Malicious Website Detection Based on URLs Static Features

Chun-ming WU¹⁺, Min Li¹, Li YE¹, Xian-chun ZOU¹ and Bao-hua QIANG²

¹School of Computer and Information Science, Southwest University, Chongqing 400715, China
²Guangxi Cooperative Innovation Center of Cloud Computing and Big Data, Guilin University of
Electronic Technology, Guilin 541004, China

*Corresponding author

Keywords: Malicious URL, Static features, Feature extraction, Machine learning.

Abstract. Real-time and effectiveness are two basic requirements in URL-based malicious website
detection. Based on the analysis of malicious URLs construction pattern, this paper puts forward a
method to detect malicious website based on URLs static features. All features used in classification
are extracted using statistic and there is no need to online access to obtain additional information
about website, which reduces detection time greatly. At the same time, a feature reduction method is
given in feature vector construction. Combined with reasonable selection of machine learning
algorithm, our method can achieve both high accuracy and high efficiency. A series of comparative
experiments proved the effectiveness of our method.

Introduction

In last decade, the issue of malicious website detection has attracted considerable concern of
scholars. Technically, the existing works can be classified to three categories: 1) Blacklist-based, it
has the advantage of efficiency and high accuracy, but it cannot detect unknown malicious website. 2)
Content-based, it detects a webpage by examining its code or behavior [1]. Due to the need of online
content analysis, it is difficult to give a real-time response. 3) URL-based, it recognize URLs patterns
using machine learning to detect unknown URLs. This method has many advantages [10], e.g.
avoiding downloading page content which may damage user's computer, it is a lightweight operation
with high efficiency, it is applicable to any context in which URLs are found, etc. Due to these
advantages, URL-based method has been extensively studied [2-11].

Real-time and effectiveness are two basic requirements in URL-based detection. The existing
works usually put emphasis on improving accuracy while efficiency is neglected, which shown in the
following aspects. 1) In feature extraction, many works use external features obtained online (e.g. IP).
In practice, obtaining these features may cost a lot of time, cannot meet real-time requirement. 2) In
vector construction, there is no discussion about efficient computation of high dimensional features.
3) In classifier selection, some algorithms (e.g. SVM) can achieve high accuracy in small dataset, but
they have very low efficiency in large one, which make these algorithms unpractical.

This paper will discuss the above three aspects respectively and puts forward a detection method
based on URLs static features which can get both high accuracy and high efficiency.

Malicious URL Detection Using Machine Learning

Malicious URL detection can be defined as a binary classification task. Given a dataset with T URLs
\{(u₁, c₁),…,(uₜ, cₜ)\}, where uₜ (t=1,…,T) is a URL and cₜ \{1,0\}={malicious; 0=benign} is class label, the
detection process can be expressed as a function f(u)=c and the core problem is how to define the
mapping function f. The supervised machine learning is widely used in existed works.

There is a basic assumption that malicious and benign URLs have different feature distributions.
Given a URL uₜ, feature representation is to find a mapping g: uₜ \rightarrow xₜ, where xₜ \in \mathbb{R}^{d} is a d-dimensional
feature vector. A basic feature extraction method is to split URL by ‘/’, ‘.’ or other delimiters, forming
a series of tokens named as Token Features. The tokenize granularity can be very different. In [11],
URL will be split into only three parts: host, path and file, and each part is named as ‘segment’. In [8], to enhance the quality of token features, a combination of two adjacent words is extracted named as Bi-gram Features. [4] notices the concatenated words used in URL (e.g. ‘activatealert’) and split these words into its components. In addition, many works also take account of many external features, such as IP address and server location [10], WHOIS and DNS [7, 10], Google page rank and quality guidelines [2], directory and file name [5], etc. Experiments show that these external features can improve the classification accuracy to a certain extent.

When get feature vectors \( x \in R^d \), the next step is to train prediction model \( f : R^d \rightarrow R \). Many mature classification algorithms can be used directly, such as Maximum Entropy [4], Logistic Regression [2,10], Support Vector Machine (SVM) [10], Naive Bayesian [3], etc. In addition, some online learning algorithms, including PA, CW, AROW, have been proposed [5]. In 2016, [1] uses deep learning approach to detect malicious JavaScript code, which provides a new idea for the research.

Construction Patterns of Malicious URLs and Feature Analysis

URL features can be divided into two classes: dynamic and static. Dynamic features refer to the attributes obtained through online inquiry, while static features are those acquired directly through statistical analysis of URL strings. Studies have shown that using lexical features can achieve high accuracy (only 1% decrease compared to using full features) [5]. This suggests that only using static features is sufficient and it can provide a reasonable trade-off between accuracy and efficiency.

For timeliness, deception and other factors, malicious URLs tend to show certain patterns different from benign URLs. Based on the analysis of URL samples, we summarize the patterns as follows:

I. Lexical obfuscating. Malicious URLs always strive to ‘look’ normal, so some words like ‘login’, ‘account’, and ‘google’ often appear in them, which let users believe that these URLs are related with user login, account management or well-known companies, thus, trick user to click the link.

II. Domain name obfuscating. Malicious URLs often use this means to confuse users. II-1. Use IP address instead of domain name, which scarcely exists in normal web service. II-2. Use ‘@’ to hide the real domain, e.g. in ‘http://ibm.com@hacker.com/’, the actual effective domain is ‘hacker.com’. II-3. Use a lots of ‘.’ in domain. II-4. Use abnormal number of ‘-’ in domain. II-5. Use automatic generation technique to generate random and long string as domain name. II-6. Social engineering fraud. Add digits or substitute some letters with digits to imitate well-known domain name.

III. Abnormal path length. Many malicious URLs have irrational depth, i.e. a lot of ‘/’ in URL.

IV. Some keywords like ‘.exe’, ‘.dat’ often appear in file extension.

Based on pattern analysis, we can extract URL features pertinently. In this paper, we classify the static features into three categories: 1) Lexical feature, it includes the words and phrases most commonly used in URL that represents the habits of word usage, corresponding to pattern I and IV. For example, ‘login’ often appears in malicious URLs. 2) Grammatical feature, it includes the delimiters and numbers used in URL that represents the structural statistical characteristics of malicious URL, corresponding to pattern II and III. For example, the length of malicious URL is usually longer. 3) Token feature, it includes a set of token combinations of URL string, corresponding to pattern I and IV. For example, "swu.edu" is most likely to be the hostname of an educational institution named ‘swu’, while "swu.htm" is more likely to be a web page about ‘swu’.

![Figure 1. Framework for malicious URLs detection based on static features.](image)
Detection Model Based on URLs Static Features

Based on the analysis of malicious URLs construction pattern, we present a model of malicious website detection based on URLs static features (shown in Figure 1).

Static Features Extraction

All extracted features are based on statistics, so four datasets are first collected (see Table 1). DS1 is downloaded from the well-known phishing identifying platform - ‘Phishtank’, and we regard it as the standard dataset. DS3 has the same amount as malicious URLs and it used for comparative analysis.

<table>
<thead>
<tr>
<th>Class</th>
<th>Name</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious</td>
<td>DS1</td>
<td>26378</td>
<td>Download from ‘<a href="http://www.phishtank.com/%E2%80%99">http://www.phishtank.com/’</a></td>
</tr>
<tr>
<td></td>
<td>DS2</td>
<td>66232</td>
<td>crawling and collection from Internet</td>
</tr>
<tr>
<td>Benign</td>
<td>DS3</td>
<td>93000</td>
<td>crawling and collection from Internet</td>
</tr>
<tr>
<td></td>
<td>DS4</td>
<td>240000</td>
<td>crawling and collection from Internet</td>
</tr>
</tbody>
</table>

Lexical Features Extraction. To count the lexical frequency of a given URL set, an easy way is to tokenize URLs with delimiters (e.g. ‘/’, ‘-’) and count the number of each word in turn. However, we notice that concatenated words are widely used in URL, e.g. ‘bankofamerica’, ‘volksbank’, ‘hsbc_bank’, in which we more care about the frequency of ‘bank’, but simple tokenize method cannot achieve our goal. So, we choose Longest Common Substring (LCS) method. First, extract all LCS from DS1. Second, select lexical features combined with DS2 and DS3.

Some words have no effect on classification, such as ‘http’, ‘www’ and ‘com’, which can be removed in advance. Second, one word may have different forms which can be normalized based on the idea of ontology, e.g. ‘email’ and ‘mailbox’ can be normalized to ‘mail’. In addition, too short LCS do not have effects on classification, we only extract words with three or more letters. Figure 2 shows the occurrence number of top 20 LCS, from which we know that the usage of LCS accord well with Zipf’s law - the number of lexical occurrences is inversely proportional to their frequency. That is to say, a few words will be frequently used in malicious URLs. Obviously, not all LCS are suitable as lexical features for some words are also widely used in benign URLs. Suppose the number of LCS occurred in malicious samples (DS1+DS2) and benign samples (DS3) is $N_m$ and $N_b$ respectively, where $f(i)$ is the $i$th LCS, we define the normalized ratio as quantitative criterion for feature selection (shown
in Eq. 1). The larger the $\text{Ratio}_i$ indicates the more non-uniform distribution of $f(i)$ in malicious and benign URLs, and $f(i)$ is more suitable to be a lexical feature. We choose 0.5 as threshold.

$$
\text{Ratio}_i = \left| \frac{f(D_i)}{\max_{k} f(D_k)} \right|
$$

(1)

**Grammatical and Token Feature Extraction.** Based on pattern II and III described above, seven Grammatical features was extracted, shown in Table 2. In Token Feature extraction, we collect three kinds of token granularity. Besides of a set of words tokenized by delimiters (a kind of fine-grained tokenization), inspired by [8] and [11], we also extract ‘segment’ and ‘bi-gram’ as token features. Segment means tokenize a URL at a coarse-grained level, while bi-gram is at a modest level. For domain "swu.edu.cn", its bi-gram will be {"swu.edu", "edu.cn"}.

**Vector Construction**

Referring to text classification, we choose VSM as URL vector model. After feature extraction to a URL set $D$, we get a dictionary $R^k$, and $u \in D$ can be regarded as a vector $IV = \langle R^k, W \rangle$, where $r_i \in R^k$ is $i$th feature and $w_i \in W$ is the weight of $r_i$. For some words are more important than others in classification (e.g. ‘virus’, ‘.exe’), to highlight these words, we use TFIDF to calculate $w_i$. TFIDF has a good performance in classification because it takes account of the feature distribution both in document and in corpus. However, before computing TF and IDF, an $nxk$ matrix must be created ($n$ is the number of $D$). Huge $n$ or $k$ may lead to high memory utilization even to memory collapse. An efficient solution is to reduce the value of $k$ and we use the Hashing method. Suppose the hash function $H$ hash $i$th ($i=1, \ldots, k$) feature to position $j$, donated as $H(i) = j$, the term frequency of $i$th feature will be added to $j$th variable $\delta(j)$ (see Eq. 2), and $j$ is artificially specified, usually far less than $k$ ($j << k$). $\zeta$ is a pre-defined hash function aimed at reducing the collision probability. We use HashingVectorizer model provided in Python3.5 to reduce the feature dimension.

$$
\delta(j) = \sum_{i \in H^{-1}(j)} \xi(i) \phi(i)
$$

(2)

In experiments, we will compare and evaluate the performance of TFIDF and Hashing from accuracy and efficiency and analyze their respective scenarios.

**Classification Algorithm Selection**

To select suitable classifier, we have conducted experiments on several classical binary classification algorithms, including Naive Bayesian (NB), Logistic Regression (LR), Support Vector Machine (SVM) and Decision Tree (DT). Three kernel functions used in SVM were tested, donated as SVM-RBF, SVM-Linear, and SVM-Sigmoid. The test data is from Table 1 with a total number of 127683 and be divided into training and testing set in 2:1. The results are shown in Table 3.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>Mean accuracy</th>
<th>Training Time</th>
<th>Testing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>NB</td>
<td>0.964</td>
<td>0.964</td>
<td>0.963</td>
<td>0.964</td>
<td>0.101 s</td>
<td>0.016 s</td>
</tr>
<tr>
<td>LR</td>
<td>0.952</td>
<td>0.953</td>
<td>0.951</td>
<td>0.953</td>
<td>4.796 s</td>
<td>0.006 s</td>
</tr>
<tr>
<td>DT</td>
<td>0.939</td>
<td>0.940</td>
<td>0.962</td>
<td>0.940</td>
<td>144.3 s</td>
<td>0.061 s</td>
</tr>
<tr>
<td>SVM-RBF</td>
<td>0.613</td>
<td>0.783</td>
<td>0.688</td>
<td>0.782</td>
<td>665.1 s</td>
<td>216.5 s</td>
</tr>
<tr>
<td>SVM-Linear</td>
<td>0.960</td>
<td>0.960</td>
<td>0.958</td>
<td>0.960</td>
<td>1536.2 s</td>
<td>152.9 s</td>
</tr>
<tr>
<td>SVM-Sigmoid</td>
<td>0.611</td>
<td>0.782</td>
<td>0.686</td>
<td>0.782</td>
<td>855.3 s</td>
<td>282.9 s</td>
</tr>
</tbody>
</table>

As seen from Table 3, NB is better than other algorithms in terms of Precision, Recall, F1-score and Mean accuracy, and the training time is the shortest. Mean accuracy of LR is a bit lower than NB, but the testing time is the fast. SVM-Linear and DT also have a satisfactory accuracy, but training and testing time is too high to be practical. As a result, NB classifier is selected in our method.

For a given URL $u$, suppose the extracted feature vector is $x_i \in R^k$, by using Bayesian theory, we can calculate the posterior probability of $x_i$ ($i \in \{1, k\}$) to determine which class $C = \{1, 0\} \{\text{malicious, benign}\}$
does $u$ belong to (see Eq. 3). In Eq. 3, $P(C)$ is the prior probability of whether $u$ is malicious (or benign); $P(x_i | C)$ is the appearance probability of feature $x_i$ under the premise that $u$ is malicious (or benign); $\alpha$ is data smoothing coefficient to avoid $x_i$ to be empty, known as the Laplace/Lidstone factor. We choose $\alpha = 0.35$ based on a series of comparative experiments.

$$
    y = \arg \max_{c_j} P(C_j = 1 | u) \prod_{i=1}^{R} P(x_i | C_j = 1 | u)
$$

(3)

**Experimental Evaluation**

**Lexical Features Analysis**

According to Eq. 1, 63 lexical features are extracted at last. We classify these words into five categories based on their semantics (see Table 4). 1) Authentication related words (e.g. 'login') are often used to entice user to take corresponding operations. 2) Management related words (e.g. 'update') are often used to deceive users. 3) The hostname of well-known enterprises, banks or financial institutions (e.g. 'google') are often be mimicked. 4) Some document type names (e.g. 'email') are often used. 5) The words such as ‘rand’, ‘id’, ‘cmd’ are often used as the anchor name. These habits are consistent with our intuition, indicating that these Lexical features are representative.

<table>
<thead>
<tr>
<th>Class</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication related</td>
<td>login, user, admin, verify, verification, validate, auth, confirm, submit, secure, sign, client</td>
</tr>
<tr>
<td>Management related</td>
<td>update, admin, access, import, content, help, load, cache, recovery, server, drive, session, redirect, manage</td>
</tr>
<tr>
<td>Well-known websites related</td>
<td>google, dropbox, dpbx , alibaba, amazon, paypal, adobe, apple, facebook, gucci, hotmail, outlook, yahoo, wells Fargo, account, chase</td>
</tr>
<tr>
<td>File related</td>
<td>inc, mail, image, img, exe, pdf, png, js, blog, doc, arch, file, css, app, plugin</td>
</tr>
<tr>
<td>Query related</td>
<td>rand, id (fid, userid), err (error), page, res, cmd</td>
</tr>
</tbody>
</table>

**Comparison of Vectorization Method**

For comprehensive assessment of TFIDF and Hashing method, we choose TF for comparison and use Naive Bayesian as machine learning algorithm. We know from the result that Hashing method has the highest efficiency in both training and testing (see Figure 3), while TFIDF is the best in accuracy and higher than TF and Hashing about 2% and 4-6% respectively (see Figure 4). Therefore, TFIDF is the most suitable vectorization method when the amount of data is not too big, while Hashing can be used when there is a need to make a trade-off between accuracy and efficiency.

![Figure 3. Detection time comparison.](image)

![Figure 4. Accuracy comparison.](image)

**Comparison of Classification Result**

In order to verify the validity of selected features, we choose three group of features to compare their contribution to the classification accuracy, which are Token feature (TF), Token and Grammatical...
feature (TGF) and full feature (three kind of static features, donated as TGLF). We can see from Figure 5, 1) Only Token feature can achieve the accuracy of 94.8-97.3%, which shows the effectiveness of combining segment and bi-gram features. 2) Adding Grammatical and Lexical features can steadily improve the accuracy, it proves these features have a good effect on classification. 3) When the training number reaches about 50%, its contribution to accuracy began to weaken, which means that it is not desirable to improve the accuracy through increasing the number of training samples infinitely.

Figure 6 shows the training and testing time under different datasets in full feature. It can be found that learning time increases linearly with the number of training data while testing time is relatively stable, which means a trained NB classifier can give the detection result to a new URL in a relatively fixed and short period of time. It shows the efficiency of NB algorithm.

Summary

In URL-based malicious website detection, the quality of feature representation is critical for the success of a machine-learning model. Experiments proves that it can get both high accuracy and high efficiency using the static features extracted by this paper.

Acknowledgement

This research was financially supported by Natural Science Foundation of Southwest University (No. SWU112017) and Guangxi Cooperative Innovation Center of Cloud Computing and Big Data (No. YD16E04, YD16509).

References


