Detecting Malicious Landing Pages in Malware Distribution Networks

Gang Wang
Computer Science
UC Santa Barbara
Santa Barbara, CA 93106
gangw@cs.ucsb.edu

Jack W. Stokes
Microsoft Research
One Microsoft Way
Redmond, WA 98052
jstokes@microsoft.com

Cormac Herley
Microsoft Research
One Microsoft Way
Redmond, WA 98052
cormac@microsoft.com

David Felstead
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
davfeld@microsoft.com

***Abstract*—Drive-by download attacks attempt to compromise a victim's computer through browser vulnerabilities. Often they are launched from Malware Distribution Networks (MDNs) consisting of landing pages to attract traffic, intermediate redirection servers, and exploit servers which attempt the compromise.**

In this paper, we present a novel approach to discovering the landing pages that lead to drive-by downloads. Starting from partial knowledge of a given collection of MDNs we identify the malicious content on their landing pages using multiclass feature selection. We then query the webpage cache of a commercial search engine to identify landing pages containing the same or similar content. In this way we are able to identify previously-unknown landing pages belonging to already identified MDNs, which allows us to expand our understanding of the MDN.

We explore using both a rule-based and classifier approach to identifying potentially malicious landing pages. We build both systems and independently verify using a high-interaction honeypot that the newly identified landing pages indeed attempt drive-by downloads. For the rule-based system 57% of the landing pages predicted as malicious are confirmed, and this success rate remains constant in two large trials spaced five months apart. This extends the known footprint of the MDNs studied by 17%. The classifier-based system is less successful, and we explore possible reasons.

Keywords*—*Drive-by download; malware distribution network; signature;

I. Introduction

The reach and scale of the Internet has fostered a parasitic industry of those who seek illegal profit. A common strategy is to infect innocent users' machines with malicious code which can then be used to harvest passwords, send spam, retrieve contact lists, participate in a botnet, *etc*. A malware author needs three things [1]: bad code, a way to get it running, and an introduction to the user. The second and third often represent the challenge in running a cybercrime business; that is, finding users and getting the code to run on their machines is more of a challenge than writing the malware. Social engineering, the process of using false pretence to lure a user to install the software himself, has met with considerable success. Numerous studies have shown that users can be manipulated into installing malware, ignoring security warnings, and disabling protection mechanisms [2], [3]. The introduction to the user in this case is often provided in the form of a spam campaign.

A second approach attempts to exploit un-patched vulnerabilities in the applications on the user's machine. Large complex applications such as Adobe Acrobat, Microsoft Excel, *etc*, often have vulnerabilities. Opening a malicious document with a vulnerable application can be enough to give the attacker the opening to get malicious code running on the user's machine. Again, in this case spam is often the introduction vector. For example, many spam campaigns try to get a user to open an attachment with lures such as "your tax request has been denied" or "your package delivery failed" in the subject line of the email.

A drive-by download is a particular case where the vulnerable application is a browser. Some browser vulnerabilities will allow malicious code to begin running without the user's knowledge or consent. A user who visits a malicious webpage with a vulnerable browser could get infected. This opens various possibilities for attackers. An attacker could set up websites that host malicious content and then wait for vulnerable browsers to come by. The number of users infected will then be related to the amount of traffic that the site can gain. It is certainly possible to employ Search Engine Optimization (SEO) techniques to maximize traffic to a page that has nothing except malicious code to offer. However, since even legitimate websites compete vigorously for visitors, getting traffic is by no means a trivial proposition.

A more common approach is to infect an innocent website with code that directs a browser to load malware from a second site. A particularly attractive aspect of this approach is that it allows the attacker to piggyback on someone else's traffic: the introduction to the user is provided by the web-traffic that a site is already attracting. Rather than deface, or interfere with the performance of the infected site the attacker generally injects a malicious script that eventually redirects the browser to a server hosting a malicious payload. Thus visitors with browsers that possess the targeted vulnerabilities will become infected. The innocent site is called the *landing page*, and the site with the malware is called the *exploit server*. In this way the attacker gets to infect many clients without having to earn the traffic.

Often the path from the landing page to the exploit server contains many redirects. For example, if the attacker succeeds in infecting the innocent webserver at `foo.com` he can direct all traffic to load the malicious content from `evil.com`. This can be done indirectly, so that the page at `foo.com` points to `a.com`, which points to `b.com`, which points to `c.com`

978-1-4799-0181-4/13/$31.00 ©2013 IEEE

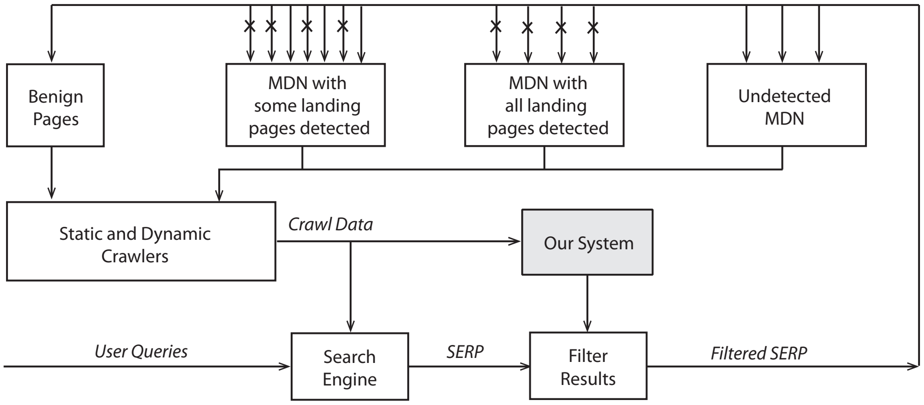

Fig. 1: High-level overview. Our system processes webpage content from the static crawler and drive-by download detection results from the dynamic crawler. It identifies malicious webpages belonging to MDNs based on similar content which can then be used to filter the search engine result pages.

and so on, until it eventually reaches `evil.com`. Often there will be many landing pages that share a small collection of exploit servers. They may also share some of the links in their redirection paths to the exploit servers. The collection of landing pages, exploit servers and intermediate redirection pages is collectively known as a Malware Distribution Network (MDN) [4].

Addressing this exploitation of users involves a multi-pronged approach. Browser vendors aggressively seek to identify and patch vulnerabilities. This is complicated by the apparent reluctance of users to promptly install security updates. For example, 22% of Internet Explorer users were still using IE6 more than four years after the launch of IE7 and 18 months after the launch of IE8, and 80% of users studied by Trustseer were using un-patched versions of Flash Player [5]. Having updates installed automatically by default has improved this situation somewhat, but machines that are compromised through vulnerabilities for which available patches have not been installed remains a serious problem. Anti-virus software, of course, is the main line of defense for most users.

Search engines actively seek to identify webpages associated with malicious content. Such webpages can be lowered, or dropped altogether, in the ranked results returned to users. However, the architecture of the MDN makes the task of identifying exploit servers very difficult. Search crawlers typically retrieve the contents of the document at a site, and do not run any scripts on the page. This is an unavoidable consequence of scale: large search engines index billions of pages per day. Rendering a page and running all scripts, as a browser would do, can take orders of magnitudes more resources than simply loading the main document. Thus the malicious actions performed by the scripts on a landing page are largely invisible to search crawlers. We outline previous attempts at MDN detection in the Related Work Section.

In this paper, we study drive-by download attacks with a focus on MDN landing pages. We conjecture that landing pages within an MDN will exhibit a certain level of similarity regarding the webpage content. We validate this hypothesis using a large-scale dataset from a production search engine. In an MDN, this content maybe be similar across completely malicious webpages (webpages created by the attacker), legitimate but compromised webpages, or a combination of both. For malicious webpages, content within an MDN most likely varies to avoid filtering from search results by duplicate detection. However like the compromised webpages, a small amount of common, malicious code is used to initiate the drive-by download attack.

We propose a technique to extract the MDN specific redirection patterns and use them to detect previously-unknown drive-by download landing pages. Our technique is to start from a seed set of landing pages from already-known MDNs. We identify the common malicious content on these pages and then seek other pages in the search crawler cache that contain this content. These are good candidates to be previously-unknown landing pages associated with the MDN in question. We validate the results by submitting the found pages to a high-interaction honeypot. For example, 57% of the suspicious pages found by the rule-based system in Section III are verified as pointing to malicious drive-by downloads.

A high-level overview of the system environment is provided in Figure 1. The static crawler collects content from a wide range of webpages, most of which are benign, and gives this to the search engine. During this process, the static crawler may also fetch content from malicious webpages belonging to different MDNs. In parallel, the dynamic crawler is scanning unknown webpages in order to identify malicious pages, and the dynamic crawler output is used to identify individual MDNs. Our system consumes the static and dynamic crawl data, correlates the web content for pages belonging to individual MDNs, and identifies the malicious web content within each individual MDN. When a user enters a new query into the search engine, the search engine returns one or more SERPs (Search Engine Result Pages). Once the MDN content which is suspected as being malicious has been validated by the dynamic crawler, the corresponding webpage can be removed from the SERP producing a filtered SERP for the end user. The filtered SERP may block all, some, or none of the landing pages within an MDN.

This paper is organized as follows. We introduce necessary background material in the next section. In Section III we describe and evaluate a rule-based approach to the problem. In Section IV we present and evaluate a classifier approach to the same problem. Section V discusses the differences and merits of the two approaches. We review related work in Section VI.

II. Background

In this section, we provide background on three key aspects related to our system, namely malware distribution networks, the static search crawler, and the dynamic crawler.

A. Malware Distribution Networks

A malware distribution network (MDN) consists of three components: landing pages, one or more redirection servers, and the exploit servers. The attack starts when a browser requests content from the landing page. Sometimes the landing page redirects the user to an exploit server directly, but more often the landing page will redirect the user to other redirection servers before reaching the final exploit server. Multi-hop redirection usually exists in more sophisticated drive-by download attacks where the redirection servers in the middle of the infection process further examine different conditions ($e.g.$ browser type, version, plugins etc.) to decide which exploit server the browser should be directed to. For example, one redirection path might be followed by Firefox browsers while another would be taken by particular versions of Internet Explorer. The landing page will trigger the first hop of redirection. For compromised webpages, redirection is usually caused by injected content. In an MDN, an exploit server may handle traffic from a large number of landing pages.

This architecture offers several advantages to the attackers. It separates the functions of traffic acquisition, traffic redirection and exploit serving. The redirection path can be obfuscated. The architecture facilitates management and auditing of traffic in the redirection layer, which allows for the possibility that the compromise of landing pages and redirection layers are controlled by a different party than the one hosting the exploit servers.

B. Static Search Crawler

Search engines employ crawlers to retrieve content for indexing. For this project we had access to the crawler of a large commercial search engine. The crawler runs continuously visiting new pages as found, and revisiting existing pages based on a schedule determined by the pages' changefulness and ranking. The crawler retrieves the content of a page for analysis. Some, but not all, of the links identified in the content are added to the list of pages to be subsequently crawled. The crawler does not however fetch embedded images or execute any scripts on the page, or attempt to render the page as a browser would. As it retrieves the static, but not dynamic, content from webpages it is commonly referred to as the static crawler. For example, in fetching the page www.nytimes.com (fetched April 16, 2012) in a browser a total of 176 requests were issued to 31 different domains. Of the 176 requests, 54 were jpeg, 40 gif, 4 png, 7 css, 12 flash, and 33 were javascript objects. None of these objects would be fetched by the static crawler, which limits itself to static text and HTML content. Thus a server that uses javascript to point to a server hosting malicious content does not exhibit suspicious behavior to the static crawler.

TABLE I: An example of the output of the dynamic crawler or DCTrace.

DCTrace	
Landing Page	www.foo.com/index.html
Redirection URLs	www.a.com/redirect.js www.b.com/check.php www.c.com/hack.js
Exploit URL	www.evil.com/malware.exe
IPs	www.foo.com (23.21.215.24) www.a.com (192.168.0.1) www.b.com (192.168.0.2) www.c.com (192.168.0.3) www.evil.com (192.220.74.179)
File Hash	E21AD55HCCSAD7DC21B....74R
Is Drive-By Successful?	True

Servers that host malicious content (as opposed to pointing to other servers that host it) will often attempt to hide their nature from crawlers by cloaking [6]. This is a technique that involves delivering malicious content to potentially vulnerable visitors, but innocent content to crawlers. Since web crawlers for major search engines operate from easily-identified blocks of IP addresses, and strictly obey any crawling policies put in place by `robots.txt`, it is simple to offer them different content from regular web users. Thus, an exploit server which seeks to evade discovery will not typically be reached via the links placed in innocent pages and has no difficulty showing an innocent face to any crawler that finds it by another path.

C. Dynamic Crawler

While the information retrieved by the static crawler suffices for search indexing, it does not reveal if a webpage is attempting to infect the user's machine with a drive-by download or not. Thus, a malicious embedded script which causes the browser to follow a series of redirects terminating at the exploit server, will never be followed.

Thus, many search engines, in addition to the *static crawler*, have a second *dynamic crawler* which examines a web page more thoroughly. The dynamic crawler can be thought of as a active, client-side honeypot. It visits a site posing as a vulnerable browser, and runs all the scripts on the page. In the www.nytimes.com example above, it would fetch all of the Flash and javascript objects and execute them. If those scripts involve fetching other links, these links would also be followed.

The dynamic crawler uses different vulnerable browsers and OS components to trigger possible malicious reactions. If any attempts to exploit known vulnerabilities are detected, the site will be flagged as potentially malicious. Since all of this must happen in an isolation environment, it is orders of magnitude slower than the static crawler. It is simply infeasible to comprehensively crawl a significant fraction of the web using the dynamic crawler. Thus the dynamic crawler must be reserved for pages that are suspected of being malicious. We refer to the output of dynamic crawler as the *DCTrace*. This contains all logged activity associated with the page load. Thus it has hostnames and IP addresses of all links followed. An example is shown in Table I.

As shown in Figure 2, our system takes two inputs: the DCTraces from the dynamic crawler (DCTrace) and web content of landing pages from the static crawler (LPages). Here we focus on the malicious DCTraces that are identified as a drive-by download by the dynamic crawler.

III. Rule-Based Landing Page Detection

In this section, we present our study on drive-by download attacks and our system for landing page detection. In the spirit of SNORT [7], we propose a rule-based detection method which involves learning a set of rules, in this case the presence or absence of a set of strings and clusters of strings, that are used to detect individual MDNs.

The MDN represents the coordinated work of a person or group attempting to funnel traffic from potentially innocent landing pages to exploit servers. Thus it is likely that similar (or the same) content occurs on many of the landing pages. Given the scale of many MDNs (which contain hundreds or even thousands of landing pages as shown in Figure 3) it is likely that the process of injecting malicious content into compromised pages is done by script, rather than by hand. Thus, it should be possible to identify previously-unknown landing pages by searching the static crawl cache for the malicious content already found on known-bad MDN landing pages. This can be seen to be the case in Table II and Table III, where we show strings suspected of being injected content found on the landing pages associated with several MDNs and the corresponding domains or URLs.

The basic architecture of our system is shown in Figure 2. We start with an initial set of MDNs. We use Arrow as described by Zhang *et al* [8], but our system is agnostic in this regard: we merely need a collection of MDNs to get started. From the landing pages of these MDNs we extract strings which are candidates to be the common content which in many cases, causes the first redirection. We then cluster those strings and use multiclass feature selection to identify features that best represent a particular MDN. We then search in the static crawl cache for pages that possess those strings. In this way we are able to discover pages within the static crawl cache that are, with high likelihood, previously-undetected members of already-known MDNs. Finally, we submit those pages to the dynamic crawler to verify if they indeed lead to exploit servers. We now review these steps in more detail.

A. Initial MDN Discovery

We use Arrow to discover an initial set of MDNs. Here we only briefly review the Arrow system and refer readers to the paper [8] for details. Arrow works from a set of exploit servers to find MDNs. It takes the DCTrace (illustrated by Table I) from the dynamic crawler, and then groups malicious traces that lead to the same exploit server into one MDN. This initial set of malicious DCTraces were detected by the production dynamic crawler during normal webpage scanning.

MDNs can have complex structures. In general they consist of a set of landing pages, redirection servers and exploit servers. However, in order to prevent easy blacklisting, MDNs often use fast-flux techniques whereby they change IP addresses and hostnames frequently. Rather than use single a domain name or IP address to denote a server, Arrow develops a Host-IP-Cluster (HIC) to identify servers in the traces. Simply put, Host-IP-Cluster is a data structure to represent a group of hostnames that share a large percentage of IPs. Details regarding HIC computation can be found in [8]. This technique adds a level of robustness to Arrow's ability to identify MDNs. We use Host-IP-Clusters just as Arrow does. As shown in Figure 2, we feed the output from the dynamic crawler (DCTraces) to Arrow and receive the MDNs as output. The original landing pages produced by Arrow then become the initial seeds for our system.

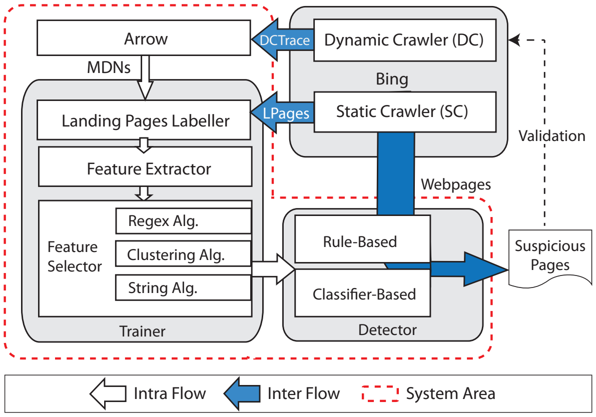

Fig. 2: System Diagram

B. Extracting Malicious Features from Landing Pages

Having found a collection of MDNs, we now examine the web content of landing pages within an MDN. The first task is to detect common, malicious code segments in the landing pages of the identified MDNs. We do this by examining the content of these pages in the static crawler cache. From the content, we extract all of the strings that potentially cause malicious redirection. HTML elements such as <script>, <iframe>, <form>, <frame>, <object> or <embed> are potential sources of redirection. This stage produces many strings per page, the majority of which will obviously be innocent. For example, the www.nytimes.com page considered earlier contained 28 <script> strings (obviously we use this page as example only, and it is not an MDN landing page). We consider each string as a potential feature.

After processing a large volume of landing pages, we have a feature space that contains many times more features (*i.e.*, strings) than the number of landing pages. Each landing page, indexed by its associated MDN, will also have a binary vector indicating whether it contains a particular string or not. This vector is very sparse; *i.e.*, most strings appear on only a small number of the landing pages.

C. String Clustering

In the next section, the feature selection process selects the most discriminative features (*i.e.*, strings) for particular MDNs. However, these features can be brittle: some MDNs use

TABLE II: String features suspected of being injected content on known MDN landing pages. Observe that each is a link, and that some show slight polymorphism. The feature number refers to a string-cluster rather than a string. Note: these features are merely suspicious, and have not been validated.

Feature ID	Feature String
642	<script language=“javascript” type=“text/javascript” src=“http://js.users.51.la/2406109.js”></script>
642	<script language=“javascript” type=“text/javascript” src=“http://js.users.51.la/4456469.js”></script>
642	<script language=“javascript” type=“text/javascript” src=“http://js.users.51.la/627317.js”></script>
442	<iframe src=”http://zlocorp.com/1010/in.cgi?14” width=”100%” height=”1” scrolling=”no” frameborder=”0” ...
442	<iframe src=”http://zlocorp.com/1010/in.cgi?14” width=”100%” height=”2” scrolling=”no” frameborder=”0” ...
442	<iframe src=”http://zlocorp.com/1010/in.cgi?14” width=”100%” height=”7” scrolling=”no” frameborder=”0” ...
2203	<script type=”text/javascript”>/* <![CDATA[*/ bmone2n.makeAd(’14216.1.1.1’); /*]]> */</script>
2203	<script type=”text/javascript”>/* <![CDATA[*/ bmone2n.makeAd(’14216.1.1.12’); /*]]> */</script>
2203	<script type=”text/javascript”>/* <![CDATA[*/ bmone2n.makeAd(’14216.1.1.2’); /*]]> */</script>
2203	<script type=”text/javascript”>/* <![CDATA[*/ bmone2n.makeAd(’14216.1.1.7’); /*]]> */</script>

TABLE III: Domains or URLs of landing pages which include members of the string cluster features in Table III. Some of these string clusters, 642, occur on a wide range of domains, others are found within the same top tier domain, 2203, while others are found on landing pages within the same hostname, 442. All of these were determined to be malicious at some time in the past.

Feature ID	URL
642	bzmc.flash-soul.com/html/skill/201103/1292.html
642	www.xn–4pvoj466jvub.net/faq.php
642	kdpiao.cn
642	www.fushi123.cn
642	www.zbzwow.cn/bbs/viewthread.php
642	www.hhhhu.com/info.html
442	fcguy.atspace.name/89.html
442	fcguy.atspace.name/site-12.html
2203	www.akw81.yoyo.pl/art_30.php
2203	www.amalysz.yoyo.pl/l.html
2203	www.iwadala.yoyo.pl/darmowa-wersja-gry-fim-speedway-grand-prix-2-do-pobrania.html
2203	stowarzyszenie-amicus.yoyo.pl

polymorphism that impairs the effectiveness of the feature selection. In certain MDNs, the malicious content varies slightly from landing page to landing page. For example, in Table II, the first three strings are almost, but not quite, identical. Though the injected content is essentially the same code “exact matching” may be ineffective in the feature selection process. This polymorphism of the injected content is quite common and has the effect of expanding the feature space considerably. As a result, feature selection based on “exact matching” fails to learn the similar malicious strings employed in one MDN resulting in a failed detection.

To address this problem, we develop a clustering module for the extracted strings based on the string similarity. Even though the polymorphic content is in a different form, the main body of the code and the code logic remain the same. So for each entity, we transform each string into a set of trigrams, and use the Jaccard distance between two strings defined as:

$$D_{12} = 1 - \frac{Intersection(Set_1, Set_2)}{Union(Set_1, Set_2)},$$

where Set_k is the set of trigrams generated from the k-th string. For example, if one string contains trigrams a, b, c and d, and a second contains b, d, e, f and g the distance between them would be $1 - 2/7 \approx 0.71$. This proves a powerful technique in grouping the variants of polymorphic injection: minor polymorphic variations end up being close under this distance measure. This allows us to cluster the sets of strings into groups. We cluster these trigram sets using the $ISODATA$ [9] algorithm which does not require a pre-selected number of clusters.

This technique may reduce the total number of candidate features from a very large number of strings to a much smaller number of string clusters. After feature selection, we end up with a set of rules consisting of string clusters which we refer as STRING CLUSTER. A string cluster may consist of one string which can be viewed as a cluster containing a single item. This feature set will also be used as one type of features in the classifier-based approach in the following section.

D. Feature Selection

Our task is now a feature selection problem: we seek to select the strings or string clusters that best represent a particular MDN but are not indicative of webpages from either legitimate sites or other MDNs. Recall that landing pages of infected sites have little in common other than their membership of the same MDN. Thus, strings that are common, or even similar, between them are good candidates to have been written by the MDN owner rather than the owners of the landing pages. That is, strings that appear on the landing pages of one MDN but seldom (or never) on those of other MDNs or on benign pages are good features to characterize this particular MDN. The strings that do best in distinguishing between MDNs, and more importantly legitimate pages, emerge in this feature selection phase.

We use a feature selection algorithm based on the mutual information [10] between the i-th MDN and the k-th feature. Define A (resp. C) as the number of landing pages not in the i-th MDN that do not contain (resp. do contain) the k-th feature. Define B (resp. D) as the number of landing pages in the i-th MDN that do not contain (resp. do contain) the k-th feature. Then a maximum likelihood estimate of the information provided about membership in the i-th MDN by the k-th feature is:

$$\begin{aligned} R(f) &= \frac{D}{N}\log_2\frac{N\cdot D}{(\hat{BD})(\hat{CD})}+\frac{B}{N}\log_2\frac{N\cdot B}{(\hat{AB})(\hat{BD})} \\ &+ \frac{C}{N}\log_2\frac{N\cdot C}{(\hat{CD})(\hat{AC})}+\frac{A}{N}\log_2\frac{N\cdot A}{(\hat{AB})(\hat{AC})} \end{aligned}$$

where $\hat{AB} = (A + B)$, $\hat{AC} = (A + C)$, $\hat{BD} = (B + D)$, $\hat{CD} = (C + D)$, and $N = A + B + C + D$.

Finally, the set of potential features is ranked for each MDN based on the scores. We select the top 5 ranked features for each MDN, which best discriminate that particular MDN with other MDNs and benign webpages under consideration. This method not only effectively selects malicious code, but also excludes benign injected code, like normal third-party web tracking code (*e.g.*Google Analytics). As normal web tracking code frequently appears in the benign webpage set, their ranking in MDNs would be lowered in the feature selection process.

E. False Positive Pruning

The rule-based approach is very straightforward: we take the strings and string-clusters found in Sections III-B and III-C, and then search for pages that contain them in the static crawler cache. Recall that these strings are common among the already-found landing pages of a single MDN, and are thus with high probability injected malicious content. However, it is possible that landing pages that belong to very different sites contain common content that is not malicious. This can happen if both sites are using a template or web-site authoring tool that produces content in a certain form.

With a view to reducing the possibility of such false positives we do as follows. We search for each of our candidate strings in a randomly selected sparse 10% subset of the static crawler cache. Any string that co-occurs on landing pages as result of a template or authoring tool should be found many times in this sparse set. Many of these pages will also have been scanned by the dynamic crawler. A feature is retained if 60% or more of the pages that contain the feature and have been previously scanned by the dynamic crawler were also detected as being malicious. Otherwise, we regard the string as a benign feature and remove it from the candidate feature list. This step is used as a sanity check to avoid producing large numbers of false positives.

F. Rule-Based Experimental Results

We describe the experimental results used to validate our proposed rule-based method in the following section. We begin by describing the preprocessing step which identifies the MDNs determined from a large collection of DCTraces. Next, the setup and the rule-based system results are analyzed. Finally the validation results on the dynamic crawler for this system is provided.

1) PreProcessing: Referring to the system diagram in Figure 2, we randomly selected 6.0 million malicious DCTraces that were diagnosed as drive-by download attacks by the production dynamic crawler. The traces are selected within a 60-day period ending on two dates, August 25, 2011 and January 2, 2012. The first 60-day trace ending in August 2011 is used to conduct preliminary experiments with the rule-based system and also evaluate the classifier-based approach described in the following section. Here, for brevity, we mainly describe the details of the January 2, 2012 collection which serves as our primary DCTrace data set.

We use Arrow to group the DCTraces into MDNs and use these landing pages in each MDN as the input to the remainder of our system for evaluation. After filtering out small MDNs containing no more than 20 distinct DCTraces, Arrow discovered 53 MDNs with central servers and a total of 719,089 landing pages. (We choose to study MDNs with a central server topology because these architectures can be more complex: the central server sometimes uses fast-flux techniques to control a collection of exploit servers [8].) Of these we randomly selected 32,000 landing pages as malicious webpage samples.

From a separate source we obtain 32,000 pages for our benign data set. These webpages are known to be legitimate since they are from reputable websites and have been manually graded by human analysts. These webpages are a subset of the those used to train the search engine's ranker.

Thus, we have 32,000 each of known-good and known-bad pages for training. We next obtain the webpage content for these from the static crawler and retain only the strings as described in Section III-B. We then generate a sparse binary data set where each row represents a webpage and each column represents a string found on at least one of the webpages.

The final step for creating this experiment's data set is to label each webpage. Each MDN has its own label and malicious landing pages are labeled based on the MDN it belongs to. On the other hand, all 32,000 benign pages are grouped under the same label. Thus our feature dataset has webpages as rows and features as columns. A column is set to true if the page in question contains that feature. There is one additional column, which contains the label to indicate which group the page belongs to (here group means a particular MDN or the collection of benign pages).

2) Rule-Based Detection System: The features for the rule-based system consist of individual strings and clusters of strings. Selecting a maximum of 5 string or string-cluster features per MDN, feature selection yielded 538 total individual strings from the 53 MDNs.

An example of some of these strings is shown in Table II. As detailed in Section III-E, we submitted these candidate features to a 10% sub-sample of the daily crawl results from the static crawler. This produced over 2.9M URLs, which might be considered suspicious. We prune the list by discarding

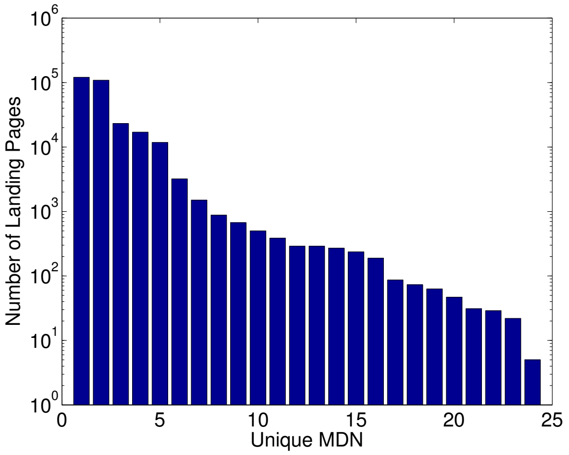

Fig. 3: Number of landing pages predicted to belong to each individual MDN as determined by the rule-based system after false positive pruning. The MDN landing page count varies by several orders of magnitude with the largest MDN having over 100,000 pages.

features that return less than 60% of detections on pages marked malicious by the dynamic crawler. This reduces the candidate feature list to a final set of 128 high-precision features. This feature list was then submitted to the static crawler to determine all web pages scanned on January 20, 2012 containing at least one of these features. This final step produced a list of 289,087 URLs which we suspect to be landing pages belonging to 24 MDNs. The number of landing pages per MDN is provided in Figure 3.

3) Validation: The last step is to validate the effectiveness of the system. To do so, we compare our method to three systems: the production dynamic crawler, the Nozzle [11] system, and the Zozzle [12] system. We first submitted a subset of the discovered URLs to the production dynamic crawler. The validation results for the rule-based system for the August 25, 2011 and January 2, 2012 datasets are given in Table IV. Forced to be parsimonious in our use of the production dynamic crawler (which is a heavily used resource) we were able to submit only a portion of our newly detected URLs. We submitted only 48,189 pages randomly selected from the August dataset and 88,503 from the January dataset.

The dynamic crawler detected 57.2% of the webpages as malicious for August 25, 2011. Likewise, 58.1% of the 88,503 evaluated by the dynamic crawler for January 2, 2012 dataset were determined to be malicious. This indicates that the proposed system has very high precision, given that malicious webpages on the internet constitute a very small fraction of the whole and random sampling is not viable. In fact, in a production environment, a automated detection system is regarded as actionable if 1% of the pages that are predicted to be malicious are confirmed. Unfortunately we cannot analyze the system's recall since it is impossible to know how many false negatives there are. Using the dynamic crawler to scan this web-scale dataset is prohibitive for us due to the computational costs. While 58.1% were verified malicious only 42.6% had not previously been discovered. At this rate, the expected number of new landing pages for these MDNs is $289,087 \times 0.426 = 123,071$. Thus, since we started with $719,089$ landing pages we have expanded the verified footprint of the MDNs by $100 \times 123/719 \approx 17.1\%$. It's also worth noting that the detection rates remained consistent over a period of five months.

In addition, we also evaluated these same webpages using the Nozzle and Zozzle systems. Nozzle detects heap sprays in the malicious code in runtime, while Zozzle is a classification system to detect malicious javascript. Table IV shows that Nozzle did not find any of the webpages identified by our method while Zozzle detected four for the first dataset and 350 for the second. We make several observations from these results. First, the rule-based detection system does a very good job of identifying malicious webpages; our system provided webpages that were almost 60% malicious. Furthermore, the detection system is essentially orthogonal to the Nozzle and Zozzle systems. That is, our system is complementary to their method of discovery.

IV. Classifier-Based MDN Detection

The rule-based method for detecting MDN landing pages exhibited high precision as indicated in Table IV: over 57% of pages the system deemed suspicious turned out to be hosting a drive-by download attack. We now explore a second, classifier-based landing page detection method and examine whether it can produce a better result.

A. Classifier Features and Regular Expression Generation

The classifier borrows much from the rule-based system, including the preprocessing step (identifying the MDNs from DCTraces, described in Section III-F1) and the string cluster features (Section III-C). A departure from the rule-based method is that we also investigate two additional types of features: individual string features in isolation, and regular expression features.

The regular expression generator is based on the algorithm proposed in [13] which demonstrated considerable efficacy in accurately capturing spam URLs. With a set of strings as input, it generates one or more regular expressions that match the strings in the set. This may be able to capture more generic forms of features than the string cluster features. The three types of features are labeled as String Match, String Cluster and Regex Cluster in the evaluation figures.

B. Classifier Training

We use the 60-day DCTraces ending on August 25, 2011 to train the classifiers and understand the performance of different feature sets. After the MDN labeling and feature selection steps, we next construct labeled, sparse binary datasets for each of the three feature sets. Rows represent webpages and columns represent features. Each dataset consists of 64,000 rows half of which are constructed from pages belonging to our benign webpage collection and the other half from landing pages of known MDNs. An element in the dataset is set to true if the webpage associated with the row contains the feature associated with the column (*i.e.*, contains the string, an element of the string cluster, or matches the regular expression).

TABLE IV: Dynamic Crawler Evaluation.

DC Trace Data End Date	Total URLs Scanned	Proposed Method	Pages Detected By Proposed Method	Malicious Webpage Detection Rate (%)	Nozzle	Zozzle
Aug 25, 2011	48,189	Rule-Based	27,563	57.2	0	4
Jan 2, 2012	88,503	Rule-Based	51,567	58.3	0	350
Jan 2, 2012	110,982	Classifier-Based Highest Probability	1,363	1.2	0	0
Jan 2, 2012	102,919	Classifier-Based Random Sampling	1,617	1.6	0	2

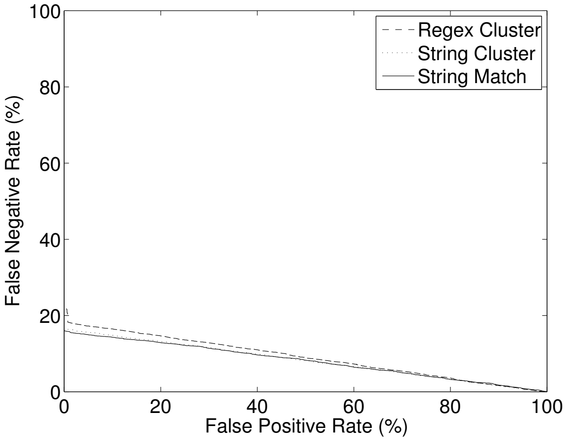

Fig. 4: Detection error tradeoff curves for the three proposed feature sets for dynamic trace data over 60 days ending August 25, 2011.

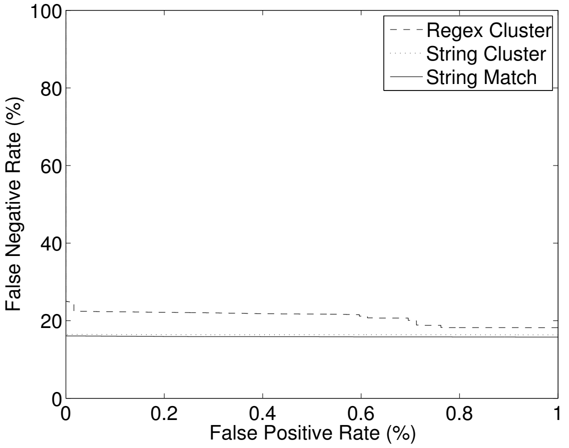

Fig. 5: A zoomed-in view of the detection error tradeoff curves in Figure 4 for a maximum false positive rate of 1% .

We train separate classifiers for each feature set using multi-class logistic regression [14]. Each MDN is considered one class, and all of the benign pages are considered as belonging to a single class. Once training is complete, an unknown webpage is evaluated by calculating its feature vector and predicting which class (*i.e.*, particular MDN or the benign set) it is most likely to belong.

C. Classifier Performance Evaluation

Figure 4 provides the 5-fold cross validation, Detection Error Trade-Off (DET) curves for the STRING MATCH, STRING CLUSTER and REGEX CLUSTER classifiers. The low false positive region of these curves is highlighted in Figure 5. These curves demonstrate that the three algorithms are comparable, with STRING MATCH algorithm offering a minor improvement.

The shape of the DET curves requires further investigation. Figure 6 provides the histogram of the STRING CLUSTER classifier's score, $s(x_i)$, which is the log odds that landing page i is malicious. The log odds is computed as $s(x_i) = \log(P(y_i = Malicious|x_i)/P(y_i = Benign|x_i))$ where y_i and x_i are the label and the feature vector determined by the set of strings extracted from the i-th webpage, respectively. Bars in dark color in Figure 6 represent samples from benign webpages while malicious pages are indicated by the white bars. We first observe a nice separation between the distributions of the malicious and benign pages. We also note five false positives in this particular fold of the classifier training. The false positive rate increases in the DET curves as we sweep a threshold from right to left in the log odds histogram.

The distribution of the STRING CLUSTER classifier probabilities for a random sample of one million URLs is shown in Figure 7. Noting the log scale, this figure indicates many distinct, and potentially large, sets of URLs which include one or more of the string cluster features. This distribution indicates that in many cases, the classifier will either identify a large number of truly malicious webpages, or an extremely large number of false positives.

D. Classifier Validation

We also evaluate the results of the STRING CLUSTER classification system. To be parsimonious in our use of the production dynamic crawler we submit only the URLs produced by the STRING CLUSTER system, since there is little difference between STRING CLUSTER and the STRING MATCH approach, and both outperform REGEX CLUSTER.

The validation results for the classifier-based system for the January 2012 dataset are given in Table IV. We evaluated this method in two ways, namely Highest Probability and Random Sampling. For the Highest Probability experiment, we selected URLs which were predicted to be malicious by

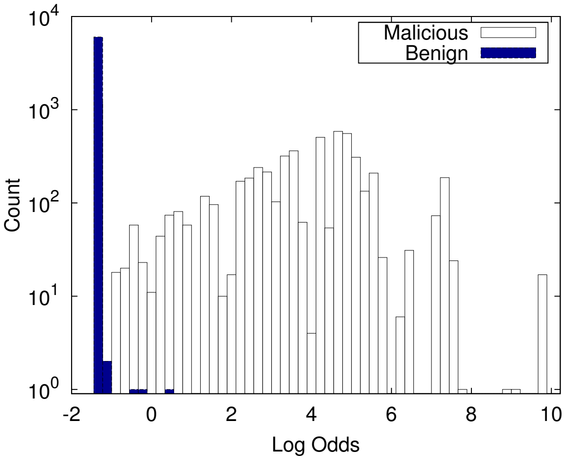

Fig. 6: String cluster classifier score (i.e. log odds) for the for dynamic crawler trace data over 60 days ending August 25, 2011. The malicious and benign classes separate nicely, but we do note five false positives in the figure.

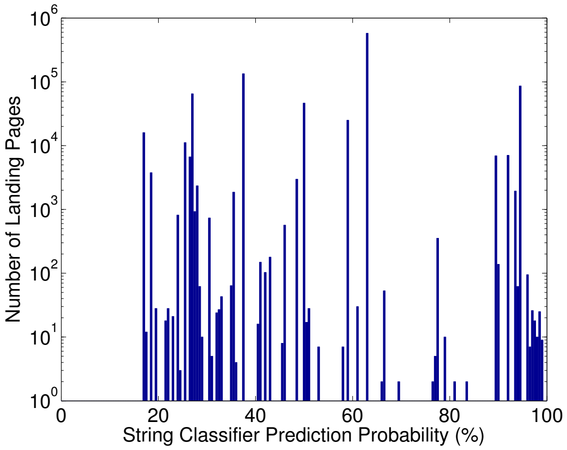

Fig. 7: Histogram of string cluster classifier probabilities for a subsample of one million URLs. Distinct sets of features may be included in an extremely large number of webpages.

the classifier with the highest probability. Similarly for the Random Sampling experiment, we chose URLs at random.

From the table, we make several observations. First, the classifier-based method detects a much lower percentage of URLs than the rule-based system. Second, the detection rate for the Highest Probability and Random Sampling methods are roughly comparable. Third, the classifier performs nowhere near as well as Figure 4 might lead us to hope. That is, Figure 4 suggests that a false negative rate of less than 20% can be achieved with essentially no false positives. In practice, Table IV shows a false positive rate of 98.4%: the classifier fails to deliver the performance that it displayed on the test set in training. We explore some of the reasons for this failure in Section V.

Finally, in the following experiment, we seek to understand whether the classifier-based approach can detect the same sets of URLs that are detected by the rule based approach. To do so, we compute the overlap of the rule-based system compared to the classifier-based system with High Probability URL selection, $\hat{O}_{RB,HP}$:

$$\hat{O}_{RB,HP} = \frac{\mid RBD \cap HPD \mid}{\mid HPD \mid} = 70.5\% \qquad (1)$$

where RBD is the set of detected URLs from the rule-based method and HPD is the set of detected URLs from the high probability classifier method. By detected, we mean the set of URLs predicted to be malicious by a method and later confirmed by the dynamic crawler. Similarly, the overlap of the rule-based system with respect to the classifier-based system with Random Sampling URL selection is $\hat{O}_{RB,RS} = 23.1\%$. From these estimates, we note the rule-based method tends to select landing pages which are predicted by the classifier to have a higher probability of being malicious. We also confirm that the classifier-based approach only detects a subset of the URLs detected by rule-based approach.

V. Discussions

In the previous two sections, we proposed systems to detect MDN landing pages utilizing either rules or a classifier. The results from Table IV clearly indicate that the rule-based system outperforms the classifier in terms of validation rate. This improvement is primarily due to the inclusion of the false positive pruning stage. As shown in Figure 7, the classifier predicts that some pages are more likely to be malicious than others. At the start of the study, we expected that this would allow us to select an operating threshold for the classifier to automatically detect malicious webpages while avoiding false positive pruning. However, the results from Table IV failed to support this hypothesis. As confirmed by the dynamic crawler, the false positive rates for the classifier are significantly higher than expected given the DET results obtained during training for both the high probability and random sampling experiments.

One possible reason for the high false positives of the classifiers is that our training data set, especially benign set, has limited coverage. For example, we discovered that some of the false positives produced by the classifier were caused by a string related to accessing an analytics service from the Russian search engine Yandex. Upon further investigation, we confirmed that none of the content from our legitimate webpages accessed this service, but several of the MDNs using Russian URLs (*i.e.*, .ru) did. This implies that if we include a more comprehensive benign set (some of which may use this analytic service), we can effectively reduce the false positives of the classifiers.

A possible usage case for the second classifier-based strategy is to rank the detected webpages so that more malicious webpages can be verified first. At the present time, we cannot use the dynamic crawler to scan all possible URLs detected by our detector. However, one could consider verifying a more targeted subset of URLs with a production dynamic crawler. Instead of validating a random URL sample or a sample corresponding to the highest classifier probabilities, one could instead sample a fixed number of URLs (e.g. 100) which include one or more of the features identified during feature selection and validate individual features based on the percentage of landing pages confirmed to be malicious. This is similar to the proposed false positive pruning strategy, but instead of considering only URLs which have been scanned by the dynamic crawler in the past, we verify all possible injected strings. In this way, the method could consider new string cluster features which have not been found on previously detected webpages.

There are several cases where attackers could evade or abuse the detection system. For example, in the false positive pruning stage of rule-based detection, a feature is retained if 60% or more of the pages containing the feature were previously scanned by the dynamic crawler and detected as malicious. Hypothetically, attackers can abuse the pruning stage by including this feature in 41% of the landing pages which do not launch an attack. However, we argue that this is unlikely to happen in practice considering the overhead and difficulty of injecting malicious code in legitimate landing pages. Giving up 41% of injected landing pages is a big loss for attackers.

In addition, polymorphic and obfuscated injection is another challenge for our system. Today, attackers tend to hide the real purpose of the injected code by making it unreadable. There are limitless ways to obfuscate a piece of code to give it a totally different and unreadable appearance. Our scheme works against content that is either obfuscated or polymorphic, but not both. In an extreme case where the attacker obfuscated the code in different ways on every single landing page in the MDN, the feature selection algorithm will fail to recognize these malicious strings.

Our system works together with both the dynamic and static crawlers. There are several common challenges that crawlers have that may affect our system. First, the dynamic crawler exposes vulnerable components to webpages and detect malicious responses. Thus it is possible that the dynamic crawler may fail to detect some of the attacks due to a limited configuration of vulnerable components. In addition, cloaking is concern for both the static and dynamic crawlers. Attackers can identify the crawler (*e.g.*, by knowing the IP addresses from which it operates or detecting the presence of the virtual machine), and evade detection by not attempting to exploit a vulnerability or returning legitimate content.

Finally, our system is limited in the fact that it requires an initial set of MDNs. Thus our system can be considered supplemental to other detection techniques rather than a standalone system.

VI. Related Work

To defend against drive-by download attacks, a number of systems have been proposed from both the research and industrial communities. These systems use different strategies to detect drive-by downloads, including classifying malicious URLs or web content, actively exploiting web servers with client-side honeypot, identifying attacks from a single instance or correlating multiple landing pages within a campaign. We categorize these systems and techniques from the following perspectives.

First, most systems start from landing pages: they visit the malicious websites, execute the embedded script, and traverses the secondary URLs while monitoring suspicious state changes of the operating system in order to detect drive-by downloads. For example, systems like [15], [16], [17], [18] will dynamically execute the web content and capture drive-by downloads based on either signatures or anomaly detection. PhoneyC [19] and WebPatrol [20] use a signature-based, low-interaction honeypot to detect rogue websites. In addition, Nozzle [11] is a runtime monitoring infrastructure which detects malicious heap spray attempts, while JSAND [21] takes a machine learning approach to classify malicious javascript. Rozzle [22] is another javascript virtual machine. It explores multiple execution paths within a single execution in order to expand the possibility of triggering the malicious scripts. Finally, Blade [23] leverages user behavior models for drive-by download detection. All of these systems have shown good detection results. However, it is usually costly to follow the full redirection path and monitor each script execution in runtime. Moreover, their accuracy is highly dependent on the webpage's malicious response to vulnerable components.

Some systems work in the reverse direction: they start with servers hosting malicious exploits and reverse the redirection path to discover the corresponding landing pages. Leveraging the drive-by download traces contributed by dynamic crawlers or public anti-virus databases, these systems could discover the campaign-like malware distribution networks (MDNs) [4]. Existing systems [8], [24] fall into this category. WebCop [24] uses static hyperlinks to build a web graph of malware distribution networks. The limitations of WebCop are the static web graph and exact match approach, which are less resilient to dynamic changes of the web. Arrow [8] detects and generates URL signatures for the central server in complex MDNs in order to detect more landing pages sharing these central servers. In this paper, we also follow this approach in discovering MDNs. Unlike [8], [24], we leverage both the static and dynamic crawlers and focus on the malicious code in the landing page content instead of URLs. Arrow can only find additional webpages which have previously scanned, but undetected, by the dynamic crawler.

In addition, static analysis is a technique adopted by many existing systems [12], [25], [26] to detect malicious webpages. Most of these systems explore multiple features from malicious javascript code: for example, Zozzle [12] focuses on the contextual features in javascript code to detect heap spraying. Systems like Prophiler [25] consider both javascript features and additional features extracted from the HTML content and URLs of malicious pages. A recent system EvilSeed [26] generates gadgets by analyzing the page content, DNS traces and link topology of known malicious pages, and then uses the gadgets to discover similar pages. Unlike the dynamic analysis of the honeypot approach, the mentioned static analysis is more light-weight and less time-consuming. Our work differs from existing work in this detection scope and considered features: previous systems are generic detectors for generally malicious content in webpages, while our system precisely targets the landing page features of specific MDNs.

Finally, SURF [27] and deSEO [28] study the malware distribution campaigns from the perspective of blackhat Search Engine Optimization (SEO). In a typical drive-by download attack, one important function of landing pages is to drive traffic to the exploit servers. Attackers have been trying different SEO techniques to rank malicious landing pages high in the search results for popular queries. SURF uses the features extracted from the *search-then-visit* browsing sessions to detect malicious redirections from the search results. It can work as a scanner in the dynamic crawler. deSEO, on the other hand, is a signature-based detector using the patterns of the malicious URLs in the search results. In our approach, we use URLs in the redirection paths to discover large malware distribution networks, but our detector focuses more on the malicious code in the web content.

VII. CONCLUSIONS

In this paper, we analyze landing pages belonging to malware distribution networks (MDNs) which lead to drive-by download attacks. Using a large set of drive-by download traces from a production dynamic crawler, we show that landing pages within the same MDN have a certain level of similarity in their malicious content that causes redirection. With the aid of multiclass feature selection, we are able to identify the MDN-specific features that best discriminate landing pages of one MDN with those of other MDNs and benign webpages.

We propose and implement two new solutions to efficiently detect malicious landing pages. The first, rule-based system, which is based on matching clusters of strings, exhibits high precision in identifying malicious injected code in the landing pages within the MDN. This system produces a list of URLs out of which over 57% are independently validated as malicious by a production dynamic crawler. This success rate remains constant in two large trials spaced five months apart. This extends the known footprint of the MDNs studied by 17%.

The second system, which implements a classifier, also produces set of URLs of which approximately 1% are independently confirmed to be malicious. Compared to the rule-based method, we conclude that the classifier offers little utility in this setting without using a far greater training set. We include the results of the classifier system in the paper for completeness.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their feedback, Yinglian Xie and Fang Yu for providing software for computing regular expressions, Christian Seifert for insights on search engines and crawlers, and Sarmad Fayyaz for guidance on running experiments.

REFERENCES

[1] C. Jackson, "Improving Browser Security Policies," Ph.D. dissertation, Stanford University, 2009.

[2] R. Dhamija, J. D. Tygar, and M. Hearst, "Why phishing works," in *Proc. CHI*, 2006.

[3] S. Schechter, R. Dhamija, A. Ozment, I. Fischer, "The Emperor's New Security Indicators: An evaluation of website authentication and the effect of role playing on usability studies," in *Proc. IEEE Symposium on Security and Privacy*, 2007.

[4] N. Provos, P. Mavrommatis, M. Abu Rajab and F. Monrose, "All your iframes points to us," in *Proc. USENIX Security*, 2008.

[5] Trustseer Security Advisory, "Flash Security Hole Advisory," http://www.trusteer.com/files/Flash_Security_Hole_Advisory.pdf, 2009.

[6] S. S. David Wang and G. M. Voelker, "Cloak and dagger: Dynamics of web search cloaking," in *Proc. ACM CCS*, 2011.

[7] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proc. USENIX LISA*, 1999.

[8] J. Zhang, C. Seifert, J. W. Stokes, and W. Lee, "Arrow: Generating signatures to detect drive-by downloads," in *Proc. WWW*, 2011.

[9] G. Ball and D. Hall, "ISODATA, a novel method of data analysis and pattern classification," DTIC Document, Tech. Rep., 1965.

[10] C. D. Manning, P. Raghavan, and H. Schtze, *An Introduction to Information Retrieval*. Cambridge University Press, 2009.

[11] P. Ratanaworabhan, B. Livshits, and B. Zorn, "Nozzle: A defense against heap-spraying code injection attacks," in *Proc. USENIX Security*, 2009.

[12] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "Zozzle: Low-overhead mostly static javascript malware detection," in *Proc. USENIX Security*, 2011.

[13] Y. Xie, F. Yu, K. Achan, R. Panigraphy, G. Hulten and I. Osipkov, "Spamming botnets: Signatures and characteristics," in *Proc. ACM SIGCOMM*, 2008.

[14] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[15] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy, "A crawler-based study of spyware on the web," in *Proc. NDSS*, 2006.

[16] C. Seifert and R. Steenson, "Capture - honeypot client (capture-hpc)," https://projects.honeynet.org/capture-hpc, 2006.

[17] C. Seifert, R. Steenson, T. Holz, B. Yuan and M. A. Davis, "Know your enemy: Malicious web servers," http://www.honeynet.org/papers/mws/, 2007.

[18] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen and S. King, "Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities," in *Proc. NDSS*, 2006.

[19] J. Nazario, "Phoneyc: A virtual client honeypot," in *Proc. USENIX LEET*, 2009.

[20] K. Z. Chen, G. Gu, J. Zhuge, J. Nazario, and X. Han, "Webpatrol: automated collection and replay of web-based malware scenarios," in *Proc. ASIACCS*, 2011.

[21] M. Cova, C. Kruegel and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proc. WWW*, 2010.

[22] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert, "Rozzle: De-cloaking internet malware," in *Proc. IEEE Symposium on Security and Privacy*, 2012.

[23] L. Lu, V. Yegneswaran, P. Porras and W. Lee, "Blade: An attack-agnostic approach for preventing drive-by malware infections," in *Proc. ACM CCS*, 2010.

[24] J. W. Stokes, R. Andersen, C. Seifert and K. Chellapilla, "Webcop: Locating neighborhoods of malware on the web," in *Proc. USENIX LEET*, 2010.

[25] D. Canali, M. Cova, C. Kruegel, and G. Vigna, "Prophiler: A fast filter for the large-scale detection of malicious web pages," in *Proc. WWW*, 2011.

[26] L. Invernizzi, S. Benvenuti, P. M. Comparetti, M. Cova, C. Kruegel, and G. Vigna, "Evilseed: A guided approach to finding malicious web pages," in *Proc. IEEE Symposium on Security and Privacy*, 2012.

[27] L. Lu, R. Perdisci, and W. Lee, "SURF: detecting and measuring search poisoning," in *Proc. ACM CCS*, 2011.

[28] J. P. John, F. Yu, Y. Xie, A. Krishnamurthy, and M. Abadi, "eSEO: Combating search-result poisoning," in *Proc. USENIX Security*, 2011.