# Shady Paths: Leveraging Surfing Crowds to Detect Malicious Web Pages

Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna
University of California, Santa Barbara
{gianluca, chris, vigna}@cs.ucsb.edu

## ABSTRACT

The web is one of the most popular vectors to spread malware. Attackers lure victims to visit compromised web pages or entice them to click on malicious links. These victims are redirected to sites that exploit their browsers or trick them into installing malicious software using social engineering.

In this paper, we tackle the problem of detecting malicious web pages from a novel angle. Instead of looking at particular features of a (malicious) web page, we analyze how a large and diverse set of web browsers reach these pages. That is, we use the browsers of a collection of web users to record their interactions with websites, as well as the redirections they go through to reach their final destinations. We then aggregate the different redirection chains that lead to a specific web page and analyze the characteristics of the resulting redirection graph. As we will show, these characteristics can be used to detect malicious pages.

We argue that our approach is less prone to evasion than previous systems, allows us to also detect scam pages that rely on social engineering rather than only those that exploit browser vulnerabilities, and can be implemented efficiently. We developed a system, called SpiderWeb, which implements our proposed approach. We show that this system works well in detecting web pages that deliver malware.

## Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Invasive software; H.3.5 [**Online Information Services**]: Web-based services

## General Terms

Security

## Keywords

HTTP redirections, detection, malware


*CCS'13,* November 4–8, 2013, Berlin, Germany.

http://dx.doi.org/10.1145/2508859.2516682.

## 1. INTRODUCTION

The world-wide web is one of the main vectors used to spread malware. The most common way of infecting victims with malware is to present them with a web page that performs a *drive-by download attack* [18]. In a drive-by download attack, the malicious web page tries to exploit a vulnerability in the victim's browser or in a browser plugin. If successful, the injected code instructs the victim's computer to download and install malicious software. In other cases, attackers rely on simple social engineering scams to infect victims. That is, the web page tricks users into downloading and installing malicious software without exploiting any client-side vulnerability; an example of this are *fake antivirus scams* [23]. In this paper, we refer to the pages used to deliver both types of attacks as *malicious web pages*.

Existing systems to identify malicious web pages fall into two main categories. The first category uses techniques to statically analyze web pages and embedded code (such as JavaScript or Flash), looking for features that are typical of malicious activity. Such features can be patterns in the Uniform Resource Locators (URLs) of web pages [14,31], the presence of words or other information that are associated with malicious content [16, 25, 26], or the presence of malicious JavaScript snippets that are typical for exploits [2, 3]. The second category leverages dynamic techniques; these approaches rely on visiting websites with an instrumented browser (often referred to as a *honeyclient*), and monitoring the activity of the machine for signs that are typical of a successful exploitation (e.g., the creation of a new process) [15, 20, 21, 28].

Although existing systems help in making users safer, they suffer from a number of limitations. First, attackers can obfuscate their pages to make detection harder and, in many cases, they are able to successfully evade feature-based systems [8]. Second, attackers frequently hide their exploits by leveraging a technique called *cloaking* [27, 30]. This technique works by fingerprinting the victim's web browser, and revealing the malicious content only in case the victim is using a specific version of the browser and a vulnerable plugin. Additionally, miscreants check the IP address of their visitors, and display a benign page to those IP addresses that belong to security companies or research institutions. Cloaking makes the work of dynamic analysis approaches much harder, because defenders need to run every combination of web browsers and plugins to ensure complete coverage (or use special techniques to work around this requirement [9]). Also, defenders require elaborate routing infrastructures to hide the source of their traffic. A third limitation is that many techniques, in particular honeyclients, can only detect malicious pages that exploit vulnerabilities — they cannot identify scams that target users with social engineering. Fourth, most dynamic techniques introduce a considerable amount of overhead into the instrumented browser, making these approaches difficult to deploy as online detectors.

In this paper, we propose a novel approach to detect malicious web pages. To this end, we leverage a large population of web surfers as diverse, distributed "sensors" that collect their interactions with the web pages that they visit. The interactions we are interested in are *HTTP redirection chains* that are followed by clients before reaching web pages. Redirections are widely used by attackers, because they make detection harder, and they give the flexibility of changing any intermediate step in the chain, without having to update all the entry points (i.e., the pages containing the links that start the redirection process). Redirection chains are an important part of the malware delivery infrastructure and, unlike the web page contents, they are not easy to obfuscate. Of course, HTTP redirections have many legitimate uses too. For example, advertising networks make extensive use of redirections to make sure that a suitable ad is displayed to the right user. The key difficulty is distinguishing between chains that correspond to malicious activity and those that are legitimate.

A few previous systems, SURF and WarningBird [10, 13], have incorporated redirection chain information into their detection approaches. To make the distinction between malicious and benign redirects, however, these systems all require additional information for their analysis. For example, WarningBird [10] analyzes features of the Twitter profile that posted a malicious URL. Unfortunately, attackers can easily modify the characteristics of those Twitter profiles, and make their detection ineffective.

In our approach, we aggregate the redirection chains from a collection of different users, and we build *redirection graphs*, where each redirection graph shows how a number of users have reached a specific target web page. By analyzing both the properties of the set of visitors of a page and its corresponding redirection graph, we can accurately distinguish between legitimate and malicious pages. In particular, we show that malicious web pages have redirection graphs with very different characteristics than the ones shown by legitimate ones. No information about the destination page is required. It is important to note that our approach can only detect malicious web pages that make use of redirection chains. However, this is not a severe limitation, because, as previous work pointed out, the use of redirection chains is pervasive in the malware delivery process [10, 13, 23, 29].

Our approach overcomes the limitations listed previously. We do not look at the content of web pages at all, and therefore, we are not affected by attempts to obfuscate the elements on a page or the scripts that are executed. Moreover, we do not check for exploits, so the system can cover a broader class of malicious pages. Also, the effect of cloaking is mitigated, because of the diverse population of web surfers involved in the data collection. As a last aspect, the data collection from the user browsers is lightweight, because data is collected as users browse the web, without any additional computation. Therefore, the performance penalty experienced by the users is negligible.

We implemented our approach in a tool called SpiderWeb. We ran our system on a dataset representative of the browsing history of a large population of users, and we show that SpiderWeb is able to detect malicious web pages that evade state-of-the-art systems.

In summary, this paper makes the following contributions:

- We propose a new way of detecting malicious web pages by collecting HTTP redirection data from a large and diverse collection of web users. The individual redirection chains are combined into redirection graphs that allow for the distinction between legitimate and malicious pages.
- We developed a system, called SpiderWeb, which is able to detect malicious web pages by looking at their redirection graphs, and we show that the system works well in practice, by running it on a dataset collected from the users of a popular anti-virus tool.
- We show that SpiderWeb is a useful tool to improve the detection provided by state-of-the-art systems, and that a redirection-graph based detection is hard to evade by attackers.

## 2. HTTP REDIRECTIONS

HTTP redirections are used by web developers to redirect the user to a different URL than the one originally accessed. Typically, this happens by sending a specific HTTP response to the user's browser, followed by the new page that the client should access. The URL of this new page is sent in the `location` header field of the HTTP response. When the browser receives a redirection response, it automatically redirects the user to the target page. Modern browsers transparently redirect their users when they receive a `301`, `302`, `303`, `304`, or `307` HTTP response [4]. Although each code has a different meaning, browsers behave in the same way upon receiving any of them, so we do not treat them differently either. Other types of redirections include HTML-based redirections (triggered by the `META` tag in a web page), JavaScript-based redirections, or Flash-based redirections. In our analysis, we do not differentiate between these classes of redirections.

### 2.1 HTTP Redirection Graphs

In this section, we describe how we build redirection graphs. We start with introducing HTTP redirection chains, as they represent the building blocks of redirection graphs.

**Formalizing HTTP redirection chains.** To define HTTP redirection chains, we start by defining two types of entities: *users* and *web pages*. A *user* is represented by a tuple

$$\mathbf{U} = < I, O, B, C >,$$

where $I$ is the IP address of the user, $O$ is a string that identifies her operating system, $B$ is a string that identifies her browser's configuration (the browser type, version, and the different plugins that are installed), and $C$ is the country corresponding to $I$. A *web page* is represented by a tuple

$$\mathbf{W} = < URL, D, L, TLD, Pg, Pa, I, C >,$$

where $URL$ is the URL to the web page, $D$ is the domain of the URL (including the Top-Level domain), $L$ is the length of the domain name (in characters), $TLD$ is the Top-Level domain of the URL (notice that we take care of special cases such as `.co.uk`), $Pg$ is the name of the page in the URL (more precisely, the sub-string of the URL that goes from the last "/" character to either the "?" character or the end of the URL, whichever comes first), $Pa$ is a set of strings that holds the names of the parameters in the URL, $I$ is the IP address that hosts the web page, and $C$ is the country in which the web page is hosted. For example, consider the URL `http://evil.com/a/search.php?q=foo&id=bar`. For this page, the full URL is $U$, $D$ is `evil.com`, $L$ is 4, $TLD$ is `.com`, $Pg$ is `search.php`, $Pa$ is `(q,id)`, and the IP address returned by the DNS server for `evil.com` is $I$. $C$ is calculated by performing geolocation on $I$.

Given our notion of users and web pages, we define a redirection chain as a tuple

$$\mathbf{RC} =< \mathbf{U}_{rc}, \mathbf{G}, \mathbf{Ref}, \mathbf{Lan}, \mathbf{Fin} >,$$

where $\mathbf{U}_{rc}$ is the user visiting the redirection chain and $\mathbf{G}$ is a graph $< \mathbf{V}, \mathbf{E} >$ where $\mathbf{V}$ is the set of web pages that are accessed in the redirection chain and $\mathbf{E}$ is a set of edges that specifies the order in which the web pages in $\mathbf{V}$ are accessed. Each edge is a tuple $< \mathbf{v}_1, \mathbf{v}_2 >$ containing the starting web page instance $\mathbf{v}_1$ and the destination web page instance $\mathbf{v}_2$. **Ref**, **Lan**, and **Fin**, are three "special" web pages in the redirection chain:

- **Ref** represents the *referrer*, which is the URL the user was coming from when she entered the redirection chain. The referrer could be the URL of the web page that contains the link that the user clicked on. Alternatively, it could be the URL of the web page that hosts an iframe. If the user inserted the URL manually into the browser address bar, the referrer will be empty. In this paper, we consider the referrer to be the first node of a redirection chain.
- **Lan** is the *landing page*, which is the web page on which the first redirection happens.
- **Fin** is the *final page*, which is the last web page in the redirection chain. The final page is the web page in **G** that has no outgoing edges, except for self-loops.

As we will see later, referrer, landing, and final page carry important information about the redirection chain that is useful to assess the maliciousness of a web page.

**Building a redirection graph.** As mentioned previously, we want to combine the redirection chains directed to the same destination into a redirection graph. A redirection graph is defined as:

$$\mathbf{RedGraph} =< \mathbf{R}, \mathbf{C}_{rg}, \mathbf{U}_{rg}, \mathbf{G}_{rg}, \mathbf{Ref}_{rg}, \mathbf{Lan}_{rg}, \mathbf{Fin}_{rg} >,$$

where $\mathbf{R}$ is the target node where all the chains in the graph end. In the basic case, $\mathbf{R}$ will be a single web page with a specific URL. However, in other cases, it might be useful to group together web pages that have different URLs, but share common characteristics. For example, in the case of a malicious campaign that infects websites using always the same web page name, the domain names of the infected pages will be different for different URLs. However, other parts of these URLs will be the same.

To support graphs that point to pages with different URLs, we allow groups in which $\mathbf{R}$ has certain fields that are marked with a *wildcard*. We consider a web page $\mathbf{W}$ to match $\mathbf{R}$ if, for each field of $\mathbf{R}$ that is not a wildcard, $\mathbf{W}$ and $\mathbf{R}$ contain the same value. In Section 3.2 we describe different ways for specifying $\mathbf{R}$ and, hence, for aggregating redirection chains into a redirection graph.

In our definition of redirection graphs, $\mathbf{C}_{rg}$ is a set that contains all the individual redirection chains that make up the graph **RedGraph**. $\mathbf{U}_{rg}$ is a set that contains all users in the redirection chains $\mathbf{C}_{rg}$. $\mathbf{G}_{rg} =< \mathbf{V}_{rg}, \mathbf{E}_{rg} >$ is a graph composed of the union of all the redirection chains in $\mathbf{C}_{rg}$. $\mathbf{Ref}_{rg}$, $\mathbf{Lan}_{rg}$, and $\mathbf{Fin}_{rg}$ are sets that contain all the referrers, the landing pages, and the final pages, respectively, of the redirection chains in $\mathbf{C}_{rg}$.

Given a set of redirection chains, and an instance of $\mathbf{R}$, we build the redirection graph **RedGraph** as follows:

**Step 1:** Initially, $\mathbf{U}_{rg}$, $\mathbf{G}_{rg}$, $\mathbf{R}_{rg}$, $\mathbf{Lan}_{rg}$, and $\mathbf{Fin}_{rg}$ are empty.

**Step 2:** We compare each redirection chain to $\mathbf{R}$. For each redirection chain, we compare its final page **Fin** to $\mathbf{R}$. If **Fin** matches $\mathbf{R}$, we will use the chain to build the redirection graph. Otherwise, the redirection chain is discarded.

**Step 3:** For each redirection chain that matches $\mathbf{R}$, we add the user $\mathbf{U}$ to $\mathbf{U}_{rg}$, the referrer **Ref** to $\mathbf{Ref}_{rg}$, the landing page **Lan** to $\mathbf{Lan}_{rg}$, and the final page **Fin** to $\mathbf{Fin}_{rg}$.

**Step 4:** For each redirection chain that matches $\mathbf{R}$, we compare the graph $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$ to the graph $\mathbf{G}_{rg} =< \mathbf{V}_{rg}, \mathbf{E}_{rg} >$. For each vertex $\mathbf{v}$ in $\mathbf{V}$, if $\mathbf{V}_{rg}$ does not have a vertex with the same exact values as $\mathbf{v}$, we add $\mathbf{v}$ to $\mathbf{V}_{rg}$. For each edge $\mathbf{e} =< \mathbf{v}_1, \mathbf{v}_2 >$ in $\mathbf{E}$, if $\mathbf{E}_{rg}$ does not contain an edge starting at $\mathbf{v}_1$ and ending at $\mathbf{v}_2$,we add the edge $\mathbf{e}$ to $\mathbf{E}_{rg}$.

## 2.2 Legitimate Uses of HTTP Redirections

HTTP redirections have many legitimate uses. The most straightforward one is to inform a visitor that a certain website has moved. For example, if a user accesses a web page under an old domain, she will be automatically redirected to the new one. As another example, HTTP redirections are used to perform logins on websites. If a user accesses a page without being logged in, she is redirected to a login page. After providing her credentials, the web page will check them and, in case they are correct, redirect the user back to the page she originally tried to access. Advertising networks (ad networks) are another example for legitimate uses of HTTP redirections. Typically, advertisements undergo a series of redirections, to allow the advertiser to find the ad that is the right match for a user [24].

Since redirection chains are so pervasive, we cannot simply flag all of them as malicious. In certain ways, malicious and benign redirection graphs look very similar. Fortunately, there are some features that are characteristic of malicious redirection graphs, and these are the ones that we can use for detection. We describe them in detail in the next section.

## 2.3 Malicious Redirection Graphs

To build our system, we are interested in the characteristics that distinguish malicious redirection graphs from the ones generated by legitimate uses.

**Uniform software configuration**. The exploits and/or malicious code used by cyber-criminals target a specific (set of) browser, plugin, or operating system versions. To avoid giving away their exploit code, or triggering content-aware detection systems when not needed, attackers typically use cloaking, and serve the malicious JavaScript only when the exploit has a chance of successfully compromising the user's machine. Cloaking can be performed by the malicious page itself, by fingerprinting the user's browser on the client side. Alternatively, it can be performed during an intermediate step of the redirection chain on the server side [29]. In the latter case, the victim will be redirected to the exploit page only in case her operating system and/or browser are vulnerable to the exploits that the malicious web page is serving.

From the point of view of our system, cloaking performed during the redirection process will lead to the situation where only victims with a uniform software configuration visit the malicious web page. Users who do not run a browser that matches a vulnerable software configuration will be redirected to a different page, possibly on a different domain. Note that modern exploit kits target multiple browsers and plugins versions. For this reason, in some cases, we do not observe a single browser version accessing a malicious web page, but a few of them. However, the diversity of browser configurations visiting the attack page will still be lower than the one visiting a popular legitimate web page.

**Cross-domain redirections**. Malicious redirection chains typically span over multiple domains. This happens mostly because it makes the malicious infrastructure resilient to take-down efforts. It also makes it modular, in the sense that a redirection step can be replaced without having to modify all the web pages in the redirection chain. On the other hand, benign redirection chains typically contain one or more pages on the same domain. An example of this behavior is the login process on a website; a user who is not authenticated is first redirected to a login page, and, after providing valid credentials, back to the original page.

**Hubs to aggregate and distribute traffic**. It is common for victims who click on malicious links to be first redirected to an aggregating page, which forwards them to one of several backend pages [28]. This behavior is similar to the one seen with legitimate advertising networks, where the user is first redirected to a central distribution point. From this point, the user will then be forwarded to one of the many different ads. Attackers use this type of infrastructure for robustness. By having a hub, it is easy for them to redirect their victims to a new, malicious web page in case the current one is taken down. Since the aggregation point does not serve the actual attack, it is more difficult to shut it down.

**Commodity exploit kits**. The underground economy has evolved to the point that exploit kits are sold to customers like any other software, and even offered as software as a service [5]. An exploit kit is a software package that takes care of exploiting the victims' browsers and infecting their computers with malware. After buying an exploit kit, a cybercriminal can customize it to suit his needs, and install it on a number of servers. Typically, this customization includes the selection of a specific template that is used to generate one-time URLs on the fly. Previous work showed that detecting such templates is important, and can be leveraged for malicious URL detection [31]. We assume that different groups of attackers might use the same URL-generation templates for each server they control.

**Geographically distributed victims**. Cyber-criminals often try to reach as many victims as possible, irrespective of the country they live in. As a result, the victims of a campaign will come from a diverse set of countries and regions. On the other hand, many legitimate sites have a less diverse user population, since a large fraction of their users come from the country in which the website is located.

# 3. SYSTEM DESIGN

We leveraged the observations described in the previous section to build a system, called SpiderWeb, which determines the maliciousness of a web page by looking at its redirection graph. Our approach operates in three steps: First, we need to obtain input data for SpiderWeb. To this end, we collect redirection chains from a large set of diverse web users. In the second step, SpiderWeb aggregates the redirection chains leading to the same (or similar) page(s) into redirection graphs. As a last step, SpiderWeb analyzes each redirection graph, and decides whether it is malicious.

## 3.1 Collecting Input Data

SpiderWeb requires a set of redirection chains as input. For this work, we obtained a dataset from a large anti-virus (AV) vendor. This dataset contains redirection chain information from a large number of their global users. The data was generated by the users who installed a browser security product, and voluntarily shared their data with the company for research purposes. This data was collected according to the company's privacy policy. In particular, the data did not contain any personal identifiable information: The source IP of the clients was replaced with a unique identifier and the country of origin. Moreover, the parameter values in URLs along the redirection chains were obfuscated. Thus, the data contained only the domain, path and parameter name information of the URLs, which is sufficient for our approach. We will discuss the dataset in more detail in Section 4.

Note that SpiderWeb is not limited to this particular dataset. Any collection of redirection data from a crowd of diverse users would represent a suitable input.

## 3.2 Creating Redirection Graphs

Given a set of redirection chains, the second step is to aggregate them into redirection graphs. To this end, the system leverages the algorithm described in Section 2.1.

A key challenge is to determine which pages should be considered "the same" by this algorithm. The most straightforward approach is to consider pages to be the same only when their URLs (domain, path, parameter names) match exactly. However, certain malware campaigns change their URLs frequently to avoid detection. This can be done by changing the domain or by varying the path and parameter values. If SpiderWeb would only aggregate redirection chains that lead to the same exact URL, it might miss campaigns that obfuscate their URLs. To overcome this problem, we propose five different ways to recognize variations of destination pages, where all chains that lead to one of these pages should be aggregated. In other words, there are five different ways to specify page similarity via $\mathbf{R}$.

**Domain + page name + parameters:** This grouping is the most specific one. Pages are considered the same only when all parts of the URL (domain, path and name of the page, and parameter names) match.

**Top-level domain + page name:** With this grouping, two pages will be considered the same when they share the same top-level domain and page name. Cyber-criminals frequently register their domains in bulk, using less popular and less vigilant domain registrars (e.g., `.co.cc` or `.am`) [7]. However, they use an automated setup for their exploit infrastructure, and hence, the page names remain unchanged.

**Top-level domain + page name + parameters:** This grouping is similar to the previous one, but also takes into

| | |
|---|---|
| Domain+Page+Parameters: | evil.com,index.php,id |
| TLD+Page: | com,index.php |
| TLD+Page+Parameters: | com,index.php,id |
| Domain length+TLD+Page+Param.: | 4,com,index.php,id |
| IP: | IP address for evil.com |

**Table 1: Different fields used for grouping for the URL `http://evil.com/dir/index.php?id=victim`.**

account the name of the parameters in the URL. This grouping works well for campaigns that use specific URL parameters. It is also useful to detect malicious campaigns in cases where the previous grouping would be too general (according to our popularity filter, described below).

**Domain name length + Top Level Domain + page name + parameters:** For this grouping, we consider the length of the domain name instead of the concrete name itself. This grouping is useful to detect campaigns that create domains based on a specific template, such as random strings of a specific length.

**IP address:** This grouping leverages the observation that even if attackers completely vary their URLs (and hence, could bypass the previous groupings), they only use a small set of servers to host their malicious web pages (typically, on bulletproof hosting providers). In such cases, the IP address grouping will allow SpiderWeb to aggregate all chains that end at the same IP address.

To see an example of the different parts of the URL that are considered for each grouping, consider Table 1. In addition to groupings, SpiderWeb leverages two thresholds during its operation. The first threshold discards those groupings that are too general, while the second one discards redirection graphs that have low complexity, and do not allow an accurate decision.

**Popularity filter.** Intuitively, certain ways of grouping similar pages might be too generic and result in the combination of chains that do not belong together. For example, grouping all chains that end in URLs that have *.com* as their top-level domain and *index.html* as their page name would not produce a meaningful redirection graph. The reason is that most of the landing pages in $\mathbf{Fin}_{rg}$ will not be controlled by the same entity, and, thus, will not share similar properties. To exclude unwanted redirection graphs from further analysis, we introduce a *popularity metric.* More precisely, for each redirection graph, we look at its redirection chains and extract all domains that belong to the chains' final pages. If any domain has been accessed more than a threshold $t_p$ times in the *entire* dataset, the graph is discarded. The intuition is that one (or more) domain(s) included in the graph are so popular that it is very unlikely that this graph captures truly malicious activity. In Section 4.2, we describe how we determined a suitable value for threshold $t_p$ for our dataset.

**Complexity filter.** On the other side of the spectrum (opposite of too generic graphs) are graphs that are too small. That is, these graphs contain too few chains to extract meaningful features. Typically, these graphs belong to legitimate sites where one landing page redirects to one specific destination. Intuitively, this makes sense: A malicious website is likely to be composed of redirections that start at many different web sites (e.g., infected web pages). For this reason, SpiderWeb discards any redirection graph that contains less than $t_c$ distinct redirection chains. Again, we discuss our choice for the value of $t_c$ in Section 4.2.

## 3.3 Classification Component

In the third step, SpiderWeb takes a redirection graph as input and produces a verdict on whether the final web page(s) of this graph are malicious or not. This step represents the core of our detection technique. We developed a set of 28 features that describe the characteristics of a redirection graph. These features fall into five main categories: *client features*, *referrer features*, *landing page features*, *final page features*, and *redirection graph features.* Their purpose is to capture the typical characteristics of malicious redirection graphs as reflected in the discussion in Section 2.3.

An overview of our features is given in Table 2. It can be seen that most features (21 out of 28) have not been used before. The reason is that many features rely on the fact that we collect data from a distributed crowd of diverse users. Specifically, these include *client features* and *redirection graph features.* In the following, we describe our features in more detail. Then, we discuss how the features are leveraged to build a classifier.

**Client features**. These features analyze characteristics of the user population $\mathbf{U}_{rg}$ that followed the redirection chains in the graph **RedGraph**. The *Country Diversity* feature is computed by counting the number of different countries for the users in $\mathbf{U}_{rg}$ and dividing this value by the size of $\mathbf{C}_{rg}$ (which is the number of redirection chains in **RedGraph**). As discussed previously, many legitimate web pages are visited by users in a small number of countries. The *Browser Diversity* and *Operating Systems Diversity* features reflect the diversity in the software configurations of the clients that access a certain web page. Similarly to the *Country Diversity* feature, these two features are computed by taking the number of distinct browsers and operating systems in $\mathbf{U}_{rg}$ and dividing it by the size of $\mathbf{C}_{rg}$. If all clients that visit a page share a similar software configuration, this is often the result of profiling their browsers to deliver effective exploits.

**Referrer features**. These features capture characteristics of the referrers that lead users into redirection chains. Referrers provide information about the locations of the links on which users click to reach the actual final web page, or information about the locations of iframes, in case iframes are used to trigger redirections.

The *Distinct Referrer URLs* counts the number of different referrer URLs in $\mathbf{Ref}_{rg}$, divided by the size of $\mathbf{C}_{rg}$. Malicious redirection chains are typically accessed by a smaller number of referrers than legitimate ones. For example, a malicious campaign that spreads through social networks will show only referrers from such sites (e.g., Twitter of Facebook), while a legitimate news site will be pointed to by a higher variety of web pages (e.g., blogs or other news sites). The *Referrer Country Ratio* feature is computed similarly to the *Country Diversity* described in the previous section. This ratio characterizes how geographically distributed the various starting points (referrers) that lead to a certain web page are. Certain types of legitimate web pages (e.g., news sites), are accessed mostly through links posted in specific countries. On the other hand, malicious web pages are often linked by websites that reside in different countries. The *Referrer Parameter Ratio* feature is computed by taking the number of distinct parameters $Pa$ in $\mathbf{Ref}_{rg}$, divided by the size of $\mathbf{C}_{rg}$. This feature provides information about the di-

| Feature Category | Feature Name | Feature Domain | Novel |
|---|---|---|---|
| *Client Features* | Country Diversity | Real | ✔ |
| | Browser Diversity | Real | ✔ |
| | Operating System Diversity | Real | ✔ |
| *Referrer Features* | Distinct Referrer URLs | Integer | ✔ |
| | Referrer Country Ratio | Real | ✔ |
| | Referrer Parameter Ratio | Real | ✔ |
| | Referrer Has Parameters | Real | ✔ |
| *Landing Page Features* | Distinct Landing Page URLs | Integer | [10] |
| | Landing Country Ratio | Real | ✔ |
| | Landing Parameter Ratio | Real | ✔ |
| | Landing Has Parameters | Real | ✔ |
| *Final Page Features* | Distinct Final Page URLs | Integer | [10] |
| | Final Parameter Ratio | Real | ✔ |
| | Top-Level Domain | String | [25] |
| | Page | String | [25] |
| | Domain is an IP | Boolean | [13] |
| *Redirection Graph Features* | Maximum Chain Length | Integer | [10, 13] |
| | Minimum Chain Length | Integer | [10, 13] |
| | Maximum Same Country of IP and Referrer | Boolean | ✔ |
| | Maximum Same Country Referrer and Landing Page | Boolean | ✔ |
| | Maximum Same Country Referrer and Final Page | Boolean | ✔ |
| | Minimum Same Country of IP and Referrer | Boolean | ✔ |
| | Minimum Same Country Referrer and Landing Page | Boolean | ✔ |
| | Minimum Same Country Referrer and Final Page | Boolean | ✔ |
| | Intra-domain Step | Boolean | ✔ |
| | Graph has Hub 30% | Boolean | ✔ |
| | Graph has Hub 80% | Boolean | ✔ |
| | Self-loop on the Final Page | Boolean | ✔ |

**Table 2: Features used by SpiderWeb to detect malicious domains. It can be seen that most features used by our system are novel. For those that have been used in the past, we included a reference to the works that used such features.**

versity of the pages that lead to the redirection chain. The *Referrer Has Parameters* feature measures how many of the web page entities in $\mathbf{Ref}_{rg}$ have a value for $Pa$, divided by the size of $\mathbf{C}_{rg}$. The last two features help in understanding how a web page is accessed. For example, in the case of a malicious web page that is promoted through Search Engine Optimization (SEO), one expects to see a high number of distinct search engine queries that lead to a malicious page. This is not the case for legitimate pages, which will be found with a limited number of search keywords.

**Landing page features**. Analyzing the landing pages of a redirection graph provides interesting insights into the diversity of the URLs that are used as landing pages. The four features in this category (*Distinct Landing Page URLs*, *Landing Country Ratio*, *Landing Parameter Ratio*, and *Landing Has Parameters*) are calculated in the same way as for the referrer features, by dividing the distinct value of URLs, countries, parameters, and the number of non-empty parameters in $\mathbf{Lan}_{rg}$ by the size of $\mathbf{C}_{rg}$. Legitimate and malicious landing pages differ in a similar way to the referrer.

**Final page features**. Similar to the referrer and the landing page analysis, examining the characteristics of the final pages helps us in distinguishing between legitimate and malicious redirection graphs.

The *Distinct Final Page URLs* feature value is calculated in the same way as the value for *Distinct Referrer URLs*. Having a large number of distinct URLs for the final page is suspicious – it indicates that attackers are generating one-time URLs. The *Final Parameter Ratio* is calculated in the same way as for the referrer and the landing pages. Based on our observations, benign redirection graphs exhibit a higher variety of GET parameters than malicious ones. One reason for this is that legitimate web pages often use single pages that provide multiple services, and each different service is selected by a set of specific GET parameters. The *Top-Level Domain* feature identifies the set of top-level domain(s). Some top-level domains are more likely to host malicious web pages than others, because of the cost of registering a new domain that contains such a top-level domain [12]. The *Page* feature represents the page names, if any (e.g., `index.html`). As explained previously, many exploit kits use recognizable URL patterns. The *Domain is an IP* feature indicates if, for any destination web page, the domain value is the same as the IP. Many malicious web pages are accessed through their IP addresses directly, as previous work noted [13].

**Redirection graph features**. These features characterize the redirection graph as a whole. In Section 4.3, we show that this category of features is critical for malicious web page detection, because it allows us to flag as malicious a large number of malicious web pages that would look legitimate otherwise.

The *Maximum* and *Minimum Chain Length* features capture the maximum and minimum number of redirection steps observed for a single redirection chain in $\mathbf{C}_{rg}$. Previous work showed that a long redirection chain can be a sign of malicious activity [10,13]. We then introduce six Boolean features that model the country distributions for important nodes that are involved in the redirection process. The calculation of these six features works as follows: For all redirection chains in $\mathbf{C}_{rg}$, we examine pairs of important elements $s_1$ and $s_2$. The elements $s_1$ and $s_2$ can be either the user and the referrer, the referrer and the landing page, or the referrer and the final page. For each pair $s_1$ and $s_2$, we check whether the corresponding entities are located in the same country. If this is true, we set the value $V$ for this chain to *true*, otherwise we set it to *false*. In the next step, for each of the three pairs, we compute the logic **AND** as well as the logic **OR** for the values $V$. We expect legitimate redirection chains to have more geographic consistency across the different steps of the chain (i.e., having multiple steps located in the same country). On the other hand, malicious redirection chains, which are often made by compromised servers, tend to have multiple steps located in different countries. A similar observation was made by Lu et al. [13]. Notice that, for the features in this category, we only considered the minimum and maximum values, and not the average. The reason for this is that the average values turned out not to be discriminative at all. We perform a more detailed analysis of the features that we decided to use in Section 4.3.

The *Intra-domain Step* feature captures whether, along any of the redirection chains in $\mathbf{C}_{rg}$, there is at least one step where a page redirects to another page on the same domain. If this holds, the feature is set to *true*, otherwise, it is set to *false*. Legitimate redirection chains often contain intra-domain steps, while malicious chains typically contain domains that are all different.

Two *Graph has Hub* features (30% and 80%) characterize the shape of the redirection graph. More precisely, they capture the fact that a hub is present in a redirection graph. To locate a hub, we look for a page (other than any final page in $\mathbf{Fin}_{rg}$) that is present in at least 30% (or 80%) of the redirection chains, respectively. These features are based on the assumption that malicious redirection graphs are much more likely to contain hubs.

The *Self-loop on Final Page* is true when any of the redirection chains in $\mathbf{C}_{rg}$ has a self-loop for the final page. A self-loop is a redirection that forwards a user to the same page from where she started. This behavior is often present in benign redirection graphs, where users are redirected to the same page but with different parameters. On the other hand, malicious redirection graphs typically send the victim from a compromised (legitimate) page to a page that has been prepared by the attacker. This is the page that serves the actual exploit, and there is no self-loop present.

**Building and using a classifier**. We use the features described previously (and a labeled training set) to build a classifier. This classifier takes a redirection graph as input. If the graph does not contain at least $t_c$ distinct redirection chains, we discard it as benign (because of the complexity filter). If any of its domains has been visited by more than $t_p$ distinct IP addresses, we also mark the graph as benign (because of the popularity filter). Otherwise, we compute values for our features and submit the resulting feature vector to a classifier based on Support Vector Machines (SVM) trained with Sequential Minimal Optimization (SMO) [17]. We used the implementation provided by the Weka machine-learning tool to this end [6][1]. When the SVM determines that the redirection graph should be considered as malicious, SpiderWeb marks all final page(s) that belong to this graph as malicious.

# 4. EVALUATION

In this section, we describe how we evaluated the ability of SpiderWeb to detect malicious web pages. For our evaluation, we leveraged a dataset $\mathbf{S}$ that contains redirection chains collected over a period of two months, from February 1, 2012 to March 31, 2012 from the customers of a browser security tool developed by a large AV vendor.

[1]We deal with features of different types as follows: we convert strings to integers, by substituting each string with an index, and we treat boolean values as 0 and 1.

The users that generated this traffic are representative of the users of this security product: they are distributed across the globe, and they use a variety of different software configurations. For each HTTP redirection chain, the company logged a time stamp, the obfuscated IP address and country of origin of the client, the initial URL that is requested, the referrer of this request, the final destination URL, all the steps along the chain, and the operating system and browser of the user. In addition, some records are labeled as malicious or benign, a verdict made by the AV tool running on the client. This was done by applying heuristics to identify malicious content in the final pages. If the final page triggered a heuristics, the record is flagged as malicious. On the other hand, the AV vendor employs a whitelist of known, good websites to flag records as benign. If a page is neither detected as bad nor good, that record remains unlabeled.

The entire dataset $\mathbf{S}$ was composed of 388,098 redirection chains, which lead to 34,011 distinct URLs. To estimate the total number of users in the dataset, we calculated the average number of distinct (obfuscated) IP addresses observed during periods of one week (over all weeks in our dataset). Although this number does not give us a precise number of users, it gives us an idea of their magnitude. The average number of distinct IP addresses per week is 13,780. The users that generated the dataset were located in 145 countries, and therefore the produced dataset is representative of the surfing behavior of different people across the globe.

## 4.1 Building a Labeled Dataset

To train the classifier needed by SpiderWeb to detect malicious web pages, we need a labeled dataset of benign and malicious redirection graphs. To this end, we look at redirection graphs built based on the data collected during the first month by the AV vendor (February 2012). As a preliminary filter, we set $t_c$ to 2, which means that we considered only redirection graphs that were composed of at least two distinct redirection chains.

For labeling purposes, if the AV vendor flagged the domain of $\mathbf{Fin}$ as malicious, we manually assessed the maliciousness of the URL. For example, we considered domains that did not exist anymore at the time of our experiment as clearly malicious. Also, we checked the verdict given by the AV vendor to all the redirection chains leading to the same URL. If not all the verdicts agreed in considering the page as malicious, we did not consider that redirection graph for our ground truth. The detection techniques used by the AV vendor are conservative. That is, when they flag a page as malicious, it is very likely that the page is indeed bad. The opposite, however, is not true and, as we will show later, they miss a non-trivial fraction of malicious pages. In total, we picked 2,533 redirection chains flagged as malicious by the AV vendor, leading to 1,854 unique URLs.

We then selected, among the redirection chains in $\mathbf{S}$ whose domain was flagged explicitly as benign by the AV vendor, those redirection chains relative to domains that we consider reputable and legitimate. In particular, we picked both popular websites, such as Google and Facebook, as well as less popular sites that appeared legitimate after careful manual inspection (e.g., local news sites). In the end, we picked 2,466 redirection chains, redirecting to 510 legitimate URLs. More precisely, the final pages of the legitimate redirection chains include all the top 10 Alexa domains [1], and 48 of the top 100. In the remaining of the paper, we refer to the ground truth set, composed of the manually labeled malicious and legitimate redirection chains, as **Tr**.

## 4.2 Selecting Thresholds

As discussed in Section 3, SpiderWeb relies on two thresholds: a complexity threshold $t_c$, to discard any redirection graph that is not complex enough to guarantee an accurate classification, and a popularity threshold $t_p$, to discard those groupings that are too general. In this section, we discuss how we selected the optimal value for these thresholds.

**Calculation of the complexity threshold.** We needed to understand what is the "complexity" that allows for a reliable classification for a redirection graph. With complexity, we mean the number of distinct redirection chains in $\mathbf{C}_{rg}$. Ideally, having graphs of a certain complexity would allow SpiderWeb to detect all malicious redirection graphs with no false positives.

We set up our experiment as follows. First, we built the *Domain + page + parameters* groupings from **Tr**. Then, for each threshold value $i$ between 2 and 10, we removed any redirection graph that had complexity lower than $i$. We call the remaining set of groupings $\mathbf{Gr}_i$. We then performed a ten-fold cross validation on each of the $\mathbf{Gr}_i$ sets, by using Support Vector Machines with SMO, and analyzed the performance of the classifier for each threshold value. As we increase the threshold, the number of false positives decreases. With a threshold of 6, the number of false positives flagged by SpiderWeb is zero. At this threshold, the number of false negatives also reaches zero. This gives us confidence that, by being able to observe enough people surfing the web, SpiderWeb could reliably detect any malicious redirection graph.

Unfortunately, as the threshold value increases, the number of possible detections decreases. This happens because the dataset **S** is limited, and the number of graphs that have high complexity is low. For example, with a threshold $t_c$ of 6, $\mathbf{Gr}_6$ is composed by only 131 redirection graphs, of which only 3 are malicious. Thus, to detect a reasonable number of malicious web pages in our dataset, we have to use a lower $t_c$, and accept a certain rate of false positives. We decided to set $t_c$ to 4. At this threshold, the number of redirection graphs in $\mathbf{Gr}_4$ is 196, of which 25 are malicious. A 10-fold cross-validation on $\mathbf{Gr}_4$ results in a 1.2% false positive rate, and in a true positive rate of 83%.

**Calculation of the popularity threshold.** To derive a suitable value for $t_p$, we analyzed the *Top-Level Domain + page* groupings calculated from **Tr**. We then ran a *precision versus recall* analysis. The reason we did not use the same grouping as for the evaluation of $t_c$ is that, as we said previously, $t_p$ aims at detecting those groupings that are too general for our approach to perform an accurate detection. For this reason, we used *Top-Level Domain + page* groupings, which are the most general among the ones used by SpiderWeb. For each value of $t_p$, we discarded any grouping that had at least one domain that had been visited by more than $t_p$ distinct IP addresses. For the remaining groupings, we analyzed the URLs contained in it. If all the domains that the URLs belonged to ever got marked as malicious by the security company, we flagged the grouping as a true positive. Otherwise, we flagged the grouping as too general (i.e., a false positive). Note that, in this experiment, a single domain flagged as benign by the AV vendor is sufficient to flag the whole grouping as too general. Figure 1 shows the

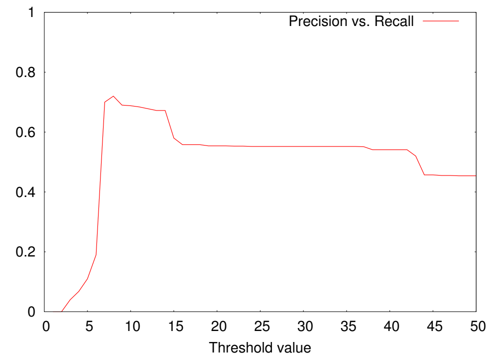


**Figure 1: Precision vs. Recall analysis of the popularity threshold value.**

outcome of the precision versus recall analysis. In the case of our dataset, the optimal value for $t_p$ turned out to be 8. Notice that this analysis is specific to the dataset **S**. In case SpiderWeb would be run on a different dataset, an additional analysis of $t_c$ should be performed, and the optimal value of the threshold might be different.

## 4.3 Analysis of the Classifier

Given the labeled ground truth set **Tr** and suitable threshold values for $t_c$ and $t_p$, we wanted to understand how well our classifier works in finding malicious pages. We performed this analysis for different choices for **R** (which determines how redirection chains are grouped into graphs). More precisely, using the dataset **Tr**, we first generated the redirection graphs leading to the *Top-Level Domain + Page*, the *Top-Level Domain + Page + Parameters*, the *Length + Top-Level Domain + Page + Parameters*, and the *Domain + Page + Parameters* groupings, respectively[2]. We then removed any redirection graph that exceeded the popularity threshold $t_p$, or that did not have at least complexity equal to $t_c$. Each set of redirection graphs (one set of graphs for each choice of $R$) was used in a 10-fold cross validation experiment. The results of these experiments are shown in Table 3. As it can be seen, the false positive rate for the different groupings is generally low. On the other hand, the number of false negatives could appear high. However, as we discussed previously, being able to observe a higher number of connections would allow SpiderWeb to use a higher value for $t_c$, and lower both false positives and false negatives. Table 3 also shows how our system compares to two classifiers based on previous work (SURF and WarningBird). We describe these experiments in Section 4.5.

We then analyzed the expressiveness of our features. From our experiments, the most effective features that characterize a malicious redirection graph are, in order, the "Referrer Country Ratio," the "Graph has Hub 80%," and the "Final Page Domain is an IP" features. On the other hand, the most expressive features that characterize benign redirection graphs are the "Landing Page Country Ratio," the "Intra-domain Step," and the "Top-Level Domain" feature.

[2] Note that we did not use the *IP address* grouping. The reason is that when we obtained the dataset, it was already a few weeks old. Unfortunately, many of the malicious domains did no longer resolve. In those cases, it was impossible for us to determine the IP addresses that corresponded to malicious domains.

| System | # of chains | # of URLs | False Positive rate | False Negative rate | F-Measure |
|---|---|---|---|---|---|
| *TLD + Page* | 112 | 1,945 | 2.5% | 16.1% | 0.881 |
| *TLD + Page + Parameters* | 112 | 1,642 | 0% | 21.4% | 0.88 |
| *Length + TLD + Page + Parameters* | 139 | 1,233 | 3.4% | 20.4% | 0.857 |
| *Domain + Page + Parameters* | 196 | 815 | 1.2% | 17% | 0.863 |
| No graph-specific features | 196 | 815 | 1.2% | 67% | 0.43 |
| No client-specific features | 196 | 815 | 1.2% | 21% | 0.844 |
| No client and graph-specific features | 196 | 815 | 1.5% | 84.78% | 0.113 |
| SURF-based classifier | 196 | 815 | 0% | 81% | 0.319 |
| WarningBird-based classifier | 196 | 815 | 2.9% | 62% | 0.492 |

**Table 3: Summary of the classification performance for different groupings and systems.**

Interestingly, features that would intuitively make sense turned out not to be the most discriminating. For example, it is commonly believed that a long redirection chain is indicative of a malicious web page [10, 13]. As it turns out, the redirection chain length is not among the most important features. In particular, having a short redirection chain is the $8^{th}$ most important characterizing feature for a malicious web page, while having a long redirection chain is the $9^{th}$ most characterizing feature for a benign chain.

As an additional experiment, we also wanted to evaluate how much the features that can only be collected by a crowd of users, and not by a centralized collector, contribute to the detection of malicious web pages. First, we disabled the twelve graph-specific features in the classifier, and we performed another 10-fold cross validation using this feature set. We used the *Domain + Page + Parameters* groupings for this experiment. Surprisingly, the ten-fold cross validation returned the same false positive rate as the one for the full classifier (1.2%). However, the true positive rate dropped to only 33%, compared to the 83% of the full classifier. Then, we ran the 10-fold cross validation with the three client-specific features disabled. Again, we obtained the same false positive rate than for the full classifier, but the true positive rate dropped to 79%. By disabling both the graph-specific and the client-specific features, the true positive rate drops to 15.2%. This means that the features that can only be collected by observing a crowd of users are important to achieve a successful classification. Comparative results with the full classifier are shown in Table 3. These results give us confidence that SpiderWeb indeed improves the state of the art of malicious web page detection, allowing us to perform detection based on features that are specific of the malicious web page ecosystem's structure, rather than on the web page content. Since the redirection-specific features are harder to obfuscate than the content-specific ones, we believe that our technique can help in performing a more reliable malicious web page detection. To underline this, we performed an experiment that shows that SpiderWeb outperforms previous systems (assuming that attackers perfectly succeed in evading the context-specific features). We discuss this experiment in Section 4.5.

## 4.4 Detection Results on Entire Dataset

In the next experiment, we evaluated SpiderWeb on the redirection graphs built from the entire dataset **S** collected by the security company. After checking for complexity and popularity, SpiderWeb generated 3,549 redirection graphs from this dataset, obtained with the *Top-Level Domain + page* (700 graphs), the *Top-Level Domain + page + parameters* (825 graphs), the *Domain Length + Top-Level Domain + page + parameters* (957 graphs), and the *Domain + page + parameters* (1,067 graphs) groupings.

Of the 3,549 redirection graphs built from **S** (using the four groupings), 562 graphs were flagged as malicious by our detection algorithm. In total, these graphs led to 3,368 URLs. We manually identified seven large-scale campaigns, composed of URLs that were sharing common traits.

To check how accurate the results produced by SpiderWeb are, we compared our detections with the ones performed by the AV vendor on the same dataset. To this end, we built a set $\mathbf{Fin}_a$ of analyzed URLs. For each redirection graph in **S**, we added the final URLs in $\mathbf{Fin}_{rg}$ to $\mathbf{Fin}_a$. In total, the AV vendor flagged 3,156 URLs in $\mathbf{Fin}_a$ as malicious. For 2,590 URLs, SpiderWeb and the defenses deployed by the AV vendor agreed. We then manually analyzed the remaining 556 URLs that the AV vendor detected, but SpiderWeb did not. 59 of them turned out to be legitimate web sites that hosted malicious JavaScript, probably because of a cross-site scripting (XSS) vulnerability. SpiderWeb was not designed to detect this kind of web pages, whose redirection graphs, as it turns out, do not show the characteristics that are typical of malicious redirection graphs. The redirection graphs leading to the remaining 497 URLs had low complexity (on average, they were accessed by five distinct redirection chains). As we show in Section 4.2, SpiderWeb needs to observe six or more distinct redirection chains to form a graph that leads to no false positives or false negatives. We are confident that, if more data were available to us, SpiderWeb would have been able to detect also these URLs as malicious.

On the other hand, SpiderWeb detected 778 URLs as malicious that went undetected by the AV vendor. We manually analyzed them, to assess false positives. The vast majority of URLs returned non-existing domain errors when we tried to resolve them. Others showed clear patterns in their URLs typical of exploit kits. For 51 (out of 778) URLs, we could not confirm the maliciousness of the page. We consider all of them as false positives of SpiderWeb. This accounts for 1.5% of the total.

Even if our system has false positives, we think that these false positives are so low in number that it would be possible to whitelist such domains, and use SpiderWeb to detect and blacklist previously-unknown malicious web pages. In addition, SpiderWeb is able to perform more comprehensive detections than existing techniques. Interestingly, in several cases, the software deployed by the AV vendor identified only a few of the malicious web pages that belong to a certain campaign, because only those pages triggered their signature-based detection. The detection of the other pages was successfully evaded by the attackers. On the other hand, once SpiderWeb detects a redirection graph as malicious, it flags each URL in $\mathbf{Fin}_{rg}$ as malicious. Because of the nature of the redirection graphs, and their importance for the attacker, evading detection by SpiderWeb is more difficult.

## 4.5 Comparison with Other Systems

To the best of our knowledge, SpiderWeb is the first system that is able to detect whether a web page is malicious or not just by looking at the redirection chains that lead to it, and at the characteristics of the clients that access it.

However, there are a few recent systems that do make use of redirection information to detect malicious web pages. Unlike SpiderWeb, these systems also leverage "context" (domain-specific) information associated with the redirection chain itself, such as information about the page on which links are posted. In this section, we show that previous systems depend on such contextual information, and that their detection capability suffers significantly if this information is not present. The first problem with these approaches is that attackers can easily modify such context information, for example by changing the way in which they disseminate malicious links. A second problem is that the context information leveraged by previous systems is only available in specific situations, like in the case of a URL posted on Twitter or of a SEO campaign. SpiderWeb, on the other hand, can operate regardless of where the initial URL has been posted. To show this, we compared how SpiderWeb performs compared to two classifiers based on SURF [13] and WarningBird [10].

WarningBird is a system that aims at detecting malicious web pages, such as spam and malware pages, whose links have been posted on Twitter. It takes advantage of redirection graphs, as well as information about the accounts that posted the URLs on Twitter. SURF, on the other hand, is a system that analyzes redirection chains to detect malicious pages that use Search Engine Optimization (SEO) to attract victims. To aid its detection, SURF looks at the search words that attackers poisoned to make their malicious pages appear as results on search engines. Although both systems, to some extent, leverage redirection graphs (SURF visits links twice to evaluate the consistency of a redirection, and WarningBird has a notion of hubs), the analysis performed by SpiderWeb is more comprehensive, and allows one to make decisions just by looking at the graph. Both SURF and WarningBird work very well in detecting malicious web pages that use redirection chains, in the context that they were designed to operate. For instance, SURF is very effective in detecting malicious webpages advertised by SEO, while WarningBird is effective in detecting malicious URLs on Twitter. However, attackers can evade detection by these systems by modifying the way in which they post their links, or they can leverage different methods to spread their malicious content (other than Twitter and SEO). We show that, in the presence of an attacker evading the context information leveraged by previous systems, SpiderWeb performs a better detection.

We developed two classifiers based on the redirection-chain features of WarningBird [10] and SURF [13]. In the case of SURF, we used the *Total redirection hops*, the *Cross-site redirection hops*, the *Landing to terminal distance*, and the *IP-to-name ratio*. In particular, we re-implemented the *Landing to terminal distance* as follows: For each redirection chain, we considered the country of the landing and the terminal domain. If the countries resulted being the same, we set this feature to 1. Otherwise, we set it to 0. Note that we could not re-implement the *Redirection consistency* and the *Page to load/render errors* features, since this information was not available in **S**. In fairness, this could have impacted the performance of our SURF-based classifier in our experiment, since the authors mention that these are among the most discriminative features in their classifier.

In the case of WarningBird, we used the *Redirection chain length*, the *Frequency of entry point URL*, the *Number of different initial URLs*, and the *Number of different landing URL* features. Note that, in WarningBird's terminology, an entry point is what we call a hub. Also, notice that we did not use the *Position of entry point URL* feature. The reason is that WarningBird had a uniform set of referrers (`twitter.com` pages), while we do not. For this reason, it was impossible for us to calculate this feature.

We then applied a ten-fold cross validation using SMO for the two classifiers, using *Domain + Page + Parameter* groupings built from **Tr** as a training dataset. The results are reported in Table 3. As it can be seen, SpiderWeb has by far a better recall than both other classifiers, and a comparable precision to the SURF-based classifier, which is the system that generated the least false positives. To be fair, both SURF and WarningBird achieve better detection rates by leveraging contextual information, which our system does not consider by design. Of course, our system could include contextual information to enhance its detection capabilities, but this is out of the scope of this paper.

Interestingly, the SURF-based and WarningBird-based classifiers detect redirection graphs as malicious that SpiderWeb misses, and vice versa. In particular, SpiderWeb fails in detecting five redirection graphs in $\mathbf{Tr}_4$ as malicious. However, the SURF-based classifier can detect three of those as malicious, and the WarningBird-based classifier can detect the other two. This means that, by using the three systems together, one could reach a complete recall.

## 4.6 Possible Use Cases for SpiderWeb

We envision two possible deployments for our tool: as an offline detection system, and as an online protection system.

**Offline detection.** SpiderWeb can be used offline, to analyze a dataset of historic redirection chains (similar to the one we used for our evaluation). This allows the system to identify a set of malicious URLs. The knowledge of these URLs is useful in several scenarios. For example, it can help to reveal blind spots in alternative detection tools.

**Online prevention.** The problem with the offline approach is that it introduces a possibly significant delay between the appearance of a malicious page and its detection. A better way is to deploy SpiderWeb online, so that it receives a real-time feed of redirection chains from users. Note that the AV vendor already receives such a feed from its users; hence, deployment would be relatively straightforward. Whenever a new redirection chain is received, SpiderWeb adds it to the current set of redirection graphs. Once a graph has reached a sufficient complexity, the classifier is executed. When a page is detected as malicious, this information can be immediately pushed out to the clients, blocking any subsequent access to this page.

Of course, this approach has the limitation that users are potentially exploited until the system observes enough distinct redirection chains to identify a malicious page. To evaluate the extent of this problem, we set up the following experiment: First, we took the 562 redirection graphs in our dataset that SpiderWeb detected as malicious (as discussed in Section 4.4). We then looked at each redirection chain in **S**, in chronological order (based on the time stamp when the redirection chain was entered by the user). The goal was to simulate an online deployment, checking how many users would get compromised before SpiderWeb recognized and blocked a malicious page. We proceeded as follows:

**Step 1:** We keep a list of occurrences $\mathbf{O}$. Each element in $\mathbf{O}$ is a tuple $< \mathbf{R}_i, \mathbf{C}_i >$, where $\mathbf{R}_i$ is a grouping that identifies a malicious redirection graph, and $\mathbf{C}_i$ is a set that contains the distinct redirection chains that previously lead to URLs in $\mathbf{R}_i$. At the beginning, we have an entry in $\mathbf{O}$ for each of the 562 malicious redirection graphs. The set $\mathbf{C}_i$ for each of these graphs is empty. We also keep a value $u_c$ of users who have possibly been compromised, and a value $u_p$ of users who have been protected by SpiderWeb. At the beginning, both these values are set to zero.

**Step 2:** For each redirection graph $\mathbf{G}$ in $\mathbf{S}$, we check its final page $\mathbf{Fin}$ against each grouping $\mathbf{R}_i$ in $\mathbf{O}$. If $\mathbf{G}$ matches $\mathbf{R}_i$, we add the redirection chain $\mathbf{C}$ to $\mathbf{C}_i$.

**Step 3:** If the size of $\mathbf{C}_i$ is lower than 6, it means that the user visiting the page could have potentially been compromised[3]. Therefore, we increase $u_c$ by one. Otherwise, we consider the user as safe, because SpiderWeb would have already blocked the page, and we increase $u_p$ by one.

Running this simulation, we find that the number of compromised users $u_c$ is 723, while the number of protected users $u_p$ is 10,191. This means that out of 10,914 visits of malicious web pages, SpiderWeb would have successfully blocked 93% of them. Note that this is a lower bound, because in many cases less than 6 distinct redirection chains are enough to make a successful detection. Clearly, SpiderWeb cannot provide perfect protection, but its coverage compares favorably to existing blacklists [5]. Also, after identifying a page as malicious, the system knows which users have been potentially infected. These observations could be used to inform the host-based components (e.g., anti-virus scanners) that are already running on those machines.

### 4.7 Limitations and Evasion

Although SpiderWeb is a useful tool for detecting malicious web pages, it has some limitations. The first limitation is that SpiderWeb requires redirection graphs of a certain complexity to perform an accurate classification. We acknowledge that this might be an issue in some cases. The second limitation is that the groupings currently used by SpiderWeb might miss some advanced ways of generating URLs, such as creating a different domain, as well as a different web page name, for each request. As a result, SpiderWeb might fail grouping such URLs together. Although this is a problem, as we discussed in Section 3.2, we picked such groupings because they were the only ones available to us, given the dataset $\mathbf{S}$. In a real scenario, other groupings, such as the IP address at which a domain is hosted, could be used. A grouping by IP addresses would be non-trivial to evade. A third limitation is that, in principle, the attacker might redirect its victim to a popular, legitimate page after having infected her. By doing this, this particular redirection chain would be merged with the others going to the popular site, and SpiderWeb would likely not detect the threat. This is a potential concern, although we did not observe such practice in the wild. A solution to this problem is that SpiderWeb could be modified to build redirection graphs ending at pages other than the final page.

[3]As shown in Section 4.2, with redirection graphs of complexity 6 or higher, SpiderWeb reaches complete coverage.

Attackers could also evade detection by making their redirection graphs look similar to the legitimate ones. However, in case miscreants successfully evaded detection by our system, this would make them easier to detect by previous work. For example, attackers could stop using cloaking. By doing this, their websites would look like regular web pages, accessed by a diverse population of browsers and plugins. However, this evasion would leak their malicious JavaScript in all cases, making it easier for researchers to analyze and block the malicious code. As another example, attackers could stop using hubs. By doing this, they would need to redirect their victims to multiple servers under their control (or compromised servers). The effort of setting up this type of infrastructure raises the bar for attackers. As an alternative, miscreants could redirect their users to the final page directly. However, this would make their redirection chains less complicated, and easier to be flagged by previous work.

## 5. RELATED WORK

Detecting malicious web pages is an important problem, and a wealth of research has been conducted on this topic. Typically, proposed systems aim at detecting a particular type of "malicious" page (e.g., spam or malware). Existing approaches fall in three main categories: *honeypots*, *intra-page feature*- and *inter-page feature*-based detection systems.

**Honeypots**. Systems based on honeypots typically visit web pages with a virtual machine, and monitor the system for changes that are typical of a compromise (e.g., a registry change). HoneyC, capture-hpc, and PhoneyC are examples of such systems [15, 20, 21]. Wang et al. presented a system that runs virtual machines with multiple combinations of browsers and plugins [28]. Although this approach has a higher chance of detecting malicious web pages than the ones using a single browser version, it does not scale, as the number of configurations is constantly increasing.

**Intra-page features**. The second category of systems looks at features that are typical of malicious web pages in order to detect them. Some systems leverage the observation that the URLs used by certain cyber-criminals to point to malicious web pages have recognizable patterns [14,31]. Other systems analyze the actual web page content, looking for words that are indicative of the content to be malicious (e.g., typical of spam or SEO pages) [16, 25, 26]. Another approach is to analyze the JavaScript code in the web page looking for features that are typical of malicious code. This can be done statically [3], or dynamically, by either visiting the page with an emulated browser [2] or a real one [19]. In general, a handful of commercial products leverage data collected in a distributed fashion to detect malicious web pages. However, such products look at the reputation of the pages, or at their content. Systems based on intra-page features are prone to evasion by obfuscation. For example, attackers might obfuscate their malicious JavaScript, making it impossible for an analyzer to detect it.

**Inter-page features**. The third category of systems looks at features of the different pages that are involved in the malicious process, rather than just of the final malicious page. Zhang et al. developed a system to generate signatures that identify URLs that deliver malware, by looking at the redirection chains followed by honeypots as they are directed to the malicious domains [32]. Stokes et al. proposed WebCop, a system that, starting from known malicious domains, traverses the web graph to find malware landing pages [22].

Similarly to SpiderWeb, other systems leverage HTTP redirection chains to detect malicious domains. SURF is designed to detect malicious SEO sites [13], while WarningBird detects malicious web pages posted on Twitter [10]. However, both systems use contextual information to make their detection accurate. In particular, SURF looks at the search engine results that attackers poison, while WarningBird looks at the characteristics of the social network account that posted the link. While they make detection easier, these features are relatively easy to evade by the cyber-criminals. MadTracer [11] is a system that aims at detecting malicious web advertisements by looking at the redirection chains that the users follow. Unlike SpiderWeb, this system uses contextual information about the web advertisement process. Since SpiderWeb does not use any contextual information about the purpose of malicious web pages, it is more generic, and it can detect multiple types of malicious campaigns. To the best of our knowledge, we are the first ones to develop a system that is able to make a decision about the maliciousness of a domain by looking at HTTP redirections only.

## 6. CONCLUSIONS

We presented SpiderWeb, a system that is able to detect malicious web pages by looking at the redirection chains that lead to them. We showed that SpiderWeb can detect malicious web pages in a reliable way. We believe that SpiderWeb complements nicely previous work on detecting malicious web pages, since it leverages features that are harder to evade.

## 7. ACKNOWLEDGMENTS

This work was supported by the Office of Naval Research (ONR) under grant N000140911042, the Army Research Office (ARO) under grant W911NF0910553, the National Science Foundation (NSF) under grants CNS-0845559 and CNS-0905537, and Secure Business Austria. We would like to thank TrendMicro for their support in the project, as well as the anonymous reviewers for their insightful comments.

## 8. REFERENCES


[1] Alexa, the Web Information Company. `http://www.alexa.com`.

[2] M. Cova, C. Kruegel, and G. Vigna. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript code. In *International Conference on World Wide Web*, 2010.

[3] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. Zozzle: Low-overhead mostly static javascript malware detection. In *USENIX Security Symposium*, 2011.

[4] A. Doupè, B. Boe, C. Kruegel, and G. Vigna. Fear the EAR: discovering and mitigating execution after redirect vulnerabilities. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[5] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, et al. Manufacturing compromise: The emergence of exploit-as-a-service. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[6] Hall, M. and Frank, E. and Holmes, G. and Pfahringer, B. and Reutemann, P. and Witten, I.H. The WEKA Data Mining Software: An Update. In *SIGKDD Explorations*, 2009.

[7] McAfee Inc. Mapping the Mal Web. Technical report, 2010.

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

[9] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert. Rozzle: De-Cloaking Internet Malware. In *IEEE Symposium on Security and Privacy*, 2012.

[10] S. Lee and J. Kim. WARNINGBIRD: Detecting Suspicious URLs in Twitter Stream. In *Symposium on Network and Distributed System Security (NDSS)*, 2012.

[11] Z. Li, K. Zhang, Y. Xie, F. Yu, and X.F. Wang. Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[12] H. Liu, K. Levchenko, M. Félegyházi, C. Kreibich, G. Maier, G.M. Voelker, and S. Savage. On the effects of registrarlevel intervention. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2011.

[13] L. Lu, R. Perdisci, and W. Lee. SURF: Detecting and Measuring Search Poisoning. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[14] J. Ma, L.K. Saul, S. Savage, and G.M. Voelker. Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[15] J. Nazario. PhoneyC: a Virtual Client Honeypot. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[16] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting Spam Web Pages Through Content Analysis. In *International Conference on World Wide Web*, 2006.

[17] J. Platt et al. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, 1998.

[18] N. Provos, P. Mavrommatis, M.A. Rajab, and F. Monrose. All Your Iframes Point to Us. In *USENIX Security Symposium*, 2008.

[19] P. Ratanaworabhan, B. Livshits, and B. Zorn. Nozzle: A defense against heap-spraying code injection attacks. In *USENIX Security Symposium*, 2009.

[20] C. Seifert and R. Steenson. Capture-honeypot Client (capture-hpc), 2006.

[21] C. Seifert, I. Welch, P. Komisarczuk, et al. Honeyc: the Low-interaction Client Honeypot. *Proceedings of the 2007 NZCSRCS*, 2007.

[22] J.W. 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 (LEET)*, 2010.

[23] B. Stone-Gross, R. Abman, R. Kemmerer, C. Kruegel, D. Steigerwald, and G. Vigna. The Underground Economy of Fake Antivirus Software. In *Workshop on the Economics of Information Security (WEIS)*, 2011.

[24] B. Stone-Gross, R. Stevens, A. Zarras, R. Kemmerer, C. Kruegel, and G. Vigna. Understanding fraudulent activities in online ad exchanges. In *ACM SIGCOMM Conference on Internet Measurement*, 2011.

[25] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and Evaluation of a Real-time URL Spam Filtering Service. In *IEEE Symposium on Security and Privacy*, 2011.

[26] T. Urvoy, E. Chauveau, P. Filoche, and T. Lavergne. Tracking Web Spam with HTML Style Similarities. *ACM Transactions on the Web (TWEB)*, 2008.

[27] D.Y. Wang, S. Savage, and G.M. Voelker. Cloak and Dagger: Dynamics of Web Search Cloaking. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[28] Y.M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated Web Patrol with Strider Honeymonkeys. In *Symposium on Network and Distributed System Security (NDSS)*, 2006.

[29] Y.M. Wang and M. Ma. Detecting Stealth Web Pages That Use Click-Through Cloaking. Technical report, Microsoft Research Technical Report, MSR-TR-2006-178, 2006.

[30] B. Wu and B.D. Davison. Detecting Semantic Cloaking on the Web. In *International Conference on World Wide Web*, 2006.

[31] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. In *ACM SIGCOMM Computer Communication Review*, 2008.

[32] J. Zhang, C. Seifert, J.W. Stokes, and W. Lee. ARROW: Generating Signatures to Detect Drive-By Downloads. In *International Conference on World Wide Web*, 2011.