Learning to Detect Malicious URLs

JUSTIN MA, University of California, Berkeley
LAWRENCE K. SAUL, STEFAN SAVAGE and GEOFFREY M. VOELKER, University of California, San Diego

30

Malicious Web sites are a cornerstone of Internet criminal activities. The dangers of these sites have created a demand for safeguards that protect end-users from visiting them. This article explores how to detect malicious Web sites from the lexical and host-based features of their URLs. We show that this problem lends itself naturally to modern algorithms for online learning. Online algorithms not only process large numbers of URLs more efficiently than batch algorithms, they also adapt more quickly to new features in the continuously evolving distribution of malicious URLs. We develop a real-time system for gathering URL features and pair it with a real-time feed of labeled URLs from a large Web mail provider. From these features and labels, we are able to train an online classifier that detects malicious Web sites with 99% accuracy over a balanced dataset.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; I.5.1 [**Pattern Recognition**]: Models—*Statistical*

General Terms: Algorithms, Security

Additional Key Words and Phrases: Online learning, malicious Web sites

ACM Reference Format:
MA, J., Saul, L. K., Savage, S., and Voelker, G. M. 2011. Learning to Detect Malicious URLs. ACM Trans. Intell. Syst. Technol. 2, 3, Article 30 (April 2011), 24 pages.
DOI = 10.1145/1961189.1961202 http://doi.acm.org/10.1145/1961189.1961202

1. INTRODUCTION

From an early age we learn to manage personal risk; to infer which situations may be dangerous and avoid them accordingly. For example, most cities have a "rough part of town" that is understood to be more dangerous for casual visitors. However, this same notion translates poorly to the context of the Internet: there are few effective heuristics to differentiate safe Web sites from dangerous ones. Indeed, Internet criminals *depend* on the absence of such indicators to prey on their marks.

Criminal Web sites support a wide range of socially undesirable enterprises, including spam-advertised commerce (e.g., counterfeit watches or pharmaceuticals), financial fraud (e.g., via phishing or 419-type scams) and malware propagation (e.g., so-called "drive-by downloads"). Although the precise commercial motivations behind these schemes may differ, the common thread among them is the requirement that unsuspecting users visit their sites. These visits can be driven by email, Web search

This work was supported by National Science Foundation grants NSF-0238323, NSF-0433668 and NSF-0829469, by the Office of Naval Research MURI grant N000140911081, and by generous research, operational and in-kind support from Cisco, Google, Microsoft, Yahoo and the UCSD Center for Networked Systems.
Authors' addresses: J. Ma (corresponding author), Department of Electronic Engineering and Computer Science, University of California, Barkeley; email: jtma@eecs.barkeley.edu; L. K. Saul, S. Savage, G. M. Voelker, University of California, San Diego.

DOI 10.1145/1961189.1961202 http://doi.acm.org/10.1145/1961189.1961202

results, or links from other Web pages, but all require the user to take some action, such as clicking, that specifies the desired Uniform Resource Locator (URL). Thus, each time a user decides whether to click on an unfamiliar URL that user must implicitly evaluate the associated risk. Is it safe to click on that URL, or will it expose the user to potential exploitation? Not surprisingly, this can be a difficult judgment for individual users to make.

Aware of this difficulty, security researchers have developed various systems to protect users from their uninformed choices. By far the most common technique, deployed in browser toolbars, Web filtering appliances, and search engines, is "blacklisting." In this approach, a third-party service compiles the names of "known bad" Web sites (labeled by combinations of user feedback, Web crawling, and heuristic analysis of site content) and distributes the list to its subscribers. While such systems have minimal query overhead (just searching for a URL within the list) they can only offer incomplete protection because no blacklist is perfectly comprehensive and up-to-date [Sinha et al. 2008]. Thus, a user may click on a malicious URL before it appears on a blacklist (if it ever does). Alternatively, some systems also intercept and analyze each Web site's full content as it is downloaded. Though this analysis may detect undesirable sites with higher accuracy, it incurs far more runtime overhead than querying a blacklist; it may also inadvertently expose users to the very threats they seek to avoid.

In this article, we pursue a different approach, analyzing the URL itself to *predict* whether a Web site contains undesirable content. In particular, we make predictions from the lexical and host-based features of URLs *without* examining the actual content of Web pages. We are motivated by prior studies suggesting that malicious URLs exhibit certain common distinguishing features [Chou et al. 2004; McGrath and Gupta 2008]. Once automated, our approach attempts to improve on blacklisting while providing a lightweight, low-overhead alternative to systems that download and analyze the full content of Web pages. The resulting classifiers may also be used as a low-cost prefilter for more expensive techniques. One important contribution of this article is to demonstrate that the lexical and host-based features of URLs contain a wealth of information for detecting malicious Web sites. Indeed, our best-performing systems for URL classification extract and analyze *millions* of these features.

Learning algorithms for classification can operate in either batch or online settings. Batch algorithms only adapt the classifier after making a pass over the entire dataset of training examples. In contrast, online algorithms incrementally adapt the classifier after processing individual training examples.

The first studies of URL classification focused on the batch setting [Kan and Thi 2005; Garera et al. 2007; Ma et al. 2009a]. A second important contribution of this article is to demonstrate the power of the online setting. Online algorithms for URL classification have two important advantages. First, they can process large numbers of examples far more efficiently than batch methods. Second, they can more rapidly adapt to the evolving distribution of features that characterize malicious URLs over time.

To realize these advantages, we have built a URL classification system that processes a live feed of labeled URLs and collects features for these URLs in *real time* (see Figure 4). The feed of labeled URLs was supplied to us by a large Web mail provider. For our application, we show that online methods learn more accurate classifiers than batch methods because they are able to train on more data. Experimenting with different online algorithms, we obtain our best results (up to 99% accuracy over a balanced dataset) using a recently proposed method for confidence-weighted learning [Dredze et al. 2008]. We only achieve this level of performance in the online setting: the best-performing classifiers are those that continuously adapt to new features appearing in the live feed of labeled URLs. While our initial results with online learning were

presented in earlier work [Ma et al. 2009b], in this article we provide a much more complete description of our system.

The rest of the article begins by introducing the problem of URL classification and reviewing the online algorithms that we implemented for our experiments. Next we provide details of our system implementation, particularly the components for data collection and feature extraction. Finally, we present our experimental results and conclude with an overall discussion.

2. APPLICATION

Uniform Resource Locators (URLs), sometimes known as "Web links," are the primary means by which users locate resources on the Internet. Our goal is to detect malicious Web sites from the lexical and host-based features of their URLs. This section provides an overview of the problem, background on URL resolution, and discussion of the features we use for our application.

2.1. Problem Overview

For our purposes, we treat URL reputation as a binary classification problem where positive examples are malicious URLs and negative examples are benign URLs. This learning-based approach to the problem can succeed if the distribution of feature values for malicious examples is different from benign examples, the ground-truth labels for the URLs are correct, and the training set shares the same feature distribution as the testing set. (Later, we reexamine the last assumption, noting that new features for URL classification are introduced over time as shown in Figure 3.)

Significantly, we classify sites based only on the relationship between URLs and the lexical and host-based features that characterize them, and we do not consider two other kinds of potentially useful sources of information for features: the URL's page content, and the context of the URL (e.g., the page or email in which the URL is embedded). Although this information has the potential to improve classification accuracy, we exclude it for a variety of reasons. First, avoiding downloading page content is strictly safer for users. Second, classifying a URL with a trained model is a lightweight operation compared to first downloading the page and then using its contents for classification. Third, focusing on URL features makes the classifier applicable to any context in which URLs are found (Web pages, email, chat, calendars, games, etc.), rather than dependent on a particular application setting. Finally, reliably obtaining the malicious version of a page for both training and testing can become a difficult practical issue. Malicious sites have demonstrated the ability to "cloak" the content of their Web pages, that is, serving different content to different clients [Niu et al. 2007]. For example, a malicious server may send benign versions of a page to honeypot IP addresses that belong to security practitioners, but send malicious versions to other clients. Nonetheless, we show in Section 6 that classifying on lexical features of the URL and features about the host are sufficient for highly accurate prediction.

2.2. Background on URL Resolution

Just as we use filenames to locate files on a local computer, we use Uniform Resource Locators (URLs) to locate Web sites and individual Web resources. One way users visit a site is by typing a URL into the browser's address bar. An arguably easier way is to click a link which is contained within a page that is already rendered by the browser, or an email message rendered by an email client. In either case, the link contains the URL of the desired site. Because sites link to other sites, the network of links resembles a spider web (hence the naming of the Web).

URLs are human-readable text strings that are parsed in a standard way by client programs. Through a multistep resolution process, browsers translate each URL into

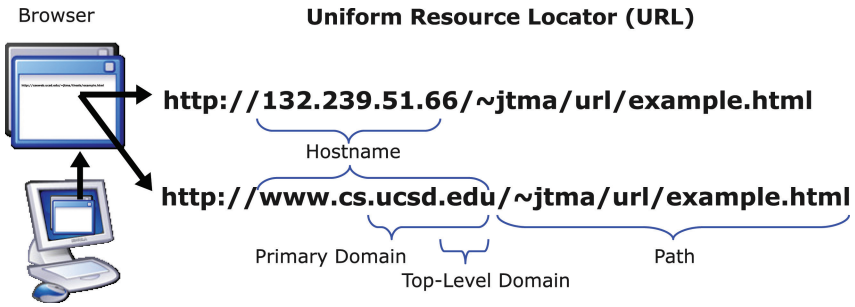


Fig. 1. Example of a Uniform Resource Locator (URL) and its components.

instructions that locate the server hosting the site and specify where the site or resource is placed on that host. To facilitate this machine translation process, URLs have the following standard syntax.

```
<protocol>://<hostname><path>
```

The `<protocol>` portion of the URL indicates which network protocol should be used to fetch the requested resource. The most common protocols in use are Hypertext Transport Protocol or HTTP (`http`), HTTP with Transport Layer Security (`https`), and File Transfer Protocol (`ftp`). In Figure 1, all of the example URLs specify the HTTP protocol. Although we do not include the protocol as a classification feature (most of the URLs we encounter in our training sources use `http`), we observe that phishing exploits often insert tokens such as `http` into the URL's path to trick the user into thinking that the URL is legitimate. For example, if we own `maliciousssite.com` but want to trick the user into thinking they are visiting `www.cs.ucsd.edu`, then we may construct a URL such as `http://maliciousssite.com/http:/www.cs.ucsd.edu`.

The `<path>` of a URL is analogous to the path name of a file on a local computer. In Figure 1, the path in the example is `/~jtma/url/example.html`. The path tokens, delimited by various punctuation such as slashes, dots, and dashes, show how the site is organized. However, criminals sometimes obscure path tokens to avoid scrutiny, or they may deliberately construct tokens to mimic the appearance of a legitimate site, as in the case of phishing.

The `<hostname>` is the identifier for the Web server on the Internet. Sometimes it is a machine-readable Internet Protocol (IP) address, but more often (especially from the user's perspective) it is a human-readable domain name.

IP is a network protocol that makes it possible for a host to send network packets to another host on the Internet, regardless of the underlying networking hardware. A key principle behind IP is that all hosts on the Internet have an IP address enabling them to reach one another. In IP version 4 (IPv4) [USC Information Sciences Institute 1981], addresses are 32-bit integers that are typically represented as a dotted quad. In dotted quad notation, we divide the 32-bit address into four 8-bit bytes. Then, from most to least significant byte, we print each byte in decimal notation delimited by dots (e.g., 132.239.51.66 in Figure 1). This ordering of the bytes is significant because it lets us represent hierarchies in the address space using IP prefix notation (Section 2.2.3). Other representations such as hexadecimal, dotted quad of hexadecimal numbers, dotted quad of octal numbers, and binary are interchangeable with the dotted quad of decimal numbers we described. However, we restrict our attention to the dotted quad of decimal numbers because it is the most popular representation. Finally, we restrict our attention to IPv4 because the newer IP version 6 [Deering and Hinden 1998] (with 128-bit addresses) is not as widely deployed at the moment.

Since hostnames represented as IP addresses do not convey high-level information about the host (such as which organization it belongs to), URLs typically employ human-readable domain names instead. A domain name is a series of text tokens delimited by dots "."; they describe the hostname's place in a multilevel naming hierarchy. The rightmost token indicates the domain name's major categorization, for example, `.com` and `.edu` for generic domain names, and `.cn` and `.uk` for international domain names. Moving from right-to-left, each token identifies the hostname at progressively lower levels of the hierarchy (known as "subdomains") and provides a blueprint for translating the domain name into an IP address. For example, the domain `www.cs.ucsd.edu` from Figure 1 represents a host (`www.cs.ucsd.edu`) contained within the Computer Science and Engineering department (`cs.ucsd.edu`) at UC San Diego (`ucsd.edu`), whose domain registration is handled by EDUCAUSE (the `.edu` registrar). To better understand the mechanisms behind the domain name, we describe how a name is translated into an IP address (Section 2.2.1) and how a name is established (Section 2.2.2).

Having described URLs at a high level, we now provide further background on the mechanisms for constructing, resolving, and hosting a Web site's URL. We relate these mechanisms to the goal of detecting malicious URLs, hinting at their applications as useful indicators (features) for automated detection and deferring a detailed discussion about the features to Section 2.3.

2.2.1. Domain Name System Resolution. The Domain Name System (DNS) is a hierarchical network of servers responsible for translating domain names into IP addresses and other kinds of information [Mockapretis 1987a; 1987b]. During the DNS resolution process, the tokens in the domain name are traversed from right-to-left to direct the client's DNS resolver to query the appropriate DNS name servers.

Let us take the hostname `www.cs.ucsd.edu` from Figure 1 as an example. First, the resolver will query a DNS root server to resolve the `.edu` portion of the domain name. The response from the root server will be a Name Server (NS) record that directs the resolver to the `.edu` name server. Next, the resolver will query the `.edu` name server for the records corresponding to `ucsd.edu`; in return the resolver receives the address for the `ucsd.edu` name server. Then, a request to the `ucsd.edu` name server for `cs.ucsd.edu` would return the NS record pointing to the `cs.ucsd.edu` name server. Finally a query to the `cs.ucsd.edu` name server with a request containing `www.cs.ucsd.edu` would return an Address (A) record containing the IP address of `www.cs.ucsd.edu` (which in this case is 132.239.51.66 now, but may change over time). Alternatively, we could query the `cs.ucsd.edu` name server for the Mail Exchanger (MX) record containing the IP address of the mail server associated with the `cs.ucsd.edu` domain.

The A, MX, and NS records are IP addresses of hosts associated with a domain name. Under the hypothesis that the hosting infrastructure for malicious URLs is distinct from that for benign URLs, the various DNS records become useful differentiators. The A records for malicious Web servers might be hosted on compromised residential machines or in disreputable providers. The NS records would represent the DNS infrastructure that leads a victim to a malicious site; this infrastructure is also likely to be hosted on compromised machines or disreputable providers. Moreover, if the set of hosts responsible for hosting malicious sites are affiliated with the hosts that send spam for a mailing campaign, then the associated MX records would be reliable indicators of malice. Malicious sites tend to be hosted in bad places, as illustrated by the stories of the McColo, Troyak, and Group 3 hosting providers.

—McColo provided hosting for major botnets, which in turn were responsible for sending 41% of the world's spam just before McColo's takedown in November 2008 [Keizer 2008].

—Troyak and Group 3 were responsible for hosting 90 out of 249 command-and-control servers for the Zeus botnet before their takedown in March 2010 [McMillan 2010].

On a related note, there is a special DNS record type called the pointer (PTR) record. Its purpose is to enable reverse DNS lookups: given an IP address as a query, the associated domain name is returned. To perform a reverse lookup on `132.239.51.66`, a client queries the domain name `66.51.239.132.in-addr.arpa` (note the reverse byte order of the IP address) and receives `www.cs.ucsd.edu` as the PTR record response. The existence of a PTR record is a reliable indicator that the domain name is well-established; as such, the PTR record is a potentially useful classification feature.

Finally, although A, MX, and NS records show promise as classification features, individual IP addresses may be too fine-grain for characterizing malicious hosting activity. To address this problem, we describe the standard for representing an aggregate set of IP addresses in Section 2.2.3. Next, we cover how the binding between name server and domain name is established.

2.2.2. Domain Name Registration. Besides the IP addresses associated with a domain name, there is useful information associated with domain name registration.

Registration establishes which name servers are associated with a domain name. Typically, the registrant registers the primary domain name (a term we define shortly) with the registrar; the registrant is the owner of the domain name, and the registrar is the organization responsible for hosting the NS record that points to the primary domain's servers.

The Top-Level Domain (TLD) is the rightmost token in a domain name (e.g., `.com`, `.edu`, `.cn`, `.uk`). A registrar is usually in charge of a single TLD, although it is possible for a registrar to delegate that authority to other smaller registrars. The primary domain is the highest-level domain name that a registrant can register. It usually consists of the two rightmost tokens in the domain (e.g., `google.com`), although it may be the three rightmost tokens in international domain names (e.g., `google.co.uk`). In the Figure 1 example, `.edu` is the TLD and `ucsd.edu` is the primary domain.

The information associated with a domain name registration can indicate whether certain entities have a higher tendency of registering domains associated with malicious sites, as well as whether a site is newly registered and has yet to establish its credibility. This information includes vital data about the registrant, the registrar, date of registration, date of expiration, date of the latest update, and other attributes associated with the record; it is typically stored in databases accessible through the WHOIS query/response protocol [Daigle 2004]. WHOIS works as follows.

(1) The client contacts the registrar's WHOIS server with a name query.
(2) The registrar returns a plaintext response or redirects the query to a smaller registrar in charge of the designated WHOIS record.

The ad hoc elements of the WHOIS server infrastructure and plaintext response format stands lead to much longer query times than DNS (i.e., on the order of seconds), especially for queries to registrars with slower infrastructure. In light of these longer query times, some registrars put a quota on the number of WHOIS queries that a user can perform in a day. These quotas can affect the large-scale operation of online learning algorithms that must gather features in real time. Mindful of these issues in query overhead and rate limiting, we experimented with classifiers that either included or omitted WHOIS features in Section 4.2 of Ma et al. [2009a]. Although the inclusion of WHOIS features yielded the highest accuracy, the exclusion of WHOIS features still resulted in highly accurate classification.

2.2.3. Routing and Forwarding. As we alluded in Section 2.2.1, individual IP addresses are very fine-grain because there are more than 4 billion possible addresses; recording enough of them to characterize the location of malicious URLs can be overwhelming. Thus, it is beneficial for us to represent an IP address as belonging to a block of IP addresses. This more coarse-grain view of representing addresses is important because individual hosts can have address churn (e.g., Dynamic Host Configuration Protocol [Droms 1997]). Moreover, we may be able to identify aggregate behaviors if certain activities tend to occur in specific subsections of the Internet (see the McColo example in Section 2.2.1). Fortunately, we can use existing representations from Internet routing and forwarding for these address blocks.

Routing is the process of setting up paths along which IP packets are forwarded. Internet Service Providers (ISPs) negotiate these paths using the Border Gateway Protocol (BGP). In BGP, ISPs are also known as Autonomous Systems (ASes), which are identified by their AS numbers. Thus, we can treat AS numbers as de facto identifiers for ISPs: if an ISP is responsible for hosting a collection of malicious sites, then we can use the AS number as a salient feature (e.g., McColo's AS number was 26780).

Similarly, we can also treat IP prefixes as de facto identifiers for ISPs. During packet forwarding, routers must redirect incoming packets by looking up the next-hop router responsible for forwarding to the packet's destination address. Storing individual IP addresses in the forwarding table would consume a prohibitive amount of memory, especially for routers at the Internet core. However, routers save space by referencing groups of IP addresses using Classless InterDomain Routing (CIDR) aggregation [Fuller and Li 2006]. An IP prefix (or CIDR block) is a compact way of denoting a block of IP addresses. In Internet Protocol version 4 (IPv4), the notation `<address>/<prefix_len>` represents all IP addresses that share the `<prefix_len>` most significant bits with `<address>`. For example, `137.110.222.0/24` represents all IP addresses in the range `137.110.222.0`–`137.110.222.255`. Typically, authorities assign IP prefixes to ISPs, thereby making these prefixes useful identifiers of providers (for McColo, it was 208.66.192.0/22).

We note that the mapping from AS numbers to IP prefixes is not necessarily isomorphic. It is possible for an AS to contain multiple IP prefix blocks, or for an IP prefix block to be spread across multiple ASes. Since AS numbers provide a slightly different granularity than IP prefixes, we view these sets of features as complementary and include both in our evaluations.

Now that we have described how URLs are resolved and noted the valuable information that can be mined from this process, we next describe the list of features we use in our URL classification system.

2.3. Features

As in our previous study [Ma et al. 2009a], we analyze lexical and host-based features because they contain information about the URL and host that is straightforward to collect using automated crawling tools. Thus, the list of features is extensive, but not necessarily exhaustive.

Table I lists the lexical and host-based feature types we consider and the number contributed by each type, and Figure 2 gives a high-level illustration of how an individual feature vector is constructed. Overall, lexical types account for 62% of features and host-based types account for 38%. We next describe the feature types and the motivation behind including them for classification.

Lexical Features. The justification for using lexical features is that URLs to malicious sites tend to "look different" in the eyes of the users who see them. Hence, including lexical features allows us to methodically capture this property for classification purposes, and perhaps infer patterns in malicious URLs that we would otherwise miss through ad hoc inspection.

Table I. Feature Breakdown on Day 100 of the Experiments

Lexical		Host-Based	
Feature type	Count	Feature type	Count
Hostname	835,764	WHOIS info	917,776
Primary domain	738,201	IP prefix	131,930
Path tokens	124,401	AS number	39,843
Last path token	92,367	Geographic	28,263
TLD	522	Conn. speed	52
Lexical misc.	6	Host misc.	37
Lexical	1,791,261	Host-Based	1,117,901

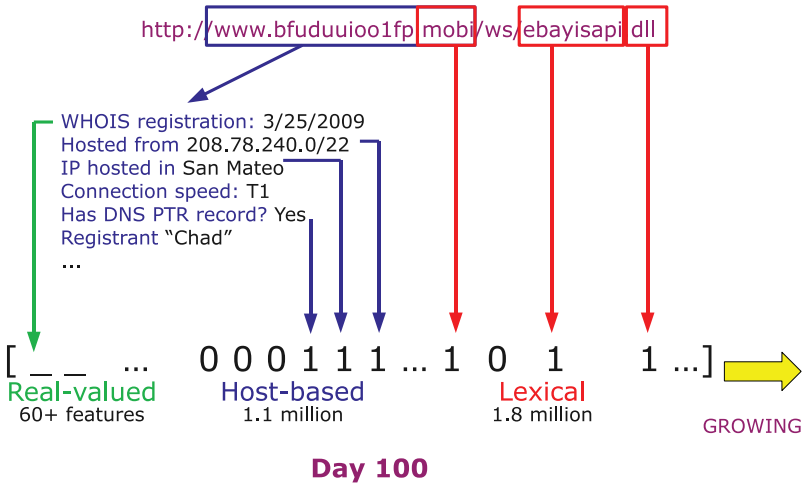


Fig. 2. Example of how to convert lexical and host-based features of a URL into vector format. Number of features shown for Day 100.

For the purpose of this discussion, we want to distinguish the two parts of a URL: the hostname and the path. As an example, with the URL `www.geocities.com/usr/index.html`, the hostname portion is `www.geocities.com` and the path portion is `usr/index.html`.

Lexical features are the textual properties of the URL itself (not the content of the page it references). We use a combination of features suggested by the studies of McGrath and Gupta [2008] and Kolari et al. [2006]. These properties include the length of the hostname, the length of the entire URL, as well as the number of dots in the URL; all of these are real-valued features. Additionally, we create a binary feature for each token in the hostname (delimited by ".") and in the path URL (strings delimited by "/", "?", ".", "=", "-" and "_"). This is also known as a "bag-of-words." Although we do not preserve the order of the tokens, we do make a distinction between tokens belonging to the hostname, the path, the Top-Level Domain (TLD), and primary domain name (the domain name given to a registrar). More sophisticated techniques for modeling lexical features are available, such as Markov models of text. However, even with the bag-of-words representation, we can achieve very accurate classification results.

Host-Based Features. Host-based features describe "where" malicious sites are hosted, "who" they are managed by, and "how" they are administered. We use these features because malicious Web sites may be hosted in less reputable hosting centers, on machines that are not conventional Web hosts, or through disreputable registrars.

The following are properties of the hosts (there could be multiple) that are identified by the hostname part of the URL. We note some of these features overlap with lexical properties of the URL.

(1) *IP address properties.* Are the IPs of the A, MX, or NS records in the same Autonomous Systems (ASes) or prefixes of one another? To what ASes or prefixes do they belong? If the hosting infrastructure surrounding malicious URLs tends to reside in a specific IP prefix or AS belonging to an Internet Service Provider (ISP), then we want to account for that disreputable ISP during classification.
(2) *WHOIS properties.* What is the date of registration, update, and expiration? Who is the registrar? Who is the registrant? Is the WHOIS entry locked? If a set of malicious domains are registered by the same individual, we would like to treat such ownership as a malicious feature. Moreover, if malicious sites are taken down frequently, we would expect the registration dates to be newer than for legitimate sites.
(3) *Domain name properties.* What is the Time-To-Live (TTL) value for the DNS records associated with the hostname? Additionally, the following domain name properties are used in the SpamAssassin Botnet plugin for detecting links to malicious sites in emails: Does the hostname contain "client" or "server" keywords? Is the IP address in the hostname? Is there a PTR record for the host? Does the PTR record in-turn resolve one of the host's IP addresses (known as having a "full-circle" record)?
(4) *Blacklist membership.* Is the IP address in a blacklist? For the evaluations in Section 6, 55% of malicious URLs were present in blacklists. Thus, although this feature is useful, it is still not comprehensive.
(5) *Geographic properties.* In which continent/country/city does the IP address belong? As with IP address properties, hotbeds of malicious activity could be concentrated in specific geographic regions.
(6) *Connection speed.* What is the speed of the uplink connection (e.g., broadband, dial-up)? If some malicious sites tend to reside on compromised residential machines (connected via cable or DSL), then we want to record the host connection speed.

The list of features we use is hardly exhaustive: one can always create more features by generating or aggregating new meta-information about the URL (e.g., popularity rankings in Netcraft, indexing in Google). Nevertheless, the list is still extensive, and taken as a whole, it assembles many pieces of information about the URL and host. Much of this information is publicly available; thus we can collect it efficiently through the automated crawling tools at our disposal.

Figure 3 shows the cumulative number of features for each day of the evaluations. Each day's total includes new features introduced that day and all old features from previous days (see Section 6 on our methodology for new features). The individual URL feature vectors are sparse and have 116 nonzero features on average (with a standard deviation of 17 features). However, the dimensionality of the dataset as a whole grows quickly because we assign a binary feature for every token we encounter among the URL lexical tokens, as well as WHOIS and location properties. As we show in Section 6.3, by accounting for new features like the ones in Figure 3, we can significantly improve the detection of new malicious URLs.

3. ONLINE ALGORITHMS

This section briefly describes the online learning algorithms we use for our evaluations. Formally, the algorithms are trying to solve an online classification problem over a sequence of pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_T, y_T)\}$, where each $\mathbf{x}_t$ is an example's feature vector and $y_t \in \{-1, +1\}$ is its label. At each time step t during training, the algorithm

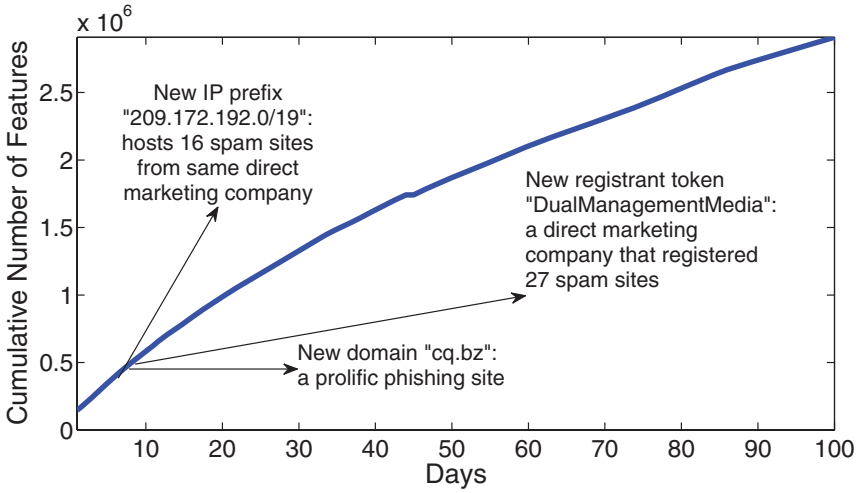


Fig. 3. Cumulative number of features observed over time for our live URL feeds. We highlight a few examples of new features at the time they were introduced by new malicious URLs.

makes a label prediction $h_t(\mathbf{x}_t)$. For linear classifiers, the prediction is given by $h_t(\mathbf{x}) = \text{sign}(\mathbf{w}_t \cdot \mathbf{x})$, where $\mathbf{w}$ is the classifier's weight vector.

After making a prediction, the algorithm receives the actual label y_t. (If $h_t(\mathbf{x}_t) \neq y_t$, we record an error for time t.) Then, the algorithm constructs the hypothesis for the next time step h_{t+1} using h_t, $\mathbf{x}_t$ and y_t.

As practitioners, we have no vested interest in any particular strategy for online learning. We simply want to determine the approach that scales well to problems of our size and yields the best performance. To that end, the online methods we evaluate are a mix of classical and recent algorithms. We present the models in order of increasing sophistication with respect to the objective functions and the treatment of classification margin (which we can also interpret as classification confidence).

Perceptron. This classical algorithm is a linear classifier that makes the following update to the weight vector whenever it makes a mistake [Rosenblatt 1958].

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t \tag{1}$$

The advantage of the Perceptron is its simple update rule. However, because the update rate is fixed, the Perceptron cannot account for the severity of the misclassification. As a result, the algorithm can overcompensate for mistakes in some cases and undercompensate for mistakes in others.

Logistic Regression with Stochastic Gradient Descent. Many batch algorithms use gradient descent to optimize an objective function that is expressed as a sum over individual examples. Stochastic Gradient Descent (SGD) provides an *online* method for approximating the gradient of the original objective, whereby the model parameters are updated incrementally by the gradients that arise from individual examples. In this article we evaluate SGD as applied to logistic regression.

Let $P(y_t = +1|\mathbf{x}_t) = \sigma(\mathbf{w} \cdot \mathbf{x}_t)$ be the likelihood that example t's label is $+1$, where the sigmoid function is $\sigma(z) = [1 + e^{-z}]^{-1}$. Moreover, let $L_t(\mathbf{w}) = \log \sigma(y_t(\mathbf{w} \cdot \mathbf{x}_t))$ be the log-likelihood for example t. Then the update for each example in logistic regression with SGD is

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \gamma \frac{\partial L_t}{\partial \mathbf{w}} = \mathbf{w}_t + \gamma \Delta_t \mathbf{x}_t, \tag{2}$$

where $\Delta_t = \frac{y_t+1}{2} - \sigma(\mathbf{w}_t \cdot \mathbf{x}_t)$ and γ is a constant training rate. We do not decrease γ over time so that the parameters can continually adapt to new URLs. The update resembles a Perceptron, except with a learning rate that is proportional to Δ_t, the difference between the actual and predicted likelihood that the label is $+1$. This multiplier allows the model to be updated (perhaps by a small factor) even when there is no prediction mistake.

SGD has received renewed attention because of recent results on the convergence of SGD algorithms and the casting of classic algorithms as SGD approximations [Bottou 1998; Bottou and LeCun 2004]. For example, the Perceptron can be viewed as an SGD minimization of the hinge-loss function $Loss(\mathbf{w}) = \sum_t \max\{0, -y_t(\mathbf{w} \cdot \mathbf{x}_t)\}$.

Passive-Aggressive (PA) Algorithm. The goal of the Passive-Aggressive algorithm is to change the model as little as possible to correct for any mistakes and low-confidence predictions it encounters [Crammer et al. 2006]. Specifically, with each example PA solves the following optimization.

$$\begin{aligned} \mathbf{w}_{t+1} \leftarrow \underset{\mathbf{w}}{\text{argmin}} \quad & \tfrac{1}{2}\|\mathbf{w}_t - \mathbf{w}\|^2 \\ \text{s.t.} \quad & y_t(\mathbf{w} \cdot \mathbf{x}_t) \geq 1 \end{aligned} \tag{3}$$

Updates occur when the inner product does not exceed a fixed confidence margin, that is, $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) < 1$. The closed-form update for all examples is as follows.

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_t y_t \mathbf{x}_t \tag{4}$$

where $\alpha_t = \max\{\frac{1-y_t(\mathbf{w}_t \cdot \mathbf{x}_t)}{\|\mathbf{x}_t\|^2}, 0\}$. (The details of the derivation are in Crammer et al. [2006].) The PA algorithm has been successful in practice because the updates explicitly incorporate the notion of classification confidence.

Confidence-Weighted (CW) Algorithm. The idea behind Confidence-Weighted classification is to maintain a different confidence measure for each feature so that less confident weights are updated more aggressively than more confident weights. The "Stdev" update rule for CW is similar in spirit to PA. However, instead of describing each feature with a single coefficient, CW describes per-feature confidence by modeling uncertainty in weight w_i with a Gaussian distribution $\mathcal{N}(\mu_i, \Sigma_{ii})$ [Dredze et al. 2008; Crammer et al. 2009]. Let us denote $\boldsymbol{\mu}$ as the vector of feature means, and $\boldsymbol{\Sigma}$ as the *diagonal* covariance matrix (i.e., the confidence) of the features. Then the decision rule becomes $h_t(\mathbf{x}) = \text{sign}(\boldsymbol{\mu}_t \cdot \mathbf{x})$, which is the result of computing the average signed margin $\mathbf{w}_t \cdot \mathbf{x}$, where $\mathbf{w}_t$ is drawn from $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, and then taking the sign.

The CW update rule adjusts the model as little as possible so that $\mathbf{x}_t$ can be correctly classified with probability η. Specifically, CW minimizes the KL divergence between Gaussians subject to a confidence constraint at time t.

$$\begin{aligned} (\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) \leftarrow \underset{\boldsymbol{\mu}, \boldsymbol{\Sigma}}{\text{argmin}} \; & \mathrm{D}_{KL}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)) \\ \text{s.t. } y_i(\boldsymbol{\mu} \cdot \mathbf{x}_t) \geq & \Phi^{-1}(\eta)\sqrt{\mathbf{x}_t^\top \boldsymbol{\Sigma} \mathbf{x}_t} \end{aligned} \tag{5}$$

where Φ is the cumulative distribution function of the standard normal distribution. This optimization yields the following closed-form update.

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &\leftarrow \boldsymbol{\mu}_t + \alpha_t y_t \boldsymbol{\Sigma}_t \mathbf{x}_t \\ \boldsymbol{\Sigma}_{t+1}^{-1} &\leftarrow \boldsymbol{\Sigma}_t^{-1} + \alpha_t \phi u_t^{-\frac{1}{2}} \, \text{diag}^2(\mathbf{x}_t) \end{aligned} \tag{6}$$

where α_t, u_t and ϕ are defined in Crammer et al. [2009]. However, we can see that if the variance of a feature is large, the update to its mean will be more aggressive. As for performance, the runtime of the update is linear in the number of nonzero features in $\mathbf{x}$.

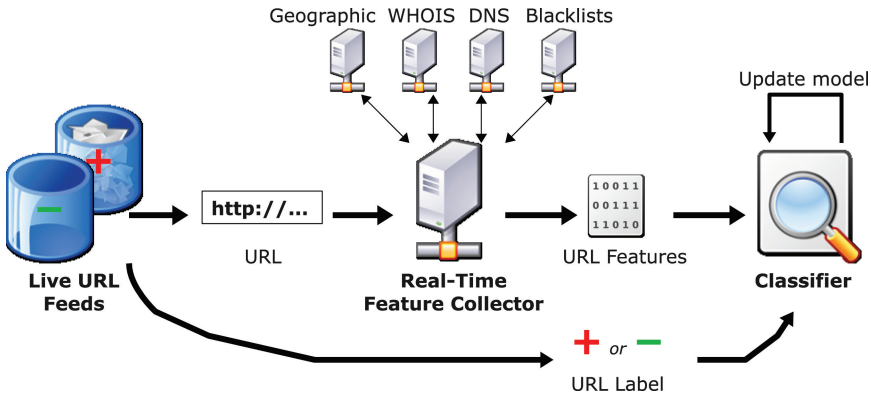


Fig. 4. Overview of real-time URL feed, feature collection, and classification infrastructure.

The CW algorithm is especially well-suited to detecting malicious URLs from a real-time feed of labeled examples. In particular, by estimating confidences on the weights of individual features, CW can more aggressively update the weights on features that have recently been introduced into the data feed. At the same time, it can moderate the updates for weights of recurring features that have already been estimated with high confidence. Of the algorithms we considered, only CW can manage the dynamic mix of new and recurring features in this way.

Related Algorithms. We also experimented with online kernel-based algorithms for nonlinear classification, such as the Forgetron [Dekel et al. 2008] and the Projectron [Orabona et al. 2008]. To make computation tractable, these algorithms budget (or at least try to reduce) the size of the support set used for kernel calculations. Despite the potentially greater power of nonlinear classifiers, our preliminary evaluations revealed no improvement over linear methods for URL classification.

4. DATA COLLECTION

This section describes our live sources of labeled URLs and the system we deploy to collect features in real time. Figure 4 illustrates our data collection architecture, which starts with two feeds of malicious and benign URLs.

We obtain examples of malicious URLs from a large Web mail provider, whose live, real-time feed supplies 6,000–7,500 examples of spam and phishing URLs per day. The malicious URLs are extracted from email messages that users report as spam; they are then verified manually after running a filter to eliminate (easily detected) false positives from the user reports.

We randomly draw our examples of benign URLs from Yahoo's directory listing. A random sample from this directory can be generated by visiting the link `http://random.yahoo.com/bin/ryl`.

Combining these two feeds, we collect a total of 20,000 URLs per day, with an average 2:1 ratio of benign-to-malicious URLs. By measuring the total error rate on URLs that appear in this ratio, we implicitly value the cost of a false positive at twice the cost of a false negative. As different applications may assign different costs to these errors, we explore the consequences of training with different imbalances in Section 6.4. We ran our experiments for 100 days, collecting nearly 2 million URLs. (There were feed outages during Days 35–40 and 45.) Note, however, that the feeds only provide the actual URLs, not the associated features that are required for classification.

In addition to monitoring the feeds of benign and malicious URLs, further infrastructure is required to gather the features in real time. The real-time gathering is important

because we want to obtain the features of URLs when they were first captured by the feed (which ideally match the features when they were first introduced into the wild). For each incoming URL, our feature collector immediately queries DNS, WHOIS, blacklist, and geographic information servers, as well as processing IP address-related and lexical-related features.[1]

Note that the real-time demands of our application are greater than those posed by other datasets for large-scale learning (e.g., the `webspam` dataset from the PASCAL Large-Scale Learning Challenge [Sonnenburg et al. 2008]). At the same rate as our application acquires URLs from the live feed, it must also fetch lexical and host-based features to construct the dataset on an ongoing basis. By contrast, the `webspam` dataset is a static representation of Web pages (using strings), not URLs; for benchmarks on this dataset, the passage of time does not play a limiting role in the gathering of features.

Our live feed provides a real-time snapshot of malicious URLs that reflects the evolving strategies of Internet criminals. As we shall see, the success of any real-world deployment is likely to depend on the "freshness" of this data and the ability to process it efficiently on the largest possible scale.

5. IMPLEMENTATION

In this section we discuss the implementation of our system for URL classification. At a high level, we implemented the feature collection using a series of custom Bash and Ruby scripts. For each URL, the individual scripts handling lexical, IP address, WHOIS, DNS, blacklist, and geographic features each output their results to an intermediate format. This intermediate format is then converted to a Matlab sparse matrix. After feature collection, we perform classification using online and batch algorithms, which we implement in Matlab. The following sections provide more details about our infrastructure for feature collection and our data types for feature representation.

5.1. Feature Collection Infrastructure

We construct the feature vector for each URL in real time. When our feature collection server receives a URL, it attempts to query several external servers to construct the host-based portion of the feature vector. The host-based feature collection has the following components.

—For IP address features, we look up the IP prefix and AS associated with a given IP address using a Routing Information Base (RIB), which can be downloaded from the Route Views Project [University of Oregon Advanced Network Technology Center 2010]. There is a small, fixed overhead of loading and indexing the database (30–40MB in size) in memory before feature collection begins. Once the database is loaded in memory, the lookup for IP prefixes and AS numbers is efficient. However, practitioners should keep the database up-to-date by periodically downloading the latest RIBs from Route Views.

—For WHOIS data, we constructed a command-line PHP script around `PHPWhois` [Jeftovic and Saez 2010]. The script handles the parsing of WHOIS entries, which are typically stored as flat text files with no standard format. WHOIS queries are high-latency operations which take 1–3 seconds for most domain names and on the order of several seconds for domains hosted from smaller registrars. We set a query timeout of 7 seconds in our implementation to avoid undue delays in feature collection.

[1]An anonymized version of this dataset is available at `http://archive.ics.uci.edu/ml/datasets/URL+Reputation`

—We collect DNS data (e.g., IP addresses for the domain name's associated A, NS, and MX records) by parsing the output of the `host` command. The latency for collecting these features is low (less than a second) because DNS is structured to handle a high query volume.
—We query six blacklists (and one white list) run by SORBS, URIBL, SURBL, and Spamhaus. These queries produce seven binary features, and the overhead is no more than performing a set of DNS queries.
—We collect geographic features using the NetAcuity service [Digital Element 2010]. The query latency is very low because we have a dedicated NetAcuity server for the campus.

Using only a single machine, our implementation takes 3.5 seconds on average to collect the features for each URL. We believe there is potential to reduce the per-URL feature collection latency by parallelizing high-cost feature queries (such as WHOIS data) across multiple machines.

5.2. Feature Representation

We implement the classification algorithms in Matlab, which requires us to store the URL features as sparse vectors in a high-dimensional Euclidean space. Multiple feature vectors (from successive URLs in the feed) are stored as the rows of a sparse matrix. Since new features are continually generated by previously unseen URLs, the number of columns in this matrix also grows over time.

In our implementation, we collect data in day-by-day chunks and construct the feature vectors for classification at the end of each day. The feature vector at day N contains elements for all the new features collected on day N plus all the elements of the feature vector on the previous day. For examples that precede the first occurrence of a newly added feature, we assign zero values to all features that have not yet appeared in the feed. New features also generate new elements of the weight vector for linear classification. We also assign zero values to initialize the elements of weight vectors corresponding to newly added features.

5.3. Suggestion for Production Deployment

In our evaluations, we deliberately decouple the acts of feature collection and classification in order to perform controlled experiments. In particular, we collect the data and features *in real time* while classifying the URLs at the end of each day, preserving the order of the feed. For a Web-scale deployment, however, this daily cycle would be too infrequent: we need a data representation that dynamically accounts for growing feature vectors. For a production system, we recommend to store each URL's features (as well as each classifier's weight vector) as a small hash table of `<feature, value>` pairs. The sparseness of individual feature vectors yields efficient hash table-based implementations of all the algorithms we consider for online learning.

6. EVALUATION

In this section, we evaluate the effectiveness of online learning over the live URL feed. To demonstrate this effectiveness, we address the following questions: Do online algorithms provide any benefit over batch algorithms? Which online algorithms are most appropriate for our application? Is there a particular training regimen that fully realizes the potential of these online classifiers? And how does the ratio of benign-to-malicious training examples affect classification outcomes?

By "training regimen", we refer to: (1) when the classifier is allowed to retrain itself after attempting to predict the label of an incoming URL, and (2) how many features the classifier uses during training.

For (1), we compare "continuous" versus "interval-based" training. Under the "continuous" training regimen, the classifier may retrain its model after each incoming example (the typical operating mode of online algorithms). In the "interval-based" training regimen, the classifier may only retrain after a specified time interval has passed. In our experiments, we set the interval to be one day. Batch algorithms are restricted to interval-based training, since continuous retraining would be computationally impractical. Unless otherwise specified, we use continuous retraining for all experiments with online algorithms (and then evaluate the benefit of doing so in Section 6.3).

For (2), we compare training using a "variable" versus "fixed" number of features. Under the fixed-feature regimen, we train using a predetermined set of features for all evaluation days. For example, if we fix the features to those encountered up to Day 1, then we use those 150,000 features for the whole experiment (see Figure 3). Under the variable-feature regimen, we allow the dimensionality of our models to grow with the number of new features encountered; on Day 8, for instance, we classify with up to 500,000 features. Implicitly, examples that were introduced before a feature i was first encountered will have value 0 for feature i. Unless otherwise specified, we use the variable-feature training regimen for all algorithms (and then evaluate the benefit of doing so in Section 6.3).

As for the sizes of the training sets, online algorithms implicitly train on a cumulative dataset, since they can incrementally update models from the previous day. For batch algorithms, we vary the training set size to include day-long and multi-day sets (details in Section 6.1).

Finally, we are interested in generating confidence intervals for our results. Although our data collection from Section 4 consists of 2 million examples, we create 10 subsampled datasets (i.e., folds) for our evaluations. When we generate each fold, every example starting from Day 1 has a 0.5 probability of being included in the sample. (The exception is that we do not subsample Day 0's 16,000 URLs, the initial data from which we initialize all models.) Thus, the effective size of each fold is about 1 million URLs, and in our results we show the average classification rates with error bars indicating the minimum and maximum classification rates among the 10 subsamples. (Note that while the error bars are present in all the figures, in some cases, due to limitations of scale, the spread among subsample classification rates is obscured by the much larger spread among average rates from different algorithms.)

6.1. Advantages of Online Learning

We start by comparing batch versus online methods for classification and examining whether the efficiencies of the latter come at the expense of worse accuracy. Specifically, we compare the online Confidence-Weighted (CW) algorithm against four different training set configurations of a Support Vector Machine (SVM). We use the `LIBLINEAR` 1.24 implementation of an SVM with a linear kernel as our canonical batch algorithm [Fan et al. 2008]. Evaluations with other batch algorithms such as logistic regression yielded similar results.

Figure 5 shows the classification rates (with error bars at one standard deviation) for CW and for SVM using four types of training sets. We tuned all classifier parameters over one day of holdout data, setting $C = 100$ for SVM, and $\eta = 0.90$ for CW. The x-axis shows the number of days in the experiment, and the y-axis shows the cumulative error rate: the percentage of misclassified examples for all URLs encountered up to that date.

The SVM-once curve represents training once on Day 0's data and using that model for testing on all other days. The cumulative error steadily worsens to 3.5%, and the cumulative false negative rate approaches 5.1%. These high error rates suggest that, to

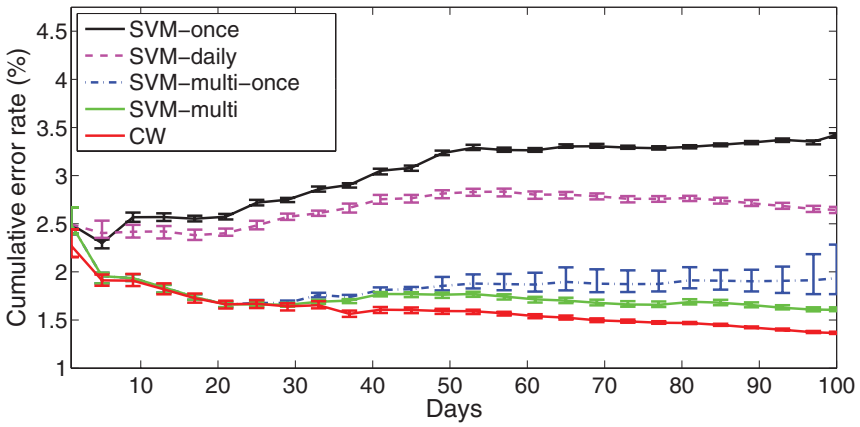


Fig. 5. Cumulative error rates for CW and for batch algorithms under different training sets. Note the y-axis starts at 1%.

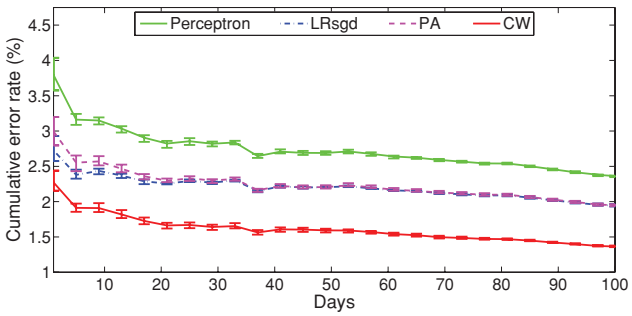
achieve better accuracy, the model must train on fresh data to account for new features of malicious *and* benign URLs encountered over time.

SVM-daily retrains only on data collected the previous day, for example, Day 6 results reflect training on the URLs collected on Day 5, and testing on Day 6 URLs. The only exception is that we do not retrain during the feed outages on Days 35–40 and 45. As a result, the cumulative error is 2.6%, most of which is due to the cumulative 4.3% false negative rate, whereas the cumulative false positive rate is 1.8%. Although fresh data eventually helps SVM-daily improve over SVM-once, one day's training data is still insufficient.

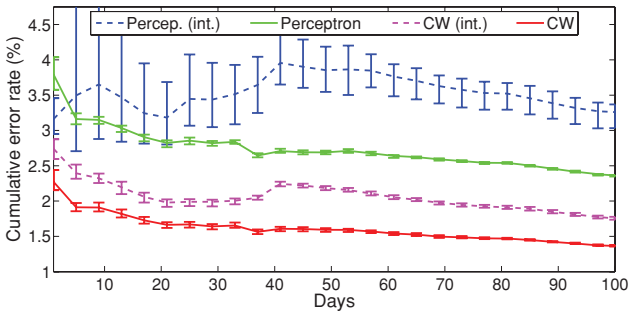
We use multi-day training sets to address this issue by training on as much data as our evaluation machine can handle: 21 days worth, or about 210,000 examples. (Although the machine had 4GB RAM, the size of the training set was limited by the `LIBLINEAR` implementation.) SVM-multi-once is the multi-day analog to SVM-once. Here, SVM-multi-once trains on data from Days 0 to 20, and from Day 21 on it uses that fixed model for testing on subsequent days. The improvement over SVM-once shows the benefit of more training data, but the steadily worsening error again demonstrates the dynamic nature of the URL dataset.

SVM-multi is the multi-day analog of SVM-daily. Here, SVM-multi trains on the previous 21 days worth of data. The resulting cumulative error reaches 1.6%. SVM-multi's improvement over SVM-multi-once suggests that because new URLs continuously introduce new features, we need to use as much fresh data as possible to succeed. Overall, these results suggest that more training data yields better accuracy. For our application, we conclude that single-machine, batch implementations of SVMs are limited by the necessarily bounded amount of training data that can be loaded in memory. Moreover, this limitation seems fundamental.

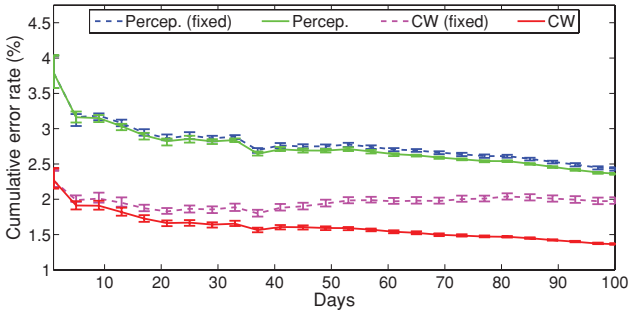
Online algorithms do not suffer this limitation, and they have the added benefit that they can incrementally adapt to new data and features. As we see in Figure 5, the accuracy for CW exceeds SVM-multi. Since the online algorithm makes a *single pass* over a cumulative training set, it does not incur the overhead of loading the entire dataset in memory. Also, since the CW classifier is trained incrementally, it is capable of adapting to new examples in real time, whereas batch algorithms are restricted to retraining at the next available interval. (For more on interval-based training, see Section 6.3.) Generally speaking, in our experiments, these advantages enable online algorithms to outperform batch ones.



(a) error rates for online algorithms; all use continuous/variable-feature training



(b) benefits of using continuous training over interval-based training



(c) benefits of using variable-feature sets over fixed-feature sets

Fig. 6. Comparing the effectiveness of various online algorithms, their use of continuous versus interval training, and their use of fixed versus variable feature sets.

6.2. Comparison of Online Algorithms

Given the demonstrated benefits of online learning over batch learning, we next evaluate which of the online algorithms from Section 3 are best suited to malicious URL detection. The main issue that these experiments address is whether recent developments in online algorithms, which include optimizing different objective functions, adjusting for classification confidence, and treating features differently, can benefit the classifiers in our application.

Figure 6(a) shows the cumulative error rates for the online algorithms. All algorithms in this experiment adopt the continuous training regimen. We also note that

the error rates improve steadily over time for all classifiers, reaffirming that training on cumulative data is beneficial.

The Perceptron is the simplest of the algorithms, but it also has the highest error rates across all of the days at around 2.4–3%. This result suggests that because the Perceptron treats mistakes equally (and ignores all correct classifications), its updates are too coarse to accurately keep up with new examples. There needs to be a more fine-grain distinction between misclassified and correctly classified examples with respect to their impact on model updates.

Both Logistic Regression with stochastic gradient descent (LRsgd) and the Passive-Aggressive (PA) algorithm achieve a cumulative error approaching 1.9%, improving over the Perceptron results. (Here we tuned the LRsgd learning rate to $\gamma = 0.01$ over one day of holdout data.) Presumably, this improvement occurs because LRsgd and PA account for classification confidence. Specifically, LRsgd updates are proportional to Δ_t, and PA updates are proportional to the normalized classification margin α_t. These results are slightly worse than SVM-multi.

The CW results suggest that the final leap comes from *treating features differently*, both in terms of how they affect classification confidence, and how quickly they should be updated. With an error approaching 1.4%, CW clearly outperforms the other algorithms. Most of the gap between CW and the other online methods comes from CW's lower false negatives; CW has a cumulative false negative rate of 1.8%, whereas the false negative rate for others is 3.0–3.6%. We believe the gap occurs because CW updates select portions of its model very aggressively to account for new malicious features, all without perturbing more established features.

Overall, we find that the more recent online algorithms outperform the simpler ones. Because the live combined URL feed contains a dynamic mix of new and recurring features, CW's per-feature confidence weighting can exploit that structure to achieve the best accuracy.

6.3. Training Regimen

In this section, we show that there is a significant advantage to continuous training versus interval-based training. We also demonstrate that there is significant benefit to adding newly encountered features as opposed to using a fixed feature set. The aforementioned training regimens can help online algorithms stay abreast of newly introduced URL features. Thus, choosing the right training regimen can be just as important as choosing the right algorithm.

Figure 6(b) shows the value of using continuous training over interval training with the CW and Perceptron algorithms. The higher error rates for interval training show that there is enough variation between days that a model can become stale if it is not retrained soon enough. In particular, the higher number of false negatives for interval-trained CW is responsible for the persistent gap with continuously trained CW. Notwithstanding the aforementioned feed outages on Days 35–40 and 45, the 1% error difference between continuous and interval-based Perceptron is due to spikes in the false positive/negative rates for the interval-trained Perceptron. Thus, continuous retraining yields as much improvement for the simpler Perceptron as it does for CW.

In addition to continuous retraining, accounting for new features is critical to an algorithm's success. Figure 6(c) shows the value of using variable-feature training over fixed-feature training. In this graph, "fixed features" means that we restrict the model to using the features encountered up to Day 1 only (150,000 features total). We see that the performance for fixed-feature CW degrades over time. Interestingly, variable-feature Perceptron only achieves a marginal improvement over fixed-feature Perceptron. One explanation is that, even though variable-feature Perceptron can occasionally benefit from adding new features, it does not update the new feature weights

Table II. Cumulative Error, False Positive (FP) and False Negative (FN) Rates with Standard Deviations for Online Evaluations of CW for Different Ratios ($N : P$) of Benign-to-Malicious URLs in Training

	Sampled training data only			All data		
Ratio	Error (%)	FP (%)	FN (%)	Error (%)	FP (%)	FN (%)
100:1	0.31 ± 0.01	0.07 ± 0.01	25.49 ± 0.63	8.45 ± 0.14	0.07 ± 0.01	25.70 ± 0.43
10:1	1.24 ± 0.01	$0.60 \pm <0.01$	7.90 ± 0.14	2.99 ± 0.06	0.59 ± 0.01	7.95 ± 0.12
2:1	1.86 ± 0.02	1.53 ± 0.02	2.54 ± 0.04	1.85 ± 0.01	1.52 ± 0.01	2.55 ± 0.03
1:1	1.83 ± 0.03	2.12 ± 0.03	1.53 ± 0.03	1.91 ± 0.02	2.09 ± 0.03	1.54 ± 0.02
1:10	0.84 ± 0.01	5.25 ± 0.09	0.38 ± 0.01	3.69 ± 0.05	5.29 ± 0.08	0.39 ± 0.01

See text for details.

aggressively enough to correct for future errors. By contrast, the CW algorithm updates new features aggressively by design, and hence can reap the full benefits of variable-feature training.

Overall, continuous retraining with a variable feature set allows a model to successfully adapt to new data and new features on a subday granularity. And this adaptiveness is critical to realizing the full benefits of online algorithms.

6.4. Effect of Mismatched Testing Conditions

In the previous experiments, the classifier was trained and tested on twice as many benign URLs as malicious URLs. In this section, we explore the consequences of mismatched testing conditions; that is, when the ratios of benign-to-malicious URLs differ significantly in training and testing. Such conditions can arise when the classifier is deployed in a different manner than it was trained. For example, suppose that a traditional spam filter is used to eliminate URLs from suspicious emails with product advertisements. With such URLs already flagged, the goal of the classifier would shift to detecting phishing sites, which (because they are clearly illegal in most jurisdictions) do not exist in nearly the same abundance as sites that merely sell spam-advertised products. Antiphishing experts estimate the ratio of nonphishing to phishing sites as anywhere from 100:1 to 1000:1; by contrast, 90% of all email messages are estimated to contain some form of spam [MAAWG 2010]. Thus, for classifiers deployed in this way, the ratio of benign-to-malicious URLs might be several orders of magnitude lower in testing than training.

To explore the consequences of mismatched testing conditions, we evaluate how our approach performs with different ratios of benign-to-malicious URLs in training and testing. Specifically, we experiment by varying the ratio of negative to positive examples used to train CW classifiers while fixing the ratio of negative to positive examples used to test them. We use random sampling to simulate a particular ratio $N : P$ of negative to positive examples in training. In particular, we include each negative example that we encounter with probability p_- and each positive example with probability p_+, where the probabilities $p_\pm$ are set to obtain an expected ratio $N : P$ of benign to malicious URLs and to include 30% of each fold's data. (As before, we average our results over 10 different folds of the data.) The 30% subsampling is needed to ensure that evaluations for different ratios $N : P$ receive the same number of expected training examples; here we are limited by the number of malicious URLs in each fold.

Table II displays the cumulative results for training CW over 100 days with different ratios of benign-to-malicious URLs in training. The results include error rates over the sampled training data as well as the over the entire dataset (which includes examples that were not included for training). As expected, CW performs best in matched testing conditions when the same ratio (2:1) of negative to positive examples is used in both training and testing. (The 1.85% error rate in this table is higher than the 1.4% error rate in previous sections because only 30% of data was used for training.)

We gain more insight by examining False Positive (FP) and False Negative (FN) rates in addition to overall error rates. In particular, we observe three distinct trends. First, FN/FP rates over training examples match well with the FN/FP rates seen over all the data. This suggests that even though the overall error rate in training may not match testing conditions, the FN and FP rates in training will be similar to those in testing. Second, the ratio (FN/FP) of false negative and false positive rates roughly tracks the ratio of benign-to-malicious URLs used in training. Third, when we increase the ratio of benign-to-malicious URLs in training by two orders of magnitude (e.g., from 1:1 to 100:1, or from 1:10 to 10:1), the false positive rate drops roughly by one order of magnitude; at the same time, however, the false negative rate jumps by nearly the same amount. We believe that these general trends would be observed for other regimes of mismatched testing conditions.

The preceding results suggest that practitioners should exploit any prior knowledge of the ratio of benign-to-malicious URLs that occur in the wild. In particular, to obtain the best overall error rates, they should tune the training set as much as possible to match the expected testing conditions. The problem of mismatched testing conditions occurs in many applications of machine learning; see Zadrozny et al. [2003] for further discussion of sampling-based training strategies in this situation.

7. RELATED WORK

Many researchers have examined the statistics of suspicious URLs in some way. Our approach in this article differs from previous work in the following respects: we employ a more comprehensive set of features, we train on larger datasets, and we focus on machine learning in the online (as opposed to batch) setting. At the same time, our approach also borrows important insights from previous studies. Next we review the previous work in feature engineering and machine learning that motivated our own approach.

Kan and Thi [2005] provide one of the early studies of machine learning for URL classification. To train their models quickly, they only analyze the lexical features of URL strings; like our approach, they do not analyze page content, but unlike our approach, they do not extract features about each URL's host. To generate feature vectors, they use a bag-of-words representation of tokens in the URL; their representation also takes into account where the tokens appear within the URL (e.g., hostname, path, etc.). They derive a large number of features from consecutive n-grams of tokens, ordered bigrams of nonconsecutive tokens, and the lengths of different parts of the URL. Comparing classifiers of lexical features and page content features, they obtain the noteworthy result that the former can achieve 95% of the accuracy of the latter. In this article, we use a similar but simpler set of lexical features; in our case, a reduced set of lexical features sufficed to achieve similarly high levels of accuracy.

The work by Garera et al. is the most closely related to ours [Garera et al. 2007]. They use logistic regression over 18 hand-selected features to classify phishing URLs. The features include the presence of red flag keywords in the URL, features based on Google's Page Rank, and Google's Web page quality guidelines. They achieve a classification accuracy of 97.3% over a set of 2,500 URLs. It is difficult to make a direct comparison with our approach without access to the same URLs and features. While sharing the same motivation and methodology, our approach differs significantly from theirs in both scope (detecting other types of malicious activity) and scale (orders-of-magnitude more features and training examples).

McGrath and Gupta do not construct a classifier but nevertheless perform a comparative analysis of phishing and nonphishing URLs [McGrath and Gupta 2008]. With respect to datasets, they compare nonphishing URLs drawn from the DMOZ Open Directory Project [Netscape] to phishing URLs from PhishTank [OpenDNS] and a

nonpublic source. The features they analyze include IP addresses, WHOIS thin records (containing date and registrar-provided information only), geographic information, and lexical features of the URL (length, character distribution, and presence of predefined brand names). We build on their work by incorporating similar sources and features into our approach.

Provos et al. perform a study of drive-by exploit URLs and use a patented machine learning algorithm as a prefilter for Virtual Machine (VM)-based analysis [Provos et al. 2008]. Unlike our approach, they extract content-based features from the page, indicating (for example) whether inline frames are "out of place" (an "IFrame" is a window within a page that can contain another page), whether there is obfuscated JavaScript, and whether IFrames point to known exploit sites. In their evaluations, the ML-based prefilter can achieve 0.1% false positives and 10% false negatives. We cannot directly compare our results to theirs as they are based on different datasets. We note only that our evaluations in Section 6.4 yielded similar performance without considering content-based features.

CANTINA classifies phishing URLs by thresholding a weighted sum of 8 features (4 content-related, 3 lexical, and 1 WHOIS-related) [Zhang et al. 2007]. Among the lexical features, it looks at dots in the URL, whether certain characters are present, and whether the URL contains an IP address. The WHOIS-related feature examined by CANTINA is simply the age of the domain. These features are a subset of those we use in our approach. In fitting linear classifiers, we also make use of more recent developments in the field of machine learning. Again, while we cannot compare directly to their results, our evaluations reveal the significant improvements in performance obtained from more sophisticated approaches to learning linear classifiers (e.g., the improvement of CW over Perceptron learning in Section 6.2).

Guan et al. [2009] focus on classifying URLs that appear in Instant Messaging (IM). Although they use several URL-based features, they also take advantage of a number of IM-specific features such as message timing and content. Among the URL-based features are the age of the domain (WHOIS lookup), Google rank, and a number of lexical features (IP address in the hostname, presence of particular tokens in the URL, as well as the length of certain hostname tokens). They use an ad hoc linear classifier where the weight of each feature is proportional to the difference in the number of benign examples that possess the feature and the number of malicious examples that do not. As discussed earlier, we believe that our approach has benefited significantly from better learning algorithms for linear classifiers.

7.1. Machine Learning and URL Features in Related Contexts

The work in this article deals with classifying URLs in a generic setting, irrespective of the applications in which they appear. However, there is a body of related work that uses URL-based features for classifying email messages and Web pages (not the URLs themselves).

Fette et al. use statistical methods in machine learning to classify phishing messages [Fette et al. 2007]. Their classifiers examine the properties of URLs contained within a message (e.g., the number of URLs, number of domains, and number of dots in a URL), but unlike our approach they also consider features of the email structure and content. Bergholz et al. further improve the accuracy of Fette et al. by introducing models of text classification to analyze email content [Bergholz et al. 2008]. Abu-Nimeh et al. compare different classifiers over a corpus of phishing messages, using as features the frequency of the 43 most popular words in the corpus [Abu-Nimeh et al. 2007].

Kolari et al. use URLs found within a blog page as features to determine whether the page is spam [Kolari et al. 2006]. They use a "bag-of-words" representation of URL

tokens, and we use a similar set of features in our approach which contribute to a highly accurate classifier.

7.2. Nonmachine Learning Approaches

Several projects have explored operating systems-level techniques where the client visits the Web site using an instrumented Virtual Machine (VM). The VM can emulate any client-side exploits that occur as a result of visiting a malicious site, and the instrumentation can detect whether an infection has occurred. In this way, the VM serves as a protective buffer for the user. Moshchuk et al. use VMs to analyze downloaded trojan executables, relying on third-party antispyware tools to detect whether VMs were infected by executables [Moshchuk et al. 2006]. Wang et al. detect drive-by exploits by using behavioral detection (monitoring anomalous state changes in the VM) as well as detecting exploits of known vulnerabilities [Wang et al. 2006]. Provos et al. monitor VM state changes and use multiple antivirus engines to detect VM infection [Provos et al. 2008]. Moshchuk et al. also construct a VM-based Web proxy defense that uses behavior-based detection and adds only a few seconds of overhead to page rendering for the end-client [Moshchuk et al. 2007]. These efforts have distinct benefits (direct detection of certain classes of sites such as drive-by exploits) and limitations (inadvertently exposing the user to undetected drive-by exploits and other malicious sites that do not fit their precise detection criteria); thus, they are complementary to our approach.

8. CONCLUSION

Despite existing defenses, malicious Web sites remain a scourge of the Internet. To protect end users from visiting these sites, we can attempt to identify suspicious URLs by analyzing their lexical and host-based features. A particular challenge in this domain is that URL classifiers must operate in a dynamic landscape; one in which criminals are constantly evolving new strategies to counter our defenses. To prevail in this contest, we need algorithms that continually adapt to new examples and features. In this article, we experimented with different approaches for detecting malicious URLs with an eye toward ultimately deploying a real-time system.

Experiments on a live feed of labeled examples revealed the limitations of batch algorithms in this domain. Most fundamentally, their accuracy appears to be limited by the number of training examples that can fit into memory. After observing this limitation in practice, we investigated the problem of URL classification in an online setting. We found that the best-performing online algorithms (such as CW) yield highly accurate classifiers, with errors rates around 1% on a balanced dataset. Our results suggested that these classifiers owe their strong performance to continuous retraining in the face of new features. Going forward, it is our hope that this work provides valuable lessons for other applications of machine learning to computer security.

ACKNOWLEDGMENTS

We thank Alvin AuYoung, Boris Babenko, Koby Crammer, Mark Dredze, Kirill Levchenko, Fernando Pereira, Patrick Verkaik, and Michael Vrable for many insightful comments on early versions of this work. We also thank the reviewers for their valuable feedback.

REFERENCES

Abu-Nimeh, S., Nappa, D., Wang, X., and Nair, S. 2007. A comparison of machine learning techniques for phishing detection. In *Proceedings of the Anti-Phishing Working Group eCrime Researchers Summit*.

Bergholz, A., Chang, J.-H., Paass, G., Reichartz, F., and Strobel, S. 2008. Improved Phishing Detection using Model-Based Features. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*.

Bottou, L. 1998. Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK.

Bottou, L. and LeCun, Y. 2004. Large scale online learning. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA.

Chou, N., Ledesma, R., Teraguchi, Y., Boneh, D., and Mitchell, J. C. 2004. Client-side defense against web-based identity theft. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

Crammer, K., Dekel, O., Shalev-Shwartz, S., and Singer, Y. 2006. Online passive-aggressive algorithms. *J. Mach. Learn. Res. 7*, 551–585.

Crammer, K., Dredze, M., and Pereira, F. 2009. Exact convex confidence-weighted learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Daigle, L. 2004. WHOIS protocol specification. RFC 3912.

Deering, S. and Hinden, R. 1998. Internet protocol, version 6 (IPv6) specification. RFC 2460.

Dekel, O., Shalev-Shwartz, S., and Singer, Y. 2008. The forgetron: A kernel-based perceptron on a budget. *SIAM J. Comput. 37,* 5, 1342–1372.

Digital Element. 2010. NetAcuity. `http://www.digital-element.com/ip_intelligence/ip_intelligence.html`.

Dredze, M., Crammer, K., and Pereira, F. 2008. Confidence-weighted linear classification. In *Proceedings of the International Conference on Marchine Learning (ICML)*.

Droms, R. 1997. Dynamic host configuration protocol. RFC 2131.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. 2008. LIBLINEAR: A library for large linear classification. `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`.

Fette, I., Sadeh, N., and Tomasic, A. 2007. Learning to detect phishing emails. In *Proceedings of the International World Wide Web Conference (WWW)*.

Fuller, V. and Li, T. 2006. Classless inter-domain routing (CIDR): The internet address assignment and aggregation plan. RFC 4632.

Garera, S., Provos, N., Chew, M., and Rubin, A. D. 2007. A Framework for Detection and measurement of phishing attacks. In *Proceedings of the ACM Workshop on Rapid Malcode (WORM)*. Alexandria, VA.

Guan, D. J., Chen, C.-M., and Lin, J.-B. 2009. Anomaly based malicious url detection in instant messaging. In *Proceedings of the Joint Workshop on Information Security (JWIS)*.

Jeftovic, M. and Saez, D. 2010. PHPWhois. `http://sourceforge.net/projects/phpwhois/`.

Kan, M.-Y. and Thi, H. O. N. 2005. Fast webpage classification using url features. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*.

Keizer, G. 2008. Spam plummets after calif. hosting service shuttered. `http://www.computerworld.com/s/article/9119963/Spam_plummets_after_Cali%f._hosting_service_shuttered`.

Kolari, P., Finin, T., and Joshi, A. 2006. SVMs for the blogosphere: Blog identification and splog detection. In *Proceedings of the AAAI Spring Symposium on Computational Approaches to Analysing Weblogs*.

Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. 2009a. Beyond blacklists: Learning to detect malicious web sites from suspicious URLs. In *Proceedings of the SIGKDD Conference*.

Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. 2009b. Identifying suspicious URLs: An application of large-scale online learning. In *Proceedings of the International Conference on Machine Learning (ICML)*. 681–688.

MAAWG. 2010. MAAWG email metrics program: The network operators' perspective. Rep. #12 – Third and Fourth Quarter 2009.

McGrath, D. K. and Gupta, M. 2008. Behind phishing: An examination of phisher modi operandi. In *Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*.

McMillan, R. 2010. Zeus botnet dealt a blow as ISP troyak knocked out. `http://www.itworld.com/government/100020/zeus-botnet-dealt-blow-isp-tro%yak-knocked-out`.

Mockapretis, P. 1987a. Domain names – Concepts and facilities. RFC 1034.

Mockapretis, P. 1987b. Domain names – Implementation and specification. RFC 1035.

Moshchuk, A., Bragin, T., Deville, D., Gribble, S. D., and Levy, H. M. 2007. SpyProxy: Execution-based detection of malicious web content. In *Proceedings of the USENIX Security Symposium*.

Moshchuk, A., Bragin, T., Gribble, S. D., and Levy, H. M. 2006. A Crawler-based study of spyware on the web. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.

Netscape. 2011. DMOZ Open Directory Project. `http://www.dmoz.org`.

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

OpenDNS. 2011. PhishTank. `http://www.phishtank.com`.

ORABONA, F., KESHET, J., AND CAPUTO, B. 2008. The projectron: A bounded kernel-based perceptron. In *Proceedings of the International Conference on Machine Learning (ICML)*.

PROVOS, N., MAVROMMATIS, P., RAJAB, M. A., AND MONROSE, F. 2008. All your iFRAMEs point to Us. In *Proceedings of the USENIX Security Symposium*.

ROSENBLATT, F. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev. 65,* 6, 386–408.

SINHA, S., BAILEY, M., AND JAHANIAN, F. 2008. Shades of grey: On the effectiveness of reputation based blacklists. In *Proceedings of the International Conference on Malicious and Unwared Software (Malware)*.

SONNENBURG, S., FRANC, V., YOM-TOV, E., AND SEBAG, M. 2008. PASCAL large scale learning challenge. `http://largescale.first.fraunhofer.de/workshop/`.

UNIVERSITY OF OREGON ADVANCED NETWORK TECHNOLOGY CENTER. 2010. Route views project. `http://www.routeviews.org`.

USC INFORMATION SCIENCES INSTITUTE. 1981. Internet protocol: DARPA internte program protocol specification. RFC 791.

WANG, Y.-M., BECK, D., JIANG, X., ROUSSEV, R., VERBOWSKI, C., CHEN, S., AND KING, S. 2006. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.

ZADROZNY, B., LANGFORD, J., AND ABE, N. 2003. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*.

ZHANG, Y., HONG, J., AND CRANOR, L. 2007. CANTINA: A content-based approach to detecting phishing web sites. In *Proceedings of the International World Wide Web Conference (WWW)*.

Received March 2010; revised August 2010; accepted November 2010