

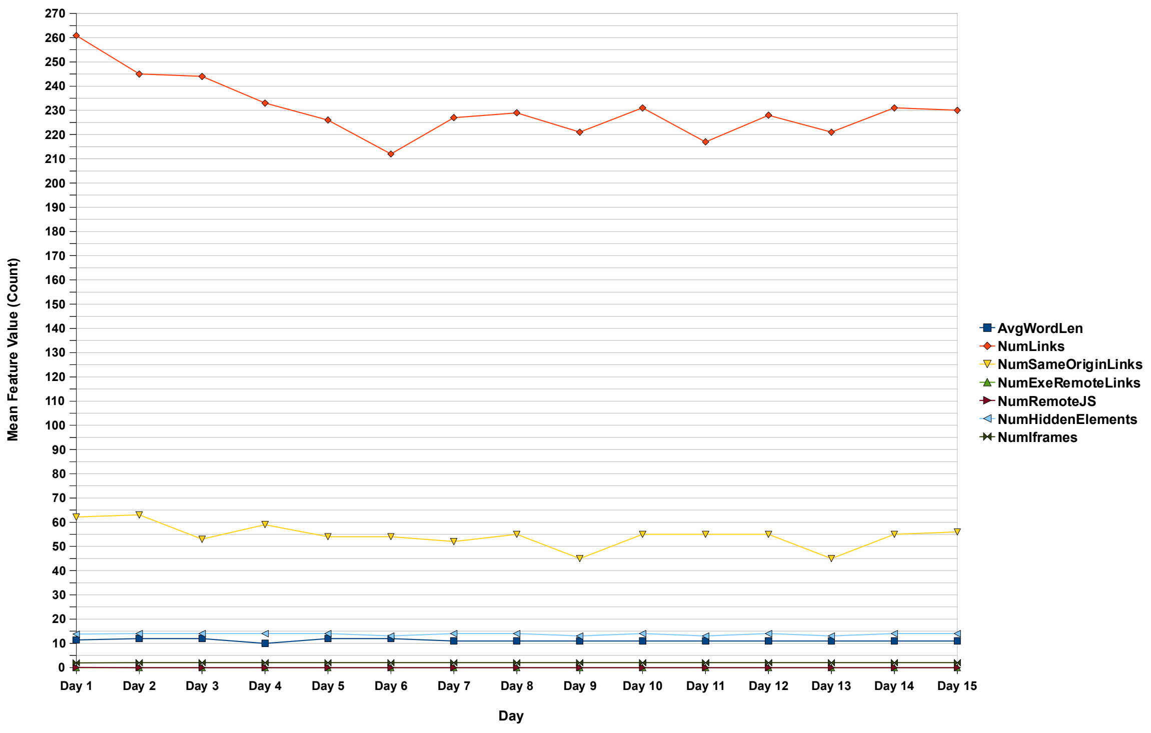


Figure 4.4: Feature value evolution of HTML features over a period of 15 days. These features are separately plotted for the sake of clarity.

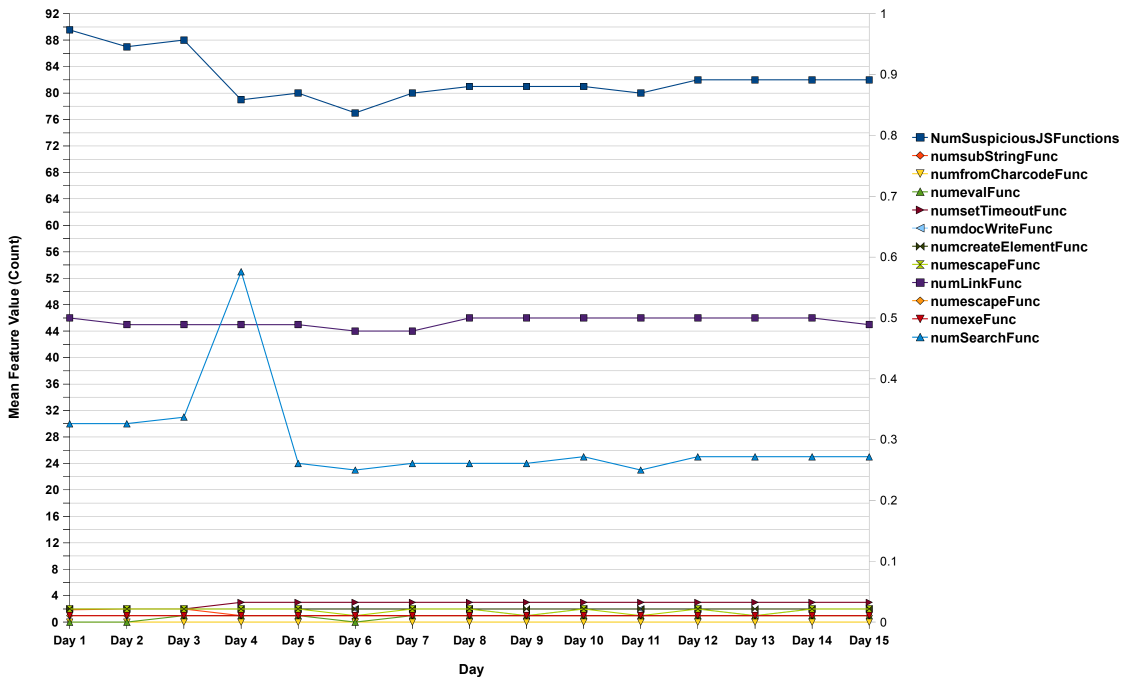


Figure 4.5: Feature value evolution of JavaScript features over a period of 15 days.

4.3 Approach

The principal idea in our approach is to improve learning-based detection of malicious web pages and reinforce it with evolutionary searching and optimization to build more accurate detection models. By doing so, we ensure that the models learned are aligned with the evolution of web page artifacts, especially of malicious web pages. The approach is evolution-aware by design and scalable-enough to automatically evaluate how robust the new model is with respect to the changes in the threat landscape and healthy evolution of web pages.

Figure 4.6 shows the high-level operational framework of EINSPECT. It is organized into four components: *Crawling and Feature Extraction*, *Candidate Models Generation*, *Evolutionary Searching and Optimization*, and *Detection*. EINSPECT relies on a crawler that bases its crawling on seeds from trending topics on the Web to harvest potentially malicious URLs. The result of the crawling is enriched with samples from publicly-known blacklists (for malicious) and whitelists (for benign). The feature extraction engine extracts potentially relevant features pertinent to URL string, HTML, JavaScript, and reputation of web pages on social networking websites. These features are reused from Chapter 3.

Using the extracted features, the Candidate Models Generation component creates a randomly clustered set of features for which multiple classifiers are trained to generate candidate models. The Evolutionary Searching and Optimization component takes the candidate models as initial population and iterates over a series of generations (by applying selection, crossover, and mutation) to ultimately select the fittest model(s) based on which unknown web pages are detected. This whole workflow in EINSPECT is repeated whenever there is a change in (1) feature sets (2) learning algorithms or (3) dataset used to generate candidate models.

In the rest of this section, we describe our approach in more detail by shading light on how an evolutionary technique, a GA in particular, is leveraged to improve the precision of detecting malicious web pages.

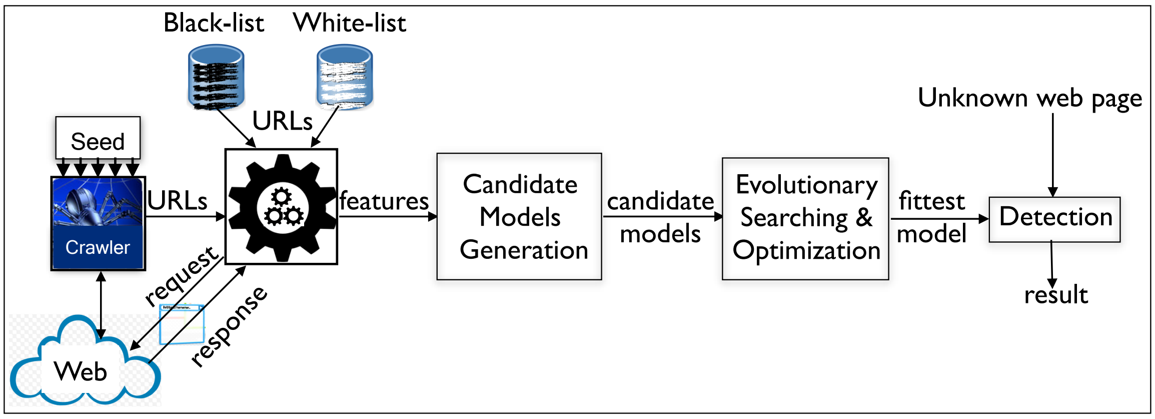


Figure 4.6: Operational Framework of EINSPECT.

4.3.1 Crawling and Feature Extraction

The crawler is periodically fed with seed URLs. Trending topics from Google, Twitter, and Wikipedia are used as search queries to collect the seed URLs. In order to enrich the data collected using the crawler, we also use publicly-endorsed and constantly-updated blacklists (e.g., Google blacklist, PhishTank database) and whitelists (e.g., Alexa Top Sites, DMOZ directory). Before using the collected web pages for our experiments, we verify them using a custom-built honeyclient to discard irrelevant samples such as unreachable pages.

Features we use in EINSPECT are those described in Chapter 3 in which we demonstrated the effectiveness of 10 new features and 29 existing features reused from the literature [13, 54]. The characterization of web pages is based on 11 URL string features, 10 HTML features, 15 JavaScript features, and 3 features on reputation metadata of URLs in social networking websites.

A point worth-mentioning about the HTML and JavaScript features is that, unlike most prior work which extracts HTML and JavaScript features statically [13, 54], we use an emulated browser to render the page first and then run feature extraction to capture JavaScript artifacts generated on page-load and the side-effects of the execution on the HTML content and structure generated in effect. Not to repeat ourselves here, we suggest the interested reader to refer to Chapter 3 for the detailed description about large-scale evaluation as to the effectiveness of the features we reuse in EINSPECT.

4.3.2 Candidate Models Generation

The candidate models generation step is described in Algorithm 2. Given a labeled set of web pages called the training set $T = \{URL_i | i = 1, ..., n\}$ of size n, for each URL_i in T, we extract d classes of features (e.g., URL string features (F_1), HTML features (F_2), JavaScript features (F_3), reputation features (F_4)). Then, for each feature class F_j ($j = 1, ..., d$), the extracted features are encoded as feature vectors to a set of supervised learning algorithms $\{A_k | k = 1, ..., m\}$ to generate a $model_{j,k}$ over the training set T. At the end, for d distinct feature classes, and m distinct supervised learning algorithms, a total of d×m candidate models are generated. Put differently, the candidate model generation could also be illustrated as a tree with the training set at the root (see Figure 4.7). In the tree structure, traversing the tree from the root (T) to any one of the leaves results in a unique candidate model.

In practice, not all of the generated candidate models are good enough to make it to the initial population of candidate models which are used to initialize the GA. By setting a threshold on the accuracy of the models, those models with accuracy below a certain threshold are eliminated to reduce noise from the set of candidate models right from the outset.

Algorithm 2 Candidate Models Generation.

```
1: $d$ :# of feature classes
2: $m$ :# of learning algorithms
3: $T \leftarrow getTrainigSet()$
4: for j:=1 ; j $\leq$ d ; j++ do
5:     $FeatureValues_j \leftarrow extractFeatures(\mathrm{T}, F_j)$
6:     for k= 1 ; $k$ $\leq$m ; k++ do
7:         $model_{j,k} \leftarrow generateModel(FeatureValues_j, A_k)$
8:     end for
9: end for
```

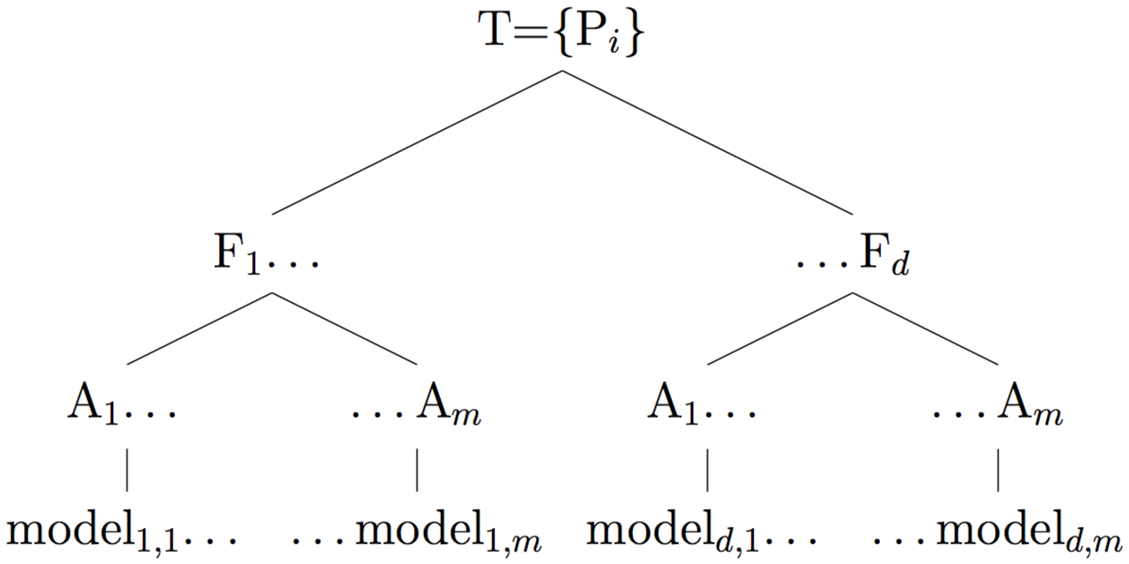


Figure 4.7: Candidate models generation tree structure.

4.3.3 Evolutionary Searching and Optimization

The key intuition behind GA is that given a problem for which there are a number of good solutions of which the best one is unknown, the alternative solutions (among other possible solutions) are evolved toward the best solution(s) [104]. The individual solutions in a GA are called *chromosomes* and the whole collection of the solutions is called a *population*.

A generic GA is shown in Algorithm 3. Initially, a GA starts with a random set of chromosomes called the *initial population* (line 1 in Algo-

Algorithm 3 Genetic Algorithm (general).

```
1: $P \leftarrow setInitialPopulation()$
2: repeat
3:     $N \leftarrow size(P)$
4:     $P' \leftarrow \{\}$
5:     repeat
6:         $chromosome_1 \leftarrow select(P)$
7:         $chromosome_2 \leftarrow select(P)$
8:         $offspring_1, offspring_2 \leftarrow crossover(chromosome_1, chromosome_2)$
9:         $offspring_1 \leftarrow mutate(offspring_1)$
10:        $offspring_2 \leftarrow mutate(offspring_2)$
11:        $P' \leftarrow P \cup \{offspring_1, offspring_2\}$
12:    until size(P') = N
13:    $P \leftarrow P'$
14: until termination-criteria = true
```

rithm 3). In each generation, a GA applies three genetic operations (lines 6-11 in Algorithm 3) in the order: *selection* (picking the best individuals to breed), *crossover* (the breeding), and *mutation* (apply certain changes to new individuals so as to favor diversity in the population). The selection operation relies on a *fitness score*, obtained using a *fitness function*[2], which is assigned depending on how "good" a chromosome is. The measure of the "goodness" of a chromosome is specific to the problem at hand. The selection-crossover-mutation operations continue until the fittest chromosome(s) surviving the evolution is (are) obtained as optimal solution(s). Commonly, the GA terminates (Line 14 in Algorithm 3) when either a maximum number of generations is reached, or a satisfactory fitness level is reached for the population.

Algorithm 4 shows the GA adapted in the context of our approach. While the basic GA workflow is the same, in EINSPECT, we have to train the newly created offsprings (models) at each iteration (Line 9 in Algorithm

[2]Fitness function: a particular type of objective function that is used to summarize, as a single figure of merit, how close a given solution is to achieving the expected aims.

4). The training is required in order to compute the fitness (Line 10 in Algorithm 4) of the new models and compare them with the rest of the models in the population, for conducting selection.

Algorithm 4 GA-Based Model Searching and Optimization.

```
1: $initPop \leftarrow genCandidateModels(F, A, T)$
2: $curPop \leftarrow genRandomPop(initPop, popSize)$
3: **repeat**
4:    **repeat**
5:       $model_1, model_2 \leftarrow select(curPop, fitness)$
6:       $offspring_1, offspring_2 \leftarrow crossover(model_1, model_2)$
7:       $offspring_1 \leftarrow mutate(offspring_1)$
8:       $offspring_2 \leftarrow mutate(offspring_2)$
9:       $train(offspring_1, offspring_2, F, A, T)$
10:      $computeFitness(offspring_1, offspring_2)$
11:      $newPop \leftarrow getNewPop(curPop, offspring_1, offspring_2)$
12:      $curPop \leftarrow newPop$
13:      $gen \leftarrow gen + 1$
14:   **until** $fittestModelFound$ **or** $gen = maxGen$
15: **until** $executionTime = maxExecTime$
```

Running Example. To illustrate how the major genetic operations are contextualized for the purpose of our approach, suppose that we have a labelled training set T = {[URL$_1$, benign], [URL$_2$, malicious], [URL$_3$, benign], [URL$_4$, malicious]} and two feature sets F$_1$ = {urlLen, pathLen, remoteLnk} and F$_2$ = {zeroSizeIframes, nativeJSFunc}. Let the algorithms to generate the models be Decision Tree (DT) and Support Vector Machine (SVM).

Using Algorithm 2 on T, we obtain number of feature classes to be d=2 and number of algorithms to be m=2. The number of candidate models is d×m=2×2=4 meaning that the combination of the feature sets (F$_j$'s — j=1, ..., d) and the algorithms (A$_k$'s — k =1, ..., m) generates the following 4 candidate models:

$$\mathbf{model}_{1,1} = \begin{bmatrix} T & urlLen & pathLen & remoteLnk & DT \end{bmatrix}$$

$$\mathbf{model}_{1,2} = \begin{bmatrix} T & urlLen & pathLen & remoteLnk & SVM \end{bmatrix}$$

$$\mathbf{model}_{2,1} = \begin{bmatrix} T & zeroSizeIframes & nativeJSFunc & DT \end{bmatrix}$$

$$\mathbf{model}_{2,2} = \begin{bmatrix} T & zeroSizeIframes & nativeJSFunc & SVM \end{bmatrix}$$

Chromosome. In EINSPECT, what we refer to as a chromosome is encoded in a form of an $n \times m$ matrix $M_{n \times m}$ where n is the size of the training set, m is the number of features, and m$[i, j]$ is the feature value of the j^{th} feature of the i^{th} training example for a given learning algorithm A$_k$.

For instance, a chromosome representation for model$_{1,1}$ of the running example could look like the following:

$$\mathbf{model}_{1,1} = \begin{bmatrix} \textbf{URL} & \textbf{Alg} & \textbf{urlLen} & \textbf{pathLen} & \textbf{remoteLnk} \\ 1 & DT & 56 & 21 & 5 \\ 2 & DT & 123 & 68 & 25 \\ 3 & DT & 82 & 15 & 15 \\ 4 & DT & 245 & 81 & 33 \end{bmatrix}$$

Fitness Function. The fitness function is an objective function that maximizes the accuracy of the candidate model while minimizing its false signals. More precisely, models with higher Detection Rate (DR), lower False Positives (FPs), and lower False Negatives (FNs) are more likely to gain more fitness scores. In EINSPECT, given a model m with Detection

Rate DR and False Positive Rate FPR, we use the following objective function to compute the fitness score for m:

$$fitness(m) = \frac{(DR_m - FPR_m)}{100} \tag{4.1}$$

Selection. The fitness score determines which pair of chromosomes to pick for crossover (Line 5 in Algorithm 4). In addition to the fitness score, a valid pair of chromosomes is one from the same algorithm but distinct feature sets to avoid useless selection. A common selection method is to use an elite-based selection in which the most fit individuals are picked for crossover. While elite-based selection is simple and favors the most fit ones, it penalizes the chromosomes with lower fitness scores and reduces the randomness assumption in GA. To this end, in EINSPECT we use the *Tournament* (see Algorithm 5) selection. Selection of models is done by first randomly picking k models from the population (Lines 4-8 in Algorithm 5). Then, the one with the highest fitness score is selected for crossover (Lines 9-13 in Algorithm 5). The parameter k is called the tournament size which tells the number of times the tournament is run.

Algorithm 5 Tournament-Based Selection for k=2.

```
 1: $n$ :# of models
 2: $selected \leftarrow model[0]$
 3: for i=1 ; i ≤ n ; i++ do
 4:     $a \leftarrow random(1, n)$
 5:     $b \leftarrow random(1, n-1)$
 6:     if $b \geq a$ then
 7:         $b \leftarrow b + 1$
 8:     end if
 9:     if $fitness[a] > fitness[b]$ then
10:         $selected \leftarrow model[a]$
11:     else
12:         $selected \leftarrow model[b]$
13:     end if
14: end for
```

Crossover. After two chromosomes are selected (using Algorithm 5), the semantics of crossover in our approach is such that one or more of the features of the selected chromosomes are swapped to produce two offsprings. Which feature to swap is determined by applying an n-point crossover operation where indices of the features to be swapped are selected randomly at each generation. For instance, to apply a 3-point crossover, at each generation when doing crossover, 3 numbers in the range $[1, m]$ are randomly selected where m is the number of features. Then after, the features at the selected positions of the models are swapped to complete the crossover operation.

From our running example, suppose that the following models are selected for the crossover:

$$\mathbf{model}_{1,1} = \left[\begin{array}{ccccc} \textbf{URL} & \textbf{Alg} & \textbf{urlLen} & \textbf{pathLen} & \textbf{remoteLnk} \\ 1 & DT & 56 & 21 & 5 \\ 2 & DT & 123 & 68 & 25 \\ 3 & DT & 82 & 15 & 15 \\ 4 & DT & 245 & 81 & 33 \end{array}\right]$$

$$\mathbf{model}_{2,1} = \left[\begin{array}{cccc} \textbf{URL} & \textbf{Alg} & \textbf{zeroSizeIframes} & \textbf{nativeJSFunc} \\ 1 & DT & 0 & 5 \\ 2 & DT & 5 & 23 \\ 3 & DT & 1 & 7 \\ 4 & DT & 10 & 49 \end{array}\right]$$

The chromosomes, $\text{model}_{1,1}$ and $\text{model}_{2,1}$, are from the same algorithm (DT in this case) and have distinct feature sets (F_1 and F_2 respectively). An instance of a valid crossover operation could be swapping the last two features of the two chromosomes. More precisely, the last two features, *pathlen* and *remoteLnk*, of $\text{model}_{1,1}$ are swapped symmetrically with the

last two features, *zeroSizeIframes* and *nativeJSFunc*, of model$_{2,1}$ resulting in the following two offsprings:
offspring$_1$: crossover(**model**$_{1,1}$,**model**$_{2,1}$)

$$\textbf{offspring1}_1 = \begin{bmatrix} URL & Alg & urlLen & \textbf{zeroSizeIframes} & \textbf{nativeJSFunc} \\ 1 & DT & 56 & \textbf{1} & \textbf{11} \\ 2 & DT & 123 & \textbf{6} & \textbf{50} \\ 3 & DT & 82 & \textbf{0} & \textbf{2} \\ 4 & DT & 245 & \textbf{9} & \textbf{35} \end{bmatrix}$$

offspring$_2$: crossover(**model**$_{2,1}$,**model**$_{1,1}$)

$$\textbf{offspring1}_2 = \begin{bmatrix} URL & Alg & \textbf{pathLen} & \textbf{remoteLnk} \\ 1 & DT & \textbf{36} & \textbf{8} \\ 2 & DT & \textbf{300} & \textbf{12} \\ 3 & DT & \textbf{50} & \textbf{13} \\ 4 & DT & \textbf{259} & \textbf{18} \end{bmatrix}$$

In **offspring**$_1$ and **offspring**$_2$, the values in **bold** are the values for the features swapped as a result of the crossover operation. Once the crossover is done, the fitness score for the offsprings is computed. In this context, computing the fitness score requires training the DT algorithm on **offspring**$_1$ and **offspring**$_2$ to get the DR and FPs from which the fitness score is computed (using Equation 4.1) for the next generation of the GA.

After the crossover operation, chromosomes for which the algorithm and features are the same, i.e., redundant chromosomes, may emerge. In such a case, the more fit chromosome is maintained. If redundant chromosomes happen to have the same fitness score, one of them is selected at random and maintained in the population.

Mutation. The goal of mutation is to introduce a reasonable diversity to the population under evolution by making slight changes to offsprings. A common mutation strategy is to modify the values of features using a certain mutation probability. For instance, after the crossover, a small value (e.g., 0.02) is added to the feature values of $\mathbf{offspring}_1$ and $\mathbf{offspring}_2$ to get a slightly modified varieties of the offsprings. To imitate evolution over a time-frame, we may also consider replacing unreachable URLs (e.g., pages taken-down after malicious activity) in the training set and take this replacement as a mutation operation (we call it *Inherent Mutation*). This would potentially be a realistic mutation operation for malicious web pages. An alternative mutation strategy is to introduce new feature(s) to both of the offsprings and compute the fitness score after the mutation.

Termination Criteria. In general, a GA is terminated either when the best fitness score is attained by at least one chromosome or a certain maximum iteration of generations is reached. In our approach, the termination criteria is fulfilled either when a desired threshold for FPs, FNs, and DR is attained by the best chromosome or after a maximum of k iterations. For example, 0.01% FPs, 0.01% FNs and 99% DR within the first k or less iterations could be used for termination. If the GA does not converge to the threshold after k iterations, we just pick the chromosome with the highest fitness score to terminate the evolution. The best chromosome, i.e., the fittest model, that is obtained at the end of the evolution, is used to detect unknown web pages to classify them as benign or malicious.

4.4 Dataset and Setup

In this section, we first highlight the implementation details of EINSPECT. Then, we discuss the dataset description and experimental protocol.

4.4.1 Implementation

Our crawler is a customized instance of HERITRIX [4], an open-source, web-scale, and archival-quality crawler. We reused the feature extraction engines from Chapter 3 in which the URL feature extraction engine is implemented based on the URL class of Java. The HTML and JavaScript feature extraction engine is implemented as a wrapper on a custom headless browser based on HtmlUnit [81], and we used the respective APIs of Facebook [30], Twitter [92], and Google$^+$ [36] to extract reputation features.

We used seven standard learning algorithms namely: J48 Decision Tree, Random Tree, Random Forest, Naive Bayes, Bayes Network, Support Vector Machine, and Logistic Regression classifier on the WEKA [38] machine learning suite. We have also used these algorithms and evaluated them on a large-scale dataset in Chapter 3. Therefore, we decided to start with and build up on effective algorithms and make them more effective by using the GA. The GA is implemented in Python to accept the feature matrix and accuracy values of candidate models and iterate over generations to select the best model.

4.4.2 Dataset Collection and Validation

To maintain the representativeness of the dataset, we collected a total of $22,891$ samples from our crawler and other data sources. The breakdown of our dataset is shown in Table 4.1.

For benign web pages, we used the Alexa [1] top global websites, the Yahoo [44] random URL generator, and the DMOZ [23] directory to collect a total of $6,867$ pages. For malicious web pages, we used the malware and phishing blacklist of Google Safe Browsing [35], the PhishTank [68] database of verified phishing pages, and the MalwareURL [60] list of malware-serving URLs to collect a total of 16, 024 unique URLs. In preparing the dataset, we made sure that the dataset (specially for benign pages) is not skewed to a particular group of web pages by limiting the proportion of web pages belonging to the same domain. As a sanity check, we validated the entire data set with a custom-made honeyclient and removed from the dataset those URLs that responded with HTTP 404 (page not found) error.

Table 4.1: Dataset for training and testing of EINSPECT.

Purpose	**Benign**	**Malicious**	**Total**
Training	5, 055	10, 969	16, 024
Testing	2, 166	4, 701	6, 867

4.4.3 Experimental Setup

Candidate Models Generation. To evaluate EINSPECT, we divided the dataset into a training set (70%) and a test set (30%) as shown in Table 4.1 with a benign-to-malicious ratio of 1:2. Using Algorithm 2, we generated 35 candidate models by training the 7 learning algorithms on 5 feature classes. All the candidate models were trained using 10-fold cross validation and the accuracy of the candidate models is summarized in Table 4.2 with the percentage of FPs/FNs for each model across the 5 feature classes.

Benchmark Selection. Depending on how much we tolerate FPs or FNs, the best model from the candidates is either Random Forest on all features or J48 on all features (see values in **bold** in Table 4.2). In classifying web pages, the ultimate goal is to minimize (eliminate if possible) FNs (malicious web pages deemed benign) but reasonable number of FPs (benign web pages deemed malicious) are tolerable at the cost of more resource for analysis. As a result, we selected the J48 classifier on all fea-

Table 4.2: EINSPECT: Percentage of FPs/FNs of candidate models with 10-fold cross validation on the training set.

Feature Sets: % FP/% FN					
Classifiers	URL	HTML	JavaScript	Reputation	All Features
J48	33.2/1.8	24.7/13.3	40.1/10.7	36.8/0.4	**6.4/1.9**
Random Tree	33.7/1.9	26.8/12.9	38.7/10.9	34.9/0.2	9.1/4.4
Random Forest	33.2/1.8	19.5/14.0	38.7/10.8	34.8/0.2	**4.9/2.7**
Naive Bayes	41.8/4.7	80.6/2.9	86.3/5.5	98.1/0.0	67.4/4.4
Bayes Net	38.9/4.9	22.3/24.9	42.7/16.4	40.4/7.4	9.5/15.3
SVM	32.8/1.7	74.3/0.4	38.3/11.0	34.8/0.6	73.7/0.2
Logistic	33.7/2.9	73.5/3.4	88.9/2.3	93.6/0.1	23.4/4.1

tures as it has the lowest FNs (1.9%) with reasonably low FPs (6.4%). We used the performance of this model as a benchmark to compare it with the performance of the models obtained after applying the evolutionary model searching and optimization.

GA-Based Model Searching and Optimization. We used the *EvolutionarySearch*[3] library for GA-guided feature searching and optimization. We ran the GA 6 times with 20, 30, 40, 60, 80, and 100 generations. For all the 6 iterations of the GA, we used the binary *Tournament selection* (Algorithm 5), *2-point crossover* with crossover probability of 0.6, and *BitFlip*[4] *mutation* with mutation probability of 0.1. At the end of the execution of the GA, we retrain the seven classifiers on the features selected by the GA to evaluate the models' fitness. When the GA is not able to improve the fitness of the best model any more, we save the best model to use it for detection. Finally, using the same test set used for the best model prior to the GA, we evaluate the fittest model for accuracy and compare it with the benchmark (the best model trained without the GA).

[3]`http://weka.sourceforge.net/packageMetaData/EvolutionarySearch/`

[4]Bit-Flip Mutation: a bit at a random position in a binary chromosome string is flipped.

4.5 Evaluation

In this section, we evaluate EINSPECT from the standpoints of: accuracy of detection, error rates, performance overhead, and practical significance of the evolution-guided detection of malicious web pages.

4.5.1 Accuracy and Error Rate

Benchmark Performance. On the training set, without running the GA, the best model (J48 with all features) achieved 96.6% accuracy with 6.4% FPs, and 1.9% FNs. On the test set, it achieved classification accuracy of 60.1% with 4.5% FPs and 58.5% FNs.

Significance of the GA. The best model after running the GA is still on the J48 classifier but with only 12 features (4 URL, 6 JavaScript, and 2 reputation features) that survived the evolution and the accuracy on the training set is 96.5% with 8.1% FPs and 1.7% FNs. Comparing it with the best model before running the GA, the FNs are reduced from 1.9% to 1.7% amounting to 10.5% reduction in FNs. This reduction means that the GA-guided model is able to avoid a misclassification of about 1152 malicious web pages as benign in the training set, proving that the GA-guided model is 10.5% more precise than the benchmark counterpart.

As can be seen from Figure 4.9, the fittest model is found when running the GA for 20 generations. The best model produced by the GA did not improve for iterations starting from 60 generations onwards. The results of evaluating the best model by the GA on the separate test set also shows an improvement in precision of the fittest model. More precisely, the fittest model achieved classification accuracy of 74.8% with 7.1% FPs and 33.5% FNs. In comparison to the performance of the benchmark model on the same test set, the GA-guided model improved the detection accuracy by 14.7% while significantly reducing the FPs by 25%. In fact, the significant

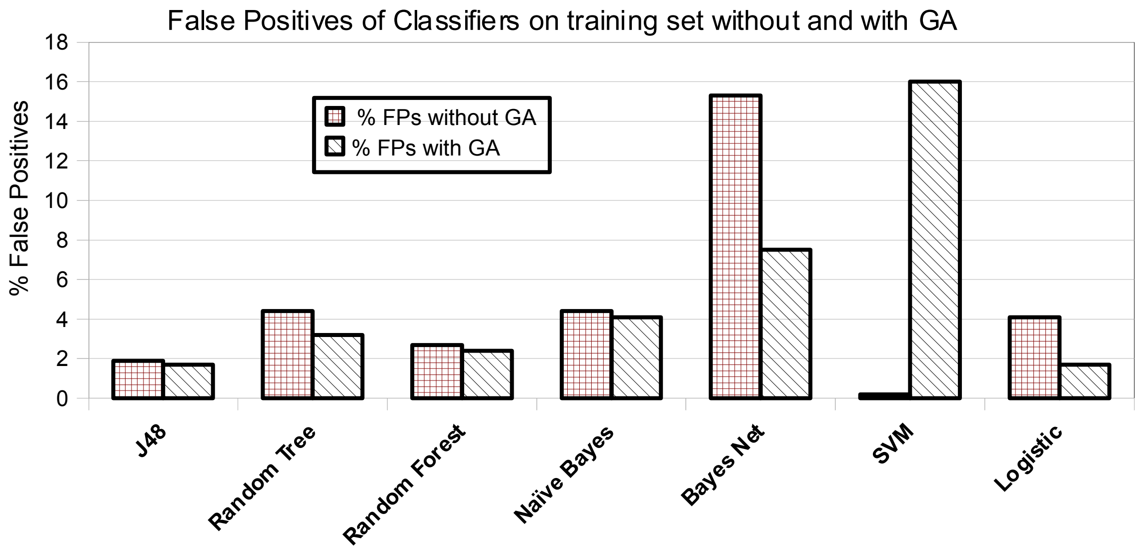


Figure 4.8: EINSPECT: False Positives for classifiers before and after the GA on the training set.

improvement using the GA is achieved at a reasonable cost of a little more FPs which is tolerable given the the criticality of misclassifying malicious web pages as benign.

Another interesting insight from using the GA is that the FPs of most classifiers decreased after running the GA for 20 generations. We took the best models for each classifier in Table 4.2 (precisely values in the last column) to evaluate the impact of the GA on the rest of the classifiers apart from the best model in the candidates. As shown in Figure 4.8, the FPs of six out of the seven classifiers is reduced as a result of the GA with the exception of SVM for which the FPs increased significantly with the GA.

4.5.2 Performance Overhead

The computing resource we used in our experiment is an Intel i7 dual-core 2.66GHz CPU with 8GB of RAM running 64-bit MacOSX operating system. Under a fairly normal load of the machine, we measured the

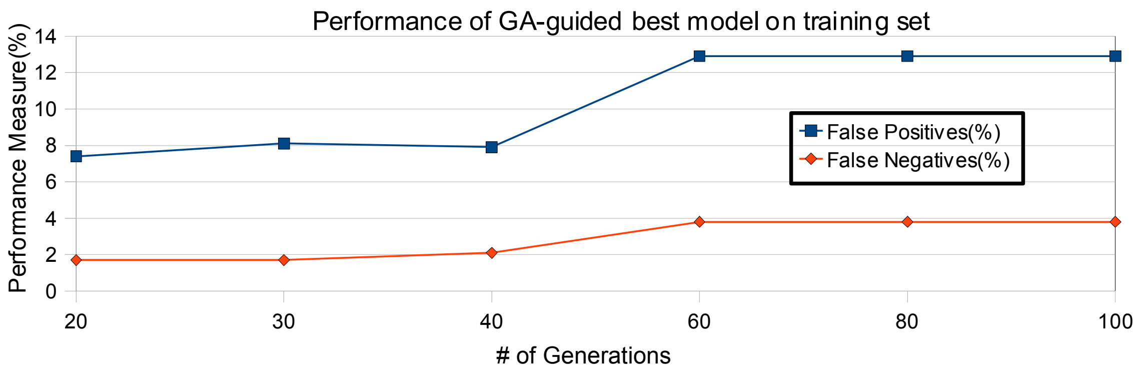


Figure 4.9: EINSPECT: False Positives and False Negatives of the GA-guided fittest model on the training set.

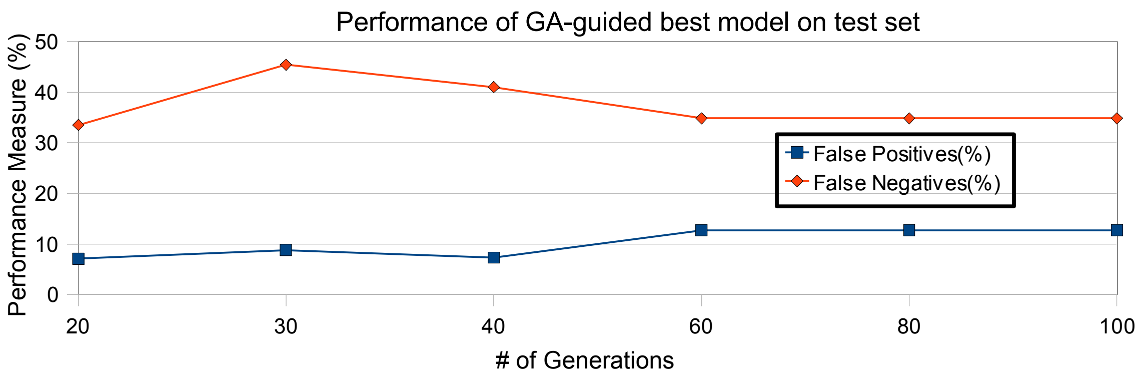


Figure 4.10: EINSPECT: False Positives and False Negatives of the GA-guided fittest model on the test set.

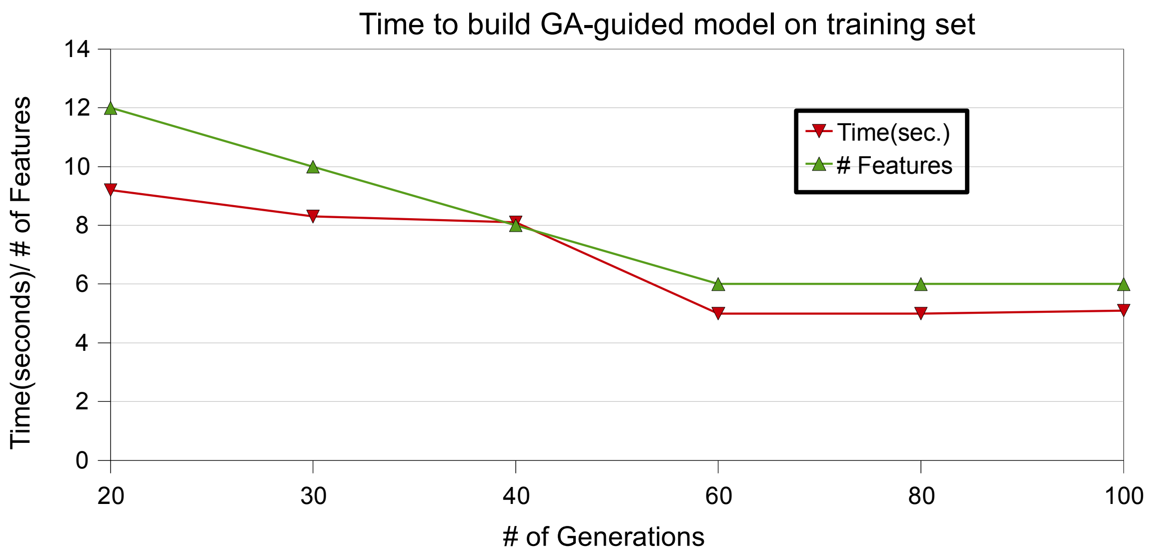


Figure 4.11: CPU clock overhead in relation to number of features selected by the GA to build the best detection model from the training set with increasing number of generations.

average time to build a model without the GA and compared it with the average time it takes to build a model when using the GA-guided model generation.

While the time spent to build the best model without the GA is only 2.76 seconds, the GA-guided model building took 9.2 seconds. This overhead is understandable as the GA spends extra time to search for the best combination of features before training. As can be seen from Figure 4.11, the bigger the number of generations the less time the GA needs to build the detection model. This gain in speed is due to the fact that the number of features selected by the GA gets fewer as the number of generations increase. Consequently, making the model generation faster, but the accuracy of the model drops as important features are eliminated in an attempt to converge the GA.

4.5.3 Results from a Public Service

To examine the effectiveness of EINSPECT, we submitted the test set to WEPAWET [93], a publicly available, anomaly-based, dynamic analysis service to analyze and detect web-based threats triggered by visiting a web page. Differently from WEPAWET, in EINSPECT, we use static analysis with lightweight emulation and the analysis and detection is rather learning-based with a reinforcement from evolutionary technique using a GA.

With respect to the 92.8% correct classification of benign web pages from the test set by EINSPECT, WEPAWET correctly flagged 92% of the benign web pages with 2 benign pages reported as malicious (one of them also reported malicious by EINSPECT). In addition, WEPAWET flagged 11 benign pages as suspicious (most of which are flagged as benign in EINSPECT), and on 161 benign pages it returned error. On the malicious test set, WEPAWET returned 2.34% as suspicious, 97% as benign, 2.7% as error, and 0% as malicious. On the other hand, EINSPECT correctly flagged 65.5% of the malicious test set with 35% misclassification error. Manual analysis of the majority of malicious web pages that WEPAWET classified as benign shows that these web pages are no longer served with malicious content. This difference is reasonably valid as the time at which we extracted features from these web pages and the time at which we submitted these web pages to WEPAWET has a difference of 5 days.

4.5.4 Immunity to Possible Evasion

To completely evade our technique, an attentive attacker needs to study all the features we use in EINSPECT and come up with attack payloads to mislead and ultimately escape detection by our model. To do so would not be an easy task as the features we use to capture web page artifacts are reasonably comprehensive and to avoid most (all) the features in an attempt to render our detection model ineffective forces the attacker to have an easy-to-detect web page that can be caught by simple static analysis or signature matching techniques.

Above all, at the core of building our detection model is a GA which, depending on the fitness of the candidate models, hides the details of the fittest model (features and algorithm) at a given time. As a result, the adversary has no (very limited) way of deducing which particular features are in use due to the change in the relevance of the features every time the GA is run.

4.6 Conclusions

In line with the evolving and polymorphic nature of online threat, techniques devised to defend web users from malicious content need to be up-to-date enough to automatically tune themselves to cope with the threat landscape on the Internet. Keeping detection models inline with the constantly changing artifacts of web pages in general and malicious web pages in particular is a challenge in devising precise countermeasures against malicious web pages.

In this chapter, we presented EINSPECT, an inherently evolution-aware and learning-based approach that is guided by a GA to analyze and detect malicious web pages leveraging static analysis and lightweight dynamic analysis.

Our evaluation on a fairly large-scale dataset shows that EINSPECT is able to significantly improve the precision (especially False Negatives) in detection of malicious web pages by enhancing learning-based detection using evolutionary model searching and optimization with the help of a GA.

And more importantly, as we implemented EINSPECT on top of BINSPECT (from Chapter 3), we proved how the evolution-aware approach we presented in this chapter can reinforce a learning-based approach.

Chapter 5

Detection of Exploit Kits

In this chapter, we turn our attention to one of the preferred mean used by cyber-criminals in spreading web-borne malware to infect innocent victims, i.e., prevalence of Exploit Kits.

5.1 Overview

Cybercrime on the Internet has seen a proliferation in recent times [12, 87]. This proliferation has been largely due to the development of criminal, for-profit, software infrastructure for conducting attacks on endusers. In this infrastructure, Exploit Kits play a central role as they facilitate the infection of users through malware that exploits client vulnerabilities. An Exploit Kit comes as a piece of off-the-shelf software that can be licensed from the underground market, which when installed and configured on a web server, carries out a malicious campaign targeting innocent victims. Examples of attacks that are launched by Exploit Kits include drive-by-downloads, spam and denial-of-service. A website hosting an Exploit Kit is advertised through URLs disseminated through spam links, search campaigns, social networking platforms, web postings or website hijacking. Innocent victims that click on these URLs have their systems compromised through drive-by-download attacks and the infected hosts are subsequently used for staging further criminal activities (e.g., spam campaigns as part of a Botnet).

Given the role of Exploit Kits in cybercrime, it is a natural question to ask whether a given URL points to an Exploit Kit. This is a question that has significant implications for the safety of end-users on the Internet, given the proliferation of criminal activities in recent years and the change in the "business model" of the underground market from selling crimeware to providing it as a service akin to software-as-a-service [37, 82].

Identifying Exploit-Kit-hosted URLs has the potential to be useful to a broad stakeholders. For example, search engines can use such detection techniques to prevent indexing of such URLs as these URLs are usually accompanied with black-hat SEO techniques to boost their rank. They also can pass on such blacklisted URLs to end users and browser vendors, who can protect their customers from harmful infection. Anti-virus companies can also keep their signature datasets up-to-date by analyzing such URLs. Moreover, take-down operations against malicious activities on the Internet can be reinforced by a detection technique that identifies Exploit Kits.

Previous work on detecting Exploit Kit hosted pages primarily relied on either looking for anomalous characteristics of the downloaded code [18, 75] or detect [71, 99] changes to the system properties. The former is harder to define for new exploits included by the Exploit Kits (which are continuously updated) while the latter involves the difficult task of correct execution of malicious code.

In this chapter, we tackle the problem of detecting whether a given URL is hosted by an Exploit Kit with a new approach that uses machine learning to realize fast detection of Exploit Kit hosted pages. Through an extensive analysis of the workflows of 38 different Exploit Kits, we develop an approach that uses machine learning to detect whether a given URL is hosting an Exploit Kit. Central to our approach is the design of distinguishing features drawn from the analysis of attack-centric and self-defense behaviors of Exploit Kits. This design is based on observations drawn from Exploit Kits that we installed in a laboratory setting as well as live Exploit Kits that were hosted on the Web. We implemented our approach in a tool that we call WebWinnow. Moreover, to allow resource-constrained deployment of WebWinnow, we also built a fast pre-fitering engine. Evaluation of WebWinnow with real world malicious URLs suggest that it is highly effective in the detection of malicious URLs hosted by Exploit Kits with very low false-positives.

The contributions of the approach we present in this chapter are the following:

- a detailed analysis and characterization of Exploit Kits based on source code analysis and runtime probing to shade light on the attack-centric and self-defense behaviors of Exploit Kits.
- a machine learning approach that takes advantage of distinguishing features drawn from the attack-centric and self-defense behaviors of Exploit Kits to detect malicious URLs.
- an implementation and evaluation of the approach and a fast pre-filtering front-end to allow resource-constrained deployment of the approach.

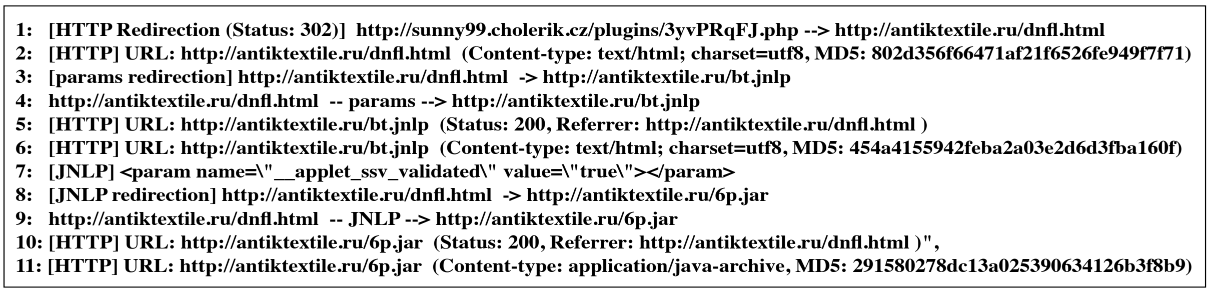
The remainder of this chapter is organized as follows. In Section 5.2, we present a brief background on Exploit Kits. Section 5.3 presents the details of the attack-centric and self-defense behaviors of Exploit Kits and how we leverage these behaviors to draw effective features for detection of malicious URLs linked with Exploit Kits. A brief overview of our approach is discussed in Section 5.4. The discussion on the training infrastructure and detection are presented in Section 5.5 and Section 5.6 respectively. We present the data collection methodology, dataset, and experimental setup in Section 5.7. In Section 5.8, we discuss the evaluation results of our approach. Finally, Section 5.9 concludes this chapter.

5.2 Exploit Kits

An Exploit Kit is a pre-packaged malicious software that is used to exploit vulnerabilities found in software applications (e.g., operating system, browsers, geo-location, browser plugins) for the purpose of spreading malware. Exploit Kits are one of the most common methods used in cybercrime and their emergence has been well documented in recent work [37]. Cybercriminals target vulnerabilities in operating systems, web browsers, office software, and third-party plugins to infect victim machines. The advent of Exploit Kits dates back to 2006 when the MPack kit was discovered for the first time [51].

Exploit Kits come with pre-written exploit code and they usually include installer scripts, a number of exploits, configuration details, administration features for the kit owner. The payload of the exploits could be key-loggers, botnets, fake anti-virus engines, or trojans. Exploit Kits are marketed in the underground marketplace and like legitimate software there are regular bug fixes, feature enhancements, customer support, end-user license agreements, and even competitions among the Kit developers. Most Exploit Kits: are written in PHP (some in C/C^{++}), use fingerprinting code, use extensive obfuscation, and use a number of third-party code (e.g., JavaScript and CSS). Examples of popular Exploit Kits include RedKit, CrimePack, CrimeBoss, Cool, Blackhole, SweetOrange, Eleanore, Fragus, NuclearPack, Sakura, NeoSploit, Siberia Private, and Styx.

According to the 2013 Internet Security Threat Report by Symantec [87], Blackhole took 41% of all Web-based attacks in 2012. There was also the release of an updated version of the kit (Blackhole 2.0) hinting that



```
1:  [HTTP Redirection (Status: 302)]  http://sunny99.cholerik.cz/plugins/3yvPRqFJ.php --> http://antiktextile.ru/dnfl.html
2:  [HTTP] URL: http://antiktextile.ru/dnfl.html  (Content-type: text/html; charset=utf8, MD5: 802d356f66471af21f6526fe949f7f71)
3:  [params redirection] http://antiktextile.ru/dnfl.html  -> http://antiktextile.ru/bt.jnlp
4:  http://antiktextile.ru/dnfl.html  -- params --> http://antiktextile.ru/bt.jnlp
5:  [HTTP] URL: http://antiktextile.ru/bt.jnlp  (Status: 200, Referrer: http://antiktextile.ru/dnfl.html )
6:  [HTTP] URL: http://antiktextile.ru/bt.jnlp  (Content-type: text/html; charset=utf8, MD5: 454a4155942feba2a03e2d6d3fba160f)
7:  [JNLP] <param name=\"__applet_ssv_validated\" value=\"true\"></param>
8:  [JNLP redirection] http://antiktextile.ru/dnfl.html  -> http://antiktextile.ru/6p.jar
9:  http://antiktextile.ru/dnfl.html  -- JNLP --> http://antiktextile.ru/6p.jar
10: [HTTP] URL: http://antiktextile.ru/6p.jar  (Status: 200, Referrer: http://antiktextile.ru/dnfl.html )",
11: [HTTP] URL: http://antiktextile.ru/6p.jar  (Content-type: application/java-archive, MD5: 291580278dc13a025390634126b3f8b9)
```

Figure 5.1: Part of an activity trace of the RedKit Exploit Kit on July 10, 2013. Visited from IE6.0 on Windows XP SP3 with Adobe Acrobat Reader Version 9.1.0, Java plugin Version 1.6.0.32, and Shockwave Flash Version 10.0.64.0.

its impact is likely to continue. Another Exploit Kit is Sakura which took 22% of overall kit usage in 2012. The next 3 in the top 5 kits in 2012 are Phoenix (10%), RedKit (7%), and Nuclear (3%).

```
function qweqwewqe(hid) {
    var info = {
    plugins : {
    java: plg_all_vers('Java'),
    adobe_reader: plg_ver('AdobeReader'),
    flash: plg_ver('Flash'),
    quick_time: plg_ver('QuickTime'),
    real_player: plg_ver('RealPlayer'),
    shockwave: plg_ver('Shockwave'),
    silver_light: plg_ver('Silverlight'),
    vlc: plg_ver('VLC'),
    wmp: plg_ver('WMP')
    }
   }

   var pass = rnd_str(1+Math.floor(Math.random()*10));
   var obj = {};
   obj["h"+rnd_str(1+Math.floor(Math.random()*10))] = hid; // host id
   obj["p"+rnd_str(1+Math.floor(Math.random()*10))] = pass; // XOR pass
   obj["i"+rnd_str(1+Math.floor(Math.random()*10))] =
       kor(JSON.stringify(info), pass);

   $("body").load("c"+rnd_str(1+Math.floor(Math.random()*10)), obj);
  }

  function plg_all_vers(name) {
    var info = PluginDetect.getInfo(name);
    var vers = info.All_versions;
    if(!vers)
     return '';
   return info.All_versions.join(';')
  }

  function plg_ver(name) {
   var info = PluginDetect.getVersion(name);
   return info;
  }
```

Listing 5.1: An excerpt from a client-fingerprinting code in the Neutrino Exploit Kit [58].

Figure 5.1 shows an excerpt from interaction with a real Exploit Kit on July 10, 2013. When a browser was pointed to `http://sunny99.cholerik.cz/plugins/3yvPRqFJ.php`, it was automatically redirected to `http://antiktextile.ru/dnfl.html`, a landing page for the RedKit Exploit Kit (Lines 1-2 in Figure 5.1). What happens next is a parameter-based redirection (Lines 3-6) to initiate the JNLP (Java Network Launch Protocol) to download and execute a remote JAR file by exploiting the security warning bypass vulnerability (CVE-2013-2423) on Java Web Start plugin (Lines 7-11).

As we discussed in Chapter 2 (Section 2.5), a vital step in the workflow of Exploit Kits is the client-profiling (fingerprinting) step. The subsequent steps of the attack chain are determined based on the outcome of this step. As an example, Listing 5.1 shows a fingerprinting code from the landing page of the Neutrino Exploit Kit. This code is a customized version of the popular *PluginDetect*[1] library that is used in several Exploit Kits. As

[1]`http://www.pinlady.net/PluginDetect/`

can be seen from Listing 5.1, the Exploit Kit checks details about the type and version number of the common browser plugins (Lines 1-14) and a *PluginDetect* object is used to get these details (Lines 26 and 34).

5.3 From Behavior to Features

In this section, we discuss how, based on our analysis of 38 distinct Exploit Kits, we derive distinguishing features based on which we train our classifiers for detection.

As we discussed earlier, the mission of Exploit Kits is to infect victims' devices with malware and compromise the victims' environment to use it to conduct further attacks. To accomplish this mission, Exploit Kits exhibit several *attack-centric* characteristics. At the same time, we have observed that Exploit Kits have to do a great deal of *self-defense* to evade detection.

We got access to the source code of 38 distinct Exploit Kits. We analyzed them with the purpose of leveraging their characteristics to automatically detect URLs linked with them. The analysis involved semi-automated source code inspection, deployment and then runtime probing of the Kits in a contained setting. A detailed characterization of the 38 Exploit Kits we analyzed is summarized in Table 5.1. To complement our in-house analysis, we used a virtualized sandbox environment to probe live Exploit Kits on the Web. In the rest of this section, we describe how we take advantage of the attack-centric and self-defense behaviors of Exploit Kits to develop features we use to train our classifiers for detecting malicious URLs hosted by Exploit Kits.

5.3.1 Attack-Centric Behaviors

From the point of view of detecting malicious URLs that lead to Exploit Kits, we noticed that the attack chain of an Exploit Kit when it is visited by

Table 5.1: Characterization of Exploit Kits based on their attack-centric and self-defense behaviors. Ver= Version, CP= Client Profiling, IPB= IP Blocking, Sel= Exploit Selection, Obf= Exploit Obfuscation, Blklst= Blacklist Lookup, Sign.Ev.= Signature Evasion, Rbt.Blk= Robot Blocking, Cod.Prt= Code Protection, Add.= Allow Adding Exploit.

Kit	**Ver.**	**CP**	**IPB**	**Sel.**	**Obf.**	**Blklst.**	**Sign.Ev.**	**Cloak**	**Rbt.Blk**	**Cod.Prt.**	**Lang.**	**Add.**
0x88	3.0	Yes	Yes	Yes	Yes	No	No	No	Yes	None	PHP	No
Adrenalin	NA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	ZendGuard	PHP	Yes
Armitage	1.0	Yes	Yes	Yes	Yes	No	No	Yes	No	None	PHP	No
Blackhole	1.1.0	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	IonCube	PHP	Yes
BleedingLife	2.0	Yes	No	Yes	No	No	No	Yes	No	IonCube	PHP	No
CrimePack	3.1.3	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	IonCube	PHP	Yes
Cry	NA	Yes	Yes	No	No	No	No	Yes	Yes	IonCube	PHP	No
Eleonore	1.4.1	Yes	Yes	Yes	Yes	No	No	Yes	Yes	None	PHP	No
El Fiesta	1.8	Yes	No	Yes	Yes	No	No	No	No	None	PHP	No
FirePack	0.18	Yes	Yes	Yes	Yes	No	No	Yes	No	ZendGuard	PHP	No
Fragus	1.0	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	IonCube	PHP	Yes
GPack	NA	Yes	Yes	Yes	Yes	No	No	Yes	No	IonCube	PHP	No
IcePack	5	Yes	Yes	Yes	Yes	No	No	No	No	None	PHP	No
Liberty	NA	Yes	Yes	Yes	Yes	No	No	No	No	None	PHP	No
Luckysploit	NA	Yes	Yes	Yes	Yes	No	No	Yes	No	Crypto	PHP	No
MPack	0.99	Yes	Yes	Yes	Yes	No	No	Yes	No	Custom	PHP	No
MultiSploit	NA	Yes	Yes	Yes	Yes	No	No	Yes	No	None	PHP	No
MyPolySploit	NA	Yes	Yes	Yes	Yes	No	No	Yes	No	None	PHP	No
NeoSploit	2.1	Yes	No	Yes	Yes	No	No	No	No	Custom	PHP	No
Neon	NA	Yes	Yes	Yes	Yes	No	No	Yes	No	ZendGuard	PHP	No
Nuke	NA	Yes	Yes	Yes	Yes	No	No	No	Yes	None	PHP	No
Phoenix	2.1	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	IonCube	PHP	No
RDS	2.0	Yes	Yes	No	Yes	No	No	Yes	No	None	PHP	No
SALOPack	NA	Yes	Yes	Yes	Yes	No	No	Yes	No	None	PHP	No
Fiesta	NA	Yes	Yes	Yes	Yes	No	No	Yes	No	None	PHP	No
SEOSploitPack	NA	Yes	No	Yes	Yes	No	No	Yes	No	None	PHP	No
Sava	NA	Yes	Yes	Yes	Yes	No	No	Yes	No	None	PHP	No
Siberia	NA	Yes	Yes	Yes	Yes	No	No	Yes	Yes	None	PHP	No
SmartPack	NA	Yes	Yes	Yes	Yes	No	No	Yes	Yes	None	PHP	Yes
Sploit25	NA	Yes	Yes	Yes	Yes	No	No	Yes	Yes	None	PHP	No
SpyEye	1.4.1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Custom	PHP	Yes
TargetExploit	0.06	Yes	Yes	Yes	Yes	No	No	Yes	Yes	None	PHP	No
ToRPack	NA	Yes	Yes	Yes	Yes	No	No	Yes	Yes	ZendGuard	PHP	No
Tornado	NA	Yes	Yes	Yes	Yes	No	No	Yes	No	ZendGuard	PHP	No
Unique	1.4.1	Yes	Yes	Yes	Yes	No	No	No	No	IonCube	PHP	Yes
Yes	2.0	Yes	No	Yes	Yes	No	No	Yes	No	IonCube	PHP	No
ZuesKit	1.2.1.8	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	None	PHP	Yes
zoPack	NA	Yes	Yes	Yes	Yes	No	No	Yes	Yes	None	PHP	No

a victim client gives useful insights as to the characterization of Exploit Kit behaviors. From our analysis, we identify the following four attack-centric behaviors of Exploit Kits.

Client Profiling

From our analysis, we noticed that an essential component in all Exploit Kits is the use of a fingerprinting routine in a form of obfuscated JavaScript code that is unpacked and executed on the client browser. Fingerprinting in most Exploit Kits has two components performed in the order *identification* and then *validation*. Identification is aimed at collecting the client profile which includes the type and version of the operating system, browser, and installed browser plugins. In particular, most Exploit Kits (34 out of 38) check the presence and version number of Acrobat Reader, Flash Player, and the Java WebStart plugin. All the 38 Exploit Kits we analyzed perform identification of the visiting-client.

Validation involves further inspection of the client personality to check if the client is a real user as opposed to a bot. As per our analysis, 36 out of the 38 Exploit Kits perform validation with the exception of Eleonore and RDS which proceed to preparing the exploit based only on the identity of the client.

The crucial features we capture from client profiling are routines invoked when an obfuscated JavaScript is unpacked and executed on the client side. These are fingerprinting specific features manifesting themselves as routines from the common PluginDetect[2] library that is widely used by Exploit Kits to collect identifying information of the client. For example, de-obfuscation of the obfuscated JavaScript code used by Exploit Kits for fingerprinting consistently shows the occurrence of the flagship functions such as `getjavainfo.jar`, `pdpd()`, and `getversion()`. We confirmed the

[2]`http://www.pinlady.net/PluginDetect/`

presence of these functions in 28 of the 38 Exploit Kits while the other 10 Exploit Kits used function aliases to wrap these functions. The feature *PluginDetect Routines* under the Interaction-Specific feature class in Table 5.2 is used to capture the unique number of these functions during client profiling.

Chain of Redirections

Probing if live Exploit Kits shows that the Kits take clients through a chain of redirections to ultimately land them on the page that hosts the actual exploit. Depending on the remote resource they want to interact with (access), they use different redirect types. The redirect types include window-open, HTTP (with different status codes like HTTP 3XX), script source (mostly used to point to remote JavaScript), params (redirection based on URL parameters), applets (remote invocation of Java applet), JNLP (Java Network Launch Protocol invocation), iframe (specially hidden and small iframes as legitimate iframes are large enough to be visible), and HREF-based redirection. According to our observation when probing live Exploit Kits on the Web, we confirmed that Exploit Kits such as RedKit, SweetOrange, Blackhole, and Cool are particularly notorious in using complex redirection chains by combining these redirect methods.

Exploit Obfuscation

Except two Exploit Kits, BleedingLife and Cry217, all the rest (about 95%) of the Exploit Kits we analyzed obfuscate the malicious JavaScript code they use in client profiling, exploit preparation, and exploit delivery. Obfuscation renders static analysis techniques ineffective and further complicates the task of dynamic analysis. Usually, the obfuscation technique is one of the components to be revamped when a new version of an Exploit Kit is released. Because, by the time the Kits are popular on the Web, the obfuscation technique is likely to be uncovered and countermeasure is developed to thwart it.

Allowing Additional Exploits

According to our analysis, 7 out of the 38 Exploit Kits allow uploading an additional exploit binary on top of the exploits they are shipped with. To confirm the addition of new exploits, we first added a key-logger executable to CrimePack3.1.3 and Fragus1.0 Exploit Kits. Then, we deployed them on a server and pointed a vulnerable client to both Exploit Kits. In both cases, the key-logger executable was successfully downloaded to the client. Although not available in most Exploit Kits, the possibility to include additional exploits is a functionality to allow skillful attackers to extend the Exploit Kit with "user-defined" exploits.

5.3.2 Self-Defense Behaviors

The primary purpose of Exploit Kits is to be effective in infecting victims with malware. However, to continuously infect victims, they need to escape detection techniques, for which they have to be armed with self-defense functionalities. In the following, we discuss six self-defense behaviors we inferred from our analysis.

IP Blocking

In an attempt to escape from automated analysis techniques, almost all Exploit Kits we analyzed are equipped with IP blocking mechanisms against bots (e.g., GoogleBot) and IP addresses from ToR networks. About 87% of the Exploit Kits we analyzed have features to block IP addresses of known services from Anti-Virus companies or security researchers. The only Exploit Kits that do not use IP blocking are Bleeding Life, El Fiesta, NeoSploit, SEOSploitPack, and the Yes Exploit Kit.

Blacklist Lookup

Exploit Kits check if their URL is blacklisted in one or more public blacklisting services. If so, the kit owner (administrator) changes (relocates) the URL and the kit continues to operate until it is discovered and blacklisted again. In total, 6 of the 38 Exploit Kits in our analysis check their URL against one or more blacklists. CrimePack, for instance, has a feature that allows it to lookup the Kit's URL in 8 major blacklists including Google Safe Browsing.

Signature Evasion

Anti-virus engines and Intrusion Detection Systems use string signatures to detect malicious content. Some Exploit Kits lookup the signature databases of online malware (virus) scanning services (such as virtest[3], scan4you[4]) so as to check if their signatures belong there. For example, the latest version of Blackhole (Version 2.1.0) checks for its own signatures in VIRTEST and SCAN4YOU online virus-scanning services. The 5 Exploit Kits that check for their own signatures in online ant-virus engines are Adrenalin, Blackhole, Fragus, SpyEye, and ZeusKit.

Cloaking

We confirmed that 15 out of the 38 Exploit Kits use cloaking. When visited after a successful exploit delivery, most (13) of them respond with the HTTP 404 (page not found) error while a few (2) responded with a blank page with no content and / or action. There are two cases in which an Exploit Kit might use cloaking. One is upon failure to exploit a client

[3]`http://virtest.com`
[4]`http://scan4you.com`

either because the client turns out to be invulnerable or it is a bot. In this scenario, the most common practice is to show HTTP 404 error or to simply redirect the client to a harmless page. Some Exploit Kits throw back to the client a random exploit just to try their chance (we noticed this in 0x88, Eleonore, Fragus, and Sava). The other case in which cloaking is used is when an already infected victim visits the Exploit Kit page. In this case, most Exploit Kits respond with `HTTP 404` while some show a blank page with no action.

Blocking of Robots

We found the `robots.txt` file in 14 of the 38 Exploit Kits and in all the cases the kits disallow any indexing attempt. Search engine crawlers rely on the `robots.txt` file placed in the public directory of websites to check permission for indexing of web pages. Some Exploit Kits (e.g., CrimePack, Eleonore, Fragus, Nuke, Phoenix, Siberia) disallow indexing by blocking all bots. This is done in an attempt to escape from a more advanced analysis by search engines to filter-out (from their index) suspicious or malicious web pages. Although not all Exploit Kits block robots, our analysis shows that when combined with attack-centric characteristics, such as client profiling and redirections, this behavior contributes to the identification of Exploit Kit URLs in practice.

Code Protection

Our analysis shows 3 distinct outcomes as to the code protection scheme used by Exploit Kits. About 8 (24%) use IonCube[5], about 5 (13%) use ZendGuard[6], and about 4 (10%) use custom code protection scheme (e.g., custom cryptography). The motivation in code protection is primarily to

[5]`http://www.ioncube.com/`

[6]`http://www.zend.com/en/products/guard/`

Table 5.2: Features used in WebWinnow to detect malicious URLs hosted by Exploit Kits.

Aggregate-Behavioral		
Feature Name	**Feature Description**	**Behavior**
Exploits	known exploits from exploit-db.com	Attack
Redirections	count of all types of redirections (used in [18])	Attack
Connections	count of all connections made to other resources	Attack/Defense
Locations	count of all remote locations contacted	Attack/Defense
Files	count of all files received from remote locations	Attack/Defense
Interaction-Specific		
Feature Name	**Description**	**Behavior**
Window Opens	count of window.open redirections	Attack (redirect)
HTTP Redirects	count of HTTP 3xx redirects (used in [18])	Attack (redirect)
JNLP Redirects	count of JNLP initiations	Attack (redirect)
Param Redirects	count of parameter-based redirections	Attack (redirect)
Script-SRC Redirects	count of script src redirections	Attack (redirect)
Link Redirects	count of link redirections	Attack(redirect)
iFrame Redirects	count of iframe-based redirections	Attack (redirect)
Familiar Kit URLs	count of familiar Exploit Kit URL patterns	Attack (redirect)
PluginDetect Routines	count of functions related to PluginDetect library	Defense (anti-robot)
Java WebStart Routines	count of Java WebStart initiations	Attack (redirect)
URL Translation Redirects	count of URL translation redirections	Attack (redirect)
Connection-Specific		
Feature Name	**Description**	**Behavior**
ActiveX Attempts	count of attempts to load ActiveX controls	Attack (obfuscation)
Heap Spray Attempts	count of attempts to allocate very large heap size	Attack (obfuscation)
Unique Countries	count of unique countries traversed	Attack(redirect)/ Defense (cloaking)
Top Level Domains	count of unique TLDs involved in interaction	Attack (redirect) /Defense (cloaking)
Content-Specific		
Feature Name	**Description**	**Behavior**
HTML	size in bytes of HTML content	Defense (cloaking)
CSS	size in bytes of the CSS content	Attack (obfuscation)
JavaScript	size in bytes of JavaScript code	Defense (obfuscation)
JAR	size in bytes Java Archive files	Defense (obfuscation)
Octet-Stream	size in bytes of arbitrary content	Defense (obfuscation)
Command	size in bytes of commands (e.g. shell command injection)	Defense (obfuscation)
Plain Text	size in bytes of plain text delivered to client	Defense (obfuscation)
Compressed Content	size in bytes of compressed content	Defense (obfuscation)
XML	size in bytes of XML document	Defense (obfuscation)
Portable Document	size in bytes of portable document	Defense (obfuscation)

prevent code stealing while at the same time giving hard time for detection techniques that use code analysis. IonCube, for instance, not only allows encoding of PHP code but also allows binding of code to a certain IP address or domain. Despite these code protection mechanisms, source code is sometimes leaked. To combat source code leakage, some Exploit Kits (e.g., Blackhole) are shifting their "business model" to only rental mode (exploit-as-a-service [37, 82]) to ensure that they do not give away even the protected source code.

5.3.3 Features

The features we draw from the attack-centric and self-defense behaviors of Exploit Kits are summarized in Table 5.2. We designed a total of 30 features of which 5 are *Aggregate-Behavioral*; 11 are *Interaction-Specific*; 4 are *Connection-Specific*; and 10 are *Content-Specific* features. Of all the the features, the interaction-specific features are the most discriminating features because we characterize fingerprinting and the various redirection attempts using fine-grained details of interaction types. In the following, we briefly describe these features. In Section 5.8, we evaluate the statistical significance of these features based on the training dataset.

Aggregate-Behavioral

These 5 generic features provide a course-grained characterization in terms of the number of known *exploits* (based on CVE entry in Exploit Database[7]), aggregated count of *redirections* (similar to the redirection feature in [18]), total number of *connections* made to other destinations, total number of *locations* from which remote content is fetched, and total number of distinct *files* dropped on the client.

[7] `http://exploit-db.com`

Interaction-Specific

In this class of features, 8 out of the 11 features have not been used by previous work. Only *HTTP Redirects*, *Script-SRC Redirects*, and *Link Redirects* have been used in previous work [18] to detect malicious web pages. We noticed that these fine-grained redirection features are effective in separating Exploit Kit URLs from non Exploit Kits. Another particularly useful feature is the *PluginDetect Routines*, which counts the number of fingerprinting routines used by Exploit Kits.

Connection-Specific

The 4 features in this class are those that are captured when connections are made to remote sources. They include *ActiveX* loading attempts (e.g., presence of Microsoft XMLHTTP ActiveX), *heap-spraying* attempts (e.g., when the argument to the `eval()` function is too large), unique number of *countries* traversed and *Top Level Domains (TLDs)* involved in the connection. Features such as these are attributed to the strategy followed by cyber-criminals to complicate the traceability of the Exploit Kit infrastructure during take-down operations by Law Enforcement.

Content-Specific

We extract a total of 10 features from the content-related activity logs. Some MIME types (e.g., PDF, Shockwave file) are particularly attractive for Exploit Kits as they target vulnerable plugins that render such file types. More precisely, we extract the size of content delivered to the client for common MIME types used on the Web with emphasis on content-type used by Exploit Kits. In particular, these features include size (in bytes) of HTML, CSS, JavaScript, Octet-Stream (e.g., long byte pattern), command, plain text, compressed content (e.g., *. zip, *.gz, *.tar), XML, PDF, and Postscript content. Notice that where there is obfuscation these content-specific features are extracted after de-obfuscation so as to obtain feature values that can distinguish content delivered by Exploit Kits from content delivered by benign web sites.

5.4 WebWinnow Overview

In a nutshell, our approach is formulated as a machine-learning based technique for the detection of malicious URLs hosted by Exploit Kits. At the core of our approach is that, based on the workflows of Exploit Kits, we leverage their *attack-centric* and *self-defense* behaviors to design distinguishing features. Using the features, we train precise classifiers to detect malicious URLs hosted by Exploit Kits.

Our approach resembles techniques that combine honeyclients and learning to analyze the side-effects of malicious activities. These techniques (e.g., [65, 99]) inspect the pre-execution and post-execution snapshots of a honeyclient system properties (e.g., processes, memory access). However, in our approach, we base the characterization of malicious activities on what happens *during execution* instead of analyzing the side-effects. In effect, the goal of the analysis is shifted from examining side-effects to analyzing firsthand execution dynamics to reveal malicious activities. Moreover, we avoid the overhead of taking system snapshots before and after execution. The following three steps summarize the pipeline of our approach:

1. As we have seen in Section 5.3, we analyze source code and runtime behavior of Exploit Kits by deploying them in a controlled setting. In addition, we probe live Exploit Kits on the Web to capture their activity and the content they deliver to the client. By combining these observations, we identify attack-centric and self-defense mechanisms of

Exploit Kits. Based on careful examination of the attack-centric and self-defense behaviors, we then draw features that are precise enough in distinguishing URLs hosted by Exploit Kits.

2. We extract these features from samples of (1) Exploit Kits we installed locally (2) live Exploit Kits on the Web and (3) non Exploit Kit URLs. Using the features, we train and evaluate supervised learning algorithms from which we generate detection models. We discuss the training in detail in Section 5.5.

3. Given an unknown URL, we query the detection models to give the verdict as to whether the URL is an Exploit Kit URL or not. In Section 5.6, we describe the detection in detail.

As the notion of malicious URLs is broad, before we continue to describe the training and detection, we highlight the semantics of what we call an "Exploit Kit URL". The notion of an Exploit Kit URL in our approach is a URL that when a victim lands on it: fingerprints the victim's personality; involves a series of redirections; and eventually attempts to download and execute arbitrary binary on the victim's environment without the knowledge of the victim. In addition, when visited again (e.g., after a few seconds), the same URL responds with benign-looking page, page not found error, or another variant of the content it tried to download and execute in the first encounter. Alternatively, when a victim directly visits a landing page of an Exploit Kit and the fingerprinting and exploit delivery proceeds without redirections, we also call such a URL an Exploit Kit URL. Hence, it is such a URL that we focus on in this approach.

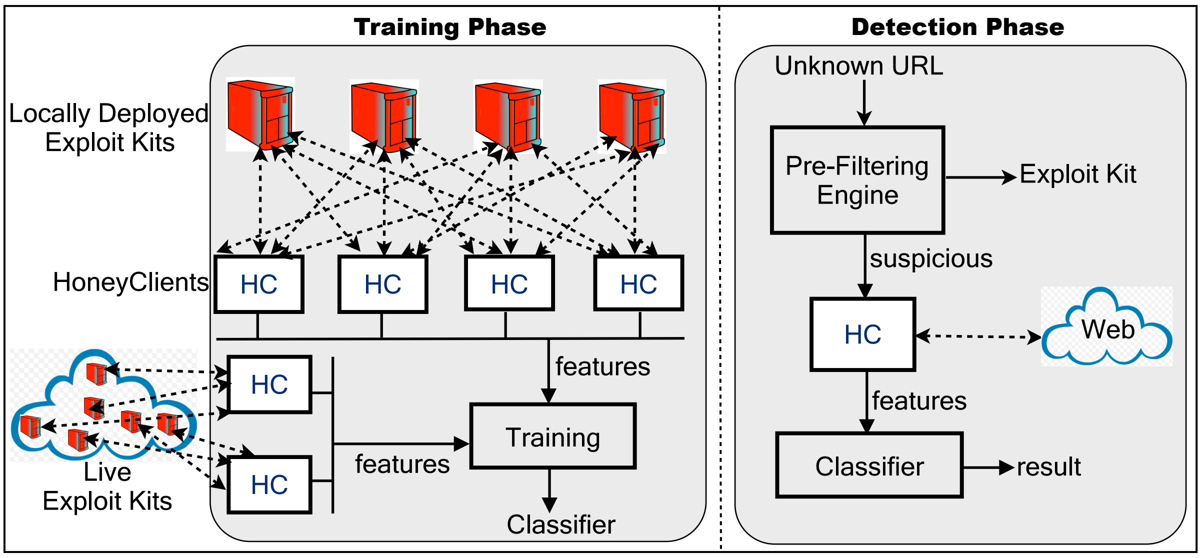


Figure 5.2: WebWinnow Training and detection pipeline. HC = HoneyClient.

5.5 Training

We now present the details of the training phase of WebWinnow by describing the training infrastructure for Exploit Kits we installed locally, live Exploit Kits on the Web, and non Exploit Kit URLs.

Based on the features derived from the behavioral characterization of Exploit Kits we described in Section 5.3, the goal of the training is to generate a model that is used to detect Exploit Kit URLs. To enrich our training set, we capture as much activity-centric details as possible of both Exploit Kit and non Exploit Kit URLs in a contained environment. To this end, we divide the acquisition of our training dataset into three parts. The first part probes locally deployed Exploit Kits. The second part deals with probing of live Exploit Kit URLs. The third part involves probing of non-Exploit Kit URLs. In what follows, we describe these three contained probing steps in detail.

5.5.1 Locally Installed Exploit Kits

To inspect the attack-centric and self-defense behavior of Exploit Kits in action, we installed Exploit Kits for which we could get the source code. The advantage of having locally installed Exploit Kits is twofold. First, it gives a deeper understanding of the operational mechanics of the Exploit Kits, specially how they orchestrate attacks. Second, it gives insights as to how the infection statistics is collected and the mechanisms that the Exploit Kits employ to evade detection techniques. In a virtualized environment with multiple Exploit Kits deployed on the server-side, we schedule honeyclients on the client-side to visit the Exploit Kit URLs with a vulnerable user-agent personalities (details in Section 5.7.1). The honeyclients are directly pointed to and visit the landing page of the Exploit Kits so as to collect whatever the Exploit Kits throw back to the client.

More precisely, we collect HTML and CSS content, script loaded (mainly JavaScript), exploit payload dropped into the honeyclient, remote content fetched, and shell-code execution attempts, and more importantly the whole HTTP transaction. After a few seconds, we repeat the visit in order to compare the activity logs with the first visit. In most Exploit Kits, if the first visit succeeds in dropping an exploit payload, the subsequent visits from the same address of the honeyclient are usually served with benign-looking response or error codes. As explained earlier, most Exploit Kits make use of cloaking.

5.5.2 Live Exploit Kits on the Web

For probing the activity log of live Exploit Kits on the Web, we rely on our data sources of live Exploit Kits in the wild. Given a live Exploit Kit URL at hand, we launch one of our honeyclients and probe the server to collect its activity logs with details similar to the Exploit Kits installed locally. The honeyclients we use for live Exploit Kits are separate from the ones we use for the locally installed kits. More importantly, we reset the honeyclients after collecting the activity logs of a given Exploit Kit before moving on to visit the next Exploit Kit URL.

An attentive reader might wonder about the distinction between the contained probing of locally installed and live Exploit Kits on the Web. There are two fundamental variations pertinent to the typical steps involved in Exploit Kit workflow we described in Chapter 2. In the first place, live Exploit Kits are engaged in a series of redirections before reaching the landing page while the locally installed ones are not (unless we configure them to have redirections). Secondly, compared to the versions of the kits we deployed locally, we presume that live Exploit Kits are likely to be more recent versions which, we assume, give us a fresher insight of their workflow.

5.5.3 Non Exploit Kit URLs

For this class of URLs, we use publicly known benign URLs to monitor their execution activity using the same configuration of honeyclients used for locally installed Exploit Kits and live Exploit Kits. The difference in this case, however, is that we do not repeat the probing as we assume that these URLs have a fairly stable activity log apart from changes in page content and client-side code, which we do not analyze directly as we only measure the size in bytes of content delivered to the honeyclient. We also use separate honeyclient instance to probe non Exploit Kit URLs to ensure the sanity of the activity logs we collect.

5.5.4 Model Generation

To verify the Exploit Kit samples (of both locally installed and those live on the Web), we semi-automatically inspect the activity logs we collected to ensure that they drop into the honeyclient at least one exploit payload. Similarly, for non Exploit Kit samples, we manually check for the presence of suspicious samples that might arise due to compromised legitimate websites that might have been hijacked by cyber-criminals and injected with an iframe leading to an Exploit Kit URL. After verifying the samples collected, we extract the 30 features described in Section 5.3. Finally, we label the samples as EK (Exploit Kit) for samples from locally installed and live kits on the Web or as NEK (Non-Exploit Kit) for those which are known to be benign.

Using the labeled samples, we use established supervised learning algorithms (described in Appendix A) to train classifiers and evaluate them to decide which classifier to use for detection. The metrics to evaluate the classifiers is True Positive Rate (TPR) and False Positive Rate (FPR) on the training set using stratified 10-fold cross validation. We used 3 tree-based learning algorithms namely: J48 Decision Tree, Random Tree, and Random Forest. In addition, we also used one stochastic learning algorithm (Bayes Network) and one function-based learning algorithm (Logistic Regression). The details of performance evaluation on these algorithms during training and testing are discussed in Section 5.8.

5.6 Detection

Given an unknown URL, to detect if it leads to an Exploit Kit site, WebWinnow's detection phase works as follows. If resource is a constraint, a pre-filtering component is invoked to check if the URL under examination can be quickly matched against pre-compiled rules based on URL anatomy of popular Exploit Kits. If not, the URL is directly analyzed using the learning-based classification. Note that the pre-filtering component in detection is not necessary if resource is not a constraint. In the rest of this section, we discuss these two components of the detection in more detail.

5.6.1 Rule Based Pre-Filtering

Detecting malicious content using rules based on heuristics that match the payload of malware is commonly used in Anti-Virus engines and Intrusion Detection Systems. When the samples to analyze are resource intensive, fast scanning of the samples saves resources by filtering out easy-to-detect samples. Such a need for a fast but imprecise filtering front-end for the detection of malicious URLs is recognized and used by Provos et al. [70, 71] on finding potentially malicious web pages on the Web and by Canali et al. [13] on web pages that launch drive-by-download pages.

In light of facilitating resource-constrained deployment of our approach, we analyzed the URL anatomy of Exploit Kits and specified pattern recognition rules similar to Intrusion Detection rules. Our analysis of the lexical aspects of the URLs of Exploit Kits shows that most Exploit Kits have reasonably stable URL patterns for the landing pages. Parts of the URLs used for exploit delivery also have predictable patterns except for the dynamically generated elements of the URL parameters. However, the URL patterns vary across different classes of Exploit Kits and in some cases across different versions of an Exploit Kit. For instance, prior to June 2013, the Blackhole Version 2.0.1 exploit delivery URL did not use uppercase letters in parameter names. With the upgrade to Blackhole Version 2.1.0 in June 2013, it started to use uppercase letters in parameter names along with a few changes in the way other parameter values are generated (e.g., using seldom encoding of exploit payload) [47].

To take into account these kinds of variations, we specify separate rule sets for each Exploit Kit and each version of the same Exploit Kit. In the course of specifying the rules, we consider the landing URL, its parameter names, parameter values, and the various parameter encodings used on payloads (e.g., hex-encode). The rule set is updated either when the URL anatomy of an Exploit Kit changes or when a new Exploit Kit is discovered in the wild.

To this end, we compiled rules for 18 currently prevalent Exploit Kits after carefully studying the anatomy of landing pages and exploit delivery URLs of each kit, including the variations in URL anatomy across different versions of the same kit. In addition to compiling the rules ourselves, we also validated and reused some rules made publicly available by other authors [57, 59]. Table 5.3 shows a summary of number of rules for landing page(s) and exploit specific URLs we compiled up until the time of this writing. Notice that the number of rules refers to all the versions of kits for which we were able to analyze their URL anatomy.

5.6.2 Learning-Based Classification

Given an unknown URL, first a honeyclient probes the remote server the URL points to in order to collect activity logs from the beginning to the end of the interaction. From the transactions of the interaction, activity-centric features are extracted and an already trained model is queried to classify the URL as an Exploit Kit URL or not. The main assumption in the activity-centric characterization to distinguish Exploit Kit URLs from other types of URLs is the intrinsic workflow in Exploit Kits that involves client fingerprinting, series of redirections, attempts to deliver and then execute exploit payload, along with typical self-defense behaviors (e.g., cloaking) we already discussed.

Table 5.3: Summary of rules compiled for 18 popular Exploit Kits.

Exploit Kit	Landing URL(s)	Exploit	Total
Blackhole	11	4	15
Cool	2	0	2
CrimeBoss	2	1	3
CritXPack	2	1	3
Fiesta	2	3	5
g01Pack	0	3	3
Impact	1	1	2
Neutrino	1	3	4
Nuclear	1	2	3
Popads	1	4	5
Private	1	0	1
RedKit	1	3	4
SafePack	0	2	2
Sakura	1	0	1
Sofosfo	1	1	2
Styx	3	8	11
SweetOrange	2	3	5
TDS	2	0	2

5.6.3 Implementation Overview

Pre-Filtering. To specify the rules based on the URL anatomy of Exploit Kits, we use YARA[8], an open source, multi-platform malware identification and classification system. YARA provides a rich and flexible rule specification language similar to the struct construct in the C language. Using YARA, one can specify rules that detect strings, instruction sequences, regular expressions, and byte patterns of different encodings. In addition to the rule specification language, YARA provides a scanning engine that can be invoked given rules and samples to identify. For example,

[8]http://code.google.com/p/yara-project/

YARA rules for detecting some landing URLs for the Blackhole Exploit Kit and certain URLs used in dropping malware payloads of the RedKit Exploit Kit might respectively look like the following:

```
rule BLACKHOLE : exploit_kit
{
    strings:
        $bh_url= /.php?.*?:[A-Za-z0-9:]{6,}&.*?&/
    condition:
        $bh_url
}
```

```
rule REDKIT_BINARY : exploit_kit
{
    strings:
        $rk_binary_url = //d{2}.htmls/
    condition:
        $rk_binary_url
}
```

As shown in the above examples, a YARA rule is a block composed of two parts namely: *strings* and *condition*. The first is used to define string patterns through a Perl-Compliant Regular Expression (PCRE) syntax. The second is where we specify the logical expression to match the patterns and regular expressions in the *strings* block. The pre-filtering engine that takes the YARA rules as input to do the matching is implemented in

Python taking advantage of the YARA-Python[9] wrapper.

Learning-Based Component. To probe remote URLs, we use Thug[10], an open source honeyclient written in Python. Thug is configured on a virtual machine image that we replicate as needed. We used the Thug honeyclient for two reasons. First, it is well integrated with the YARA system. Second, and more importantly, the level of granularity in the activity logs captured when probing a remote URL is suitably detailed for the characterization of the typical workflow in Exploit Kits. Before probing Exploit Kit URLs, identifying parameters of the honeyclient personality such as user-agent string (e.g., browser type and version, operating system type and version), referrer, DOM events, and plugins are properly configured not only to simulate a realistic browsing behavior but also to maximize the chance of capturing the end-to-end infection chain that is orchestrated by Exploit Kits.

5.7 Data Collection and Setup

In this section, we discuss the data collection methodology, the dataset, and setup for evaluation.

5.7.1 Data Collection

To collect the dataset we use for our evaluation, we analyzed Exploit Kits in the period April-August 2013 for about 5 months in a contained environment. In this period, we scheduled the honeyclients to periodically visit and record the interactions involved with the Exploit Kits we installed locally. At the same time, we also probed on a daily basis live Exploit Kit URLs reported by online detection services and blacklists.

[9] `https://pypi.python.org/pypi/yara`
[10] `https://github.com/buffer/thug`

Using the locally installed Exploit Kits on the server side, we deployed 4 virtual machines as clients with the Thug honeyclient configured with Internet Explorer 6.0 on Windows XP SP 3 with plugins for Adobe Acrobat Reader (Version 9.1.0), Java plugin (Version 1.6.0.32), and Shockwave Flash (Version 10.0.64.0). We decided to use this user-agent configuration details because it has vulnerabilities that are highly likely to attract Exploit Kits. We noticed that most Exploit Kits check for such combination of browser, operating system, and plugins. As for the DOM events' configuration, we used a standard configuration which enables *load()* and *mousemove()* events.

With the same honeyclient configuration, for live Exploit Kits on the Web, we relied on the URLs reported as Exploit Kit URLs by the URL-Query[11]service. Whenever we find these Exploit Kit sites live, we launch our honeyclient to probe the URL and record the whole transaction. For each live Exploit Kit URL that succeeds in dropping an exploit binary to the honeyclient, we re-probe it after 10 seconds in case of different behavior (e.g., to check if it performs cloaking). It is important to note that collecting the execution dynamics of live Exploit Kits was not an easy task as Exploit Kits have a very short lifetime (usually less than 24 hours). This is because the kit owners relocate them once their URL is blacklisted by detection services. During the data collection, we noticed quite frequently that these URLs are taken down or relocated in a matter of few minutes after they are publicly reported.

The data collection from live Exploit Kits involved 11 kits on a total of 500 Exploit Kits for which we semi-automatically verified the delivery of at least one exploit binary when the Kits are probed. Figure 5.3 shows the percentage distribution of each Exploit Kit in the collected dataset for training. In the top 3 are g01pack (28%), Styx (21%), and Cool (20%)

[11]`http://urlquery.net`

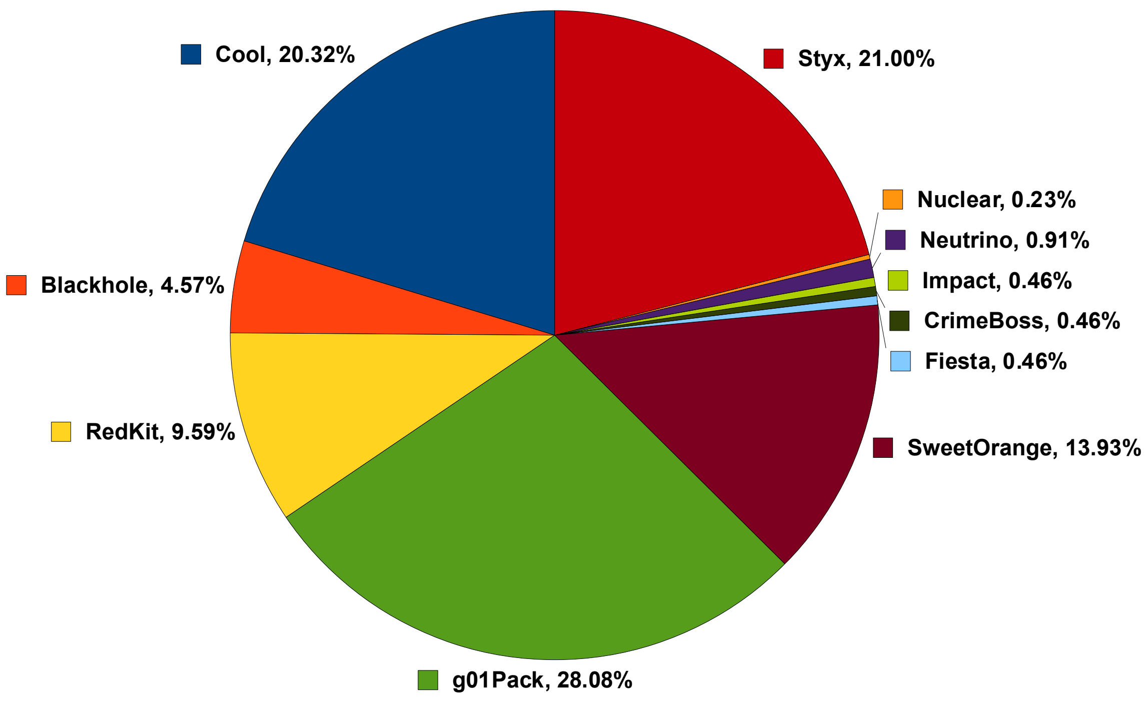


Figure 5.3: Percentage distribution of live Exploit Kits successfully probed during the data collection period.

comprising nearly 70% of the dataset followed by SweetOrange (14%), RedKit (10%), and Blackhole (5%). Interestingly, on June 26, 2013 McAfee released a report [84] that shows correlation with the high percentage of the Styx Exploit Kit in the data we collected. According to the report, from February 2013 to June 2013, a spike in the prevalence of the Styx Exploit Kit was observed. This time-frame overlaps roughly by 3 months (April-June) with our data collection period.

5.7.2 Dataset

For training the classifier, we rely on samples collected from the execution dynamics of the Exploit Kits we installed and executed in a contained environment. In addition, we use live Exploit Kit URLs that we successfully probed on the Web during the data collection period. The majority of these live URLs are collected from `http://urlquery.net` while few more are collected from [8, 90, 91, 97]). For non Exploit Kit samples, we used the Alexa top sites as we, like other approaches [13, 18, 42], assume that these sites (at least those in the top 500) are less likely to host Exploit Kits. The collected dataset is divided into labeled training and testing set as shown in Table 5.4. The training set is used to train different classifiers and evaluate their performance while the testing set is used to evaluate the performance of the trained models on a dataset disjoint with the training set.

To evaluate the pre-filtering engine, we used three datasets of URLs. The first is the Alexa Top 1 Million sites. Secondly, we used the blacklist of Malware Domain List. The third is the set of URLs used in training and testing the classifiers.

Table 5.4: Dataset for training and testing of WebWinnow.

Purpose	**Non Exploit Kits**	**Exploit Kits**	**Total**
Training	500	500	1000
Testing	1117	512	1629

5.7.3 Setup

Training. Using the features collected on the training set, the WEKA machine learning suite [39] was used to train five supervised learning algorithms to derive detection models. The algorithms are: J48 Decision Tree, Random Tree, Random Forest, Bayes Network, and Logistic Regression. The training was done with 10-fold cross validation for all the algorithms.

Testing. On the testing set, the trained classifiers were independently ran and the classification results were analyzed for accuracy and false positives.

Pre-Filtering. First, the rule-based pre-filtering engine was ran against the Alexa Top 1 Million sites and the Malware Domain List URLs collected over an extended period. Secondly, the pre-filtering was ran on both the training and testing set.

5.8 Evaluation

This section discusses the evaluation of WebWinnow in light of the effectiveness of the accuracy of the models generated, accuracy of classifiers on a separate test set, effectiveness of the pre-filtering engine, analysis of false positives, and comparison of WebWinnow with a similar system. More precisely, our evaluation is aimed at answering the following research questions:

RQ$_1$: *Does workflow analysis of Exploit Kits and characterizing them with distinguishing features based on their attack-centric and self-defense behavior improve the detection accuracy of existing techniques in malicious web content detection?*

RQ$_2$: *Since client-side attacks such as drive-by-downloads are mostly initiated by Exploit Kits, can we build a mechanism to combat Exploit Kits in real time?*

RQ$_3$: *In modeling the dynamic behavior of Exploit Kits, can we leverage interaction of a potentially vulnerable client and an Exploit Kit server so as to build better models?*

5.8.1 Analysis of Features

Based on the training dataset we used in this work, we now discuss the statistical significance of the features we presented in Section 5.3.

Aggregate-Behavioral Features. In the training set, the average number of redirections (of all types) for Exploit Kit URLs is 102 while for non Exploit Kit URLs is 82. On average, we found one exploit in the Exploit Kit dataset as opposed to zero in non Exploit Kit dataset. The average number of connections made to remote sources is 25 for Exploit Kit dataset while it is 20 in non Exploit Kits. These aggregate values of features, in fact, are fairly discriminative in putting apart Exploit Kit and non Exploit Kit URLs.

Interaction-Specific Features. While we observed an average of 1 JNLP-based redirection in Exploit Kit URLs, we could not come across even a single JNLP-based redirection attempt in non Exploit Kit URLs. The average number of HTTP 3XX redirections shows that the average is 5 in Exploit Kits compared to an average of 2 in non Exploit Kit dataset. One of the useful features in this class is the *PluginDetect Routines* with an average count of 2 for Exploit Kit dataset as opposed to zero for non Exploit Kit URLs.

Connection-Specific Features. The average number of ActiveX loading attempts is 4 for Exploit Kit dataset while it is 2 for non Exploit Kit dataset. The average number of unique countries traversed and TLDs involved is 4 and 3 respectively for Exploit Kit URLs. Whereas in non Exploit Kit dataset, it is 1 for unique-countries and TLDs.

Content-Specific Features. On average, the size (in bytes) of plain text, HTML, CSS, and XML delivered by Exploit Kit URLs is less than content delivered by non Exploit Kit URLs. On the other hand, the average size of JavaScript, Octet-Stream, and portable document is greater in the case of Exploit Kit URLs. For instance, Octet-Stream average size in Exploit Kit dataset is 13 times that of non Exploit Kit octet-stream content. Similarly, the average JavaScript content delivered by Exploit Kit URLs is about 1.5 times bigger in size than JavaScript in non Exploit Kit URLs. These statistical variations are important indicators of the discriminative power of these features in practice.

5.8.2 Accuracy of Models on Training Set

The performance of the classifiers derived from our training set is summarized in Table 5.5. We use the usual meanings of True Positive Rate (TPR, the percentage of correctly classified URLs for both labels), and False Positive Rate (FPR, the percentage of misclassified URLs for both labels). Using 10-fold cross validation, J48 Decision Tree and Bayes Network classifier have 100% TPR and 0% FPR. In fact, only the Random Tree classifier has the lowest TPR and the highest FPR (misclassification of 29 in 1000 URLs). Overall, the classifiers are quite precise which indicates the discriminative power of the features we derived by leveraging the attack-centric and self-defense behavior of Exploit Kits' workflow.

Table 5.5: WebWinnow: Performance of models on the training set.

Classifier	**TPR**	**FPR**
J48 Decision Tree	100%	0.000
Random Tree	97.1%	0.029
Random Forest	99.8%	0.003
Bayes Network	100%	0.000
Logistic Regression	99.9%	0.001

5.8.3 Accuracy of Classifiers on Testing Set

Table 5.6 shows the accuracy of the classifiers on an independent testing set. Overall, all the classifiers except Random Tree achieved above 99% accuracy with very low FPR. As shown in the results, the classifiers are precise enough to correctly classify samples disjoint with the training set, which shows that our system is effective for realtime detection of malicious URLs hosted by Exploit Kits and hence the results are in favor of $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$.

Table 5.6: WebWinnow: Performance of classifiers on a separate testing set.

Classifier	**TPR**	**FPR**
J48 Decision Tree	99.9%	0.001
Random Tree	89.9%	0.032
Random Forest	99.7%	0.002
Bayes Network	99.8%	0.003
Logistic Regression	99.4%	0.005

5.8.4 Effectiveness of the Pre-Filtering

We first evaluated performance of the rule based pre-filtering on the top 1 million Alexa sites over a randomly selected dates from July 18, 2013 to Sep 14, 2013. Figure 5.4 shows the number of URLs flagged as Exploit Kit out of the top 1 million. Over this period, assuming that the analysis of each URL consumes the same amount of computational resource, the average reduction of performance overhead by the pre-filtering is 99.52%. Hence, if one has to use WebWinnow under a resource-constrained scenario to analyze the top 1 million Alexa sites, an average of only about 0.5% of the URLs are to be forwarded to the resource-intensive honeyclient-based analysis component.

Zooming out the pre-filtering output on August 18, 2013, about 99.49% of the URLs did not match any of the Exploit Kit rules while 5060 URLs were pre-filtered as potentially Exploit Kit URLs. This translates to about 0.51% of the 1 million. Further inspection of the pre-filtered URLs showed that 2138 of the 5060 URLs are related to a URL pattern on youtube.com domain, which is of the form `http://youtube.com/user/USER-NAME` with the `USER-NAME` part referring to the user name of the account owner on youtube.com. Wondering if there are Exploit Kit URLs linked with this URL pattern on *youtube*, we searched for incidents on the Web. Interest-

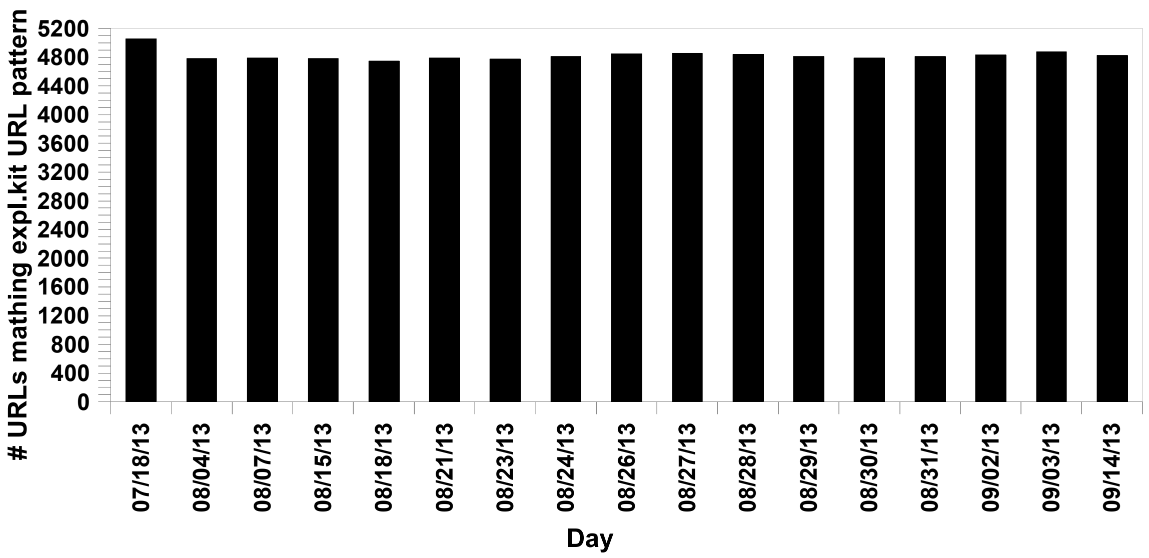


Figure 5.4: WebWinnow: Pre-filtering performance on the Alexa Top 1 Million sites.

ingly, we came across a news [61] reporting about malicious advertisements redirecting visitors of youtube channels to the RedKit Exploit Kit landing page, which seems to correlate with the output of our pre-filtering engine.

Another evaluation of the pre-filtering engine was on a public blacklist of malicious URLs from Malware Domain List[12]. We ran the pre-filtering on this blacklist from Aug 16, 2013 to Sep 14, 2013 and the results turned out to be quite similar for the most part. Of the total of about 86,446 URLs in the database, an averge of about 6658 (7.73%) are flagged as Exploit Kit URLs by our pre-filtering engine. In fact, the database does not label URLs with specific malicious activity and all the URLs in this database are believed to be malicious. Supposing that all the URLs in this database are to be analyzed for maliciousness, our pre-filtering engine reduces the performance overhead by about 8%.

A detailed distribution of the Exploit Kit types in the pre-filtering output is shown in Figure 5.6. As can be seen from the figure, Private, g01Pack, Blackhole, Cool, and TDS comprise the top 5 percentage of the

[12]`http://malwaredomainlist.com/mdl.php`

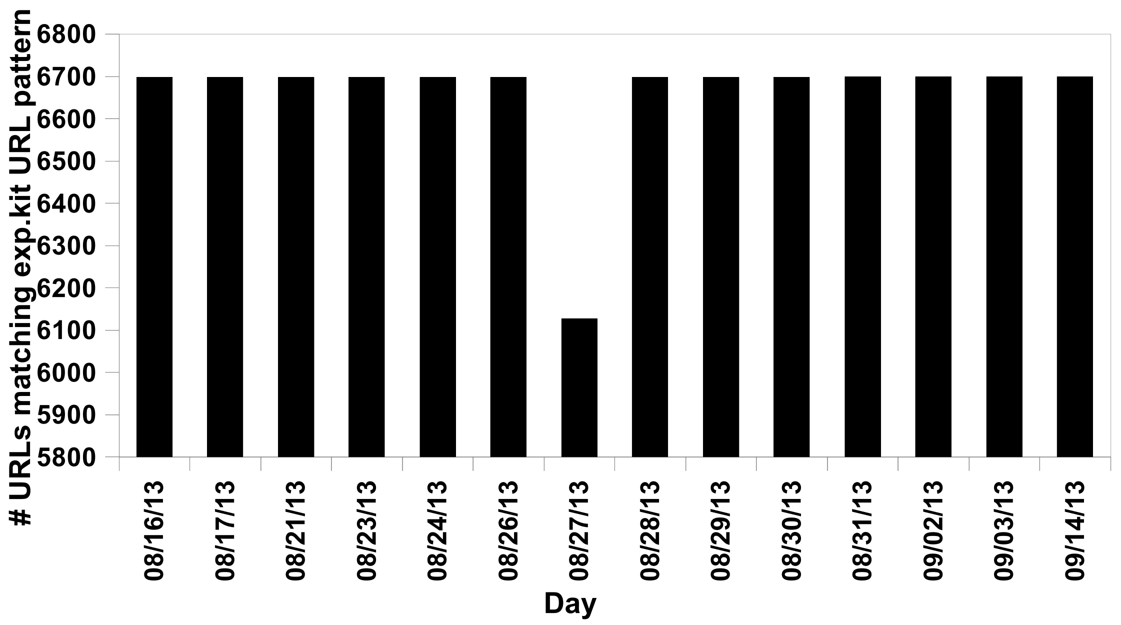


Figure 5.5: WebWinnow: Pre-filtering performance on the Malware Domain Blacklist.

pre-filter followed by RedKit and SweetOrange. This list correlates by 5 kits with the top 6 in the dataset we collected during our experiment (Figure 5.3) with the exception of Private (from Figure 5.7) and Styx (from Figure 5.3).

Table 5.7: WebWinnow: Effectiveness evaluation of the pre-filtering on the dataset used for training and testing of the classifiers.

Class	**Flagged as Exploit Kit**	**Total**
Non Kit URLs	5(0.32%)	1584
Exploit Kit URLs	61(3.4%)	1817

Table 5.7 shows the performance of the pre-filtering engine on the dataset used for training and testing. As can be seen from the table, the pre-fitering engine is pretty accurate in filtering-out the non Exploit Kit URLs with 99.68% accuracy. In this case, only 5 URLs are passed to the more resource-intensive analysis step. As for the Exploit Kit URLs, the pre-filtering

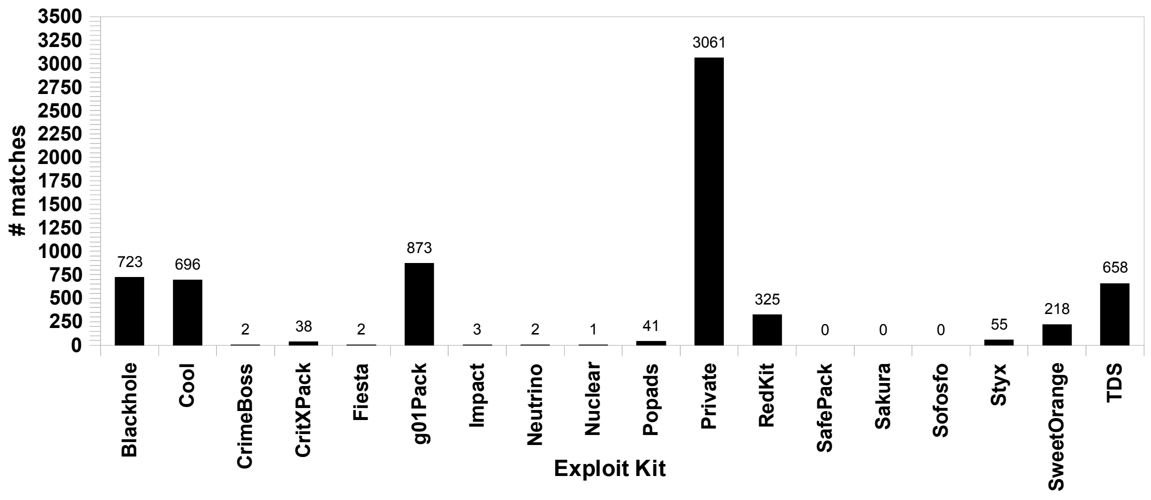


Figure 5.6: Distribution of Exploit Kit types in the pre-filtering output of 6698 URLs from Malware Domain List that matched Exploit Kit URL rules.

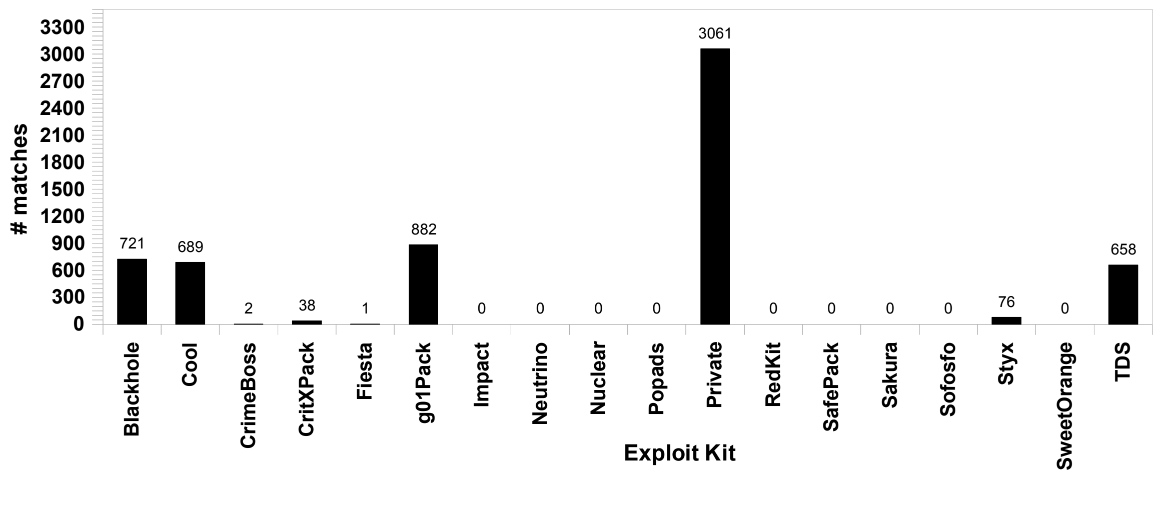


Figure 5.7: Distribution of Exploit Kit types in the pre-filtering output from Malware Domain Blacklist.

identified about 3.4% of the URLs as Exploit Kits, while the rest are to be analyzed by the learning-based component of WebWinnow. Overall, on the whole dataset, the pre-filtering is able to reduce the performance overhead by 48.2%, which is a substantial saving in a resource-constrained deployment of WebWinnow.

Table 5.8: Detection accuracy comparison between WebWinnow and Wepawet.

Class	WebWinnow	Wepawet [**93**]
Benign	63	67
Malicious	31	19
Error	6	14

5.8.5 Error Analysis

Here, using manual analysis, we reason out as to what caused the misclassification of URLs on the testing set.

The J48 Decision Tree classifier misclassified 2 Exploit Kit samples as non Exploit Kit URLs. One of these URLs is a file-sharing website from which users download content by interacting with the Website and it apparently has no redirections up on first encounter. However, when we manually clicked on the "Download' link, we witnessed 3 redirects to lead to a file to download. For the other URL, the (limited) data we had was not sufficient to attribute a clear reason for the misclassification.

The Random Tree classifier, misclassified 23 Exploit Kits as non exploit URLs. As shown in the results, this classifier was not as precise as the others. Manual inspection showed that the feature values for octet-stream and JavaScript content are less than the average size of these features for Exploit Kit URLs in the training set, which, we suspect, might have slightly skewed the classifier.

The Random Forest classifier misclassified only 3 Exploit Kit URLs as non Exploit Kit URLs. In this case, 2 of the URLs span a single country and top-level domain, which is below the average feature value of 4 and 3 respectively for Exploit Kit URLs in the training set.

5.8.6 Comparison with a Similar System

To evaluate how WebWinnow performs in comparison with online malicious content detection systems, we prepared a separate comparison set of 100 URLs reported to be linked with Exploit Kits by the URLQuery[13] service. We then submitted the dataset to WebWinnow and a public service, Wepawet [93], at about the same time. The results are summarized in Table 5.8. Overall, WebWinnow classified 31 URLs as Exploit Kits while Wepawet classified 19 as malicious and suspicious combined (2 and 17 respectively). Of the 31 URLs flagged as Exploit Kits by WebWinnow, Wepawet flagged 2 as malicious, 4 as suspicious, 23 as benign, and 2 as error. Manual analysis of the activity logs of these 31 URLs shows that they have indeed dropped a binary payload to WebWinnow's honeyclient. We speculate that the large number of mismatch with Wepawet's output is probably attributed to the fact that it is a public service and the Exploit Kits might have blacklisted it to serve it with benign response. This mistmatch partially confirmed with Wepawet's detailed reports for 7 of the URLs classified as benign to be pages with HTTP 404 (page not found) response.

Wepawet classified 67 URLs as benign and WebWinnow classified 63 as non exploit of which 33 URLs match the benign, and 12 URLs match the suspicious classification of Wepawet. WebWinnow was not able to finish the analysis of 6 URLs while Wepawet terminated with time-out error on 14 URLs. Overall, this experiment demonstrates that WebWin-

[13] `http://urlquery.net`

NOW is at least as accurate as existing tools in detecting malicious URLs hosted by Exploit Kits.

5.9 Conclusions

Exploit Kits are prevalent on the Internet and they contribute to a significant portion of web-based threats. Understanding how they: function, attack victims, and evade detection systems is crucial in designing techniques to detect them in realtime. In this chapter, we presented an approach that leverages the firsthand observation of the attack-centric and self-defense behavior of Exploit Kits to develop a machine learning approach that precisely detects malicious URLs hosted by Exploit Kits.

We implemented our approach in a tool called WebWinnow with a fast pre-filtering component to allow resource-constrained deployment. Our evaluation of WebWinnow shows that the classifiers we trained based on the features derived from the Exploit Kits' workflow perform very well with very low false positives. Moreover, we have also shown that the pre-filtering engine of WebWinnow substantially reduces the performance overhead when resource is a constraint.

Chapter 6

Related Work

In this chapter, we discuss the most notable body of related work in the context of and in comparison with the approaches presented in this dissertation.

Broadly, there are two complementary techniques that have been proposed to detect malicious activities on the Web, with a focus on malicious web pages. These techniques are:*static analysis* and *dynamic analysis*. The former uses source code of web pages and some static features such as URL structure and host details to characterize malicious payload. The latter focuses on capturing behavior manifested when a page is loaded and executed in a real or emulated browsing environment (e.g., sandbox). A strategy common to both static and dynamic analysis is that they involve extraction of features that allow identification of patterns of malicious payloads in web pages, based on which a classification scheme is devised, usually using machine learning techniques.

The remainder of this chapter is organized as follows. In Section 6.1, we present the related work that use static analysis. Section 6.2 presents dynamic analysis approaches. Finally, in Section 6.3, we discuss approaches which do not fall well into neither static nor dynamic analysis techniques.

6.1 Static Analysis

Lexical elements of a URL string (e.g., query string, path string), host identity information (e.g., WHOIS details, DNS records), and artifacts pertinent to HTML (e.g., iframes, remote links) and JavaScript content (e.g., suspicious native functions) have been demonstrated to be successful artifacts to quickly characterize malicious web pages [13, 14, 53, 54, 55, 79, 89]. The major assumption behind such static aspects of web pages is that statistical distribution of features from the static aspects of malicious web pages tend to differ from the benign counterparts. Once these features are extracted, their values are encoded to train machine learning techniques to build classifiers.

A notable strength of static analysis techniques is the quick extraction of features without loading the web page in the browser. The major limitation of static analysis techniques is that it is difficult to detect attacks that exploit execution of the page. More specifically, a limitation in using only page source is the high risk of obfuscated content (e.g., obfuscated JavaScript) and overlooking malicious JavaScript that exploits vulnerabilities of browser plug-ins (because the attack is initiated when the plugins are invoked). In addition, if we consider the host identity details of fresh (benign) websites, registered by registrars with low reputation, such websites are likely to be misclassified as malicious due to low reputation scores. In effect, there is a high risk of false positives.

Conversely, false negatives may arise as old and well-reputed registrars may host malicious websites which have escaped the static analysis effort. Another cause of false negatives are websites that use free hosting services or already compromised sites with a benign-looking URL and host information. For static analysis techniques that extract lexical URL features to characterize malicious web pages, an attacker may evade these features to confuse detection techniques by carefully crafting malicious URLs to make them look statistically indistinguishable from the benign ones.

Garera et al. [33] developed a framework for the detection and large-scale Internet-wide measurement of phishing attacks. In this work, the authors identify several fine-grained heuristics used to distinguish between a phishing URL and a benign URL. These heuristics are used to train a logistic regression classifier. In addition to heuristics pertinent to obfuscation (e.g., obfuscating with large host names), the authors introduce several general heuristics (e.g., page rank, word-based features) by leveraging the Google index infrastructure. The classification technique achieves an accuracy of 97.3%. In addition to the detection, using 12 days of Google Toolbar URLs, they also conducted a large-scale measurement aimed at shedding light on the prevalence of phishing attacks on the Internet. According to the results of the measurement, on average they found about 777 unique phishing URLs daily and an average of 8.24% of the users who visit phishing pages are likely to be victims of the phishing attack.

Seifert et al. [77] proposed a heuristic-based technique to build signatures of known attack payloads. These signatures are used by Anti-Virus engines or Intrusion Detection Systems to scan a web page and flag it as malicious if its heuristic pattern matches signatures in the database. While precise for known attacks, signatures can be evaded by attackers (mainly with obfuscation); the heuristic also fails to detect novel attack vectors. In addition, the rate at which the signature database of heuristic-based systems is updated is slower than the pace at which attackers overwhelm victims with novel attacks, resulting in zero-day exploits.

Ma et al. [54] proposed a technique based on lightweight extraction and analysis of lexical aspects and host-based information of URLs to derive a model that learns a classifier on a dataset of mostly spam and phishing URLs. The core assumption in this technique is that URL tokens and host-based values of malicious URLs tend to be significantly different from their benign counterparts. According to the experimental results in this work, using Naive Bayes, Support Vector Machines, and Logistic Regression Classifiers on a large dataset of URLs, Logistic Regression gave the best result: 99% detection accuracy with 0.1% FPR on about 30,000 URLs drawn from different public data sources. The approaches we presented in Chapter 3 and Chapter 4 reuse 8 URL features from this work and we introduce three new URL features. Our analysis, however, combines static analysis and lightweight dynamic analysis —using an emulated browser to visit and render the page and execute client-side code upon page load. Moreover, we use 4 more learning algorithms for classification and in Chapter 3, instead of relying on the best classifier, we use confidence-weighted majority vote classification to classify unknown web pages.

In a follow-up work, Ma et al. [55] enhanced the technique they proposed in [54] by applying online learning algorithms over URL and host-based features via live feed of URLs and feature enhancement on the fly. The premise is that batch learning techniques fail to cope with the continuously changing distribution of features over time. The online learning algorithms used in this work are: Perceptron, Logistic Regression with Stochastic Gradient Descent, Passive-Aggressive Algorithm, and Confidence-Weighted Learning. The best classification accuracy (99%) is achieved with Confidence-Weighted algorithm over a balanced data set from a webmail provider. By contrast, in our approach in Chapter 4, we aim at addressing the evolution of web page artifacts using Genetic Algorithms while in this work online learning algorithms are used for the same purpose. Moreover, the underlying analysis and feature set in our approach is not limited to static aspects. Instead, the analysis in our approach spans static, dynamic, and metadata features of URLs.

Hou et al. [42] proposed an obfuscation-resilient approach based on machine learning to detect malicious web content, with a focus on malicious DHTML. The features considered are mainly page content related features (such as text content, native JavaScript functions and objects, ActiveX objects, and iframe size). The approach is based on standard supervised learning algorithms, namely: Naive Bayes, Decision Trees, Support Vector Machines, and Boosted Decision Trees. The Boosted Decision Tree classifier gave the best classification accuracy (96%) with 7.6% FPR on a dataset of 176 malicious and 965 benign samples. The page-source features we presented in Chapter 3 and, in particular the JavaScript features, are similar to the features used in this work. However, our page-source feature set is different in that we re-factor some features (e.g., remote links) into fine-grained features to capture intrinsic details of malicious web pages.

Canali et al. [13] proposed PROPHILER, a pre-filtering mechanism based on machine learning to optimize resource consumption of an expensive dynamic analysis and detection backend system [93]. PROPHILER is a purely static pre-filtering technique that deems web pages as likely malicious or likely benign and it focuses on web pages that launch drive-by-download attacks. PROPHILER achieved a very low false positive rate over a large testing set of URLs using 78 features on URL, host details, HTML, and JavaScript features. While PROPHILER reuses features from [42] and [54], there are 48 new features introduced in this approach and demonstrated to be effective for fast pre-filtering of likely malicious web pages. Unlike PROPHILER where the best classifiers are used for detection, in the approach we presented in Chapter 3, we use confidence-weighted majority vote for classification that relies on seven classifiers. Compared to PROPHILER's dimension of features (78 features), the two approaches we presented in Chapter 3 and Chapter 4 use only 39 features. In fact, all the approaches we presented in this dissertation aim at deeming unknown web pages as either benign or malicious, while in PROPHILER the goal is to deem a web page as likely benign or likely malicious.

Xu et al. [107] proposed JStill, a mostly static approach to detect obfuscated malicious JavaScript code. JStill uses function invocation analysis to capture essential characteristics of obfuscated malicious code. Moreover, it leverages the combination of static analysis and lightweight runtime inspection not only to detect, but also to prevent the execution of obfuscated malicious JavaScript code in browsers. The authors report that evaluation of JStill on real-world malicious JavaScript samples and websites selected from the Alexa top sites demonstrates high detection accuracy and low false positives with negligible performance overhead. From the standpoint of the analysis, JStill resembles the approach we presented in Chapter 3 in that the side-effects of the runtime analysis of JavaScript code are used to enrich the static artifacts.

6.2 Dynamic Analysis

Behavior-based or execution monitoring approaches have been shown to be effective in the analysis and detection of malicious web pages [11, 17, 18, 43, 65, 72, 75, 78]. Such techniques could be deployed at a proxy-level (e.g., [65]) to intercept requests from or responses to the user, visit the URL in a controlled environment (e.g., disposable virtual machine), analyze its execution dynamics for hints of malicious activity (e.g., unusual process creation, repeated redirection), and decide if it is safe to render the page in the browser. Alternatively, client-side sandboxing of critical page content (e.g., JavaScript code) could be used to log critical actions (e.g., invoking a plugin) and match logs with known patterns of malicious activities (e.g., as in [22]).

Among dynamic analysis techniques, *honeyclients* [72] are the predominantly adopted systems that mimic a human visitor and use a dedicated sandbox environment (e.g., virtual machine) to visit a web page. When a page is rendered, the execution dynamics is captured and analyzed to infer evidences for attack payloads. Honeyclients are of two types, namely: low-interaction and high-interaction. Honeyclients that use simulated browser and minimal OS features for rendering a web page are called low-interaction honeyclients while those that use real browser and full OS features for rendering a web page are termed as high-interaction honeyclients.

Low-interaction honeyclients (e.g., HoneyC [41], Monkey-Spider[1], PhoneyC[2], Thug[3]) are typically limited to monitoring the traces of activities during the interaction against pre-defined signatures. As a result, they can not detect zero-day exploits due to the static nature of the reference signatures. On the contrary, high-interaction honeyclients (e.g., Capture-HPC [88], MITRE HoneyClient [64], Microsoft HoneyMonkey [99], Shelia[4]) check integrity changes in system states. The integrity check requires monitoring: file system, registry entries, processes, network connection, and anomalies in memory and CPU consumption. The inherent advantage of honeyclients, especially high-interaction ones, is the deep insight they provide on the malicious activities embedded in malicious web pages.

While effective at detecting daunting malicious web pages, dynamic analysis approaches including honeyclients are resource intensive. This is because they need to load and execute individual pages under analysis and modern web pages are usually stuffed with rich client-side code and multimedia which take a long analysis time. Moreover, not all web pages are likely to launch attacks upon visiting. There are web pages which demand user interaction or wait for time/logic-bomb to take action. From the

[1]`http://monkeyspider.sourceforge.net`
[2]`http://code.google.com/p/phoneyc/`
[3]`https://github.com/buffer/thug`
[4]`http://www.cs.vu.nl/~herbertb/misc/shelia/`

evadability point of view, IP addresses of honeyclients can be black-listed by malicious servers, their virtual machines be detected through advanced fingerprinting techniques, and they may also be victims of Turing verifications that necessarily require the action of human visitor.

Wang et al. [99] designed and implemented a system called Strider HoneyMonkey, which consists of an array of monkey programs running possibly vulnerable browsers on virtual machines with different patch levels. The goal is to patrol the Web to discover and classify websites that exploit browser vulnerabilities. The classification is done by correlating unusual changes to system sates (e.g., unauthorized file creation) to a successful exploit. Our approach in Chapter 5 resembles Strider HoneyMonkey in using honeyclients. However, instead of changes in system state, we focus on the actual activities when a web page is executed in a honeyclient system to detect malicious behavior.

Moshchuk et al. [65] proposed SpyProxy, an execution-based Web content analysis system to protect users from Internet-borne malware. The SpyProxy system intercepts and evaluates web content in transit from web servers to the browser. When a browser requests a web page, the proxy front end intercepts the request, retrieves the root page, and statically analyzes it for safety. If the root page cannot be deemed safe based on the static analysis, the front end forwards the URL to a Virtual Machine worker. A browser in the Virtual Machine downloads and renders the page content. If the page is safe, the Virtual Machine notifies the front end, and the page content is released to the browser. In case the page under analysis has already been cached and was previously determined to be safe, the front end forwards it directly to the client. According to the authors, the evaluation of SpyProxy shows that it detected every malware threat to which it was exposed, with a minimal delay of only 600 milliseconds to analyze a web page.

Ratanaworabhan et al. [74] designed NOZZLE, an approach to detect heap spraying attacks. NOZZLE intercepts calls to the browser's memory manager and tries to detect heap spraying attacks by observing the objects on the heap. NOZZLE treats local objects as they were code and tries to interpret them, thus detecting potentially malicious code. The NOZZLE lightweight emulator scans these objects for valid x86 code. Once found, such code sequences are disassembled and a control flow graph is built, which can then be analyzed using network packet processing methods. NOZZLE is integrated into a browser and is hence easy to use even for inexperienced users. Moreover, it has a very low false positive rate and at the same time detects heap spraying attacks very effectively. According to the authors, NOZZLE was able to detect all heap spraying attacks it was evaluated against. Even though NOZZLE protects quite well against heap spraying attacks, these are not the only attacks Web users have to be protected against.

Kapravelos et al. [49] examined the security model that high-interaction honeyclients use and evaluated their weaknesses in practice. They introduced and discussed a number of possible attacks, and tested them against several popular, well-known high-interaction honeyclients. In particular, they have introduced three novel attack techniques (JavaScript-based honeyclient detection, in-memory execution, and whitelist-based attacks) and analyzed known attacks. These attacks evade the detection of the tested honeyclients, while successfully compromising regular visitors. Furthermore, they suggest several countermeasures aiming to improve honeyclients. By employing these countermeasures, a honeyclient will be better protected from evasion attempts and will provide more accurate results.

Cova et al. [18] developed an approach that aims at analyzing malicious JavaScript code with the goal of detecting drive-by-download attacks. The approach combines emulation, anomaly detection, and machine learning for detecting drive-by-download attacks. Based on this approach they built WEPAWET [93], an emulation-based dynamic analysis and detection framework for malicious content (mainly malicious JavaScript and malware). WEPAWET is available as a public service. WEPAWET is reported by the authors to have a low false negative rate, particularly for drive-by-download web pages.

Dewald et al. [22] proposed ADSANDBOX, a client-side JavaScript sandboxing and heuristics-based analysis technique that executes JavaScript embedded in a page within an isolated environment and logs every critical action to detect malicious web pages. ADSANDBOX uses the Mozilla JavaScript engine SPIDERMONKEY [66] to execute JavaScript code and log every action during the execution. For detection, it uses heuristics (in a form of regular expressions). The approach is implemented as a Browser Helper Object (BHO) on the Internet Explorer browser. Whenever malicious content is detected, the user is warned by the BHO and the malicious content is blocked unless the user explicitly wants to load it anyway. ADSANDBOX achieved false positive close to zero but at a high performance overhead.

Rieck et al. [75] proposed CUJO, a system embedded in a web proxy for efficient detection of drive-by-downloads with an emphasis on malicious JavaScript. The authors use Support Vector Machines to characterize, analyze, and detect drive-by-download attacks using a combination of static and dynamic aspects of JavaScript code. CUJO is reported to achieve about 95% detection rate with low false alarms and an average analysis time of 0.5 seconds per a single web page. In Chapter 3 and Chapter 4, we also use Support Vector Machines among other learning algorithms with some static as well as dynamic features of JavaScript code (in particular suspicious JavaScript function such as `eval()`, `unescape()`, `fromCharCode()`, and `createElement()`). Nevertheless, the approaches we presented in this dissertation are attack-type-agnostic to target a wider array of attacks.

Heiderich et al. [40] built IceSheild, a lightweight in-browser system to perform in-line dynamic JavaScript code analysis and de-obfuscation. They use features of ECMA Script 5 to freeze DOM object properties so that objects cannot be modified at runtime.

Curtsinger et al. [20] proposed Zozzle, a low-overhead JavaScript malware detection system that combines static and dynamic analysis. Zozzle is based on a Bayesian classifier using features extracted from the hierarchical aspects of the Abstract Syntax Tree of JavaScript code. Because Zozzle is trained using samples collected by Zozzle [74], it is more effective in detecting heap spray attacks. The authors report that Zozzle is able to detect JavaScript malware through mostly static code analysis effectively and it has an extremely low false positive rate of 0.0003%.

Seifert et al. [78] proposed Rozzle, a JavaScript multi-execution virtual machine, as a way to explore multiple execution paths within a single execution so that environment-specific malware will reveal itself. Using large-scale experiments, the authors report that Rozzle increases the detection rate for offline runtime detection by almost seven times. In addition, Rozzle triples the effectiveness of online runtime detection. Rozzle is also reported to incur virtually no runtime overhead.

Thomas et al. [89] reused the static analysis portion from [54] and [55] with an introduction of a dynamic analysis component to design a system called Monarch, a real-time system that crawls URLs as they are submitted to web services to determine whether the URLs direct to spam content.

Xu et al. [106] leveraged URL features, host-based features, and features derived from network-traffic (based on execution). This approach is similar to the approach we presented in Chapter 3 and 4 in its aim, i.e., combining static and dynamic analysis for effective detection of malicious web pages. However, in place of the the minimalistic emulation we use in our approaches, the authors use network traffic. While we use an emulated browser, they use honeyclient.

6.3 Other Techniques

Kotov and Massacci [50] conducted a preliminary analysis of the source code of 30 different Exploit Kits. As per their analysis, the key strength of Exploit Kits is the functionalities to support the kit owner to manage exploits, escape detection, and monitor traffic. On the other hand, they conclude that Exploit Kits use limited number of and unsophisticated vulnerabilities focusing on large volume of infection traffic to maximize the rate of successful infection of victims. One of the findings in this work is the fact that Exploit Kits have similar functionality, which also coincides with our preliminary analysis. Nonetheless, in the approach we presented in Chapter 5, we analyzed 38 different Exploit Kits and versions of some of the Exploit Kits in our set are different from theirs. Our motivation of the analysis is to leverage Exploit Kit workflow to design a detection technique while the authors' goal is to analyze and characterize Exploit Kits as software artifacts.

Allodi et al. [2] discussed MalwareLab, an experience from a controlled experimental evaluation of resilience of 10 Exploit Kits with respect to changes in software configuration (e.g., browser personality of victims). As per the findings, there seem to be two types of Exploit Kits with regards to resilience to changes in software configuration. One type are Exploit Kits designed to be effective for an extended period of time at the expense of lower infection rate. The other type are Exploit Kits designed to be effective in infecting as many victims as possible within a very short period, at the expense of low resilience. The analysis of Exploit Kits used in our approach in Chapter 5 is similar to MalwareLab in some aspects, such as the use of virtual machines and automation of execution when feasible. However, our goal, as already discussed, is to have a deeper insight as to how the Exploit Kits function to perform attacks and to evade detection techniques.

Grier et al. [37] conducted a large-scale analysis on the emergence of exploit-as-a-service model on the malware ecosystem by examining the current landscape of drive-by-downloads on the Internet. For the analysis, the authors analyze 77,000 malicious URLs from Google Safe Browsing and a crowd-sourced feed of blacklisted URLs known to lead to Exploit Kits. These URLs led to over 10,000 distinct executables. The results of contained execution of these binaries show about 32 families of malware (among which are the prominent ones) are carried around on the Web via drive-by-downloads supported by Exploit Kits (with Blackhole accounting for 29% of all malicious URLs). In addition, the authors also used passive DNS data to conclude that the infrastructure that hosts these Exploit Kits is short-lived (receives traffic only for about 2.5 hours). Although we did not measure the lifetime of Exploit Kits in Chapter 5, we also noticed that they do not live long.

Chapter 7

Conclusions and Future Work

In this chapter, we summarize the problems addressed in this dissertation, highlight approaches we proposed, and finally discuss possible lines of future work.

7.1 Conclusions

Cyber-criminals, motivated by illegal financial gain, abuse the convenience and flexibility of the Web through malicious activities targeting vulnerable browsing environments and unsuspecting victims. The malicious activities are initiated when an unsuspecting victim visits a web page. Just after a mere visit of a web site, the victim may lose its invaluable credentials (e.g., credit card details) or get its device infected with malware that is used in orchestrating more complex attacks.

While the current state-of-the-art to mitigate malicious activities on the Web is commendable, the effectiveness of existing defense mechanisms is impeded by (1) partial and course-grained analysis and characterization of attack payloads (2) evolution of web page artifacts which renders detection models out-of-date and (3) prevalence of exploit kits that spread web-borne malware. This dissertation tackled these problems and presented approaches to address them with a common goal of effective analysis, characterization, and detection of malicious activities on the Web.

To address the problem of partial and course-grained analysis and characterization of attack payloads, we presented, in Chapter 3, a holistic approach that leverages static analysis and minimalistic emulation to detect malicious web pages. In this approach, we introduced and demonstrated the effectiveness of 10 novel features and we used confidence-weighted majority vote classification. We showed, through a large-scale evaluation, that the approach works well both in terms of detection accuracy and its reasonably low performance overhead.

To address the evolution of web page artifacts, we presented, in Chapter 4, an evolution-aware approach that exploits evolutionary searching and optimization using Genetic Algorithm to ensure that, inline with the evolution of web page artifacts, the best combination of features and learning algorithms are used in detecting malicious web pages. We demonstrated the significance of the GA-guided detection model in reducing the false negatives up to 10.5% on a fairly large-scale experimental corpus of web pages.

On the front of defending Web users from Exploit Kits, we presented, in Chapter 5, an approach in which we (1) analyzed the source code and runtime behavior of 38 distinct Exploit Kits in a contained setting (2) probed live Exploit Kits on the Web in pursuit of capturing their typical workflow (3) combined the observations of the analyses to leverage the attack-centric and self-defense characteristics of Exploit Kits to draw distinguishing features to characterize Exploit Kits and (4) trained precise classifiers to automatically detect malicious URLs linked with Exploit Kits. By evaluating our approach, we demonstrated that it is possible to very precisely detect malicious URLs hosted by Exploit Kits.

7.2 Future Work

With the aim of evading web-borne malicious activity detection systems, cyber-criminals delay the actual exploit when the page loads. Delayed exploits wait for conditions to execute attack payloads. Typically, the condition is either a time bomb (e.g., after 5 seconds of page load) or a logic bomb (e.g., on first mouse click). Both time bombs and logic bombs are intrinsically random –which makes the characterization and detection of delayed exploit a difficult task. Hence, one interesting direction of future work is to investigate ways in which delayed exploits can be characterized and then detected.

Another line of future work is regarding the integration of Binspect, Einspect, and WebWinnow. We plan to combine the three systems to eventually deploy them as a framework to analyze and detect web-borne malicious activities.

Bibliography

[1] Alexa. Alexa Top 500 Global Websites. http://www.alexa.com/topsites, July 2011.

[2] Luca Allodi, Vadim Kotov, and Fabio Massacci. MalwareLab - Experimentation with Cybercrime Attack Tools. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2013.

[3] Jesse Alpert and Nissan Hajaj. We Knew the Web Was Big... http://googleblog.blogspot.it/2008/07/we-knew-web-was-big.html, July 2008.

[4] Internet Archive. HERITRIX Crawler. http://crawler.archive.org/index.html, July 2012.

[5] Armorize. Mysql.com Hacked: Infecting Visitors with Malware. http://blog.armorize.com/2011/09/mysqlcom-hacked-infecting-visitors-with.html, September 2011.

[6] Tim Berners-Lee and Mark Fischetti. *Weaving the Web*. HarperOne, Secaucus, NJ, USA, 1999.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] Malware Blacklist. Malware Blacklist. `http://www.malwareblacklist.com/showMDL.php`, June 2013.

[9] Leo Breiman and Adele Cutler. Random Forests. `http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.html`, October 2013.

[10] Interactive Advertising Bureau. IAB Internet Advertising Revenue Report. `http://www.iab.net/media/file/IAB_Internet_Advertising_Revenue_Report_HY_2013.pdf`, October 2013.

[11] K. Byung-Ik, I. Chae-Tae, and J. Hyun-Chul. Suspicious Malicious Web Site Detection with Strength Analysis of a JavaScript Obfuscation. In *International Journal of Advanced Science and Technology*, pages 19–32, 2011.

[12] Davide Canali and Davide Balzarotti. Behind the Scenes of Online Attacks: An Analysis of Exploitation Behaviors on the Web. In *NDSS 2013, 20th Annual Network and Distributed System Security Symposium*, February 2013.

[13] Davide Canali, Marco Cova, Giovanni Vigna, and Christpher Kruegel. Prophiler: A Fast Filter for the Large-Scale Detection of Malicious Web Pages. In *Proceedings of the 20th international conference on World Wide Web*, WWW'11, pages 197–206. ACM, 2011.

[14] Hyunsang Choi, Bin B. Zhu, and Heejo Lee. Detecting Malicious Web Links and Identifying their Attack Types. In *Proceedings of the 2nd USENIX conference on Web application development*, WebApps'11, pages 11–11. USENIX Association, 2011.

[15] Credit Counselors Corporation. Phishing Scams Lead to Iden-

tity Theft. http://www.cccindy.com/credit-counseling-blog/phishing-scams-lead-to-identity-theft/, October 2013.

[16] M. Cova, C. Kruegel, and G. Vigna. There is No Free Phish: An Analysis of Free and Live Phishing Kits. In *Proceedings of the USENIX Workshop On Offensive Technologies (WOOT)*, San Jose, CA, August 2008.

[17] Marco Cova. *Taming the Malicious Web: Avoiding and Detecting Web-based Attack*. PhD thesis, University of California, Santa Barbara, 2010.

[18] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 281–290, New York, NY, USA, 2010. ACM.

[19] Marco Cova, Corrado Leita, Olivier Thonnard, Angelos D. Keromytis, and Marc Dacier. An Analysis of Rogue AV Campaigns. In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, RAID'10, pages 442–463, Berlin, Heidelberg, 2010. Springer-Verlag.

[20] Charlie Curtsinger, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 3–3, Berkeley, CA, USA, 2011. USENIX Association.

[21] Dancho Danchev. Scareware Pops-Up at FoxNews. http://www.zdnet.com/blog/security/scareware-pops-up-at-foxnews/3140, April 2009.

[22] Andreas Dewald, Thorsten Holz, and Felix C. Freiling. ADSandbox: Sandboxing JavaScript to Fight Malicious Websites. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1859–1864, New York, NY, USA, 2010. ACM.

[23] DMOZ. Open Directory Project. `http://www.dmoz.org/`, September 2011.

[24] Manuel Egele, Engin Kirda, and Christopher Kruegel. Mitigating Drive-by Download Attacks: Challenges and Open Problems, 2009.

[25] Manuel Egele, Gianluca Stringhini, Christopher Krgel, and Giovanni Vigna. COMPA: Detecting Compromised Accounts on Social Networks. In *Proceedings of the Network and Distributed System Security Symposium*. The Internet Society, 2013.

[26] Birhanu Eshete. Effective Analysis, Characterization, and Detection of Malicious Web Pages. In *Proceedings of the 22nd international conference on World Wide Web companion*, WWW '13 Companion, pages 355–360, Rio de Janeiro, Brazil, 2013. ACM.

[27] Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam. Malicious Website Detection: Effectiveness and Efficiency Issues. In *Proceedings of SysSec Workshop*, pages 123–126. IEEE, 2011.

[28] Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam. BINSPECT: Holistic Analysis and Detection of Malicious Web Pages. In *Proceedings of Security and Privacy in Communication Networks*, SecureComm'12, pages 149–166. Springer-Verlag, 2012.

[29] Birhanu Eshete, Komminist Weldemariam, Adolfo Villafiorita, and Mohammad Zulkernine. EINSPECT: Evolution-Guided Analysis and

Detection of Malicious Web Pages. In *Proceedings of the 37th Annual International Computer Software & Applications.*, pages 375–380, Kyoto, Japan, July 2013. IEEE.

[30] Facebook. Facebook Graph API. `https://developers.facebook.com/docs/reference/api/`, March 2012.

[31] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L. Wiener. A Large-Scale Study of the Evolution of Web Pages. In *Proceedings of the Twelfth Conference on World Wide Web*, Budapest, Hungary, 2003. ACM Press.

[32] Sean Ford, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Analyzing and Detecting Malicious Flash Advertisements. In *ACSAC*, pages 363–372, 2009.

[33] Sujata Garera, Niels Provos, Monica Chew, and Aviel D. Rubin. A Framework for Detection and Measurement of Phishing Attacks. In *Proceedings of the 2007 ACM workshop on Recurring malcode*, WORM '07, pages 1–8, New York, NY, USA, 2007. ACM.

[34] Sally A. Goldman, Manfred K. Warmuth, and David Haussler. Learning Binary Relations Using Weighted Majority Voting. In *Machine Learning*, pages 453–462. ACM Press, 1995.

[35] Google. Google Safe Browsing API. `http://code.google.com/apis/safebrowsing/`, August 2011.

[36] Google. Google Plus URL share count API. `https://clients6.google.com/rpc`, July 2012.

[37] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair

Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Manufacturing Compromise: The Emergence of Exploit-as-a-Service. In *Proceedings of the 19th ACM Conference on Computer and Communication Security*, October 2012.

[38] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11, 2009.

[39] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. Weka 3: Data Mining and Open Source Machine Learning Software in Java. `http://www.cs.waikato.ac.nz/ml/weka/`, July 2013.

[40] Mario Heiderich, Tilman Frosch, and Thorsten Holz. IceShield: Detection and Mitigation of Malicious Websites with a Frozen DOM. In *Proceedings of the 14th international conference on Recent Advances in Intrusion Detection*, RAID'11, pages 281–300, Berlin, Heidelberg, 2011. Springer-Verlag.

[41] HoneyClient Project. Honeyc. `https://projects.honeynet.org/honeyc`, July 2011.

[42] Yung-Tsung Hou, Yimeng Chang, Tsuhan Chen, Chi-Sung Laih, and Chia-Mei Chen. Malicious Web Content Detection by Machine Learning. *Expert Syst. Appl.*, 37(1):55–60, January 2010.

[43] Ali Ikinci, Thorsten Holz, and Felix C. Freiling. Monkey-Spider: Detecting Malicious Websites with Low-Interaction Honeyclients. In *Sicherheit*, volume 128 of *LNI*, pages 407–421. GI, 2008.

[44] Yahoo Inc. Yahoo Random URL Generator. `http://random.yahoo.com/bin/yrl/`, October 2011.

[45] Internet World Stats. World internet usage and population statistics. `http://www.internetworldstats.com/stats.htm`, June 2012.

[46] Danesh Irani, Marco Balduzzi, Davide Balzarotti, Engin Kirda, and Calton Pu. Reverse Social Engineering Attacks in Online Social Networks. In *Proceedings of the 8th international conference on Detection of intrusions and malware, and vulnerability assessment*, DIMVA'11, pages 55–74. Springer-Verlag, 2011.

[47] Kafiene. Blackhole Exploit Kit goes 2.1.0: Shows New URL Patterns. `http://malware.dontneedcoffee.com/2013/06/blackhole-exploit-kit-goes-210-shows.html`, June 2013.

[48] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna. Revolver: An Automated Approach to the Detection of Evasive Web-based Malware. In *USENIX Security*, 2013.

[49] Alexandros Kapravelos, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Escape from Monkey Island: Evading High-Interaction Honeyclients. In *Proceedings of the 8th international conference on Detection of intrusions and malware, and vulnerability assessment*, DIMVA'11, pages 124–143. Springer-Verlag, 2011.

[50] Vadim Kotov and Fabio Massacci. Anatomy of Exploit Kits - Preliminary Analysis of Exploit Kits as Software Artefacts. In *ESSoS*, pages 181–196, 2013.

[51] Robert Lemos. MPack Developer on Automated Infection Kit. `http://www.theregister.co.uk/2007/07/23/mpack_developer_interview/`, July 2007.

[52] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 674–686, New York, NY, USA, 2012. ACM.

[53] Justin Ma. *Learning to Detect Malicious URLs*. PhD thesis, University of California, San Diego, 2010.

[54] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 1245–1254, New York, NY, USA, 2009. ACM.

[55] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying Suspicious URLs: An Application of Large-Scale Online Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 681–688, New York, NY, USA, 2009. ACM.

[56] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[57] Malc0de. Malc0de. `http://twitter.com/malc0de`, July 2013.

[58] Malforsec. Neutrino Exploit Kit Landing Page Demystified. `http://malforsec.blogspot.no/2013/03/neutrino-exploit-kit-landing-page.html`, March 2013.

[59] MalwareSigs. MalwareSigs: Helping Network Analysts Detect Malware. `http://www.malwaresigs.com`, July 2013.

[60] MalwareURL. Malware URLs. http://www.malwareurl.com/, September 2011.

[61] MalwarSigs. Malvertising on Youtube.com Redirects to Sweet Orange Exploit Kit. http://www.malwaresigs.com/2013/07/30/malvertising-on-youtube-com-redirects-to-sweet-orange-ek/, July 2013.

[62] McAfee. McAfee Site Advisor. http://www.siteadvisor.com, Sep 2013.

[63] Trend Micro. Web Threats. http://apac.trendmicro.com/apac/threats/enterprise/web-threats/, November 2012.

[64] MITRE. The MITRE HoneyClient Project. http://search.cpan.org/~mitrehc, November 2011.

[65] Alexander Moshchuk, Tanya Bragin, Damien Deville, Steven D. Gribble, and Henry M. Levy. SpyProxy: Execution-Based Detection of Malicious Web Content. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 3:1–3:16, Berkeley, CA, USA, 2007. USENIX Association.

[66] Mozilla. SpiderMonkey. https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey, October 2013.

[67] NetCraft. December 2012 Web Server Survey. http://news.netcraft.com/archives/2012/12/04/december-2012-web-server-survey.html, December 2012.

[68] PhishTank. PhishTank Developer Information. http://www.phishtank.com/developer_info.php, September 2011.

[69] Niels Provos. Safe Browsing: Protecting Web Users for 5 Years and Counting. `http://googleonlinesecurity.blogspot.it/2012/06/safe-browsing-protecting-web-users-for.html`, June 2012.

[70] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The Ghost in the Browser: Analysis of Web-Based Malware. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, pages 4–4. USENIX Association, 2007.

[71] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iFRAMEs point to Us. In *Proceedings of the 17th conference on Security symposium*, SS'08, Berkeley, CA, USA, 2008. USENIX Association.

[72] Mahmoud Qassrawi and Hongli Zhang. Detecting Malicious Web Servers with Honeyclients. *Journal of Networks*, 6(1), 2011.

[73] Moheeb Abu Rajab, Lucas Ballard, Nav Jagpal, Panayiotis Mavrommatis, Daisuke Nojiri, Niels Provos, and Ludwig Schmidt. Trends in Circumventing Web-Malware Detection. Technical report, Google Inc., July 2011.

[74] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin Zorn. NOZZLE: A Defense Against Heap-Spraying Code Injection Attacks. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 169–186, Berkeley, CA, USA, 2009. USENIX Association.

[75] Konrad Rieck, Tammo Krueger, and Andreas Dewald. Cujo: Efficient Detection and Prevention of Drive-by-Download Attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 31–39, New York, NY, USA, 2010. ACM.

[76] Toby Segaran. *Programming Collective Intelligence.* O'Reilly, first edition, 2007.

[77] C. Seifert, I. Welch, and P. Komisarczuk. Identification of Malicious Web Pages with Static Heuristics. In *Proceedings of the Australasian Telecommunication Networks and Applications Conference*, pages 91–96, 2008.

[78] C. Seifert, B. Zorn, B. Livshits, and C. Kolbitsch. Rozzle: Decloaking Internet Malware. *2012 IEEE Symposium on Security and Privacy*, 0:443–457, 2012.

[79] Christian Seifert, Ian Welch, Peter Komisarczuk, Chiraag Uday Aval, and Barbara Endicott-Popovsky. Identification of Malicious Web Pages Through Analysis of Underlying DNS and Web Server Relationships. In *Local Computer Networks LCN*, pages 935–941. IEEE, 2008.

[80] Daily SEO. Facebook and Twitter's Influence on Google's Search Rankings. `http://www.seomoz.org/blog/facebook-twitters-influence-google-search-rankings`, May 2012.

[81] Gargoyle Software. HTMLUnit. `http://htmlunit.sourceforge.net/`, March 2012.

[82] Aditya K. Sood and Richard J. Enbody. Crimeware-as-a-ServiceA Survey of Commoditized Crimeware in the Underground Market. *International Journal of Critical Infrastructure Protection*, 06(1):28–38, March 2013.

[83] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and

Giovanni Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 635–647, New York, NY, USA, 2009. ACM.

[84] Santosh Surgihalli and Varadharajan Krishnasamy. Styx Exploit Kit Takes Advantage of Vulnerabilities. `http://blogs.mcafee.com/mcafee-labs/styx-exploit-kit-takes-advantage-of-vulnerabilities`, June 2013.

[85] Symantec. Symantec Report on Attack Kits and Malicious Websites. `http://symantec.com/content/en/us/enterprise/other_resources/b-symantec_report_on_attack_kits_and_malicious_websites_21169171_WP.en-us.pdf`, July 2011.

[86] Symantec. Symantec Web based Attack Prevalence Report. `http://www.symantec.com/business/threatreport/topic.jsp?id=threat_activity_trends&aid=web_based_attack_prevalence`, July 2011.

[87] Symantec. Internet Security Threat Report. `http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v18_2012_21291018.en-us.pdf`, April 2013.

[88] The HoneyClient Project. Capture-hpc. `https://projects.honeynet.org/capture-hpc`, October 2011.

[89] Kurt Thomas, Chris Grier, Justin Ma, Vern Paxson, and Dawn Song. Design and Evaluation of a Real-Time URL Spam Filtering Service. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2011.

[90] Spy Eye Tracker. Spy Eye Tracker. `https://spyeyetracker.abuse.ch/monitor.php?browse=binaries`, June 2013.

[91] Zues Tracker. Zues Tracker. `https://zeustracker.abuse.ch/monitor.php?browse=binaries`, June 2013.

[92] Twitter. Twitter URL API. `http://urls.api.twitter.com/1/urls/`, March 2012.

[93] UCSB. Wepawet. `http://wepawet.cs.ucsb.edu`, Sep 2013.

[94] Blase E. Ur and Vinod Ganapathy. Evaluating Attack Amplification in Online Social Networks. In *W2SP'09: 2009 Web 2.0 Security and Privacy Workshop*, Oakland, California, May 2009.

[95] Ashlee Vance. Times Web Ads Show Security Breach. `http://www.nytimes.com/2009/09/15/technology/internet/15adco.html?_r=0`, Sep 2009.

[96] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[97] VxVault. Vx Vault. `http://vxvault.siri-urz.net/ViriList.php`, June 2013.

[98] Gang Wang, Jack W. Stokes, Cormac Herley, and David Felstead. Detecting Malicious Landing Pages in Malware Distribution Networks. In *DSN*, pages 1–11. IEEE, 2013.

[99] Yi Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites that Exploit Browser Vulnerabilities. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS'06, 2006.

[100] WebSense. eWeek Web Site Leads Users to Rogue Anti-Virus (AV) Application. `http://securitylabs.websense.com/content/alerts/3310.aspx`, February 2009.

[101] WebSense. Websense 2010 Threat Report. `http://www.websense.com/content/threat-report-2010-highlights.aspx/`, July 2011.

[102] Yi wei Chen and Chih jen Lin. Combining SVMs with Various Feature Selection Strategies. In *Taiwan University*. Springer-Verlag, 2005.

[103] Aaron Weiss. Top 5 Security Threats in HTML5. `http://www.esecurityplanet.com/trends/article.php/3916381/Top-5-Security-Threats-in-HTML5.htm`, October 2011.

[104] Darrell Whitley. A Genetic Algorithm Tutorial. *Statistics and Computing*, 4:65–85, 1993.

[105] Colin Whittaker, Brian Ryner, and Marria Nazif. Large-Scale Automatic Classification of Phishing Pages. In *Network and Distributed Systems Security*. The Internet Society, 2010.

[106] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. Cross-Layer Detection of Malicious Websites. In *Proceedings of the third ACM conference on Data and application security and privacy*, CODASPY ’13, pages 141–152, New York, NY, USA, 2013. ACM.

[107] Wei Xu, Fangfang Zhang, and Sencun Zhu. JStill: Mostly Static Detection of Obfuscated Malicious JavaScript Code. In *Proceedings of the third ACM conference on Data and application security and privacy*, CODASPY ’13, pages 117–128, New York, NY, USA, 2013. ACM.

Appendix A

Supervised Learning Algorithms

A.1 Decision Tree

A decision tree is a tree with internal nodes corresponding to feature names, branches corresponding to feature values, and leaf nodes corresponding to class labels. The learning, i.e., building the decision tree, is done by selecting the attribute that best splits the training examples into their proper classes, i.e., benign and malicious. At each stage of building the decision tree, the algorithm chooses the attribute that could split the training examples into their classes in the best possible manner.

Most implementations of decision tree use the *information gain* ratio [56], which is a measure based on *entropy.* Intuitively, entropy is the measure of disorder in a set —meaning a low value of entropy in a set entails the homogeneity of the set and a zero entropy value means the set is composed of entirely one type of items. Once the splitting attribute is identified, the splitting criteria is used to push the rest of the training examples down the tree. Examples that satisfy the splitting criteria are pushed down the "Yes" branch, while examples that do not pass the splitting criteria are pushed down the "No" branch of the tree. This process repeats recursively until each node contains examples of the same class, at which point it stores the class label.

During classification, a decision tree classifier predicts the class of an unknown example by checking the attribute value against the criteria at each node, beginning at the root node. If the attribute matches the criteria, the classifier follows the “Yes” branch; otherwise, it follows the “No” branch. This process is repeated until an endpoint (a leaf node) is reached, which is the predicted class [76].

A.2 Random Forest

Random Forest builds many decision trees during training. To classify an unknown sample, the input feature vector is queried agains each tree in the forest. Each tree predicts the class of the unknown sample. The overall output of detection is the class label with the highest number of votes over all the trees in the forest. Each tree in the forest is built as follows [9]:

1. If the number of cases in the training set is N, sample N cases at random, but with replacement, from the original data. This sample will be the training set for growing the tree.

2. If there are M input variables, a number $m << M$ is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.

3. Each tree is grown to the largest extent possible,without pruning.

A.3 Naive Bayes

A Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes’ theorem with strong (naive) independence assumptions. Quite widely used in text classification and spam filters, the fundamental assumption in this model is that, for a given label (e.g., 1 = malicious and 0 = benign), the individual features of instances (e.g., web pages) are distributed independently of the values of other features [7, 76].

Supposing that $P(\mathrm{x}|y)$ denotes the conditional probability of the feature vector x given its label y, the model assumes $P(\mathrm{x}|y) = \prod_{j=1}^{d} P(x_j|y)$, where x_j is the j^{th} feature and d is the number of features. Then, from Bayes rule, assuming that malicious and benign web pages occur with equal probability, a posterior probability that the feature vector x belongs to the class of malicious web pages is computed as:

$$P(y = 1|x) = \frac{P(x|y = 1)}{P(x|y = 1) + P(x|y = 0)} \tag{A.1}$$

Finally, the right hand side of Equation A.1 can be thresholded to predict a binary label (0 or 1) for the feature vector x.

A.4 Logistic Regression

Logistic Regression is a parametric model for binary classification. In this model, examples are classified based on their distance from a hyperplane decision boundary [7]. The decision rule is expressed in terms of the sigmoid function $\sigma(z) = [1 + e^{-z}]^{-1}$, which converts these distances into probabilities that feature vectors have positive or negative labels. The conditional probability that feature vector x has a positive label y = 1 is computed using the following equation:

$$P(y = 1|x) = \sigma(wx + b) \tag{A.2}$$

where the weight vector $w \in R^d$ and scalar bias b are parameters to be estimated from training data. In practice, the right hand side of Equation

A.2 is thresholded to obtain a binary prediction for the label of the feature vector x.

A widely used training scheme for logistic regression models is using a regularized form of the maximum likelihood estimation. More precisely, one can choose the weight vector w and bias b to maximize the following objective function:

$$L(w, b) = \sum_{i=1}^{n} \log P(y_i|x_i) - \gamma \sum_{\alpha=1}^{d} |w_\alpha| \tag{A.3}$$

In Equation A.3, the purpose of the first term is to compute the conditional log-likelihood that the model correctly labels all the examples in the training set. The second term in Equation A.3 is used to penalize large values (in magnitude) of the elements in the weight vector w. This penalty scheme is called l_1-norm regularization.

The main advantage of this regularization that it serves as a measure against over-fitting. In addition, it also encourages sparse solutions in which many elements of the weight vector are precisely zero. The relative weight of the second term in Equation A.3 is determined by the regularization parameter. The value of γ is usually determined by applying cross validation.

A.5 Support Vector Machine

A Support Vector Machine (SVM) model is a representation of the examples as points in space, mapped so that positive and negative examples are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a class based on which side of the gap they fall on. SVMs are widely acknowledged as effective models for binary classification of high dimensional data due to the kernel trick, which applies different functions, called kernel functions, to transform a linearly inseparable data into a higher dimensional space in pursuit of one or more dividing lines between the data.

The principal objective in training SVMs is to maximize the margin of correct classification [96]. The decision rule in SVMs is expressed in terms of a kernel function $K(x, x')$ that computes the similarity between two feature vectors and non-negative coefficients $\{\alpha_i\}_{i=1}^{n}$ (n is the size of the training set) that indicate which training examples lie close to the decision boundary. SVM finds the hyperplane with the largest distance to the nearest training data points of positive (e.g., malicious web pages) and negative (e.g., benign web pages) examples, called *functional margin*. Functional margin optimization is done by maximizing the following equation:

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{A.4}$$

subject to:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, 0 \leq \alpha_i \leq C, i = 1, 2, ..., n \tag{A.5}$$

where α_i and α_j are coefficients assigned to training samples x_i and x_j. $K(x_i, x_j)$ is a kernel function used to measure similarity between the two samples. After specifying the kernel function, SVM computes the coefficients which maximize the margin of correct classification on the training set. C is a regularization parameter used for controlling the trade-off between training error and margin size.

Appendix B

Publications

1. Birhanu Eshete, Venkat Venkatakrishnan. **WebWinnow: Leveraging Exploit Kit Workflows to Detect Malicious URLs**. *ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2014 (to appear).

2. Birhanu Eshete, Adolfo Villafiorita, Komminist Weldemariam, Mohammad Zulkernine. **EINSPECT: Evolution-Guided Analysis and Detection of Malicious Web Pages**. *In Proceedings of the International Conference on Computer Software and Applications (COMPSAC)*, pages 375-380, IEEE, 2013.

3. Birhanu Eshete, Adolfo Villafiorita, Komminist Weldemariam, Mohammad Zulkernine. **Confeagle: Automated Analysis of Security Configuration Vulnerabilities in Web Applications**. *In Proceedings of the International Conference on Security and Reliability (SERE)*, pages 188-197, IEEE, 2013.

4. Birhanu Eshete, Adolfo Villafiorita. **Effective Analysis, Characterization, and Detection of Malicious Web Pages**. *In Proceedings of the International Conference on World Wide Web (WWW) Campanion*, pages 355-360, ACM, 2013.

5. Birhanu Eshete, Adolfo Villafiorita, Komminist Weldemariam. **BINSPECT: Holistic Analysis and Detection of Malicious Web Pages**. *In Proceedings of the International Conference on Security and Privacy in Communication Networks (SECURECOMM)*, pages 149-166, Springer-Verlag, 2012.

6. Birhanu Eshete, Adolfo Villafiorita, Komminist Weldemariam. **Malicious Website Detection : Effectiveness and Efficiency Issues**, *In Proceedings of the System Security Workshop (SysSec)*, pages 123-126, IEEE, 2011.

7. Birhanu Eshete, Komminist Weldemariam, Adolfo Villafiorita. **Early Detection of Security Misconfiguration Vulnerabilities in Web Applications**. *In Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, pages 169-174, IEEE, 2011.

8. Valentino Sartori, Birhanu Eshete, Adolfo Villafiorita. **Measuring the Impact of Different Metrics on Software Quality: a Case Study in the Open Source Domain**. *In Proceedings of the International Conference on Digital Society (ICDS)*, 2011.

9. Birhanu Eshete, Dawit Bekele, Komminist Weldemariam, Adolfo Villafiorita. **Context Information Refinement for Pervasive Medical Systems**. *In Proceedings of the International Conference on Digital Society (ICDS)*, pages 210-215, IEEE, 2010.

10. Biniyam Asfaw, Dawit Bekele, Birhanu Eshete, Komminist Weldemariam, Adolfo Villafiorita. **Host-based Anomaly Detection for Pervasive Medical Systems**. *In Proceedings of the International Conference on Risks and Security of Internet and Systems (CRiSIS)*, pages 1-8, IEEE, 2010.