DETECTING MALICIOUS SHORTENED URLS USING MACHINE LEARNING

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Pujitha Vemuri

SPRING
2018

DETECTING MALICIOUS SHORTENED URLS USING MACHINE LEARNING

A Project

by

Pujitha Vemuri

Approved by:

_____, Committee Chair
Dr. Yuan Cheng

_____, Second Reader
Dr. Scott Gordon

_____
Date

Student: Pujitha Vemuri

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

_____, Graduate Coordinator _____
Dr. Jinsong Ouyang Date

Department of Computer Science

Abstract

of

DETECTING MALICIOUS SHORTENED URLS USING MACHINE LEARNING

by

Pujitha Vemuri

With advances in technology, there is a constant need of sharing resources in the form of social media, blogs, and through links to websites. To make sharing easier, URL shortening services like bit.ly, goog.gl, tinyurl.com, ow.ly are used, but often lead to unforeseen issues. The seemingly benign short URLs conceal malicious content. For example, a user who visits malicious website can become a victim of malicious activities such as phishing, spamming, social engineering, and drive-by-download.

This project aims to detect malicious shortened URLs with the help of machine learning techniques. Random Forest is one of the best classification algorithms that has a higher accuracy rate as it employs the use of higher number of trees, splitting points and the bagging concept. The model is trained with the shortened URL dataset, along with its features, thus achieving the accuracy of 96.29% for this project.

An extension for Chrome is developed to detect the shortened malicious URLs with the use of our generated machine learning model. While using the extension, if it encounters a malicious shortened URL, it informs user with details like normal form of

the URL, risk percentage (which depends on accuracy) and with the option of ‘still load the webpage’ or ‘go back’. This extension acts as a barrier between users and malicious websites, helping them educate choices to ensure their safety and privacy.

_____, Committee Chair
Dr. Yuan Cheng

_____
Date

DEDICATION

To My Parents, Friends & Well wishers

ACKNOWLEDGEMENTS

I would first like to thank my project advisor Dr. Yuan Cheng for helping me through every step with his knowledge and patience. Prof. Cheng is always available when there are any questions about my project or struck somewhere. He steered me in the right direction and allowed me to complete with confidence. His feedback helped me improve myself in many ways. I am greatly indebted for his valuable suggestions throughout my project.

I would also like to acknowledge Dr. Scott Gordon as my second reader for this project. He helped me when required and I am grateful for that.

I would also like to thank entire faculty and staff of the Department of Computer Science, CSUS for their guidance and resources.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figures Page

1. INTRODUCTION

Nowadays the Internet has become a platform to communicate and share information. Nevertheless, it is a dangerous place that supports a wide range of criminal activities. Technological advancements have given rise to advanced techniques that attack and scam users. Although motivations behind these criminal activities may differ, most of them require users to perform some action, such as clicking, which redirects to a specific webpage or downloads some content. Most of the attacking techniques are spread through sharing of compromised websites. Compromised URLs are websites that contain malicious content on a trusted website [1]. URL, also known as Uniform Resource Locator, is the universal address for a resource on the Internet. It is used as a vector that makes Internet users susceptible to cybercrimes. Popular attacks by using these compromised URLs include phishing, spamming, social engineering, and drive-by-download. These attacks are performed by exploiting vulnerabilities of websites and tend to cause billions of dollars' worth of damage every year. According to Ponemon Institute, it costs around $5 million on average when an attack succeeds [2]. Figure 1 shows how types of effects when a cyberattack is successful. This includes productivity loss, information theft, reputation damage, infrastructure damage, etc.

There are different techniques to make malicious URLs look legitimate. The most commonly used technique is obfuscation, which is classified into four types: obfuscation of host with IP, with another domain, with large hostnames, and misspelling [3]. A new method for obfuscating malicious URLs is through shortening services.

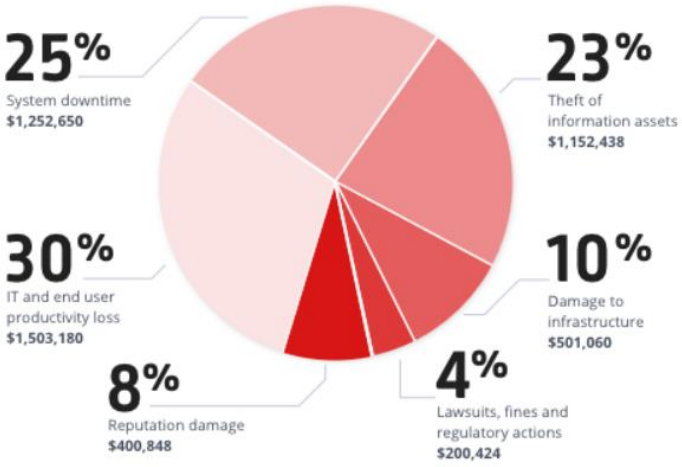

Figure 1: Cost of Endpoint Attacks [2]

URL shortening services have emerged in the recent years. There are hundreds of shortening services available for everyone, usually for free. The shortening services obfuscate the actual link that make it difficult for users to identify if the website is malicious or benign. To avoid this, many shortening services use blacklists to filter the malicious URLs even before shortening them. Few shortening service provides a preview or statistics of the webpage. For example, google and bitly shortening services use '+' symbol at the end of the short URL; consequently, tinyurl shortening service use 'preview' before the website name. For example, http://**preview.**tinyurl.com/abcXYZ.

Many users are becoming victims by not calculating the risk when clicking a suspicious URL. If there is any way to inform users beforehand that the website is

malicious, many people can be saved. This project aims to provide users with an effective solution by detecting those malicious URLs using machine learning approaches.

1.1 Report Organization

This project report is divided into nine chapters. Chapter 1 discusses the introduction of the malicious shortened URLs and the technologies used. Chapter 2 briefly specifies the work done by fellow researchers. Then Chapter 3 discusses the ways to detect malicious URLs while Chapter 4 gives information about various kinds of features that can be extracted to build a good predictive model. The classification algorithms in detail are explained in Chapter 5. Chapter 6 describes the actual implementation steps for a machine learning model. The extension service provided by the deduced algorithm is discussed in Chapter 7 along with its implementation. Finally, Chapter 8 concludes the report and Chapter 9 talks about the future work or scope of the predicted model.

1.2 Technologies Used

Python: Python is a high-level object-oriented programming language. Python can be easily learned and understood for beginners. It is a highly used scripting language. For this project, Python is used in feature extraction and machine learning model.

PyCharm: PyCharm is an Integrated Development Environment (IDE) mostly used for Python programming, along with HTML. It analyses, debugs, and tests the code line-by-line. PyCharm makes programming easier and simpler with use of libraries like NumPy,

SciPy, Pandas, Matplotlib, ggplot, etc. PyCharm is used for the machine learning part of the project.

JavaScript: JavaScript is a high-level interpreted programming language. Majority of the web browsers contains built-in JavaScript engine. It can be implemented in client-side web browsers, server-side web servers, databases, and runtime environment for mobile and desktop applications. Frameworks for JavaScript include AngularJS, Vue.js, Ember.js and some popular libraries are jQuery, React, p5.js. While building extension, JavaScript and HTML are used to interact with web browser.

Chrome API's: Google Chrome allows developers to make use of several available APIs. We used webRequest and Native Messaging API's for the Chrome extension of our model. The former is used to request details directly from website through parameters, while the latter is used for passing messages for extensions and applications.

2. RELATED WORK

Many researchers experimented on various techniques to detect several types of attacks from phishing emails, spamming, to drive-by exploits which are performed just by using a URL. Few worked on blacklists while others worked on machine learning models with help of many kinds of features. In this project, different approaches are combined along with short URL feature which is not used in other researchers work.

Michael [4] considered the use of lexical features alone for the classification of malicious webpages. Only the phishing and malware types are used to train the classifier with use of n-gram modelling. It is proven that the full potential of traditional lexical approach is enough to achieve higher accuracy.

Few researchers [5][6][7] even worked with instrumented Virtual Machines (VM) to protect the actual system from any client-side exploits that might occur by visiting a malicious website. Also, the instrumentation can detect if the machine is attacked.

Ma et al. [8] combined lexical and host-based features with modest false positive rate. However, some features that read the content of website are excluded due to safety purposes.

Naive Bayes approach is used by Sayamber et al. [9] to classify and further identify the malicious URL. They classified the malicious URL into spam, phishing, and

malicious. The model is designed to identify if a given URL is benign or the other three classes with the help of posterior value calculated for each class.

Lee et al. [10] proposed their Twitter suspicious URL detection ‘WarningBird’ as a real-time service for Twitter through redirections and achieved higher detection accuracy than Twitter’s detection system. However, WarningBird is unable to catch long URLs that appear after long intervals.

3. MALICIOUS URL DETECTION

This section describes the main points for detection of malicious URLs as mentioned by researchers and security practitioners. The key principles of detecting malicious URLs can broadly be classified into blacklisting, heuristic, and machine learning approaches.

3.1 Blacklisting Approach

It is most widely adopted approach by many search engines, browser toolbars, etc., to prevent users from accessing the flagged websites. Blacklist comprises a database of URLs that have been reported by users. These malicious URLs are identified only after successful execution of attacks. However, it is difficult to maintain such exhaustive list as new ones are added every second. They can't identify zero-hour phishing instances, and it is impossible to detect new threats as new URLs can be generated algorithmically, which helps them evade blacklists. Despite the limitations, blacklists are still used by many anti-virus software and web filtering applications [11].

3.2 Heuristic Approach

This approach can be defined as an extension to blacklist methods, where the idea is to create signatures for the malicious sites [1]. Signatures are assigned to web pages based on the behavior of common attacks that can be used by intrusion detection systems. The generalization capabilities for this approach are better compared to the previous one, since they can detect threats in new URLs by flagging the webpage if any suspicious

behavior is identified. However, this method is only determined for limited known attacks and generalizing the type of attack can become difficult at times. An advanced heuristic approach requires visiting webpage while searching for the signature, and thus user can be affected. For this drawback, this method is mostly implemented in virtual machines where the actual system cannot be affected.

3.3 Machine Learning Approach

Machine Learning approach has evolved recently to identify malicious URLs with less risk of exposing the user to attacks. This method analyzes URLs by extracting key features directly. Features can be defined as static and dynamic. Static features are extracted from URL without executing the content of the webpage like JavaScript. This is safer than dynamic approach since lexical, host-based information is extracted. The dynamic analysis concentrates on inspecting anomalies from the behavior of victim systems. For this project, we focus on static analysis which has a better impact on machine learning approaches, according to Sahoo et al. [1]. Machine learning model generates the rules to classify a URL as malicious or benign based on the data that is fed to the algorithm.

4. FEATURE SELECTION

To obtain a good predictive model, the quality of feature selection plays a crucial role, which further increases the quality of training data. All the relevant features of the URL are collected, and then we preprocess the unstructured data to use it for the selected model. There are several types of features that can be used to identify malicious URLs. These include blacklist features, lexical, host-based, content-based, network, and context-based features.

Information about many features can be obtained for a URL. Use of several types of features may provide a better model that can achieve higher accuracies, but it is not feasible. First, the time taken to extract features will be more. Second, we need to consider security risks while collecting some features like content-based and network-based. Finally, to make the generated model useful for people, features must be obtained from free sources unlike in the case of popularity features obtained from tools such as Alexa [12], which has an expensive fee to use.

Features that can be used for machine learning model should be considered by evaluating the associated risks, cost, time, and the need to depend on external dependencies. When using blacklist features, the time to collect and query can be high even though it can be used as the first step in eliminating already flagged websites. Content features require the user to first download the contents of the webpage, which might create problems if that website contains malicious code within it. Host-based

features are very time consuming to obtain. Lexical features have high dimensionality and are easy to extract directly from URL string.

For this project, lexical features with the combination of host-based features are used to obtain a better model to detect malicious URLs.

5. CLASSIFICATION ALGORITHMS

Since we need to classify a URL as either benign or malicious, we make use of classification algorithms like Logistic Regression, Naive Bayes Classifier, Support Vector Machine, Random Forest, and Neural Networks.

5.1 Decision Trees

Decision Trees can be used for both categorical and numerical/continuous instances. The result of categorical decision trees will either be {0/1} or {Yes/No} or {True/False}; however, numerical decision trees produce an actual predicted value. Decision trees evaluate data by creating trees, splits them by assigning parent and child nodes to the subtrees. These subtrees are the homogeneous groups of the entire dataset. Overfitting can be a disadvantage while using decision trees in some cases. This can be solved with the help of Random Forest classifier by limiting the features considered for each subtree. Pruning can also be used to avoid overfitting.

5.2 Naive Bayes

This classification algorithm is based on Bayes Theorem from Bayesian statistics with naive independent assumptions, which assumes that all the features are self-determined with respect to each other. Figure 2 shows the formulae for Bayes Theorem, in general. The Naive Bayes classifier calculates the probability by treating the features independently, even if they depend on each other or upon the existence of other features. For example, the length of the URL needs to be calculated directly from URL, but this

classifier treats them independently. Each feature will have an independent contribution to the probability to label the data.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Figure 2: Bayes Theorem [13]

This algorithm is mostly used in text classifications and for complicated predictions with high-dimensional data. Naive Bayes probabilistic classifier generates rules for the model based on assumptions from initial knowledge and previous assumptions. It uses a similar attribute to predict the probability of different classes.

5.3 Random Forest

Random Forests is based on the concept of Decision Trees and can be used for both classification and regression purposes of a discrete or continuous variable. This algorithm can be used as feature selection when there are a lot of features with the need to reduce them. Figure 3 shows the basic representation of Random Forest. It is a collection of decision trees and each decision tree is a collection of trees. Each tree's prediction is calculated, and a mean average of those predictions is performed at the end.

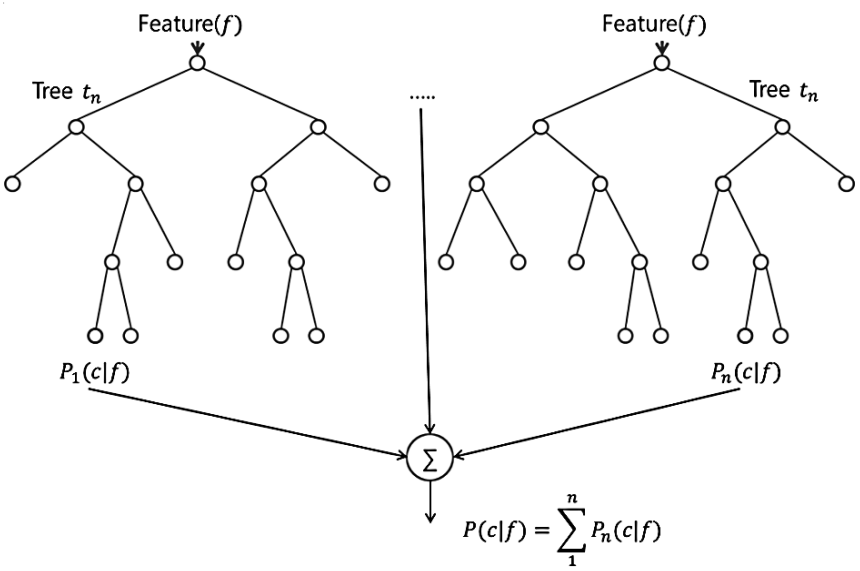

Figure 3: Random Forest Tree [14]

Random Forest algorithm avoids overfitting when an enormous number of trees are considered. Random Forest can handle missing values by itself which is the main advantage over other algorithms. The concept of bagging is used in this algorithm to avoid overfitting, which is a common and significant issue in machine learning algorithms. Multiple decision trees are created with different samples of training dataset and their predictions are combined. This way of building a model is called as bootstrap aggregation or simply bagging. Since the observations for each decision trees are selected randomly and without replacement, overfitting is not an issue for Random Forest classifier. Greedy algorithm is used to create each tree and the split points are evaluated with the help of either Gini Index or residual sum of squares. The split points are chosen by calculating the values of attributes. Random Forest evaluates each decision tree, which in turn consists of individual trees and calculates accuracy. The accuracy of the trees can

be estimated using cross-validation. The final accuracy of the random forest will be the mean of accuracies generated by all the decision trees.

The accuracy of the model can be increased by few parameters and can be simplified for training. It is better to use max_features as the model will have more options to consider. But, too many features can decrease the speed of the algorithm. The number of trees will be directly proportional to the accuracy of the model until the maximum number of trees reaches its threshold. This can be found by calculating the accuracy with the different number of trees. Once the accuracy reaches maximum, it remains the same even with an increase in the number of trees.

6. IMPLEMENTATION OF MACHINE LEARNING MODEL

Machine learning can be categorized as supervised, unsupervised, and semi-supervised: data that is identified and labeled by someone, data that is not yet labeled, and a part of data identified and labeled. Labels in this project determine a URL as benign or malicious. Algorithms such as Logistic Regression, Decision Tree, Random Forest falls under the category of supervised algorithms, whereas k-means come under unsupervised which uses clustering in diverse groups and Markov Decision process for semi-supervised learning.

6.1 Data Collection

This section provides information about the sources of labeled URLs. A large dataset of about 4.5M URLs is collected from various sources of the Internet such as PishTank, OpenPish, BlockList, Yahoo's directory listing, and from other Machine Learning researchers [15], [16]. Data is required to be collected along with its label for URL identification as either benign or malicious. The collected data for this project is labeled as benign and malicious and the observations for each is showed in Figure 4.

```
0.0    342363
1.0     75456
Name: malicious, dtype: int64
```

Figure 4: Data Classification Types

However, we can only collect the URLs, not the features. Thus, we develop a program to extract features from URL itself. Since this project is focused on lexical features, the short URLs must be converted to its original form as the lexical analysis is designed to extract features only from full-length URLs. This task is performed with the help of URL expander service ‘urlex.org’. The retrieved original form of the URL can be used for rest of the project. The next step is to extract informative features which can describe the URL and can also be interpreted mathematically by machine learning models.

6.2 Feature Extraction

This project mainly concentrates on lexical features for their simplicity, cost-effectiveness, and easy accessibility. Lexical features enable the model to capture the properties of URL for classification purposes. All the features are named in simple terms for better understanding. From the URL, we extract key features such as host, path, tokens, executable files in URL, delimiter count, number of dots, and length of URL. Now some of these features are used to further extract other features. The extracted first level features are further used in the extraction process of second level features. We also get the count of delimiters, dots, and security susceptible words. These are the delicate words that are used in most of the malicious URLs to make them look trustworthy.

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 417820 entries, 0 to 417819
Data columns (total 22 columns)
Short_URL                  417820 non-null object
URL                        417820 non-null object
Host                       417820 non-null object
Path                       417718 non-null object
Length_of_URL              417820 non-null int64
Length_of_Host             417820 non-null int64
No_of_Dots                 417820 non-null int64
Deli_Cnt                   417820 non-null int64
Avg_Token_Len              417820 non-null float64
Token_Cnt                  417820 non-null int64
Largest_Token              417820 non-null int64
Avg_Domain_Token_Len       417820 non-null float64
Domain_Token_cnt           417820 non-null int64
Largest_Domain             417820 non-null int64
Avg_Path_Token             417820 non-null float64
Path_Token_Cnt             417820 non-null int64
Largest_Path               417820 non-null int64
Sec_Sen_Word_Cnt           417820 non-null int64
IPaddress_Presence         417820 non-null int64
Exe_in_URL                 417820 non-null int64
ASNno                      417820 non-null int64
Malicious                  417820 non-null int64
dtypes: float64(3), int64(15), object(4)
memory usage: 1.7+ MB
```

Figure 5: Extracted Features

Figure 5 provides all the features extracted in this phase and information about those attributes

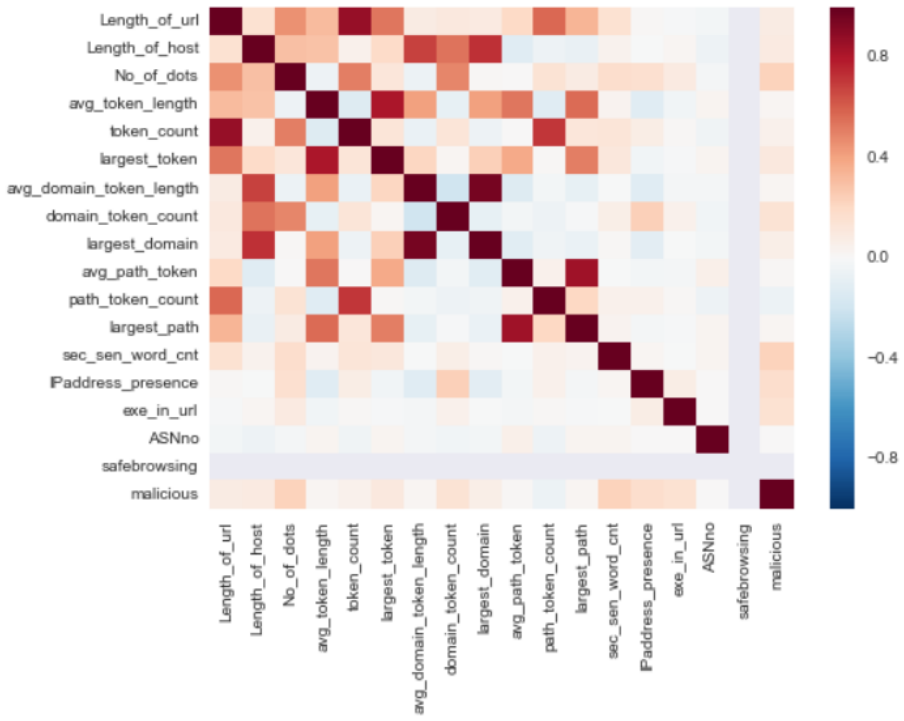

Figure 6: Correlation Matrix for Obtained Features

Correlation matrix for the features is showed in Figure 6, is used to understand the relation between target variable (malicious) and its predictors (features). The matrix is created with the help of seaborn library. We can determine the features that may be important in the detection of malicious URLs from the matrix with high correlation. Features with less correlation can be removed from the feature set as they just complicate the model. Topmost features identified are the length of the URL, number of dots, length of the host, executable files in URL, and security sensitive words.

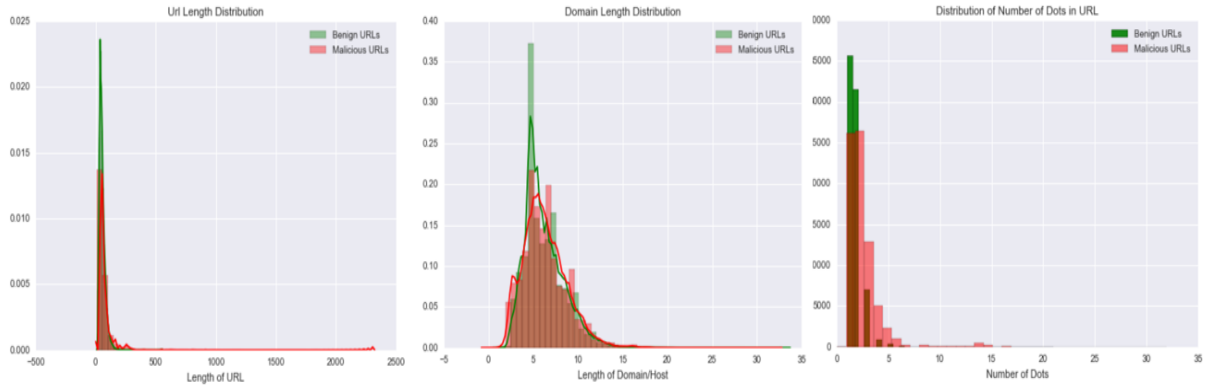

Figure 7: Graphical Representation of Length of URL, Host, and Number of Dots

Graphical representation for some features are created with help of matplot and seaborn library, can be seen in Figure 7. The graphs show that malicious URLs contain a very high number of dots, longer URL and domain lengths.

6.3 Data Preprocessing

Once the features are extracted, first, we need to preprocess the data. Preprocessing involves handling missing values and data formatting, which can be referred to ‘data cleaning’. Each machine learning model needs data in a specific format. For example, a few algorithms cannot handle null values and that needs to be managed from the original unformatted dataset. This is the crucial step for machine learning models as the results depend mostly on this step. The performance of the model can be improved by reducing the noise of data.

Another aspect is that dataset should be formatted in such a way that more than one machine learning algorithm can be executed with the same dataset, and best out of them is chosen.

6.4 Model Selection

To determine which model is best, the dataset is trained on few selected algorithms: Naive Bayes, Random Forest, and Decision Trees. An algorithm is chosen based on its accuracy, training time, parameters and features. This can be done by comparing the results generated by different classification algorithms.

```
Implementing Naive Bayes
-------------------------------------
Training time: 171.28 s
Predicting time: 0.048 s
Average Accuracy: 83.394061238 %
Classification Report:
         fscore  precision    recall
class
0.0    0.891756   0.842004  0.947757
1.0    0.267786   0.446084  0.191317
```

Figure 8: Results Obtained for Naive Bayes Classifier

Naive Bayes is selected to study if there is any major difference when compared to regular model like decision tree. Figure 8 shows the obtained results for Naïve Bayes classification.

```
Implementing Decision Trees
-------------------------------------
Training time: 196.89 s
Predicting time: 0.09 s
Average Accuracy: 89.2167170871 %
Classification Report:
         fscore  precision    recall
class
0.0    0.928850   0.886279  0.975718
1.0    0.558941   0.795943  0.430696
```

Figure 9: Results Obtained for Decision Tree

Decision Tree algorithm is chosen to compare with Random Forest to test the implementation of bagging concept in Random Forest. As we can see in Figure 9 and Figure 10, the accuracy obtained by decision tree is comparatively less than the accuracy of Random Forest, thus proving Random Forest is better for this project.

```
Implementing Random Forest
-------------------------------------
Training time: 291.979sec
Predicting time: 0.16sec
Average Accuracy: 93.7011735516%
Classification Report:
         fscore  precision    recall
class
0.0    0.961801   0.948919  0.975036
1.0    0.812152   0.870245  0.761329
```

Figure 10: Results Obtained for Random Forest Classifier

Among these, Random Forest algorithm is selected with its highest accuracy of 93.70%. Now for the existing algorithm, a few modifications like considering a diverse

set of features, choosing different split points, and different evaluation methods are made to improve the accuracy of the model.

6.5 Detailed Explanation of Random Forest Algorithm

A simple flowchart by Hawamdah [17] to show how Random Forest works internally is shown in Figure 11. Basically, we start with choosing training data subset for each tree and computing Gini index at each split point to choose the best split and predict for each tree.

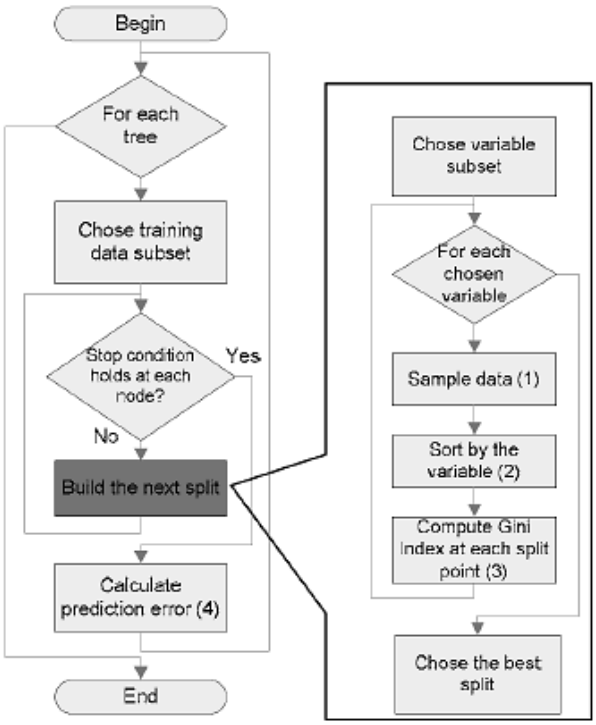

Figure 11: A Simple Flow Chart of Random Forest Algorithm [17]

First, the dataset is divided into k-folds where the number of features for each fold is considered as max_features and the observations are selected from dataset without

replacement. Deep trees are built with a specified maximum depth till the minimum size of '1' is obtained for each subtree. Trees are constructed by calculating the best split point with the lowest cost. Observations for each decision tree are chosen randomly and without replacement so that each variable is only considered once for each decision tree. The function getsplit() shown in Figure 12 is used to collect the required features and observations for each tree.

```
def get_split(dataset, n_features):
        class_values = list(set(row[-1] for row in dataset))
        b_index, b_value, b_score, b_groups = 999, 999, 999, None
        features = list()
        while len(features) < n_features:
                index = randrange(len(dataset[0])-1)
                if index not in features:
                        features.append(index)
        for index in features:
                for row in dataset:
                        groups = test_split(index, row[index], dataset)
                        gini = gini_index(groups, class_values)
                        if gini < b_score:
```

Figure 12: Random Forest - getsplit() Function

Now Gini index is then used to evaluate each split point of the tree in the forest. It calculates the purity of the subtrees at split point. The result value should be in between 0 and 1. Figure 13 shows how the Gini index is calculated for each group.

```
def gini_index(groups, classes):
        # count all samples at split point
        n_instances = float(sum([len(group) for group in groups]))
        # sum weighted Gini index for each group
        gini = 0.0
        for group in groups:
                size = float(len(group))
                # avoid divide by zero
                if size == 0:
                        continue
                score = 0.0
                # score the group based on the score for each class
                for class_val in classes:
                        p = [row[-1] for row in group].count(class_val)
                        score += p * p
                # weight the group score by its relative size
                gini += (1.0 - score) * (size / n_instances)
        return gini
```

Figure 13: Random Forest – Calculating Gini Index

Performance of the model is calculated with k-fold cross validation as shown in Figure 14. The whole dataset is divided into k folds and variables are assigned to each fold. Once the model is trained, predicted results are evaluated with actual ones as shown in Figure 15.

```
def cross_validation_split(dataset, n_folds):
        dataset_split = list()
        dataset_copy = list(dataset)
        fold_size = int(len(dataset) / n_folds)
        for i in range(n_folds):
                fold = list()
                while len(fold) < fold_size:
                        index = randrange(len(dataset_copy))
                        fold.append(dataset_copy.pop(index))
                dataset_split.append(fold)
        return dataset_split
```

Figure 14: Random Forest - Calculating k-fold Cross Validation

```
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
        folds = cross_validation_split(dataset, n_folds)
        scores = list()
        for fold in folds:
                train_set = list(folds)
                train_set.remove(fold)
                train_set = sum(train_set, [])
                test_set = list()
                for row in fold:
                        row_copy = list(row)
                        test_set.append(row_copy)
                        row_copy[-1] = None
                predicted = algorithm(train_set, test_set, *args)
                actual = [row[-1] for row in fold]
                accuracy = accuracy_metric(actual, predicted)
                scores.append(accuracy)
        return scores
```

Figure 15: Random Forest - Evaluation

In this model, after testing a different number of trees we considered 20 as the threshold and the number of folds equals to 5. The entire dataset, i.e., 417820 observations, are divided into 83564 observations for each decision tree randomly using the best split function. Gini function then checks the purity of all 5 folds. Finally, the model predicts the URLs with an accuracy of 92-98%. Figure 16 shows the average or mean accuracy as 96.29% of the modified model.

```
Random Forest Algorithm
trees: 20
Max_depth: 10
folds: 5
Scores: [92.48143, 95.74181, 97.21713, 97.78151, 98.23962]
Mean Accuracy: 96.29230
```

Figure 16: Random Forest – Final Result

7. MALICIOUS URL DETECTION AS AN EXTENSION

Detection of shortened malicious URLs with the help of machine learning approach can be used to build a real-world malicious URL detection system for both short and regular URLs. For this, we propose an extension for Google Chrome that uses the proposed machine learning model to detect and block the malicious URLs in real-time. Some researchers like Lee et al. [10] and Alshboul et al. [18] focused on providing services to online social networks like Twitter.

The extension for Chrome can be divided into 3 steps: 1) Preparing extension to block URLs, 2) Integration of machine learning model in Python to Chrome, and 3) Connection of the modified script to Chrome. Chrome extension only allows HTML, CSS, and JavaScript files [19]. Due to this, we are required to update and integrate the developed Python scripts for feature extraction and machine learning model.

The extension of Chrome starts with preparing manifest.json file that basically contains all the details required for that particular extension like name, description, permissions required, manifest version, and browser actions, etc. Whenever the extension encounters a URL, it needs to block the URL first before redirecting to the site. Helper functions are used to send and read messages with the help of JSON. Code for send() and read() is displayed in Figure 17.

```
def send_message(message):
  message= message.encode('utf-8')
  sys.stdout.buffer.write(struct.pack('@I', len(message)))
  sys.stdout.buffer.write(message)
  sys.stdout.buffer.flush()

def read_message():
    # Reads the first 4 bytes of the message (which designates message length).
    text_length_bytes = sys.stdin.buffer.read(4)
    # Unpacks the first 4 bytes that are the message length. [0] required becaus
    text_length = struct.unpack("i", text_length_bytes)[0]
    # Reads and decodes the text (which is JSON) of the message.
    text_decoded = sys.stdin.buffer.read(text_length).decode("utf-8")
    # Converts the message string into a dictionary.
    text_dict = json.loads(text_decoded)
    # Returns the dictionary.
    return text_dict['link']
```

Figure 17: Extension - send() and read() Messages

This can be done using Chrome webRequest API which runs in the background. Source can be obtained from https://developer.chrome.com/extensions/webRequest. It may take a few seconds to extract the required features from the URL and process them through the model to determine if that site is benign or malicious. When it comes to short URL, it will take a few extra seconds to first redirect it to its actual website and then continue the process. The user is redirected to background.js where the actual machine learning process takes place, then to urlCheck.html where the obtained output is displayed, and then redirected to the actual website if it is benign.

The trained machine learning model is saved as a pickle file to use for this step and included in background.js to test the new URLs. This program will take *url* parameters necessary for Python script via http get Parameters function. The ‘urlCheck’ page will be loaded when this program runs in the background. After getting the result,

the ‘urlCheck’ page is reloaded as ‘site is malicious’ along with actual URL, risk percentage and the option to ‘still continue to site’ or ‘go back’ if the site is determined as malicious. If the site is benign, it just informs that the site is safe along with the actual URL and automatically redirects to the destination page.

Already tested websites are maintained as a list, so that there is no need to double check if the extension come across the same website. The malicious percentage is also shown on the extension icon/badge in omnibar for every webpage. The icon is designed with the help of simple CSS transition in such a way that it turns to green color if the site is benign and red when it is malicious. The icon is also programmed with ‘onClick’ function to rescan the webpage when a user clicks on it. For this, the website along with the malicious percentage is stored as a Java object in localStorage.settings.urls which is the fastest lightweight way to store data in Chrome. Unlimited storage permission is added in the manifest file to store all the data. Modified Python code is integrated with the help of Chrome Native Messaging API [20] which is a well-implemented technology. Extensions and applications will be able to exchange messages with the native messaging application using this API.

7.1 Extension Screenshots

This section demonstrates how the developed extension works with necessary screenshots to prove. Figure 18 shows the description, version, memory needed, and permissions required for the extension “Malicious URLs Detection” after it is loaded to the Chrome browser by selecting developer mode in Chrome settings.

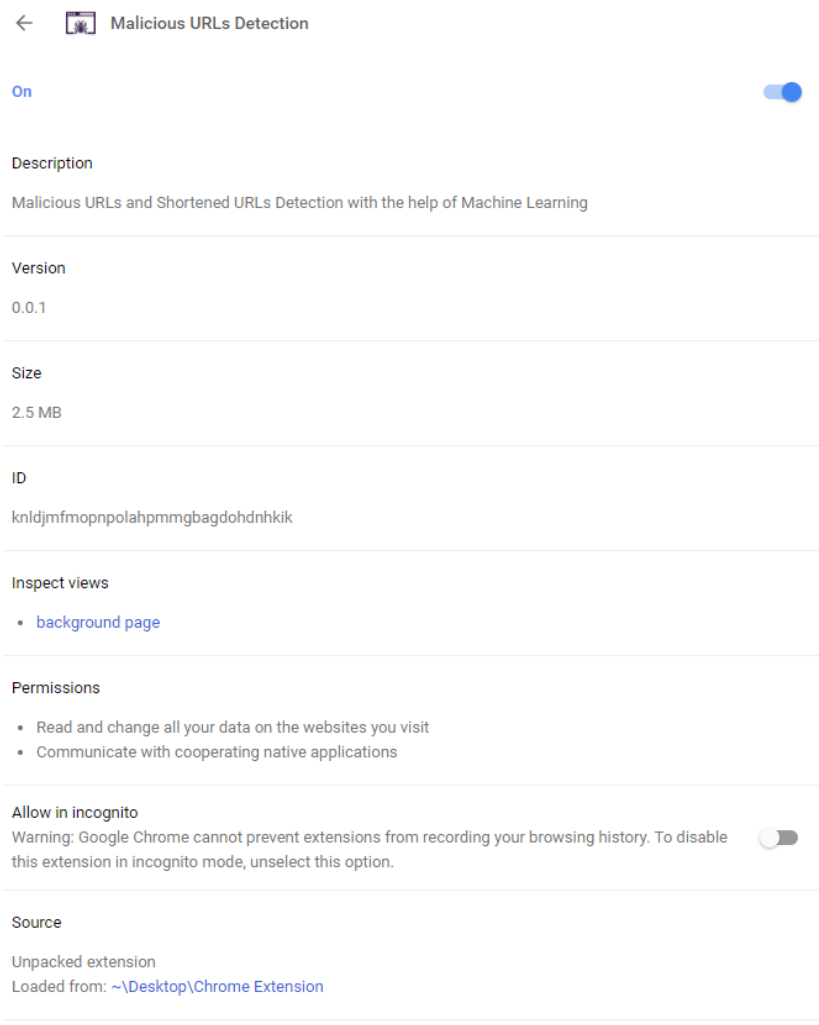

Figure 18: Loaded Chrome Extension with Required Information

First, we check the extension on a benign csus website “http://library.csus.edu” which can be seen in Figure 19. The extension is programmed in such a way that it will directly redirect to the website if it is safe.

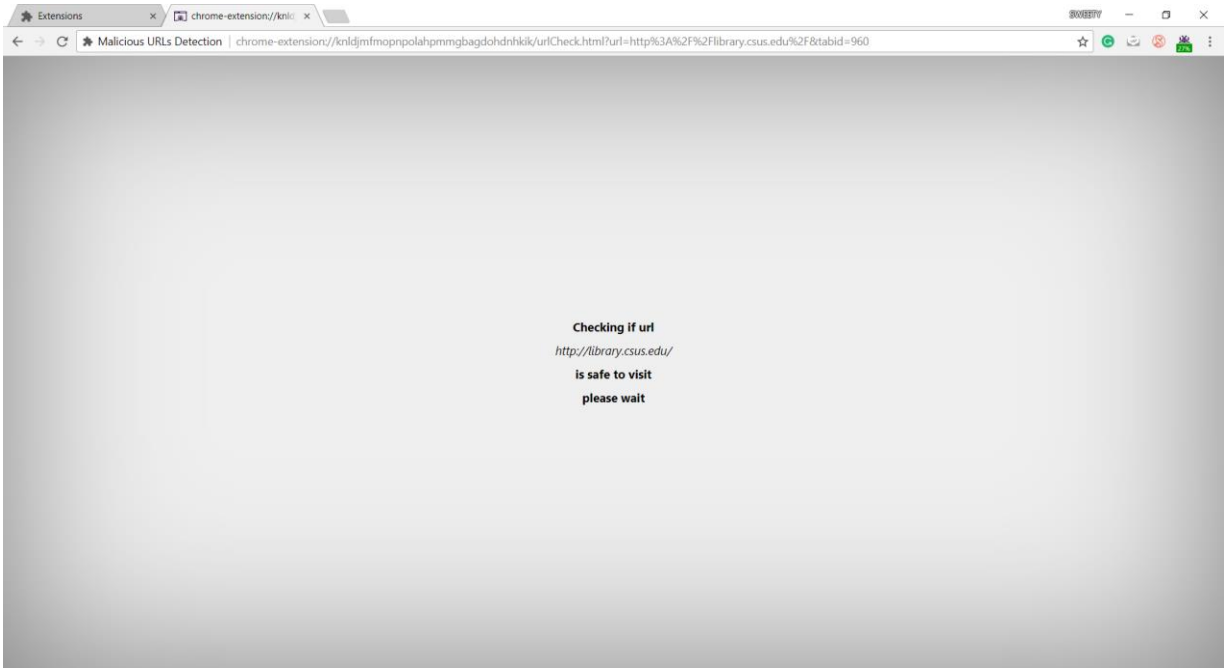

Figure 19: Extension Testing on Benign Website

The redirected site can be seen in Figure 20, along with the risk percentage calculated by machine learning model on top of the page. Here the risk percentage is 20.

Then we check the extension for a known malicious website “http://www.icilarache.com” that is collected from the blacklist. This can be seen in Figure 21. Extension calculates that the given website is 83% malicious. When observed Figure 22, we can see that the extension is giving two options: to accept the risk and continue or be safe.

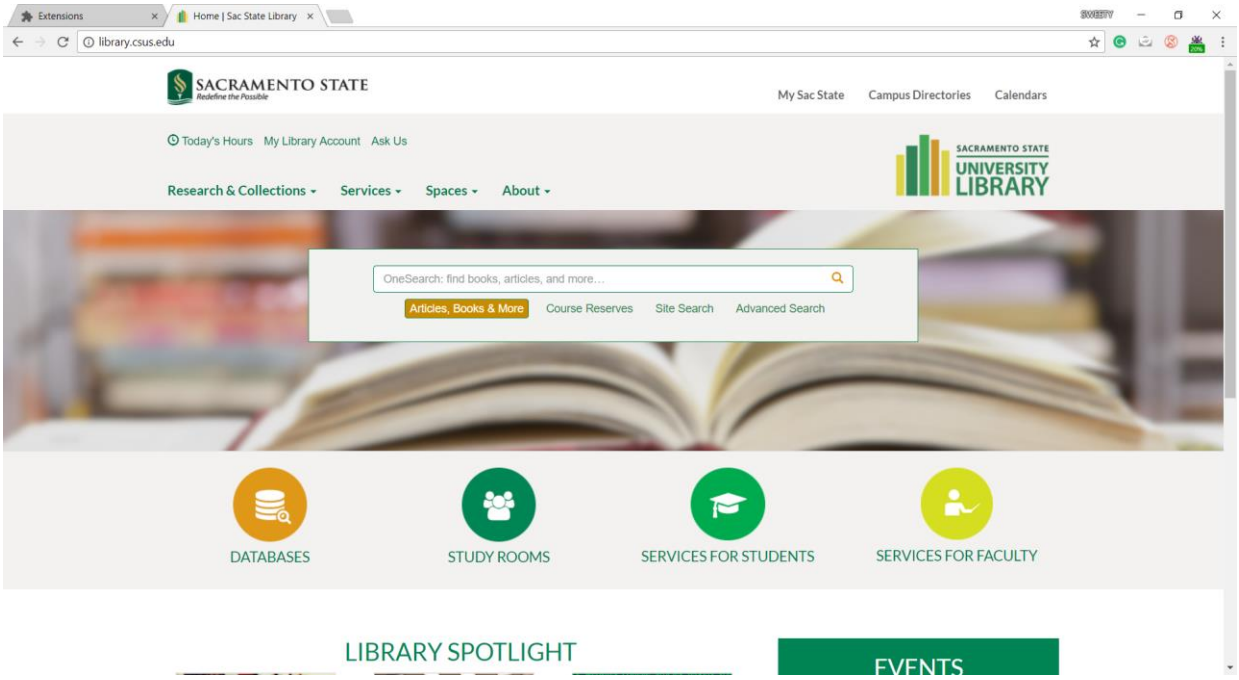

Figure 20: Extension Determined URL is to be Safe

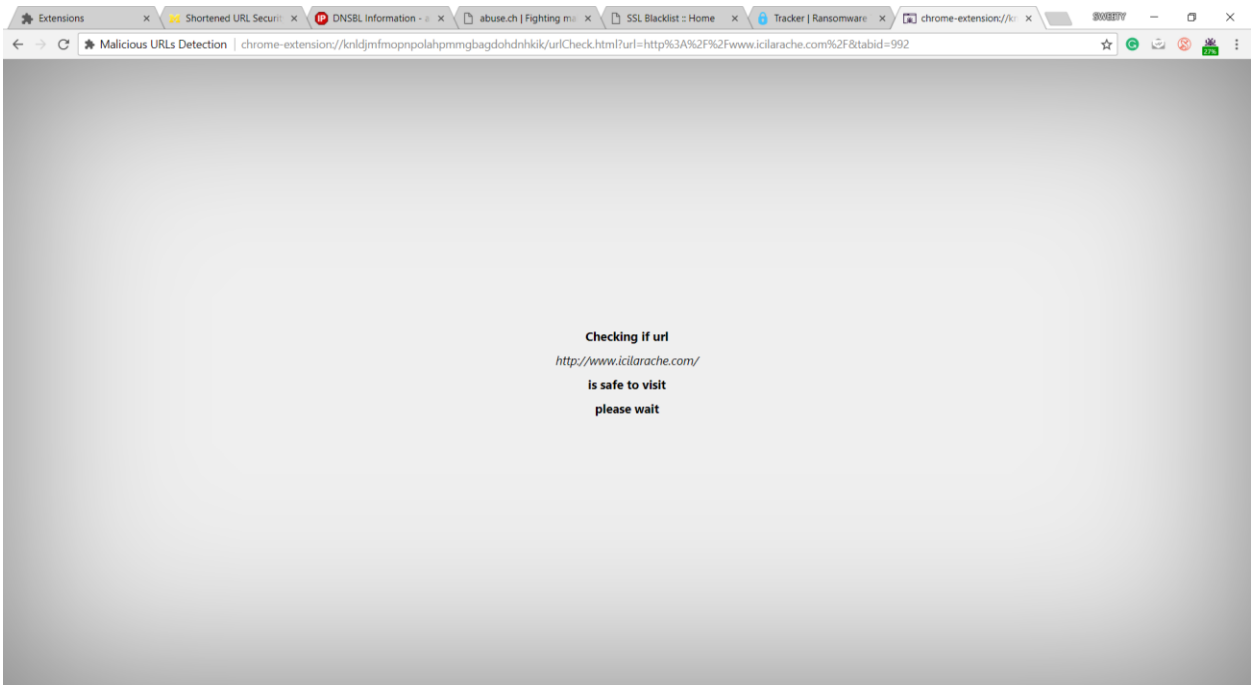

Figure 21: Extension Testing on Malicious Website

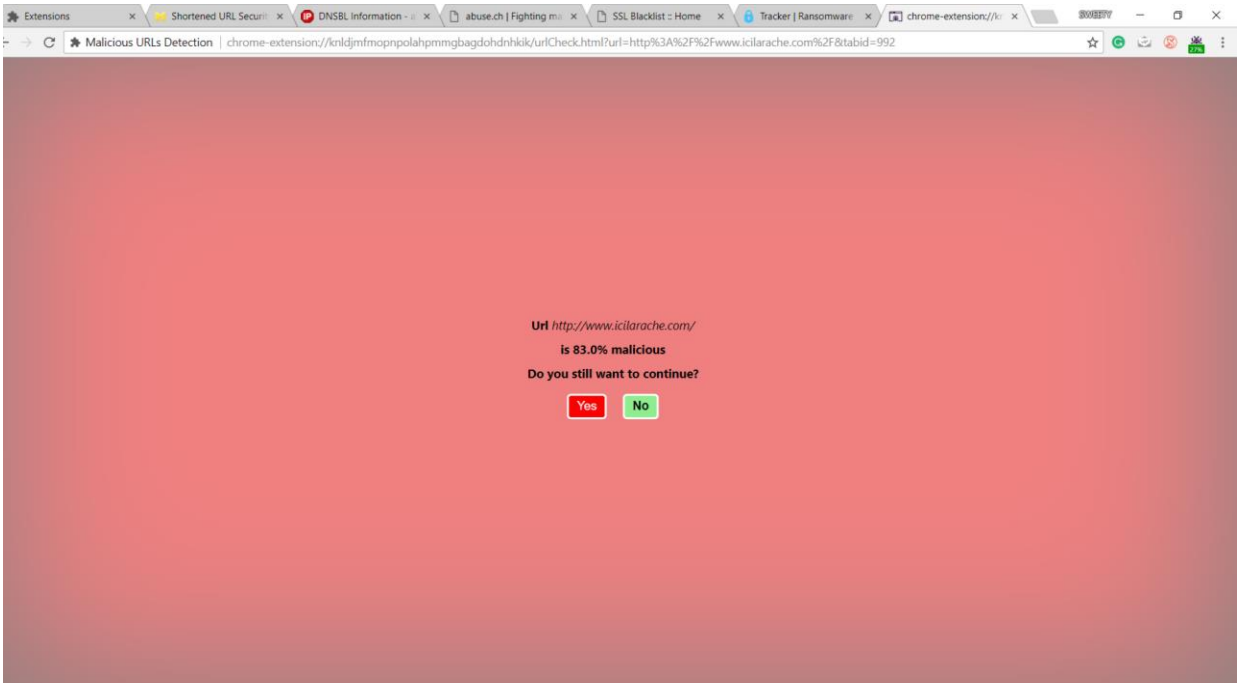

Figure 22: Extension Determined URL to be Malicious

7.2 Problems Encountered

Even with the increase of the types of ‘blocker’ extensions, none of them incorporated machine learning techniques to detect suspicious websites. Since there is no related prior work, no documented approaches were found on how to handle the problems encountered during the extension phase. Some of the difficulties faced are listed below.

- It is impossible to create custom error pages in Chrome yet, so there is need to create another page and redirect when the script runs in the background.
- The simplest error while defining parameters results in an unusual behavior of the webpage, such as loading outside of the extension or performing custom actions on the current page when reloading.

- The badge needs to be updated whenever user loads a webpage; every page has at least a few HTTP requests to load. And, Python script needs to be rewritten for accessibility by JavaScript files.
- There are other implementations that can be used to run Python script through third-party services, but the dependencies of the script are too difficult to compile in a raw implementation using any JavaScript library.
- When we first ran extension on the Macintosh OS, it produced an empty result even when tried running the script from node to bash. But found an alternate method by opening the Chrome browser directly through the terminal. No such problems found with Windows OS. We assume the problem might be with Chrome Native Messaging API.

8. CONCLUSION

Malicious websites have become very prominent these days and threatened the security of the Internet. Machine learning approach is most effective for the detection of malicious URLs when compared to general blacklisting technique. According to Sahoo et al. [1], blacklisting cannot detect new malicious URLs, whereas a properly trained machine learning model can identify a malicious URL by studying its features, with higher accuracy. There is motivation to trust that the extent of issues increments with present-day advances, for example, shortening services, where no recognition framework is intended to use for short URLs can go undetected. Detecting malicious websites before we get affected in one or other way is very crucial. Keeping that in the mind, with the availability of many URL shortening services, there are no prior methods that classify the short URLs as malicious or benign.

In this project, a lightweight machine learning approach with lexical features is designed to detect malicious shortened URLs. With this proposed method, malicious websites can be detected just by using the URL string. Most of the existing features that identify a suspicious URL are categorized in the Feature Representation section. With an eye over developing a real-time shortened URL classification, this project is extended to provide a browser plugin that classifies any given short or regular URL as malicious or benign. Along with the classification, this machine learning extension also provides the risk percentage of the website along with the original URL. The selected algorithm,

Random Forest, presents a diverse set of features that will help the model achieve a higher accuracy rate.

9. FUTURE WORK

In this project, different machine learning techniques are used to detect the malicious URLs. Along with this, one can do an additional study by considering the site-based popularity, lexical features of the URL and Host-based features. This study can be helpful in classifying the URLs and complement the machine learning models.

This paper classifies the URL as either malicious or benign. We can dig deeper into this model and classify the types of malicious URLs into phishing, spamming, and malware. This classification will help the user in taking better decisions. Incorporation of the best concepts from various machine learning algorithms is suggested to achieve a better accuracy with reduced training and prediction time.

Online approach should be considered to collect live data, which keeps the model up-to-date by training with recent malicious URLs that might use new techniques that are yet to be determined.

Furthermore, the plugin which is created as part of this project is a static plugin. The machine learning model should be able to store all the encountered URLs from the browser on the server and update the model to give us more accurate results in real time. Updating the trained model might complicate the entire process, but we can perform weekly or monthly updates to reduce the complexity.

REFERENCES

[1] D. Sahoo, C. Liu, and S. Hoi, "Malicious URL Detection using Machine Learning: A Survey", *CoRR*, abs/1701.07179, pp. 1-17, 2017.

[2] J. Crove, "10 Must-Know Cybersecurity Statistics for 2018", 2018. [Online]. Available: https://blog.barkly.com/2018-cybersecurity-statistics. [Accessed: Jan 2018].

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

[4] M. Darling, "A Lexical Approach for Classifying Malicious URLs", 2015. [Online]. Available: http://digitalrepository.unm.edu/ece_etds/63. [Accessed: Oct 2017].

[5] A. Moshchuk, T. Bragin, S. Gribble, and H. Levy, "A Crawler-Based Study of Spyware on the Web", *Proceedings of the 13th Network and Distributed System Security (NDSS)*, pp. 1-15, 2006.

[6] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose, "All Your iFRAMEs Point to Us", *Proceedings of the 17th Conference on Security Symposium*, pp. 1-15, 2008.

[7] Y. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities", *Network and Distributed System Security (NDSS)*, pp. 1-13, 2006.

[8] J. Ma, L. Saul, S. Savage, and G. Voelker, “Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs”, *Knowledge Discovery in Databases (KDD)*, pp. 1-9, 2009.

[9] A.Sayamber and A. Dixit, “Malicious URL Detection and Identification”, *International Journal of Computer Applications(0975-8887)*, vol. 99, no. 17, 2014.

[10] S. Lee and J. Kim, "WarningBird: A Near Real-Time Detection System for Suspicious URLs in Twitter Stream," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 3, pp. 183-195, 2013.

[11] S. Sinha, M. Bailey, and F. Jahanian, “Shades of grey: On the effectiveness of reputation-based blacklists”, *Proceedings of 3rd International Conference on Malicious and Unwanted Software(MALWARE)*, pp. 57-64, 2008.

[12] “Find Website Traffic, Statistics, and Analytics”. [Online]. Available: https://www.alexa.com/siteinfo. [Accessed: April 2018].

[13] Bayes’ Theorem. (n.d.). In Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Bayes%27_theorem. [Accessed: Mar 2018].

[14] N. Donges, “The Random Forest Algorithm”. [Online]. Available: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd. [Accessed: Feb 2018].

[15] A. J, “Malicious_n_Non-Malicious URL”, 2017. [Online]. Available: https://www.kaggle.com/antonyj453/urldataset/data. [Accessed: December 2017].

[16] J. Ma, L. Saul, S. Savage, and G.Voelker, "Identifying Suspicious URLs: An Application of Large-Scale Online Learning", *Proceedings of the 26th International Conference on Machine Learning*, pp. 1-7, 2009.

[17] M. Hawamdah, "Random Forest", 2012. [Online]. Available: https://www.slideshare.net/m80m07/random-forest. [Accessed: March 2018].

[18] R. Nepali, Y. Wang, and Y. Alshboul, "Detecting Malicious Short URLs on Twitter", *21st Americas Conference on Information Systems*, pp. 1-6, 2015.

[19] "Learn Extension Basics: Overview". [Online]. Available: https://developer.chrome.com/extensions/overview. [Accessed: Feb 2018].

[20] "Apps: Native Messaging". [Online]. Available: https://developer.chrome.com/apps/nativeMessaging. [Accessed: Feb 2018].