A Lexical Approach for Classifying Malicious URLs

Michael Darling & Greg Heileman
Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM, USA
{*michaelcdarling, gregheileman*}@*gmail.com*

Gilad Gressel, Aravind Ashok, & Prabaharan Poornachandran
Amrita Center for CyberSecurity
Amrita Vishwavidyapeetham University
Vallikavu, Kerala, India
giladgressel@gmail.com, {*aravindashok, praba*}@*am.amrita.edu*

***Abstract*—Given the continuous growth of malicious activities on the internet, there is a need for intelligent systems to identify malicious web pages. It has been shown that URL analysis is an effective tool for detecting phishing, malware, and other attacks. Previous studies have performed URL classification using a combination of lexical features, network traffic, hosting information, and other strategies. These approaches require time-intensive lookups which introduce significant delay in real-time systems. In this paper, we describe a lightweight approach for classifying malicious web pages using URL lexical analysis alone. Our goal is to explore the upper-bound of the classification accuracy of a purely lexical approach. We also aim to develop a scalable approach which could be used in a real-time system. We develop a classification system based on lexical analysis of URLs. It correctly classifies URLs of malicious web pages with 99.1% accuracy, a 0.4% false positive rate, an F1-Score of 98.7, and 0.62 milliseconds on average. Our method also outperforms similar approaches when classifying out-of-sample data.**

I. Introduction

The Internet provides a wide-reaching platform for fraud. It can be extremely difficult for Internet users to be on guard for the latest scams, malware (malicious software), and other malicious activities. Therefore, there has been much study dedicated to protecting Internet users from harm. Machine learning algorithms have been shown to be an effective way of detecting maliciousness [1]. Previous work has implemented several effective classification systems for malicious web page detection [2], [3].

Internet users have come to expect quick results: they may become impatient if there is any extra latency in content delivery. Therefore, any classifier that is built to protect the user from malicious content must deliver its decisions quickly. However there is a tradeoff between efficiency and effectiveness. We must decide how much safety we are willing to sacrifice in order to deliver content as seamlessly as possible.

The reality is that the most accurate approaches for classifying malicious web pages tend to take the most time. Dynamic classifiers execute the page thoroughly to look for malicious content in the behavior of the page (e.g. honeypot clients). This is necessary when negative content is obscured in a sophisticated way. However, due to the resource and time-consuming nature of this approach, it is not compatible with the uninterrupted service expected by the user. In other words, the task is to quickly detect malicious web pages while maintaining a high degree of accuracy [4].

Static approaches attempt to classify a web page based on its URL and content without executing the page. As such, classification based on static features is efficient and scalable. However, classifiers built to analyze static features have limited success with highly sophisticated schemes such as obfuscated JavaScript based drive-by-downloads [5]. Furthermore, the characteristics of malicious web pages differ based on the type of exploitation techniques used (e.g. phishing, drive-by-downloads, and infected adware networks).

Savvy Internet users can often detect, for example, phishing scams with a quick glance at a link embedded in an email. Machine learning techniques can do the same with a high degree of accuracy. It has been shown that URLs analysis alone can provide a means to detect malicious content [1].

Previous studies have built classifiers that detect malicious websites using a combination of URL lexical features, network traffic, hosting information, and other strategies [6], [7]. These studies used lexical features with a bag-of-words approach that yields very large feature vectors (on the order of hundreds of thousands). These studies also made use of URL features which required hosting information to be looked up on a remote server. The use of these lookups introduces significant latency when classifying URLs. This latency precludes the use of these approaches in a real-time system.

In this paper we present an approach which uses an n-gram model to develop a new classification system that adheres to the strict time-constraints required for a real-time system. The system increases accuracy on out-of-sample testing data while maintaining overall classification accuracy comparable to previous work [1]–[3], [6]. Our approach uses the J48 decision tree algorithm to perform classification of URLs using 16 features extracted from an n-gram model and 71 features from other lexical properties. J48 is an open-source implementation of the C4.5 algorithm [8].

The summary of our contributions are:

- We achieve a 99.1% classification accuracy with .627 millisecond average classification time when run on a 1.8GHz system.
- We apply an n-gram model to malicious web page classification.
- We build a URL classifier strictly with lexical features that detects both phishing and malware attacks.
- We improve classification accuracy of out-of-sample data compared to similar previous approaches.

This was achieved while training on a data set of 131,402 URLs (50% benign, 50% malicious).

TABLE I. Sample of Average URL Characteristic Values

Feature	Benign	Malware	Phishing
Length of URL	64.03	58.93	101.8
Length of Parameters	14.91	10.65	21.75
Longest Token in Hostname	7.083	8.39	12.60
Count of most frequent token in URL	0.425	0.239	0.351
Black-List Word Count in URL	0.112	0.121	0.939
Digit Count in Hostname	0.336	0.546	1.059
Dashes Count in Path	1.252	0.780	0.285
Vowel/ Consonant Ratio in Hostname	0.447	0.391	0.552
Digit / Letter Ratio in Hostname	0.021	0.061	0.041
Digit / Letter Ratio in URL	0.156	0.166	0.194

II. Approach

We first explain our general approach, discuss the features we utilized, the data we collected, and our selection of the J48 algorithm.

The goal of this study was to explore how quickly and accurately we could classify malicious web pages using only their associated URLs. This minimal approach leads to the fastest possible classification time since all the features are extracted from the characters of the URL and not by visiting the web page itself. Therefore we collected URLs from blacklisted and trusted websites, labeling them malicious and benign respectively.

We collected two kinds of malicious URLs: phishing and malware. Phishing URLs, in order to collect private information, are designed to invite users to click on links that lead to fraudulent web pages. Sites that contain malware seek to infect the user's system by downloading and executing harmful programs–users are often infected without realizing it.

Table I shows a sample of average feature values for our dataset. The distinguishing characteristics between the three types of URLs are what allows a machine learning approach to be effective in this problem space. Phishing URLs are often filled with seemingly legitimate tokens which is why they have more tokens overall. Malware URLs are often abandoned frequently in order to avoid detection and are not typically designed to attract clicks. Therefore Malware authors seek easily available domain names which often do not contain English words. Thus there are differences in the ratios and lengths when compared to benign or phishing URLs.

Malicious content has been detected by analyzing static components of web-pages as well as the dynamically executed content. The more efficient static methods include the collection of http header information, html tags and host-based features at classification time. We chose not to extract any information beyond the URL of the web page. While this additional information would likely yield higher classification accuracy [6], by eliminating the collection of features found on the page, the classification time is greatly reduced.

Furthermore, most previous work all relied on publicly available information such as WHOIS and geolocation. Due to the lookups required to obtain these features, these approaches require a minimum of a few seconds to classify a single URL. Further complicating this problem is that WHOIS servers limit the number of requests over a given time period.

As a baseline we implemented a bag-of-words model based on previous works [1], [9] while removing all time-consuming components. Unlike previous approaches, our system utilizes an n-gram language model which improved on the performance of the bag-of-words model when classifying out-of-sample data (data of a type that the classifier has not trained on).

A. Modeling the Language of URLs

The goal of a language model is to assign a probability to any sentence of a particular language. A good language model will assign high probabilities to common and grammatical sentences while assigning low probabilities to uncommon and ungrammatical sentences. Building on this concept we attempted to construct a model of the *language* of *benign* URLs. A standard language model approach would calculate the probability of that a given URL is benign. However, we calculate what we call the *similarity* value of a given URL. The *similarity* value of a URL measures how closely its properties are related to a URL in the benign set (this concept is further explained in the n-gram section).

A common method when constructing language models is to parse n-sized grams of the dataset. The grams can be words, phonemes, syllables, or letters. N-grams are built using Markov chains. Markov chains make the assumption that the probability associated with any event depends only on the probability of the events directly preceding it. This concept is called the independence assumption. A first order Markov chain assumes that the probability for the occurrence of a given gram is conditioned only by the preceding gram. A second-order chain calculates the probability based on the preceding two grams (in this case the previous state is defined as the preceding two grams). These chains are referred to as bigrams and trigrams. We used n-gram models to calculate the probability of a sequence of characters occurring in a URL. Much research has shown this model to be effective in the task of language modeling and difficult to improve on [10].

While modeling URLs with an n-gram approach has been done [11], this method has not, to our knowledge, been been previously applied to malicious URL detection.

B. Bag-Of-Words Model

As a baseline we followed previous studies [1], [6], [7], [9] by building a binary bag-of-words model which tokenizes the URL. The bag-of-words is a set that contains every single unique token in the training data. The feature vectors contain an element representing each token in the bag. If a given URL contains a token found in the bag, the corresponding element in the feature vector is given the value of 1. This creates highly sparse vectors which contain all zeros except for a few ones which represent the tokens of the given URL and the other chosen lexical features. On average our training data contained 122,000 binary features when using the bag-of-words model.

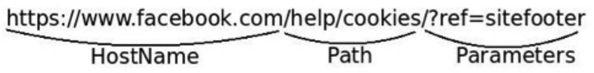

Fig. 1. URL Components

C. Features

We implemented and expanded upon features based on the previous work of Ma et al. [1] and Whittaker, Ryner, and Nazif [12].

Phishing URLs, tend to contain a lot of key words and symbols out of place since they are designed to deceive users. For example, it is common to see trusted brand names such as “paypal” or “google” distributed throughout a phishing URL. In order to capture this observation we created a feature which performs similarity checks on common domain names, another feature searches for any brand names that are out of place.

Following Blum et al. [9], we separated the URL into three sections. We chose to separate the URL into hostname, path, and parameter components (see Figure 1). Each component was then further separated into tokens. Hostname tokens are separated by dots, path tokens by slashes, and parameter tokens by question marks. Within each section we further tokenized strings using the delimiters specified in Ma et al. [1], namely ‘/’ , ‘?’ , ‘.’ , ‘=’, ‘-’ and ‘_’.

We separated the URL into component parts in order to preserve the context of these tokens. A feature will yield a different value when extracted from the hostname, than the same feature in the path. For example, the average number of dots will be 2 or 3 in a hostname, while often 0 in the path. We calculated nearly all features on all three sections of the URL as well as the entire URL.

In total we developed 87 features, which are categorized into five groups: n-grams, lengths, counts, binaries, and ratios.

1) n-grams: In our analysis we implement bigrams, trigrams and fourgrams. We also calculate a unigram probability which is context-independent. We created our n-gram models for the benign training data only.

To calculate any given n-gram, we divide each URL of the benign set into strings of a particular size. More specifically, a unigram model maps single characters whereas a bigram model maps two consecutive characters (e.g., ‘i’ and ‘it’ respectively). The unigram model does not implement Markov chains, it calculates probability based on the frequency distribution of single character grams, the bi-gram model implements 1st-order Markov chains.

In general, we define δ as an n-gram string of length L (where $L = 1$, specifies a unigram model, $L = 2$, specifies a bi-gram model, and so on), we generate an n-gram model to contain all unique occurrences of strings of size L for a given section. Therefore the n-gram map of our training data is represented by the set:

$$\Delta = \{\delta_1^L, \delta_2^L, \delta_3^L, \ldots, \delta_m^L\}$$

where each δ_i is a unique n-gram and m is the total number of unique L-sized grams found in the data. Let θ be the total number of occurrences of each δ in the entire data set D.

We now have a set Θ where:

$$\Theta = \{\theta_1^L, \theta_2^L, \theta_3^L, \ldots, \theta_m^L\}$$

Where each θ_i represents the number of occurrences of its corresponding δ_i.

Once we have a map Θ we can calculate the probability ρ_i for any given θ_i. We calculate the unigram frequency by dividing the total occurrences of δ by the sum of all $\theta^{L=1}$. To calculate a bigram probability of a given δ_i occurring in any URL in D we take the quotient of two grams. Where the numerator is the given bi-gram and the denominator is the unigram which makes up the first half of that bi-gram. The general equation for any ρ_i is given here

$$\rho_i = \begin{cases} \dfrac{\theta_i^L}{\sum_{i=1}^{m} \theta_i^L} & \text{if } L = 1 \\ \dfrac{\theta_i^L}{\theta_i^{L-1}} & \text{if } L > 1 \end{cases}$$

Given any δ^L we calculate its probability by conditioning it with the δ^{L-1} sequence which is contained within it. For example, the probability of a string “ate” depends entirely on the probability of string “at”. In other words we calculate the probability of “ate” occurring, given that we know “at” has already occurred.

We can now represent the probability of a single occurrence of any δ_i with the set

$$P = \{\rho_1^L, \rho_2^L, \rho_3^L, \ldots, \rho_m^L\}$$

We calculated our n-gram models using the benign portion of the data set only. The intuition is that if we model probabilities on only one class, the other class will have lower values for the feature, which will aid the classifier in its decision. This approach is similar to a one class classification problem.

By modeling on the benign portion of the data we capture two key insights. First, the set of benign URLs changes at slower rate than the set of malicious URLs. Second, malicious data will have very low values for these features, this increases the classifier’s performance on out-of-sample data (see table V).

For each test we built the n-gram model dynamically based on the samples in the training data for that given test. This way the model contained no prior information on the testing data.

In order to assign real feature values, each URL was parsed into segments of size L. We then looked up each value of ρ_i^L and *summed* them all together. If a value was not found in P^L, the lookup table, we simply added 0. After summing all ρ_i^L we divide by total number of δ present in the URL, this effectively normalizes the n-gram feature value by the length of the URL.

The n-gram features are *not* probabilistic representations of the URL. A probabilistic feature would have summed all log-probabilities of δ_i^L. We tested this approach, however we achieved higher results with the summation of the probabilities of δ. Our choice of probability addition yields the union of the probabilities for all δ_i:

$$Similarity(URL) = \frac{\rho_1 \cup \rho_2 \cup \ldots \cup \rho_k}{k}$$

Where k is the number L-sized grams in the given URL. For example the bi-gram similarity value of "google.com" would be:

$$Similarity(google.com) =$$

$$\frac{\rho(go)+\rho(oo)+\rho(og)+\rho(gl)+\rho(le)+\rho(e.)+\rho(.c)+\rho(co)+\rho(om)}{9}$$

Summation, or what we will call *similarity*, resulted in higher classification accuracy (see Table VIII).

Besides those based on the n-gram models, the rest of our feature set includes:

2) Length Features: We implemented a total of 10 length features. These features are the length of the: hostname, first-directory, URL, path, parameters, top-level domain, and second-level domain. We also calculated the longest-token in the hostname, path, parameter and URL.

3) Counting Features: count occurrences of a character. Some examples of character counts are the number of: delimiters,'-', '@', '?', '.', '=', '%', 'http', 'www', digits, numbers, letters, tokens, non-alpha numeric characters, and directories. We implemented a total of 29 count features.

4) Pattern Features: look for specific patterns within the URL and count occurrences of that pattern. Pattern counts were performed on all three subsections of the URL as well as the entire URL. Some examples are: case changes, most consecutive occurrences of a character, most frequent token, similarity to blacklist words, blacklist word count. Our list of blacklist words was drawn from Garera et al. [13]. We implemented a total of 15 pattern features.

5) Binary Features: are the following: '.com' out of place, IP address for a hostname, extract the similarity to alexa-top-10 domains, and alexa top-10 domain out of place. We used the Damerau-Levenshtein algorithm to check the edit distance in order to detect phishing attacks that use domain squatting (e.g., gooogle.com). We checked distances from the top-10 alexa domains. Domain-out-of-place determines if a domain word from the Alexa top-ten list is not in its correct position. For example the URL "www.malicious.com/www.google.com" shows "google.com" (the number one Alexa site as of this writing), as out of place.

6) Ratio Features: calculate the ratios between different types of characters or tokens. Some examples are: vowel / consonant ratio, digit / letter ratio, average length of tokens for each subsection and the entire URL. There are a total of 12 ratio features.

We implemented a total of 87 features for our decision tree based system.

D. Data Sets

We drew our data from six sources. Phishing data was collected from Phishtank.com and OpenPhish.org. Both web sites contain block lists of phishing URLs. The Phishtank data is verified by human users. Malware data was drawn from two block lists: MalwareDomainlist.com and Malware-Domains.com. The malware lists provide only domain names. Therefore we configured a web crawler to extract links from web pages contained on the Malware domains. From this set, we removed any URLs which pointed to a domain outside the block list since malicious pages may contain links to trusted domains. Our training data contains a total of 68,031 malicious URLs.

To collect our benign data set, we configured our crawler to collect links from the Dmoz.org directory and the Alexa.com top 1000 domains. DMOZ is a manually managed directory of the entire Internet. DMOZ links are posted by approved human editors. Alexa.com ranks websites based on traffic. As with previous studies [5], [14], we assumed the most highly trafficked websites to be the most highly scrutinized, and therefore, safe. However, since pornographic sites tend to be sources of malware, and yet highly trafficked, we manually removed any site whose domain name suggested the presence of pornography. In total, our crawler obtained 122,550 benign URLs, a breakdown is shown in Table II.

TABLE II. TRAINING DATA

Source	**URLs**	**Subset**	**Training Set**
Alexa	105,806	Benign 122,550	Benign 122,550
DMOZ	16,744		
Phishtank	17,531	Phishing 25,388	
OpenPhish	7,857		Malicious 68,031
MalwareDomains	39,230	Malware 42,643	
MalwareDomainList	3,413		

Finally, we cross-referenced the domains from the malicious and benign sets, if any domain was present in both sets, we removed them entirely from our training data. This became necessary since a highly trafficked Chinese website, which is ranked in the Alexa top 1000, had been reported as distributing malware on one of the block lists.

E. Classification Algorithms

In this study we chose to explore several classification methods. As our baseline we built a linear classifier using regularized logistic regression. Logistic regression is a parametric model for binary classification where examples are classified by their distance from a decision boundary. We implemented the L1 regularized logistic regression model from the LibLinear package as was done by Ma et al. [1].

For the implementation of the classifier based on n-gram modeling we used the J48 algorithm (an implementation of the C4.5 decision tree). J48 is one of many classification algorithms available in the popular Weka machine learning suite [8]. We chose J48 due to its reputation of success in this domain [15] and after evaluating its performance in comparison to Bayesian Logistic Regression, Logistic Regression, Naive Bayes, and K-Nearest Neighbors classifiers. In order to evaluate performance of each algorithm, we compared their

TABLE III. ROC AREA UNDER CURVE VALUE

Classifier	AUC
NaiveBayes	91.1
BayesianLogisticRegression	92.8
LogisticRegression	98.7
Knn	99.0
J48	**99.3**

overall accuracy and receiver operating characteristic (ROC) curve.

A classifier's accuracy is not the only necessary metric to evaluate its performance. The ambiguity lies in the fact that classification accuracy will have equal misclassification costs: an accuracy of 99% tells nothing about the false positive and negative rates. In fact, in a real-world deployment of any classifier, it is often true that the error rate of one class of data comes at a higher cost than misclassifying other types of data. For the problem of malicious web page detection, a false-negative is potentially much more harmful than a false-positive since it could result in an infected system.

An ROC curve shows the false-positive and false-negative rates on the X and Y axes respectively. Therefore, the curve represents the predictive quality of the classifier independent of error costs (and class imbalance in the training data) [16]. In order to choose our classifier we examined the ROC Area-Under-the-Curve value (AUC) and accuracy rate of each type of classifier. Table III shows J48 outperforming its counterparts in AUC of its ROC. Furthermore, it achieved the highest overall classification accuracy for the n-gram model.

F. J48

J48, a top-down single decision tree, is a recursive algorithm which seeks to best divide the data based on its attributes. It creates child nodes to split the data based on the highest value of either the information gain or gain ratio of the given attributes. The selection of splitting criterion is a free parameter choice.

Let T be the the set of choices of size S at a particular node. Then the information gain is defined as:

$$gain = info(T) - \sum_{i=1}^{S} \frac{|T_i|}{|T|} \cdot info(T_i)$$

where $info(T)$ is the entropy function:

$$info(T) = -\sum_{j=1}^{n} \frac{freq(C_j,T)}{|T|} \cdot log_2(\frac{freq(C_j,T)}{|T|})$$

where n is the number of data classes, and C_j is a particular class [17].

Choosing information gain minimizes the entropy found in the children of the split. However, this tends to reward splits which contain large numbers of remaining samples. Information gain ratio, the default setting for J48, divides the information gain by the information provided by the children of the split [18]:

$$Split(T) = -\sum_{i=1}^{S} \frac{|T_i|}{|T|} \cdot log_2(\frac{|T_i|}{|T|})$$

This process continues until J48 hits one of its base cases which are:

- All remaining samples belong to the same class
- Information gain ratio is zero for all features
- A class value is seen for the first time

If a feature does not provide the best split of the data at any of the nodes in the tree, it is never used. Both during and after the construction of the tree, J48 implements pruning features in order to avoid overfitting the training data.

G. Effectiveness Metrics for Features

As we implemented each feature we compared the effectiveness of the classifier to its previous iteration using the following effectiveness metrics: F1-score, accuracy, false-negative rate (FNR), and false-positive rate (FPR). Accuracy was measured on the entire data set, whereas the F1-score, FNR, and FPR are defined from the point-of-view of the malicious data. Therefore, accuracy is simply the percentage URLs that are correctly classified, FPR rate is the percentage of benign URLs classified as malicious, and FNR rate is the percentage of malicious URLs classified as benign.

III. RESULTS

We implemented a bag-of-words model with L1 logistic regression (LR) in order to have a base-line comparison with previous work. The overall classification results were quite similar as seen in Table IV. However additional tests (see Table V) show that the J48 model outperforms LR-based model when testing on out-of-sample data.

Table IV shows the results of both models tested with 10-fold cross validation using a 1:1 ratio of benign to malicious URLs on the entire training data. FPR and FNR are the malicious false positive and false negative rates respectively. Both classifiers take less than one millisecond to classify a URL. All tests were run on a computer with a 1.8 GHz Intel i5 processor and 8GB of memory.

TABLE IV. CLASSIFIER PERFORMANCE, 10-FOLD CROSS VALIDATION ON TRAINING SET SIZE OF 131,520.

	Accuracy	F1-Score	FPR	FNR
J48	99.1	98.9	1.7%	0.4%
LR	99.1	98.7	1.7%	0.5%

The tests shown in Table V demonstrate that the J48 n-gram model performs with high accuracy on out-of-sample malicious URLs. The results in Table V are the malicious True Positive Rates (TPR) when training on one source of malicious data while testing on another. MDomains refers to data drawn from the source MalwareDomains.com, MDList is

TABLE V. Train with a malicious dataset, test with another. True Positive Rate

		Trained							
		MDomains		MDList		OpenPhish		PhishTank	
		J48	LR	J48	LR	J48	LR	J48	LR
Test	Mdomains	99.5	99.6	73.2	75.2	77.8	9.1	80.8	37.4
	MDList	88.3	71.6	99.4	99.1	84	21.3	89.3	54.8
	OpenPhish	65.8	66.2	95.8	25.2	99.2	98	95.6	89.7
	PhishTank	86.3	56.6	92.9	30.2	95.6	89.7	98.5	97.5
	total avg	84.975	73.5	90.32	57.42	89.15	54.52	91.05	69.8

data from MalwaredomainList.com, OpenPhish data is drawn from openphish.com and PhishTank from phishtank.com.

By training on one source of data and testing against another, we simulate testing the classifier's performance on out-of-sample data. J48 performs with higher accuracy in the majority of cases. The Logistic Regression system has a lower TPR for a majority of the tests. Overall these tests show that the n-gram model with J48 generalizes to new data with higher accuracy than bag-of-words with Logistic Regression.

Table VI shows the confusion matrix for the J48 model with 10-fold cross validation on the entire data set. Each row represents the instances of an actual class, each column represents the instances of the predicted class by J48.

TABLE VI. Confusion Matrix : P = predicted value, R = real value.

	Malicious - P	Benign - P
Malicious-R	65316	444
Benign-R	452	65333

A. Feature Effectiveness

Table VII shows the top 25 features ranked by information gain ratio. These scores are calculated for the entire training data, which is the same calculation that J48 makes at the root of the tree. Even though some features may provide little information at the top of the tree, J48 re-calculates the information gain ratio before the creation of each node. Thus some of the lower ranked features provide more information lower in the tree.

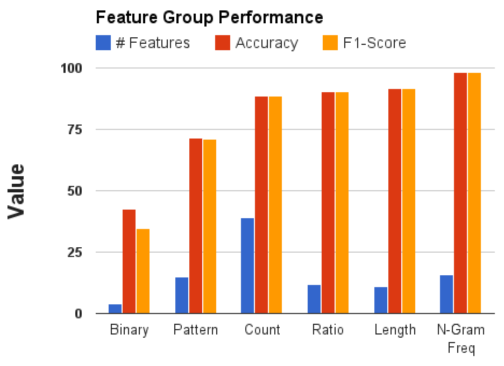

Fig. 2. Performance of J48 Classifier When Trained and Tested on Individual Feature Categories.

TABLE VII. Features Ranked by Information Gain Ratio on the Training Data

Rank	Feature	Info Gain Ratio
1	HostName_QuadGram	0.8544122
2	HostName_TriGram	0.7644311
3	HostName_BiGram	0.7359932
4	HostName_UniGram	0.7325711
5	URL_QuadGram	0.3948905
6	Path_QuadGram	0.1502406
7	Parameters_QuadGram	0.145088
8	URL_TriGram	0.1243128
9	Vowel/Consonant_Hostname	0.1225869
10	Path_TriGram	0.1151544
11	LongestToken_Hostname	0.1113489
12	AverageLengthOfTokens_Hostname	0.1051401
13	Path_BiGram	0.1011293
14	Path_UniGram	0.0905633
15	Length_Hostname	0.0846729
16	Parameters_TriGram	0.081808
17	Length_2LD	0.0795415
18	LongestToken_URL	0.0763964
19	LongestToken_Path	0.0739061
20	Digit/Letter_Path	0.072608
21	AverageLengthOfTokens_Path	0.0672172
22	AverageLengthOfTokens_URL	0.0662559
23	Length_URL	0.0606642
24	Number_NonAlphaChars_URL	0.0604375
25	Length_Path	0.0568101

In order to quantify the effectiveness of each feature type, we trained and tested the classifier with each feature group separately: count, length, binary, ratio, n-gram, and pattern. The results are shown in Figure 2. The n-gram group has the highest accuracy of 98.23% which correlates with Table VII.

1) N-Gram Similarity Features: Our first attempt at utilizing the n-gram model was to sum the log-probabilities of the n-grams. However, the accuracy was increased when these features were calculated using the summation of the probabilities themselves. We call this feature the similarity value of a URL. In summary: rather than calculating the probability that a given URL belongs to the benign set, this feature represents the similarity of the test URLs to the URLs in the benign set.

Table VIII shows the results of the J48 trained with the n-gram model using both the similarity and a probability approach. When calculating the probability value we used log probabilities to avoid underflow.

TABLE VIII. J48 N-Gram model: Training on Similarity Value Features vs Probability Value Features

	Accuracy	F1-Score	FPR	FNR
Probability	96.489	96.5	0.035	0.035
Similarity	99.01	99	0.012	0.006

B. Tuning

It is possible to tune the classifier to reduce the number of false positives or false negatives. This is done by adjusting the ratio of benign to malicious samples in the training data. An example application of tuning would be the integration of the classifier with a firewall. An administrator might want to tune the classifier to keep false positives low to reduce complaints about blocked sites. Otherwise, if it is desired to keep false negatives low, to reduce the spam and viruses in the network, this could be done as well. Table IX shows the results when tuning the J48 classifier. As the number of malicious samples is increased the false negative rate is decreased.

TABLE IX. CLASSIFICATION TUNING RESULTS. TOTAL TRAINING SAMPLES : 65,679

Ben: Mal	Acc	F1-Score	FPR	FNR
3 : 1	98.96%	99.0	1.0%	1.1%
1 : 1	98.8%	98.9	1.3%	0.8%
1 : 3	97.7%	97.8	3.3%	0.4%

C. Dataset Validation

Since many of our URLs come from the same domains, there are duplicate portions contained in multiple URLs. If there are several URLs in the training set that contain identical tokens to URLs in the testing set, this could overweight certain features within the classifier.

TABLE X. TEST WITH UNIQUE DOMAIN NAMES, TRAINING SIZE: 12,000 URLS

Classifier	Acc	F1Score	FPR	FNR
J48	94.76	94.7	9.8	1.7
LR	96.3	96.79	4.7	3.8

In order to test against this, we created a data set where all domains were unique. This increased the variation between data samples. The results of this test are shown in Table X. It is notable that the two models differ significantly in their FPR and FNR rates. This could be a deciding factor on which method to implement. The lowered accuracy of both classifiers can be accounted for in the reduced training size (12,000 URLs). If a larger training set of unique domains was constructed (on the order of 120,000), the accuracy of both classifiers would improve.

IV. RELATED WORK

Historically, the most widely adopted method for identifying malicious web pages has been the construction of searchable blacklists such as the Google Safebrowsing API. However, blacklists can quickly become outdated as scammers tend to obtain and abandon domains rapidly. Blacklists have shown to be weak in the zero hour of a phishing campaign [19] since they can take hours or days to update. Blacklist features have also been shown to be less effective than URL lexical analysis [1].

Training a classifier to detect phishing attacks has been subject to numerous studies. One approach has been to combine host-features with URL lexical analysis to form a static classifier [1]–[3], [7]. Host-based feature sets include IP address, domain name, geo-location, and WHOIS properties.

Blum et al. [9] developed an online confidence weighted classification model with URL lexical features alone. This model differs from ours in that it uses a much larger binary bag-of-words approach that yields vectors of 369,585 features. This work was largely based on Ma et al. [20] with the major difference being the absence of host based features.

Le, Markopoulou, and Faloutsos [6] studied the effectiveness of using only lexical features vs. combining them with additional features. They also researched the difference between batch classifiers and online learning classifiers in this problem space.

Canali et al. [2], created Prophiler, a static classifier which acts as a filter for a dynamic classifier. Prophiler is designed to filter the majority of benign pages while maintaining its priority of low false-negatives. This method reduces the computational load on a subsequent honey-client.

Xu et al. [4] implemented a cross-layer approach wherein corresponding network layer data was utilized for feature extraction. Their goal was to add network data to extend the accuracy of a static analysis. They obtain speeds of 4.9 seconds for feature extraction and classification of previously unseen URLs.

Thomas et al. [7], developed Monarch, a real-time URL classification system which classifies web pages using features from host information, the URL, javascript events, HTTP header information, plugin-usage, redirects, HTML tags, DNS information, geo-location information, routing data, page links, and pop-up windows. Their vectors contain 50 million distinct features. This system was designed to classify 15 million URLs a day. Monarch achieves an overall classifcation accuracy of 91% and takes 5.54 seconds to classify a single URL on average.

With the exception of Thomas et al [7] and Le, Markopoulou, and Faloutotsos [6], all of the previous work focuses on either classifying malware or phishing, but not both. No previous work has focused on a system for a lightweight real-time application and none have attempted a language model other than bag-of-words.

V. EVASION

A fundamental flaw in the study of malicious website classification is that attackers have access to most of the same data that researchers use to train detection systems [15]. Therefore, attackers can design their attacks to evade detection. A purely lexical approach relies on common characteristics of malicious URLs in order to differentiate them from benign URLs. If an attacker were to manipulate their URLs to avoid the features that detection algorithms use to detect maliciousness, it would be possible to evade detection. However, there could be a cost associated with changing these features depending on the intended use of a particular URL.

A. Evading detection with phishing URLs

URLs used for phishing attacks are designed to lure a human user into believing that a link is legitimate. For example, familiar brand names such as "google" or "facebook" are commonly distributed throughout a phishing URL. Therefore, in order to detect these URLs it is useful to look for an overuse of tokens that would normally be associated with legitimate websites. Another strategy is to determine whether these seemingly legitimate tokens are out of place (e.g., facebook.com should be classified as legitimate, whereas facebook.xyz.com should be classified as malicious). In order to evade this particular feature, attackers would need to remove the brand name tokens. However, the removal of these tokens would reduce the likelihood of fooling the user. Thus by attempting to avoid detection through this route the attackers would also be lowering the effectiveness of their attack.

B. Evading detection with malware URLs

Unlike phishing URLs, Malware URLs are not always designed to deceive. These URLs are often landed on after redirection from other domains. Therefore, the lexicon of malware URLs is less intrinsic to the attack itself. At the present time, our charactization of malware URLs seems to suggest that attackers put very little thought into the tokens seen in URLs leading to malware. However it is clear that malware URLs do not currently look like regular URLs, as long as this remains true, the n-gram Model will detect all things that are *not* benign.

C. Strength of n-gram Modeling

Many of the features in our system are based on n-gram models of the most common benign URLs. This allows the system to assign a similarity value to the likelihood that a URL is benign based on the number of legitimate tokens that are present. In order to evade detection, the characteristics of Malicious URLs will continue to change. However, since the characteristics of malicious URLs, will always differ from benign URLs by some degree of similarity, given regular retraining, the n-gram models will be able to detect URLs that are dissimilar from benign URLs.

VI. Conclusion and Future Work

In this paper we have presented a novel, lightweight approach for classifying malicious URLs. Once built, the classifier can process and classify a URL with an average speed of 0.627 milliseconds and an accuracy of 99.1%. Unlike previous work, this level of accuracy is achieved without time-consuming lookups and is capable of detecting two different types of malicious URLs. This is important as we envision our classifier being integrated with an existing real-time security system such as a firewall, browser extension or as a pre-filter to a comprehensive detection scheme.

Though the our classifier performs at a high level in its niche, it is not without its share of limitations. URL shortening services are becoming increasingly popular both with attackers and the general public. When a user clicks on a shortened URL, it redirects to the full length URL of the page. Our system is designed to measure the features of a full length URL only. Currently, our system rejects shortened URLs. We would ideally add a component to our system that would follow the shortened URL, pull the full-length version and pass it to the classification system.

In the future we would add a component to the system that specifically targets domain names that were created by Digitally Generated Algorithms (DGA) since a DGA domain name suggests the presence of malicious activity.

The characteristics of malicious web pages and URLs will continue to evolve. As with any security system, ours will need to keep up with the latest trends. We will need to implement a component of the system that would regularly retrain the classifier with the latest benign data. Though the set of highly trafficked benign websites does not change very quickly, regular retraining will be necessary.

References

[1] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious web sites from suspicious urls," in *Proceedings of the SIGKDD Conference*, 2009a, pp. 1245–1254.

[2] D. Canali, M. Cova, G. Vigna, and C. Krugel, "Prophiler: A fast filter for the large-scale detection of malicious web pages," in *Proceedings of the International World Wide Web Conference*, March 2011, pp. 197–206.

[3] G. Xiang, J. Hong, C. Rose, and L. Cranor, "Cantina+: A feature-rich machine learning framework for detecting phishing web sites," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 2, p. 21, 2011.

[4] L. Xu, Z. Zhan, S. Xu, and K. Ye, "Cross-layer detection of malicious websites," in *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 2013, pp. 141–152.

[5] K. Rieck, T. Krueger, and A. Dewald, "Cujo: Efficient detection and prevention of drive-by-download attacks," in *Annual Computer Security Applications Conference (ACSAC)*, 2010, pp. 31–39.

[6] A. Le, A. Markopoulou, and M. Faloutsos, "Phishdef: Url names say it all," in *Infocom*, 2010, pp. 191–195.

[7] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time url spam filtering service," in *IEEE Symposium on Security and Privacy*, 2011, pp. 447–462.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[9] A. Blum, B. Wardman, T. Solorio, and G. Warner, "Lexical feature based phishing url detection using online learning." in *the 3rd Workshop on Artificial Intelligence and Security*, 2010, pp. 54–60.

[10] F. Jelinek, "Up from trigrams." Eurospeech, 1991.

[11] M.-Y. Kan and H. O. N. Thi, "Fast webpage classification using url features," in *Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM, 2005, pp. 325–326.

[12] C. Whittaker, B. Ryner, and M. Nazif, "Large-scale automatic classification of phishing pages." in *NDSS*, vol. 10, 2010.

[13] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proceedings of the 2007 ACM workshop on Recurring malcode*. ACM, 2007, pp. 1–8.

[14] B. Eshete, A. Villafiorita, and K. Weldemariam, "Binspect: Holistic analysis and detection of malicious web pages," *Security and Privacy in Communication Networks*, pp. 149–166, 2013.

[15] L. Xu, Z. Zhan, S. Xu, and K. Ye, "An evasion and counter-evasion study in malicious websites detection," arXiv preprint arXiv:1408, 1993., Tech. Rep., 2014.

[16] F. J. Provost, T. Fawcett, and R. Kohavi, in *The case against accuracy estimation for comparing induction algorithms*. ICML (Vol. 98), July) 1998, pp. 445–453.

[17] S. Ruggieri, "Efficient c4. 5 [classification algorithm]," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 14.2, pp. 438–444, 2002.

[18] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, and et al., "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[19] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong, and C. Zhang, "An empirical analysis of phishing blacklists," in *Sixth Conference on Email and Anti-Spam (CEAS)*. California, USA, 2009.

[20] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious urls: An application of large-scale online learning." in *The International Conference on Machine Learning (ICML).*, 2009b, pp. 681–688.