Searching the Searchers with SearchAudit

John P. John[‡§], *Fang Yu*[§], *Yinglian Xie*[§], *Martín Abadi*[§*], *Arvind Krishnamurthy*[‡]
[‡]*University of Washington* [§]*Microsoft Research Silicon Valley*
{*jjohn, arvind*}*@cs.washington.edu* {*fangyu, yxie, abadi*}*@microsoft.com*
[*]*University of California, Santa Cruz*

Abstract

Search engines not only assist normal users, but also provide information that hackers and other malicious entities can exploit in their nefarious activities. With carefully crafted search queries, attackers can gather information such as email addresses and misconfigured or even vulnerable servers.

We present SearchAudit, a framework that identifies malicious queries from massive search engine logs in order to uncover their relationship with potential attacks. SearchAudit takes in a small set of malicious queries as seed, expands the set using search logs, and generates regular expressions for detecting new malicious queries. For instance, we show that, relying on just 500 malicious queries as seed, SearchAudit discovers an additional 4 million distinct malicious queries and thousands of vulnerable Web sites. In addition, SearchAudit reveals a series of phishing attacks from more than 400 phishing domains that compromised a large number of Windows Live Messenger user credentials. Thus, we believe that SearchAudit can serve as a useful tool for identifying and preventing a wide class of attacks in their early phases.

1 Introduction

With the amount of information in the Web rapidly growing, the search engine has become an everyday tool for people to find relevant and useful information. While search engines make online browsing easier for normal users, they have also been exploited by malicious entities to facilitate their various attacks. For example, in 2004, the *MyDoom* worm used Google to search for email addresses in order to send spam and virus emails. Recently, it was also reported that hackers used search engines to identify vulnerable Web sites and compromised them immediately after the malicious searches [20, 16]. These compromised Web sites were then used to serve malware or phishing pages.

Indeed, by crafting specific search queries, hackers may get very specific information from search engines that could potentially reveal the existence and locations of security flaws such as misconfigured servers and vulnerable software. Furthermore, attackers may prefer using search engines because it is stealthier and easier than setting up their own crawlers.

The identification of these malicious queries thus provides a wide range of opportunities to disrupt or prevent potential attacks at their early stages. For example, a search engine may choose not to return results to these malicious queries [20], making it harder for attackers to obtain useful information. In addition, these malicious queries could provide rich information about the attackers, including their intentions and locations. Therefore, strategically, we can let the attackers guide us to better understand their methods and techniques, and ultimately, to predict and prevent followup attacks before they are launched.

In this paper, we present *SearchAudit*, a suspicious-query generation framework that identifies malicious queries by auditing search engine logs. While auditing is often an important component of system security, the auditing of search logs is particularly worthwhile, both because authentication and authorization (two other pillars of security [14]) are relatively weak in search engines, and because of the wealth of information that search engines and their logs contain.

Working with SearchAudit consists of two stages: identification and investigation. In the first stage, SearchAudit identifies malicious queries. In the second stage, with SearchAudit's assistance, we focus on analyzing those queries and the attacks of which they are part.

More specifically, in the first stage, SearchAudit takes a few known malicious queries as seed input and tries to identify more malicious queries. The seed can be obtained from hacker Web sites [1], known security vulnerabilities, or case studies performed by other security

researchers [16]. As seed malicious queries are usually limited in quantity and restricted by previous discoveries, SearchAudit monitors the hosts that conducted these malicious queries to obtain an expanded set of queries from these hosts. Using the expanded set of queries, SearchAudit further generates regular expressions, which are then used to match search logs for identifying other malicious queries. This step is critical as malicious queries are typically automated searches generated by scripts. Using regular expressions offers us the opportunity to catch a large number of other queries with a similar format, possibly generated by such scripts.

After identifying a large number of malicious queries, in stage two, we analyze the malicious queries and the correlation between search and other attacks. In particular, we ask questions such as: why do attackers use Web search, how do they leverage search results, and who are the victims. Answers to these questions not only help us better understand the attacks, but also provide us an opportunity to protect or notify potential victims before the actual attacks are launched, and hence stop attacks in their early stages.

We apply SearchAudit to three months of sampled Bing search logs. As search logs contain massive amounts of data, SearchAudit is implemented on the Dryad/DryadLINQ [11, 26] platform for large-scale data analysis. It is able to process over 1.2TB of data in 7 hours using 240 machines.

To our knowledge, we are the first to present a systematic approach for uncovering the correlations between malicious searches and the attacks enabled by them. Our main results include:

- **Enhanced detection capability:** Using just 500 seed queries obtained from one hacker Web site, SearchAudit detects another 4 million malicious queries, some even before they are listed by hacker Web sites.
- **Low false-positive rates**. Over 99% of the captured malicious queries display multiple bot features, while less than 2% of normal user queries do.
- **Ability to detect new attacks:** While the seed queries are mostly ones used to search for Web site vulnerabilities, SearchAudit identifies a large number of queries belonging to a different type of attack—forum spamming.
- **Facilitation of attack analysis:** SearchAudit helps identify vulnerable Web sites that are targeted by attackers. In addition, SearchAudit helps analyze a series of phishing attacks that lasted for more than one year. These attacks set up more than 400 phishing domains, and tried to steal a large number of Windows Live Messenger user credentials.

The rest of the paper is organized as follows. We start with reviewing related work in Section 2. Then we present the architecture of SearchAudit in Section 3. As SearchAudit contains two stages, Section 4 focuses on the results of the first stage—presenting the malicious queries identified, and verifying that they are indeed malicious. Section 5 describes the second stage of SearchAudit—analyzing the correlation between malicious queries and other attacks. In this paper, we study three types of attacks in detail: searching for vulnerable Web sites (Section 6), forum spamming (Section 7), and Windows Live Messenger phishing attacks (Section 8). Finally we conclude in Section 9.

2 Related Work

There is a significant amount of automated Web traffic on the Internet [5]. A recent study by Yu et al. showed that more than 3% of the entire search traffic may be generated by stealthy search bots [25] .

One natural question to ask is: what is the motivation of these search bots? While some search bots have legitimate uses, e.g., by search engine competitors or third parties for studying search quality [8, 17], many others could be malicious. It is widely known that attackers conduct click fraud for monetary gain [7, 10]. Recently, researchers have associated malicious searches with other types of attacks. For example, Provos et al. reported that worms such as MyDoom.O and Santy used Web search to identify victims for spreading infection [20]. Also, Moore et al. [16] identified four types of evil searches and showed that some Web sites were compromised shortly after evil searches. They showed that attackers searched for keywords like *"phpizabi v0.848b c1 hfp1"* to gather all the Web sites that have a known PHP vulnerability [9]. Subsequently these vulnerable Web servers were compromised to set up phishing pages.

Besides email spamming and phishing, there are many other types of attacks, e.g., malware propagation and Denial of Service (DoS) attacks. Although there are a wealth of attack-detection approaches, most of these attacks were studied in isolation. Their correlations, especially to Web searches, have not been extensively studied. In this paper, we aim to take a step towards a systematic framework to unveil the correlations between malicious searches and many other attacks.

In SearchAudit, we derive regular expression patterns for matching malicious queries. There are many existing signature-generation techniques for detecting worms and spam emails such as Polygraph [18], Hamsa [15], Autograph [12], Earlybird [21], Honeycomb [13], Neman [24] Vigilante [6], and AutoRE [23]. Some of these approaches are based on semantics, e.g., Neman and Vigilante, and hence are not suitable for us, since query strings do not have semantic information. The remaining content-based signature-generation schemes, Hon-

eycomb, Polygraph, Hamsa, and AutoRE, can generate string tokens or regular expressions. These are more appealing to us since attackers add random keywords to query strings, and we want the generated signatures to capture this polymorphism. In this work, we choose AutoRE, which generates regular expression signatures.

In [20], Provos et al. found malicious queries from the Santy worm by looking at search results. In those attacks, the attackers constantly changed the queries, but obtained similar search results (viz., the Web servers that are vulnerable to Santy's attack). SearchAudit, on the other hand, is primarily targeted at finding new attacks, of which we have no prior knowledge. SearchAudit is thus a general framework to detect and understand malicious searches. While there might already be proprietary approaches adopted by various search engines, or anecdotal evidence of malicious searches, we hope that our analysis results can provide useful information to the general research community.

3 Architecture

Our main goal is to let attackers be our guides—to follow their activities and predict their future attacks. We use a small-sized set of seed activities to bootstrap our system. The seed is usually limited and restricted to malicious searches of which we are aware. The system then applies a sequence of techniques to extend this seed set in order to identify previously unknown attacks and obtain a more comprehensive view of malicious search behavior.

Figure 1 presents the architecture of our system. At a high level, the system can be viewed as having two stages. In the first stage, it examines search query logs, and expands the set of seed queries to generate additional sets of suspicious queries. This stage is automated and quite general, i.e., it can be used to find different types of suspicious queries pertaining to different malicious activities. The second stage involves the analysis of these suspicious queries to see how different attacks are connected with search—this is mostly done manually, since it requires a significant amount of domain knowledge to understand the behavior of the different malicious entities. This section focuses on the first stage of our system and Sections 6, 7, and 8 provide examples of the analysis done in the second stage.

Extending the seed using query logs appears to be a straightforward idea. Yet, there are two challenges. First, hackers do not always use the same queries; they modify and change query terms over time in order to obtain different sets of search results, and thereby identify new victims. Therefore, simply using a blacklist of bad queries is not effective. Second, malicious searches may be mixed with normal user activities, especially on proxies. So we need to differentiate malicious queries from normal ones, though they may originate from the same machine or IP address. To address these challenges, we do not simply use the suspicious queries directly, but instead generate regular expression signatures from these suspicious queries. Regular expressions help us capture the structure of these malicious queries, which is necessary to identify future queries. We also filter regular expressions that are too general and therefore match both malicious and normal queries. Using these two approaches, the first stage of the system now consists of a pipeline of two steps: *Query Expansion* and *Regular Expression Generation*. Since any set of malicious queries could potentially lead to additional ones, we loop back these queries until we reach a fixed point with respect to query expansion. The rest of this section presents each of the stages in detail.

3.1 Query Expansion

The first step in our system is to take a small set of seed queries and expand them. These seed queries are known to be suspicious or malicious. They could be obtained from a variety of sources, such as preliminary analysis of the search query logs or with the help of domain experts.

Our search logs contain the following information: a query, the time at which the query was issued, the set of results returned to the searcher, and a few properties of the request, such as the IP address that issued the request and the user agent (which identifies the Web browser used). Since the amount of data in the search logs is massive, we use the Dryad/DryadLINQ platform to process data in parallel on a cluster of hundreds of machines.

The seed queries are expanded as follows. We run the seed queries through the search logs to find exact query matches. For each record where the queries match exactly, we extract the IP address that issued the query. We then go back to the search logs and extract all queries that were issued by this IP address. The reasoning here is that since this IP address issued a query that we believe to be malicious, it is probably that other queries from this IP address would also be malicious. This is because attackers typically issue not just a single query but rather multiple queries so as to get more search results. This method of expansion would allow us to capture the other queries issued.

However, it must be noted that since we are using the IP address to expand to other queries, we need to be careful about dynamic IP addresses because of DHCP. In order to reduce the impact of dynamic IPs on our data, we consider only queries that were made on the same day as the seed query.

At the end of this step, we have all the queries that were issued from suspicious IP addresses on the same day.

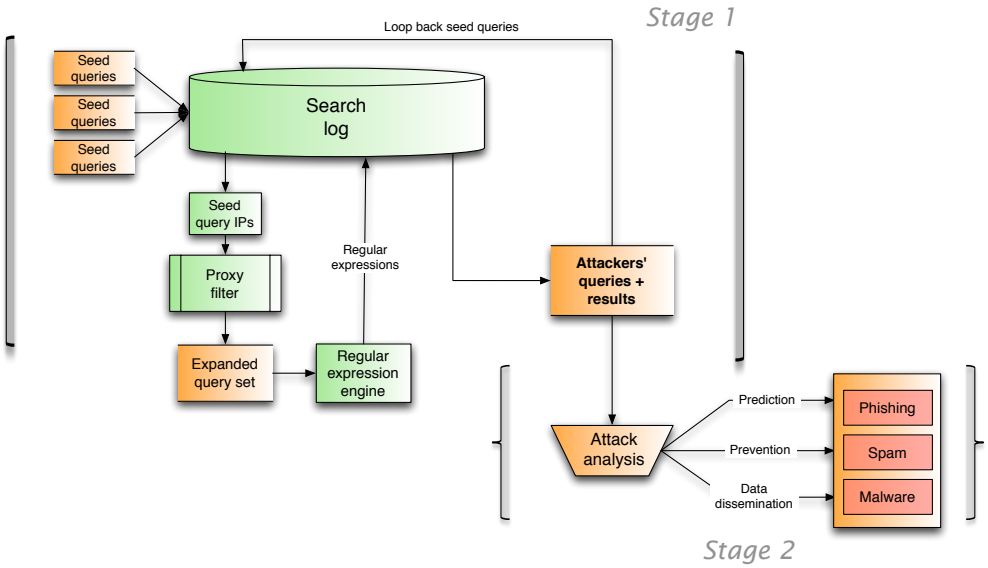

Figure 1: The architecture of the system is a pipeline connecting the query expansion framework, the proxy elimination, and the regular expression generation.

3.2 Regular Expression Generation

The next step after performing query expansion is the generation of regular expressions. We prefer regular expressions over fixed strings for two reasons. First, they can potentially match malicious searches even if attackers change the search terms slightly. In our logs, we find that many hackers add restrictions to the query terms, e.g., adding "`site:cn`" will obtain search results in the `.cn` domain only; regular expressions can capture these variations of queries. Second, as many of the queries are generated using scripts, regular expressions can capture the structure of the queries and therefore can match future malicious queries.

Signature Generation: We use a technique similar to AutoRE [23] to derive regular expressions, with a few modifications to incorporate additional information from the search domain, such as giving importance to word boundaries and special characters in a query. The regular-expression generator works as follows. First, it builds a suffix array to identify all popular keywords in the input set. Then it picks the most popular keyword and builds a root node that contains all the input strings matching this keyword. For the remaining strings, it repeats the process of selecting root nodes until all strings are selected. These root nodes are used to start building trees of frequent substrings. Then the regular-expression generator recursively processes each tree to form a forest. For each tree node, the keywords on the path to the root construct a pattern. It then checks the content between keywords and places restrictions on it (e.g., [0-9]{1,3} to constrain the intervening content to be one to three digits). In addition, for each regular expression, we compute a score that measures the likelihood that the regular expression would match a random string. This score is based on entropy analysis, as described in [23]; the lower the score, the more specific the regular expression. However, a too specific regular expression would be equivalent to having an exact match, and thus loses the benefit of using the regular expression in the first place. We therefore need a score threshold to pick the set of regular expressions in order to trade off between the specificity of the regular expression and the possibility of it matching too many benign queries. In SearchAudit, we select regular expressions with score lower than 0.6. (Parameter selection is discussed in detail in Section 4.2.)

Eliminating Redundancies: One issue with the generated regular expressions is that some of them may be redundant, i.e., though not identical, they match the same or similar set of queries. For example, three input strings `query site:A`, `query site:B`, and `query` may generate two regular expressions `query.{0,7}` and `query site:.{1}`. The two regular expressions have different coverage and scores, but are both valid. In order to eliminate redundancy in regular expressions, we use the REGEX_CONSOLIDATE algorithm described in Algorithm 1. The algorithm takes as input S, the set of input queries, $R_1, \ldots, R_n$, the regular expressions, and returns R, the subset of input regular expressions. Here, the function MATCHES(S, R_i) returns the strings $V \subseteq S$ that match the regular expression R_i.

We note that REGEX_CONSOLIDATE is a greedy algorithm and does not return the minimal set of regular ex-

Algorithm 1 REGEX_CONSOLIDATE($S, R_1, \ldots, R_n$)

$R \leftarrow \{\}$
$V \leftarrow \cup_{i=1}^{n}$ MATCHES(S, R_i)
while $|V| > 0$ **do**
  $R_{max} \leftarrow R_j$ where R_j is the regular expression that matches the most number of strings in V
  $R \leftarrow R \cup R_{max}$
  $V \leftarrow V -$ MATCHES(V, R_{max})
end while
return R

pressions required to match all the input strings. Finding the minimal set is in fact NP-Hard [4].

This ability to consolidate regular expressions has another advantage: if the input to the regular-expression generator contains too many strings, it is split into multiple groups, and regular expressions are generated for each group separately. These regular expressions can then be merged together using REGEX_CONSOLIDATE.

Eliminating Proxies: We observe that we can speed up the generation of regular expressions by reducing the number of strings fed as input to the regular-expression generator. However, we would like to do this without sacrificing the quality of the regular expressions generated. We observe in our experiments that some of the seed malicious queries are performed by IP addresses that correspond to public proxies or NATs. These IPs are characterized by a large query volume, since the same IP is used by multiple people. Also, most of the queries from these IPs are regular benign queries, interspersed with a few malicious ones. Therefore, eliminating these IPs would provide a quick and easy way of decreasing the number of input strings, while still leaving most of the malicious queries untouched.

In order to detect such *proxy-like* IPs, we use a simple heuristic called *behavioral profiling*. Most users in a geographical region have similar query patterns, which are different from that of an attacker. For proxies that have mostly legitimate users, their set of queries will have a large overlap with the popular queries from the same `/16` IP prefix. We label an IP as a proxy if it issues more than 1000 queries in a day, and if the k most popular queries from that IP and the k most popular queries from that prefix overlap in m queries. (We empirically find $k = 100$ and $m = 5$ to work well.) Note however, that the proxy elimination is purely a performance optimization, and not necessary for the correct operation of SearchAudit. Behavioral profiling could also be replaced with a better technique for detecting legitimate proxies.

Looping Back Queries: Once the regular expressions are generated, they are applied to the search logs in order to extract all queries that match the regular expressions. This is an enlarged set of suspicious queries. These queries generated by SearchAudit can now be fed back into the system as new seed queries for another iteration. A discussion on the effect of looping back queries as seeds, and its benefits, is presented in Section 4.3.3.

Matching Type	Total Queries	Uniq. Queries	IPs
Seed match	122,529	122	174
Exact match (expanded)	216,000	800	264
Regular Expression match	297,181	3,560	1,001

Table 1: The number of search requests, unique queries, and IPs for different matching techniques on the February 2009 dataset.

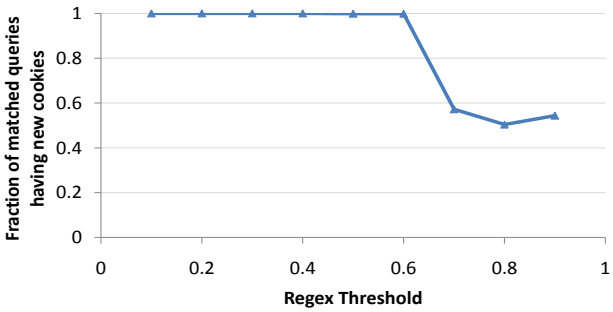

Figure 2: Selecting the threshold for regular expression scores: for regular expressions having score 0.6 or less, nearly all the matched queries have new cookies.

4 Stage One Results

We apply SearchAudit to several months of search logs in order to identify malicious searchers. In this section, we first describe the data collection and system setup. Then we explain the process of parameter selection. Finally, we present the detection results and verify the results.

4.1 Data Description and System Setup

We use three months of search logs from the Bing search engine for our study: February 2009 (when it was known as Live Search), December 2009, and January 2010. Each month of sampled data contains around 2 billion pageviews. Each pageview records all the activities related to a search result page, including information such as the query terms, the links clicked, the query IP address, the cookie, the user agent, and the referral URL. Because of privacy concerns, the cookie and the user agent fields are anonymized by hashing.

The seed malicious queries are obtained from a hacker Web site `milw0rm.com` [1]. We crawl the site and extract 500 malicious queries, which were posted between May 2006 and August 2009.

We implement SearchAudit on the Dryad/DryadLINQ platform, where data is processed in parallel on a cluster of 240 machines. The entire process of SearchAudit takes about 7 hours to process the 1.2 TB of sampled data.

4.2 Selection of Regular Expressions

As described in Section 3.2, we can eliminate proxies to speed up the regular expression generation. If we do not eliminate proxies, the input to the regular-expression generator can contain queries from the proxies, and there may be many benign queries among them. As a result, although some of the generated regular expressions may be specific, they could match benign queries. In this setting, we need to examine each regular expression individually, and select those that match only malicious queries. To do this, we use the presence of old cookies to guide us. We observe that if we pick a random set of search queries (which may contain a mix of normal and malicious queries), the number of new cookies in them is substantially low. However, for the known malicious queries (the seed queries), it is close to 100%, because most automated traffic either does not enable cookies or presents invalid cookies. (In both these cases, a new cookie is created by the search engine and assigned to the search request.) Of course, cookie presence is just one feature of regular user queries. We can use other features as well, as discussed in Section 4.5.

If proxies are eliminated, the remaining queries are from the attackers' IPs, and we find that most of them are malicious. In this case, we can simply use a threshold to pick regular expressions based on their scores. This threshold represents a trade-off between the specificity of the regular expression and the possibility of it being too general and matching too many random queries. Again, we use the number of new cookies as a metric to guide us in our threshold selection. Figure 2 shows the relationship between the regular expression score and the percentage of new cookies in the queries matched by the regular expressions. We see empirically that expressions with scores lower than 0.6 have a very high fraction of new cookies ($> 99.85\%$), similar to what we observe with the seed malicious queries. On the other hand, regular expressions with score greater than 0.6 match queries where the fraction of new cookies is similar to what we see for a random sampling of user queries; therefore it is plausible that these regular expressions mostly match random queries that are not necessarily malicious.

In our tests, proxy elimination filters most of the benign queries, but less than 3% of the unique malicious queries (using cookie-age as the indicator). Therefore it has little effect on the generated regular expressions. Consequently, all the results presented in the paper are with the use of proxy elimination. We choose 0.6 as the regular expression threshold, and this ends up picking about 20% of the generated regular expressions.

Seed Queries Used	Coverage
100 queries (pre-2009)	100%
Random 50%	98.50%
Random 25%	88.50%

Table 2: Malicious query coverage obtained when using different subsets of the seed queries.

4.3 Detection Results

We now present results obtained from running SearchAudit, and show how each component contributes to the end results.

4.3.1 Effect of Query Expansion and Regular Expression Matching

We feed the 500 malicious queries obtained from `milw0rm.com` into SearchAudit, and examine the February 2009 dataset. Using exact string match, we find that 122 of the 500 queries appear in the dataset, and we identify 174 IP addresses that issued these queries. Many of these queries are submitted from multiple IP addresses and many times, presumably to fetch multiple pages of search results. In all, there are 122,529 such queries issued by these IP addresses to the search engine. Then we use the query expansion module together with the proxy elimination module of SearchAudit and obtain 800 unique queries from 264 IP addresses. Finally we run these queries through the regular expression generation engine.

Table 1 quantifies the number of additional queries SearchAudit identifies by the use of query expansion and regular expression generation. Using regular expression matching, SearchAudit identifies 3,560 distinct malicious queries from 1001 IP addresses. Compared to exact matching of the seed queries, regular-expression-based matching increases the number of unique queries found by almost a factor of 30. We also find 4 times more attacker IPs. Thus using regular expressions for matching provides significant gains.

4.3.2 Effect of Incomplete Seeds

Seed queries are inherently incomplete, since they are a very small set of known malicious queries. In this section, we look at how much coverage SearchAudit continues to get when the number of seed queries is decreased.

First, we split the 122 seed queries into two sets: 100 queries that were first posted on `milw0rm.com` before

	IPs	Queries	% Queries with Cookies
No loopback	1,001	297,181	0.15%
Loopback 1	39,969	8,992,839	0.87%
Loopback 2	40,318	9,001,737	0.96%
Loopback 3	41,301	9,028,143	0.97%

Table 3: The number of IPs and queries captured by SearchAudit in the February 2009 dataset, with and without looping back.

2009, and the remaining 22 that were posted in 2009. We then use the 100 queries as our seed, and run SearchAudit on the same search log for a week in February 2009. We find that the queries generated by SearchAudit recover all the 122 seed queries. Therefore SearchAudit is effective in finding the malicious queries even before they are posted on the Web site; in fact we find queries in the search logs several months before they are first posted on the Web site.

Next, we choose a random subset of the original seed queries. With 50% of the randomly selected seed queries, our coverage is 98.5% out of the 122 input seed queries; and using just 25% of the seed queries, we can obtain 88.5% of the queries. These results are summarized in Table 2.

4.3.3 Looping Back Seed Queries

After SearchAudit is bootstrapped using malicious queries, it uses the derived regular expressions to generate a steady stream of queries that are being performed by attackers. SearchAudit uses these as new seeds to generate additional suspicious queries. Each such set of suspicious queries can subsequently be fed back as new seed input to SearchAudit, until the system reaches a fixed point, or until the marginal benefit of finding more such queries outweighs the cost.

To measure when this fixed point would occur, we use the February 2009 dataset, and run SearchAudit multiple times, each time taking the output from the previous run as the seed input. For the first run, we use the 500 seed queries obtained from `milw0rm.com`.

Table 3 summarizes our findings. We see that, as expected, the number of queries captured increases when the generated queries are looped back as new seeds. Also, the number of queries that have valid cookies remains quite small throughout ($< 1\%$), suggesting that the new queries generated through the loopback are similar to the seed queries and the queries generated in the first round. We observe that looping back once significantly increases the set of queries and IPs captured (from 1001 IPs to almost 40,000 IPs), but subsequent iterations do not add much information.

Therefore, we restrict SearchAudit to loop back the generated queries as seeds exactly once.

Dataset	IPs	Total Queries	Uniq. Queries
Feb-2009	39,969	8,992,839	542,505
Dec-2009	29,364	5,824,212	3,955,244
Jan-2010	42,833	2,846,703	422,301

Table 4: The number of search requests, unique queries, and IPs captured by SearchAudit in the different datasets.

4.3.4 Overall Matching Statistics

Putting it all together, i.e., using regular expression matching and loopback, Table 4 shows the number of IPs, total queries, and distinct queries that SearchAudit identifies in each of the datasets. Overall, SearchAudit identifies over 40,000 IPs issuing more than 4 million malicious queries, resulting in over 17 million pageviews. One interesting point to note here is the significant spike in the number of unique queries found in the December dataset. The reason for this spike is the presence of a set of attacker IPs that do not fetch multiple result pages for a query, but instead generate new queries by adding a random dictionary word to the query, thereby increasing the number of distinct queries we observe.

4.4 Verification of Malicious Queries

Next, we verify that the queries identified by SearchAudit are indeed malicious queries. As we lack ground truth information about whether a query is malicious or not, we adopt two approaches. The first is to check whether the query is reported on any hacker Web sites or security bulletins. The second is to check query behavior—whether the query matches individual bot or botnet features.

For each query q returned by SearchAudit, we issue a query "q AND (dork OR vulnerability)" to the search engine, and save the results. Here, the term "dork" is used by attackers to represent malicious searches. We add the terms "dork" and "vulnerability" to the query to help us find forums and Web sites that discuss these queries. We then look at the most popular domains appearing in the search results across multiple queries. Domains that list a large number of malicious searches from our set are likely to be security forums, blogs by security companies or researchers, or even hacker Web sites. These can now be used as new sources for finding more seed queries. We manually examine 50 of these Web sites, and find that around 60% of them are security blogs or advisories. The remaining 40% are in fact hacker forums. In all, 73% of the queries reported by SearchAudit contain search results associated with these 50 Web sites.

Next we look at two sets of behavioral features that would indicate whether the query is automated, and whether a set of queries was generated by the same

script. The first set of features applies to individual bot-generated queries, e.g., not clicking any link. They indicate whether a query is likely to be scripted or not. The second set of features relates to botnet group properties. In particular, they quantify the likelihood that the different queries captured by a particular regular expression were generated by the same (or similar) script.

Note that although these behavior features could distinguish bot queries from human-generated ones, they are not robust features because attackers can easily use randomization or change their behavior if they know these features. In this work, we use these behavior features only for validation rather than relying on them to detect malicious queries.

4.4.1 Verification of Queries Generated by Individual Bots

To distinguish bot queries from those generated by human users, we select the following features:

- *Cookie:* This is the cookie presented in the search request. Most bot queries do not enable cookies, resulting in an empty cookie field. For normal users who do not clear their cookies, all the queries carry the old cookies.
- *Link clicked:* This records whether any link in the search results was clicked by the user. Many bots do not click any link on the result page. Instead, they scrape the results off the page.

We compare queries returned by SearchAudit with queries issued by normal users for popular terms such as `facebook` and `craigslist`. Table 5 and Table 6 show the comparison results. We see that for SearchAudit returned queries, 98.8% of them disable cookies, as opposed to normal users, where only 2.7% disable cookies. We also see that on average, all the queries in a group returned by SearchAudit had no links clicked. On the other hand, for normal users, over 85% of the searches resulted in clicks. All these common features suggest that the queries returned by SearchAudit are highly likely to be automated or scripted searches, rather than being submitted by regular users.

4.4.2 Verification of Queries Generated by Botnets

Having shown that individual queries identified by SearchAudit display bot characteristics, we next study whether a set of queries matched by a regular expression are likely to be generated by the same script, and hence the same attacker (or botnet). For all the queries matched by a regular expression, we look at the behavior of each IP address that issued the queries. If most of the IP addresses that issued these queries exhibit similar behavior, then it is likely that all these IPs were running the same script. We pick the following four features that are representative of querying behavior:

- *User agent:* This string contains information about the browser and the version used.
- *FORM ID:* This field records where the search was issued from, e.g., search home page, browser toolbar.

Some botnets use a fixed user agent string or FORM ID, or choose from a set of common values. For each group, we check the percentage of IP addresses that have identical values or identical behavior, e.g., changing value for each request. If over 90% of the IPs show similar behavior, we infer that IPs in this group might have used the same script.

- *Pages per query:* This records the number of search result pages retrieved per query.
- *Inter-query interval:* This denotes the time between queries issued by the same IP.

Queries generated by the same script may retrieve a similar number of result pages per query or have a similar inter-query interval. For these two features, we compute median value for each IP address and then check whether there is only a small spread in this value across IP addresses ($< 20\%$). This allows us to infer whether the different IPs follow the same distribution, and so belong to the same group.

Table 7 and Table 8 show the comparison between malicious queries and regular query groups. We see that for query groups returned by SearchAudit, a significant fraction of the queries agree on the FORM ID feature. For regular users, one usually observes a wide distribution of FORM IDs. We see a similar trend in the user-agent string as well. For regular users, the user-agent strings rarely match, while for suspicious queries, more than half of them share the same user-agent string. With respect to the number of pages retrieved per search query, we see that regular users typically take only the first page returned. On the other hand, groups captured by SearchAudit fetch on average around 15 pages per query. This varies quite a bit across groups, with many groups fetching as few as 5 pages per query, and several groups fetching as many as 100 pages for a single query.

The average inter-query interval for normal users is over 2.5 hours between successive queries. On the other hand, the average inter-query interval for bot queries is only 7 seconds, with most of the attackers submitting the queries every second or two. A few stealthy attackers repeated search queries at a much slower rate of once every 3 minutes.

For each regular expression group, we sum up the botnet features that it matches. Figure 3 shows the distribution. A majority (87%) of the groups have at least one similar botnet feature and 69% of them have two or

Field	Fraction of Queries within a Group with Same Value
Cookie enabled = *false*	87.50%
Link clicked = *false*	99.90%

Table 5: The fraction of search queries within each regular expression group agreeing on the value of each field.

Field	Fraction of Queries within a Group with Same Value
Cookie enabled = *false*	2.70%
Link clicked = *false*	14.23%

Table 6: The fraction of search queries by normal users agreeing on the value of each field.

Feature	Fraction of Queries within a Group with Same Value
User Agent	51.30%
FORM ID	87.50%
Pages per Query	14.82
Inter-query interval	6.98 seconds

Table 7: The fraction of search queries within each SearchAudit regular expression group agreeing on botnet features.

Feature	Fraction of Queries within a Group with Same Value
User Agent	4.02%
FORM ID	21.80%
Pages per Query	1.07
Inter-query interval	9275.5 seconds

Table 8: The fraction of search queries by normal users agreeing on botnet features.

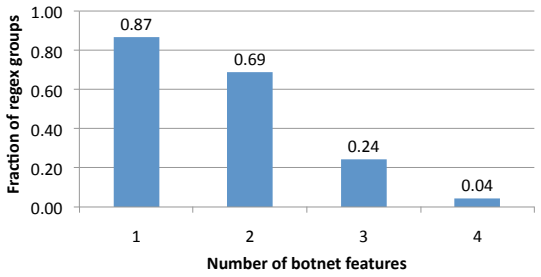

Figure 3: Graph showing the fraction of regular expressions that match one or more botnet features.

more features, suggesting that the queries captured by SearchAudit are probably generated by the same script.

4.5 Discussion

Network security can be an arms race and the generated regular expressions can become obsolete [20]. However, we believe that the signature-based approach is still a viable solution, especially if we have good seed queries. In the paper, we show that even a few hundred seed queries can help identify millions of malicious queries. In addition, SearchAudit can also identify new hackers' forums or security bulletins that can be used as additional sources for seed queries. As long as there are a few IP addresses participating in different types of attacks, the query expansion framework of SearchAudit can be used to follow attackers and capture new attacks.

With the publication of the SearchAudit framework, attackers may try to work around the system and hide their activities. Attackers may try to mix the malicious searches with normal user traffic to trick SearchAudit to conclude that they are using proxy IP addresses. This is hard because behavior profiling requires attackers to submit queries that are location sensitive and also time sensitive. As many attackers use botnets to hide themselves, their IP addresses are usually spread all over the world, making it a challenging task to come up with normal user queries in all regions. In addition, as we mentioned in Section 3, proxy elimination is an optimization and it can be disabled. In such settings, both the normal queries and malicious queries can generate regular expressions. But the regular expressions of normal queries will be discarded because they match many other queries from normal users.

Attackers may also try to add randomness to the queries to escape regular expression generation. The regular expression engine looks at frequently occurring keywords to form the basis of the regular expression. Therefore, even if one attacker can manage not to reuse keywords for multiple queries, he has no control over other attackers using a similar query with the same keyword. An attacker may also simply avoid using a keyword, but since the query needs to be meaningful in order to get relevant search results, this approach would not work.

In this work, we use the presence of old cookies to help us choose regular expressions that are more likely to be malicious; old cookies are a feature associated with normal benign users. We use the cookies as a marker for normal users because it is very simple, and works well in practice. If the attackers evolve and start to use old cookies, possibly by hijacking accounts of benign users, we can rely on other features such as the presence of a real browser, long user history, actual clicking of search results, or other attributes such as user credentials.

Even if a particular attacker is very careful and manages to escape detection, if other attackers are less careful and use similar queries and get caught by SearchAudit, the careful attacker should still be found.

5 Stage 2: Analysis of Detection Results

In this section, we move on to the second stage of SearchAudit: analyzing malicious queries and using search to study the correlation between attacks.

The detected suspicious queries were submitted from more than 42,000 IP addresses across the globe. Large countries such as USA, Russia, and China are responsible for almost half the IPs issuing malicious queries. Looking at the number of queries issued from each IP, we find a large skew: 10% of the IPs are responsible for 90% of the queries.

SearchAudit generates around 200 regular expressions. Table 9 lists ten example regular expressions, ordered by their scores. As we can see, the lower the score, the more specific the regular expression is. The last one `.{1,25}comment.{2,21}` is an example of a discarded regular expression, with a score 0.78. It is very generic (searching for string `comment` only) and hence may cause many false positives.

By inspecting the generated regular expressions and the corresponding query results, we identify two associated attacks: finding vulnerable Web sites and forum spamming. We describe them next.

Vulnerable Web sites: When searching for vulnerable servers, attackers predominantly adopt two approaches:

1. They search within the structure of URLs to find ones that take particular arguments. For example,

   `index.php?content=[^?=#+;&:]{1,10}`

   searches for Web sites that are generated by PHP scripts and take arguments (`content=`). Attackers then try to exploit these Web sites by using specially crafted arguments to check whether they have popular vulnerabilities like SQL injection.

2. They perform malicious searches that are targeted, focusing on particular software with known vulnerabilities.

   We see many malicious queries that start with `"Powered by"` followed by the name of the software and version number, searching for known vulnerabilities in some version of that software.

Forum spamming: The second category of malicious searches are those that do not try to compromise Web sites. Instead, they are aimed towards performing certain actions on the Web sites that are generated by a particular piece of software. The most common goal is Web spamming, which includes spamming on blogs and forums. For example, a regular expression
`"/includes/joomla.php" site:.[a-zA-Z]{2,3}`
searches for blogs generated by the Joomla software. Attackers may have scripts to post spam to such blogs or forums.

Windows Live Messenger phishing: Besides identifying malicious searches generated by attackers, SearchAudit is also useful to study malicious searches triggered by normal users. In April 2009, we noticed in our search logs a large number of queries with the keyword `party`, generated by a series of Windows Live Messenger phishing attacks [25]. We see these queries because the users are redirected by the phishing Web site to pages containing the search results for the query. Since the queries are triggered by normal users compromised by the attack, expanding the queries by IP address will not gain us any information. In this case we use SearchAudit only to generate regular expressions to detect this series of phishing attacks.

In the next three sections, we study these three attacks (compromise of vulnerable Web sites, forum spamming, and Windows Live Messenger phishing) in detail. We aim to answer questions such as how do attackers leverage malicious searches for launching other attacks, how do attacks propagate and at what scale do they operate, and how can the results of SearchAudit be used to better understand and perhaps stop these attacks in their early stages.

6 Attack 1: Identifying Vulnerable Web Sites

As vulnerable Web sites are typically used to host phishing pages and malware, we start with a brief overview of phishing and malware attacks before describing how malicious searches can help find vulnerable Web sites.

6.1 Background of Phishing/Malware Attacks

A typical phishing attack starts with an attacker searching for vulnerable servers by either crawling the Web, probing random IP addresses, or searching the Web with the help of search engines. After identifying a vulnerable server and compromising it, the attacker can host malware and phishing pages on this server. Next, the attacker advertises the URL of the phishing or malware page through spam or other means. Finally, if users are tricked into visiting the compromised server, the attacker can conduct cyber crimes such as stealing user credentials and infecting computers.

Regular Expression	Score
"/includes/joomla\.php" site:\.[a-zA-Z]{2,3}	0.06
"/includes/class_item\.php" site:[^?=#+@;&:]{2,4}	0.08
"php-nuke" site:[^?=#+@;&:]{2,4}	0.16
"modules\.php\?op=modload" site:\.[a-zA-Z0-9]{2,6}	0.16
"[^?=#+@;&:]{0,1}index\.php\?content=[^?=#+@;&:]{1,10}	0.24
"powered by xoopsgallery" [^?=#+@;&:]{0,23}site:[a-zA-Z]{2,3}	0.30
"[^?=#+@;&:]{0,12}\?page=shop\.browse".{0,9}	0.35
.{0,8}index\.php\?option=com_.{3,17}	0.40
[^?=#+@;&:]{0,3}webcalendar v1\..{3,17}	0.43
.{1,25}comment.{2,21}	0.78

Table 9: Example regular expressions and their scores. The last row is an example of a regular expression that is not selected because it is not specific enough.

Currently, phishing and malware detection happens only after the attack is live, e.g., when an anti-spam product identifies the URLs in the spam email, when a browser captures the phishing content, or when anti-virus software detects the malware or virus. Once detected, the URL is added to anti-phishing blacklists. However, it is highly likely that some users may have already fallen victim to the phishing scam by the time the blacklists are updated.

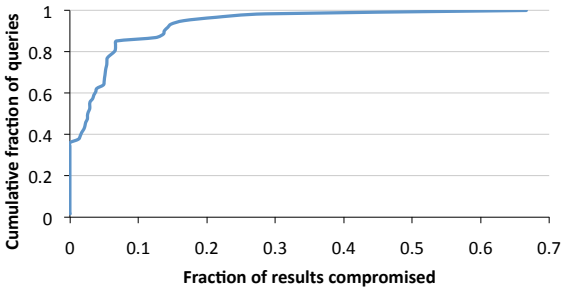

Figure 4: The fraction of search results that were present in phishing/malware feeds for each query.

6.2 Applications of Vulnerability Searches

With SearchAudit, we can potentially detect phishing/malware attack at the very first stage, when the attacker is searching for vulnerabilities. We might even proactively prevent servers from getting compromised.

To obtain the list of vulnerable Web sites, we sample 5,000 queries returned by SearchAudit. For every query q we issue a query "q -dork -vulnerability" to the search engine and record the returned URLs. Here we explicitly exclude the terms "dork" and "vulnerabilities" because we do not want results that point to security forums or hacker Web sites that discuss and post the vulnerability and the associated "dork". Using this approach, we obtain 80,490 URLs from 39,475 unique Web sites.

Ideally, we would like to demonstrate that most of these Web sites are vulnerable. Since there does not exist a complete list of vulnerable Web sites to compare against, we use several methods for our validation. First, we compare this list and a list of random Web sites against a list of known phishing or malware sites, and show that the sites returned by SearchAudit are more likely to appear in phishing or malware blacklists. Second, we test and show that many of these sites indeed have SQL injection vulnerabilities.

6.2.1 Comparison Against Known Phishing and Malware Sites

For the potentially vulnerable Web sites obtained from the malicious queries, we check the presence of these URLs in known anti-malware and anti-phishing feeds. We use two blacklists: one obtained from PhishTank [2] and the other from Microsoft. In addition, we submit these queries to the search engine again at the time of our experiments in order to obtain the latest results.

In both cases, the results are similar: 3-4% of the domains listed in the search results of malicious queries are in the anti-phishing blacklists, and 1.5% of them are in the anti-malware blacklist. In total, around 5% of the domains appear in one or more blacklists. This is significantly higher than other classes of Web sites we considered.

Not all malicious queries may be equally good at finding vulnerable servers. Figure 4 shows the distribution of compromised search results across queries. For the top 10% of the queries, at least 15% of the search results appear in the blacklists.

6.2.2 SQL Injection Vulnerabilities

Next, we show that a subset of these Web sites do indeed have vulnerabilities. Given that SQL injection is a popular attack, since many Web sites use database backends, we test for SQL vulnerabilities.

The best way to prove that a server has SQL injection vulnerabilities would be to actually compromise the server; however, we were not comfortable with doing this. Instead, we limit ourselves to checking if the inputs appear to be sanitized by performing the following study. For the malicious queries, we look at the search results and crawl all of the links twice. For each link, the first time we crawl the link as is, and the second time we add a single quote (') to the first argument to test whether the server sanitizes the argument correctly. Note that we consider URLs that take an argument. We then compare the Web pages obtained from the successive crawls. If the two pages are identical, then it suggests that the input arguments are being properly sanitized, so there is no obvious SQL injection vulnerability. However, if the pages are different, it does not necessarily mean that the input is not being sanitized—it could just be an advertisement that changes with each access. Instead, we look at the `diff` between the two pages, and check whether the second page contains any kind of SQL error. If there is an SQL error in the second page, but not in the first, it shows that the input string is not being filtered properly. While the presence of unsanitized inputs does not guarantee SQL injection vulnerabilities, it is nevertheless a strong indicator.

We examine a sample of 14,500 URLs obtained from the results of malicious queries, and find that 1,760 URLs (12%) do not sanitize the input strings and therefore may be vulnerable to SQL injection. Note that this is a conservative estimate since these URLs only account for Web sites that take arguments in the URL. Other Web sites that take `POST` arguments or have input forms on their pages could also be susceptible to SQL injection attacks.

7 Attack 2: Forum-Spamming Attacks

Using the seed queries from `milw0rm` (which were for the purpose of finding vulnerable Web sites), SearchAudit additionally identifies forum-spamming attacks. In this section, we study the forum-spamming searches in detail.

Dataset	Forum-Searching IPs	Total Searches
February 2009	22,466	5,828,704
December 2009	20,309	1,130,337
January 2010	31,071	567,445

Table 11: Stats on forum-searching IPs.

7.1 Attack Process

Forum spamming is an effective way to deliver spam messages to a large audience. In addition, it may be used as a technique to boost the page rank of Web sites. To do so, spammers insert the URL of the target Web site that they want to promote in a spam message. By posting the message in many online forums, the target Web site would have a high in-degree of links, possibly resulting in a high page rank.

While there are several studies on the effect of forum spamming [19, 22], this section focuses on exploring the ways spammers perform forum spamming. In particular, we show how they discover a large number of forum pages in the first place.

Table 10 shows a few example forum-related queries captured by SearchAudit. There are two types of queries: the first being general like "post a new topic", and the second being more specific, tailored for a particular piece of software. For example,"`UBBCode: !JoomlaComment`" searches for pages generated by the JoomlaComment software. For both types of queries, random keywords are added to increase the search coverage. The randomness is especially useful if spammers use botnets, as each bot will get different query results and they can focus on spamming different forums in parallel.

7.2 Attack Scale

From the regular expressions generated by SearchAudit, we manually identified 46 regular expressions that are associated with forum spamming. Using these regular expressions, we proceeded to study the matched queries and IP addresses. Table 7.2 shows that the number of IPs used for forum searching stayed quite constant in 2009, but in 2010, the number of IP addresses increased by 50%.

Most IPs have transient behavior. Comparing the IPs in December 2009 to those in January 2010, only 3115 (10-15%) IPs overlap. This shows that the forum-spamming hosts either change frequently, or may reside on dynamic IP ranges and hence their IPs change over time. Both these possibilities suggest that they are likely to be botnet hosts. In fact, when we apply the group similarity tests to check botnet behavior (defined in Section 4.4.2), all forum groups have at least one group similarity features.

Regular Expression	# of IPs	Group Similarity Features	Targeted Forum Generation Software
[^?=#+@;&:]{2,7} "Commenta" !JoomlaComment -""#R#	253	3	Joomla
[^?=#+@;&:]{6,11} "ips, inc"	9159	4	IP.Board
[^?=#+@;&:]{1,8} "Message:" photogallery#R#	253	3	PhotoPost
[^?=#+@;&:]{1,9} "Be first to comment this article" akocomment#R#	255	4	AkoComment
[^?=#+@;&:]{1,6} "UBBCode:" !JoomlaComment -""#R#	255	3	JoomlaComment
[^?=#+@;&:]{1,8} "The comments are owned by the poster\. We aren't responsible for their content\." sections#R#	253	3	PHP-Nuke, Xoops, etc.
[a-zA-Z]{4,12} post new topic	1028	1	phpBB, Gallery, etc
[^?=#+@;&:]{5,13} Board Statistics.{0,10}	302	1	Invision Power Board (IP.Board), MyBB, etc.
BBS [a-zA-Z]{4,12}	1861	1	Infopop etc.
yabb [a-zA-Z]{4,14}	388	1	yaBB
ezboard [a-zA-Z]{4,11}	388	1	ezboard
VBulletin [a-zA-Z]{4,11}	360	1	Vbulletin

Table 10: Example regular expressions related to forum searches, their scale, and the targeted forum generation software.

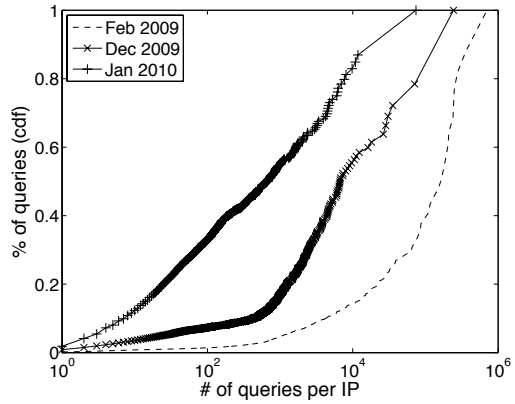

Figure 5: CDF of the distribution of queries among IPs based on the query volume.

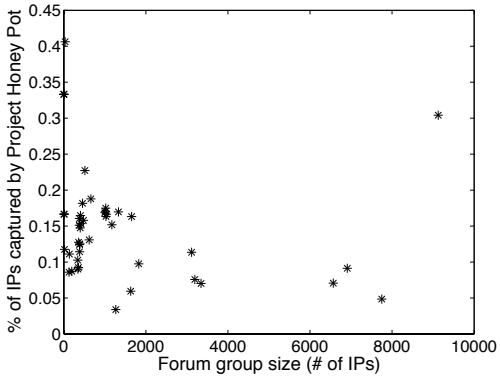

Figure 6: Fraction of IP addresses appearing in the *Project Honey Pot* list vs. the forum group size.

It is interesting to note that, although the number of IPs increased, the total number of queries decreased. As shown in Figure 5, IPs are becoming more stealthy. In February 2009, more than 80% of forum queries were originated from very aggressive IPs that submitted thousands of queries per IP. Those IPs could be spammers' own dedicated machines. In Jan 2010, less than 20% of forum queries are from aggressive IPs. The majority of the queries are from IPs that search at a low rate.

7.3 Applications of Forum Searching Queries

Knowledge of forum-searching IPs and query search terms can be used to help filter forum spam. After a malicious search, we can follow the search result pages to clean up the spam posts. More aggressively, even before the malicious search, by recognizing the malicious query terms or the malicious IP addresses, search engines could refuse to return results to the spammers. Web servers could also refuse connections from IPs that are known to search for forums.

We validate the forum-spamming IPs using Project Honey Pot [3]. Project Honey Pot is a distributed honeypot network that aims to identify Web spamming. Participating Web sites embed a piece of software that dynamically generates a page containing a different email address for each HTTP request. Requests are recorded and the generated email addresses are also monitored. If later they receive emails (which must be spam, since these email addresses are unused), Project Honey Pot

will know which IP addresses obtained those email addresses, and which IP addresses sent the spam emails.

Around 12% of the forum searching IPs found by SearchAudit were captured by Project Honey Pot. In contrast, among IP addresses that conduct normal queries such as `craigslist`, only 0.5% of them were listed. This shows that the captured forum searching IPs have a much higher chance of being caught spamming than the IP addresses of normal users.

Figure 6 plots the matching percentages of different regular expression groups related to forum searching. We can see that, across different groups, the percentages of forum IPs appeared in Project Honey Pot are all significant. This suggests that most of the forum-spamming groups are involved in email address scraping as well. For the largest forum-spamming group, which has 9125 IP addresses, more than 30% of the IP addresses appeared in Project Honey Pot. It is possible that the remaining 70% are also associated with spamming, but they could have targeted Web sites that are not part of their network, and are hence not captured. Hence, the analysis of search logs complements Project Honey Pot. It offers a unique view that allows us to observe all the IP addresses conducting forum searches, while Project Honey Pot allows us to see what the attackers do after performing the searches.

8 Attack 3: Windows Live Messenger Phishing Attacks

In this section, we study a series of Windows Live Messenger phishing attacks. The queries were not issued by attackers directly. Rather, they were triggered by normal users. In this section, we use SearchAudit to generate regular expressions and study this series of attacks.

8.1 Attack Process

The scheme of these phishing attacks operates as follows:

1. The victim (say Alice) receives a message from one of her contacts, asking her to check out some party pictures, with a link to one of the phishing sites.
2. Alice clicks the link and is taken to the Web page that looks very similar to the legitimate Windows Live Messenger login screen and asks her to enter her messenger credentials. Alice enters her credentials.
3. Alice is now taken to a page `http://<domain-name>.com?user=alice`, which redirects to image search results from a search engine (in this case, Bing) for `party`.

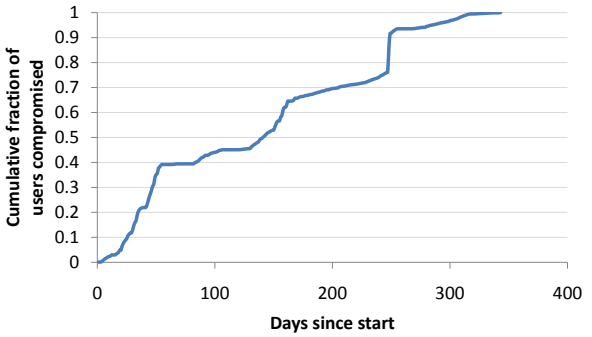

Figure 7: The rate at which new users were compromised by the phishing attack.

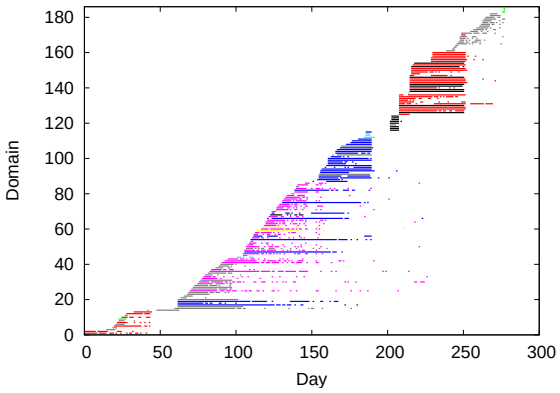

Figure 8: The timeline of how different domain names were used during the phishing attack. All lines of the same color correspond to the same IP address.

4. The attackers now have Alice's credentials. They log in to Alice's account and send a similar message to her friends to further propagate the attack.

We believe there are two reasons why the attackers use a search engine here. First, using images from a search engine is less likely to tip the victim off than if the images were hosted on a random server. Second, the attackers do not need to host the image Web pages themselves, and can thus offload the cost of hosting to the search engine servers.

8.2 Attack Scale

Since this attack generated search traffic that contains the keyword `party`, we feed this keyword as the seed query into SearchAudit. Since all the queries of this attack are identical or similar, we modify SearchAudit to focus on the query referral field, which records the source of traffic redirection. SearchAudit generates two regular expressions from the query referral field:

`1.http://[a-zA-Z0-9._]*.<domain-name>/`
`2.http://<domain-name>?user=[a-zA-Z0-9._]*`

In the second regular expression, the pattern `[a-zA-Z0-9._]*` may seem like a random set of letters and numbers, but it actually describes usernames. In our example attack scenario, when Alice is redirected to the image search results, the HTTP-referrer is set to `http://<domain-name>.com?user=alice`. Using this information, we can identify the set of users whose credentials may have been compromised.

Using these regular expressions, SearchAudit identifies a large number of unique user names in the log collected from May 2008 to July 2009. Figure 7 shows the cumulative fraction of users compromised by this attack over time. When the attack first started, there was an exponential growth phase, similar to other worm or virus breakouts. This phase ended around day 50, when most of the domains got blacklisted (see Figure 8). This attack then transited into a steady increase phase, until day 250 when it broke out again.

There are over 400 unique phishing domain names associated with this attack. The top domains targeted more than 10^5 users. Around one third of the domains phished fewer than 100 users each. These domains were the ones that were quickly blacklisted. Figure 8 plots the timeline of how different domains were used over time. For readability, the plot contains only the top domains (out of the total 400 domains) that were responsible for compromising at least 1000 users. The figure plots the domains on the Y-axis, and the days on which that domain was active on the X-axis. Each horizontal line corresponds to the set of days a particular domain was seen in our search log. The different colors correspond to the different IP addresses on which the Web pages were hosted. We observe that though there were over 180 domain names in circulation, they were all hosted on only a dozen different IP addresses. It can also be seen that multiple domain names were associated with an IP address at the same time. Therefore, it is not the case that a new domain name was registered and used only after an older one was blocked.

8.3 Characteristics of Compromised Accounts

We find that the compromised accounts had a large number of short login sessions (lasting less than one minute). These short login sessions were initiated from IPs in several different `/24` subnets. Figure 9 shows the comparison between the short logins from multiple subnets for compromised users and for the other users. We see that for typical users, 99% of the short logins happened from fewer than 4 different subnets. However, for the compromised users, we see that more than 50% had short logins from 15 or more different subnets.

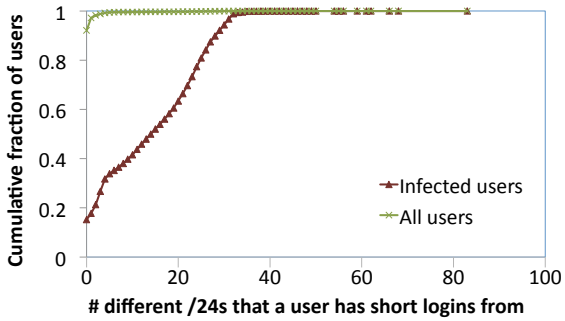

Figure 9: Number of different `/24` subnets from which short logins happen.

We also observe that many of the short logins came from IPs which were located in Hong Kong. Given that the phishing sites were also mostly located in Hong Kong, the attackers might have resources in Hong Kong, where they logged in to the compromised accounts and sent messages to spread the phishing attacks.

Using these characteristics, we can then look back at the login patterns of all Windows Live Messenger users to identify more user accounts with similar suspicious login patterns, thus enabling us to take remedial actions for protecting a larger number of compromised users.

9 Conclusion

In this paper we present SearchAudit, a framework to identify malicious Web searches. By taking just a small number of known malicious queries as seed, SearchAudit can identify millions of malicious queries and thousands of vulnerable Web sites. Our analysis showes that the identification of malicious searches can help detect and prevent large-scale attacks, such as forum spamming and Windows Live Messenger phishing attacks. More broadly, our findings highlight the importance of analyzing search logs and studying correlations between the various attacks enabled by malicious searches.

Acknowledgements

We thank Fritz Behr, Dave DeBarr, Dennis Fetterly, Geoff Hulten, Nancy Jacobs, Steve Miale, Robert Sim, David Soukal, and Zijian Zheng for providing us with data and feedback on the paper. We are also grateful to anonymous reviewers for their valuable comments.

References

[1] milw0rm.com. `http://www.milw0rm.com/`.

[2] PhishTank - Join the fight against phishing. `http://www.phishtank.com`.

[3] Project Honey Pot. http://www.projecthoneypot.org/home.php.

[4] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[5] G. Buehrer, J. W. Stokes, and K. Chellapilla. A large-scale study of automated Web search traffic. In *the 4th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2008.

[6] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of Internet worms. In *the 12th ACM Symposium on Operating Systems Principles (SOSP)*, 2005.

[7] N. Daswani and M. Stoppelman. The anatomy of Clickbot.A. In *the 1st Conference on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[8] E. N. Efthimiadis, N. Malevris, A. Kousaridas, A. Lepeniotou, and N. Loutas. An evaluation of how search engines respond to greek language queries. In *HICSS*, 2008.

[9] D. Eichmann. The RBSE spider - Balancing effective search against Web load, 1994.

[10] S. Frantzen. Clickbot. http://isc.sans.org/diary.html?storyid=1334.

[11] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *European Conference on Computer Systems (EuroSys)*, 2007.

[12] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *the 13th Conference on USENIX Security Symposium*, 2004.

[13] C. Kreibich and J. Crowcroft. Honeycomb: Creating intrusion detection signatures using honeypots. In *the 2nd Workshop on Hot Topics in Networks (HotNets-II)*, 2003.

[14] B. W. Lampson. Computer security in the real world. *IEEE Computer*, 37(6):37–46, June 2004.

[15] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez. Hamsa: Fast signature generation for zero-day polymorphic worm with provable attack resilience. In *IEEE Symposium on Security and Privacy*, 2006.

[16] T. Moore and R. Clayton. Evil searching: Compromise and recompromise of Internet hosts for phishing. In *13th International Conference on Financial Cryptography and Data Security*, 2009.

[17] H. Moukdad. Lost in cyberspace: How do search engines handle Arabic queries. In *the 32nd Annual Conference of the Canadian Association for Information Science*, 2004.

[18] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*, 2005.

[19] Y. Niu, Y. Wang, H. Chen, M. Ma, and F. Hsu. A quantitative study of forum spamming using context based analysis. In *Network and Distributed System Security (NDSS) Symposium*, 2007.

[20] N. Provos, J. McClain, and K. Wang. Search worms. In *the 4th ACM Workshop on Recurring Malcode (WORM)*, 2006.

[21] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Operating Systems Design and Implementation (OSDI)*, 2004.

[22] Y. Wang, M. Ma, Y. Niu, and H. Chen. Spam double-funnel: Connecting Web spammers with advertisers. In *World Wide Web Conference (WWW)*, 2007.

[23] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming botnets: Signatures and characteristics. In *SIGCOMM*, 2008.

[24] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha. An architecture for generating semantics-aware signatures. In *the 14th USENIX Security Symposium*, 2005.

[25] F. Yu, Y. Xie, and Q. Ke. Sbotminer: Large scale search bot detection. In *International Conference on Web Search and Data Mining (WSDM)*, 2010.

[26] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Operating Systems Design and Implementation (OSDI)*, 2008.