An Efficient and Usable Client-Side Cross Platform Compatible Phishing Prevention Application

SECOND REVIEW

Submitted by

N. Dhanush(2016103021)

G. Santhosh(2016103057)

S. Ben Stewart(2016103513)

Guide
Dr. Angelin Gladston
Associate Professor
Department of Computer Science & Engineering

College of Engineering, Guindy

ANNA UNIVERSITY: CHENNAI 600 025

JANUARY 2020

1 PROBLEM STATEMENT

Phishing is a crime where the victim is contacted by an attacker posing as a trustworthy source and lure them into providing sensitive information like credit card details and personal identification numbers. These attacks are currently being blocked by web browsers that have a list of such phishing links. It takes several days and intense computing resources to prepare the list. Having a time lag in this process means that many victims are vulnerable at that point in time to such an attack. Inorder to make the process more efficient, the functionality required will be ported to the client side of the web browser. This makes sure that the time delay is averted and the phishing attack can be thwarted with fewer computational resources.

To solve the above mentioned problem, we will be implementing a web browser add-on that works as a background script on the client side. All the background scripts required will be made cross platform compatible to make the development easier and more efficient.

2 INTRODUCTION

In this chapter we will be discussing how the issue of phishing began and how it was tried to be curbed both in policy writing and that of its implementation.

2.1 THE INTERNET

The Internet had its humble beginnings as a research project to share and access resources in other computers. And in fact these "other" computers were the main frames of the time that were located in the research facilities and college laboratoires of the United States of America. This Internet is not the traditional Internet that we know as the World Wide Web. It was just a network of computers that could be operated from other nodes in the network. One main roadblock that was encountered was that the main frames were expensive and far exceeded the computational requirements of the then public. As a result of this there were only a handful of expensive mainframes with a few government and private institutions in a single country. Never would they have imagined the used for the internet unless the next era of personal computers boomed. As this phase had computers be considered almost as national assets they were sometimes compromised by enemy nations performing acts of sabotage.

2.2 THE WORLD WIDE WEB

In the stage of the personal computers, people bought computers to perform basic operations like spreadsheets and graphics related tasks. Once the internet had been introduced to those machines, they were still restricted by the slow DSL connections that the telephone companies provided. But once this threshold was broken, we would think that the Internet as we know today would have started to thrive. But it did not. The way in which the information is accessed was not intuitive till the World Wide Web made its appearance in 1989 developed by Tim Berners-Lee at CERN. This era with the internet has its own set of unethical activities that were performed using computers or against the computers. Some elaborate sabotage attempts even involved using the internet connections of those computers.

2.3 THE INTERNET OF CONTENT

The World Wide Web started the phase known as the "Internet of Content" were the websites of different organisations could be accessed by any person on the internet to get to know the organisation and other details like the availability time, recruitment offers and so on. The impact of the World Wide Web was accelerated once the search engines were developed to index and display the billions of web pages that were loaded into servers connected to the Internet. This made it possible for people to access almost anything hosted on the Internet without knowing upfront the URL related to the resource. This opened a plethora of problems related to the act of performing unethical activities using or against the World Wide Web. The most common internet based crimes were phishing and site defamation. We will look into them a bit later, once after we have explained the next era of the internet. And this is where the roots of performing malicious activities for the gain of nations or individuals can be traced to.

2.4 THE INTERNET OF E-COMMERCE

For the next era to come upon, there were a few changes that were required on the internet. They were the ability to host dynamic content based on the users and the ability for the user to interact with such content. These abilities were provided by the advent of web based programming languages with Javascript leading the way. This made it possible that people could perform online cash based transactions for the services that the internet made possible in the first place. This kind of opened Pandora's box for the cyber related crimes of this day. The notoriety of phishing greatly increased because of the scope that you could have the banking credentials of several thousands of users.

2.5 THE INTERNET OF PEOPLE

And before deep diving into the domain of phishing let us have a short look into the other eras of the internet that have come along. We’ve had the “Internet of People” which was brought by the advent of social media platforms like FaceBook, Twitter and LinkedIn. People now share much more data online about themselves over the internet to the public. This has led to even more problems like social media addiction, anxiety and attention deficit among the users. But let us not just paint a dark picture of this era and move on to what the future has in store.

2.6 THE INTERNET OF MACHINES

We are currently in this era of the “Internet of Machines” were more and more IoT devices with the capability to connect to the Internet and use it to communicate with other IoT devices and some centralised computers. This is probably exciting times as even the standards of the Internet of Machines has yet to be decided and wonder if we would be having another World Wide Web like platform available with the machines in mind. Even in this age, the unethical activities can be performed as was done in the previous eras of the Internet.

2.7 PHISHING

Now that we have an understanding of what the Internet is and why it is so important and how it is being used, let us dive into the topic of phishing. If a definition were needed, phishing is any social engineering made to trick the people to access the malicious resources that may get critical information such as passwords, bank account numbers etc. from the victims. Phishing is done not only for the monetary incentives it provides but also for the impersonation of people in social media or to compromise the networks of organisations or countries. They are targeted upon the users who have access to such details like an e-commerce customer or a company manager.

2.8 SOCIAL ENGINEERING

The most common ways to phish are to send emails to those who are related. What such emails contain are the malicious links and other content to convince the users that it is indeed legitimate. The same strategy is applied to the hosted pages that are pointed to by those links. As a result of this many unsuspecting people are tricked. And in order to prevent these instances, many methods have been devised. But the most important thing is to be constantly vigilant that phishing is possible and that you might be a target and never clicking on such links. Some such phishing sites have also been indexed and are even placed above the real site in search engines. So, care must be taken even when the links are provided by the search engine.

2.9 PHISHING PREVENTION

Let us look into the other methods to prevent phishing. Since search engines have to index all pages to be displayed for the user's query, it seems logical to use some mechanism to find such links while indexing and use the same in browsers to notify users that they are accessing a page which is probably used for phishing. This works for most cases, but fails for those dynamically created pages which are not indexed by the search engines and newly created sites which have yet to be crawled because it takes a few hours for search engines to index new pages.

Thus to prevent the problems caused by the above method, new strategies based on the content in the page and also the link that the user accesses, is taken into consideration. This brings with it the advantages its own set of disadvantages like, it having to work on the client side machines and also take the time to render the webpages. Many optimisation techniques and strategies have been developed to overcome this, which will be discussed in the next chapter.

3 RELATED WORK

This chapter would be a deep dive into the methods that have been implemented or at least served in the process of creating one to detect phishing locally in the client side machines.

3.1 AUTOMATIC PHISHING CLASSIFICATION

The work by Colin Whittaker, Brian Ryner and Marria Nazif for Google provided the base benchmark for most of the future works and so would be better if we have a better understanding of what they did. For creating the base dataset required to train a machine learning model, they used the links from the Gmail spam filters and also those that were submitted by other users. The features used are the URL, the contents of the HTML page and also where the page is hosted. These features are then used by the model which is a logistic regression classifier to find if the site is used for phishing or not.

The training for the model is done offline using the blacklist for the last three months. This has to be done to account for the temporal resilience required from such models. This method of using a published blacklist introduces another risk, which is the risk of feedback loops, that pass down the same error to the classifier. This is because the list might have some false recognitions for the web pages that are submitted manually by other users. This means that the whole dataset has to be manually checked and such wrong classifications be removed. Though the percentage of such instances are very low the fact that this list has to be verified manually really is a black mark for this method. Though they achieve an impressive false positive rate of under 0.1%. Even though we consider that the black list is perfect, the fact that the system uses a black list to work means that there will be an inevitable time lag between the time the phishing site is up and that the page is detected to be used for phishing.

This time lag has to be reduced by decreasing the time taken to develop the model and thereby help the users to find even the most recent phishing pages. In spite of the shortcomings that this work had, the features for the model are

1. The URL of the page
2. The HTML page contents
3. The host server details

3.2 CANTINA

This work was further taken up by Guang Xiang, Jason Hong, Carolyn P. Rose and Lorrie Cranor in their CANTINA+ which provides a feature-rich machine learning framework for detecting phishing web sites. It is split into two phases. In the first phase, the task is to find the feature values for all the records in the database. Once this is done, the second phase is to find if the site is used for phishing or not. The above feature is limited to 15 features which are used in both the phases.

This method for performance optimization and to reduce false positives uses a hash based page removal model in which the similar looking components of the website are removed, making it easier to find the differences. Once, the similar components are removed, the presence of a login form in the HTML content is searched for. If the content matched that of the login page, then the pretrained model is used.

Since the creation of the model requires an updated list of phishing sites, the list provided by PhishTank's verified blacklist is being used. And as far as the time required to find the similar looking sites is concerned, it is greatly reduced by using the SHA 1 hashing algorithm. It is noted that though this hashing algorithm can be easily broken, it is being used for the efficiency and the high accuracy with which it finds out the phishing sites.

Though this model provides a faster way of finding those sites, it definitely comes with its own set of drawbacks. The first one being that SHA 1 algorithm that defeats the whole purpose of the malicious actor never being able to circumvent the system.

Though this system has it's disadvantages, the features are a few things that can be taken from them. They include the embedded domain, IP address, number of dots in URL, suspicious URL, number of sensitive words in URL, out of position top level domains (TLD), bad forms, bad action fields, non-matching URLs, out of position brand names, the age of domain, page in top search results, page rank and page results while searching for copyright company name and host name. Many models were used and it was found out that the Bayesian models and the Random Forests performed remarkably well.

3.3 AUTO UPDATED WHITELIST

Ankit Kumar Jain and B. B. Gupta provided another approach to protect against phishing attacks at client side using auto-updated white-list. The accurate and fast detection of phishing sites in real time environment is paramount. The time constraints of the above methods are because they use a visual similarity based approach. This can be reduced by using a heuristic based approach which depends mainly on the feature set, the classifier and the training data.

The hypothesis is based on the fact that though the phishing pages look similar to the corresponding real website, they do differ steeply in the functionality they offer. But almost all but the critical phishing functionality redirect to the corresponding real website.

To provide such functionality, a whitelist is used in this method. The whitelist contains the domain name and IP address as the parameters. The whitelist provides for the faster running time and to reduce the false positive rate. The working of the whitelist is as follows.

First when the user has never visited any site, the whitelist has no records. But when the user does so and visits a site, the whitelist is checked if it has the domain along with the IP address. If the record is present, the page is said to be a safe site. Else, the second component which is almost the same as that of the previous models kicks in to find if the requested site is a phishing site or not.

Thus, because of the whitelist the model has the advantage of being language independent and is capable of finding the embedded components in the phishing website that can cause a DDOS attack.

The model was developed further into a usable application down the lane. And this paper provided the following findings. The model provided the base for using auto-updated whitelists to speed up the process of finding out if the page is used for phishing or not. This is highly advantageous because most of the sites that a person accesses will not be a phishing page, significantly reducing the average time taken to process the site, as most of the websites visited by the end user will be safe sites and would not be ones used for phishing and become a threat.

3.4 OFF-THE-HOOK

The work by Samuel Marchal, Giovanni Armano, Tommi Grondahl, Kalle Saari, Nidhi Singh, and N. Asokan implemented the client-side phishing prevention application named Off-the-hook.

The main improvements that they provided were better privacy, realtime protection, resilience to dynamic phishing and effective warnings. It also has an emphasis on making a software application that can be easily used by the general public to protect themselves against phishing.

A brand independent, evolution resistant model to detect the phishing pages has been developed. This means that sites of all domains will be identified correctly and the temporal resistance maintained. This is that even when the malicious actor knows how the system works, and tries to figure out a work around, the system learns and adapts itself to the changing conditions.

The main upside for this project is the cross platform capabilities that it intends to provide and the user base that it can have based on the ease of use that the application provides.

The downside is as follows. All the functionality required cannot be implemented inside the browser using Javascript alone. As a result of this, machine dependence creeps into the equation. This in the long run will be a major block to the ease of use that the application says will provide.

The other main disadvantage is that the model does not take into account the static IP addresses on which the page might be hosted. This model relies on the assumption that such IP addresses that host the phishing pages will be blacklisted and removed by the host provider.

Thus based on all the above mentioned related works, we will be redeveloping the application Off-the-hook to follow the basic networking and socket connections so that the applications both running inside the browser and within the operating system of the user will remain compatible.

3.5 FUZZY ROUGH SET FEATURE SELECTION

The conference paper by Mahdieh Zabihimayvan and Derek Doran on Fuzzy Rough Set Feature Selection to Enhance Phishing Attack Detection gives us the ways of choosing the best possible features for training the model over. It uses the Fuzzy Rough Set (RFS) theory to select the most effective features and finds out that the Random Forest method gave the maximum possible F-score.

3.6 COMPARISON

Table 3.1 gives a comprehensive list of all the papers with their publications and problems solved along with the limitations that these solutions have.

Table 3.1 Comparison Table

Paper	**Publication**	**Solved**	**Limitations**
Large-Scale Automatic Classification of Phishing Pages	Proc. Netw. Distrib. Syst. Security Symp., 2010	Machine learning models can be used with reliable accuracy.	Needs blacklist for updating.
CANTINA: A feature-rich machine learning framework for detecting phishing Web sites	ACM Trans. Inf. Syst. Secur., 2011	SHA1 based similarity check for similar looking sites.	SHA1 could be manipulated.
A novel approach to protect against phishing attacks at client side using auto-updated white-list	EURASIP J. Inf. Secur., vol. 2016, no. 1, Dec. 2016	Auto-updated whitelist for faster detection of sites on average.	Not temporally resilient.
Fuzzy Rough Set Feature Selection to Enhance Phishing Attack Detection	IEEE International Conference on Fuzzy Systems, June 2019	Feature selection.	Not a user oriented application.

4 HIGH LEVEL BLOCK DIAGRAM

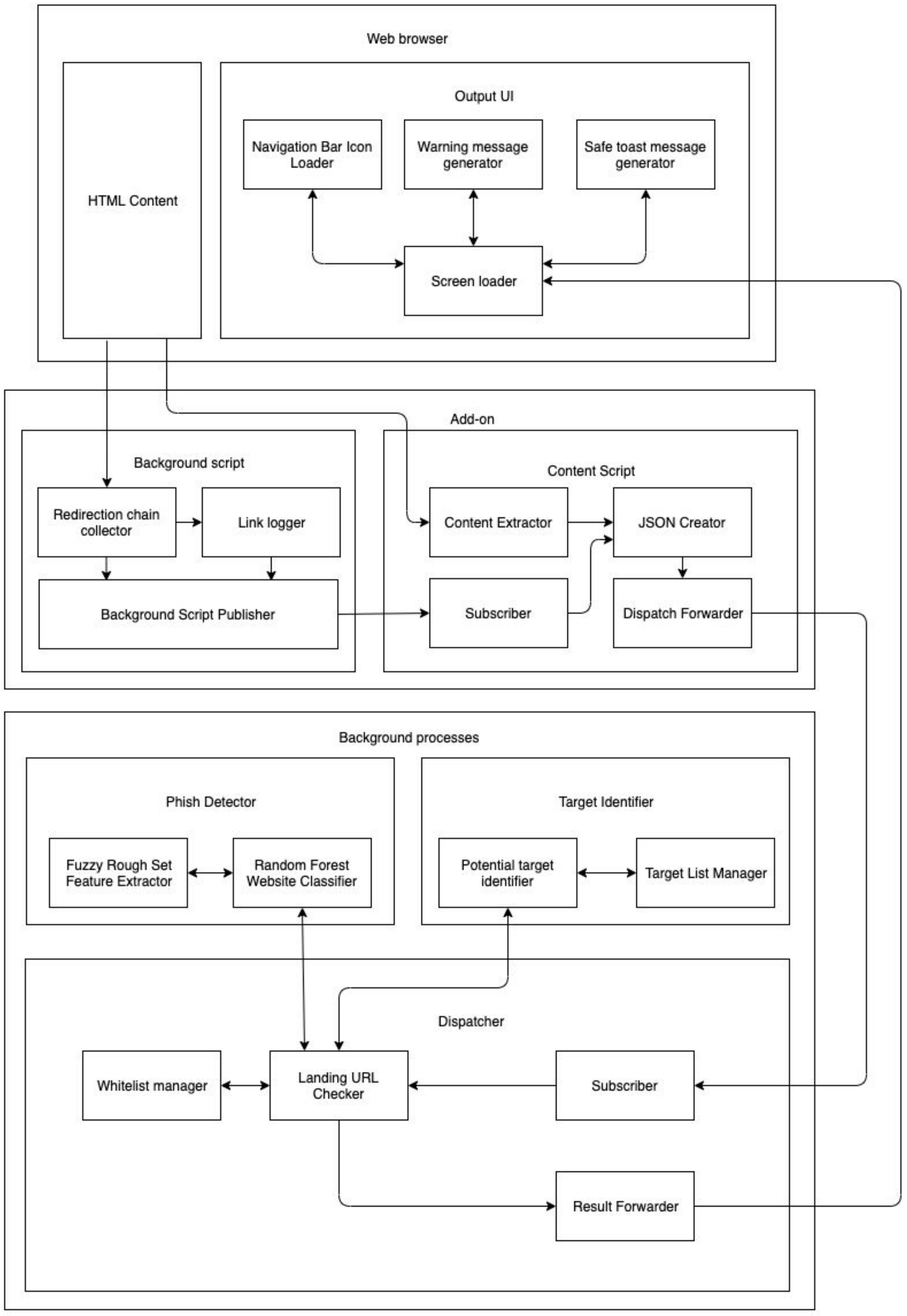


Figure 4.1 High Level Block Diagram

The above Figure 4.1 explains the detailed working of the project with the pipelines between the multiple components. To get an easier understanding, a simpler diagram has also been drawn as Figure 4.2

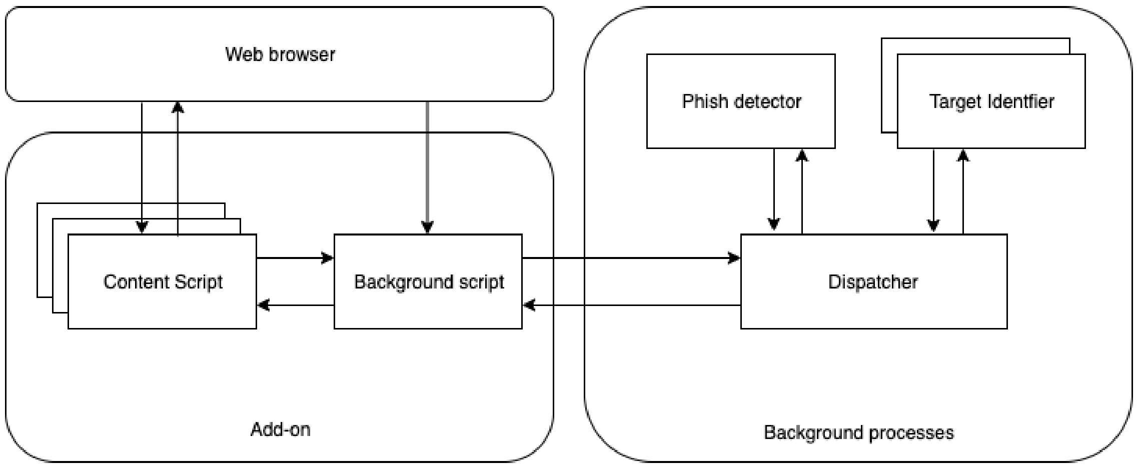


Figure 4.2 System Architecture

5 DETAILED MODULE DESIGN

This chapter will explain in detail the modules and submodules that complete the functionality for this project.

5.1 MODULES

Our system has three modules that complete its functionality. They are as follows.

1. Add on
    a. Background script
    b. Content script
2. Background process
    a. Dispatcher
    b. Phish Detector
    c. Target Identifier
3. Web Browser
    a. HTML content
    b. Output UI

5.2 ADD ON

An add on is required in the form of an extension because of the reasons that the frameworks provided by Javascript for tasks related to machine learning are still in its infant stages. And so the addon performs the two important tasks of getting the page contents from the browsers and also making changes to the elements in the web browser to convey the threat level for phishing.

5.2.1 BACKGROUND SCRIPT

The task to be done by this component is to get the list of redirection URL chains from the user's web browser, and publish it to the content script while also logging the links for reference purposes.

```
Begin
    For each page load redirect
        Add listener to that event
        Get the list of redirects from listener
        If page is fully loaded
            Send the list of redirects to content script
    Done
End
```

The URL items that have to be sent by the background script are as follows.

```
url: pathItem.url,
status: pathItem.status_line,
redirect_type: pathItem.redirect_type,
redirect_url: pathItem.redirect_url,
meta_timer: pathItem.meta_timer
```

5.2.2 CONTENT SCRIPT

The task to be done by this component is to get the landing URL of the page and the contents of the page from the browser and also the URL redirection list from the background script and then combine those and send it to the background process which will have to do further computation.

```
Begin
    For each page load redirect
        If page is fully loaded
            Get the URL from the tab
            Get the HTML content from innerHTML tag
            Get redirection list from background script
            Send them to the background process
    Done
End
```

The main functionality of getting the URL and also loading the HTML content is done using the following snippets.

```
//Retrieve URL JS
tablink = tab.url;


//Retrieve Page content PHP
$site=$_POST['url'];
$html = file_get_contents($site);
```

5.3 BACKGROUND PROCESS

The background process gets the contents from the content script and identifies if the site is used for phishing or not. It has the following subcomponents which are discussed in detail.

1. Dispatcher
2. Phish Detector
3. Target Identifier

5.3.1 DISPATCHER

The dispatcher is used for the performance enhancements it provides by using the whitelisted addresses that can be used without even having to run the model. It has direct control over the phish detector and target identifier.

```
Begin
    If page address is in whitelist
        Send the GREEN signal
    Else
        Send content to phish detector
        Get results from phish detector
        If phish is FALSE
            Send the GREEN signal
        Else
            Send the RED signal
            Send  content to target identifier
            If target is found
                Publish target
            Else
                No target matched
End
```

5.3.2 PHISH DETECTOR

The phish detector gets the content from the dispatcher and gives the result which is either the site is a phish or not. It is done by using a machine learning model.

```
Begin
    For each page URL
        Get the fuzzy set feature values for the URL
        Load the saved random forest model
        Publish the result
    Done
End
```

The features used are as follows,

1. Have IP address
2. URL length
3. Shortening service
4. Having @ symbol
5. Double slash redirecting
6. Prefix suffix
7. Having sub domain
8. Domain registration length
9. Favicon
10. HTTPS token
11. Request URL
12. URL of anchor
13. Links in tags
14. Server form handler
15. Submitting to email
16. Abnormal URL
17. IFrame redirection
18. Age of domain

19. Web traffic
20. Google index
21. Statistical Reports

The following is used to extract the features using the Fuzzy Rough Set Theory.

```
Begin
    Compute indiscernibility matrix M(A)
    Reduce M using absorption laws
    d - number of non-empty fields
    Initialise all fields
    For all fields
        Compute fields using formulas R=SUT
    Done
End
```

The model is a Random Forest Classifier which has the pseudocode as follows.

```
Begin
    For each record in dataset
        Get the fuzzy set feature values
        Create an arff file to save results
    Done
    Train the dataset with at least 7 splits as random forest
    Save the model as pkl file
End
```

5.3.3 TARGET IDENTIFIER

Once the dispatcher gets the signal that a site is phishing, it can be useful to find which site is being used as a template so that the unsuspecting user if fooled. This is done by using the similarity of hashes between the phishing and target website. The SHA algorithm is as follows.

```
Begin
    Input is an array 8 items long where each item is 32 bits.
    Calculate all the function boxes and store those values.
    Store input, right shifted by 32 bits, into output.
    Store the function boxes.
    Store (Input H + Ch + ( (Wt+Kt) AND 2^31 ) ) AND 2^31 As mod1
    Store (sum1 + mod1) AND 2^31 as mod2
    Store (d + mod2) AND 2^31 into output E
    Store (MA + mod2) AND 2^31 as mod3
    Store (sum0 + mod3) AND 2^31 into output A
    Output is an array 8 items long where each item is 32 bits.
End
```

5.4 WEB BROWSER

Though the above described components play major roles in this project, the one that the user will be able to view is this component. And so care has to be taken to make it look as professional as possible.

5.4.1 HTML CONTENT

The add on must be published as extensions for the browsers and so the UI of the components must be taken care of. And the tasks of the background scripts must run as helper tasks and not interfere with the main script, otherwise the UI of the extension will appear to be jittery.

5.4.2 OUTPUT UI

The two possible results for this project are that either the site is a phish or not. And if a site is not a phish no changes have to be made to notify the user. But if the site is found out to be a phish the changes made to the UI must meaningfully convey to the user that the site is a phish.

```
Begin
    If site is phish
        Change icon to red
        Display warning message
        If site has target
            Display target link
        Else
            Display no target
    Else
        Change icon to green
        Display safe to proceed message
End
```

6 IMPLEMENTATION

This chapter will discuss in detail how this project was implemented along with the key results and snapshots for better understanding.

6.1 ADD ON

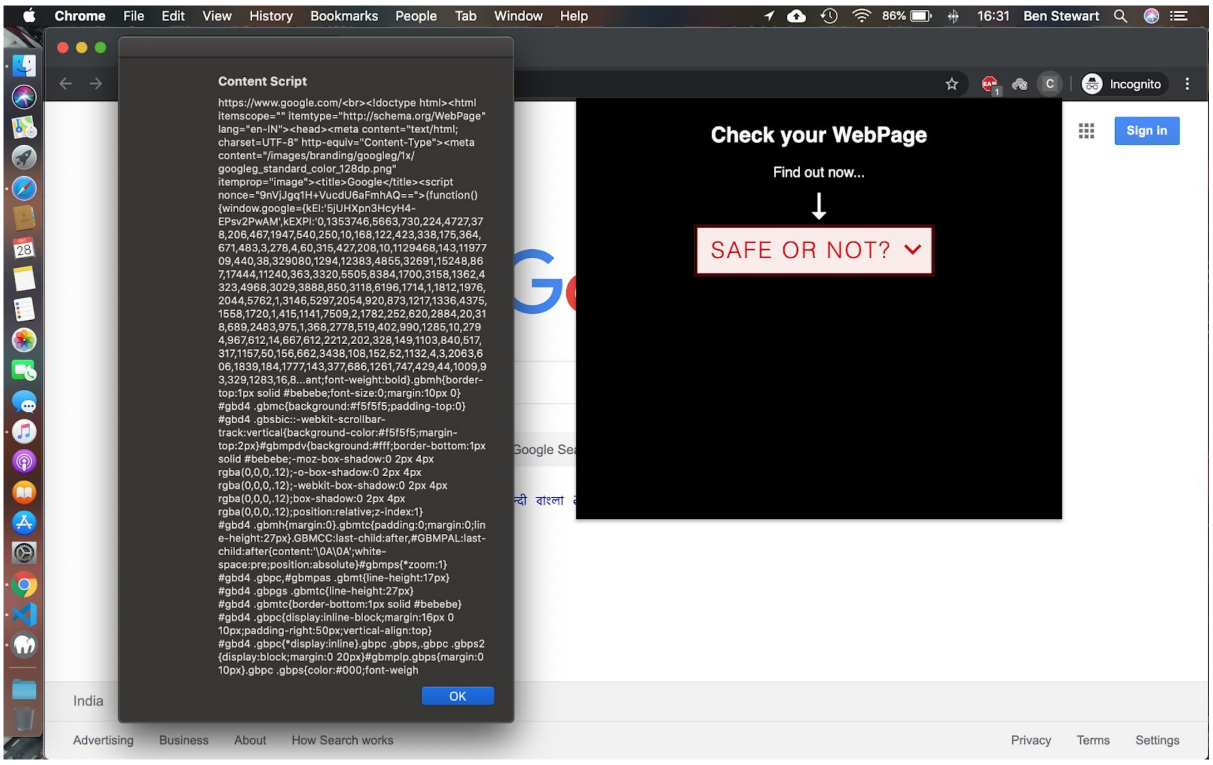
The addon has the following components that were implemented using javascript and php as a Google Chrome extension.

6.1.1 CONTENT SCRIPT

This component takes the input which is the URL of the page and it's contents and sends it to the background process along with the redirection URL from the background script. A design of this component is required as the Chrome Extension cannot get the tab HTML content in the background and so a script in php is used to get the content of the URL from the Javascript code segment.

tablink = tab.url;

$html = file_get_contents($site);



Content Script

```
https://www.google.com/<br><!doctype html><html
itemscope="" itemtype="http://schema.org/WebPage"
lang="en-IN"><head><meta content="text/html;
charset=UTF-8" http-equiv="Content-Type"><meta
content="/images/branding/googleg/1x/
googleg_standard_color_128dp.png"
itemprop="image"><title>Google</title><script
nonce="9nVjJgq1H+VucdU6aFmhAQ==">(function()
{window.google={kEI:'5jUHXpn3HcyH4-
EPsv2PwAM',kEXPI:'0,1353746,5663,730,224,4727,37
8,206,467,1947,540,250,10,168,122,423,338,175,364,
671,483,3,278,4,60,315,427,208,10,1129468,143,11977
09,440,38,329080,1294,12383,4855,32691,15248,86
7,17444,11240,363,3320,5505,8384,1700,3158,1362,4
323,4968,3029,3888,850,3118,6196,1714,1,1812,1976,
2044,5762,1,3146,5297,2054,920,873,1217,1336,4375,
1558,1720,1,415,1141,7509,2,1782,252,620,2884,20,31
8,689,2483,975,1,368,2778,519,402,990,1285,10,279
4,967,612,14,667,612,2212,202,328,149,1103,840,517,
317,1157,50,156,662,3438,108,152,52,1132,4,3,2063,6
06,1839,184,1777,143,377,686,1261,747,429,44,1009,9
3,329,1283,16,8...ant;font-weight:bold}.gbmh{border-
top:1px solid #bebebe;font-size:0;margin:10px 0}
#gbd4 .gbmc{background:#f5f5f5;padding-top:0}
#gbd4 .gbsbic::-webkit-scrollbar-
track:vertical{background-color:#f5f5f5;margin-
top:2px}#gbmpdv{background:#fff;border-bottom:1px
solid #bebebe;-moz-box-shadow:0 2px 4px
rgba(0,0,0,.12);-o-box-shadow:0 2px 4px
rgba(0,0,0,.12);-webkit-box-shadow:0 2px 4px
rgba(0,0,0,.12);box-shadow:0 2px 4px
rgba(0,0,0,.12);position:relative;z-index:1}
#gbd4 .gbmh{margin:0}.gbmtc{padding:0;margin:0;lin
e-height:27px}.GBMCC:last-child:after,#GBMPAL:last-
child:after{content:'\0A\0A';white-
space:pre;position:absolute}#gbmps{*zoom:1}
#gbd4 .gbpc,#gbmpas .gbmt{line-height:17px}
#gbd4 .gbpgs .gbmtc{line-height:27px}
#gbd4 .gbmtc{border-bottom:1px solid #bebebe}
#gbd4 .gbpc{display:inline-block;margin:16px 0
10px;padding-right:50px;vertical-align:top}
#gbd4 .gbpc{*display:inline}.gbpc .gbps,.gbpc .gbps2
{display:block;margin:0 20px}#gbmplp.gbps{margin:0
10px}.gbpc .gbps{color:#000;font-weigh
```

OK

Figure 6.1 Content Script

6.1.2 BACKGROUND SCRIPT

This script is based on the open sourced code on GitHub that demos how to get the background URL redirects of the current tab. It handles multiple types of redirects and also their security levels based on the URL redirects. This component finally returns the list of path components that the page had traversed through. The path item is as follows,

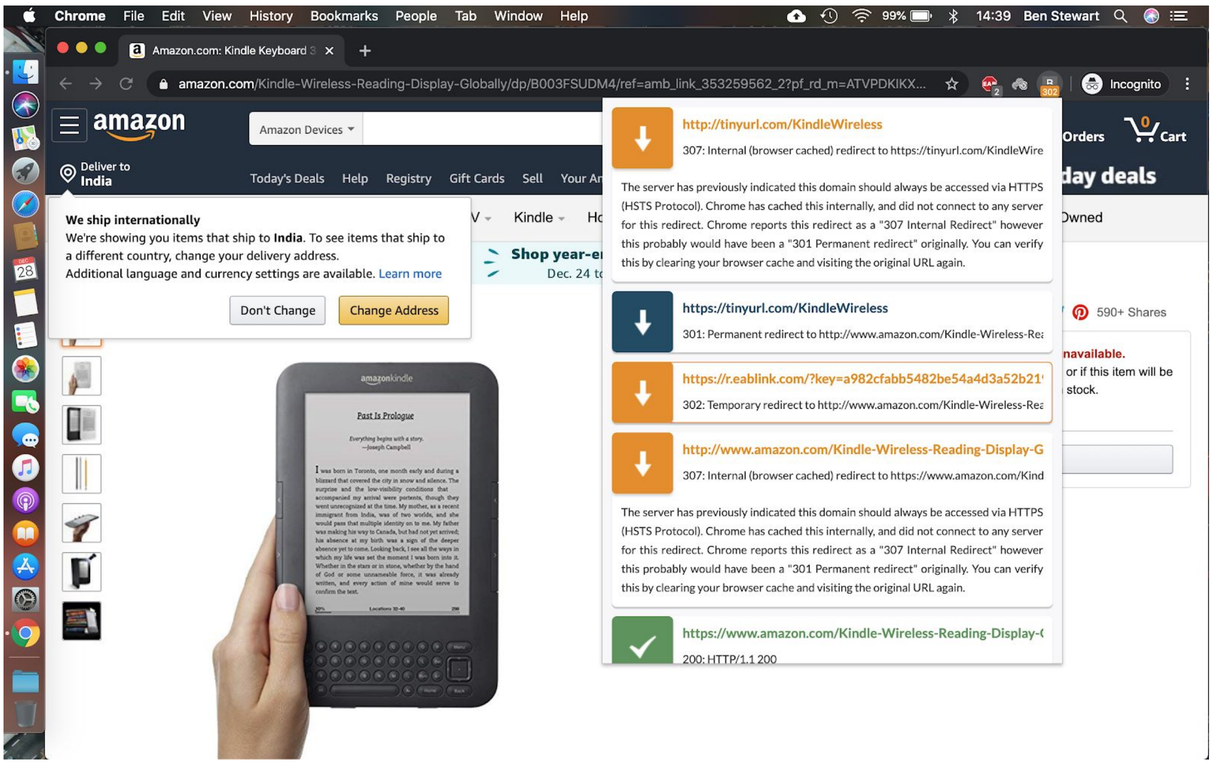
url: pathItem.url,

status: pathItem.status_line,

redirect_type: pathItem.redirect_type,

redirect_url: pathItem.redirect_url,

meta_timer: pathItem.meta_timer



Chrome File Edit View History Bookmarks People Tab Window Help
99% 14:39 Ben Stewart
Amazon.com: Kindle Keyboard
amazon.com/Kindle-Wireless-Reading-Display-Globally/dp/B003FSUDM4/ref=amb_link_353259562_2?pf_rd_m=ATVPDKIKX...
Incognito
amazon
Amazon Devices
Deliver to India
Today's Deals Help Registry Gift Cards Sell
Kindle
Shop year-e
Dec. 24 t
Orders
Cart
day deals
Owned
590+ Shares
navailable.
or if this item will be
stock.
Don't Change
Change Address

We ship internationally
We're showing you items that ship to India. To see items that ship to a different country, change your delivery address.
Additional language and currency settings are available. Learn more

http://tinyurl.com/KindleWireless
307: Internal (browser cached) redirect to https://tinyurl.com/KindleWire

The server has previously indicated this domain should always be accessed via HTTPS (HSTS Protocol). Chrome has cached this internally, and did not connect to any server for this redirect. Chrome reports this redirect as a "307 Internal Redirect" however this probably would have been a "301 Permanent redirect" originally. You can verify this by clearing your browser cache and visiting the original URL again.

https://tinyurl.com/KindleWireless
301: Permanent redirect to http://www.amazon.com/Kindle-Wireless-Rea

https://r.eablink.com/?key=a982cfabb5482be54a4d3a52b21
302: Temporary redirect to http://www.amazon.com/Kindle-Wireless-Rea

http://www.amazon.com/Kindle-Wireless-Reading-Display-G
307: Internal (browser cached) redirect to https://www.amazon.com/Kind

The server has previously indicated this domain should always be accessed via HTTPS (HSTS Protocol). Chrome has cached this internally, and did not connect to any server for this redirect. Chrome reports this redirect as a "307 Internal Redirect" however this probably would have been a "301 Permanent redirect" originally. You can verify this by clearing your browser cache and visiting the original URL again.

https://www.amazon.com/Kindle-Wireless-Reading-Display-
200: HTTP/1.1 200

Figure 6.2 Background Script

6.2 BACKGROUND PROCESS

This component receives the data from the addon and orchestrates the decision making using the dispatcher.

6.2.1 PHISH DETECTOR

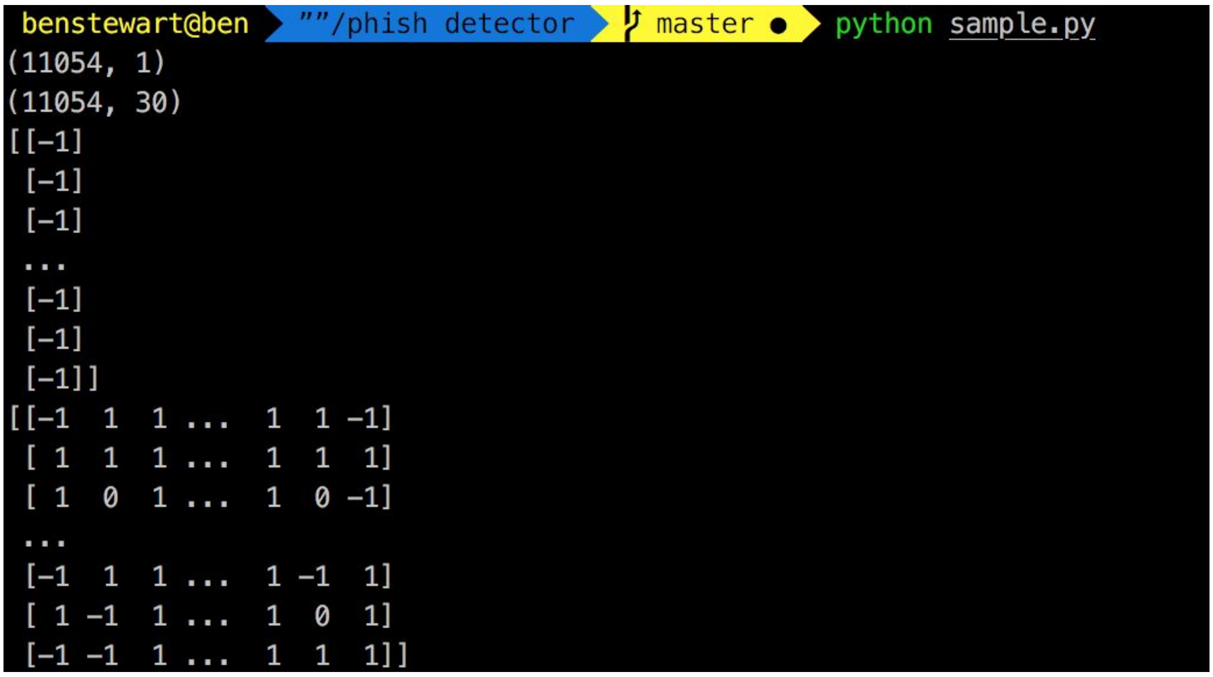
The dataset used for this project was scraped from PhishTank and has the records for 23827 such phishing sites with 30 features for each one of them. As a model using all features will take more time to execute, the most important features have to be extracted.

The phish detector is a random forest model trained using the features derived from the Fuzzy Rough Set Theory for feature selection. A few roadblocks faced while developing this model were that the feature selection kit required the data set to be as integers.

The base component was using the scikit_roughsets package available in Python. The code is as follows and the output features were used to train the random forest model.

selector = RoughSetsSelector()

X_selected = selector.fit(X, y).transform(X)



```
benstewart@ben  ""/phish detector  master ●  python sample.py
(11054, 1)
(11054, 30)
[[-1]
 [-1]
 [-1]
 ...
 [-1]
 [-1]
 [-1]]
[[-1  1  1 ...  1  1 -1]
 [ 1  1  1 ...  1  1  1]
 [ 1  0  1 ...  1  0 -1]
 ...
 [-1  1  1 ...  1 -1  1]
 [ 1 -1  1 ...  1  0  1]
 [-1 -1  1 ...  1  1  1]]
```

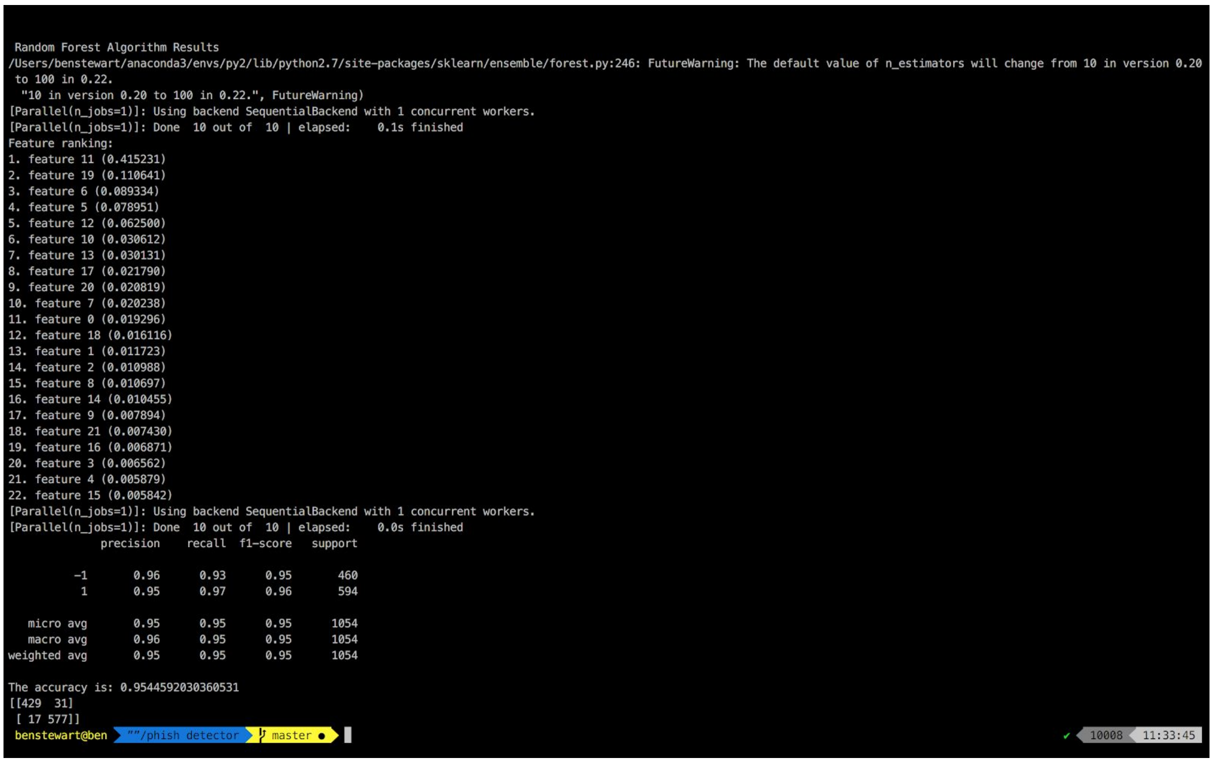
Figure 6.3 Feature Selection

The model was generated using the following code and stored as a pickle in the file.

```
//create model
clf4=RandomForestClassifier(min_samples_split=7)
clf4.fit(features_train, labels_train)
//save the model
joblib.dump(clf4, 'classifier/random_forest.pkl', compress=9)
```

Figure 6.4 gives the confusion matrix and the feature weightage score using the code as follows.

```
//feature weightage
importances = clf4.feature_importances_
//confusion matrix
print metrics.confusion_matrix(labels_test, pred4)
```



```
 Random Forest Algorithm Results
/Users/benstewart/anaconda3/envs/py2/lib/python2.7/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The default value of n_estimators will change from 10 in version 0.20
 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.1s finished
Feature ranking:
1. feature 11 (0.415231)
2. feature 19 (0.110641)
3. feature 6 (0.089334)
4. feature 5 (0.078951)
5. feature 12 (0.062500)
6. feature 10 (0.030612)
7. feature 13 (0.030131)
8. feature 17 (0.021790)
9. feature 20 (0.020819)
10. feature 7 (0.020238)
11. feature 0 (0.019296)
12. feature 18 (0.016116)
13. feature 1 (0.011723)
14. feature 2 (0.010988)
15. feature 8 (0.010697)
16. feature 14 (0.010455)
17. feature 9 (0.007894)
18. feature 21 (0.007430)
19. feature 16 (0.006871)
20. feature 3 (0.006562)
21. feature 4 (0.005879)
22. feature 15 (0.005842)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.0s finished
              precision    recall  f1-score   support

          -1       0.96      0.93      0.95       460
           1       0.95      0.97      0.96       594

   micro avg       0.95      0.95      0.95      1054
   macro avg       0.96      0.95      0.95      1054
weighted avg       0.95      0.95      0.95      1054

The accuracy is: 0.9544592030360531
[[429  31]
 [ 17 577]]
 benstewart@ben  ""/phish detector  master ●                                                                 10008  11:33:45
```

Figure 6.4 Random Forest Model

6.2.2 TARGET IDENTIFIER

The target identifier uses the following code to find the similarity between two web pages using the following code.

```
tags1 = get_tags(lxml.html.parse(path1))
tags2 = get_tags(lxml.html.parse(path2))
diff = difflib.SequenceMatcher()
diff.set_seq1(tags1)
 diff.set_seq2(tags2)
```

The dispatcher once confirms that the site is a phish from the phish detector and then writes the url into the single-sites.json file. The url is taken from there and scraped along with the html content and is then compared using the above method.

```
params['url'] = url
response = requests.get(url, headers=headers, params=params)
```

6.2.3 AUTO UPDATED WHITELIST

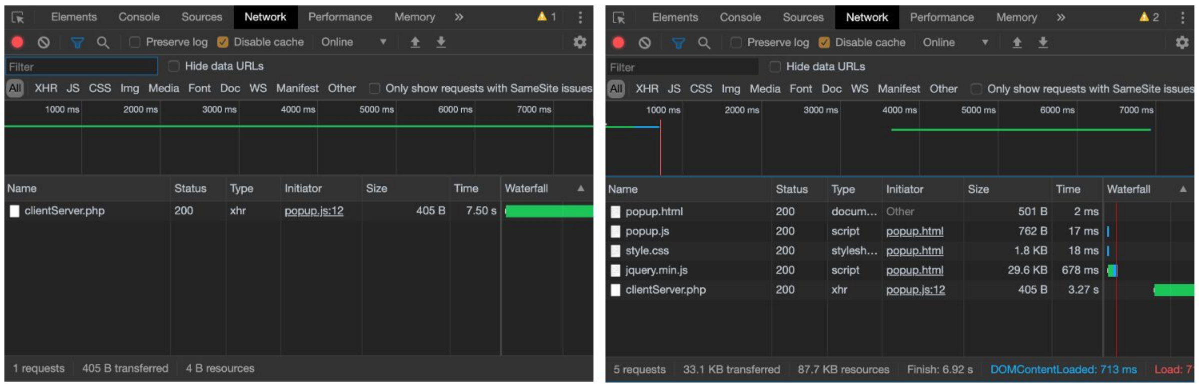
The whitelist is automatically updated by the dispatcher if the url is detected to be safe using the following code.

```
//insert into whitelist
white_list_file=open('whitelist.txt', "a+")
white_list_file.write(url)
```

The whitelist is searched for before using the ML model to detect the site is phishing or not, to save execution time.

```
white_list_file = open('whitelist.txt').read()
white_list = white_list_file.split('\n')
if url in white_list:
#url is safe
```

Figure 6.5 gives the shortening of execution time as the url is stored in the whitelist.


Elements
Console
Sources
Network
Performance
Memory
Preserve log
Disable cache
Online
Filter
Hide data URLs
All XHR JS CSS Img Media Font Doc WS Manifest Other
Only show requests with SameSite issues
1000 ms 2000 ms 3000 ms 4000 ms 5000 ms 6000 ms 7000 ms

Name	Status	Type	Initiator	Size	Time	Waterfall
clientServer.php	200	xhr	popup.js:12	405 B	7.50 s	

1 requests | 405 B transferred | 4 B resources

Name	Status	Type	Initiator	Size	Time	Waterfall
popup.html	200	docum...	Other	501 B	2 ms	
popup.js	200	script	popup.html	762 B	17 ms	
style.css	200	stylesh...	popup.html	1.8 KB	18 ms	
jquery.min.js	200	script	popup.html	29.6 KB	678 ms	
clientServer.php	200	xhr	popup.js:12	405 B	3.27 s	

5 requests | 33.1 KB transferred | 87.7 KB resources | Finish: 6.92 s | DOMContentLoaded: 713 ms | Load: 7

Figure 6.5 Execution Time with Auto Updated Whitelist

6.2.4 DISPATCHER

The dispatcher is called using the php script of the content script using the following statement.

$decision=exec("python test.py $site 2>&1 ");

echo $decision;

The command is run in the default shell of the machine and the output is passed to the Output UI of the addon that reflects the same in the web browser.

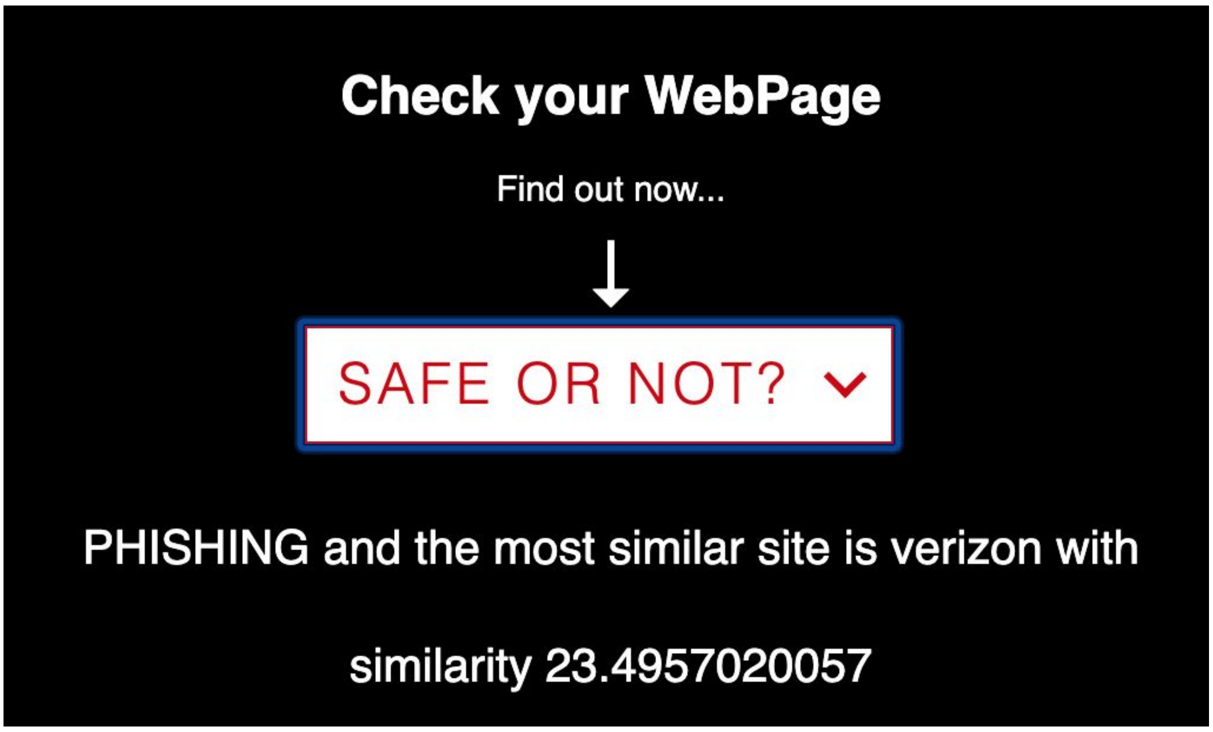
6.3 WEB BROWSER

This component is what takes care of how and what is displayed to the end user when the application is being used.

6.3.1 OUTPUT UI

The content from the dispatcher is loaded into the Google Chrome Extension using the Output UI which has the content loaded into the div that has the HTML, JS and CSS designed for it upfront.

$("#div1").text(xhr.responseText);



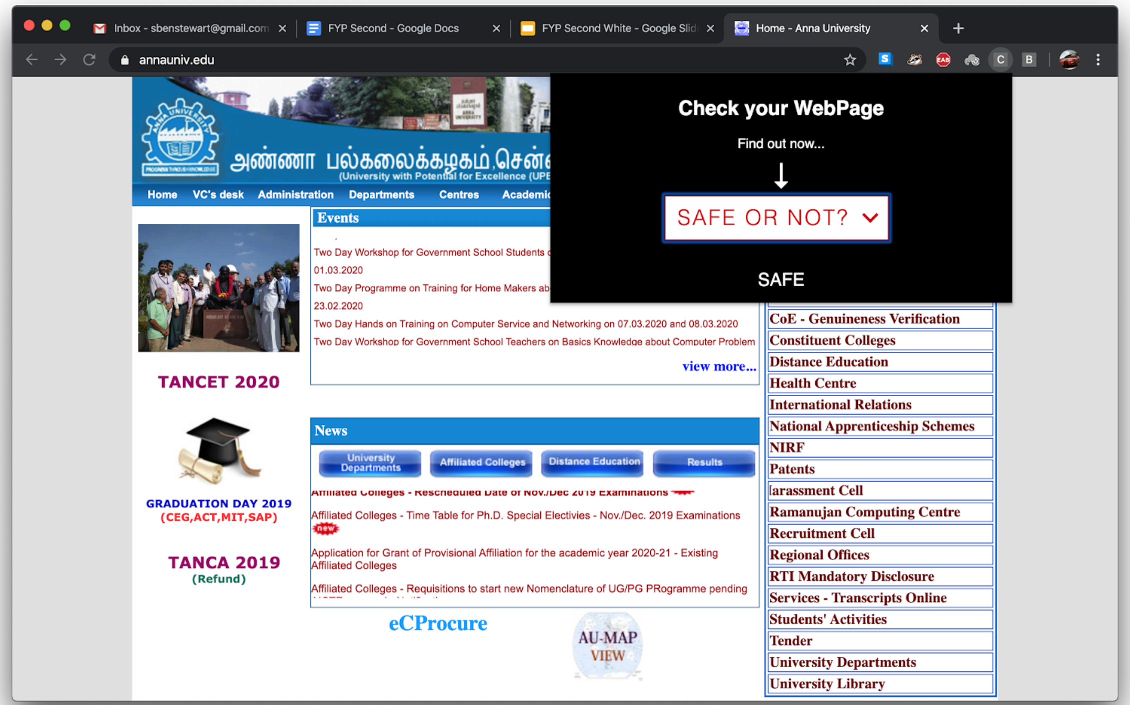
PHISHING and the most similar site is verizon with similarity 23.4957020057

Figure 6.6 Output UI

6.3.2 HTML CONTENT

The web browser must also take care of the whole extension so that the components do not get hidden and also are not visible to the user. The following css snippet makes sure the above holds good.

```
body{
  width:500px;
  height:100px;
  display:inline-block;
  align-items: center;
  }
```



Two Day Workshop for Government School Students o
01.03.2020
Two Day Programme on Training for Home Makers ab
23.02.2020
Two Day Hands on Training on Computer Service and Networking on 07.03.2020 and 08.03.2020
Two Dav Workshop for Government School Teachers on Basics Knowledge about Computer Problem

Affiliated Colleges - Rescheduled Date of Nov./Dec 2019 Examinations
Affiliated Colleges - Time Table for Ph.D. Special Electives - Nov./Dec. 2019 Examinations
Application for Grant of Provisional Affiliation for the academic year 2020-21 - Existing Affiliated Colleges
Affiliated Colleges - Requisitions to start new Nomenclature of UG/PG PRogramme pending

Figure 6.7 HTML Content

7 EVALUATION METRICS

This application must be evaluated for the following performance metrics, which become important as this is meant for users who will never need to know about computer programming.

Phish detection accuracy: This metric is really important because the main task for this application is to notify the people if the site is a phish or not. It is defined as the ratio of the total number of correct classifications to the total number of classifications.

$$Accuracy = (TP+TN)/(TP+TN+FP+FN)$$

Target detection ratio: This metric is to measure the ease of use for the user by providing them with the original site which is being mimicked by the phish. This will be the ratio of phish sites whose target has been found to that of the total number of phish sites.

$$Detection\ Ratio = (TP)/(TP+TN+FP+FN)$$

Memory usage profiling: Since this is an application to be used by many people who might have different configurations of machines, we must take into consideration that the application must use as little memory as possible.

Current total memory usage = Total Memory - (Free + Buffers + Cached)

Addon rendering time: Since the addon has a background component which has to collect the data from multiple tabs that could be running simultaneously, the addon must be tested to check if it is stable and does not take time to render and thereby blink.

Rendering Time = End time - Start time

Temporal resilience accuracy: This is a metric which says that the application must be resilient to adaptations by the malicious actor, over time. Thus this can be measured by checking if the accuracy does not reduce with the passage of time.

Accuracy = (TP+TN)/(TP+TN+FP+FN)

REFERENCES

[1] Mahdieh Zabihimayvan and Derek Doran, "Fuzzy Rough Set Feature Selection to Enhance Phishing Attack Detection", *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, June 2019.

[2] S. Marchal, G. Armano, T. Gröndahl, K. Saari, N. Singh, N. Asokan, "Off-the-hook: An efficient and usable client-side phishing prevention application", *IEEE Trans. Comput.*, vol. 66, no. 10, pp. 1717-1733, Oct. 2017.

[3] A. K. Jain, B. B. Gupta, "A novel approach to protect against phishing attacks at client side using auto-updated white-list", *EURASIP J. Inf. Secur.*, vol. 2016, no. 1, Dec. 2016.

[4] G. Xiang, J. Hong, C. P. Rosé, L. Cranor, "CANTINA: A feature-rich machine learning framework for detecting phishing Web sites", *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 2, 2011.

[5] *Implementation for the Usage of Google Safe Browsing APIs (v4)*, 2019, [online] Available: https://github.com/google/safebrowsing.

[6] C. Whittaker, B. Ryner, and M. Nazif, “Large-scale automatic classification of phishing pages,” in Proc. Netw. Distrib. Syst. Security Symp., 2010, pp. 1–14.