Internship Report for March - April 2018

Yash Saboo

2015-2019 Batch

T.Y.C.S.

15070121170

**MORE DETAILS THE PROJECT**

**Problem Statement:**

To develop a Google Chrome Web Extension which can detect phishing/imitation websites and alert users about the same based on previously browsed websites.

**Technology Used and Technical Details:**

Following technologies were used:

**For Layer 1:**

1. PHP for algorithm implementation.
2. Composer, MySQL and Apache Server in the integration with wamp64 for PHP execution.
3. Python to download the dataset from an online repository.

**For Layer 2:**

1. JavaScript for implementing the extension in Chrome.
2. Python to implement the algorithms
3. Flask, a web-framework, to host python scripts
4. REST APIs for connecting Client (Chrome Extension) to Server (Flask)
5. jQuery AJAX for using POST method of REST
6. Jupyter Notebooks for easy debugging of Python scripts.
7. Open\_SSL software to create certificates to enable HTTPS connections on server.

These technologies were learnt and implemented, as when required, using various resources available on Internet. I enrolled in an online course on Udemy called "Chrome Extensions”: Develop 5 chrome extensions from scratch", taught by Vishwas Gopinath.

Certificate Link: <http://ude.my/UC-99EO7F48>

**TASKS DONE**

**Layer 1:** The algorithm exploration and POC development went hand-in-hand by following the following flow: (**Continued** from previous report)

**Flowchart 1**

* **Choosing Algorithm and Technique** (February)
* **Data Selection** (February)
* **Data Preprocessing** (February)
* **Development** (February – 1st week of March)
  + It was completed by first week of March with few tweaks in the algorithm implementation with some additional functions which facilitated in testing phase.
* **Testing** (2nd week of March)
  + A PHP script was written to test all the algorithms which would iterate through the dataset automatically and display the True Negative (TN), False Negative (FN), False Positive (FP) and True Positive (TP) for each test case along with various other parameters of hash.
  + The script would store all this information in a CSV Output file and summarize the complete testing a Text Document automatically.
  + The testing was done in two phases:
    - *Training Set*

The algorithms were tested on a small set of data and cross-checked manually whether the results were up to the mark or not.

* + - *Testing Set*

The algorithms were now tested on the complete dataset to prepare the benchmarking of each algorithm.

* **Benchmarking** (2nd and 3rd week of March)
  + To benchmark the algorithms, I found the sensitivity (also called the true positive rate) and specificity (also called the true negative rate) of each algorithm.
  + Following definitions were used:

|  |  |  |
| --- | --- | --- |
|  | *Phishing is present* | *Phishing is absent* |
| *Detects Phishing* | True Negative (TN) | False Negative (FN) |
| *Doesn’t Detect Phishing* | False Positive (FP) | True Positive (TP) |

* + - Sensitivity = TP / (TP + FN)
    - Specificity = TN / (FP + TN)

Note: This image hashing algorithm was currently being termed as **“Layer 1”** among other layers/features used to detect Phishing.

The forthcoming layers/features which will be incorporated in the extension will go through the same tasks as given in the **“Flowchart 1”.** The ‘Data Selection’ and ‘Data Preprocessing’ tasks might get skipped because the processed data used in other previous layers might be reused.

**Pre-Tasks:**

Before implementing the Layer 2, the following pre-tasks had to be done like:

* **Developing Chrome Extension**
  + The base extension was already available on Symantec Bitbucket.
  + I understood the variable and function descriptions so that I could amend it later with ease.
  + I made few amends in so that I can implement the Layers without any fuss.
* **Set Up Client-Server model**
  + The forthcoming algorithms were supposed to be implemented on Python.
  + Now as browsers are only capable of debugging JavaScript, I had to host Python on some server which had a Python interpreter installed on it.
  + I faced many challenges and learnt a lot while I was solving them. I describe few of them and they are as follows:
    - As I had used WAMP server for previous little tasks, I looked up if I could run Python on it. I installed mod\_wsgi Apache Module and made few changes in the server httpd.conf file so that I could run Python Script. I successfully could run the Python script but I had little trouble in connecting it to Chrome Extension because I had to adhere to few tricky Chrome permissions.
    - I had to enable HTTPS for the server or, else, Chrome denied me the connection. So, I downloaded Open-SSL, created a certificate and managed to configure the server with secure connection.
    - I still had complications because Google Chrome didn’t allow Invalid Certificates, as the certificate I had created was not signed by legit Certificate Authority.
    - While researching on all of this, I came across Flask and Django frequently. So, I tried my luck with Flask and was able to run Python script on the browser easily as Flask hosts its own server. I had to again make few changes in Flask scripts to enable HTTPS for the Flask server. It was comparatively easier than Wamp Server.
  + Once the server was ready, I had to use to REST APIs to implement message passing between them. I used POST method with jQuery Ajax in Chrome Extension to connect to Flask.
  + Python has a library called ‘requests’ which facilitated in implementing the server-side connection.
  + This client-server architecture was tested and was successfully ran by resolving some minor changes.
  + Now that the client-server model was working properly, I focused only on the Server-side scripts as all layers will be developed individually and the connection will only take place with the combined version of all the layers.

**Layer 2:**

* **Choosing Algorithm and Technique** (March)
  + The next technique required scraping the website using URL and implement the algorithm to define how similar and dissimilar are the website from other cached websites.
* **Data Selection** (March)
  + The dataset was once again downloaded from an online repository, PhishTank.
  + This layer could not use the previous dataset because it required “Verified” and “Online” phishing websites so that the page content generated by the script was genuine.
* **Data Preprocessing** (March)
  + A Python script was written in Jupyter Notebook with following functions to automate the process:
    - Download Latest Phistank Data
    - Unzip Phistank Data
    - Parse to JSON format
    - Gets URLs from the data which are ONLINE + VERIFIED + TARGET="?"
    - Handles both JSON Parsing and Getting URLs
* **Development** (March)
  + The algorithm was written in Jupyter Notebook using following libraries:

|  |
| --- |
| from flask import Flask, request, json  from bs4 import BeautifulSoup  import difflib  import urllib.request |

Beautiful Soup is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping.

Extract the URL's from for an input website

|  |
| --- |
| from bs4 import BeautifulSoup import requests url = raw\_input("Enter a website to extract the URL's from: ") r  = requests.get("http://" +url) data = r.text soup = BeautifulSoup(data) for link in soup.find\_all('a'):     print(link.get('href')) |

Extract HTML content of URL specified and compare the title and domain of the website

|  |
| --- |
| import urllib.request  from bs4 import BeautifulSoup  first\_url = "https://www.netflix.com/in/"  domainName = first\_url.split("//")[-1].split("/")[0].split('?')[0].split(".")[0] #Extract domain name  page1 = urllib.request.urlopen(first\_url) #Get URL HTML contents  text1 = BeautifulSoup(page1, 'html.parser') #Convert to BeutifulSoup  textx = str(text1.prettify()) #Convert the HTML in its form  titleOfPage = text1.title.string) #Extract title of the page from HTML <title> tag in <head>  if(titleOfPage == domainName):  print("Same Title as the Domain")  else:  print("Different Title as the Domain") |

Similarly, to extract iframes

|  |
| --- |
| for iframe in iframexx:  response = urllib2.urlopen(iframe.attrs['src'])  iframe\_soup = BeautifulSoup(response) |

The difflib module contains tools for computing and working with differences between sequences. It is especially useful for comparing text, and includes functions that produce reports using several common difference formats.

A unified diff includes only the modified lines and a bit of context. The unified\_diff() function produces this sort of output.

|  |
| --- |
| difflib\_unified.py  import difflib  from difflib\_data import \*  diff = difflib.unified\_diff(  text1\_lines,  text2\_lines,  lineterm='',  )  print('\n'.join(diff)) |

* + Further development of the algorithm is in progress.
* **Testing** (April)
* **Benchmarking** (April)

Currently, I am amidst Development stage.

All these files are uploaded regularly on the Bitbucket of Symantec using Git.

**FUTURE WORK**

Once the Layer 2 has been properly implemented, layers will either be combined or next layer will be implemented. They will be described in the next report (March - April).

Thank You.