9-1-2015

A Lexical Approach for Classifying Malicious URLs

Michael Darling

Michael Darling

Candidate

Electrical & Computer Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. Greg Heileman , Chairperson

Dr. Ramiro Jordan

Dr. Chris Lamb

A Lexical Approach for Classifying Malicious URLs

by

Michael Darling

Bachelor of Science, University of New Mexico 2012

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Engineering

The University of New Mexico

Albuquerque, New Mexico

July, 2015

Acknowledgments

I would like to thank my committee members, Professors Greg Heileman, Ramiro Jordan, and Chris Lamb. I would also like to thank Professor Praba Poornachandran for suggesting this topic and Aravind Ashok for his critical support and suggestions. Most of all I would like to thank Gilad Gressel for working on this project with me.

A Lexical Approach for Classifying Malicious URLs

by

Michael Darling

Bachelor of Science, University of New Mexico 2012

M.S., Computer Engineering, University of New Mexico, 2015

Abstract

Given the continuous growth of illicit 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. This paper describes a lightweight approach for classifying malicious web pages using URL lexical analysis alone. The goal is to explore the upper-bound of the classification accuracy of a purely lexical approach. Another aim is to develop an approach which could be used in a real-time system. These goal culminate in the development of 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 requires 0.62 milliseconds on average. This method substantially outperforms previously published algorithms on out-of-sample data.

Contents

List of Figures

List of Tables

Glossary

Uniform Resource Locator (URL) Specifies the location of a resource on a network.

Natural Language Processing (NLP) A field of computer science and computational linguistics which researches the interactions between computers and human languages.

Feature A quantified property of a phenomenon being observed.

Phishing Is a scheme designed to steal private user information.

Malware Also known as Malicious software, is designed to harm computer systems.

Lexical Is an adjective describing the relation to a vocabulary of words.

Classification is the problem of finding the category to which a data sample belongs based on a trained data model.

Classifier A program that implements a classification algorithm.

Classification Accuracy The number of samples that a classifier has correctly classified out of the total number of samples.

Hypertext Transfer Protocol (HTTP) An application protocol for various information systems.

IP Address, domain name, geo-location, and WHOIS is a numerical identifier for a device operating in a network.

Domain Name is a identifier string which defines a realm of control on a network.

WHOIS is a protocol that is used to query databases that store users of an Internet resource, such as a domain name, or block of IP Addresses.

False Positive (FP A sample that is actually benign that was classified as malicious.

True Positive (TP) A sample that is actually malicious that was classified as malicious.

False Negative (FN) A malicious sample that has been classified as benign.

True Negative (TN) A malicious sample that has been classified as malicious.

F-Measure

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

Where precision and recall are defined respectively as:

$$\frac{TP}{TP + FP} \text{ and } \frac{TP}{TP + FN}$$

Chapter 1

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, 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 [11]. Previous work has implemented several effective classification systems for malicious web page detection [2, 23].

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 trade-off 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 characteristics of malicious web pages differ based on the type of exploitation techniques used (e.g, phishing, drive-by-downloads, and infected adware networks). Therefore, the more comprehensive detection schemes take longer to execute as they have more to look for.

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 malicious 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 that users expect. In other words, the ongoing task is to quickly detect malicious web pages without sacrificing a high degree of accuracy [25].

The appeal of static tools are the speed at which they can classify compared to their dynamic counterparts. 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 easily scalable. However, classifiers built to analyze static features have limited success with highly sophisticated schemes such as obfuscated JavaScript based drive-by-downloads [17]. Obfuscation is a technique designed to elude static detectors. Static detection tools can be evaded when the meaning of known malicious strings are scrambled.

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 URL analysis alone can provide a means to detect malicious content [11]. With the right training a machine learning- based system can extend this skill far beyond the capabilities of the human eye. These systems can detect malicious characteristics that would be subtle to even the most scrupulous human user.

Previous studies have built static classifiers that detect malicious websites using a combination of URL lexical features, network traffic, hosting information, and other strategies [10, 20]. 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. Though it would be possible to remedy this latency by caching the results of these queries, any sort of comprehensive cache would be on the order of hundreds of gigabytes and preclude their use in a lightweight application such as in a browser extension.

Though previous work had utilized URL Lexical analysis as a component, what was lacking was the exploration of the full potential of a purely lexical approach. This potential could prove to be beneficial both as a component in a more comprehensive system or as a tolerable substitute in applications that demanded a lightweight and/or fast approach.

This paper presents an approach which uses an n-gram model to develop a new lightweight 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 [11, 10, 2, 23]. This 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, Java implementation of the C4.5 algorithm [6].

Chapter 2

Background

2.1 Introduction

Since the Internet was introduced to the general public, the security of computers on the network has been a concern. The first attack to gain widespread attention was the Morris worm unwittingly unleashed by a Cornell graduate student in 1988 [15]. This worm infected 1 in 20 computers while reducing their computing power to a fraction of normal capacity. Since then, the types of attacks have only grown in scope and maliciousness. Because of this growth, Cybersecurity has become an important field of research and the resultant anti-virus software has become a critical part of most computing environments.

As industries have moved many vital services on to the Internet, the breadth of the Internet as well as its number of subscribers have grown exponentially. Due to the naivet of many Internet users, malicious web pages have been an effective tool for pilfering money and private information. These web pages commonly contain two types of attack schemes: phishing and the distribution of malicious software.

The content of Phishing web pages usually mimics that of popular legitimate web pages. These web pages are usually visited by users who receive seemingly legitimate emails with links that obscure their fraudulent destinations. When users visit, for example, a website which imitates that of a bank's they will enter their usernames and passwords when prompted to do so. Thus the user surrenders the access to their bank account (and perhaps more since many people use the same usernames and passwords for many different services).

Websites that distribute malware will often display desirable content while downloading software to the user's system in the background. Users also unintentionally infect their computers by downloading software which they believe to be legitimate. There are many types of malware that perform various functions. A common malady is for the infected computer to become part of a botnet. Botnets are made up of an army of infected computers which are then able to further spread malware or a host of other insidious activities.

Due to the large number of malicious web pages infecting the Internet, there have been ongoing efforts to develop methods to identify those sites which should be avoided.

2.2 Previous Work

Historically, the most widely adopted method for identifying malicious web pages has been the construction of blacklists such as the Google Safebrowsing API. These blacklists are constructed by reporting web sites after they have already successfully executed their attacks any number of times. 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 to update. Numerous machines may potentially be infected in the meantime. Extracting machine learning features from blacklist features have also been shown to be less effective than features extracted by URL lexical analysis [11].

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 [11, 20, 23, 2]. Host-based feature sets include IP address, domain name, geo-location, and WHOIS properties.

Blum et al. [1] 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. [12] with the major difference being the absence of host based features.

Le, Markopoulou, and Faloutsos [10] 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. [25] 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 to extract the features and classify previously unseen URLs.

Thomas et al. [20], 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. [20] and Le, Markopoulou, and Faloutsos[10], 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.

Chapter 3

Approach

3.1 Overview

The general approach is explained first, followed by discussions on the features utilized, the data collected, and selection of the J48 algorithm for classification.

The goal of this study was to explore how quickly and accurately malicious web pages could be classified using only their associated URLs. This minimal approach leads to the fastest possible classification time since, unlike most other approaches, 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 lure users into clicking on links that lead to fraudulent web pages. Sites that contain malware (malicious software) seek to infect the user's system by downloading and executing harmful programs–users are often infected without realizing it.

Malicious content has been detected by analyzing static components of web pages as well as the dynamically executed content. The more efficient static methods often include the collection 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 [10], by the eliminating the collection of all features found on the page itself, the classification time is greatly reduced.

Furthermore, most previous work all relied on information which required lookups of 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. A possible solution to this problem could be to store the lookup information in a local database. However this local database would need to be on the order of 100 gigabytes and, therefore, preclude the use of the system in a lightweight application such as a browser extension.

As a baseline we implemented a bag-of-words model based on previous work without utilizing any time-consuming components. Our approach utilizes an n-gram language model which improved on the performance of the bag-of-words model when classifying out-of-sample data. While modeling URLs with an n-gram approach has been done [9], this method had not, to our knowledge, been been previously applied to malicious URL detection.

3.1.1 Modeling the Language of URLs

Standard Language Models

The goal of a language model is to assign a probability to any string of a particular language. A good language model will assign high probabilities to common and grammatical strings while assigning low probabilities to uncommon and ungrammatical strings. Building on this concept we attempted to construct a model of the *language* of *benign* URLs. A standard language model would calculate the probability of the occurrence of a given benign URL. 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).

N-gram Models

A common method when constructing language models is to parse N-sized grams of the dataset, where N is an integer. 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 proven this model to be effective in the task of language modeling and difficult to improve on [8].

Bag-Of-Words Model

As a baseline we followed previous studies [11, 1, 10, 20] by building a binary bag-of-words model which parses the URL into tokens which are unique in terms of their content and location. The bag-of-words is a set that contains every single unique token in the training data. 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, containing all zeros except for a few ones which represent the tokens of the given URL and any other standard lexical features utilized. On average our training data contained 122,000 binary features when using the bag-of-words model (the variability in the number of features was due to different data sets being used for each the tests).

3.1.2 Features

We implemented and expanded upon features based on the previous work of Ma et al.[11] and Whittaker, Ryner, and Nazif [21].

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.

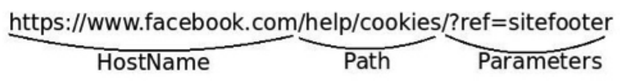


Figure 3.1: URL Components

URL Processing

Following Blum et al. [1], we separated the URL into three sections. Unlike Blum et al., we chose to separate the URL into hostname, path, and parameter components (see Figure 3.1). 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.[11], namely '/' , '?' , '.' , '=', '-' and '_'.

We separated the URL into component parts in order to preserve the context of their 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. So if the path contained an number of dots, it would be seen as unusual. We calculated nearly all features on all three sections of the URL as well as the entire URL. Though we implemented many of the features on the entire URL as well as the subsections we were careful not to introduce any feature redundancies.

In total we developed 87 features, which are categorized into five groups: n-grams, Lengths, Counts, Binaries, and Ratios.

Features Extracted from N-gram Models

In our analysis we implemented bigrams, trigrams and fourgrams. We also calculated a unigram probability which is context-independent. We created our n-gram models for the benign training data only. This was done based on the hypothesis that benign URLs are drawn from a smaller set than the URLs of malicious websites. Therefore there is a higher degree of variability in the characteristics of malicious URLs. Rather than having to capture all of this variability in the training set, the system would be better able to differentiate malicious URLs since their grams would not be characterized by the n-gram models. This hypothesis was confirmed by testing the system with inclusion of the malicious URLs in the models. These tests yielded lower classification accuracies.

To calculate any given n-gram, we divided 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 in the data set, the bigram, trigram, and fourgram models implement 1st, 2nd, and third order Markov chains.

In general, we define δ as an gram string of length L (where $L = 1$, specifies a unigram model, $L = 2$ specifies a bigram 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 bigram and the denominator is the unigram which makes up the first half of that bigram. 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.

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 4.2).

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 divided by total number of δ present in the URL, this normalizes the n-gram feature value by the length of the URL.

These features are *not* probabilistic representations of the URL. A probabilistic feature would would have summed all log-probabilities of δ_i^L. We tested this approach. However, we achieved higher results with summation of the probabilities of δ.

Our choice of probability addition yields the union of the probabilities for all δ_i:

$$Similarity(URL) = \frac{\rho_1 \cup \rho_2 \cup \ldots \cup \rho_m}{Length(URL)}$$

Summation, or what we will call *similarity*, resulted in higher classification accuracy (see Table 4.5).

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

Length

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 host, path, parameter and URL.

Counting Features

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

Pattern Features

Pattern features look for specific patterns within the URL and count occurrences of the pattern. Pattern counts were performed all 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. [5]. We implemented a total of 15 pattern features.

Binary Features

Binary features are the following: '.com' out of place, IP address for a hostname, similarity to alexa-top-10 domains, and the presence of an alexa top-10 domain out of place. We used the DamerauLevenshtein algorithm to check the edit distance in order to detect phishing attacks that use domain squatting (e.g. gooogle.com). We checked distances for 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.

Ratio Features

Ratio features calculate the ratio 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.

3.1.3 Data Sets

We began by collecting the same training data as Ma et al. [11]. We then sought out further sources of data in an attempt to build a more comprehensive training set. The training data was drawn from six web sites: Phishtank.org, Openphish.com, malwaredomainlist.org, malwaredomains.org, DMOZ.org, and Alexa.com. The training data set contains a total of 68,031 malicious and 122,550 benign URLs.

Malicious Data Collection

Phishing data was drawn from Phishtank.com and OpenPhish.com. Both web sites contain lists of phishing URLs. The Phishtank data is verified by human users. OpenPhish data is confirmed as phishing by autonomous algorithms. Since it was launched in June 2014, none of the literature that we studied had access to this data.

The URLs from websites dispensing malware were drawn from lists found on malwaredomainlist.com and malwaredomains.com. These block lists provide only domain names. Therefore we configured a web crawler to extract links from web pages contained in these inventories of Malware domains. From this set, we removed any URLs which pointed to domains outside the block lists since malicious pages may contain links to trusted domains.

Benign Data Collection

To collect the benign data set, we configured a crawler to collect links from the DMOZ.org directory and the Alexa.com top 1000 domains. DMOZ is a directory of the entire Internet which is manually managed. DMOZ links are posted by approved human editors. Alexa.com ranks websites based on traffic. As with previous studies [3, 17], 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.

Table 3.1: 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, qq.com, which is ranked in the Alexa top 1000, had been reported as distributing malware on one of the block lists (URLs from qq.com were found in and removed from both of our benign and malicious data sets). A breakdown of the whole dataset can be seen in Table 3.1.

Data Analysis

Table 3.2 shows that the average values for length and count attributes tend to be largest for phishing URLs, and smallest for malware URLs. Phishing and Benign URLs are designed to attract users: much thought is put into the domain and path tokens. Malware URLs, on the other hand, are often only seen after the user has been redirected from other pages: their utility is not in the attraction of user clicks.

URL Characterizations			
Attribute	**Benign**	**Malware**	**Phishing**
Length	64.06	58.94	101.97
Delimiter Count	6.13	5.68	8.30
Digit / Letter Ratio	0.13	0.15	0.21
Number of Dashes	1.40	1.13	1.16
Count of Blacklist Words	0.13	0.16	0.95
Number of Non-Alphanumeric Characters	47.27	43.05	83.75
Length of Parameter	15.19	10.89	21.96
Digit Count in Host	0.34	0.55	1.06
Non-Alphanumeric Count in HostName	13.14	14.14	20.64
Length of 1st directory	8.50	10.09	11.21
Vowel / Consonant Ratio in Hostname	0.45	0.39	0.55

Table 3.2: Examples of Average Feature Values

3.1.4 Classification Algorithms

In this study we compared the effectiveness of several classification methods with our data and feature sets.

Baseline

As a baseline for comparison 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 L1 regularized logistic regression model from the LibLinear package as was done by Ma et al.[11]. The methodology was to ensure that our system performed with higher accuracy than previous studies with our data set.

N-gram Based System

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[7]. We chose J48 due to its reputation of success in this domain [24] and after evaluating its performance in comparison to Bayesian Logistic Regression, Logistic Regression, Naive Bayes, and K-Nearest Neighbors classifiers.

Comparison of Algorithms

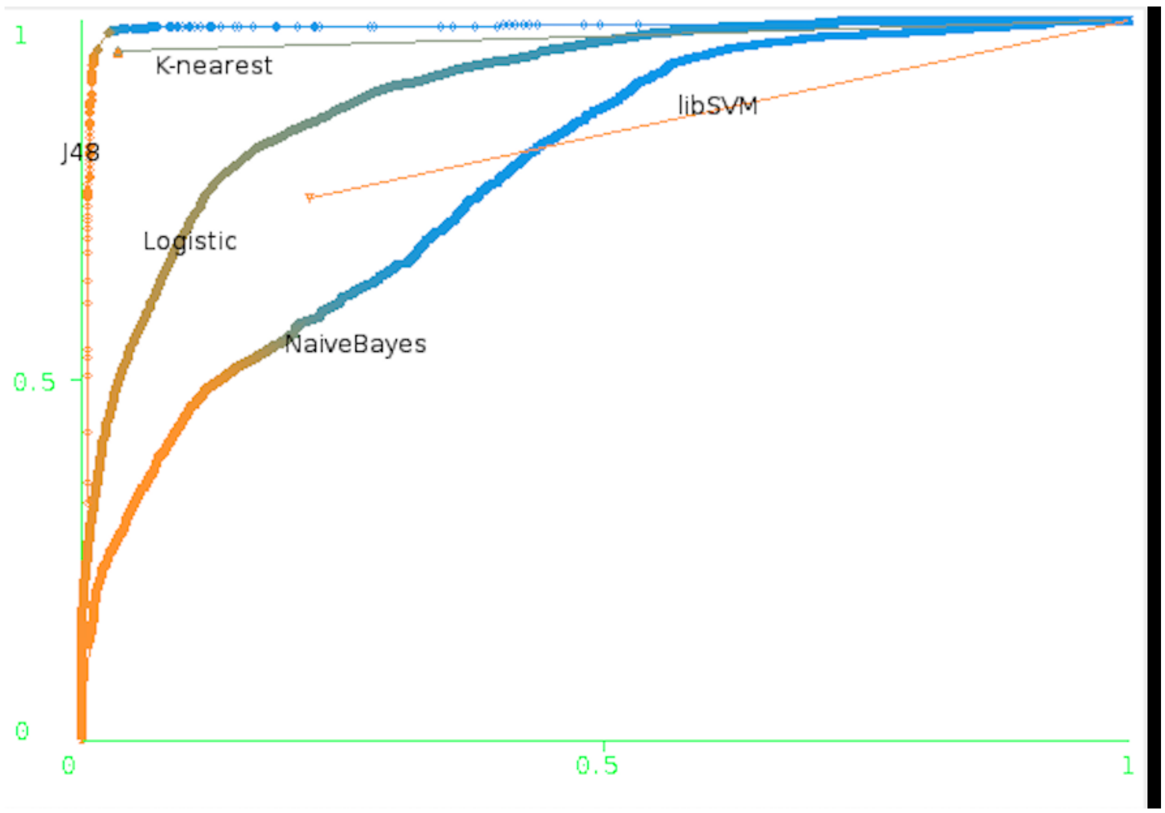
In order to evaluate performance of each algorithm, we compared their overall accuracy and receiver operating characteristic (ROC) curve.

A classifier's accuracy is not the only necessary metric to evaluate its performance.

The ambiguity lies in the fact that classification accuracy may not have equal misclassification costs: an accuracy of 98% 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.

Table 3.3: ROC Area Under Curve Values

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



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 3.3 shows J48 outperforming its counterparts in AUC of the ROC. Furthermore, the J48-based system achieved the highest overall classification accuracy for the n-gram model.

3.1.5 The J48 Algorithm

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. Both during and after the construction of the tree J48 implements pruning features in order to avoid overfitting the training data.

Derivation of J48

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 [18].

Choosing the information gain equation 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 [22]:

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

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

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

J48 Feature Selection

A feature will only be used if and when it is the choice which provides the highest information gain in determining the class of a data vector. This indicates the relevancy of a given feature: If a feature does not provide the best split of the data at any of the nodes in the tree, it is never used.

3.1.6 Effectiveness Metrics for Features

As each feature was implemented, 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.

F-Score Equation

The F-score is defined as:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

Where precision and recall are defined respectively as:

$$\frac{TP}{TP + FP} \text{ and } \frac{TP}{TP + FN}$$

Chapter 4

Results

4.1 Results Compared to Previous Work

We implemented a bag-of-words model with L1 logistic regression (LR) in order to have a baseline comparison with the most effective previous work. The overall classification results were quite similar as seen in Table 4.1. However Table 4.2 shows that the J48 model outperforms LR-based model when testing on out-of-sample data.

Table 4.1: Classifier Performance, 10-Fold Cross Validation on Training Set Size 131,520.

	Accuracy	F1-Score	FPR	FNR
J48	99.1	98.9	1.7%	0.4%
LR	99.1	98.7	1.7%	.5%

4.1.1 Similarities With Previous Work

Table 4.1 shows the results of both models are virtually identical 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.

4.1.2 Differences from Previous Work

The tests shown in Table 4.2 demonstrate that our n-gram model achieves with higher accuracy on out-of-sample malicious URLs then the previous work. The data in Table 4.2 is the resultant malicious True Positive Rate (TPR) when training on one source of malicious data set while testing on another. MDomains refers to data drawn from the source MalwareDomains.com, MDList is data drawn from MalwaredomainList.com, OpenPhish is drawn from Openphish.com and PhishTank from phishtank.com.

Table 4.2: Training with one malicious data set, while testing 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

4.1.3 Discussion of Out-of-sample Data Test

By training on one source of data and testing against the other, the classifier's performance on out-of-sample data is tested. J48 performs with higher accuracy in the majority of cases. LR has a lower TPR for a number of tests. Overall these tests show that the n-gram model with J48 generalizes to new data with higher accuracy than bag-of-words with LR. This test is atypical in that a classifier would not normally be expected to perform well on a data type that it has never seen before. However, since the J48-based system uses language models that are only based on benign websites, anything that does not "look" like a benign URL is likely to be classified as malicious.

The landscape of malicious URLs changes rapidly since attackers obtain and abandon domains rapidly to avoid detection. Whereas, the set of benign URLs changes very slowly in comparison since brand names are meticulously cultivated over time (e.g, google.com or facebook.com). Therefore, if the n-gram language models are comprehensively built to include the large majority of benign websites, this system will be well equipped to detect fly-by-night malicious operators. Since, as Table 4.2 shows, it performs well in classifying malicious data that looks nothing like anything it has ever encountered.

Table 4.3 shows the confusion matrix for the J48 model. In general we see that false positives and negatives were quite even.

Table 4.3: Confusion Matrix : P = predicted value, R = real value.

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

4.1.4 Feature Effectiveness

Table 4.4 shows the top 25 features ranked by information gain ratio for the entire training data. These scores are calculated for the entire training data which is the same calculation that J48 makes at the root of the tree: before it has made any splits of the data. 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 features that tank lower at the root, they provide higher information gain lower in the tree when the data sets are highly split.

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 4.1. The n-gram group has the highest accuracy of 98.23% which correlates with Table 4.4.

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 classification accuracy 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. This is why the classifier is highly effective in classifying malicious data that is unlike any that it has previous encountered (Table 4.2).

Table 4.5 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.

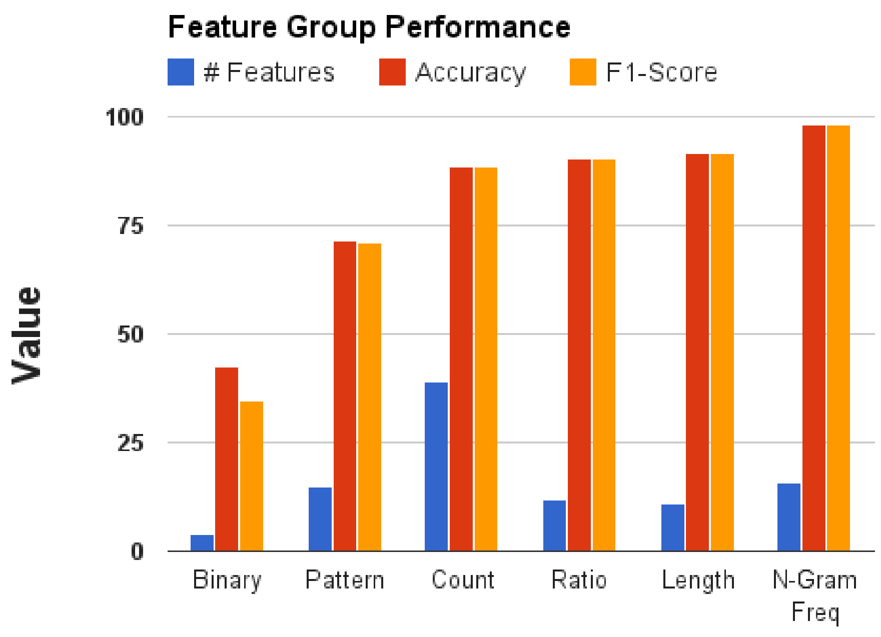


Figure 4.1: Performance of J48 Classifier When Trained and Tested on Individual Feature Categories.

4.1.5 Tuning

It is possible to tune the classifier to focus on 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 desire to tune the classifier to keep false positives low in order to reduce complaints about blocked sites. Otherwise, if it is desired to keep false negatives low in order to reduce the spam and viruses in the network, this could be done as well. Table 4.6 shows tuning results for the J48 classifier. As the number of malicious samples is increased the false negative rate is decreased.

4.1.6 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 the decision tree toward certain features.

In order to test against this we created a set of URLs where all domains were unique. This increased the variation between data samples. The results of this test are shown in Table 4.7. 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 120k), the accuracy of both classifiers would improve.

Table 4.4: 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

Table 4.5: 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

Table 4.6: 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%

Table 4.7: 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

Chapter 5

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 [24]. 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 features that detection algorithms use to detect maliciousness, it is possible to evade detection. However, there could be cost associated with manipulating these features depending on the intended use of a particular URL.

5.1 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 the 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.

5.2 Evading detection with malware URLs

Unlike phishing URLs, Malware URLs are not always designed to fool users. 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.

5.3 Evading Detection with Shortened URLs

URL shortening services are becoming increasingly popular both with attackers and the general public[14]. When a user clicks on a shortened URL, it redirects to the full length URL of the page. URL Lexical analysis designed to measure the features of a full length URL only. Currently, our system rejects shortened URLs.

5.4 Strength of n-gram Modeling

The landscape of malicious URLs changes rapidly since attackers obtain and abandon domains rapidly to avoid detection. Whereas, the set of benign URLs changes very slowly in comparison since brand names are meticulously cultivated over time (e.g, google.com or facebook.com). Many of the features in our system are based on n-gram models of 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.

Chapter 6

Conclusion and Future Work

6.1 Summary

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 small-scale security system such as a firewall or browser extension. To test production readiness we would generate network traffic and measure the performance of the system in handling a high volume of data.

Though Previous work had utilized URL Lexical analysis as a component, what was lacking was the exploration of the full potential of a purely lexical approach. This potential could prove to be beneficial both as a component in a more comprehensive system or as a tolerable substitute in applications that demanded a lightweight and/or fast approach.

6.2 Drawbacks

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 of our system that would follow the shortened URL and pull for classification the URL that it refers to.

6.3 Future Work

In the future we would also like to add a component to the system that performs detection of domain names that were created by Digitally Generated Algorithms (DGA) since a DGA domain name suggests the presence of malicious activity such as the presence of a botnet.

The characteristics of malicious web pages and URLs will continue to evolve. As with any security system, this one will need to keep up with the latest trends. To do this, there is need for the implementation 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 quickly compared to malicious websites, regular retraining will be necessary.

References

[1] 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*, pages 54–60, 2010.

[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*, pages 197–206, March 2011.

[3] Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam. Binspect: Holistic analysis and detection of malicious web pages. *Security and Privacy in Communication Networks*, pages 149–166, 2013.

[4] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[5] Sujata Garera, Niels Provos, Monica Chew, and Aviel D Rubin. A framework for detection and measurement of phishing attacks. In *Proceedings of the 2007 ACM workshop on Recurring malcode*, pages 1–8. ACM, 2007.

[6] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[7] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[8] Fred Jelinek. Up from trigrams. Eurospeech, 1991.

[9] Min-Yen Kan and Hoang Oanh Nguyen Thi. Fast webpage classification using url features. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 325–326. ACM, 2005.

[10] A. Le, A. Markopoulou, and M. Faloutsos. Phishdef: Url names say it all. In *Infocom*, pages 191–195, 2010.

[11] 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*, pages 1245–1254, 2009a.

[12] 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).*, pages 681–688, 2009b.

[13] D Kevin McGrath and Minaxi Gupta. Behind phishing: An examination of phisher modi operandi. *LEET*, 8:4, 2008.

[14] Alexander Neumann, Johannes Barnickel, and Ulrike Meyer. Security and privacy implications of url shortening services. In *Proceedings of the Workshop on Web 2.0 Security and Privacy*, 2010.

[15] Hilarie Orman. The morris worm: a fifteen-year perspective. *IEEE Security & Privacy*, 1(5):35–43, 2003.

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

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

[18] Salvatore Ruggieri. Efficient c4. 5 [classification algorithm]. *Knowledge and Data Engineering, IEEE Transactions on*, 14.2:438–444, 2002.

[19] Steve Sheng, Brad Wardman, Gary Warner, Lorrie Cranor, Jason Hong, and Chengshan Zhang. An empirical analysis of phishing blacklists. In *Sixth Conference on Email and Anti-Spam (CEAS).* California, USA, 2009.

[20] 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*, pages 447–462, 2011.

[21] Colin Whittaker, Brian Ryner, and Marria Nazif. Large-scale automatic classification of phishing pages. In *NDSS*, volume 10, 2010.

[22] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, and et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[23] 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)*, 14(2):21, 2011.

[24] L. Xu, Z. Zhan, S. Xu, and K. Ye. An evasion and counter-evasion study in malicious websites detection. Technical report, arXiv preprint arXiv:1408, 1993., 2014.

[25] Li Xu, Zhenxin Zhan, Shouhuai Xu, and Keying Ye. Cross-layer detection of malicious websites. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 141–152. ACM, 2013.

Appendix: Class Diagrams

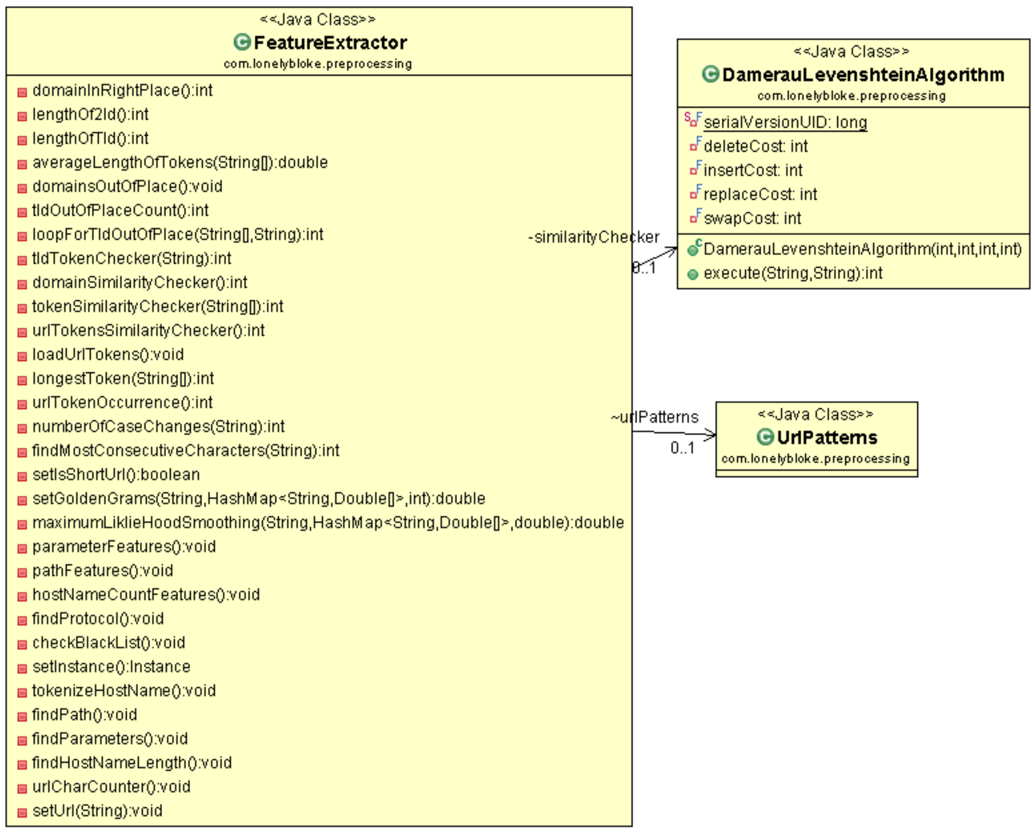

<<Java Class>>
FeatureExtractor
com.lonelybloke.preprocessing
domainInRightPlace():int
lengthOf2ld():int
lengthOfTld():int
averageLengthOfTokens(String[]):double
domainsOutOfPlace():void
tldOutOfPlaceCount():int
loopForTldOutOfPlace(String[],String):int
tldTokenChecker(String):int
domainSimilarityChecker():int
tokenSimilarityChecker(String[]):int
urlTokensSimilarityChecker():int
loadUrlTokens():void
longestToken(String[]):int
urlTokenOccurrence():int
numberOfCaseChanges(String):int
findMostConsecutiveCharacters(String):int
setIsShortUrl():boolean
setGoldenGrams(String,HashMap<String,Double[]>,int):double
maximumLiklieHoodSmoothing(String,HashMap<String,Double[]>,double):double
parameterFeatures():void
pathFeatures():void
hostNameCountFeatures():void
findProtocol():void
checkBlackList():void
setInstance():Instance
tokenizeHostName():void
findPath():void
findParameters():void
findHostNameLength():void
urlCharCounter():void
setUrl(String):void
-similarityChecker
0..1
<<Java Class>>
DamerauLevenshteinAlgorithm
com.lonelybloke.preprocessing
serialVersionUID: long
deleteCost: int
insertCost: int
replaceCost: int
swapCost: int
DamerauLevenshteinAlgorithm(int,int,int,int)
execute(String,String):int
~urlPatterns
0..1
<<Java Class>>
UrlPatterns
com.lonelybloke.preprocessing

Figure 6.1: Feature Extraction Classes

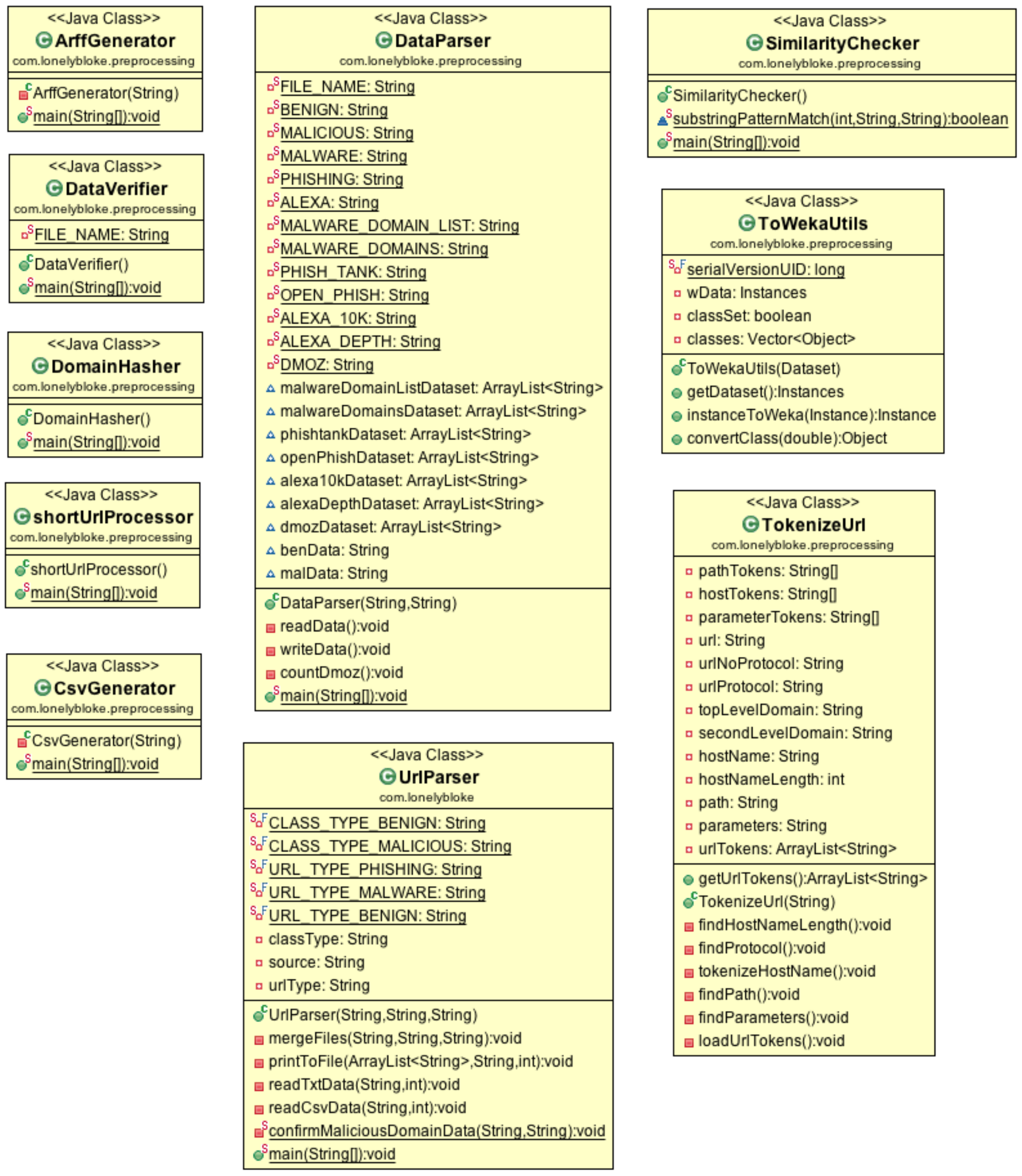
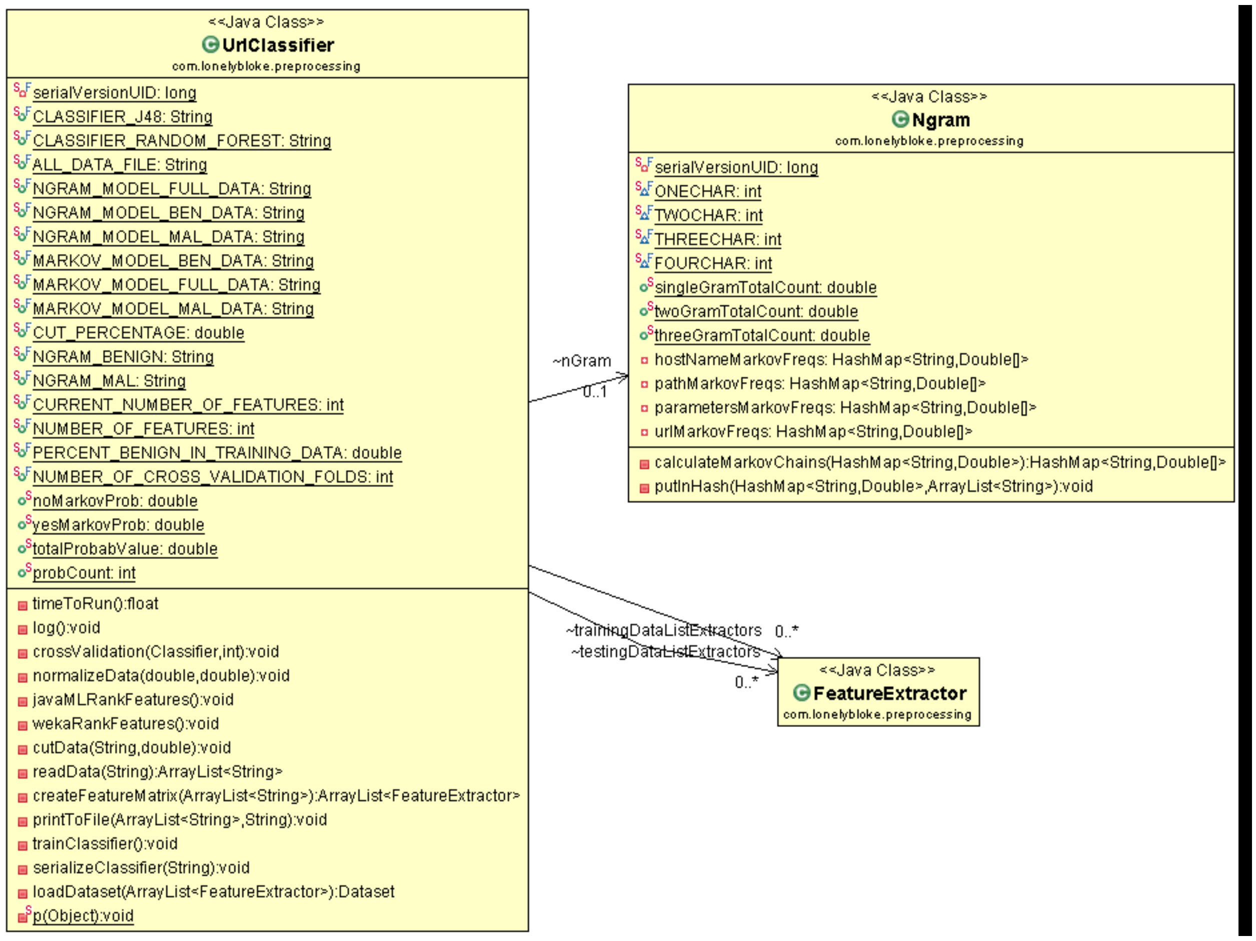


Figure 6.2: Data Processing Classes



<<Java Class>>
UrlClassifier
com.lonelybloke.preprocessing
serialVersionUID: long
CLASSIFIER_J48: String
CLASSIFIER_RANDOM_FOREST: String
ALL_DATA_FILE: String
NGRAM_MODEL_FULL_DATA: String
NGRAM_MODEL_BEN_DATA: String
NGRAM_MODEL_MAL_DATA: String
MARKOV_MODEL_BEN_DATA: String
MARKOV_MODEL_FULL_DATA: String
MARKOV_MODEL_MAL_DATA: String
CUT_PERCENTAGE: double
NGRAM_BENIGN: String
NGRAM_MAL: String
CURRENT_NUMBER_OF_FEATURES: int
NUMBER_OF_FEATURES: int
PERCENT_BENIGN_IN_TRAINING_DATA: double
NUMBER_OF_CROSS_VALIDATION_FOLDS: int
noMarkovProb: double
yesMarkovProb: double
totalProbabValue: double
probCount: int
timeToRun():float
log():void
crossValidation(Classifier,int):void
normalizeData(double,double):void
javaMLRankFeatures():void
wekaRankFeatures():void
cutData(String,double):void
readData(String):ArrayList<String>
createFeatureMatrix(ArrayList<String>):ArrayList<FeatureExtractor>
printToFile(ArrayList<String>,String):void
trainClassifier():void
serializeClassifier(String):void
loadDataset(ArrayList<FeatureExtractor>):Dataset
p(Object):void
~nGram
0..1
<<Java Class>>
Ngram
com.lonelybloke.preprocessing
serialVersionUID: long
ONECHAR: int
TWOCHAR: int
THREECHAR: int
FOURCHAR: int
singleGramTotalCount: double
twoGramTotalCount: double
threeGramTotalCount: double
hostNameMarkovFreqs: HashMap<String,Double[]>
pathMarkovFreqs: HashMap<String,Double[]>
parametersMarkovFreqs: HashMap<String,Double[]>
urlMarkovFreqs: HashMap<String,Double[]>
calculateMarkovChains(HashMap<String,Double>):HashMap<String,Double[]>
putInHash(HashMap<String,Double>,ArrayList<String>):void
~trainingDataListExtractors 0..*
~testingDataListExtractors
0..*
<<Java Class>>
FeatureExtractor
com.lonelybloke.preprocessing

Figure 6.3: Classification Classes

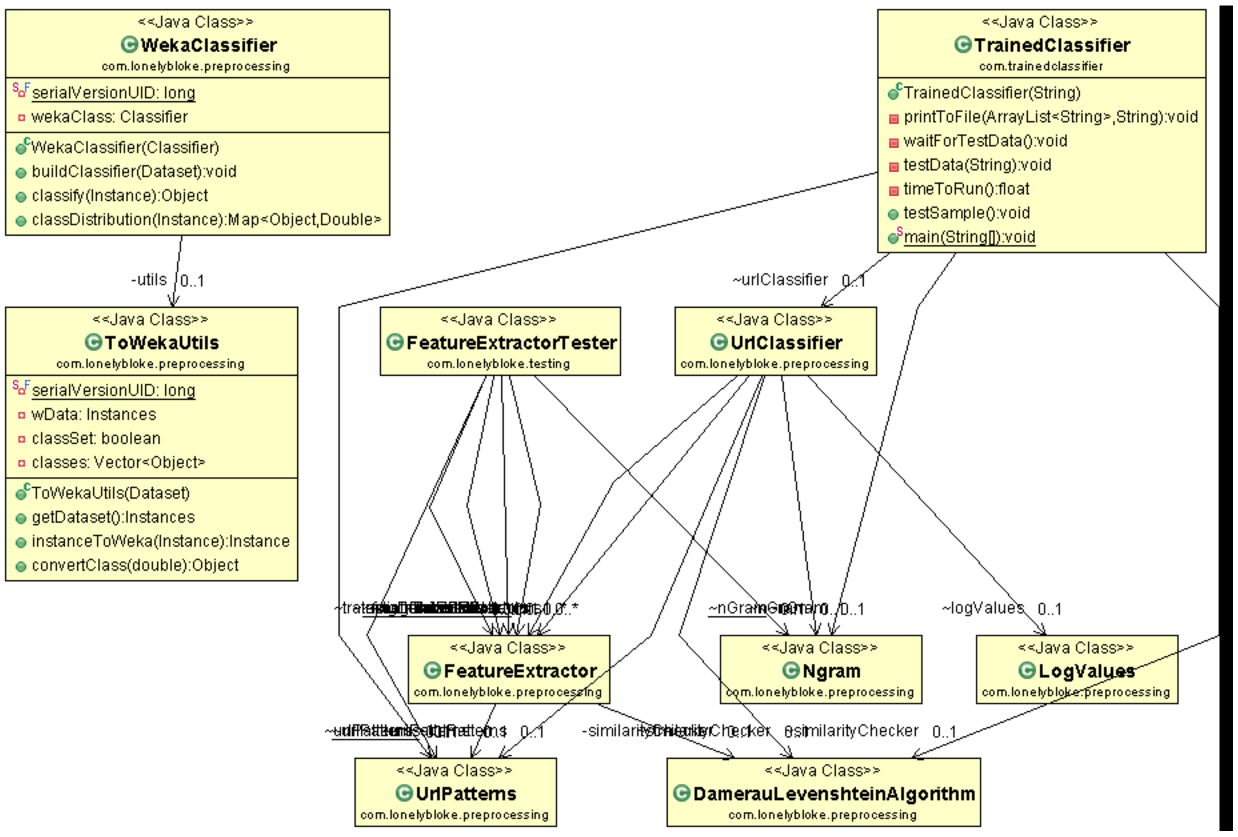


Figure 6.4: System Architecture