EVILSEED: A Guided Approach to Finding Malicious Web Pages

Luca Invernizzi
UC Santa Barbara
invernizzi@cs.ucsb.edu
Stefano Benvenuti
University of Genova
ste.benve86@gmail.com
Marco Cova
Lastline, Inc. and University of Birmingham
m.cova@cs.bham.ac.uk
Paolo Milani Comparetti
Lastline, Inc. and Vienna Univ. of Technology
pmilani@seclab.tuwien.ac.at
Christopher Kruegel
UC Santa Barbara
chris@cs.ucsb.edu
Giovanni Vigna
UC Santa Barbara
vigna@cs.ucsb.edu

***Abstract*—Malicious web pages that use drive-by download attacks or social engineering techniques to install unwanted software on a user's computer have become the main avenue for the propagation of malicious code. To search for malicious web pages, the first step is typically to use a crawler to collect URLs that are live on the Internet. Then, fast prefiltering techniques are employed to reduce the amount of pages that need to be examined by more precise, but slower, analysis tools (such as honeyclients). While effective, these techniques require a substantial amount of resources. A key reason is that the crawler encounters many pages on the web that are benign, that is, the "toxicity" of the stream of URLs being analyzed is low.**

In this paper, we present EVILSEED, an approach to search the web more efficiently for pages that are likely malicious. EVILSEED starts from an initial seed of known, malicious web pages. Using this seed, our system automatically generates search engines queries to identify other malicious pages that are similar or related to the ones in the initial seed. By doing so, EVILSEED leverages the crawling infrastructure of search engines to retrieve URLs that are much more likely to be malicious than a random page on the web. In other words EVILSEED increases the "toxicity" of the input URL stream. Also, we envision that the features that EVILSEED presents could be directly applied by search engines in their prefilters. We have implemented our approach, and we evaluated it on a large-scale dataset. The results show that EVILSEED is able to identify malicious web pages more efficiently when compared to crawler-based approaches.

***Keywords*-Web Security, Drive-By Downloads, Guided Crawling**

I. Introduction

The web has become the medium of choice for people to search for information, conduct business, and enjoy entertainment. At the same time, the web has also become the primary platform used by miscreants to attack users. For example, drive-by-download attacks are a popular choice among bot herders to grow their botnets. In a drive-by-download attack, the attacker infects a (usually benign) web site with malicious code that eventually leads to the exploitation of vulnerabilities in the web browsers (or plug-ins) of unsuspecting visitors [1], [2]. If successful, the exploit typically downloads and executes a malware binary, turning the host into a bot [3].

In addition to drive-by-download exploits, cybercriminals also use social engineering to trick victims into installing or running untrusted software. As an example, consider a web page that asks users to install a fake video player that is presumably necessary to show a video (when, in fact, it is a malware binary). Another example is fake anti-virus programs. These programs are spread by web pages that scare users into thinking that their machine is infected with malware, enticing them to download and execute an actual piece of malware as a remedy to the claimed infection- [4], [5].

The web is a very large place, and new pages (both legitimate and malicious) are added at a daunting pace. Attackers relentlessly scan for vulnerable hosts that can be exploited and leveraged to store malicious pages, which are than organized in complex malicious meshes to maximize the chances that a user will land on them. As a result, it is a challenging task to identify malicious pages as they appear on the web. However, it is critical to succeed at this task in order to protect web users. For example, one can leverage information about web pages that compromise visitors to create blacklists. Blacklists prevent users from accessing malicious content in the first place, and have become a popular defense solution that is supported by all major browsers. Moreover, the ability to quickly find malicious pages is necessary for vendors of anti-virus products who need to obtain, as fast as possible, newly released malware samples to update their signature databases.

Searching for malicious web pages is a three-step process, in which URLs are first collected, then quickly inspected with fast filters, and finally examined in depth using specialized analyzers. More precisely, one has to first collect pointers to web pages (URLs) that are live on the Internet. To collect URLs, one typically uses web crawlers, which are programs traversing the web in a systematic fashion. Starting from a set of initial pages, this program follows hyperlinks to find as many (different) pages as possible.

Given the set of web pages discovered by a crawler, the purpose of the second step is to prioritize these URLs for subsequent, detailed analysis. The number of pages discovered by a crawler might be too large to allow for in-depth analysis.

Thus, one requires a fast, but possibly imprecise, prefilter to quickly discard pages that are very likely to be legitimate. Such prefilters examine static properties of URLs, the HTML code, and JavaScript functions to compute a score that indicates the likelihood that a page is malicious. Based on these scores, pages can be ranked (sorted). For example, according to a report that describes the system deployed at Google [2], the company analyzes one billion URLs every day. To handle this volume, a prefilter is deployed that shrinks the number of pages to be inspected in depth by three orders of magnitude.

For the third step, we require detection systems that can determine with high accuracy whether a web page is malicious. To this end, researchers have introduced honeyclients. Some of these systems use static and/or dynamic analysis techniques to examine the HTML content of a page as well as its active elements, such as client-side scripting code (typically JavaScript scripts or Java applets). The idea is to look for signs of well-known exploits or anomalous activity associated with attacks [6]–[8]. Other detection systems look for changes to the persistent state of the operating system (such as additional files or processes) [2], [9], [10] once a page has been loaded. These changes often occur as a result of a successful exploit and the subsequent execution of a malware binary. Of course, detection systems are much more precise than prefilters, but they are also much slower (the analysis of a page can take seconds; prefilters can be several orders of magnitude faster).

The resources for identifying malicious pages are neither infinite nor free. Thus, it is essential to perform this search-and-analysis process in an efficient way so that one can find as many malicious pages as possible in a fixed amount of time.

In this paper, we propose an approach that improves the efficiency of the first step of the search process, augmenting it by opportunistically relying on the data that search engines collected with their crawlers. More precisely, we propose a system, called EVILSEED, which complements the (essentially random) web crawling approach with a *guided search* for malicious URLs. EVILSEED starts from a set of known pages that are involved in malicious activities. This set contains malicious pages that were directly set up by cybercriminals to host drive-by-download exploits or scam pages. The set also includes legitimate pages that were compromised and, as a result, unknowingly expose users to malicious code or redirect visitors to attack pages. In the next step, EVILSEED searches the web for pages that share certain similarities with the known malicious pages. We call this a "guided search" of the web, because it is guided by the current knowledge of known malicious web pages. Of course, these searches are not guaranteed to return only malicious pages. Thus, it is still necessary to analyze the search results with both prefilters and honeyclients. However, the key advantage of our approach is that a result of our guided search is *much more likely* to be malicious than a web page found by randomly crawling. Thus, given a fixed amount of resources, our approach allows us to find more malicious pages, and we do so quicker.

We also believe that EVILSEED would be beneficial to search engines. Although it is difficult to provide data supporting this claim, since the details of the infrastructure that search engines use to create their blacklists are confidential, we have observed several cases in which EVILSEED led our small cluster to detect malicious web pages faster then search engines. For example, in January 2010 EVILSEED identified a malware infection campaign with hundreds of URLs that, although inactive, can still be found by querying Google and Bing for "calendar about pregnancy". We observed these two search engines incrementally, and slowly, blacklisting URLs belonging to the campaign over the next ten months, until the campaign was eventually shut down. By applying EVILSEED, the lifetime of this campaign would have been much reduced, protecting the users of these engines.

Our approach is built upon two key insights. The first one is that there are similarities between malicious pages on the web. The reason is that adversaries make use of automation to manage their campaigns. For example, cybercriminals search the web for patterns associated with vulnerable web applications that can be exploited by injecting malicious code into their pages [11], [12]. Also, cybercriminals use exploit toolkits to create attack pages [13], and they often link many compromised pages to a single, malicious site to simplify management.

The second insight is that there are datasets and tools available that make it easier to find malicious URLs. Most notably, search engines (such as Google and Bing) have indexed a large portion of the web, and they make significant investments into their crawler infrastructures to keep their view of the web up-to-date. We leverage this infrastructure, as well as other datasets such as passive DNS feeds, for our guided search process. Note that we do not propose to completely replace traditional web crawlers when searching for malicious web pages. Guided search is a process that allows us to automatically and efficiently find malicious pages that are similar to ones that have already been identified. Random crawling is still necessary and useful to find new pages that are different than those already known.

In the past, cybercriminals have used search engines to find vulnerable web sites [11], [12], [14], [15]. In particular, cybercriminals perform manually-crafted search queries to find pages that contain certain keywords that indicate the presence of vulnerabilities. Previous work has extensively studied malicious search engine queries. More precisely, researchers have examined search query logs for entries that were likely to be issued by cybercriminals. Such query strings can then be extracted (and generalized) as signatures to block bad queries [11], [12]. In [12], the authors also examine the results that a search engine returns for malicious queries to identify potentially vulnerable hosts.

In this paper, we do not attempt to identify or study manually-crafted, malicious queries used by cybercriminals. Instead, we propose a *search process* that allows us to find malicious pages that are similar to those previously identified. As part of this process, we do submit queries to search engines. However, these queries are automatically generated, based on the analysis of different data sources such as a corpus of known, malicious pages and DNS data feeds (but no access to search engine logs). Moreover, our searches target a broader range of pages than those targeted by attackers who use search engines for locating vulnerable web sites. In particular, we are not only interested in finding compromised (vulnerable), legitimate pages, but also malicious pages that are directly set up by attackers.

The main contributions of this paper are the following:

- We developed a novel approach to guide the identification of malicious web pages starting from an initial set of known, malicious web pages.
- We described several novel techniques for the extraction of features from malicious web pages that can be used in queries submitted to existing search engines to identify more malicious web pages.
- We implemented our techniques in a tool and we evaluated it on a large set of malicious web pages, demonstrating the approach is effective and improves the state of the art.

II. System Overview

In this section, we describe in detail the goals of our work, and we provide a brief overview of the overall approach and the components of our system.

A. System Goal

As mentioned previously, searching for malicious pages on the web is a three-step process: Crawl to collect URLs, apply a fast prefilter to discard obviously benign pages, and use a precise-but-slow oracle to classify the remaining pages. In this paper, our goal is to improve the efficiency of the web crawling phase. More precisely, we have developed techniques that allow us to gather URLs that have a higher "toxicity" than the URLs that can be discovered through (random) crawling. With toxicity, we refer to the percentage of URLs in a set that point to malicious web pages.

Our techniques are based on the idea of searching for pages that are similar to ones that are known to be malicious. Intuitively, rather than randomly searching for malicious pages, EVILSEED focuses its searches "near" known malicious pages. More precisely, EVILSEED implements different techniques to extract from a page features that characterize its malicious nature; pages with similar values for such features (the "neighborhood" of a page) are also likely to be malicious. Then, by using the features extracted from an evil seed and by leveraging existing search engines, EVILSEED guides its search to the neighborhood around known malicious pages.

We use the notion of "maliciousness" in a broad sense, and our general techniques are independent of the exact type of threat that a particular page constitutes. For the current version of EVILSEED, we consider as malicious a page that, when visited, leads to the execution of a drive-by download exploit (possibly after redirecting the user). In addition, we consider a page to be malicious when it attempts to trick a user into installing a fake anti-virus program.

In this paper, we use the term web page and URL synonymously. That is, the actual inputs to and the "unit of analysis" of our system are URLs. In most cases, a page is uniquely identified by its corresponding URL. However, there are cases in which attackers create many URLs that all point to the same, underlying malicious page. In this situation, we would count each URL as a different, malicious page.

B. System Architecture

The general architecture of EVILSEED is shown in Figure 1. The core of our system is a set of gadgets. These gadgets consume a feed of web pages that have been previously identified as malicious (as well as other data feeds, such as domain registration feeds). Based on their input, the gadgets generate queries to search engines. The results returned by the search engines are then forwarded to an (existing) analysis infrastructure.

The key idea of our approach is that we can leverage the infrastructure of search engines and the data that they have collected. To this end, we query search engines in such a way that the URLs that they return have a higher probability of being malicious than a random page on the web (or the probability that can be achieved by directly crawling the web). Of course, the main challenge is to formulate the search queries such that the results indeed have a high probability of pointing to malicious pages. We assume that we have at our disposal an oracle (and optionally, a prefilter) that can be used to analyze a web page and determine whether it is malicious or not. Of course, the type of oracle depends on the precise notion of maliciousness that is used. We discuss possible oracles below; an example of an oracle that can detect drive-by download exploits would be a cluster of honeyclients.

In the following paragraphs, we discuss the components of our system in more detail:

Seed. The (evil) seed is a set of pages that have been previously found to be malicious. These pages form the input to gadgets. Of course, whenever gadgets discover new pages that the oracle confirms to be malicious, they can be added to the set of seed pages. One can distinguish two main types of pages in the seed. First, there are pages that were directly set up by cybercriminals. Typically, these are pages that directly contain scripting (mostly JavaScript) code that launches exploits, or links to malware binaries, such as fake AV programs. A previous paper refers to such pages as *malware distribution pages* [2]. The second type of pages is not malicious *per se*. Instead, they are legitimate pages that have been compromised.

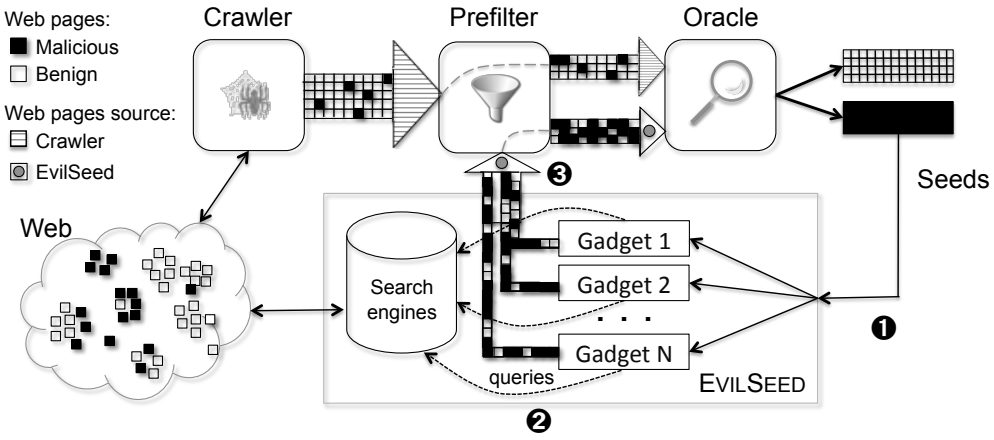

Figure 1. EVILSEED overview.

In most cases, such legitimate pages do not host any malicious code themselves. Instead, they only include a small piece of HTML or JavaScript that redirects the user to malware distribution sites. Such compromised, legitimate pages are called *landing pages* in [2].

Gadgets. Gadgets form the heart of EVILSEED. The purpose of a gadget is to find candidate pages (URLs) that are likely malicious based on the pages contained in the seed.

While different gadgets implement different techniques to achieve this goal, they all follow a common approach. More precisely, each gadget extracts certain information from the pages in the seed (❶ in Figure 1), such as content that is shared among these pages and links that are related to them.

In the next step, the information extracted from the seed pages is distilled into queries that are sent to search engines (❷). These queries can be simple searches for words or terms that appear on pages indexed by a search engine. Queries can also leverage advanced features of search engines, such as link information or restricted searches. Of course, gadgets need to consider the parts of a page that are indexed by a search engine. While it might be desirable to search for fragments of scripts that appear on malicious pages, such content is typically not indexed (or, at least, not made available through the public interfaces).

Once a query is issued, the gadgets simply collect the list of URLs that the search engines return (❸), and they forward them to the oracle (or, possibly, a prefilter) for further analysis. The gadgets that we use in EVILSEED are presented in detail in Section III.

Oracle. In the current implementation of EVILSEED, the oracle consists of three components: Google's Safe Browsing blacklist [16], Wepawet [6], and a custom-built tool to detect sites that host fake AV tools. We do not use a prefilter to analyze the URLs that EVILSEED produces, but this is certainly a possibility (as depicted in Figure 1). While a prefilter would reduce the number of pages that the oracle needs to inspect, it has no influence on the "toxicity" (fraction of malicious pages) of the URLs that the gadgets produce.

Google creates and makes available a "constantly updated blacklist of suspected phishing and malware pages." This blacklist, publicly accessible through the Safe Browsing API, is compiled by using a multi-stage process that analyzes more than one billion pages daily. At the core of the analysis infrastructure is a farm of (high interaction) honeyclients, which are particularly suitable to identify drive-by download attacks with a very low false positive rate.

Wepawet is a client honeypot that uses an instrumented browser to capture the execution of JavaScript code on web pages. Based on the recorded execution traces, the system uses anomaly-based techniques to detect drive-by download attacks. Wepawet is able to achieve high detection rates with almost no false positives [6].

As the third component, we use a custom-built detector for pages that host fake anti-virus software. This detector is a classifier that uses simple content analysis techniques (a combination of signatures and term frequency analysis) to determine if a web page misinforms users about the security of their computers and deceives them into downloading rogue security software.

III. Gadgets

The goal of a gadget is to leverage a set of known, malicious web pages to guide the search for additional, malicious content on the web. All gadgets perform the same processing steps. First, they *analyze the input seed* to identify similarities, that is, shared characteristics or properties that can be found among seed pages. Second, they *expand* the initial seed by querying one or more external search engines to identify other pages that share similar characteristics.

The assumption underlying all our gadgets is that malicious web pages often share a set of common characteristics, which are determined by the modus operandi and tools available to

Gadget	Expansion	Inputs
Links	Link topology	Seed URLs, Search Engines
Content dorks	Content similarity	Seed pages source, Search Engines
SEO	Link topology, Content similarity	Seed URLs, Search Engines
Domain registrations	Bulk registrations	Seed URLs, Domain registrations
DNS queries	Link topology	Seed URLs, DNS trace, Search Engine

Table I
GADGETS USED BY EVILSEED.

cybercriminals, the side-effects of the attacks they run, or the tell-tale signs of the vulnerabilities that enable their attacks.

We have implemented five gadgets (see Table I): The *links gadget* leverages the web topology (web graph) to find pages that link to malicious resources; the *content dorks gadget* aims at identifying vulnerable and exploited web applications; the *SEO gadget* analyzes seed pages that belong to blackhat Search Engine Optimization campaigns; the *domain registrations gadget* identifies suspicious sequences of domain registrations; and the *DNS queries gadget* analyzes traces of DNS requests to locate pages that lead to a malicious domain. We will now describe each gadget in detail.

A. Links Gadget

This gadget is designed to locate "malware hubs" on the web. Malware hubs are pages that contain links to several malicious URLs.[1] In our experience, hubs can be grouped in two categories: vulnerable sites that have been infected multiple times (this is typical, for example, of unmaintained web applications), and pages that catalog (and link to) web malware (this is the case of certain malware discussion forums, such as `malwareurl.com`).

This gadget leverages the observation that links contained on malware hubs are likely to be malicious and, thus, represent valuable candidate URLs.

Seed. The seed analyzed by the links gadget consists of all the URLs of known malicious pages.

Expansion. The gadget searches for malware hubs that link to pages in the input seed. More precisely, the gadget issues queries using the `link` operator, e.g., `link:<MALICIOUS URL>`. We sent these queries to three different search engines: Google, Bing, and Yacy. We used multiple search engines to distribute the load of our queries over multiple sites, and to increase the diversity of returned result sets. The gadget retrieves the URLs returned for the search engine queries and visits the corresponding page. For each visited page, the gadget extracts the URLs of all outgoing links. These URLs are then submitted to our oracle.

[1]We observe that links to known-malicious pages was a feature used in [2] and [17]. Our gadget reconstructs the linking structure from search engine results, rather than directly building the web graph by having access to the raw crawling data of the search engine.

B. Content Dorks Gadget

An effective technique to find vulnerable web sites is to query a popular search engine with a *Google dork*. This term delineates a set of carefully chosen keywords and operators tailored to retrieve links to vulnerable web pages. For example, the query "`index of /etc/`" will cause the search engine to locate web sites that share their configuration and list of users through their Apache web server. Likewise, a query for "powered by PhpBB 2.0.15" will return web sites that are using an older version of a popular bulletin board software with known vulnerabilities. The term Google dork was coined by Johnny Long, originally indicating "inept or foolish people as revealed by Google." A number of such dorks have been manually identified, and they are available in books [18], online databases [19], and penetration testing tools [20]. Recent research [12] has also found evidence of the large-scale use of Google dorks in the wild.

Painstakingly-assembled lists of manually identified Google dorks may be useful to find malicious web sites. However, many of the dorks lose their value over time. The reason is that application vulnerabilities are patched and the targeted applications are replaced. Thus, we propose a gadget that can *automate* the generation of relevant Google dorks.

While the underlying idea of the content dork gadget is known (and used by cybercriminals to find vulnerable sites), the novelty of this gadget is the ability to identify suitable dorks automatically. This has two advantages. First, our system produces a broad range of dorks that cover the long tail of less-popular but vulnerable web applications. Second, the system can quickly react to a wave of attacks that exploit a previously-unknown vulnerability. As soon as some malicious pages that exploit this vulnerability are included into the seed, our system can automatically extract content dorks to find other, similar sites that fell victim to the same exploit.

Seed. As discussed in Section II-B, our initial dataset of malicious pages can be divided into malware distribution pages and landing pages. As a starting point for this gadget, we are interested in landing pages only, which are originally benign but vulnerable pages that have been compromised by an attacker, as opposed to pages directly created by an attacker (e.g., pages generated with an exploit kit). The reason for focusing on landing pages is that they contain much more indexable content than malware distribution pages, and they remain online longer. Moreover, we expect that legitimate sites that were exploited because of vulnerabilities in a common, underlying content management system share characteristic strings that can be identified (similar in spirit to the "original" Google dorks). To distinguish between compromised landing pages and malware distribution pages, we use a simple, two step classification process: First, we discard pages that are no longer active. The assumption is that compromised pages, whether they

are cleaned or remain infected, will usually remain available over longer periods of time. Malware distribution pages, on the other hand, typically have a short lifetime. Likewise, we discard pages that, although apparently still active, are in fact *parking pages*, that is, pages set up by hosting providers in lieu of the removed malicious page. In the second step, we evaluate a number of HTML features that are indicative of compromised pages. For these features, we leverage previous work on static web page filters [21], [22]. Examples of the features that we use to identify a compromised page are the occurrence of script code after an HTML closing tag, the number of hidden iframes, and the presence of links to known advertising domains (as malware distribution pages typically don't have advertising).

Expansion. The queries generated by this gadget consist of n-grams of words that are extracted from the indexable content of landing pages in our seed. To generate these n-grams, we use two different techniques: *term extraction* and *n-gram selection*.

The term extraction process derives, from the content of a page, those terms that best summarize the topics of this page. This analysis typically leverages techniques from the information retrieval and natural language processing fields. We extract significant terms from a landing page in our seed by using Yahoo's Term Extraction API [23]. As an example, using it on CNN.com at the time of writing, term extraction yields *Eurozone recession*, *gay wedding*, *Facebook attack*, *graphic content*.

Cybercriminals are known to leverage popular topics to lure victims into visiting pages under their control. For example, they place popular terms onto pages, hoping to drive search engine traffic to the corresponding URLs. Term extraction allows us to identify and extract these topics automatically, as new trends are observed: In our experiments, we have seen terms that reflect topics such as smartphones, e-book readers, TV series, pharmaceutical products, and adult content. We use as content dorks all terms that are returned by the Term Extraction API.

The *n-gram selection* process extracts all sequences (of length n) of words from a landing page. Then, it ranks all n-grams according to their likelihood of occurring in a malicious page compared to their likelihood of appearing in a benign page. The intuition is that n-grams that appear much more frequently in malicious pages than in benign ones are a good indication for the maliciousness of the page. Therefore, in addition to the seed of compromised pages, we built a dataset of benign web pages by crawling the top 16,000 most popular domains, according to their Alexa ranking (we assume that these pages are legitimate). To select the most promising n-grams from a compromised page, we examine all n-grams that appear on the malicious seed pages. We discard all n-grams that are present more often in benign pages than in the malicious ones (based on relative frequencies). In the next step, we assign a score to each of the remaining n-grams. This score is equal to the difference between the relative frequency of an n-gram in the malicious dataset and its relative frequency in the benign dataset. We consider n-grams that vary from length $n = 2$ to $n = 5$. Once we have computed the score for each n-gram, we select as content dorks the top 10 n-grams.

All content dorks (those extracted by the n-gram selection and the term extraction processes) are submitted as queries to three search engines (Google, Bing, Yacy). We then retrieve the URLs (links) from the results and submit them to the oracle.

C. Search Engine Optimization Gadget

Cybercriminals are able to exploit and take control of large numbers of vulnerable web sites. However, most of these web sites are likely to be part of the "long tail" of the web, and are visited by a very small numbers of users. Therefore, drive-by attacks injected into these websites would only reach a small pool of potential victims. To reach more users, cybercriminals use a variety of techniques to drive traffic to the malicious pages under their control. Unsurprisingly, these include the use of blackhat Search Engine Optimization (SEO) techniques to increase the ranking of malicious pages in search engine results for popular search terms.

According to a report by Trend Micro [24], SEO techniques have been exploited to spread malware since at least November 2007. More recently, attackers started deploying *SEO kits* that are able to automatically generate "rings" of pages optimized for currently popular search topics [25]–[28]. An SEO kit is typically a PHP script installed on a compromised web server. It includes functionality to fetch currently popular search terms from sources such as Google trends or Twitter trends. Furthermore, given a search topic, it makes use of a search engine to obtain text and images relevant to the topic, and it automatically generates web pages from this raw material. SEO kits also use several techniques to increase ranking, such as generating a large number of *doorway* pages linking to an optimized page, or using link exchanges between pages on different exploited sites. Another common characteristic of SEO kits is the use of *semantic cloaking* [29], [30]. That is, the exploited web sites respond with completely different content depending on the source of a request. Based on information such as the source IP address of the request and the `User-Agent` and `Referer` HTTP headers, attackers may attempt to provide a benign, SEO optimized page to search engine crawlers, a malicious page to end users, and a benign page to security researchers and analysis tools.

Several characteristics of large-scale, automated blackhat SEO campaigns make it possible for EVILSEED to discover other malicious SEO-optimized web sites starting from a URL that is part of such a campaign.

- Attackers host many different web pages, optimized for different search terms, on each web site in a campaign.
- Attackers host pages optimized for the same search terms on different web sites in a campaign.
- Pages in a campaign often link to each other.

These characteristics are a consequence of the fact that attackers want to make the best use of the available resources (a finite number of compromised web sites) to obtain a high search ranking for a wide variety of popular search terms.

Seed. As a starting point, the EVILSEED SEO gadget needs at least one malicious URL that is part of an SEO campaign. To identify likely blackhat SEO URLs, we use a simple cloaking detection heuristic. The idea is that a malicious page that provides different content to search engine bots and to end users is likely trying to manipulate search engine rankings.

Detecting semantic cloaking and distinguishing it from *syntactic cloaking* [31], as well as from normal variation of web content over time, is itself not a trivial problem. In this work, we use a simple cloaking detection heuristic that is limited to detecting redirection-based cloaking, because this form of cloaking is frequently used in blackhat SEO campaigns. Research into more sophisticated cloaking detection techniques [29], [30] is orthogonal to this work, and it would be straightforward to integrate such techniques into our prototype.

To detect cloaking, we visit a URL three times, providing different values for the `User-Agent` and `Referer` HTTP headers. First, we visit the URL as a user that has reached the site from Google. For this, we use the `User-Agent` of a popular browser, and a `Referer` header corresponding to a Google search query. Then, we visit the URL with no `Referer` header, emulating a user who directly typed the URL into her browser's address bar. Finally, we visit the URL emulating the behavior of Google's indexing bot. For this, we rely on GoogleBot's well-known User-Agent string. We follow HTTP redirections and consider the final "landing" domain from which the browser receives a successful (2xx) HTTP response. We detect cloaking if, during the three visits to a URL, we observe two or more different landing domains.

Furthermore, since blackhat SEO campaigns are known to target popular searches obtained from Google and Twitter trends, we extend the seed for this gadget by fetching Google and Twitter trends, querying Google for the resulting topics, and checking the returned URLs with our cloaking detection heuristic.

Expansion. Once we have identified at least one cloaked, malicious URL, we can attempt to locate other URLs in the same blackhat SEO campaign. For this, we use a number of techniques to identify additional candidate URLs. First of all, for each domain hosting a cloaked, malicious web site, we perform a Google query using the `site:` modifier to locate other pages on that domain. We fetch the query results and add them to the set of candidate URLs. This allows us to find other malicious pages on the same site that may have been optimized for different search terms. Furthermore, we follow links in the malicious cloaked pages. Specifically, we consider the version of the page that was served to us when we surfed the page emulating the Google indexing bot, and consider any external links contained in that page. We add the target URLs of these links to the candidate URLs.

Finally, we try to identify the search terms for which a page has been optimized. For this, we do not take into account the content we downloaded from the URL. Instead, we rely exclusively on information provided by Google about that URL as part of its search results. The reason is that we cannot be certain that we have been able to deceive the malicious website into sending us the content intended for search engines (some SEO kits include a hard-coded list of search engine bot IP addresses, and perform IP-based cloaking [25]). Using Google's results allows us to sidestep this problem, and obtain a subset of the page as it was delivered to Google's bot. The information provided by Google consists of the page title and a number of snippets of the page's content. This information is typically shared between many pages in a SEO campaign [28].

The SEO gadget simply extracts the title of the page, and queries Google for the set of words in the title. The results of the query are then fetched and added to the candidate URLs.

D. Domain Registrations Gadget

Blacklists are one of the most widespread techniques to protect users against web malware. In a domain-based blacklist, a domain is added to the list as soon as it is discovered to host malicious content. As a countermeasure, cybercriminals are known to serve their content from short-lived domains, frequently switching domains to maximize the time they remain unlisted. To run an efficient business in this arms race, cybercriminals are likely to automate the domain generation and registration process. This automation leaves behind some artifacts, giving us some leverage to identify these domains. More precisely, we assume that registrations that are close in time to the registration of known malicious domains are also likely to be malicious.

Seed. The seed used by the Domain Registrations Gadget consists of all the domains that are known to host malicious pages, and domain registration records which are freely available online.

Expansion. This gadget extracts the domain of a malicious seed URL, and flags the domains that have been registered before and after as suspicious.

These domains are then used to create URLs that are scheduled for analysis. The URL creation consists of taking the closest known malicious registration (for which we know the corresponding malicious URL), and replacing the domain with the suspicious domain that we have just flagged. For example, if `a.com` has been registered just moments before `b.com`, and we know that `http://a.com/exploit` is malicious, the gadget will submit for analysis `http://b.com/exploit`.

Note that this gadget does not leverage search engines to find additional malicious URLs. However, it still follows EVILSEED's guided search philosophy.

E. DNS Queries Gadget

The DNS queries gadget analyzes recursive DNS (RDNS) traces. The goal is to identify the domain names of compromised landing pages that are likely to lead to malicious pages.

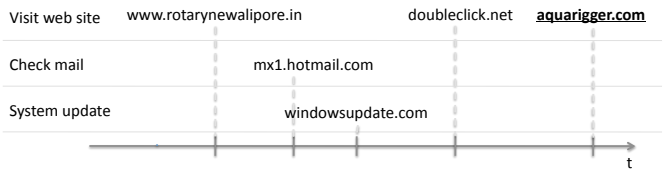

Figure 2. Example DNS trace.

The gadget works in two steps: First, it leverages temporal relationships in a DNS trace to identify domains that are likely connected to malicious domains. More precisely, the gadget checks for queries for domain D_L "shortly before" a query for domain D_P, which is known to host malicious pages. Then, the gadget identifies pages (URLs) on D_L that may redirect their visitors to an evil page P hosted on D_P.

Seed. The seed used by the DNS queries gadget consists of all the domains that are known to host malicious pages.

Expansion. This gadget's expansion relies on the fact that, often, a large number of infected pages contain links to a single, malicious page, and that DNS traces (partially) expose these connections. In practice, we passively monitor the recursive DNS traffic generated by a large user base. This traffic can be collected, for example, by deploying a sensor in front of the RDNS server of a network. We assume that a network user, during her regular Internet activity, will browse to a compromised landing page L (hosted on D_L) that redirects to P, one of the URLs that are part of our evil seed. This browsing activity appears in the DNS trace as a sequence of DNS requests issued by the same client, querying, in a short period of time, for D_L *first*, and for D_P *later*. Note that while we expect that queries for D_L and D_P be close in time, in general, they will not be consecutive. This is because of concurrent network activity on the user's machine (e.g., software updates and email checks) and because of queries needed to fetch additional resources (e.g., external images and scripts) included by L before the redirection to P. For example, consider the DNS trace in Figure 2. The trace shows the DNS requests performed by one client over time. In this interval, the client visits a web site (www.rotarynewalipore.in), checks email on hotmail.com, and performs a system update (windowsupdate.com). The trace ends with the request of a known malicious domain (aquarigger.com), which is reached during the web site visit. The gadget scans the DNS trace for queries about domains in the evil seed. Whenever one is found, the gadget considers as candidate domains those domains that were queried by the same client in the preceding N seconds (in the current implementation, this window is empirically set to four seconds). In our example, the candidate domains are www.rotarynewalipore.in, mx1.hotmail.com, windowsupdate.com, and doubleclick.net.

Given a set of candidate domains, the gadget generates URLs from these domains. Of all the possible pages on a candidate domain, the gadget chooses those that are more likely to be accessed by a web visitor. We consider three cases. First, a user is likely to access a domain by visiting its home page, for example, by typing its URL directly in the browser's address bar. Second, users may visit pages returned in the result set of a search query. This set includes highly-ranked pages from that domain. Finally, a user is likely to follow links to pages on the candidate domain contained in popular (i.e., high-ranking) pages on different domains. (Notice that these linked-to pages do not necessarily possess a high ranking, due to mechanisms such as the `nofollow` attribute value designed to mitigate the impact of web spam [32].)

Therefore, for each domain D_i in the considered time window, the gadget includes, as candidate URLs, the home page of the domain and the URLs obtained by querying search engines for:

- `site:`D_i — top-ranking pages from the candidate domain D_i,
- `"http://`D_i`/" -inurl:`D_i — URLs on D_i found on different sites, including potentially spammed pages.

In our example, the analysis of www.rotarynewalipore.in shows that it is infected with malicious JavaScript code that leads to a drive-by-download web page on aquarigger.com.

Discussion and analysis. The DNS queries gadget has a few limitations. First, the trace analysis is simplified by ignoring the effects of caching and pinning of DNS responses performed by modern browsers. In other words, we assume that whenever a user browses from L to P, her browser resolves D_L and D_P, irrespective of previous visits to these domains. If this assumption does not hold, our gadget may be unable to locate landing pages redirecting to the malicious domain. For example, this occurs if the user first visits a page on D_L that does not trigger a redirection to the malicious domain, and then browses to L after an amount of time that is larger than the window considered by the gadget, but shorter than the DNS cache expiration interval used by the browser. However, we argue that this scenario is unlikely to occur frequently in practice. The reason is that attackers have incentives to maximize the traffic to the malicious domain and, therefore, to force a redirection to the malicious domain on the first visit of a page on D_L.

Similarly, if the redirection from a landing page to the malicious page is triggered after a time delay that is longer than the window considered by our gadget, the gadget will not be able to identify D_L. Also in this case, we argue that attackers are not likely to introduce such delays. This is because the delay may allow a significant number of users to escape the attack, for example, by simply navigating away from the landing page.

Finally, we also assume that traffic to malicious pages is generated via web-based techniques: clearly, this gadget would not be effective if links to malicious domain are circulated via other medium, e.g., by spam emails or instant messages.

We also notice that, like the links gadget, this gadget relies on the network topology to locate additional malicious web content. However, unlike the links gadget, the DNS queries

gadget does not obtain topology information from search engines. This allows the DNS queries gadget to leverage network topology information that may not be visible to search engines. This is a significant advantage because attackers actively try to prevent search engines from indexing (and possibly detecting) malicious content (e.g., by using semantic cloaking). Furthermore, this provides more up-to-date results (the gadget does not need to wait for the search engine crawler to have updated topological data). Of course, only connections exposed by the visits of actual users are visible to the gadget.

IV. Evaluation

In this section, we experimentally validate our initial hypothesis that the guided search approach used in EVILSEED is effective at locating malicious pages, and it does so in an efficient manner. We use two key metrics to establish the effectiveness of our system: *toxicity* and *expansion*.

The *toxicity* is the fraction of the URLs submitted to the oracles that are, in fact, malicious. Higher values of toxicity imply that the resources needed to analyze a page (e.g., the oracle's CPU and time) are used more efficiently.

Seed *expansion* is the average number of new malicious URLs that EVILSEED finds for each seed. Intuitively, the seed expansion is a measure of the return on the investment provided by running a certain searching approach: A higher seed expansion indicates that for each malicious seed URL a larger number of malicious URLs are found. For example, if after a day of traditional crawling 100 malicious URLs are found and, when these URLs are fed to EVILSEED, the system identifies 150 additional malicious pages, then the expansion of the system is 1.5.

There is a trade-off between toxicity and seed expansion: a higher expansion can be obtained at the cost of a lower toxicity (e.g., starting a traditional crawl, which does not require any seed). Since EVILSEED is designed to achieve a higher efficiency in locating malware, our main focus will be obtaining a high toxicity.

A. Effectiveness of EVILSEED

EVILSEED is meant to complement a traditional crawler-based approach, so we ran our system in parallel with a traditional crawler. In this setting, both the crawler and EVILSEED submit URLs to the oracles, as shown in Figure 1. Whenever a malicious URL is found by the crawler, it is included in the seed used by EVILSEED. The oracles that we use for evaluation are Wepawet, Google Safe Browsing, and a custom fake AV detector (see Section II). We ran this setup for 25 days, using all gadgets except the DNS queries (for which we did not have access to a trace dataset during this time) and the domain registration gadget (which was developed at a later stage). To assess the performance of EVILSEED, we evaluated our approach against two reference approaches for finding malicious URLs: a traditional crawler coupled with a fast prefilter, and random searches.

Crawler & prefilter: In this approach, we use a traditional web crawler to traverse the web and use a fast prefilter to select the pages to analyze with our oracles. The crawler is seeded with "trending" terms from Google Trends and Twitter terms, which are known to be often abused by attackers. The fast filter uses static characteristics of a page to quickly identify and discard pages that are likely to be benign, in a vein similar to Prophiler [21]. This setup allows us to compare EVILSEED with a traditional, crawler-based infrastructure for finding malicious web pages.

Web searches: In this approach, we send a variety of queries to search engines, and submit the results to our oracles. The rationale for comparing with web searches is to contrast the toxicity of the URLs found by EVILSEED with the intrinsic toxicity of search engine results. The toxicity of search engine results is affected by two competing forces: criminals using search engine optimization, and search engines pruning malicious results.

To generate web queries, we use the following strategies:

- Random alphabetic phrases, composed of 1 to 5 words, of length from 3 to 10 characters (e.g., "asdf qwerou");
- Random phrases with words taken from the English dictionary, from 1 to 5 words (e.g., "happy cat");
- Trending topics taken from Twitter and Google Hot Trends (e.g., "black friday 2011");
- Manually-generated Google dorks, taken from an online repository (e.g., "allinurl:forcedownload.php?file=", which locates vulnerable WordPress sites) [19].

Table II shows an overview of the results of the experiments. EVILSEED submitted 226,140 URLs to the oracles, of which 3,036 were found to be malicious, for a toxicity of 1.34%. The Crawler & prefilter setup discovered 604 malicious URLs (these are the URLs we use as seeds for EVILSEED). The stream of URLs generated by this approach had a toxicity of 0.14%, which is an order of magnitude less than EVILSEED. The web search setup found 219 malicious URLs for a total toxicity of 0.34% (Table II shows results for individual search strategies). Interestingly, the toxicity of search engine results is 2–3 times higher than a random crawl, likely indicating that search engines are heavily targeted by attackers.

The seed expansion (the ratio between the number of malicious pages that were identified and the initial evil seed) shows that the gadgets can generate a stream of malicious URLs of a magnitude comparable to (or larger than) the number of seeds. The only exception is the SEO gadget, which during the time of this experiment, found only 16 malicious URLs (using 604 evil seeds). We explain this result by noticing that, during this time, the gadget was not able to identify a live SEO campaign, which it requires to produce its feed. We will show later the performance of this gadget when SEO campaigns are active.

Overall, this experiment shows that EVILSEED clearly outperforms in toxicity both crawling (1.34% vs. 0.14%) and web searching (1.34% vs. 0.34%). These results also show

Source	Seed	URLs Visited	Queries	URLs analyzed	Malicious URLs	Toxicity	Expansion
EVILSEED							
Links	604	82,391	5,470	71,272	1,097	1.53%	1.81
SEO	604	312	-	312	16	5.12%	0.02
Content Dorks (keywords)	604	13,896	1,432	13,896	477	3.43%	0.78
Content Dorks (ngrams)	604	140,660	7,277	140,660	1,446	1.02%	2.39
Total		237,259	14,179	226,140	3,036	**1.34%**	**5.02**
Crawler w/ Prefilter		3,057,697	-	437,251	604	**0.14%**	
Web Search							
Random Strings		24,137	504	24,137	68	**0.28%**	
Random Dictionary		27,242	599	27,242	107	**0.39%**	
Trending Topics		8,051	370	8,051	27	**0.33%**	
Manual Dorks		4,506	87	4,506	17	**0.37%**	

Source	Seed	Domains visited	Queries	Domains analyzed	Malicious domains	Toxicity	Expansion
EVILSEED							
Links	98	9,882	5,470	7,664	107	1.39%	1.09
SEO	98	7	-	7	5	71.42%	0.07
Content Dorks (keywords)	98	3,245	1,432	3,245	119	3.66%	1.22
Content Dorks (ngrams)	98	33,510	7,277	33,510	263	0.78%	2.68
Total		46,644	14,179	44,426	494	**1.12%**	**5.04**
Crawler w/ Prefilter		407,692	-	53,445	98	**0.18%**	
Web Search							
Random Strings		4,227	504	4,227	16	**0.37%**	
Random Dictionary		9,285	599	9,285	35	**0.37%**	
Trending Topics		1,768	370	1,768	8	**0.45%**	
Manual Dorks		3,032	87	3,032	13	**0.42%**	

Table II
EVILSEED EVALUATION: URLS AND DOMAINS.

that adding even relatively few new pages to the set of evil seeds enables EVILSEED to locate significant numbers of additional malicious pages. More precisely, during these 25 days, EVILSEED discovered over five times more malicious URLs than the crawler (the expansion is 5.2), visiting 92% less pages.

What is the cost of querying search engines? Our gadgets query search engines to identify pages that are likely malicious. Clearly, if EVILSEED required disproportionately many queries to perform its guided search, then its advantages (increased toxicity and expansion) would be outweighed by the cost of querying search engines. This is not the case with the gadgets we have implemented. Table II reports the number of queries executed by each gadget: as it can be seen, the number of search engine queries generated by EVILSEED scales linearly with the number of malicious URLs discovered: approximately 1 every 4 queries leads EVILSEED to such an URL.

Does EVILSEED *find malicious URLs on different domains?* A search procedure that finds malicious content on different domains is more valuable than one that only discovers it on a few domains: different malicious domains may be indicative of different attack campaigns, and more domains can be added to domain blacklists. Table II shows the results of our evaluation at the domain level (subdomains are not considered as different domains). We observe that, on average, the crawler finds 6.16 malicious pages per domain. These results show that EVILSEED maintains the same domain coverage as the crawler, finding an average of 6.14 malicious pages per domain.

Does EVILSEED *sustain its production of malicious URLs over time?* Figure 3 shows the number of malicious pages found by EVILSEED each day over a period of 25 days. We observe that the number of detections per day is consistent over time.

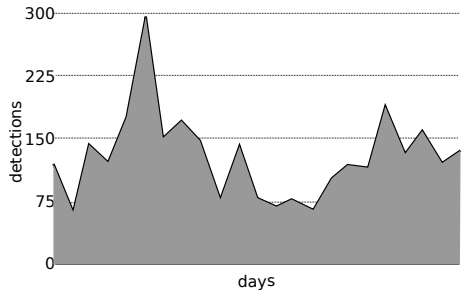

Figure 3. EVILSEED detections over time.

B. SEO Gadget

As we have seen, the SEO gadget performed poorly during our initial experiment. This is because its seed, comprising the malicious URLs found by the crawler during that time, did not contain URLs belonging to a live SEO campaign. We speculate that this may be due to a change in the Twitter API that occurred during our experiment, which may have reduced the maliciousness of seeds based on Twitter trending topics.

To better evaluate the SEO gadget, we ran a successive experiment, in which we fetched hourly the top trends for Twitter and Google Hot Trends, searched for them on Google,

and analyzed the results with our cloaking detection heuristic (Section III-C). Out of the 110,894 URLs that we collected, 248 were detected to be cloaking (at the domain level, 46 out of 54,944). We then seeded the SEO gadget with these cloaking URLs. The results of this experiment are shown in Table III. Note that, to use the oracle's resources efficiently, the SEO gadget submits only one URL per each distinct domain.

The high detection rate on these URLs indicates that our cloaking detection technique is effective and that redirection-based cloaking provides a strong indication that a website is malicious. Note that the toxicity does not take into account the number of visited pages, since the most expensive operation in our system is the oracle analysis. When taking those into account, the ratio of the malicious pages found over the visited pages is 0.93%, which is two orders of magnitude higher than the crawler (0.019%).

Expansion	Total	Cloaked	Cloaked %
URLs:			
Links	537,729	158,536	29.5%
Site Query	134,364	105,443	78.5%
Term Query	525,172	141,765	27.2%
Total	1,219,090	319,949	26.2%
Domains:			
Links	52,053	10,028	19.2%
Site Query	10,304	7,970	77.3%
Term Query	177,542	15,230	8.6%
Total	239,899	33,228	13.8%

Table IV
CLOAKING DETECTION RESULTS

Table IV provides the results of cloaking detection on all the URLs tested by the SEO gadget. Overall, 26.2% of the URLs tested were in fact cloaked, and 7.3% of the domains tested hosted at least one cloaked URL. The percentage is lower for domains because, as we observed in Section III-C, each domain in a blackhat SEO campaign hosts large numbers of pages optimized for different search terms.

The tables also show the performance of the various expansion strategies used by the SEO gadget. Only 4% of the total URLs were produced by more than one of the three expansion strategies. Site queries, where we search for other websites hosted on the same domain as a seed URL, had the best performance, with 77.3% cloaked URLs. However, such queries clearly cannot expand to previously unknown malicious domains. The remaining strategies have similar performance at around 30% cloaked URLs, but the term queries strategy provides larger numbers of distinct domains. For comparison, the last row shows the cloaking detection results on URLs obtained by querying for Google and Twitter trends topics: Only 0.2% of the tested URLs were cloaked.

C. Content Dorks Gadget

We found that the most influential factor for the success of a content dorks gadget is n, the length of the n-gram. In particular, we found that the toxicity for the results of queries ranged from 1.21% for 2-grams to 5.83% for 5-grams. In hindsight, this is not too surprising. The reason is that smaller n-grams are typically found on a larger number of pages. Thus, when we assume that a malware distribution campaign has compromised a certain number of pages, shorter n-grams means that more pages will compete for the top spots in the search engine rankings.

Although longer n-grams are typically better, it is interesting to observe that the first ten most-successful dorks in term of toxicity were five 2-grams and five 3-grams, with the first two positions taken by "Centralized Admission Policy" and "calendar about pregnancy." The former trigram led to the discovery of many compromised pages hosted at a Pakistani university. The latter trigram can be found in the payload of a large injection campaign, with more than 750 compromised domains still listed on Bing at the time of writing, including `about.com` (65^{th} in the Alexa top domains) and several `.edu` domains. The injected page is used for search engine optimization; it contains thousands of terms that are likely to be ranked highly in search engines. When visited, the page redirects to a malicious URL. The third most successful n-gram was "Gyj (SND) {var" with 19 malicious pages detected. This is part of a JavaScript snippet (`function Gyj(SND){var`) that is used in an injection campaign. Google reports 304 active pages, of which about half have been blacklisted by Google Safe Browsing.

D. Links Gadget

We have observed three main categories of sites that have been used by the Links Gadget to locate malicious content.

The first one is unmaintained web sites. As an example, the URL http://www.irkutsk.org/cgiapps/guestbook/guestbook.html leads to a guestbook that has been active for almost ten years. This guestbook contains 286 spam comments, 12 of which have been found malicious by our oracles.

The second category is domains that publish black-lists of malicious domains (e.g., http://www.mwis.ru and http://urlquery.net). The advantage here is that this gadget can automatically discover and parse these sources.

The last category is domains that list some additional information about a domain. For example, http://calimartec.info lists, for a given domain, the domains that are co-located on the same IP, the domain hosted in the same subnet, and the domains that have a similar spelling.

E. Domain Registrations Gadget

Starting from the 13^{th} of November 2010, we collected a year's worth of domain registrations for the top-level domains .COM., .NET., .ORG., .INFO. and .US (a total of $16,663,616$ domains). During this time period, the Domain Registrations Gadget identified malicious URLs on $10,435$ domains using $1,002$ domains as seeds. We can now revisit our hypothesis that malicious domains are registered close in time to each other. Let us now consider the probability P that, given that we

	Seeds	Visited	Cloaked	Analyzed	Malicious	Toxicity	Malicious / Visited	Expansion
URLs	248	1,219,090	319,949	12,063	11,384	94.37%	0.93%	45.90
Domains	46	177,542	15,230	9.270	8,763	94.53%	4.93%	190.5

Table III
CLOAKING DETECTION RESULTS

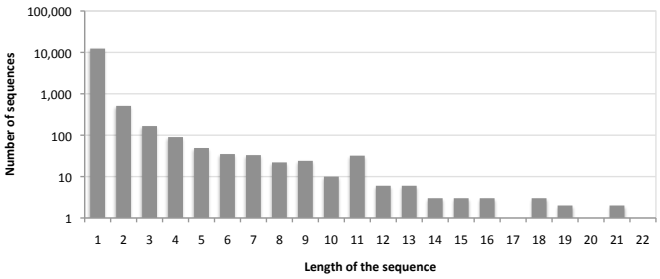

Figure 4. Consecutive registrations of malicious domains.

know that a particular domain registration is malicious, at least one of the registrations that come immediately before or after it (in chronological order) is also malicious. Given the scarcity of malicious domains (0.1% of the total domains), if the events "domain is malicious" and "domain has been registered before/after a known malicious domain" were independent, P would be 0.19%. The empirical evidence of the collected data, instead, shows that these events are correlated, as P is empirically equal to 7.51% (see Figure 4). Therefore, we can conclude that the domains that have been registered immediately before and after a known malicious domain are much more likely (more than 35 times more likely) to also serve malicious content.

F. DNS Queries Gadget

To test the DNS queries gadget, an Internet Service Provider (ISP) gave us access to a DNS trace collected from its network during 43 days in February and March 2011. The trace contained 377,472,280 queries sent by approximately 30,000 clients to one nameserver. The trace was made available to us only at the end of this collection period, which, unfortunately, introduced a significant delay between the collection of data and the time when the gadget was run.

As our seed, we considered 115 known, malicious domains that were queried at least once in the trace. From these, our gadget generated 4,820 candidate URLs (on 2,473 domains), of which 171 URLs (on 62 domains) were determined to be malicious by our oracles. Of the 115 malicious domains that were used by our gadget, only 25 were indeed "effective," i.e., led to the discovery of a malicious URL. The most effective domain guided the gadget to locate 46 malicious URLs on 16 different servers; 21 domains led to the discovery of multiple malicious URLs.

We speculate that the delay between recording the query events and analyzing them with our gadget may explain why no malicious URLs were identified for 90 of the malicious domains queried in the trace: During this time, compromised pages may have been cleaned up or removed from search engine indexes. For the future, we intend to run the gadget in parallel to the DNS data collection.

V. Discussion and Limitations

Security analysis. Our results have shown that EVILSEED is effective at finding malicious content on the web starting from an initial seed of malicious pages. However, an attacker might try to prevent EVILSEED from finding and detecting malicious pages. The most radical approach would be to make sure that these pages are not indexed by search engines in the first place. This can easily be achieved by an attacker who has full control of an exploited website and can restrict bot access using `robots.txt`. On the other hand, this approach may not be possible for cases where the attacker is able to inject malicious content into a page, but does not have any additional privileges on the exploited site. A more significant problem with this evasion approach, however, is that malicious web sites often receive the bulk of their visitors from search engines, so de-listing them would vastly reduce the number of potential victims (this is particularly evident in the case of pages of SEO campaigns).

Attackers could also try to perform evasion attacks against the detection techniques employed by our oracle (Wepawet, our custom fake AV page detector, and the Safe Browsing system). This problem is orthogonal to the techniques presented in this paper. In addition, our tool can be combined with any other oracle for the detection of malicious web pages, thus increasing the difficulty of evasion attempts.

Seed quality. The effectiveness of our gadgets is dependent on the quality and diversity of the malicious seed that they use as input. In the worst case, if the seed completely lacked a particular class of malicious pages, our gadgets would not be able to automatically generate search queries leading to such pages. To minimize this risk, we rely on a collection of malicious pages that is as large as possible, and we refresh it frequently. We can obtain input pages from external sources, such as Wepawet and public repositories of web malware, e.g., the malwaredomainlist.com forum, which are constantly updated. Wepawet currently flags several thousand new pages per month, and the malwaredomainlist.com database has increased by about 500–4,000 entries every month since 2010.

Results over time. To be useful in practice, EVILSEED needs to be able to provide a constant stream of high-quality candidate URLs, rather than exhausting its effect after one or few runs.

The number of candidate URLs identified by a single query is limited by the maximum number of results that a search

engine will return for this query. To obtain more results, we could borrow a technique used by cybercriminals, and refine the queries we issue with additional keywords, to elicit different result sets from a search engine. For instance, results from John et al. [12] show that attackers add restrictions such as `site:.com` to their dorks, for different top level domains. Moreover, as our results from running EVILSEED in online mode have demonstrated (see Section IV-A), even a small number of new seeds can be expanded into a significant number of additional, malicious pages. Thus, EVILSEED can be run continuously alongside a traditional detection system (such as Wepawet), discovering a significant number of additional, malicious URLs for each new seed that Wepawet finds.

Performance/scalability. The bottleneck of EVILSEED is the cost of performing in-depth analysis with an oracle. For example, Wepawet can process only about 100K URLs per day (one URL takes about one minute). Currently, EVILSEED runs on two ordinary servers: one runs the crawler, the other the gadgets. Each search engine query returns tens of results, and one machine can easily perform 10K searches per day without getting rate-limited. This allows us to gather 100K URLs per search engine. A single web crawling host can retrieve millions of pages. This underlines the critical importance of feeding the oracle (bottleneck) with pages of high toxicity.

Deployment. Search engines could deploy EVILSEED. This might diminish its effectiveness. However, it also mean that the vectors that EVILSEED targets were mitigated; we consider this a success. Moreover, researchers with limited resources can still deploy EVILSEED, harnessing search engines that filter results more naively. A good candidate for this is Yacy, which is a distributed crawling effort that strives to build a search engine with no filters. Also, search engines might be interested in offering to researchers an access to unfiltered results.

So far, we have presented EVILSEED from the point of view of these researchers, because we belong to this group ourselves: in fact, EVILSEED was born as a solution to the problems we were facing in collecting malware samples. However, we believe that EVILSEED can be beneficial also to search engines. EVILSEED "guided search" will not provide search engines any performance gain in the crawling phase, as they would still incur in the full crawling cost to build their index: in fact, they provide the crawling infrastructure that the small players deploying EVILSEED opportunistically leverage. Instead, EVILSEED's main contribution in the eye of search engines are the new features we propose: by adding them to their prefilters, search engines can select URLs candidates that need in-depth analysis more timely, keeping more of their users safe from malware. Also, a page selected by EVILSEED is more likely to be malicious, thus improving the prefilter efficiency. This is because the features identified by EVILSEED and the prefilter are considering different aspects: EVILSEED looks at the "neighborhood" of the candidate page, while the prefilter looks at particular features of the page (e.g., javascript obfuscation). Many details about the search engines' filtering techniques are confidential, so we cannot provide conclusive proof that these features are not already in use. However, we have observed several cases that indicate otherwise. For example, from January to November 2011 we have observed a large-scale injection campaign on legitimate web sites that was distributing malware: to this date, it can be still be found querying Google and Bing for "calendar about pregnancy". This n-gram has been generated by our EVILSEED deployment in January, when our crawler randomly visited a URL that has been infected. Over the next ten months, both Bing and Google have been slowly blacklisting more and more results of this query, until the campaign was eventually shut down. This incremental blacklisting is indicative that search engines do not implement EVILSEED: otherwise, upon marking one of the infected URLs as malicious, all the remaining URLs would be quickly sent to in-depth analysis, and blacklisted.

Evasion. Cybercriminals could try to detect our visits. To mitigate this, visits from crawler and gadgets come from a "fresh" browser (no history and previous state). Cybercriminals could still track our system's IPs: using a large, dynamic pool of IPs mitigates this problem. It is improbable that cybercriminals specializing in SEO campaigns and injections will try to evade EVILSEED by diminishing their footprint in search engines, because this will reflect negatively in the traffic on their sites, and ultimately in their income.

Gadget selection. A final question that we address is whether there are particular limitations on the gadgets that can be employed in our system. As we have seen, the only requirements on gadgets are that they are capable of identifying similarities among the pages of the evil seed, and of querying search engines for pages with the same properties. We have presented the implementation of a diverse set of gadgets, based on the analysis of the textual content of web pages, on their linking relationship, and on their behavior (cloaking). In our experience, it was also easy to extend the existing system by "plugging in" an additional gadget.

VI. Related Work

Finding malicious content on the web is a process that requires two main components: a detection procedure (or oracle), which, given a web page, decides whether or not it is malicious, and a searching procedure, which locates web pages to submit to the oracle.

Oracles. The problem of designing effective oracles for the detection of malicious web pages (mostly pages that perform drive-by download attacks [2], but also fake anti-virus pages [4] and Flash-based malicious advertisements [33]) has received considerable attention. Two main approaches have been proposed: (i) using high-interaction honeyclients to detect unexpected changes in the underlying system (e.g., new files or running processes) indicating a successful compromise [2], [9], [10], [34], [35]; and (ii) using low-interaction honeyclients or regular browsers instrumented with specialized, light-

weight detectors [6]–[8], [36]–[41]. Additional efforts have focused on preventing unconsented content execution, which is the ultimate goal of drive-by download attacks [42], and on identifying the common pieces of infrastructure (central servers in malware distribution networks) employed by large numbers of attacks [43]. In this work, we use Wepawet [6] (a low-interaction honeyclient) and the Google Safe Browsing blacklist [2] (a blacklist produced using high-interaction honeyclients) as our oracle. We use these tools as black boxes, and, therefore, it would be possible to use in EVILSEED any other available tool that is able to provide an evaluation of the malicious nature of a web page. Since honeyclients require a significant amount of resources to analyze a web page, a prefiltering step is often applied to a candidate page, so that likely-benign pages are quickly discarded [2], [21], [22], [44].

Our work is orthogonal to prefilters: We are not concerned with filtering out from a pre-existing dataset "uninteresting pages," as prefilters do. Instead, we describe ways to effectively build datasets that are more likely to contain "interesting" (i.e., malicious) web pages.

Searching for candidate pages. The second component of a system that detects malicious web content is one that gathers the web pages that are passed to the oracle. Previous work has focused on web crawling, identifying common features of malicious pages, and monitoring and learning from the attackers' behavior. Hereinafter, we discuss each of these areas.

Traditionally, researchers have relied on large web crawls to collect web pages [2], [17]. These large crawls are effective, because they provide a "complete" view of the web: Provos et al. report a 5% detection rate, after a prefiltering step in which billions of pages are quickly discarded. The downside is that these massive crawls require an extensive infrastructure, which is available only to a few organizations.

Alternatively, web crawls can be guided to favor the visit of parts of the web that, according to a number of heuristics, are more likely to contain malicious content [21], [35]. These smaller, targeted crawls are feasible also for smaller players, since they do not require building and maintaining a large infrastructure. However, they yield lower detection rates. For example, Moshchuk et al. report a 0.4% detection rate [35].

Our approach hits a sweet spot between these two crawling alternatives. In fact, EVILSEED does not require the extensive infrastructure used for large crawls (the entire system runs on one off-the-shelf server-class machine). Instead, it mines malicious samples to generate search engine queries that point to URLs with high toxicity, thereby leveraging the work that search engines have already performed.

An alternative technique to searching for malicious content is based on the observation that malicious resources are often similar to each other, for example, because they are created using the same attack toolkits. Previous efforts have focused specifically on domain names, and have identified a number of features that are characteristics of the malicious domains controlled by cybercriminals. These characteristics can be used to identify additional (previously uncategorized) domains that are likely to be malicious [5], [45]–[47]. One important difference to our approach is that we want to identify legitimate but compromised pages, rather than domains that are directly under control of the cybercriminals. In this context, the involved domain names are not decided by the cybercriminals.

A final approach to identifying potentially malicious sites consists of replicating the techniques cybercriminals use to search for vulnerable web sites. The insight, in this case, is that vulnerable web sites that are searched for by cybercriminals are likely to get compromised and become malicious [11], [12], [14]. We approach the problem from a different angle. Instead of identifying actual (manually-crafted) queries from existing search engine logs, we analyze known, malicious pages to automatically extract the searches that can be used to discover both compromised landing pages and malware distribution sites.

VII. Conclusions

As malicious activity on the web continues to increase, it is critical to improve the defenses at our disposal. An important component of our defense is the ability to identify as many malicious web pages on the Internet as possible. This is a daunting task that requires a substantial amount of resources.

In this paper, we propose a novel approach whose goal is to improve the effectiveness of the search process for malicious web pages. We leverage a seed of known, malicious web pages and extract characterizing similarities that these pages share. Then, we use the infrastructure of search engines and the data that they have collected to quickly identify other pages that show the same characteristics and, thus, are also likely malicious. We have implemented this approach in a tool, called EVILSEED, and validated it on large-scale datasets. Our results show that EVILSEED can retrieve a set of candidate web pages that contains a much higher percentage of malicious web pages, when compared to random crawling (and even to results returned for manually-crafted, malicious queries). Therefore, by using EVILSEED, it is possible to improve the effectiveness of the malicious page discovery process.

Acknowledgments

This research draws on data provided by the University Research Program for Google Search, a service provided by Google to promote a greater common understanding of the web. This work was supported by the ONR under grant N000140911042 and by the National Science Foundation (NSF) under grants CNS-0845559 and CNS-0905537, and by the European Commission through project 257007 (SysSec).

References

[1] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, "The Ghost in the Browser: Analysis of Web-based Malware," in *USENIX Workshop on Hot Topics in Understanding Botnet*, 2007.

[2] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose, "All Your iFrames Point to Us," in *USENIX Security Symposium*,

2008.

[3] M. Polychronakis, P. Mavrommatis, and N. Provos, "Ghost Turns Zombie: Exploring the Life Cycle of Web-based Malware," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[4] M. A. Rajab, L. Ballard, P. Mavrommatis, N. Provos, and X. Zhao, "The Nocebo Effect on the Web: An Analysis of Fake Anti-Virus Distribution," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2010.

[5] M. Cova, C. Leita, O. Thonnard, A. Keromytis, and M. Dacier, "An Analysis of Rogue AV Campaigns," in *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2010.

[6] M. Cova, C. Kruegel, and G. Vigna, "Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code," in *International World Wide Web Conference (WWW)*, 2010.

[7] J. Nazario, "PhoneyC: A Virtual Client Honeypot," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[8] K. Rieck, T. Krueger, and A. Dewald, "Cujo: Efficient Detection and Prevention of Drive-by-Download Attacks," in *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[9] 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 *Symposium on Network and Distributed System Security (NDSS)*, 2006.

[10] C. Seifert and R. Steenson, "Capture-HPC," http://goo.gl/vSdds.

[11] N. Provos, J. McClain, and K. Wang, "Search Worms," in *ACM Workshop on Recurring Malcode (WORM)*, 2006.

[12] J. John, F. Yu, Y. Xie, M. Abadi, and A. Krishnamurthy, "Searching the Searchers with SearchAudit," in *USENIX Security Symposium*, 2010.

[13] IBM, "Mid-Year Trend and Risk Report," Tech. Rep., 2010.

[14] T. Moore and R. Clayton, "Evil Searching: Compromise and Recompromise of Internet Hosts for Phishing," in *International Conference on Financial Cryptography and Data Security*, 2008.

[15] S. Small, J. Mason, F. Monrose, N. Provos, and A. Stubblefield, "To Catch A Predator: A Natural Language Approach for Eliciting Malicious Payloads," in *USENIX Security Symposium*, 2008.

[16] Google, "Safe Browsing API," http://goo.gl/vIYfk, 2011.

[17] J. Stokes, R. Andersen, C. Seifert, and K. Chellapilla, "WebCop: Locating Neighborhoods of Malware on the Web," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2010.

[18] J. Long, E. Skoudis, and A. v. Eijkelenborg, *Google Hacking for Penetration Testers*, 2004.

[19] J. Long, "Google Hacking Database," http://goo.gl/qmPA8.

[20] Cult of the Dead Cow, "Goolag Scanner," http://goo.gl/lBK1o.

[21] D. Canali, M. Cova, C. Kruegel, and G. Vigna, "A fast filter for the large-scale detection of malicious web pages," in *International World Wide Web Conference (WWW)*, 2011.

[22] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "Zozzle: Low-overhead Mostly Static JavaScript Malware Detection," Microsoft Research, Tech. Rep., 2010.

[23] Yahoo, "Yahoo Term Extraction API," http://goo.gl/wGacG.

[24] R. Flores, "How Blackhat SEO Became Big," Trend Micro, Tech. Rep., 2010.

[25] F. Howard and O. Komili, "Poisoned search results: How hackers have automated search engine poisoning attacks to distribute malware." SophosLabs, Tech. Rep., 2010.

[26] B. Googins, "Black Hat SEO Demystified: Abusing Google Trends to Serve Malware," 2010.

[27] B. Zdrnja, "Down the RogueAV and Blackhat SEO rabbit hole," http://goo.gl/5nDuK, 2010.

[28] D. Wang, S. Savage, and G. Voelker, "Cloak and dagger: dynamics of web search cloaking," in *18th ACM conference on Computer and communications security*. ACM, 2011.

[29] B. Wu and B. D. Davison, "Detecting semantic cloaking on the web," 2006.

[30] K. Chellapilla and D. M. Chickering, "Improving cloaking detection using search query popularity and monetizability," in *2nd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2006.

[31] B. Wu and B. D. Davison, "Cloaking and redirection: A preliminary study," in *Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005.

[32] M. Cutts and J. Shellen, "Preventing comment spam," http://goo.gl/rA9mZ, 2005.

[33] S. Ford, M. Cova, C. Kruegel, and G. Vigna, "Analyzing and Detecting Malicious Flash Advertisements," in *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[34] A. Moshchuk, T. Bragin, D. Deville, S. Gribble, and H. Levy, "SpyProxy: Execution-based Detection of Malicious Web Content," in *USENIX Security Symposium*, 2007.

[35] A. Moshchuk, T. Bragin, S. Gribble, and H. Levy, "A Crawler-based Study of Spyware in the Web," in *Symposium on Network and Distributed System Security (NDSS)*, 2006.

[36] B. Feinstein and D. Peck, "Caffeine Monkey: Automated Collection, Detection and Analysis of Malicious JavaScript," in *Black Hat Security Conference*, 2007.

[37] P. Likarish, E. Jung, and I. Jo, "Obfuscated Malicious Javascript Detection using Classification Techniques," in *Conference on Malicious and Unwanted Software (Malware)*, 2009.

[38] P. Ratanaworabhan, B. Livshits, and B. Zorn, "NOZZLE: A Defense Against Heap-spraying Code Injection Attacks," in *USENIX Security Symposium*, 2009.

[39] M. Egele, P. Wurzinger, C. Kruegel, and E. Kirda, "Defending Browsers against Drive-by Downloads: Mitigating Heap-spraying Code Injection Attacks," in *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2009.

[40] B. Hartstein, "Jsunpack: A Solution to Decode JavaScript Exploits as they Rapidly Evolve," in *ShmooCon Conference*, 2009.

[41] S. Chenette, "The Ultimate Deobfuscator," in *ToorCon*, 2008.

[42] L. Lu, V. Yegneswaran, P. Porras, and W. Lee, "BLADE: An Attack-Agnostic Approach for Preventing Drive-By Malware Infections," in *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[43] J. Zhang, C. Seifert, J. Stokes, and W. Lee, "ARROW: Generating Signatures to Detect Drive-By Downloads," in *International World Wide Web Conference (WWW)*, 2011.

[44] C. Seifert, P. Komisarczuk, and I. Welch, "Identification of Malicious Web Pages with Static Heuristics," in *Austalasian Telecommunication Networks and Applications Conference*, 2008.

[45] S. Yadav, A. K. Reddy, A. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in *Conference on Internet Measurement (IMC)*, 2010.

[46] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis," in *Symposium on Network and Distributed System Security (NDSS)*, 2011.

[47] M. Felegyhazi, C. Kreibich, and V. Paxson, "On the Potential of Proactive Domain Blacklisting," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2010.