Report on Internship Project

UNDERGONE IN DETECTION OF MALICIOUS BESTENLIST

SUBMITTED BY

SUBHASHINI S

ABSTRACT

The detection of malicious URLs is one of the highest priority issues for cyber security practitioners. Despite the large number of studies that have examined different machine learning techniques to address the issue, the most used approach remains blacklisting. The main obstacle of using machine learning is the difficulties in data collection. This paper examines the possibility of identifying malicious URLs with the help of analysis only of lexical-based futures. For the analysis, an experiment was designed. But before that, the known lexical characteristics of the malicious URLs were examined based on previous studies. The classifier showed a fairly good average accuracy rate of 94%. But it was also noticed that the classifier showed a poor FP rate, which increases the risks of encountering malicious URLs. Additionally, correlation analysis using Spearman's coefficient showed that the URL length and number of special characters are the most determining signs of malicious URLs

I. INTRODUCTION

Phishing websites are being employed to steal personal information, such as credit cards and passwords, and to implement drive-by downloads. Phishing is popular among muggers since it is easier to trick someone. In most cases, such annoying activity engages network resources intended for other use into clicking a malicious link which seems legitimate than trying to break through a computer’s defense systems. In most cases, such annoying activity engages network properties intended for other uses, and nearly always threatens the security of the network and/or its data. Properly designing and deploying a Phishing URL will help block the intruders. phishing domain (or Fraudulent Domain) characteristics, the features that discriminate them from appropriate domains, why it is important to detect these domains, and how they can be detected using machine learning techniques.

II. UNIFORM RESOURCE LOCATOR (URL)

A URL is a exclusive identifier used to locate a resource on the internet. It is also denoted to as a web address. URLs consist of multiple parts -- including a protocol and domain name -- that tell a web browser how and where to recover a resource. End operators use URLs by typing them directly into the address bar of a browser or by ticking a hyperlink found on a webpage, bookmark list, in an email or from additional application. A URL is the most collective type of Uniform Resource Identifier (URI). URIs are strings of typescripts used to identify a source over a network. URLs are vital to traversing the internet.

URL STRUCTURE :

The URL encompasses the name of the protocol required to access a resource, as well as a resource name. The first portion of a URL identifies what protocol to use as the primary access medium. The second portion identifies the IP address or domain name -- andpossibly subdomain -- where the resource is located.

MALWARE URL :

By 'malware', this paper refers to the URLs that trigger a downloading of hostile or intrusive software. Cybercriminals design malware to compromise the integrity, confidentiality and availability of a user device. The most common techniques of these are cross-site scripting (XSS) and drive-by-download attack attacks If phishing attacks rely on users' unconsciousness, then malware URLs are designed specifically for the vulnerability of web browsers or web applications that are developed on different platforms. There are several reasons for the difficulties of detecting malware URLs. The main reason is that malware attacks conducted with help of URLs are usually developed in different programming languages such as PHP, ASP or JSP. Hence, it isnecessary to individually develop safety requirements for web applications on the internet.

III PACKAGE & REQUIREMENT

PYTHON 3

Pip install pandas

Pip install numpy

Pip install scikit-learn

Pip install matplotlip

PANDAS:

Pandas is best Python library for machine learning because of providing high-performance and high-level, So is it easy to handle for data structure and data analysis.

NUMPY:

It performs statistical computations in Machine Learning.

SCIKIT LEARN:

Scikit-learn another most popular open-source Python Libraries For Machine Learning with a popular model. Model is Linear Classification, Linear Regression, Lasso-Ridge, Logistics Regression, Decision Tree, Random Forests, K-Means Clustering, KNN, Dimensionality Reduction, and many more.

MATPLOTLIP:

Matplotlib is one of those python libraries for machine learning that is used for Data Visualization.

comma-separated values:

A CSV (comma-separated values) file is a text file that has a specific format which allows data to be saved in a table structured format.

Logistic Regression in Python:

Now that you understand the fundamentals, you're ready to apply the appropriate packages as well as their functions and classes to perform logistic regression in Python. In this section, you'll see the following:

- A summary of Python packages for logistic regression (NumPy, scikit-learn, StatsModels, and Matplotlib)
- Two illustrative examples of logistic regression solved with scikit-learn
- One conceptual example solved with StatsModels
- One real-world example of classifying handwritten digits

Logistic Regression Python Packages:

There are several packages you’ll need for logistic regression in Python. All of them are free and open-source, with lots of available resources. First, you’ll need NumPy, which is a fundamental package for scientific and numerical computing in Python. NumPy is useful and popular because it enables high-performance operations on single- and multi-dimensional arrays.

Logistic Regression Assumptions:

- Binary logistic regression requires the dependent variable to be binary.
- For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
- Only the meaningful variables should be included.
- The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
- The independent variables are linearly related to the log odds.
- Logistic regression requires quite large sample sizes.

IV SOURCE CODE

```
try:

    from urlparse import urlparse
except:
    from urllib.parse import urlparse

def uri_validator(x):
    try:
        result = urlparse(x)
        return all([result.scheme, result.netloc, result.path])
    except:
        return False
val = input("Enter Any Url: ")
print(uri_validator(val))


import numpy as np
import pandas as pd
import os
import seaborn as sns
import sklearn.linear_model as lm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
dataset = pd.read_csv("~/Desktop/data.csv")
dataset.head()
y = data["label"]
 url_list = data["url"]
```

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(url_list)

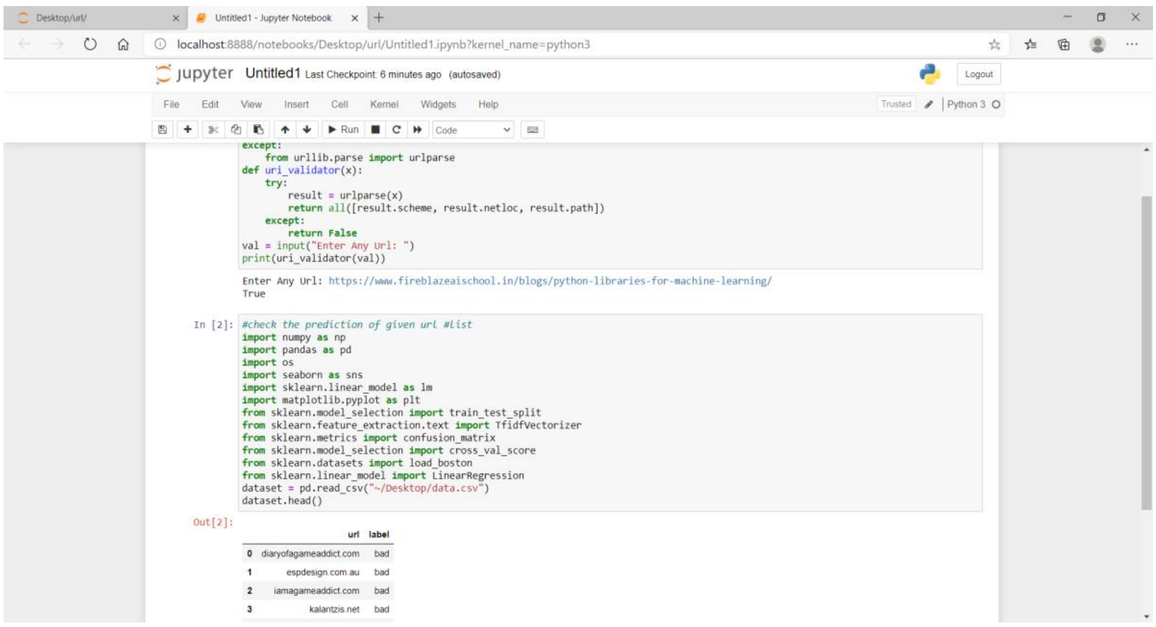
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

logit = LogisticRegression()

logit.fit(X_train, y_train)

print("Accuracy of our model is: ",logit.score(X_test, y_test))

V. OUTPUT



localhost:8888/notebooks/Desktop/url/Untitled1.ipynb?kernel_name=python3

jupyter Untitled1 Last Checkpoint: 6 minutes ago (autosaved)

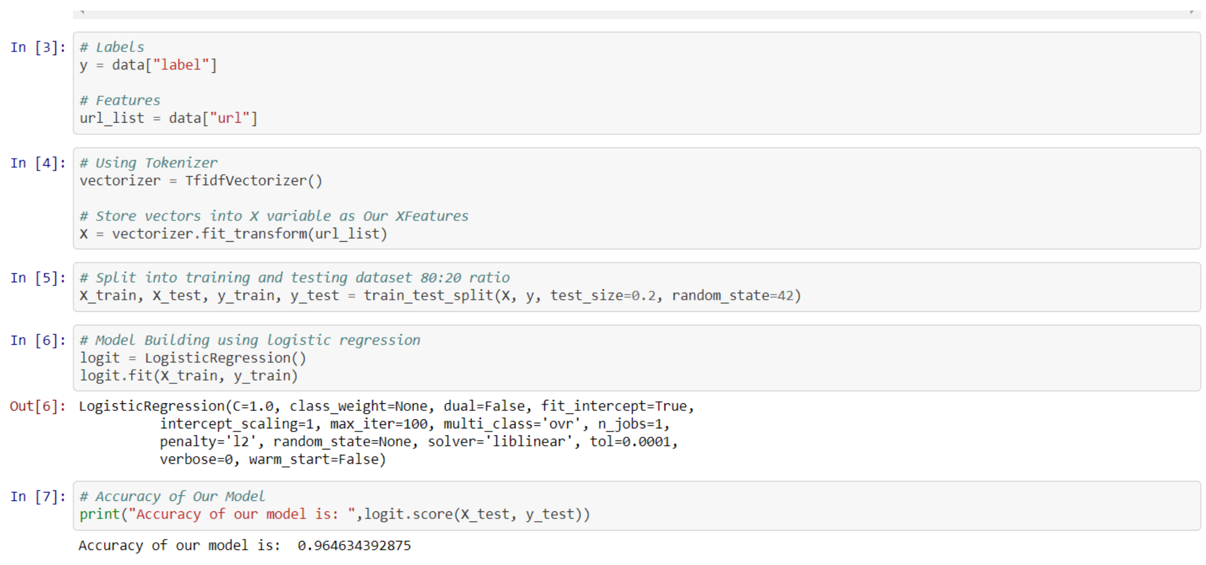
```
except:
    from urllib.parse import urlparse
def uri_validator(x):
    try:
        result = urlparse(x)
        return all([result.scheme, result.netloc, result.path])
    except:
        return False
val = input("Enter Any Url: ")
print(uri_validator(val))
```

```
Enter Any Url: https://www.fireblazeaischool.in/blogs/python-libraries-for-machine-learning/
True
```

```
In [2]: #check the prediction of given url #list
        import numpy as np
        import pandas as pd
        import os
        import seaborn as sns
        import sklearn.linear_model as lm
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.metrics import confusion_matrix
        from sklearn.model_selection import cross_val_score
        from sklearn.datasets import load_boston
        from sklearn.linear_model import LinearRegression
        dataset = pd.read_csv("~/Desktop/data.csv")
        dataset.head()
```

Out[2]:

	url	label
0	diaryofagameaddict.com	bad
1	espdesign.com.au	bad
2	iamagameaddict.com	bad
3	kalantzis.net	bad



```
In [3]: # Labels
        y = data["label"]

        # Features
        url_list = data["url"]

In [4]: # Using Tokenizer
        vectorizer = TfidfVectorizer()

        # Store vectors into X variable as Our XFeatures
        X = vectorizer.fit_transform(url_list)

In [5]: # Split into training and testing dataset 80:20 ratio
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [6]: # Model Building using logistic regression
        logit = LogisticRegression()
        logit.fit(X_train, y_train)

Out[6]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                  intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                  penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                  verbose=0, warm_start=False)

In [7]: # Accuracy of Our Model
        print("Accuracy of our model is: ",logit.score(X_test, y_test))

        Accuracy of our model is:  0.964634392875
```

VI CONCLUSION

In this work, we have described how a machine can able to judge the URLs based upon the given feature set. Specifically, we described the feature sets and an approach for classifying the given the feature set for malicious URL detection. When traditional method fall short in detecting the new malicious URLs on its own, our proposed method can be augmented with it and is expected to provide improved results. Here in this work, we proposed the feature set which can able to classify the URLs.

VII REFERENCE

Acunetix (2017) Acunetix, [online]. Available at:
<https://www.acunetix.com/blog/articles/injection-attacks> [Accessed: 1 May 2018].

Aggarwal, A., Rajadesingan, A. and Kumaraguru, P. (2012) ‘PhishAri: Automatic realtime phishing detection on twitter’, eCrime Researchers Summit, eCrime, pp. 1–12. doi: 10.1109/eCrime.2012.6489521.

Alcaide, A., Blasco, J., Galan, E. and Orfila, A. (2011) ‘Cross-Site Scripting: An Overview’, 75. doi: 10.4018/978-1-60960-765-4.ch004.

Aleroud, A. and Zhou, L. (2017) ‘Phishing environments, techniques, and countermeasures: A survey’, Computers and Security, 68(May), pp. 160–196. doi: 10.1016/j.cose.2017.04.006.
Alsharnouby, M., Alaca, F. and Chiasson, S. (2015) ‘Why phishing still works: User strategies
for combating phishing attacks’, International Journal of Human Computer Studies. doi: 10.1016/j.ijhcs.2015.05.005.
Anthony, E. (2007) ‘Phishing : An Analysis of a Growing Problem Phishing ’. Avaliable at: <https://www.sans.org/reading-room/whitepapers/threats/phishing-analysis-growing-problem-
1417> [Accessed: 1 May 2018].
Antunes, Carlos Henggeler;Henriques, C. O. (2016) Multiple Criteria Decision Analysis,
Apple Inc. (2017) macOS Sierra: Back up with Time Machine. Available at:
<https://support.apple.com/kb/PH25710?locale=ru_RU&viewlocale=en_US> [Accessed: 2 June 2017]. Banday, M. T. and Qadri, J. a. (2007) ‘Phishing – A Growing Threat to E-Commerce’, The