Fighting Parasite Hosting: Identifying and Mitigating Unauthorized Ads on Your Webserver

Shuang Hao, Adam Arrowood, Xinyu Xing, Nick Feamster
Georgia Institute of Technology

ABSTRACT
Parasite hosting is an emerging search poisoning attack that compromises Web servers and inserts links that redirects users to scam Web pages that sell products or attempt to victimize users. One of the most significant challenges for Web site operators is that compromised sites are extremely difficult to detect: Attackers adapt techniques such as code obfuscation and cloaking that make it difficult to detect compromise. Of course, because the scam content must eventually be displayed to users to enable attackers to mount their attacks, it is detectable in theory, but devising the appropriate mechanisms to reveal these activities is difficult in practice. We present a system that automatically detects when Web pages have been compromised for the purposes of parasite hosting; the system relies on periodic crawling of groups of Web pages that correspond to various keyword searches in search engines. In this paper, we describe the detection system, explain how it can automatically locate the compromised Web servers being used for parasite hosting, and present our findings regarding the prevalence of parasite hosting on the Internet today. We find that around 1,000 URLs newly indexed by search engines per day are under parasite hosting attack. The query terms explored by our system would have similar detection power compared with manually picked queries. We also identify that the major target of the compromise lies in the blog and wiki applications on the Web servers.

1. Introduction
Internet users fundamentally rely on search engines to discover content on the Web; accordingly, Web site authors and administrators go to great lengths to improve the rankings of their pages by modifying the page structure and content, through a process called search engine optimization (SEO). While many aspects of SEO are considered good Web design, miscreants are now exploiting the ranking functions of search engines as a vehicle for new families of attacks. One such family of attacks is called search poisoning, whereby an attacker modifies a Web site to falsely increase the rank for another site.

A new type of search poisoning involves attaching spam to sites hosted on reputable domains. A typical attack scenario works as shown in Figure 1: the attackers first identify high-visibility Web sites that have vulnerabilities and compromise the servers to gain access to create or change pages. The hacked page is cloaked to display different contents to different visitors. The normal content is showed to the regular visitors (as the solid line). The spam-advertised terms (like “viagra”) and redirect are triggered to search engine crawlers and clicks from search results (following dashed lines). The spam content is only visible to the people who search for the illicit products, which makes it difficult for administrators to notice that their site is compromised. The search intention of the buyers also results in a higher conversion rate [6]. More strawman attacks include posting spam links in blogs, or redirecting to spam sites regardless of the visitor types. We define these attacks as parasite hosting: Attackers compromise benign Web site or abuse the Web services to host spam-advertised pages. The attack allows spammers to use a site’s established search rankings for the domain name and incoming links to promote malicious sites. Parasite hosting increases search engine traffic; people searching for illicit products would follow the poisoned results to make purchases. The affected sites suffer from risks of being reducing their ranking by search engines or giving the public a bad impression due to the spam content. The exploited vulnerability could be further used to launch other attacks, e.g. theft of private information.

In this paper, we present a detection system to capture spam-related search terms, crawl the suspect URLs, and identify parasite hosting on the concerned sites. We present the following contributions beyond previous related work:

We build a service to inspect the integrity of each individual sites or domains. The parasite hosting adopts various ways to keep undetected from the server administrators. For example, attackers use code obfuscation to avoid disk scan [2]. Cloaking and redirection are prevalent methods to escape from the notice of the Web master [6, 10]. But spam-
mbers seek to get high visibility in the search results to sell their products, therefore the tampered spam content is not disguised to the search engines and the users looking for the illicit products. Search engines (e.g., Google, Bing, Yahoo!) become a vantage point to detect tainted Web pages. We set up a Google search on concerned domains to find new search results, which makes the newly created suspect pages get inspected nearly in real time after being indexed by search engines. The focus is not to find arbitrary malicious pages on the Internet, but to protect specified domains from parasite hosting attacks.

The spam-advertised terms are dynamically updated. The key challenge is to know the appropriate terms to query for the suspicious URLs. Not only querying could reduce the number of target pages for crawling, but also sophisticated parasite hosting pages only respond spam content to requests with certain search engine redirect, i.e., the value in parameter “q=” is set to match the spammers’ term list. Most previous research uses either random hot keywords (e.g., words listed in Google trends) [7] or a fixed set of pharmaceutical terms [6]. Instead, we find and track what Web spammers are actively advertising right now. The products that they hope to sell on the spam pages constitute “target term”. The assumption is that the benign Web sites would rarely have the spam words, and searching for “target term” returns the list of suspicious URLs on the concerned domain. In addition, we define a set of “coexistent n-gram”: the auxiliary phrases to help advertise the products or draw users’ attention, e.g., “satisfaction guaranteed”, “per pill”, “low price” etc. Querying “coexistent n-gram” assists to discover new “target term”. After crawling the URLs and verifying the spam pages, we prune the candidate term pool to eliminate the bogus terms.

We investigate the vulnerability to cause the site to be tainted. The URL crawler we built identifies what service softwares and their versions are installed on the Web servers. That would give people insight what vulnerability the attackers are going after. To the best of our knowledge, we are the first to do large-scale investigation about site platform related with parasite hosting attacks.

It is already found that .edu domains are the major targets of Web spammers [6]. We use the sites in .edu domains to evaluate the effectiveness of our prototype system. During our study, we find that:

- The querying terms explored from the known scam pages allow to detect more parasite hosting attacks. The terms with wider conception, e.g., “buy” and “price” yield more scam pages, but our method could also identify specific products, like “levitra”, “samsung” and “louisvuitton” etc.
- Our system detect around 1,000 parasite hosting URLs distributed over 100 domains on average every day. The fact indicates the attack becomes a more severe problem, and our system is effective to make detection in near real-time.
- We observe that the compromise is correlated with the applications installed on the Web servers rather than the underlying operating systems or upper libraries.

The rest of this paper is organized as follows. Section 2 surveys the related work. Section 3 describes the system architecture that we design for crawling suspect URLs. Section 4 studies the characteristics of the poisoned sites and spam pages. Section 5 shows detection performance and analysis, and Section 6 concludes.

2. Related Work

Many approaches have been proposed to detect search poisoning attacks. They are mainly based on lexical analysis on the page content [8, 9], or the hyper-link structure from sites to sites [3, 13]. It is common that attackers use techniques of cloaking and redirection to keep the spam pages hard to detect [1, 12]. Lu et al. recently proposed to combine the redirection characteristics to detect clicking on poisoned search results. Those methods are effective to detect individual search poisoning case, but they could not be directly applied to monitor and find all Web spam for a domain or organization. On the other hand, our system dedicates to protect the sites’ integrity under individual domains, and adopt the previously proposed features.

There is recent research studying the search poisoning attacks, especially focusing on the cases with compromised servers. deSEO [5] tried to use similar format in URLs to identify sites infected by the same search poisoning campaign. The difference is that we crawl and investigate the suspect pages from the end users’ perspective, rather than grouping the URLs from the search engines’ perspective. The practical scan could obtain more accurate evidence for spam detection. Leontiadis et al. did a measurement on the search poisoning particularly on the online pharmacies [6], with similar crawling methods we used in this paper. But our intention is to design a service that operators could subscribe to check whether their sites have been exploited for search poisoning. Moreover, we propose a framework to automatically extract current spam-related terms, instead of using a fixed set of search terms or random popular key words.

In a concurrent and independent work, Invernizzi et al. presented EVILSEED [4]. EVILSEED starts from an initial set of known, malicious web pages, and generates search engines queries to identify other similar pages. If the quality of the extracted words is not carefully evaluated and updated in time, it still ends up crawling many pages to get the malicious ones. In comparison, our approach keeps different categories of querying terms to generate, and eliminate the bogus terms, which would keep the term pool smaller and more efficient to find malicious pages In addition, we devote to build a near real-time service to protect concerned sites, and present the design of the crawlers to retrieve richer information and stay unnoticed by the attackers.

3. Detection Method
The general idea is to make use the search engines to find out whether there are suspicious pages on the sites for advertising illicit product. At first it sounds straightforward to submit the search query and crawl the URL for detection. But it becomes a challenging task with consideration of choosing the appropriate key words, retrieving URLs from the search results in time, and making the crawl scalable and anonymous. In this section, we illustrate system design of different components and address the challenges.

3.1 Overview of System Design

Figure 2 shows the architecture of the detection system. The arrows indicate the work flow.

- Service registration & warning notification: To provide a detection service, the system allows Web masters to sign up and get notification if parasite hosting is detected on their sites.

- Dynamic update of the search terms: We choose what words to query on the search engines in order to locate the suspect URLs. The purpose is to find out the illicit products that the spammers are advertising. The new terms are explored after more parasite hosting pages are detected. The dynamic update on the queries leads to pages more likely getting involved in parasite hosting. (Section 3.2)

- Generation of candidate URLs: The selected terms are queried on search engines to obtain suspect URLs. The queries are performed periodically and specified to get the latest indexed pages, which allows the system to detect the attacks nearly in real time. (Section 3.3)

- Crawling for the URLs: The crawler is set to probe the URLs as various roles, including regular user, the search engine crawler and click from search results. We further retrieve the search engine cache of the URL, and collect the system platform information on the servers. (Section 3.4)

- Verification of site compromise and spam pages: Based on the gathered information, we examine whether the URL leads to parasite hosting, e.g., cloaking, redirection. (Section 4)

In the rest of the section, we present the design details of each component respectively.

3.2 Dynamic Update of Search Terms

The quality of the search terms greatly affects the detection performance of the system. Our goal is to identify the terms that return parasite hosting URLs with higher probability from the search results, and eliminate those pointing to many benign pages. We define two categories of terms: “target term” are single words that indicate what illicit products the attackers are advertising through parasite hosting, such as “viagra”, “rolex” etc; “coexistent n-gram” are multi-words that occurring frequently on the spam pages, e.g. “low price”, “free shipping” to draw visitors’ attention or for descriptive purpose. The candidate terms are updated at the end of each epoch (we set as one-day period) after a batch of new parasite hosting pages are collected and identified. The term update goes through three steps, expansion for new terms, ranking of existing terms, and selection of terms for next epoch.

Expansion for new terms: The purpose is to find out new terms that are highly related with parasite hosting pages, and use them as queries to search engines to discover new attack campaigns. We modify information techniques, called Query by Document [14], to extract key phrases from the identified spam pages. Given a piece of spam content, (1) we extract a set of terms for consideration; (2) we evaluate the importance of the candidate terms to represent the spam content.

With the help of a part-of-speech tagger (POST), we first determine the word part-of-speech (e.g., noun, verb, adjective, etc) in the sentences. For example, given a sentence “Buying viagra at extra low prices”, term “Buying” is classified as a verb, “viagra” as a noun and so on. We extract all single noun words as candidates of “target term”. If we use “N”, “V”, “P” and “J” to present noun, verb, preposition and adjective respectively. The example sentence would have POST tags “VNPJJN”. Among them, the noun phrases is a sequence of part-of-speech tags match a noun phrase pattern (NPP). Example noun phrase pattern we use are “JN”, “NN”, “NNN”, “JIN” etc. “viagra” and “extra low prices” would be noun phrases in the example sentence. We consider all noun phrases as candidates of “coexistent n-gram”, and limit the number of words less than four.

To score how likely a “target term” is representative for the given parasite hosting page, We use tf/idf based function, where tf is term frequency in the spam page and idf is inverse document frequency in a background corpus. We take the past 6-day collected pages as the background corpus D to calculate idf. For a term w showing in a spam document d, \( tf(w, d) \) is the occurrence of w in d divided by total word number in d, and the \( tf/idf \) score \( f(w, d) \) is calculated as:

\[
\text{idf}(w) = \log \frac{|D|}{|\{u \in D : w \in u\}|} \\
\text{idf}(w) = \frac{tf(w, d)}{\text{idf}(w)}
\]

where \(|D|\) is total number of pages in the corpus and \(|\{u \in D : w \in u\}|\) is the number of pages where \(w\) appears. The score \(f(w)\) is the maximum value over all spam pages \(d\) detected in the epoch.

The extraction of “coexistent n-gram” is based on the point-wise mutual information (PMI) to discover multi-words that appear together not due to coincidence. Suppose a phrase \(c\) consisting of \(w_1, w_2, \ldots, w_n\), the higher the mutual information among the terms, the higher are the chances of the terms appearing frequently together to form a phrase. Since PMI calculation requires probability of \(c\) and \(w_i\) (\(\text{prob}(c), \text{prob}(w_i)\)) over the background corpus which is extremely expensive to compute, we approximate probably using \(D\)
only.

\[
prob(c) = \frac{tf(c, d)}{tf(POS_c, d)}, \text{ and } prob(w_i) = \frac{tf(w_i, d)}{tf(POS_{w_i}, d)}
\]

where \( tf(POS, d) \) is the frequency of the part-of-speech tagger in the document content.

The \( PMI \) score for a phrase \( c \) is

\[
f(c, d) = PMI(c) = \frac{prob(c)}{\prod_{i=1}^{[c]} prob(w_i)} = \frac{tf(c, d)}{tf(POS_c, d)} \prod_{i=1}^{[c]} \frac{tf(w_i, d)}{tf(POS_{w_i}, d)}
\]

Similar to “target term”, the “coexistent \( n \)-gram” score \( f(c) \) is the maximum value over all spam pages \( d \) detected in the epoch. The higher the score, the more likely the candidate term are considered to appear in parasite hosting pages and correlated with illicit products.

**Ranking of existing terms**: After crawling and analyzing the URLs returned by querying a term, we evaluate the term capability of exposing Web spam. If searching a term gives many URLs leading to parasite hosting, we keep querying the term and crawl the retrieved URLs. On the other hand, if a term returns very low percentage of spam pages, we think the term has bad quality for detecting spam pages and drop it in the selection phase. The ranking criteria is the portion of spam URLs in a time window (multiple past epochs) adjusted with Dirichlet smoothing, which favors more for terms getting many URLs. The smoothing factor \( \alpha_T \) is calculated as the 5-percentile of the URL numbers returned by the terms in the time window \( T \) (we set \( T \) as a 6-day period). Suppose a term \( m \) is active for querying during the period \( D \subset T \), and it gets URL set \( Q_{m,t} \) showed in search engines in epoch \( t \in D \), among which URL set \( P_{m,t} \) are involved in parasite hosting. The ranking score \( \bar{r}(m) \) is

\[
\bar{r}(m) = \frac{\sum_{m \in D} r(m, t)}{|D|}, \text{ where } r(m, t) = \frac{|P_{m,t}|}{|Q_{m,t}| + \alpha_T}
\]

Another concern is to avoid querying terms yielding to the same attack campaign, e.g. querying “viagra” and “silde-nafil” would give back similar set of URLs. We use a greedy algorithm to prioritize terms getting different spam URLs from others. The ranking is an iterative process. At each step, the term with the largest \( \bar{r}(m) \) score is picked to output and removed from the pool. In the next iteration, the rest of the terms will get ranked again, but their spam URL sets do not count the URLs that have been shown in the search results from the output terms. The output sequence forms the rank of the queried terms to evaluate their performance to find parasite hosting attacks.

**Selection of terms for the new epoch**: The ranking phase evaluates the efficiency of terms which have been queried and their returned URLs are crawled. The expansion phase provides the list of potential terms which are expected to find parasite poisoning pages with high probability. The purpose of the selection phase is to combine the two lists and determine the query terms for the new epoch, as in Figure 3. According to from which list the term is selected, the term has two states. If the term comes from the list of ranking phase, we call it elite, which means the term has been tested with good performance. If the term is picked from the list of expansion phase, the state is rookie, which means the term needs exploration. If a term shows in rookie state repeatedly, that indicates although the term is estimated to work well, it does not indeed. We then do not select it for querying in the next epoch. We keep a rookie-timer and a rookie-counter to track the term’s status. The timer records the passed epochs from the last time it is selected, and the counter keeps the occurrences of state rookie from the last elite state. We force a rookie-state term to wait at least exponential of rookie-timer...
counter epochs before getting picked again. The selection process is shown in Algorithm 1. \( L_E \), \( L_R \) and \( L_S \) are the term lists from expansion, ranking and selection phases respectively. \( K_E \) and \( K_R \) indiate at most how many terms we set for elite and rookie states. \( N \) is the counter to record how many times the term has been selected as in rookie states. It is reset to 0 once the term gets in elite state. \( M \) is the timer to track how many epochs the term has not been selected for querying. The timer is set to 0 after the term is selected in an epoch. Lines 2-8 demonstrate the process of selecting elite-state terms; Lines 9-17 show the algorithm to pick rookie-state terms. Especially, line 10 indicates that a rookie-state term \( t \) needs to wait \( 2^{N(t)-1} \) epochs to get selected in rookie state again. Since the purpose and effect of “target term” and “coexistent n-gram” are different, we perform the selection for the two types of terms separately.

**Algorithm 1 Algorithm for term selection**

**Input:** elite list \( L_E \), rookie list \( L_R \); \( K_E \) and \( K_R \)

1. Initialize \( L_S \) to empty. Set counter \( C_E = 0, C_R = 0 \)
2. for each \( t \in L_E \) do
3. Add \( t \) to \( L_S \) and \( C_E = C_E + 1 \)
4. \( N(t) = 0 \)
5. if \( C_E \geq K_E \) then
6. break the for loop
7. end if
8. end for
9. for each \( t \in L_R \) do
10. if \( 2^{N(t)-1} > M(t) \) then
11. Add \( t \) to \( L_S \)
12. \( N(t) = N(t) + 1 \)
13. end if
14. if \( C_R \geq K_R \) then
15. break the for loop
16. end if
17. end for
18. Update \( M(x) \):
19. \( \forall x \in L_S, M(x) = 0 \)
20. \( \forall x \notin L_S, M(x) = M(x) + 1 \)

**Output:** selection list \( L_S \)

### 3.3 Collection of Suspect URLs

The queries to collect the candidate URLs are composed by the selected terms and the concerned sites. The selected terms are dynamically updated at each epoch as we demonstrate above. The concerned sites are specified in the service registration process, during which Web masters need to prove that they own the websites. For example, if domain `xyz.edu` is under monitoring, and the term `viagra` is considered as suspect, the query submitted to Google is “viagra site:xyz.edu”. We evaluate the performance of our detection system on .edu domains, since previous studies have shown that sites in educational institutes are major victim of parasite hosting attacks [6]. In each epoch, we search for URLs multiple times and request the results sorted in dates. Therefore, once a page gets indexed by the search engine, we would collect the URL in a real-time mode. In the experiment, we set up to search the term list twice each day, specify the search time range as 24 hours and limit the result number as 100.

### 3.4 Dispatch of URL Crawling

Our scanning of candidate URLs consisted of six types of automated Web requests, proxied via Tor or VPN: Regular crawl, Google cache crawl, Googlebot User Agent based crawl, Google Referer (curl-based) crawl, and Google Referer (browser-based) crawl, application fingerprint.

**Regular crawl** (Type 1): Each URL is crawled using the curl command line and the contents returned by the crawl, including the HTTP headers returned by the server, were stored for later analysis. A random HTTP User Agent was used for each request, taken from a pool of over 2,000 HTTP User Agents acquired from the Web logs of a large campus. This crawl served as a base-line to compare with the other types of requests and it is assumed that the content retrieved is the content displayed to all “normal” Web browsers visiting the URL.

**Google cache crawl** (Type 2): The contents of the URL as cached by Google is requested and stored. The fetched cache page allows us to understand what page content Google search engine gets from the URL. Since we are retrieving candidate URLs from Google daily, the ages of the cached pages averaged from a day to only a few days old; thus are very likely to represent the content that the Web server would return to Google (and possibly other Web crawlers) and could be compared with other scans of the same URL with little risk of page being changed in the actual document. There are a small percentage of pages for which there is no Google cache or that we were unable to retrieve successfully from Google.

**Googlebot User Agent based crawl** (Type 3): Each URL was crawled using curl with an HTTP User Agent matching that of Google webcrawlers:

```
Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
```

If the website we crawl does not perform IP-based filtering, the request would be accounted for scans from Google.
bot, and the page will serve the content designed for search engine. Given that not all candidate URLs were available from Google’s cache, this scan was done to try and trigger cloaks that activate based on HTTP User Agent.

**Google Referrer (curl-based) crawl (Type 4):** Each URL was crawled with the HTTP User Agent of that of Mac Safari and an HTTP Referrer matching that of a Web search conducted via a Google search for a specific term. For example, for the candidate term “viagra” a curl request with HTTP Referrer of

```
http://www.google.com/search?q=viagra
```

was used. This referrer matches the basic syntax of the HTTP Referrer sent when a browser visits a URL via a Google search result. This request was used to try and trigger an HTTP redirect to spam target website. Such redirects are only returned to Web clients who come to a URL from a Web search for a specific term or terms.

**Google Referrer (browser-based) crawl (Type 5):** Each URL was crawled using a popular Web browser, scripted in such a way as appearing to come from a Google search for a specific term. The requests were relayed through a Web proxy and the logs resulting from each request were stored for later analysis. This type of scan was necessary to catch HTTP referrer-based redirection conducted via JavaScript (or other rich-media browser technologies, such as Flash). Due to the prevalence of code obfuscation, redirection code that requires client-side execution is very difficult to detect via command-line (curl) scanning of candidate URLs. But the downside of browser-based crawl is that the page content is not stored for analysis.

**Application fingerprint (Type 6):** Except for the page content of the URL, we hope to collect the information about the server hosting the page. We use a scanner, WhatWeb [11], to examine what system, Web servers, libraries installed on the server side. For example, using WhatWeb on our department website shows that the we server is Apache 2.2.21 and PHP 5.3.10 is installed.

The summary of different crawls is shown in Table 1. It lists whether the crawl type has been used in previous research, and what is the purpose of each crawl. Every suspect URL goes through the six scans to get a collection of snapshots, which provides the page content presented to different visitors: regular user such as typing the URL directly into a browser, requests from search engine crawlers (Googlebot), and clicks on the search result pages. We discuss how to use the different crawls to in Section 4.

4. **Verification of Spam Pages**

The types of parasite hosting attacks could be categorized as three groups: cloak, redirect and plain-text (as in Figure 4). We will show the details of each type and address the methods to detect them.

4.1 **Cloak**

Cloak is the technique to display different page content based on the visitor types. It allows the parasite hosting stay undetected for a longer period, and promote the rank of spam pages in the search engines. The heuristic for identifying cloak is to compare the pages returned from different crawls. If the titles are different, and the pages have different links which lead to out of current domains or contain the search term in it, the URL is considered as a cloaked page. Given that the crawls for a URL happens almost simultaneously and the candidate URL gets indexed by search engines after a short period (less than 24 hours), the detection method is accurate to identify live cloak pages.

**UA cloak** uses the User Agent field from the HTTP request headers to distinguish HTTP clients as user browser or search engine crawlers. The UA cloak intend to hide the malicious content to users who directly type URL in the browsers to visit the site, but make the spam-advertised terms crawled and indexed by Google. We compare the returned pages from Type 1 and Type 3 to detect UA cloak.

**IP cloak** is to use the IP address of the request to decide what content is shown to the visitors. The scammers maintain a list of IP addresses from the search engines, and the cloak is only triggered by the requests from the IP lists. The purpose is to restrict the cloaked content just visible to the search engines. A request from a non-Google IP (even with User Agent set as search engine crawlers) would not be served with the same content as Google crawler received. In order to observe what the page content Google accesses to, we perform Type 2 crawl to retrieve Google cache, and take the difference from Type 1 crawl.

**Referrer cloak** checks the “Referrer” field in the HTTP headers to determine which URL the request clicks through to visit the current page. The scanner could know whether a visitor comes from clicking the search results from the search engines, and what query the visitor is submitted. It allows the scanner to control which group of end users to see the cloak content. For example, a compromised page for drug selling could show pharmacy links to the visitors clicking through Google results on searching for “viagra”, but serve normal content to visitors search for non-drug terms (which is related to the benign content). The page difference between Type 1 and Type 4 crawl results help to identify whether there is referrer cloak on a URL.

The different types of cloaks could happen together. For example, an URL has both UA cloak and referrer cloak to show illicit product selling, which indicates the scanner hopes the search engines to get the page indexed on the
4.2 Redirect

URL redirection makes a browser to open a different URL usually without the users’ notice. The benign usages include moving a site to a different place, or short aliases for long URLs. In parasite hosting attacks, redirect is abused to send users to the illicit websites from the original page. In order to check redirect, we examine whether there are multiple hops during the connection, and whether the HTTP is redirected a website different from the original domain.

Direct redirect intends to redirect the request to a different URL regardless of the visitor type. Upon receiving a connection attempt, the server sends an HTTP redirect with status code 30x. The browser could follow the “Location” field in the response to open the new URL. Our Type 1 crawl captures the HTTP 30x status code. If the second-level domain in the new URL is not the same as in the original request, we mark it as direct redirect.

Referrer redirect means the redirect only happens if the “Referrer” field in the HTTP headers matches the attacker’s expectation. As mentioned before, the “Referrer” field indicates from clicking which page the request comes from. Attacker use referrer direct mostly to get users clicking on search engine results to see the spam content. The strategy does not directly give attackers benefits, but it will promote the spam site in the search results, and keep the compromise unnoticed unless the Web masters crawl their own pages with the User Agent set as Google crawlers as mentioned above.

Browser-based redirect a type of HTTP Referrer redirect having the additional requirement: a Web client understanding and interpreting JavaScript or HTML Frames is used to retrieve the target website or content. The redirection is done via script execution in the client Web browser and not via a 301 or 302 HTTP redirect. Command-line Web clients, such as wget or curl, will not retrieve the content, since they will get a 200 status code and could not interpret the codes to execute. Type 5 crawl launches a real browser (Safari in our system) to visit the URL. Later we could retrieve the request information from the proxy logs. If the HTTP response is code 200 and there exist requests leading to out-of-domain URLs, we consider a browser-based redirect happening.

4.3 Plain-text spam

We define plain-text spam as the pages stuffed with links pointing to illicit sites, but there is no cloak or redirect identified. Most of plain-text spam occur due to blog or wiki which accepts anonymous comments or posts. The basic idea of checking plain-text spam is to examine whether the querying terms show in the title, links or body in Type 1 crawl. Our system does not focus on detecting plain-text Web spam, since that is straightforward to find abnormal content on the page by the Web master. We exclude the analysis, but our system retains the potential to find plaintext Web spam.

5. Evaluation and Analysis

We implement a prototype of our detection system, and evaluate its performance over the .edu domains. Suppose the term “viagra” is a candidate term, the query we submit Google search to collect the URLs is in the format “viagra site:.edu”. We started the system with 58 terms manually identified from known spam pages. The system ran with dynamic term update from April 8th 2012 to April 14th 2012. The term list gets updated once a day. We use 150 “target term” and 150 “coexistent n-gram”. The numbers of elite to rookie terms for “target term” are 100 to 50; while the numbers for “coexistent n-gram” are 50 to 100. The heuristic is that we hope to keep the “target term” more stable, and explore more on “coexistent n-gram”. In total, we query up to 300 terms each day.

The system detected 6,489 parasite hosting URLs across 480 domains. We define taint rate as the number of URLs found in parasite hosting divided by the number of URLs we need to crawl. Table 2(a) shows the taint rates for different attack types obtained from the detection system. Table 2(b) shows the number and the percentage of the pages discovered by the new terms (i.e. not the 58 terms we used to boost the system). We could see even only considering the automatically added terms, the system still achieve high taint rate for finding parasite hosting attacks.

<table>
<thead>
<tr>
<th>Crawl type</th>
<th>Has been used in previous research?</th>
<th>Visit effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Regular</td>
<td>Yes</td>
<td>From normal end users</td>
</tr>
<tr>
<td>2 Google cache</td>
<td>No</td>
<td>From Googlebot IPs</td>
</tr>
<tr>
<td>3 Googlebot User Agent</td>
<td>Yes</td>
<td>From Googlebot (but different IPs than Google)</td>
</tr>
<tr>
<td>4 Google Referrer (curl-based)</td>
<td>Yes</td>
<td>From clicking Google search results</td>
</tr>
<tr>
<td>5 Google Referrer (browser-based)</td>
<td>No</td>
<td>From clicking Google search results (allowing in-browser script execution)</td>
</tr>
<tr>
<td>6 Application fingerprint</td>
<td>No</td>
<td>To identify OS and application information on the server</td>
</tr>
</tbody>
</table>

Table 1: Summary of crawls in our system.
Table 3 demonstrates the top 10 “target term” and “coexistent n-gram” found by the system with respect to the number of total parasite hosting identified. Some terms might not look obvious to human, but they are effective to find spam webpages. For example, “und” and “der” are Germany, and they would lead to spam pages in German. It is observed that words containing wider conception would provide more pages related to parasite hosting as shown in Table 3. Our system also identify more concrete terms, like “sunglasses” and “cigarettes”, to specific products, e.g., “levitra”, “samsung”, “louisvuitton”.

Next we measure the rate parasite hosting attacks across servers and domains. Figure 5 shows the distribution of hosts involved in parasite hosting per domain. The x-axis is the number of taint servers, and y-axis indicates what percentage of domains have less than or equal to that number of servers involved in parasite hosting. We observe that most of the victim domains have one or two web servers suffered from the attacks, and the largest compromise instances are around 10 (with one exception of 78 taint servers on a single domain). But given that we detect the compromise or spamming activity at current, it is still surprising to observe many domains get at least one hosts exploited within a week.

Figure 6 demonstrate how many domains and servers having been identified with parasite hosting every day. The x-axis is the day count, and y-axis shows the numbers of the taint domains and hosts identified on that day. On average, 233 Web servers in 155 domains with new pages indexed by search engines are under parasite hosting attacks. The huge number is mainly because attackers use cloak and redirect techniques to keep the compromise and the scam pages unnoticed from the Web masters and the regular visitors. Our system provides an automatic tool to quickly and automatically detect the new scam pages created by attackers. The plot also shows our system increased the detection instances during the expansion period, and reach steady detection rate over time.

At last we investigate what kind of Web servers are more likely to get parasite hosting attack. From bottom to the top, the Web server infrastructure include, operating system (e.g., Linux, Windows), Web server (e.g., Apache, IIS, Nginx, Lighttpd), Web programming language (e.g., PHP, Python, Perl), Web application (e.g., WordPress, Drupal, Joomla) and modules or libraries (jQuery, Prototyp, MooTools). Based on the system information collected from Type 6 crawl, the lower layer has very small diversity. The Web server is dominated by Apache, followed by Microsoft IIS; while the upper level, modules or libraries, has big variation across
servers. Regarding the HTTP servers, we investigate the versions of Apache servers. Figure 7 shows the percentages of the versions in the tainted servers. The x-axis is the sorted version numbers, and y-axis indicates the percentage accounting to all the tainted servers. It is noticed that version 2.2.3 is prevalent under attack, and Web masters are encouraged to upgrade Apache to higher versions.

Given that applications are quite diverse, we focus on analysing the different applications rather than the versions. Table 4 shows the top applications we identified on servers related with parasite hosting. The second column displays the application names. The third column is the percentage of the tainted servers with that application installed. Blog and Wiki packages are the major target to compromise the servers.

6. Conclusion

Traditional search poisoning attacks have switched to a new form: the attackers compromise benign Web servers and implant their scam pages to get high rank in search engines, which we define as parasite hosting. Since the attack usually adapts cloaking or redirection techniques to keep unnoticed from the Web masters and the normal visitors. Parasite hosting could last over a long period and give attackers higher conversion rate to sell illicit products. In this paper we design and evaluate a dynamic system to detect parasite hosting attacks. The system automatically retrieve important terms from the known scam pages, submit queries to search engines to collect sets of suspect URLs, and deploy multiple crawlers to examine the page content. Our system detects around 1,000 parasite hosting URLs distributed over 100 domains every day on average. The fact indicates that the attack becomes a more severe problem, and our system is effective to make detection in near real-time. The mechanism we proposed is capable of achieve similar performance compared with finding malicious pages based on a list of manually picked spam-advertised terms. It is observed that the terms with wider conception, e.g., “buy” and “price” yields more scam pages, but our method could also identify specific products, like “levitra”, “samsung” and “louisvuitton” etc. We also find that the compromise is correlated with the applications installed on the Web servers rather than the underlying operating systems, or upper libraries. The major target of the attackers is blog of wiki application.

REFERENCES