BINSPECT: Holistic Analysis and Detection of Malicious Web Pages

Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam

Fondazione Bruno Kessler (FBK-IRST), Trento, Italy
{eshete,adolfo,sisai}@fbk.eu
*

Abstract. Malicious web pages are among the major security threats on the Web. Most of the existing techniques for detecting malicious web pages focus on specific attacks. Unfortunately, attacks are getting more complex whereby attackers use blended techniques to evade existing countermeasures. In this paper, we present a holistic and at the same time lightweight approach, called BINSPECT, that leverages a combination of static analysis and minimalistic emulation to apply supervised learning techniques in detecting malicious web pages pertinent to drive-by-download, phishing, injection, and malware distribution by introducing new features that can effectively discriminate malicious and benign web pages. Large scale experimental evaluation of BINSPECT achieved above 97% accuracy with low false signals. Moreover, the performance overhead of BINSPECT is in the range 3-5 seconds to analyze a single web page, suggesting the effectiveness of our approach for real-life deployment.

Keywords: malicious web page, static analysis, lightweight emulation, machine learning

1 Introduction

The Web has become an indispensable global platform that glues together daily communication, sharing, trading, collaboration, and service delivery. Web users often store and manage critical information that attracts cybercriminals who misuse the Web and the Internet to exploit vulnerabilities for illegitimate benefits.

Malicious web pages, that exploit vulnerabilities and launch attacks for just one time visit, take an alarmingly significant share of web-based attacks in recent years [1–4]. When an unsuspecting victim visits a web page, an attacker might have compromised the page under visit (or crafted it purposefully) and the outcome of the visit could be stealing of critical credentials (e.g., credit card details) to impersonate the victim, installation of a malware binary on the victim's machine for future attacks, or even a complete takeover of the victim's system to

* We would like to thank the Computer Security Group of UC Santa Barbara for allowing us to submit and evaluate our test set with their tool, Wepawet.

remotely command and control it as a member of botnet [5–7]. In recent years, not only is the prevalence of malicious web pages on the rise but the way in which attackers trick victims to malicious web pages is also getting sophisticated [2]. It has become a common encounter to get contaminated search results from search engines on trendy terms, malicious links shared on social media, and legitimate web pages injected with malicious scripts [3].

The thus-far proposed defenses against malicious web pages fall into two major blocks, i.e., static analysis and dynamic analysis techniques. However, the use of blacklists is still a common way to facilitate and enrich these techniques by making use of heuristics and learning techniques.

Static analysis techniques, [5, 8–15], inspect web page artifacts without rendering the page in a browser. The inspection usually involves quickly extracting discriminative features from the URL string, host identity, HTML, and JavaScript code.

The major assumption behind static analysis is that the statistical distribution of URL tokens and host details of malicious URLs (e.g., spam URLs, phishing pages) tend to differ from that of benign. The feature values of URL lexical structure and host identity are encoded to train machine learning techniques to build classifiers based on which unknown web pages are classified.

In static analysis, it is difficult to detect attacks that require rendering of a page to take action. More precisely, when considering page source there is a high risk of obfuscated content (e.g., JavaScript) and overlooking of malicious JavaScript that exploits vulnerabilities of browser plugins. In addition, host identity details of fresh (benign) URLs, registered by registrars with low reputation, are likely to be misclassified as malicious due to their low reputation scores. In effect, there is a high risk of false positives. On the other hand, false negatives may arise as old and well-reputed registrars may host malicious web pages which have escaped the static analysis effort. Another cause of false negatives are web pages that use free hosting services or already compromised sites with benign-looking URL and host information. For static anlaysis relying on lexical URL features, an attentive attacker may evade these features to mislead detection techniques by carefully crafting malicious URLs which look statistically indistinguishable from the benign ones.

Dynamic analysis approaches, [11, 16–24], have also been shown to be effective in the analysis and detection of malicious web pages. Such techniques could be deployed at a proxy-level (e.g., [19]) to intercept requests (responses), 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) could be used (as in [17]) to log critical actions (e.g., invoking a plugin) and match logs with known patterns of malicious activities or apply learning-based techniques to model and classify malicious intentions.

While effective at uncovering daunting malicious web pages, dynamic analysis approaches are resource intensive as they need to load and execute the page

under analysis and modern web pages are usually stuffed with rich client-side code and content which take longer 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.

Blacklisting-based techniques maintain a list of known malicious URLs, IP addresses, and domain names collected by manual reporting, honeyclients, and custom analysis techniques. For example, `Google Safe Browsing` service [25] maintains a blacklist against which it checks URL requests from browsers to alert users if the requested URL happens to be in the blacklist. Another tool powered by blacklisting is `McAfee Site Advisor` [26] which is pluggable to `Mozilla Firefox` and `Internet Explorer` to rate safety of web pages and search engine results prior to rendering in the browser.

Although lightweight to deploy and easy to use, blacklisting is effective only if one can exhaustively patrol the Web to identify malicious web pages and timely update the blacklist. In practice, to do so is infeasible due to: fresh web pages are too new to be blacklisted even if they are malicious right from the outset, some web pages may escape from the blacklisting due to *"cloaking"*, and the attackers may frequently change where the malicious web pages are hosted. Consequently, the URLs and IP addresses may also change accordingly [5], [16].

Heuristic-based techniques (e.g., [15]) build signatures of known attack payloads to be used by antiviral systems or intrusion detection systems to scan a web page and flag it as malicious if its heuristic pattern matches signatures in the database. Unfortunately, such signatures are easily bypassed by attackers (mainly through obfuscation) and the heuristics fail to detect novel attacks. In addition, the rate at which the signature database of heuristic-based systems is updated is way slower than the pace at which attackers overwhelm victims with novel attacks, resulting in zero-day exploits.

In addition to the above-mentioned limitations, most approaches focus on one prominent attack while attack techniques are getting more and more complex whereby attackers use blended attack techniques by combining existing attack techniques to evade existing countermeasures. More importantly, applying static or dynamic analysis approaches in a complementary fashion is limited to capturing partial snapshot of a malicious web page.

To this end, the ideal solution is to leverage static and dynamic analysis to capture a comprehensive snapshot of a malicious web page and ensure that the overhead cost of analyzing a web page is optimal. This can be done by holistically characterizing and then analyzing, and detecting malicious web pages to capture a comprehensive snapshot of malicious web pages while ensuring that the analysis and detection remains lightweight in terms of its responsiveness and resource consumption.

In this paper, we present the design, implementation, and experimental evaluation of a holistic and at the same time lightweight system, called `BINSPECT`, that leverages a combination of static analysis and minimalistic emulation to apply supervised learning techniques in detecting malicious web pages pertinent to drive-by-download, phishing, injection, and malware distribution. `BINSPECT`

achieved detection accuracy above 97% with low false signals and an average performance overhead of at most 5 seconds.

The contributions of this paper are the following:

- we developed a holistic and at the same time lightweight approach to analyze and detect malicious web pages by leveraging static analysis and lightweight emulation of web page rendering with low performance overhead.
- we introduced new features and enhanced existing ones so as 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 paper is structured as follows. In Section 2, we present a real motivational example pertinent to malicious web pages. Section 3 covers details of holistic characterization of malicious web pages focusing on features we introduce as new and enhance from existing ones. In Section 4, a high-level description of our approach is presented. Details of the experimental setup and evaluation of our approach are discussed in Section 5. Section 6 positions our approach relative to prior work. Finally, Section 7 concludes the paper.

2 Motivational Example on Malicious Web Pages

In this section, we provide illustrations of real threats posed by malicious web pages.

A *malicious web page* is a web page that exploits one or more vulnerabilities of the browsing environment to launch one or more attacks when visited by an unsuspecting victim. Usually, malicious web pages perform attacks in four ways: *obfuscation* (e.g., obfuscated malicious JavaScript), *setting up malicious web pages* (e.g., using HTTP or JavaScript redirection), *victim luring* (e.g., social engineering tricks), and *victim takeover* (e.g., installing malware). To give context to what malicious web pages perform in practice, in what follows we describe a real malicious website attack that successfully compromised a high-profile website few months ago [27].

On Sept. 26, 2011, when users visited `http://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 code is shown in Listing 1.1). The malicious code embeds an iframe to `http://falosfax.in/info/in.cgi?5&ab_iframe=1&ab_badtraffic=1&antibot_hash=1255098964&ur=1&HTTP_REFERER=http://mysql.com/` malicious domain and then throws an `HTTP 302 redirection` to load the `http://truruhfhqnviaosdpruejeslsuy.cx.cc/main.php` exploit domain. This exploit domain hosts the `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. In this attack, the actual payload is an exploitation of a vulnerability of the Java runtime on the browser (`Internet Explorer 6`) to download and execute malware that steals and sends back to the attacker FTP client passwords from the user's machine. Such an attack is called *drive-by-download* [28].

```
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'
    );
 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);
}
```

Listing 1.1. De-obfuscated JavaScript exploit code of the attack [27]

Discussion. The attack described before sounds specific to a compromised legitimate website, i.e., `http://mysql.com`. However, there are a couple of interesting aspects in the attack chain. First, the attacker has to target a high-profile website with solid user-base and daily traffic. Secondly, she exploited a vulnerable spot (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 if the target in this attack is the Java plugin, in principle this could have been any one of the vulnerable browser components or its extensions (e.g., PDF Renderer, Flash Player) since the malware usually runs within the privilege of the current user. In theory, the downloaded binary could be of a scope ranging from key-logging to steal and submit passwords and credit card details to a malware that compromises the victim's machine to make it member of a botnet for later use in criminal activities (e.g., spam campaigns). Similarly, the vulnerability of the browsing environment could be of various risks depending on the client operating system, browser type and version, and browser extensions and configuration. An essential part of the attack chain is fingerprint-

ing of the environment which provides clues to vulnerable spots based on which actual exploits are invoked.

3 Holistic Characterization of Malicious Web Pages

Given an unknown web page, BINSPECT analyses and classifies the web page as malicious or benign. To do so, 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. BINSPECT considers malicious web pages that launch drive-by-download attack, phishing, injection, and malware delivery attack.

The features on which BINSPECT bases its statistical characterization of web pages leverages three classes of features, i.e., URL features, Page-Source features (HTML and JavaScript), and Social-Reputation features. The underlying assumption of using these features, both in prior work and in our work, is based on the discriminative power of the statistical distribution of benign and malicious web pages. In what follows, we describe the 39 features we extract and inspect (in particular the new features we introduced) which are the basis for the construction of the models we use to classify malicious web pages in BINSPECT.

3.1 URL Features

In BINSPECT, we rely on 11 URL features among which 8 features are reused from prior work ([5], [10]) and we introduce 3 new 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 availability of the **client** and/or **server** words in the URL. After an analysis of the F-Score measure of candidate URL features, we found the 3 new 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 features are: **length of the path** in the URL, **length of the query** in the URL, and **length of the file-path** in the URL. In Section 5, we show the experimental verification as to the effectiveness of these new URL features in practice.

3.2 Page-Source Features

While most of prior work based on static analysis extracts HTML and JavaScript features statically, we use an emulated browser to parse and render the HTML and at the same time execute JavaScript that needs to be executed on page-load so as to capture what is manifested by JavaScript code. In this sense, the granularity of most HTML features used in our work is high because the JavaScript that is executed on page-load feeds more features to our feature extraction engine, particularly enriching the HTML features. Another reason to use an emulated browser is to capture the side-effects of obfuscated JavaScript

code that is usually 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 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 (along with count of specifically suspicious functions : `subString()`, `fromCharCode()`, `eval()`, `setTimeout()`, `document.write()`, `createElement()`, `unescape()`, `escape()`, `link()`, `exec()`, and `search()`). It is worth-noting that although the Page-Source features we use are mostly from prior work, we also introduce new (e.g., `exec()` function) and enhance existing features in order to ensure a more fine-grained characterization of web pages(e.g., apart from extracting the total number of links on the page, we split links to: **number of same-origin links**, **number of different-origin links**, and **number of external JavaScript files**).

3.3 Social-Reputation Features

The ubiquitous effect of social network platforms, such as Facebook, Twitter, and Google Plus, is continuously changing the landscape of online social interaction and reputation building about what is shared online. Recently, search engines are partly relying on social network reputation of URLs to enrich their ranking algorithms because of the involvement of people in rating the URLs [29]. To evaluate if these social-reputation indicators can be used in characterizing malicious and benign URLs, we examined the statistical significance of URL-Sharing on Facebook and Twitter as these platforms keep track of the number of times a URL is shared publicly.

A preliminary experimental observation of these features suggests that for benign web pages, the share-count is usually higher as users are confident enough to share a URL that they know as harmless or they re-share after seeing that their friends have done so. On the contrary, the share-count for malicious URLs suggests that, either the URLs are not circulated across the social network or users refrain from sharing a URL they know less about.

Figure 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. The three new features we introduce are the **Facebook Share Count**, **Twitter Share Count**, and **Google Plus Share Count** which tell the number of times a URL is publicly shared on Facebook, Twitter, and Google Plus. An attentive reader may argue that these features may contribute to false negatives in the case where a malicious user publicly shares a malicious URL on a social network. However, as time passes by, the tendency that a malicious URL is circulated across the social network will reduce or the share-count of the URL does not increase because of built-in URL analysis and detection techniques in the social network platform.

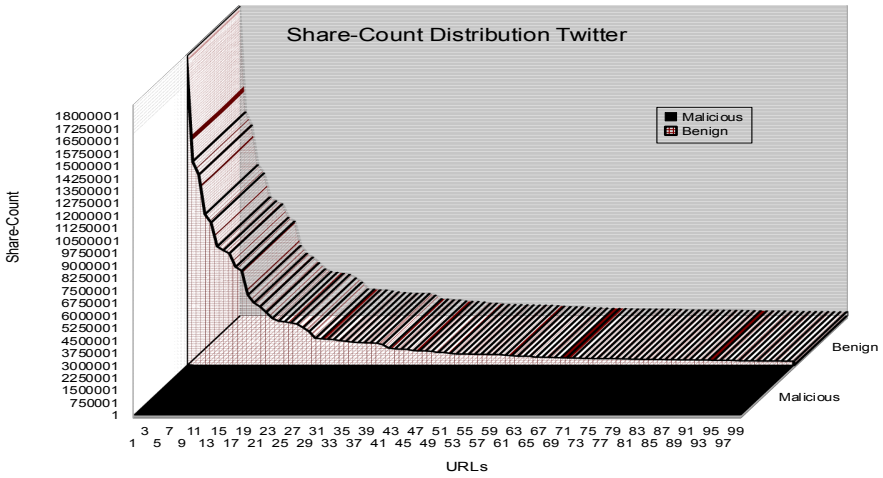

Fig. 1. Distribution of the top 100 Twitter share-counts for benign and malicious URLs on the training set.

4 BINSPECT System Overview

In this section, we give an overview of the implementation details of BINSPECT.

In a nutshell, BINSPECT has three major components: feature extraction and labeling, multi-model training, and classification. Figure 2 illustrates these three major architectural blocks of BINSPECT. In the following, we provide a high-level explanation of these major components of BINSPECT.

Feature Extraction and Labeling: As shown in Figure 2, we use a dataset of benign and malicious samples (described in Section 5) to label the samples and extract the necessary features which characterize malicious and benign web pages. The URL feature extraction is implemented based on the URL class of Java and the features are collected 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) using a feature extraction engine we implemented in Java. We customized the HTMLUnit [30] 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, a fresh instance of the emulated browser is created to ensure a unique session for each URL. We used the Facebook Graph API [31], the Twitter URLs API [32], and a custom[1] script on Google Plus to automatically extract the Social-Reputation features. Features extracted from the each web page is represented as a feature 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 $(k = 1, .., n)$,

[1] There was not a standard API for Google Plus at the time of this experiment.

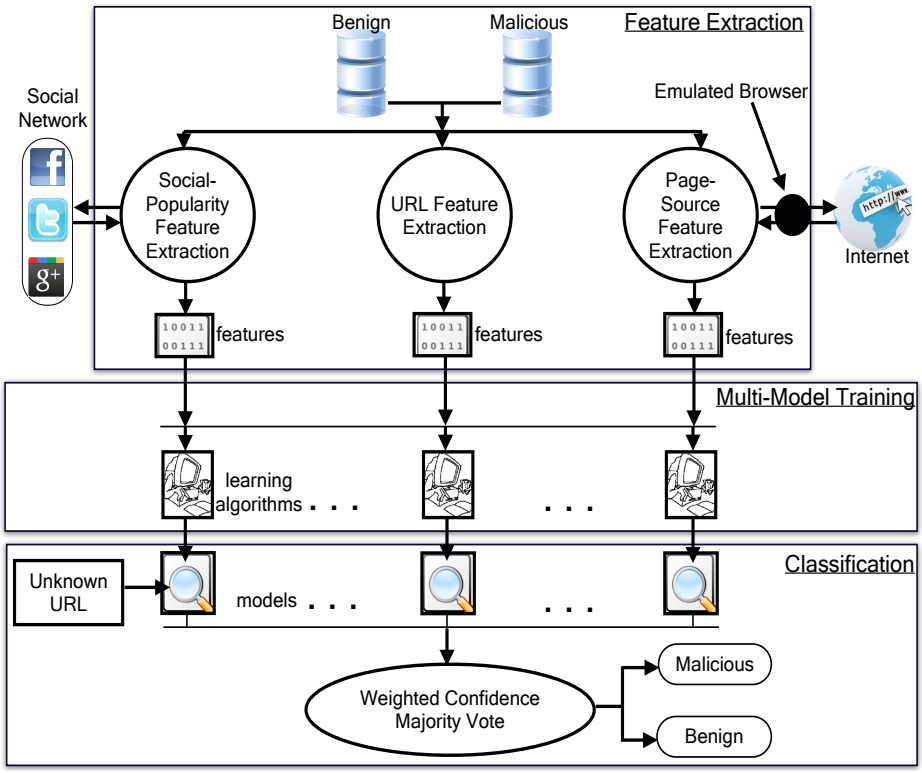

Fig. 2. BINSPECT System Overview

n is the number of features, and $class^{(i)}$ is the class label of URL$^{(i)}$ which is either `benign` or `malicious`.

Multi-Model Training: Using the extracted features, we train 7 supervised learning algorithms namely Decision Trees (J48, Random Tree, and Random Forest), Bayesian Classifiers (Naive Bayes and Bayes Net), Support Vector Machines, and Logistic Regression Classifier. At the end of the training, one model for each classifier is maintained as shown in the second block of Figure 2.

Weighted-Confidence Majority Vote Classification: For classification of an unknown web page using the learned models, we use the Weighted-Confidence Majority Vote algorithm that we customized (see Algorithm 1) to decide the class of the web page. To flag a page as either malicious or benign, instead of just taking the count of votes of the individual models, the vote count of the class label is multiplied with the sum of confidences with which the votes are made by the individual models (lines 17, 20, and 23 in Algorithm 1). The use of weighted-confidence majority vote is twofold. First, it minimizes the bias of relying on a single model as in most prior work. Secondly, it allows comparisons of different models and makes the overall result more difficult to manipulation by attackers.

Algorithm 1 Confidence-Weighted Majority Vote Classification

$Conf_{benign} \leftarrow 0$
$Conf_{malicious} \leftarrow 0$
$Vote_{benign} \leftarrow 0$
$Vote_{malicious} \leftarrow 0$
for $i = 1 \rightarrow numModels$ **do**
  $features \leftarrow extractFeatures(URL)$
  $Vote_i, Conf_i \leftarrow getPredictionWithConfidence(features, Model_i)$
  if $Vote_i == benign$ **then**
    $Vote_{benign} \leftarrow Vote_{benign} + 1$
    $Conf_{benign} \leftarrow Conf_{benign} + Conf_i$
  end if
  if $Vote_i == malicious$ **then**
    $Vote_{malicious} \leftarrow Vote_{malicious} + 1$
    $Conf_{malicious} \leftarrow Conf_{malicious} + Conf_i$
  end if
end for
if $(Vote_{malicious} \times Conf_{malicious}) > (Vote_{benign} \times Conf_{benign})$ **then**
  $Prediction \leftarrow malicious$
end if
if $(Vote_{malicious} \times Conf_{malicious}) < (Vote_{benign} \times Conf_{benign})$ **then**
  $Prediction \leftarrow benign$
end if
if $(Vote_{malicious} \times Conf_{malicious}) == (Vote_{benign} \times Conf_{benign})$ **then**
  $Prediction \leftarrow suspicious$
end if

5 Experimental Setup and Evaluation

In this section, we present the experimental setup and evaluation of BINSPECT.

First, we describe the data collection, dataset construction, and the experimental procedure. Then, we evaluate BINSPECT from from the standpoint of its accuracy, significance of the features we introduced in Section 3, its performance overhead, and its immunity to possible evasion.

5.1 Dataset and Experimental Setup

Data 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. The breakdown of the dataset is shown in Table 1. For the malicious dataset, we collected 71,919 URLs from the malware and phishing blacklist of **Google** [25], the **Phishtank** database of collaboratively verified phishing pages [33], and the malware and injection attack URL list of **MalwareURL** [34]. The dataset of 414,000 benign URLs is also drawn from three popular sources. These are the **Alexa Top sites** [35], the **Yahoo** random URL generation service [36], and the **DMOZ** directory [37].

Table 1. Dataset for training and testing

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

Experimental Protocol: Using the training set, we extracted 39 features 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 (in this case Internet Explorer 6 and Mozilla Firefox 3) and we used only the core components of the browser, i.e., Necko HTML Engine, Rhino JavaScript Engine, and the default CSS Parser in order to make the analysis lightweight. We used the Weka [38] machine learning toolbox to train the 7 standard classifiers with 10-fold cross validation.

5.2 Evaluation Results and Insights

Classification Accuracy
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, 4, 5, and 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, training on all 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.

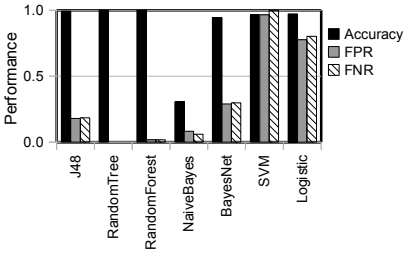

Fig. 3. With all features.

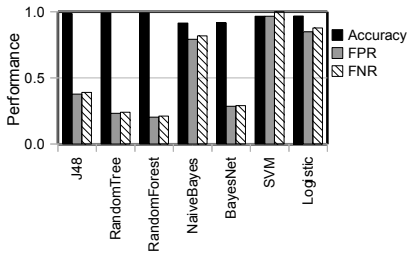

Fig. 4. With URL features.

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 4 (60%) vs. Figure 3 (30%)) on URL features probably because the URL features

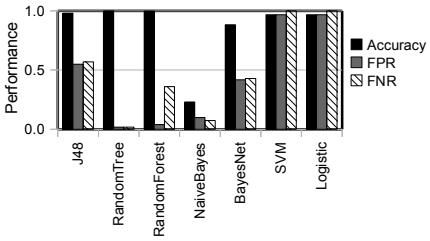

Fig. 5. With Page-Source features.

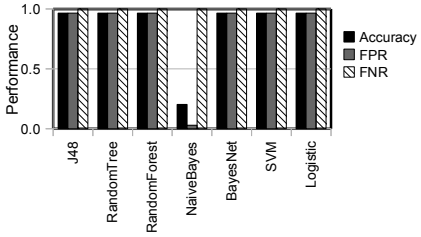

Fig. 6. With Social-Reputation features.

have a statistical distribution that fits into the high degree of independence assumed in the algorithm. Another interesting observation from Figure 6 is the high FNR of all the classifiers on social-reputation features which is attributed to the fact that 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 the union of 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, 5, and 6).

For testing, we used all the classifiers except Naive Bayes due to its poor performance on the training set. Table 2 shows the overall classification accuracy of BINSPECT over a testing set shown in Table 1. We measured the classification accuracy as the ratio of correct classifications to the total size of the testing set. We submitted the same testing set to Wepawet [39] to compare BINSPECT with a publicly deployed analysis and detection service. As can be seen from Table 2, 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 lower accuracy 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 is an indication 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 that launch drive-by-download, phishing, injection, and malware delivery attacks.

Significance of New Features

To verify whether the features we introduced are of predictive importance in enhancing the effectiveness of analysis and detection of 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 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

Table 2. Performance of BINSPECT in comparison to a public malicious web page analysis and detection service on the testing set.

Measure	BINSPECT	Wepawet [39]
Classification Accuracy	97.81%	61.62%
False Positive Rate	0.189	0.983
False Negative Rate	0.011	0.073

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 as shown in Table 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 SVM), 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. 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

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 results show that the average time it takes to train a classifier is only 1.51 seconds on the training set. Moreover, we also estimated the average time it takes to analyze and classify one web page on the testing set. BINSPECT, on average, took between 3 to 5 seconds (due to varying system

Table 4. Detailed performance analysis of BINSPECT 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	0.1
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(↓)

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 were not able to make comparison between the performance overhead of BINSPECT and Wepawet due to the long delay it took to get back the results from Wepawet server which uses queueing mechanism to process batch requests for analysis.

Immunity 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 has to do to try evading 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 likelihood of evasion is for the attacker to use highly obfuscated client-side code (e.g., JavaScript). In such a case, BINSPECT is likely to be partly tricked because of the low consideration of obfuscated content in our approach. With regards to the Social-Reputation features, the only risk is the inevitable circumstance 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 users stop sharing the link or the built-in URL scanning facility of the social network platform discovers 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.

6 Related Work

In this section, we comparatively describe related work from the standpoint of attacks considered, features used, analysis technique(s) applied, and effectiveness of detecting malicious web pages.

Canali et al. [5] proposed Prophiler, a purely static pre-filtering technique that deems web pages that launch drive-by-download attacks as likely malicious or likely benign. Prophiler achieved a very low false positive rate over a large testing set of URLS utilizing a total of 78 features composed of URL, host details, HTML, and JavaScript features. In BINSPECT , we apply static analysis and a lightweight dynamic analysis to deem a web page as benign or malicious. Unlike Prophiler where only the best classifiers during training are used for testing, BINSPECT uses confidence-weighted majority vote for classification. Except the new features we introduced here, all the other features used in BINSPECT are also used by Prophiler, with additional features. In BINSPECT, the number of features are half (39 versus 79) the number of features used in Prophiler.

Cova et al. [39] built Wepawet, an emulation-based dynamic analysis and detection framework for malicious content (mainly malicious JavaScript and

malware). It is based on anomaly detection and the analysis and detection 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. BINSPECT, however, is a learning-based approach using mostly static features with a minimalistic emulation support.

Ma et al. [10] proposed a purely static analysis based on URL lexical features and host details and applied supervised learning and online learning techniques to achieve about 99% accuracy with a very low false positive rate. However, BINSPECT differs from their approach by the fact that they use URL and host-based information only and the focus is to quickly classify URLs without further analysis of the page content and the execution dynamics in a browser. In our case, we reuse most of the URL features used by them in a statistical manner than lexical (presence/absence). More importantly, we use an emulated browser to visit and render the page and execute any client-side code up on page load.

Dewald et al. [17] proposed ADSandbox, a client-side JavaScript sandboxing and signature-based, analysis technique that executes JavaScript embedded in a page within an isolated environment and log every critical action to detect malicious web pages. ADSandbox achieved a false positive close to zero but at a relatively high performance overhead. BINSPECT, however, is a learning-based approach that is not limited to web pages that host malicious JavaScript but includes also phishing pages, malware delivery pages, and pages that initiate injection attacks.

Compared to most prior work, BINSPECT characterizes malicious web pages spanning four classes of attacks (drive-by-download, phishing, malware-delivery, and injection) to build a lightweight detection system that relies on only 39 features of URL string, HTML, JavaScript, and Social-Reputation of URLs.

7 Conclusion

Existing techniques for detecting malicious web pages are quite effective at combating specific attack types. However, they are limited to a partial snapshot of a malicious payload which limits their ability to cope up with the ever-changing and complex threats posed by malicious web pages. In this paper, we presented BINSPECT, a holistic analysis and detection approach for defending users against malicious web pages by leveraging static analysis and lightweight emulation based on machine learning techniques. We have shown through a large scale evaluation that BINSPECT is quite effective at precisely detecting malicious web pages with very low false signals. Moreover, the new features we introduced in this work are relevant enough in improving the overall performance of the analysis and detection of malicious web pages. In terms of performance overhead, our experiments suggest that BINSPECT incurs acceptable overhead cost to analyze web pages in a realistic scenario due to relatively few and effective features reused from prior work and the newly introduced features.

One limitation of BINSPECT is lack of analysis of obfuscated JavaScript and emulation of the browser with plugins. In the future, we would like to incremen-

tally improve BINSPECT by introducing these missing analysis steps. Another room to improve BINSPECT is to further investigate additional features from social networks to characterize malicious web pages from the social perspective. We would also like to make BINSPECT an evolution-aware analysis and detection framework that takes into account the evolution of features and tunes its models accordingly by applying evolutionary techniques.

References

1. 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)
2. 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)
3. WebSense: Websense 2010 threat report. http://www.websense.com/content/threat-report-2010-highlights.aspx/ (July 2011)
4. Symantec: Internet security threat report 2011 trends. http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf (April 2012)
5. Prophiler: a fast filter for the large-scale detection of malicious web pages. In: Proceedings of WWW. (2011)
6. Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., Vigna, G.: Your botnet is my botnet: analysis of a botnet takeover. In: Proceedings of the 16th ACM conference on Computer and communications security. (2009) 635–647
7. Eshete, B., Villafiorita, A., Weldemariam, K.: Malicious website detection: Effectiveness and efficiency issues. In: Proceedings of SysSec Workshop (SysSec), 2011. (2011) 123–126
8. Justin, M.: Learning to Detect Malicious URLs. PhD thesis, University of California, San Diego (2010)
9. Justin, M., K., S.L., Stefan, S., M., V.G.: Identifying suspicious urls: an application of large-scale online learning. In: Proceedings of ICML. (2009) 681–688
10. Justin, M., K., S.L., Stefan, S., M., V.G.: Beyond blacklists: learning to detect malicious web sites from suspicious urls. In: Proceedings of KDDM. (2009) 1245–1254
11. Thomas, K., Grier, C., Ma, J., Paxson, V., Song, D.: Design and Evaluation of a Real-Time URL Spam Filtering Service. In: Proceedings of the IEEE Symposium on Security and Privacy. (2011)
12. Choi, H., Zhu, B.B., Lee, H.: Detecting malicious web links and identifying their attack types. In: Proceedings of the 2nd USENIX conference on Web application development. (2011) 11–11
13. Seifert, C., Welch, I., Komisarczuk, P., Aval, C., Endicott-Popovsky, B.: Identification of malicious web pages through analysis of underlying dns and web server relationships. In: 33rd IEEE Conference on Local Computer Networks. (2008)
14. Yung-Tsung, H., Yimeng, C., Tsuhan, C., Chi-Sung, L., Chia-Mei, C.: Malicious web content detection by machine learning. Expert Syst. Appl. **37**(1) (2010) 55–60

15. Seifert, C., Welch, I., Komisarczuk, P.: Identification of malicious web pages with static heuristics. In: Proceedings of the Australasian Telecommunication Networks and Applications Conference. (2008)
16. Qassrawi, M., Zhang, H.: Detecting malicious web servers with honeyclients. Journal of Networks **6**(1) (2011)
17. Dewald, A., Holz, T., Freiling, F.C.: Adsandbox: sandboxing javascript to fight malicious websites. In: ACM Symposium on Applied Computing. (2010) 1859–1864
18. Marco, C., Christopher, K., Giovanni, V.: Detection and analysis of drive-by-download attacks and malicious javascript code. In: Proceedings of WWW. (2010) 281–290
19. Alexander, M., Tanya, B., Damien, D., Gribble, S.D., Levy, H.M.: Spyproxy: execution-based detection of malicious web content. In: Proceedings of 16th USENIX Security Symposium. (2007) 3:1–3:16
20. Ford, S., Cova, M., Kruegel, C., Vigna, G.: Analyzing and detecting malicious flash advertisements. In: Proceedings of ACSAC. (2009)
21. Ikinci, A., Holz, T., Freiling, F.: Monkey-spider: Detecting malicious websites with low-interaction honeyclients. In: Proceedings of Sicherheit, Schutz und Zuverlssigkeit. (2008) 407–421
22. Byung-Ik, K., Chae-Tae, I., Hyun-Chul, J.: Suspicious malicious web site detection with strength analysis of a javascript obfuscation. In: International Journal of Advanced Science and Technology. (2011) 19–32
23. Rieck, K., Krueger, T., Dewald, A.: Cujo: efficient detection and prevention of drive-by-download attacks. In: Proceedings ACSAC. (2010) 31–39
24. Kolbitsch, C., Livshits, B., Zorn, B., Seifer, C.: Rozzle: De-cloaking internet malware. Technical report, Microsoft (2011)
25. Google: Google safe browsing api. `http://code.google.com/apis/safebrowsing/` (August 2011)
26. McAfee: Mcafee site advisor. `http://www.siteadvisor.com` (July 2011)
27. Armorize.: mysql.com hacked:infecting visitors with malware. `http://blog.armorize.com/2011/09/mysqlcom-hacked-infecting-visitors-with.html` (September 2011)
28. Egele, M., Kirda, E., Kruegel, C.: Mitigating drive-by download attacks: Challenges and open problems (2009)
29. SEO, D.: Facebook and twitter's influence on google's search rankings
30. Software, G.: Htmlunit. `http://htmlunit.sourceforge.net/` (March 2012)
31. Facebook: Facebook graph api. `https://developers.facebook.com/docs/reference/api/` (March 2012)
32. Twitter: Twitter url api. `http://urls.api.twitter.com/1/urls/` (March 2012)
33. PhishTank: Phishtank developer information. `http://www.phishtank.com/developer_info.php` (September 2011)
34. MalwareURL: Malware urls. `http://www.malwareurl.com/` (September 2011)
35. Alexa: Alexa top 500 global websites. `http://www.alexa.com/topsites` (July 2011)
36. Inc., Y.: Yahoo random url generator. `http://random.yahoo.com/bin/yrl/` (October 2011)
37. DMOZ: Open directory project. `http://www.dmoz.org/` (September 2011)
38. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explorations **11** (2009)
39. UCSB: Wepawet. `http://wepawet.cs.ucsb.edu` (July 2011)