Computer Science Clinic

Midyear Update for
Proofpoint, Inc.

Predicting Malicious URLs

December 9, 2016

Team Members

Vidushi Ojha (Project Manager)
Aidan Cheng
Kevin Herrera
Carli Lessard

Advisor

Elizabeth Sweedyk

Liaisons

Thomas Lynam
Mike Morris

1 Introduction

Proofpoint is a cybersecurity company that provides security and data protection solutions to other companies. Amongst their many products, they provide an inbound email URL screening service that scans URLs embedded in clients' emails, and determines whether they lead to sites containing malware. Proofpoint's current solution redirects every URL embedded in an email through their servers, where they employ a filter to distinguish between URLs that should and should not be blocked. If their filter determines that the URL is suspicious, the URL is sent to Proofpoint's sandboxing environment for further testing. Sandboxing is an expensive process, and it is infeasible to sandbox most, or even many, of the billions of URLs Proofpoint's security suite sees every day. Thus, the Proofpoint Clinic team aims to improve the accuracy of the filter and decrease the number of URLs sandboxed, avoiding the expensive operation as much as possible.

2 Goals

The goals of the project are to create a classifier that classifies a URL as either clean or malicious, or indicates that the URL should be sandboxed. The classifier must satisfy the following system requirements.

- For any given input URL, the system returns a score between 0 and 1, where the score indicates the probability of the URL being malicious.
- Using the above score, each sample will be classified as either malicious, clean, or in an intermediate zone where it ought to be sandboxed for further investigation. The classification will be based on some threshold values.
- Our model will explain how the score was assigned, for instance by pointing to characteristics of the URL that make it more likely that it is malicious.

3 Current Progress

3.1 Support Vector Machines

Support Vector Machines (SVMs) have been shown to be effective classifiers for text classification problems. We used an open-source SVM classifier,

scikit-learn, on our data. The major problem we encountered was that training was slow, and the more features and training data we used, the slower it became. When we limited features and training time, the SVM did not produce meaningful predictions. Furthermore, SVMs are intended to be used offline. We anticipate there would be considerable overhead required to build an SVM that operates online. Because of this, working on a classifier using an SVM is not a priority for our team at the moment.

3.2 Naive Bayes

3.2.1 Open-source implementation

We first used open-source code to implement a Naive Bayes classifier that classifies a URL as clean or malicious. The features employed in this classification are tokens extracted from the URL: for an input URL, every section between a forward slash character (`'/'`) and every section between a period (`'.'`) was treated as a separate feature of the URL. We also used 4-grams as features, which means taking every sequence of 4 characters as a feature. The classifier produced a probability for the sample belonging to each class (clean or malicious), and we took the higher probability as the label assigned to the sample. In this way, we got the following results:

```
Caught:  105967 (76.8% of all samples)
Missed:  32015 (23.2% of all samples)
Malicious missed:  2904 (46.3% of all malicious samples)
```

From this, we can see that the accuracy is relatively high: we are correctly categorizing almost 77% of samples. However, for this project, we are most interested in not letting malicious samples slip by. In this respect, we are missing 46.3% of samples, suggesting we could improve in this metric. In comparison, Proofpoint's existing solution using a linear regression model misses only 19% of the malicious samples.

3.2.2 Scikit-learn implementation

Due to some complications with the above implementation, we also tried using the well-documented scikit-learn Naive Bayes source code. For the first few experiments, we used a Multinomial Naive Bayes classifier, and ran a 6-fold testing technique. (This means splitting the data set into six, training on five of them, and testing on the one holdout. This process is repeated

six times.) The features used in this classifier were the character 4-grams referred to above. In this case, the results were:

```
Caught:  128162 (92.9% of all samples)
Missed:  9821 (7.1% of all samples)
Malicious missed:  3904 (62.3% of all malicious samples)
```

As we can see, this implementation produces better accuracy results, but is worse in terms of catching malicious samples. We believe, however, that this is due to only having used 4-grams as features. Because of the better documentation and support, we intend to continue working with this Naive Bayes implementation.

3.3 TensorFlow

In order to take advantage of the next growing trend in machine learning, deep learning, we began researching into some deep learning software packages offered by open source libraries. Google has an open source library, TensorFlow, that provides implementations of deep learning classifiers. TensorFlow has been used for spam classification, which suggested that it could be useful in our project. Using an existing spam classifier, we constructed a classifier for our data. We currently use tokenization on the URL to generate the features that we give the classifier, tokenizing on characters such as slashes and periods, similar to what was done in the Naive Bayes portion of our project. We train our classifier for a set number of iterations, and then classify new data to see the classifier's performance in terms of precision and accuracy. One downside of our current classification method is that it only outputs its prediction of whether a URL is malicious or not. It would be more helpful in our project if the classifier output a probability of whether a URL is malicious in order to use probability thresholds to determine whether the URL should be sandboxed or not.

4 Future Work

4.1 Naive Bayes

We intend to continue working with the scikit-learn implementation of the Naive Bayes classifier, since this offers many more options to explore than the previous implementation we were using. In particular, we intend to

investigate the different types classifiers available. In addition, scikit-learn makes it straightforward to add different types of features to the pipeline of the classifier. Currently, we are only using the character 4-grams, but we intend to look into adding more features around the string of the URL itself as well as around other data we get from Proofpoint. See the appendix for more details.

Notably, we have yet to implement any kind of thresholding that would give us an indication of when URLs ought to be sandboxed. We hope to achieve better results on our data set before moving on to estimating thresholds.

4.2 TensorFlow

The TensorFlow open source code we are currently using gives room for custom parameter setting, such as the maximum number of features being used within the classifier, so next semester we plan on thoroughly testing these customizable parameters within our classifier in order to determine which changes yield the fastest and best results. Maximum number of features used is one of the parameters that we are allowed to change; thus, we will test which features and number of features improve performance. We will also test how long we should be training our classifier by analyzing the cost-benefit of spending different amounts of iterations on the training of the classifier. Most importantly, TensorFlow provides a variety of optimization algorithms to use, which differ in how they minimize the error of classifications. We will test these algorithms to determine which one works the best for our classification purposes.

5 Appendix

A more detailed list of the experiments we plan to perform for Naive Bayes is given below.

Classifiers

- Multinomial (current) – uses how frequently a feature is associated with a particular class
- Bernoulli – uses whether a feature is present or not
- Gaussian – assumes all features have a normal distribution

Features

- Tokenization on punctuation of the URL (as in our previous classifier)
- IP address associated with email
- Number of messages sent from this IP address before