In Proceedings of the 23rd USENIX Security Symposium (USENIX Security'14). San Diego, CA. August 2014.

Automatically Detecting Vulnerable Websites Before They Turn Malicious

Kyle Soska and Nicolas Christin
Carnegie Mellon University
{ksoska, nicolasc}@cmu.edu

Abstract

Significant recent research advances have made it possible to design systems that can automatically determine with high accuracy the maliciousness of a target website. While highly useful, such systems are reactive by nature. In this paper, we take a complementary approach, and attempt to design, implement, and evaluate a novel classification system which predicts, whether a given, not yet compromised website will become malicious *in the future*. We adapt several techniques from data mining and machine learning which are particularly well-suited for this problem. A key aspect of our system is that the set of features it relies on is automatically extracted from the data it acquires; this allows us to be able to detect new attack trends relatively quickly. We evaluate our implementation on a corpus of 444,519 websites, containing a total of 4,916,203 webpages, and show that we manage to achieve good detection accuracy over a one-year horizon; that is, we generally manage to correctly predict that currently benign websites will become compromised within a year.

1 Introduction

Online criminal activities take many different forms, ranging from advertising counterfeit goods through spam email [21], to hosting "drive-by-downloads" services [29] that surreptitiously install malicious software ("malware") on the victim machine, to distributed denial-of-service attacks [27], to only name a few. Among those, research on analysis and classification of end-host malware – which allows an attacker to take over the victim's computer for a variety of purposes – has been a particularly active field for years (see, e.g., [6, 7, 16] among many others). More recently, a number of studies [8,15,20,22,36] have started looking into "webserver malware," where, instead of targeting arbitrary hosts for compromise, the attacker attempts to inject code on machines running web servers. Webserver malware differs from end-host malware in its design and objectives. Webserver malware indeed frequently exploits outdated or unpatched versions of popular content-management systems (CMS). Its main goal is usually not to completely compromise the machine on which it resides, but instead to get the victimized webserver to participate in search-engine poisoning or redirection campaigns promoting questionable services (counterfeits, unlicensed pharmaceuticals, ...), or to act as a delivery server for malware.

Such infections of webservers are particularly common. For instance, the 2013 Sophos security threat report [33, p.7] states that in 2012, 80% of websites hosting malicious contents were compromised webservers that belonged to unsuspecting third-parties. Various measurement efforts [20, 25, 36] demonstrate that people engaging in the illicit trade of counterfeit goods are increasingly relying on compromised webservers to bring traffic to their stores, to the point of supplanting spam as a means of advertising [20].

Most of the work to date on identifying webserver malware, both in academia (e.g., [8, 15]) and industry (e.g., [3, 5, 14, 24]) is primarily based on detecting the presence of an *active infection* on a website. In turn, this helps determine which campaign the infected website is a part of, as well as populating blacklists of known compromised sites. While a highly useful line of work, it is by design reactive: only websites that have already been compromised can be identified.

Our core contribution in this paper is to propose, implement, and evaluate a general methodology to identify webservers that are at a high risk of becoming malicious *before* they actually become malicious. In other words, we present techniques that allow to proactively identify likely targets for attackers as well as sites that may be hosted by malicious users. This is particularly useful for search engines, that need to be able to assess whether or not they are linking to potentially risky contents; for blacklist operators, who can obtain, ahead of time, a list of sites to keep an eye on, and potentially warn these sites' operators of the risks they face ahead of the actual compromise; and of course for site operators themselves, which can use tools based on the techniques we describe here as part of a good security hygiene, along with practices such as penetration testing.

Traditional penetration testing techniques often rely on ad-hoc procedures rather than scientific assessment [26] and are greatly dependent on the expertise of the tester herself. Different from penetration testing, our approach relies on an online classification algorithm ("classifier") that can 1) automatically detect whether a server is likely to become malicious (that is, it is probably vulnerable, and the vulnerability is actively exploited in the wild; or the site is hosted with malicious intent), and that can 2) quickly adapt to emerging threats. At a high level, the classifier determines if a given website shares a set of features (e.g., utilization of a given CMS, specifics of the webpages' structures, presence of certain keywords in pages, ...) with websites known to have been malicious. A key aspect of our approach is that the feature list used to make this determination is automatically extracted from a training set of malicious and benign webpages, and is updated over time, as threats evolve.

We build this classifier, and train it on 444,519 archives sites containing a total of 4,916,203 webpages. We are able to correctly predict that sites will eventually become compromised within 1 year while achieving a true positive rate of 66% and a false positive rate of 17%. This level of performance is very encouraging given the large imbalance in the data available (few examples of compromised sites as opposed to benign sites) and the fact that we are essentially trying to predict the future. We are also able to discover a number of content features that were rather unexpected, but that, in hindsight, make perfect sense.

The remainder of this paper proceeds as follows. We review background and related work in Section 2. We detail how we build our classifier in Section 3, describe our evaluation and measurement methodology in Section 4, and present our empirical results in Section 5. We discuss limitations of our approach in Section 6 before concluding in Section 7.

2 Background and related work

Webserver malware has garnered quite a bit of attention in recent years. As part of large scale study on spam, Levchenko et al. [21] briefly allude to search-engine optimization performed my miscreants to drive traffic to their websites. Several papers [17, 19, 20, 22] describe measurement-based studies of the "search-redirection" attacks, in which compromised websites are first being used to link to each other and associate themselves with searches for pharmaceutical and illicit products; this allows the attacker to have a set of high-ranked links displayed by the search engine in response to such queries. The second part of the compromise is to have a piece of malware on the site that checks the provenance of the traffic coming to the compromise site. For instance, if traffic is determined to come from a Google search for drugs, it is immediately redirected—possibly through several intermediaries—to an illicit online pharmacy. These studies are primarily empirical characterizations of the phenomenon, but do not go in great details about how to curb the problem from the standpoint of the compromised hosts.

In the same spirit of providing comprehensive measurements of web-based abuse, McCoy et al. [25] looks at revenues and expenses at online pharmacies, including an assessment of the commissions paid to "network affiliates" that bring customers to the websites. Wang et al. [36] provides a longitudinal study of a search-engine optimization botnet.

Another, recent group of papers looks at how to detect websites that have been compromised. Among these papers, Invernizzi et al. [15] focuses on automatically finding recently compromised websites; Borgolte et al. [8] look more specifically at previously unknown web-based infection campaigns (e.g., previously unknown injections of obfuscated JavaScript-code). Different from these papers, we use machine-learning tools to attempt to detect websites that have not been compromised yet, but that are likely to become malicious in the future, over a reasonably long horizon (approximately one year).

The research most closely related to this paper is the recent work by Vasek and Moore [35]. Vasek and Moore manually identified the CMS a website is using, and studied the correlation between that CMS the website security. They determined that in general, sites using a CMS are more likely to behave maliciously, and that some CMS types and versions are more targeted and compromised than others. Their research supports the basic intuition that the content of a website is a coherent basis for making predictions about its security outcome.

This paper builds on existing techniques from machine learning and data mining to solve a security issue. Directly related to the work we present in this paper is the data extraction algorithm of Yi et al. [38], which we adapt to our own needs. We also rely on an ensemble of decision-tree classifiers for our algorithm, adapting the techniques described by Gao et al. [13].

3 Classifying websites

Our goal is to build a classifier which can predict with high certainty if a given website will become malicious in the future. To that effect, we start by discussing the properties our classifier must satisfy. We then elaborate on the learning process our classifier uses to differentiate between benign and malicious websites. Last, we describe an automatic process for selecting a set features that will be used for classification.

3.1 Desired properties

At a high level, our classifier must be efficient, interpretable, robust to imbalanced data, robust to missing features when data is not available, and adaptive to an environment that can drastically change over time. We detail each point in turn below.

Efficiency: Since our classifier uses webpages as an input, the volume of the data available to train (and test) the classifier is essentially the entire World Wide Web. As a result, it is important the the classifier scale favorably with large, possibly infinite datasets. The classifier should thus use an online learning algorithm for learning from a streaming data source.

Interpretability: When the classifier predicts whether a website will become malicious (i.e., it is vulnerable, and likely to be exploited; or likely to host malicious content), it is useful to understand why and how the classifier arrived at the prediction. Interpretable classification is essential to meaningfully inform website operators of the security issues they may be facing. Interpretability is also useful to detect evolution in the factors that put a website at risk of being compromised. The strong requirement for interpretability unfortunately rules out a large number of possible classifiers which, despite achieving excellent classification accuracy, generally lack interpretability.

Robustness to imbalanced data: In many applications of learning, the datasets that are available are assumed to be balanced, that is, there is an equal number of examples for each class. In our context, this assumption is typically violated as examples of malicious behavior tend to be relatively rare compared to innocuous examples. We will elaborate in Section 5 on the relative sizes of both datasets, but assume, for now, that 1% of all existing websites are likely to become malicious, i.e., they are vulnerable, and exploits for these vulnerabilities exist and are actively used; or they are hosted by actors with malicious intent. A trivial classifier consistently predicting that all websites are safe would be right 99% of the time! Yet, it would be hard to argue that such a classifier is useful at all. In other words, our datasets are imbalanced, which has been shown to be problematic for learning—the more imbalanced the datasets, the more learning is impacted [30].

At a fundamental level, simply maximizing accuracy is not an appropriate performance metric here. Instead, we will need to take into account both false positives (a benign website is incorrectly classified as vulnerable) and false negatives (a vulnerable website is incorrectly classified as benign) to evaluate the performance of our classifier. For instance, the trivial classifier discussed above, which categorizes all input as benign, would yield 0% false positives, which is excellent, but 100% of false negatives among the population of vulnerable websites, which is obviously inadequate. Hence, metrics such as receiver-operating characteristics (ROC) curves which account for both false positive and false negatives are much more appropriate in the context of our study for evaluating the classifier we design.

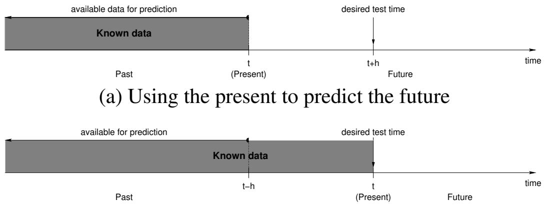
Robustness to errors: Due to its heterogeneity (many different HTML standards co-exist, and HTML engines are usually fairly robust to standard violations) and its sheer size (billions of web pages), the Web is a notoriously inconsistent dataset. That is, for any reasonable set of features we can come up with, it will be frequently the case that some of the features may either be inconclusive or undetermined. As a simple example, imagine considering website popularity metrics given by the Alexa Web Information Service (AWIS, [1]) as part of our feature set. AWIS unfortunately provides little or no information for very unpopular websites. Given that webpage popularity distribution is "heavy-tailed [9]," these features would be missing for a significant portion of the entire population. Our classifier should therefore be robust to errors as well as missing features.

Another reason for the classifier to be robust to errors is that the datasets used in predicting whether a website will become compromised are fundamentally noisy. Blacklists of malicious websites are unfortunately incomplete. Thus, malicious sites may be mislabeled as benign, and the classifier's performance should not degrade too severely in the presence of mislabeled examples.

Adaptive: Both the content on the World Wide Web, and the threats attackers pose vary drastically over time. As new exploits are discovered, or old vulnerabilities are being patched, the sites being attacked change over time. The classifier should thus be able to learn the evolution of these threats. In machine learning parlance, we need a classifier that is adaptive to "concept drift" [37].

All of these desired properties led us to consider an ensemble of decision-tree classifiers. The method of using an ensemble of classifiers is taken from prior work [13]. The system works by buffering examples from an input data stream. After a threshold number of examples has been reached, the system trains a set of classifiers by resampling all past examples of the minority class as well as recent examples of the majority class. While the type of classifier used in the ensemble may vary, we chose to use C4.5 decision trees [31].

The system is efficient as it does not require the storage or training on majority class examples from the far past. The system is also interpretable and robust to errors as the type of classifier being used is a decision-tree in



(a) Using the present to predict the future

(b) Using the past to predict the present

Figure 1: **Prediction timeline.** Attempting to predict the future makes it impossible to immediately evaluate whether the prediction was correct (a). A possible alternative (b) is to use past data to simulate a prediction done in the past (at $t-h$) that can then be tested at the present time t.

an ensemble [13]. Periodically retraining our classifiers makes them robust to concept drift as long as retraining occurs sufficiently often. Finally, the system handles class imbalance by resampling the input stream, namely, it resamples from the set of all minority class training examples from the past as well as recent majority class examples.

3.2 Learning process

The type of classification we aim to perform presents unique challenges in the learning process.

Lack of knowledge of the future: Assume that at a given time t, our classifier predicts that a given website w is likely to become compromised in the future. Because the website has not been compromised yet—and may not be compromised for a while—we cannot immediately know whether the prediction is correct. Instead, we have to wait until we have reached a time $(t+h)$ to effectively be able to verify whether the site has become compromised between t and $(t+h)$, or if the classifier was in error. This is particularly problematic, since just training the classifier—let alone using it—would require to wait at least until $(t+h)$. This is the situation illustrated in Figure 1(a).

A second, related issue, is that of defining a meaningful "time horizon" h. If h is too long, it will be impossible to even verify that the classifier was right. In an extreme case, when $h \to \infty$, the performance of the classifier cannot be evaluated.[1] Selecting a time horizon too short (e.g., $h=0$) would likewise reduce to the problem of determining whether a website is already compromised or not—a very different objective for which a rich literature already exists, as discussed earlier.

We attempt to solve these issues as follows. First, deciding what is a meaningful value for the horizon h appears, in the end, to be a design choice. Unless otherwise noted, we will assume that h is set to one year. This choice does not affect our classifier design, but impacts the data we use for training.

Second, while we cannot predict the future at time t, we can use the past for training. More precisely, for training purposes we can solve our issue if we could extract a set of features, and perform classification on an archived version of the website w as it appeared at time $(t-h)$ and check whether, by time t, w has become malicious. This is what we depict in Figure 1(b). Fortunately, this is doable: At the time of this writing, the Internet Archive's Wayback Machine [34] keeps an archive of more than 391 billion webpages saved over time, which allows us to obtain "past versions" of a large number of websites.

Obtaining examples of malicious and benign websites: To train our classifier, we must have ground truth on a set of websites—some known to be malicious, and some known to be benign. Confirmed malicious websites can be obtained from blacklists (e.g., [28]). In addition, accessing historical records of these blacklists allows us to determine (roughly) at what time a website became malicious. Indeed, the first time at which a compromised website appeared in a blacklist gives an upper bound on the time at which the site became malicious. We can then grab older archived versions of the site from the Wayback Machine to obtain an example of a site that was originally not malicious and then became malicious.

We obtain benign websites by randomly sampling DNS zone files, and checking that the sampled sites are not (and have never been) in any blacklist. We then also cull archives of these benign sites from the Wayback machine, so that we can compare *at the same time in the past* sites that have become malicious to sites that have remained benign.

We emphasize that, to evaluate the performance of the classifier at a particular time t, training examples from the past (e.g., $t-h$) may be used; and these examples can then be used to test on the future. However, the converse is not true: even if that data is available, we cannot train on the present t and test on the past $(t-h)$ as we would be using future information that was unknown at the time of the test. Figure 1(b) illustrates that data available to build predictions is a strict subset of the known data.

Dealing with imbalanced datasets: As far as the learning process is concerned, one can employ class rebalancing techniques. At a high level, class re-balancing has been studied as a means to improve classifier performance by training on a distribution other than the naturally sampled distribution. Since we sample only a random subset of sites which were not compromised, we

[1] Given the complexity of modern computer software, it is likely that exploitable bugs exist in most, if not all webservers, even though they might have not been found yet. As a result, a trivial classifier predicting that all websites will be compromised over an infinite horizon ($h \to \infty$) may not even be a bad choice.

already perform some resampling in the form of a one-sided selection.

3.3 Dynamic extraction of the feature list

Any classifier needs to use a list of features on which to base its decisions. Many features can be used to characterize a website, ranging from look and feel, to traffic, to textual contents. Here we discuss in more details these potential features. We then turn to a description of the dynamic process we use to update these features.

3.3.1 Candidate feature families

As potential candidates for our feature list, we start by considering the following families of features.

Traffic statistics. Website statistics on its traffic, popularity, and so forth might be useful in indicating a specific website became compromised. For instance, if a certain website suddenly sees a change in popularity, it could mean that it became used as part of a redirection campaign. Such statistics may be readily available from services such as the aforementioned Alexa Web Information Service, if the website popularity is not negligible.

Filesystem structure. The directory hierarchy of the site, the presence of certain files may all be interesting candidate features reflecting the type of software running on the webserver. For instance the presence of a `wp-admin` directory might be indicative of a specific content management system (WordPress in that case), which in turn might be exploitable if other features indicate an older, unpatched version is running.

Webpage structure and contents. Webpages on the website may be a strong indicator of a given type of content-based management system or webserver software. To that effect, we need to distill useful page structure and content from a given webpage. The user-generated content within webpages is generally not useful for classification, and so it is desirable to filter it out and only keep the "template" the website uses. Extracting such a template goes beyond extraction of the Document Object Model (DOM) trees, which do not provide an easy way to differentiate between user-generated contents and template. We discuss in the next section how extracting this kind of information can be accomplished in practice.

Page content can then distilled into features using several techniques. We chose to use binary features that detect the presence of particular HTML tags in a site. For instance, "is the keyword *joe's guestbook/v1.2.3* present?" is such a binary feature. Of course, using such a binary encoding will result in a rather large feature set as it is less expressive than other encoding choices. However the resulting features are extremely interpretable and, as we will see later, are relatively straightforward to extract automatically.

Perhaps more interestingly, we observed that features on filesystem structure can actually be captured by looking at the contents of the webpages. Indeed, when we collect information about internal links (e.g., *<a href="../top.html">*) we are actually gathering information about the filesystem as well. In other words, features characterizing the webpage structure provide enough information for our purposes.

3.3.2 Dynamic updates

We consider traffic statistics as "static" features that we always try to include in the classification process, at least when they are available. On the other hand, all of the content-based features are dynamically extracted. We use a statistical heuristic to sort features which would have been useful for classifying recent training examples and apply the top performing features to subsequent examples.

4 Implementation

We next turn to a discussion of how we implemented our classifier in practice. We first introduce the data sources we used for benign and soon-to-be malicious websites. We then turn to explaining how we conducted the parsing and filtering of websites. Last we give details of how we implemented dynamic feature extraction.

4.1 Data sources

We need two different sources of data to train our classifier: a ground truth for soon-to-be malicious websites, and a set of benign websites.

Malicious websites. We used two sets of blacklists as ground truth for malicious websites. First, we obtained historical data from PhishTank [28]. This data contains 11,724,276 unique links from 91,155 unique sites, collected between February 23, 2013 and December 31, 2013. The Wayback machine contained usable archives for 34,922 (38.3%) of these sites within the required range of dates.

We then complemented this data with a list of websites known to have been infected by "search-redirection attacks," originally described in 2011 [20, 22]. In this attack, miscreants inject code on webservers to have them participate in link farming and advertise illicit products—primarily prescription drugs. From a related measurement project [19], we obtained a list, collected between October 20, 2011 and September 16, 2013, of 738,479 unique links, all exhibiting redirecting behavior, from 16,173 unique sites. Amazingly, the Wayback machine contained archives in the acceptable range for 14,425 (89%) of these sites.

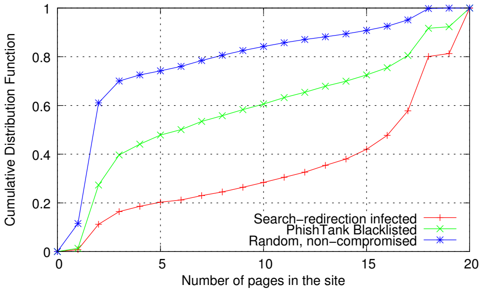


Figure 2: **Cumulative distribution function of the number of pages scraped.** Benign websites very frequently contain only a handful of pages.

We use these two blacklists in particular because we determined, through manual inspection, that a large percentage of sites in these lists have either been compromised by an external attacker or are maliciously hosted. On the other hand, various other blacklists often label sites that are heavily spammed or contain adult content as malicious, which we do not think is appropriate.

Benign websites. We randomly sampled the entire `.com` zone file from January 14th, 2014. For each domain, we enumerated the available archives in the Wayback machine. If at least an archive was found, we selected one of the available archives in the range of February, 20, 2010 to September 31, 2013. This yielded 337,191 website archives. We then removed all archives that corresponded to sites known as malicious. We removed 27 of them that were among the set of sites known to have been infected by search-redirection attacks, and another 72 that matched PhishTank entries. We also discarded an additional 421 sites found in the DNS-BH [2], Google SafeBrowsing [14], and hpHosts [23] blacklists, eventually using 336,671 websites in our benign corpus.

Structural properties. Figure 2 shows some interesting characteristics of the size of the websites we consider. Specifically, the cumulative distribution function of the number of pages each website archive contains differs considerably between the datasets. For many benign sites, that were randomly sampled from zone files, only a few pages were archived. This is because many domains host only a parking page or redirect (without being malicious) to another site immediately. Other sites are very small and host only a few different pages.

On the other hand, malicious sites from both of our blacklists contain more pages per site, since in many cases they are reasonably large websites that had some form of visibility (e.g., in Google rankings), before becoming compromised and malicious. In some other cases, some of the blacklisted sites are sites maliciously registered, that do host numerous phishing pages.

4.2 Parsing and filtering websites

We scraped web pages from the Wayback Machine using the Scrapy framework [4], and a collection of custom Python scripts.

Selecting which archive to use. The scripts took in a URL and a range of dates as inputs, and then navigated The WayBack Machine to determine all the archives that existed for that URL within the specified range.

Sites first appear in a blacklist at a particular time t. If a site appears in multiple blacklists or in the same blacklist multiple times, we use the earliest known infection date. We then search for snapshots archived by the Wayback machine between $t-12$ months and to $t-3$ months prior to the site being blacklisted. The choice of this range is to satisfy two concerns about the usefulness of the archive data. Because compromised sites are not generally instantaneously detected, if the date of the archive is chosen too close to the first time the site appeared in a blacklist, is is possible that the archived version was already compromised. On the other hand, if the archived version was chosen too far from the time at which the site was compromised, the site may have changed dramatically. For instance, the content management system powering the site may have been updated or replaced entirely.

If multiple archives exist in the range $t-3$ months–$t-12$ months, then we select an archive as close to $t-12$ months as possible; this matches our choice for $h=1$ year described earlier. We also download and scrape the most recent available archive, and compare it with the the one-year old archive to ensure that they are using the same content management system. In the event that the structure of the page has changed dramatically (defined as more than 10% changes) we randomly select a more recent archive (i.e., between zero and one year old), and repeat the process.

Scraping process. We scrape each archived site, using a breadth-first search. We terminate the process at either a depth of two links, or 20 pages have been saved, and purposefully only download text (HTML source, and any script or cascading style sheets (CSS) embedded in the page, but no external scripts or images). Using a breadth-first search allows us to rapidly sample a large variety of web pages. It is indeed common for websites to contain multiple kinds of webpages, for example forums and posts, blogs and guestbooks, login and contact pages. A breadth-first search provides an idea of the amount of page diversity in a site without requiring us to scrape the entire site. Limiting ourselves to 20 pages allows us to quickly collect information on a large number of websites, and in fact allows us to capture the vast majority of websites in their entirety, according to Figure 2,

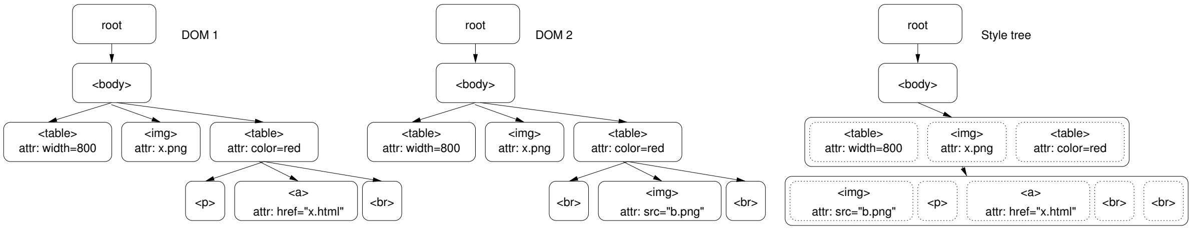


Figure 3: **DOM and style trees**. The figure, adapted from Yi et al. [38], shows two DOM trees corresponding to two separate pages, and the resulting style tree.

while—as we will see later—providing enough information to our classifier to be able to make good decisions.

Filtering. Once a batch of webpages has been saved for a given website, we filter each of them to remove user-generated content. We define user-generated content as all data in webpage, visible and invisible, which is not part of the underline template or content-management system. This includes for instance blog posts, forum posts, guestbook entries, and comments. Our assumption is that user-generated content is orthogonal to the security risks that a site a priori faces and is therefore simply noise to the classifier. User-generated content can, on the other hand, indicate that a site has *already* been compromised, for instance if blog posts are riddled with spam links and keywords. But, since our objective is to detect vulnerable (as opposed to already compromised) sites, user-generated content is not useful to our classification.

The process of extracting information from webpages is a well-studied problem in data mining [10, 11, 32, 38, 39]. Generally the problem is framed as attempting to isolate user-generated content which otherwise would be diluted by page template content. We are attempting to do the exact opposite thing: discarding user-generated content while extracting templates. To that effect, we "turn on its head" the content-extraction algorithm proposed by Yi et al. [38] to have it only preserve templates and discard contents.

Yi et al. describes an algorithm where each webpage in a website is broken down into a Document Object Model (DOM) tree and joined into a single larger structure referred to as a style tree. We illustrate this construction in Figure 3. In the figure, two different pages in a given website produce two different DOM trees (DOM 1 and DOM 2 in the figure). DOM trees are essentially capturing the tags and attributes present in the page, as well as their relationship; for instance, *¡table¿* elements are under *¡body¿*.

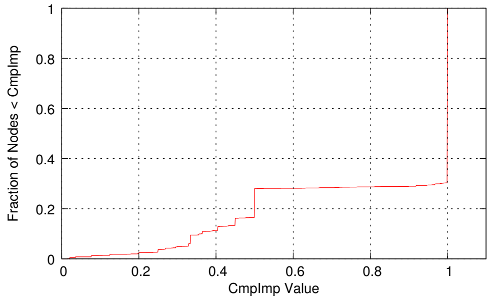


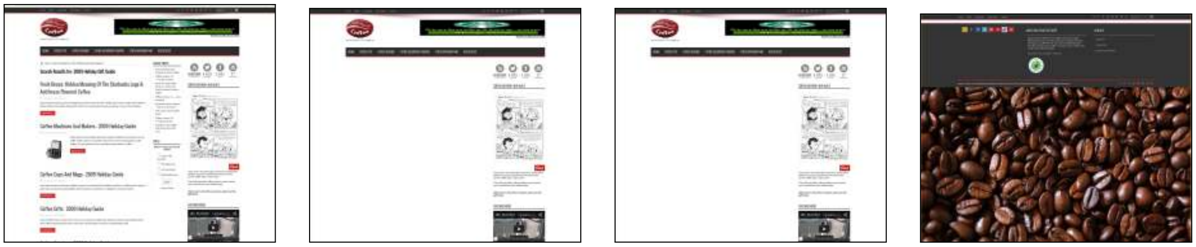
Figure 4: **C.d.f. of CompImp over the style tree generated for dailyshotofcoffee.com**. We use 1,000 pages to generate the style tree.

The style tree incorporates not only a summary of the individual pieces of content within the pages, but also their structural relationships with each other. Each node in a style tree represents an HTML tag from one or possibly many pages within the site and has a derived property called *composite importance* (CompImp), which is an information-based measure of how important the node is. Without getting into mathematical details—which can be found in Yi et al. [38]—nodes which have a CompImp value close to 1 are either unique, or have children which are unique. On the other hand, nodes which have a CompImp value closer to 0 typically appear often in the site.

While Yi et al. try to filter out nodes with CompImp below a given threshold to extract user content, we are interested in the exact opposite objective; so instead we filter out nodes whose CompImp is *above* a threshold.

We provide an illustration with the example of `dailyshotofcoffee.com`. We built, for the purpose of this example, a style tree using 1,000 randomly sampled pages from this website. We plot the CompImp of nodes in the resulting style tree in Figure 4. A large portion of the nodes in the style tree have a CompImp value of exactly 1 since their content is completely unique within the site. The jumps in the graph show that some portions of the site may use different templates or different variations of the same template. For example, particular navigation bars are present when viewing some pages but not when viewing others.

We illustrate the effect of selecting different values as a threshold for filtering in Figure 5. We consider a ran-



(a) Original (b) CompImp > 0.99 (c) CompImp > 0.6 (d) CompImp > 0.1

Figure 5: **Impact of various thresholds on filtering.** The figure shows how different CmpInt thresholds affect the filtering of the webpage shown in (a). Thresholds of 0.99 and 0.6 produce the same output, whereas a threshold of 0.1 discards too many elements.

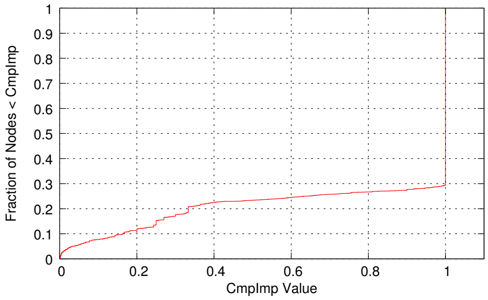


Figure 6: **C.d.f. of CompImp over all style trees generated for 10,000 random sites**. We use 1,000 pages per website to generate the style tree.

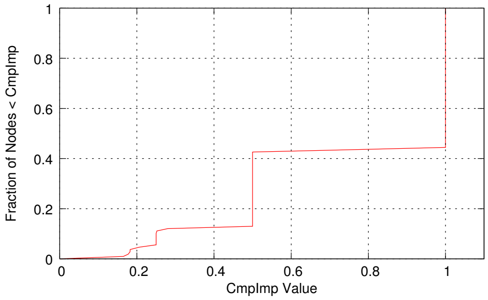


Figure 7: **C.d.f. of CompImp over the style tree generated for `dailyshotofcoffee.com` using only 5 pages.** The plot represents the fraction of nodes less than a threshold in the style tree generated from only five random pages from `dailyshotofcoffee.com`.

dom page of `dailyshotofcoffee.com` showed in Figure 5(a). In Figures 5(b), (c), and (d), we show the result of filtering with a threshold value of 0.99, 0.6 and 0.1 respectively. There is no difference between using 0.99 and 0.6 as a threshold since there are very few nodes in the style tree that had a CompImp between 0.6 and 0.99, as shown in Figure 4. There is a notable difference when using 0.1 as a threshold since portions of the page template are present on some but not all pages of the site.

In general, style trees generated for other sites seem to follow a similar distribution, as shown in Figure 6 where we plot the aggregated CompImp c.d.f. over all style trees generated for 10,000 sites. The aggregation does have a slight curve around the CompImp value 1 which indicates that a few sites do have style trees with nodes in this space. Such sites typically use a fixed template with the exception of a few pages such as 404 error pages, login, and registration pages.

A concern with applying style trees for filtering in this setting occurs in instances where there are only a few examples of pages from a particular site. Trivially, if there is only a single page from the site, then the style tree is just the page itself, and if only a few examples are found then the estimates of nodes in the style tree will be highly dependent on where in the site those pages were sampled from. In Figure 7, we plot the cumulative distribution function for CompImp over the style tree generated for `dailyshotofcoffee.com`, but this time, only using five random pages from the site. Compared to Figure 4, we see that the particular cutoffs are slightly different from when we used 1,000 pages; but the general shape still remains the same. Manual inspection over many sites has indicates that this approach still works well with as few as five pages. This serves as further justification to our design decision of only scraping 20 pages at most from each website.

With all of these experiments in mind, we selected a thresholding value of 0.99 for our system, and eliminated from our filtering process sites where we could only scrape less than five pages.

4.3 Feature extraction

We derived the set of features used in classification from two main sources, the Alexa Web Information Service

Feature	**Discretization fn.**	**Values**
AWIS Site Rank	$\lceil \log(\text{SiteRank}) \rceil$	$[0\ldots8]$
Links to the site	$\lceil \log(\text{LinksIn}) \rceil$	$[0\ldots7]$
Load percentile	$\lfloor \text{LoadPercentile}/10 \rfloor$	$[0\ldots10]$
Adult site?	(Boolean)	$\{0,1\}$
Reach per million	$\lceil \log(\text{reachPerMillion}) \rceil$	$[0\ldots5]$

Table 1: **AWIS features used in our classifier and their discretization.** Those features are static—i.e., they are used whenever available.

(AWIS) and the content of the saved web pages. Features generated from the pages content are dynamically generated during the learning process according to a chosen statistic. For the classification process we use an ensemble of decision trees so that the input features should be discrete. In order to obtain useful discrete features (i.e., discrete features that do not take on too many values relative to the number of examples), we apply a mapping process to convert continuous quantities (load percentile) and large discrete quantities (global page rank) to a useful set of discrete quantities. The mappings used are shown in Table 1.

4.3.1 AWIS features

For every site that was scraped, we downloaded an entry from AWIS on Feb 2, 2014. While the date of the scrape does not match the date of the web ranking information, it can still provide tremendous value in helping to establish the approximate popularity of a site. Indeed, we observed that in the overwhelming majority of cases, the ranking of sites and the other information provided does not change significantly over time; and after discretization does not change at all. This mismatch is not a fundamental consequence of the experiment design but rather a product retrospectively obtaining negative training examples (sites which did not become compromised) which can be done in real time.

Intuitively, AWIS information may be useful because attackers may target their resources toward popular hosts running on powerful hardware. Adversaries which host malicious sites may have incentives to make their own malicious sites popular. Additionally, search engines are a powerful tool used by attackers to find vulnerable targets (through, e.g., “Google dorks [18]”) which causes a bias toward popular sites.

We summarize the AWIS features used in Table 1. An AWIS entry contains estimates of a site’s global and regional popularity rankings. The entry also contains estimates of the reach of a site (the fraction of all users that are exposed to the site) and the number of other sites which link in. Additionally, the average time that it takes users to load the page and some behavioral measurements such as page views per user are provided.

The second column of Table 1 shows how AWIS information is discretized to be used as a feature in a decision-tree classifier. Discretization groups a continuous feature such as load percentile or a large discrete feature such as global page rank into a few discrete values which make them more suitable for learning. If a feature is continuous or if too many discrete values are used, then the training examples will appear sparse in the feature space and the classifier will see new examples as being unique instead of identifying them as similar to previous examples when making predictions.

For many features such as AWIS Site Rank, a logarithm is used to compress a large domain of ranks down to a small range of outputs. This is reasonable since for a highly ranked site, varying by a particular number of rankings is significant relative to much lower ranked site. This is because the popularity of sites on the Internet follows a long tailed distribution [9].

Dealing with missing features. Some features are not available for all sites, for example information about the number of users reached by the site per million users was not present for many sites. In these cases, there are two options. We could reserve a default value for missing information; or we could simply not provide a value and let the classifier deal with handling missing attributes.

When a decision-tree classifier encounters a case of a missing attribute, it will typically assign it either the most common value for that attribute, the most common value given the target class of the example (when training), or randomly assign it a value based on the estimated distribution of the attribute. In our particular case, we observed that when a feature was missing, the site also tended to be extremely unpopular. We asserted that in these cases, a feature such as reach per million would probably also be small and assigned it a default value. For other types of missing attributes such as page load time, we did not assign the feature a default value since there is likely no correlation between the true value of the attribute and its failure to appear in the AWIS entry.

4.3.2 Content-based features

The content of pages in the observed sites provides extremely useful information for determining if the site will become malicious. Unlike many settings where learning is applied, the distribution of sites on the web and the attacks that they face vary over time.

Many Internet web hosts are attacked via some exploit to a vulnerability in a content-management system (CMS), that their hosted site is using. It is quite common for adversaries to enumerate vulnerable hosts by looking for CMSs that they can exploit. Different CMSs and even different configurations of the same CMS leak information about their presence through content such as tags associated with their template, meta tags, and comments. The set of CMSs being used varies over time: new CMSs are released and older ones fall out of favor, as a result, the page content that signals their presence is also time varying.

To determine content-based features, each of the pages that survived the acquisition and filtering process described earlier was parsed into a set of HTML tags. Each HTML tag was represented as the tuple (type, attributes, content). The tags from all the pages in a site were then aggregated into a list *without repetition*. This means that duplicate tags, i.e., tags matching precisely the same type, attributes and content of another tag were dropped. This approach differs from that taken in typical document classification techniques where the document frequency of terms is useful, since for example a document that uses the word "investment" a dozen times may be more likely to be related to finance than a document that uses it once. However, a website that has many instances of the same tag may simply indicate that the site has many pages. We could balance the number of occurrences of a tag by weighting the number of pages used; but then, relatively homogeneous sites where the same tag appears on every page would give that tag a high score while less homogeneous sites would assign a low score. As a result, we chose to only use the existence of a tag within a site but not the number of times the tag appeared.

During the training phase, we then augmented the lists of tags from each site with the sites' classification; and added to a dictionary which contains a list of all tags from all sites, and a count of the number of positive and negative sites a particular tag has appeared in. This dictionary grew extremely quickly; to avoid unwieldy increase in its size, we developed the following heuristic. After adding information from every 5,000 sites to the dictionary, we purged from the dictionary all tags that had appeared only once. This heuristic removed approximately 85% of the content from the dictionary every time it was run.

Statistic-based extraction. The problem of feature extraction reduces to selecting the particular tags in the dictionary that will yield the best classification performance on future examples. At the time of feature selection, the impact of including or excluding a particular feature is unknown. As a result, we cannot determine an optimal set of features at that time. So, instead we use the following technique. We fix a number N of features we want to use. We then select a statistic $\hat{s}$, and, for each tag t in the dictionary, we compute its statistic $\hat{s}(t)$. We then simply take the top-N ranked entries in the dictionary according to the statistic $\hat{s}$.

Many statistics can be used in practice [12]. In our implementation, we use $N = 200$, and $\hat{s}$ to be ACC2. ACC2 is the balanced accuracy for tag x, defined as:

$$\hat{s}(x) = \left| \frac{|\{x : x \in w, w \in \mathscr{M}\}|}{|\mathscr{M}|} - \frac{|\{x : x \in w, w \in \mathscr{B}\}|}{|\mathscr{B}|} \right| ,$$

where $\mathscr{B}$ and $\mathscr{M}$ are the set of benign, and malicious websites, respectively; the notation $x \in w$ means (by a slight abuse of notation) that the tag x is present in the tag dictionary associated with website w. In essence, the statistic computes the absolute value of the difference between the tag frequency in malicious pages and the tag frequency in benign pages.

A key observation is that these top features can be periodically recomputed in order to reflect changes in the statistic value that occurred as a result of recent examples. In our implementation, we recomputed the top features every time that the decision tree classifiers in the ensemble are trained.

As the distribution of software running on the web changes and as the attacks against websites evolve, the tags that are useful for classification will also change. A problem arises when the dictionary of tags from previous examples is large. For a new tag to be considered a top tag, it needs to be observed a large number of times since $|\mathscr{M}|$ and $|\mathscr{B}|$ are very large. This can mean that there is a significant delay between when a tag becomes useful for classification and when it will be selected as a top feature, or for example in the case of tags associated with unpopular CMSs, which will never be used.

A way of dealing with this problem is to use windowing, where the dictionary of tags only contains entries from the last K sites. By selecting a sufficiently small window, the statistic for a tag that is trending can rapidly rise into the top N tags and be selected as a feature. The trade-off when selecting window size is that small window sizes will be less robust to noise but faster to capture new relevant features while larger windows will be more robust to noise and slower to identify new features.

An additional strategy when calculating the statistic value for features is to weight occurrences of the feature differently depending on when they are observed. With windowing, all observations in the window are weighted equally with a coefficient of 1, and all observations outside of the window are discarded by applying a coefficient of 0. Various functions such as linear and exponential may be used to generate coefficients that scale observations and grant additional emphasis on recent observations of a feature.

5 Experimental results

We evaluate here both our dynamic feature extraction algorithm, and the overall performance of our classifier, by providing ROC curves.

Feature	Stat.
meta{*'content': 'WordPress 3.2.1', 'name': 'generator'*}	0.0569
ul{*'class': ['xoxo', 'blogroll']*}	0.0446
You can start editing here.	0.0421
meta{*'content': 'WordPress 3.3.1', 'name': 'generator'*}	0.0268
/all in one seo pack	0.0252
span{*'class': ['breadcrumbs', 'pathway']*}	0.0226
If comments are open, but there are no comments.	0.0222
div{*'id': 'content_disclaimer'*}	0.0039

Table 2: **Selection of the top features after processing the first 90,000 examples.** These features are a chosen subset of the top 100 features determined by the system after 90,000 examples had been observed and using windowing with a window size of 15,000 examples and linear attenuation.

5.1 Dynamic Feature Extraction

We analyzed dynamic features by logging the values of the statistic AAC2 after adding every example to the system. We selected a few particular features from a very large set of candidates to serve as examples and to guide intuition regarding dynamic feature extraction. The process of feature extraction could be performed independently of classification and was run multiple times under different conditions to explore the effect of different parameters such as the use of windowing and attenuation.

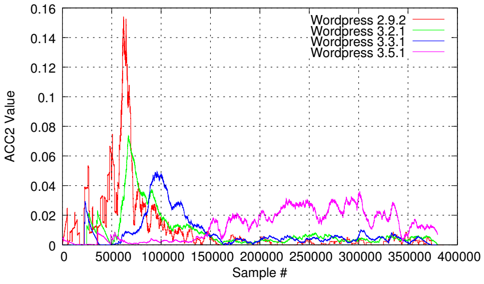


Figure 8: **Statistic value for various tags corresponding to different version of the Wordpress content management system.**

Dynamic feature extraction is essential, as it allows the system to automatically identify features useful in predicting if a domain will become malicious; this automation is imperative in a concept-drifting domain. For example, Figure 8 shows the computed statistic value for various features that correspond directly to different versions of the Wordpress CMS. Over time, the usefulness of different features changes. In general, as new versions of a CMS are released, or new exploits are found for existing ones, or completely new CMSs are developed, the set of the features most useful for learning will be constantly evolving.

Table 2 shows a selection of the 200 tags with highest statistic value after 90,000 examples had been passed to the system using a window size of 15,000 examples and a linear weighting scheme. A meaningful feature, i.e., with a large statistic value, is either a feature whose presence is relatively frequent among examples of malicious sites, or whose presence is frequent among benign sites. Of the 15,000 sites in the window used for generating the table, there were 2,692 malicious sites, and 12,308 benign ones. The feature *ul*{*'class': ['xoxo', 'blogroll']*} was observed in 736 malicious sites and 1,027 benign ones (461.34 malicious, 538.32 benign after attenuation) making it relatively more frequent in malicious sites. The feature *div*{*'id': 'content_disclaimer'*} was observed in no malicious sites and 62 benign ones (47.88 benign after attenuation) making it more frequent in benign sites. After manual inspection, we determined that this feature corresponded to a domain parking page where no other content was hosted on the domain.

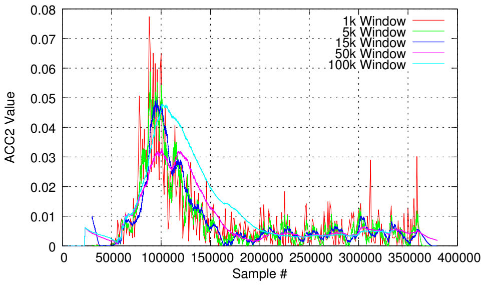


Figure 9: **Statistic value for *meta*{*'content': 'WordPress 3.3.1', 'name': 'generator'*} over time.** The statistic was computed over the experiment using window sizes of 1,000, 5,000, 15,000, 50,000 and 100,000 samples and uniform weighting.

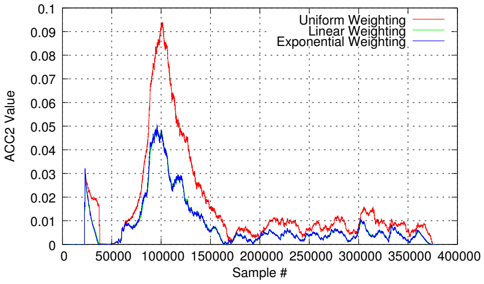


Figure 10: **Statistic value for *meta*{*'content': 'WordPress 3.3.1', 'name': 'generator'*} over time.** The statistic was computed over the experiment using a window size of 15,000 samples and various weighting techniques.

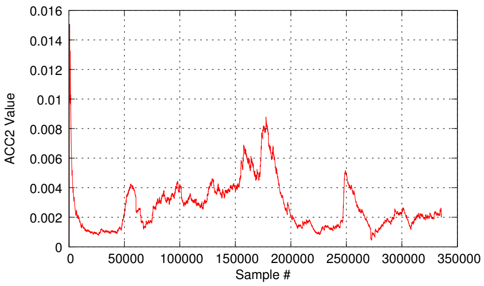


Figure 11: **Statistic value for *div*{*'id': 'content_disclaimer'*} over time.** The statistic was computed over the experiment using a window of size 15,000 samples and linear attenuation.

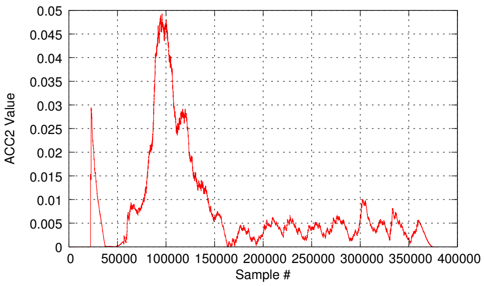


Figure 12: **Statistic value for *meta*{*'content': 'WordPress 3.3.1', 'name': 'generator'*} over time.** The statistic was computed over the experiment using a window of size 15,000 samples and linear attenuation.

The calculation of the ACC2 statistic for a feature at a particular time is parameterized by the window size and by a weighting scheme. As an example, Figure 9 shows the value of the statistic computed for the tag *meta*{*'content': 'WordPress 3.3.1', 'name': 'generator'*} over the experiment using different window sizes. When using a window, we compute the statistic by only considering examples that occurred within that window. We made passes over the data using window sizes of 1,000, 5,000, 15,000, 50,000 and 100,000 samples, which approximately correspond to 3 days, 2 weeks, 7 weeks, 24 weeks, and 48 weeks respectively.

A small window size generates a statistic value extremely sensitive to a few observations whereas a large window size yields a relatively insensitive statistic value. The window size thus yields a performance trade-off. If the statistic value for a feature is computed with a very small window, then the feature is prone to being incorrectly identified as meaningful, but will correctly be identified as meaningful with very low latency as only a few observations are needed. A large window will result in less errors regarding the usefulness of a feature but will create a higher latency.

Figure 10 shows the effect of varying the weighting scheme with a constant window size. Using a weighting scheme gives higher weight to more recent examples and the effect is very similar to simply decreasing the window size. There is almost no difference between exponential and linear decay.

Features belonging to positive (malicious) and negative (benign) examples often carry with them their own characteristics. The statistic values of negative examples tend to be relatively constant and time-invariant as the example in Figure 11 shows. These are generally features that indicate a lack of interesting content and therefore a lack of malicious content—for instance, domain parking pages. Conversely, the statistic value of positive examples tend to contain a large spike as evidenced by the example in Figure 12. The features correspond to vulnerable software and spike when an attack campaign exploiting that vulnerability is launched. Occasionally, additional spikes are observed, presumably corresponding to subsequent campaigns against unpatched software.

A design consideration when working with dynamic features is whether or not it is appropriate to use features that were highly ranked at some point in the past in addition to features that are currently highly ranked. As discussed above, negative features tend to be relatively constant and less affected, unlike positive features which fluctuate wildly. These positive features tend to indicate the presence of software with a known vulnerability that may continue to be exploited in the future.

Since it may happen that a feature will be useful in the future, as long as computational resources are available, better classification performance can be achieved by including past features in addition to the current top performing features. The result of including past features is that in situations where attack campaigns are launched against previously observed CMSs, the features useful for identifying such sites do not need to be learned again.

5.2 Classification performance

We ran the system with three different configurations to understand and evaluate the impact that different configurations had on overall performance. We send input to our ensemble of classifiers as “blocks,” i.e., a set of websites to be used as examples. The first configuration generated content features from the very first block of the input stream but did not recompute them after that. The second configuration recomputed features from every block in the input stream but did not use past features which did not currently have a top statistic value. The third configuration used dynamic features in addition to all features that had been used in the past.

For all configurations, we used a block size of 10,000 examples for retraining the ensemble of C4.5 classifiers.

We also used a window size of 10,000 samples when computing the statistic value of features, and we relied on features with the top 100 statistic values. We generated ROC curves by oversampling the minority class by 100% and 200% and undersampling the majority class by 100%, 200%, 300%, and 500%. We ran each combination of over- and undersampling as its own experiment, resulting in a total of 10 experiments for each configuration. The true positive rate and false positive rate[2] for each experiment is taken as the average of the true positive and false positive rates for each block, that is, each block in the input stream to the system is tested on before being trained on, and the rates are taken as the average over the tested blocks.

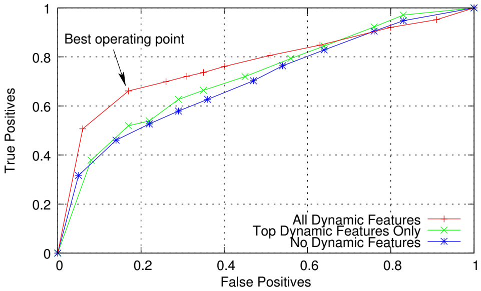


Figure 13: **ROC plot for three different strategies of dynamic features.** The classifier was run using three different configurations for dynamic features. The first configuration corresponds to classifiers trained on both current and past top features; the second corresponds to classifiers trained using only current top features; the third corresponds to classifiers trained using the top features from the first 5,000 samples.

Figure 13 shows the ROC curves generated for the three configurations described. The points resulting from the experiments have been linearly connected to form the curves. One can see that the configuration which used past features performed the best, followed by the configuration which used only current top dynamic features and the configuration which did not use dynamic features at all. The best operating point appears to achieve a true positive rate of 66% and a false positive rate of 17%.

The configuration which did not use dynamic features ended up selecting a feature set which was heavily biased by the contents of first block in the input data stream. While the features selected were useful on learning the first block, they did not generalize well to future examples since the distribution of pages that were observed had changed. This is a problem faced by all such systems in this setting that are deployed using a static set of features, unless the features set is fully expressive of the page content, i.e., all changes in the page content are able to be uniquely identified by a corresponding change in the feature values, then the features will eventually become less useful in classification as the distribution of pages changes.

The configuration which only used the current top dynamic features also performed relatively poorly. To understand why this is the case, we can see that in Figures 11 and 12 some features have a statistic value which oscillates to reflect the change in usefulness of the feature due to the time varying input distribution. One can also see that when a feature becomes useful, the corresponding increase in the statistic value lags behind since a few instances of the feature need to be observed before the statistic can obtain a high value again. During this transient period, the system fails to use features that would be useful in classification and so performance suffers. This problem may be partially addressed by shrinking the input block size from the data streams well as the window for computing the static value to a smaller value to reduce the transient. However such a strategy will still be outperformed by the strategy which remembers past features.

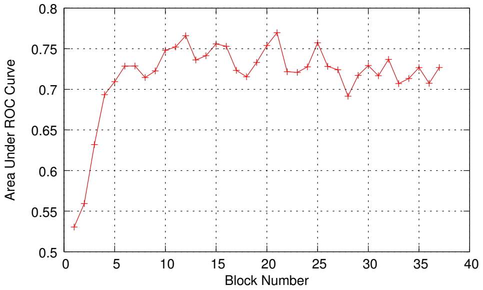


Figure 14: **AUC plot for the system over time using current and past dynamic features.** The system was run using both current and past top dynamic features. ROC curves were generated for each block of examples that was processed and the corresponding AUC value was computed.

For each input block in the experiments using past features, we recorded the true positive and false positive rates and used them to generate an ROC curve. We then used the ROC curve to approximate the area under the curve (AUC) which is a value that gives some intuitive understanding of how well the classifier performed on that block. Figure 14 shows the AUC values for each block in the experiment. The system performed relatively poorly until a sufficient number of blocks had been processed at which point the performance increased to a threshold value. We believe that the difficulty in achiev-

[2] The false negative rate and true negative rates are simple complements of the respective positive rates.

ing better performance is due to the nature of the problem, specifically it is not always the case that the content of a site and its traffic statistics are a factor in whether or not it will become compromised. We discuss this issue in more details in the limitations section.

Finally, we observed that when classification yielded a prediction that a site would become compromised, the reasoning can be read as the conjunction of conditions from the decision tree. For example the classification of `www.bisoft.org` which appeared in the search redirection data set was described as (Global Site Rank = 8) ∧ (*<META NAME="Generator" CONTENT="EditPlus">= 1*) ∧ (*<script type="text/javascript" src="/static/js/analytics.js">* = 1). The classification of benign examples was generally less obvious.

The conditions resulting in incorrect classification tended to follow a few main types. The first type are instances where a website would in the future become malicious, but very few examples like it exist at classification time. These sites contained content that would eventually yield prominent features for identifying sites that would become malicious but were not classified correctly due to latency in the dynamic feature extraction. As an example, consider in Figure 12 the samples that occurred just before the first significant spike.

The second type of instance that was incorrectly classified were examples that did not become malicious, but were classified as becoming so based on some strong positive content features that they contained. It is likely that after an initial attack campaign, vulnerable CMSs are less targeted due to the incremental number of compromises that could be yielded from them.

The third type of instance that was incorrectly classified were examples that would be become malicious for seemingly no apparent reason. These examples that would become malicious did not follow the general trend of large spikes corresponding to attack campaigns against a CMS, and have been observed with positive features both before and after its initial spike as well as with strong negative features. It is believed that these examples are cases where a site is becoming malicious for reasons completely independent of its content or traffic profile. It could be the case that an attack is launched where default login credentials for many CMSs are being attempted resulting in a few seemingly random breaks. It could also be the case that the domain in question was sold or rebuilt after observing it causing the system to erroneously predict its future malicious status from its old content.

6 Limitations

The limits on the classification performance of the system can be attributed to the following few difficulties in predicting if a site will become malicious.

Our system assumes the factors responsible for whether or not a site will become compromised can be summarized by its content and its traffic statistics. This assumption is sometimes violated, since for example sites may be compromised and become malicious due to weak administrator passwords being guessed or being retrieved via social engineering. Other examples may include adversaries who host their own sites with malicious intent. While it is often the case that such actors use similar page templates due to their participation in affiliate networks or out of convenience, such sites may introduce examples where the factors for the site being malicious are independent of its content. In such situations, the system will fail to perform well since the factors for site becoming malicious are outside its domain of inputs.

The nature of adversaries who compromise sites may also be perceived as a limitation on what our system can do. It has been observed that attack campaigns are launched where adversaries appear to enumerate and compromise sites containing a similar vulnerability. While adversaries do attack many sites which contain a particular vulnerability, it is generally not a reasonable assumption that they will systematically attack all sites containing this vulnerability both at the time of the campaign and in the future. The impact of this behavior on the system is that sites which contain similar content to those which were compromised in the campaign will be classified as becoming malicious in the future, when they actually may not since attackers have chosen to ignore them. While this does deteriorate the performance of the system, we argue that this does not take away its usefulness since these misclassifications represent sites which are still at considerable security risk and need attention.

The dynamic feature extraction system also presents at least two main limitations. The first is a correlation of features that are selected as top features at any given point in time. Tags often rise to the top of the list because they are part of some page template which has come up frequently. There may be multiple tags associated with a particular page template which all rise at the same time, and so a few of the top tags are redundant since they are identifying the same thing. It would be desirable to measure the correlation of the top features in order to select a more diverse and useful set however no attempt to do this was made in our experiments.

Another limitation of the dynamic features is that for system configurations which use past features in addition to the current top features, the size of the feature set is monotonically increasing. Thus, it will take longer over time train the classifiers and run the system. It would be useful to further investigate the trade-offs between classification performance and limited feature lists.

Last, dynamic features introduce a unique opportunity for adversarial machine learning approach to poison the performance of the system. Adversaries which control a website may attempt to remove, change, or insert tags into their pages in order to damage the effectiveness of feature generation. For example, adversaries that host or control sites that have distinguishing tags may either try to remove them or rewrite them in semantically equivalent ways to prevent the system from using them for classification. Since the sites examined by the system are typically not under adversarial control at the time of evaluation, we believe that the impact of such attacks should be minimal; but it deserves further analysis.

7 Conclusions

We discussed a general approach for predicting a websites propensity to become malicious in the future. We described a set of desirable properties for any solution to this problem which are interpretability, efficiency, robustness to missing data, training errors, and class imbalance, as well as the ability to adapt to time changing concepts. We then introduced and adapted a number of techniques from the data mining and machine learning communities to help solve this problem, and demonstrated our solution using an implementation of these techniques. Our implementation illustrates that even with a modest dataset, decent performance can be achieved since we are able to operate with 66% true positives and only 17% false positives at a one-year horizon. We are currently working on making our software publicly available.

Acknowledgments

We thank our anonymous reviewers for feedback on an earlier revision of this manuscript, Brewster Kahle and the Internet Archive for their support and encouragement, and Jonathan Spring at CERT/SEI for providing us with historical blacklist data. This research was partially supported by the National Science Foundation under ITR award CCF-0424422 (TRUST) and SaTC award CNS-1223762; and by the Department of Homeland Security Science and Technology Directorate, Cyber Security Division (DHS S&T/CSD), the Government of Australia and SPAWAR Systems Center Pacific via contract number N66001-13-C-0131. This paper represents the position of the authors and not that of the aforementioned agencies.

References

[1] Alexa Web Information Service. http://aws.amazon.com/awis/.

[2] DNS-BH: Malware domain blocklist. http://www.malwaredomains.com/.

[3] Norton safe web. http://safeweb.norton.com.

[4] Scrapy: An open source web scraping framework for Python. http://scrapy.org.

[5] Stop badware: A nonprofit organization that makes the Web safer through the prevention, remediation and mitigation of badware websites. https://www.stopbadware.org/.

[6] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *Proc. RAID'07*, pages 178–197, Gold Coast, Australia, 2007.

[7] U. Bayer, P. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *Proc. NDSS'09*, San Diego, CA, February 2009.

[8] K. Borgolte, C. Kruegel, and G. Vigna. Delta: automatic identification of unknown web-based infection campaigns. In *Proc. ACM CCS'13*, pages 109–120, Berlin, Germany, November 2013.

[9] L. Breslau, P. Cao, L. Fan, G. Philips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proc. IEEE INFOCOM'99*, pages 126–134, New York, NY, March 1999.

[10] D. Chakrabarti, R. Kumar, and K. Punera. Page-level template detection via isotonic smoothing. In *Proc. WWW'07*, pages 61–70, Banff, Canada, May 2007.

[11] S. Debnath, P. Mitra, N. Pal, and C.L. Giles. Automatic identification of informative sections of web pages. *IEEE Transactions on Knowledge and Data Engineering*, 17(9):1233–1246, 2005.

[12] G. Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of machine learning research*, 3:1289–1305, 2003.

[13] J. Gao, W. Fan, J. Han, and P. Yu. A general framework for mining concept-drifting data streams with skewed distributions. In *Proc. SIAM SDM'07*, pages 3–14, Mineapolis, MN, April 2007.

[14] Google. Google Safe Browsing API. https://code.google.com/apis/safebrowsing/.

[15] L. Invernizzi, P. Comparetti, S. Benvenuti, C. Kruegel, M. Cova, and G. Vigna. Evilseed: A guided approach to finding malicious web pages. In *Proc. 2012 IEEE Symp. Sec. & Privacy*, pages 428–442, San Francisco, CA, May 2012.

[16] J. Jang, D. Brumley, and S. Venkataraman. Bitshred: Feature hashing malware for scalable triage and semantic analysis. In *Proc. ACM CCS'11*, Chicago, IL, October 2011.

[17] J. John, F. Yu, Y. Xie, M. Abadi, and A. Krishnamurthy. deSEO: Combating search-result poisoning. In *Proc. USENIX Security'11*, San Francisco, CA, August 2011.

[18] L. Lancor and R. Workman. Using Google hacking to enhance defense strategies. *ACM SIGCSE Bulletin*, 39(1):491–495, 2007.

[19] N. Leontiadis, T. Moore, and N. Christin. A nearly four-year longitudinal study of search-engine poisoning. Tech. Rep. CyLab-14-008, Carnegie Mellon University. July 2014.

[20] N. Leontiadis, T. Moore, and N. Christin. Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade. In *Proc. USENIX Security'11*, San Francisco, CA, August 2011.

[21] K. Levchenko, N. Chachra, B. Enright, M. Felegyhazi, C. Grier, T. Halvorson, C. Kanich, C. Kreibich, H. Liu, D. McCoy, A. Pitsillidis, N. Weaver, V. Paxson, G. Voelker, and S. Savage. Click trajectories: End-to-end analysis of the spam value chain. In *Proc. 2011 IEEE Symp. Sec. & Privacy*, Oakland, CA, May 2011.

[22] L. Lu, R. Perdisci, and W. Lee. SURF: Detecting and measuring search poisoning. In *Proc. ACM CCS 2011*, Chicago, IL, October 2011.

[23] MalwareBytes. hphosts online. `http://www.hosts-file.net/`.

[24] McAfee. Site Advisor. `http://www.siteadvisor.com/`.

[25] D. McCoy, A. Pitsillidis, G. Jordan, N. Weaver, C. Kreibich, B. Krebs, G. Voelker, S. Savage, and K. Levchenko. Pharmaleaks: Understanding the business of online pharmaceutical affiliate programs. In *Proc. USENIX Security'12*, Bellevue, WA, August 2012.

[26] J. P. McDermott. Attack net penetration testing. In *Proceedings of the 2000 workshop on New security paradigms - NSPW '00*, pages 15–21, New York, New York, USA, 2000.

[27] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher. *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall PTR, 2004.

[28] PhishTank. `https://www.phishtank.com/`.

[29] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose. All your iFrames point to us. In *Proc. USENIX Security'08*, San Jose, CA, August 2008.

[30] Foster Provost and Tom Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231, 2001.

[31] J. R. Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan Kaufmann, 1993.

[32] Ruihua Song, Haifeng Liu, Ji-Rong Wen, and Wei-Ying Ma. Learning block importance models for web pages. In *Proc. WWW'04*, pages 203–211, New York, NY, May 2004.

[33] Sophos. Security threat report 2013, 2013. `http://www.sophos.com/en-us/medialibrary/PDFs/other/sophossecuritythreatreport2013.pdf`.

[34] The Internet Archive. Wayback machine. `https://archive.org/web/`.

[35] M. Vasek and T. Moore. Identifying Risk Factors for Webserver Compromise. In *Proc. Financial Crypto.'14*, Accra Beach, Barbados, February 2014.

[36] D. Wang, G. Voelker, and S. Savage. Juice: A longitudinal study of an SEO botnet. In *Proc. NDSS'13*, San Diego, CA, February 2013.

[37] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 01:69–101, 1996.

[38] L. Yi, B. Liu, and X. Li. Eliminating noisy information in web pages for data mining. In *Proc. ACM KDD'03*, pages 296–305, Washington, DC, August 2003.

[39] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proc. WWW'05*, pages 76–85, Chiba, Japan, May 2005.