APPROCHES FOR DETECTION OF VULNERABILITIES IN WEBSITES

Renuka Kondabala[1*], Dr.S. Balaji [2]

[1*]Research scholar, *Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Hyderabad 500075, Telangana, India*
Assistant Professor, Department of Information Technology, Valluripalli Nageshwara Rao Vignana Jyothi Institute of Engineering and Technology, Bachupally, Hyderabad,Telangana-500090,India.
[2] *Professor, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Hyderabad 500075, Telangana, India*

Abstract— Throughout the world, cyber security attacks continue to evolve and gain momentum. There are mainly at three places, hackers try to attack a website. Cross-Site-Scripting also called as XSS is vulnerability exists for the Websites, allows an attacker to compromise the communications with a vulnerable application when users try to make some interaction. SQL injection is a vulnerability that allows an attacker to interfere with the queries that an application makes to its database in a website. Phishing one of the Social Engineering Attacks often used to steal user's personal data, including login credentials like password, and Credit card numbers and etc. In this paper, we were able to automate the traditional and manual SQL and XSS detection approach with higher accuracy, and a robot random forest approach along with a dynamic feature extractor is used to differentiate between legitimate and phishing websites. Phishing, Payload SQL injection, XSS attacks are one of the major forms of cyberattack for criminals to carry out. In the month of March 2020 alone over 60,000 phishing websites were reported. In the year 2019 nearly 22% of breaches were involved with phishing. Organizations in India lost Rs 14cr on average to data breaches in August 2019- April 2020. There is an increase of more than 121 percent of cybercrimes in India since 2016. India has lost nearly Rs 1.24 trillion in past 12 months due to increase in cybercrime. These statistics have proven that the existing vulnerability assessment solutions and efforts are not truly effective.

Keywords— Payload SQL Injection, Cross Site Scripting, Phishing, Random Forest Algorithm, Vulnerability, Automation.

1. INTRODUCTION

Over the past years, we have seen enormous growth in internet connectivity and usage. The internet is everywhere, and it has become the main aspect for all types of users ranging from building communication between two common people to sharing highly confidential information between two countries. The Internet has become just like the five elements which are essential for human survival. Even in the financial sectors internet plays a vital role. The best way to maximize the online presence on the internet is through websites. The website had become the main expansion marketing strategy for business. Even for a small-scale business, there is a website. The reason for this is that building a basic static website without any advanced security features takes less time and cost.

Even though the internet has so many upsides there is always a risk of privacy and data safety. These days the internet is embedded into many devices increasing the risk of data privacy. Since Internet has been updating for the past decade and developers always strive to make their websites look visually more appealing rather than improving their security standards in the fear of poor performance of websites. This leads to unethical coding practices and hackers always try to take advantage of these websites and capitalize on them through stealing vital information.

The point present in a website where hackers enter to attack and break the website is generally called a vulnerability. Vulnerabilities have been a key discussion point in most of the web-application development and security discussion forums as the number of applications on web are increased. Web applications normally let the immediate and repeated usage of sensitive consumer data be captured, processed, stored, and transmitted. Thus, web applications have become

important targets for attackers benefiting from inadequate coding techniques by web developers, gaps in an application code, inadequate authorization of user's inputs, or failure by software development developers to comply with security requirements. These vulnerabilities keep remain on Server as well as on Client. These vulnerabilities are classified as Cross-site Scripting, Phishing, SQL Injection, Cross-Site Request Forgery, Broken authentication, etc.
This paper deals with the client-side vulnerabilities (Cross-Site Scripting), server-side (SQL injection), social Engineering (Phishing).

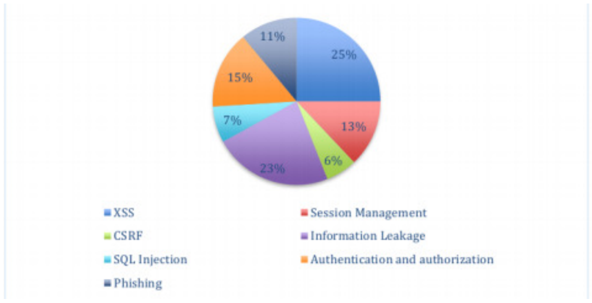


Fig.1: Pie chart of percentage of different cyber-attacks.

Cross-Site scripting is a vulnerability used by an attacker to jeopardize user engagement with a vulnerable application that exists for the websites. It usually allows an attacker to disguise as a victim, perform any kid of actions that a user can carry out, and view any data that is available. It also allows an attacker to bypass the same strategy of roots, is proposed to separate different websites. If the user has unrestricted permissions, the attacker can acquire full control over all the features and the information of the web application. It works by handling vulnerable websites so that JavaScript outputs malicious to users. The attacker can threaten their ability to interact with the application when a mischievous code runs inside the client's webpage. In a webpage, if the user present on the client-side provides a simple JavaScript code as an input on any one of the forms present in the webpage. If the website executes the JavaScript code as a server input instead of client input (as a String) then that webpage has a cross-site scripting vulnerability. In any Cross-site vulnerable websites, the hacker incorporates mischievous JavaScript code which will leak the confidential data of the organization. In general, Cross-site scripting vulnerabilities are categorized into three types. They are reflected XSS, Stored XSS, and DOM-based XSS. This paper mainly concentrates on Reflected and DOM based Cross-Site vulnerabilities. A Reflected XSS is also called as Non-Persistent XSS attack. It jumps off another website using a malicious script in the browser of a victim. The query is usually passed through the URL. The DOM Based XSS simply means that vulnerability appears in the Document Object Model instead of part of the HTML.
SQl is a structured data and machine learning algorithms can be easily applied on the data.All users of the business can use this data without any problems and this can be easily accessable to more number of tools.SQL injection is a type of website susceptibility allowing a hacker to hamper the queries made into its database by an application. It usually allows a hacker to view data which they usually cannot recover, that includes data from other users, or any other accessible to the application itself. In several cases, this data can be modified or deleted by an attacker which causes persistent changes in the content or behavior of the application. In certain scenarios, the hacker tries to break the underlying base server and perform a DOS attack. If a hacker tries to successfully perform an attack by SQL injection may lead to illegitimate access to confidential data like passwords, credit card details, or user personal data. Many well-known data breaches in subsequent years have resulted in reputational impairment and regulatory penalties caused by SQL injection attacks. In some cases, an attacker can get a permanent backdoor into the systems of an organization, which leads to a long-term compromise, which can remain unnoticed for a longer period. If the website executes the query code or special characters like a quote (') as a server input instead of client input (as a String) then that particular webpage has a SQL injection

vulnerability. In any SQL injection vulnerable websites, the hacker incorporates mischievous query code which will leak the confidential data and gains the credentials of an administrator in an organization.

Phishing is a kind of Social Engineering attack, often used for stealing user personal information, including bank details and authentication tokens. It occurs when an attacker conceals an illegitimate website into a trustworthy entity. The recipient is tried to click on a Malicious connection link that could start a malware to get installed, the system starts freezing as part of a Ransomware attack, or sensitive information being disclosed. An assault can be devastating. This includes unauthorized shopping, fund stealing, or identification of theft for individuals. In addition, Phishing also ran as a part of major attack, as in an "advanced persistent threat (APT)", to gather a foothold in private or government sectors. Therefore, in latter circumstances, employees will be compromised to sneak past security limits, install malicious objects in closed environments or access privileged confidential information. A company that gone through such attacks suffers from decreasing market share, credibility, and consumer confidence, which typically affects severe revenue damage. Depending on the scope, a phishing attempt may turn into a safety incident where a company has a hard time recuperating. Examples of phishing include hackers create a replica of a legitimate webpage and send it to various clients hoping the client might enter confidential information. since phishing is mostly dependent on the user's way of thinking and awareness of the concepts of cybersecurity. So, various machine learning algorithms are used to detect whether a webpage is considered phishing or not. This project uses the most popular random forest algorithm which provides high accuracy with less amount of time.

2. RELATED WORK

Vulnerabilities on the internet have become a major problem. The detection of vulnerabilities is more important and the Internet more securely available to users. Many researchers have suggested several models for the detection of these vulnerabilities.

“Waleed Ali and Sharaf Malebary [1], (Member, IEEE), proposes to improve the detection of phishing websites using particle swarm-based optimization weighting. The approach was proposed to use particle swarm optimization techniques to effectively weigh various features of the websites to ensure greater accuracy when phishing is being detected. The proposed PSO weighting is used to distinguish between different features in websites, based on the importance of phishing on legitimate websites.

"Rim Akrout, Eric Alata, Mohamed Kaaniche, and Vincent Nicomette[2]. An automated black-box approach for web vulnerability identification and attack scenario generation." This paper introduced a new web page clustering techniques methodology which is designed to identify web application vulnerability after a black-box analysis. The proposed approach highlights various potential scenarios of attack, including the development of several consecutive vulnerabilities, taking explicit account of their dependencies. This paper particularly focuses on vulnerabilities in code injection, like SQL injections.

"Yazan Ahmad Alsariera, Victor Elijah Adeyemo, Abdullateef Oluwagbemiga Balogun (member, IEEE), and Ammar Kareem Alazzawi[3]. AI Meta-Learners and Extra-Trees Algorithm for the Detection of Phishing Websites". In this paper, they proposed a model to outperform current ML-based models to detect phishing attacks. They have proposed four (4) different models (AdaBoost-Extra Tree (ABET), the Bagging–Extra Tree (BET), The Forest Rotation – Extra Tree (RoFBET), and the Logitboost-Extra Tree (LBET) . They have reached an accuracy of detection at 97% and drastically a Low False-Positive rate not above 0.028.

"Bakare K. Ayeni, Junaidu B. Sahalu, and Kolawole R. Adeyanju [4]. Detecting Cross-Site Scripting in Web Applications Using Fuzzy Inference System". This paper describes a smart tool for the detection of web application cross-site script flaws. It describes the way the XSS classical weaknesses can be detected, and some experimental results can be obtained. The accuracy improvement was 15 percent and the false positive rate decreased by 0.01 percent.

"Mohith Gowda, Adithya, Ganesh Prasad and Vinay [5]. Development of anti-phishing browser based on random forest and rule of extraction framework". This paper proposes a new technique for the customer to easily identify phishing websites by proposing a new browser architecture. They have used the extraction rule to extract a website's properties using the URL alone. The model was trained using a data-set of 11,055 tuples. The processes are performed with a redesigned browser architecture on the customer side. These tools have been improvised and introduced detection methods into your browser architecture called the 'Embedded Phishing Detection Browser' to determine that they can be accessible to every individual.

"Stephen W. Boyd, Angelos D. Keromytis, SQLrand: Preventing SQL Injection Attacks [6]". In this paper, a practical mechanism for protection against SQL injection attacks was presented. They used the concept of randomizing instructions set to SQL, Creating language instances that the attacker cannot foresee. They showed how the MySQL database is used with an intermediate proxy which translates the random SQL into their standard language.

"Z. Dou, I. Khalil, A. Khreishah, A. Al-Fuqaha and M. Guizani[7]" Phishing is a type of cyberattack which uses various social engineering approaches and other well known methods to yield the personal and secret information from all the website users .As per the report given by Anti Phishing workgroup on an average every year there is growth of websites which were affected by phishing for the past 6 years are 36.29% and from past 2 years 97.36% . Due to this reason awareness was created among the cyber security community. There is an development and rigorous research have been conducted to detect and analyze the attempts made on phishing to collect the content which is unique, on network, and characteristics of URL . In this paper they have shown a systematic study of schemes to detect phishing, especially which are based on software.

2.1 Existing system :

The existing phishing system takes long time (6-9 hours) to provide high accuracy phishing detection. This time is sufficient for the hackers to gain access to sensitive data of the victim. The existing browsers do not provide any protection against vulnerabilities (SQL injection and Cross Site Scripting) on client and server side. To be protected against these vulnerabilities we need to buy expensive software for each vulnerability. There is no existing system in Windows and Mac OS that can protect against all three vulnerabilities in one software.

2.2 Proposed system:

The proposed system can detect phishing (social engineering), Payload SQL injection vulnerability (server side) and Reflected XSS vulnerability (client side) using a single software. The proposed system can detect the vulnerabilities in under a minute with high accuracy. The proposed system is platform independent.

2.3 Objective:

The main objective of this paper is to develop a platform-independent, user-friendly, GUI-based application that can detect social engineering, payload SQL injection, and reflect cross-site vulnerabilities using different algorithms. This application will be visually aesthetic. The result will be a simple GUI, which will be able to detect vulnerabilities with high precision in a short amount of time. The user does not need to know the packages or technologies that were used to build this application beforehand. The customer just needs to know what type of vulnerabilities this application detects. The user provides the GUI with the URL, which indicates that the application detects vulnerabilities to the provided URL of a website.

Issues and Challenges:

The selection of appropriate features to provide high accuracy is a difficult task out of the multiple possible detection functions for Phishing. We have faced rare cases where legitimated websites, such as Quora, use multiple hyphens in their

URLs even after selecting vital features. This resulted in a reduction in the model's confidence and reduced feature accuracy.

Phishing methods are continuously evolving. There is a daily creation of thousands of new phishing websites which make maintaining a structured database of phishing and legitimate websites difficult. We were faced with the challenge of obtaining a structured model training dataset.

We used Beautiful Soup in this paper to manage all URLs and forms contained in it. The main challenge was that all websites which did not have the correct URL format and incorrect URLs were ignored. These exceptions are explicitly handled using the try-except blocks. Now, whenever an improper URL is provided in GUI it prompts the user to enter the correct URL.

The output was printed in the console instead of the GUI window after the development of a GUI. We rectified the issue by adding return statements to every function. While developing the GUI code we had given an object name for each object (button, label, input text). We could not achieve the desired result when we tried to add more visual elements, such as background colors and icons. We assigned the class a name to overcome this problem.

As we made a single GUI to detect all three vulnerabilities, we faced issues while integrating the three functionalities into a single file. The GUI either crashed or provided the wrong output. We rectified this by assigning temporary variables to the output of each function. We used these temporary variables in ordered to give the desired outputs in the end.

3. METHODOLOGY

3.1 Introduction:

The Internet has grown significantly in modern times. There is a website for tiny businesses to corporations of several millions of dollars. As the number of websites increases day by day, developers could not construct an adequate website that includes all the security aspects. Time limitations or no prior understanding of cyber security can be the factors. This leads to the development of vulnerable web applications. Hackers are trying to take advantage of it. This causes confidential information to be leaked that affects the organization. As Information Technology students, we always look forward to resolving problems with the technology we learn and to find out how it can be applied to make real-life use easy. Here we invented the idea of constructing an easy-to-use application that can accurately detect the three most important vulnerabilities of a Web app. Consider an application in which everything you need to do is enter a web URL to recognize the web application's vulnerabilities. There is no need to go through the data breach phobia. It facilitates the user experience. Our paper can help build an internet user's safe personal environment.

3.2 Proposed Approach and Flow diagram:

Our paper consists of three modules, for each vulnerability we have created a module. For both SQL Injection and Cross-Site Scripting detection modules, we have used the Beautiful Soup package to pull out all the existing forms that are present in the URL. In the SQL injection module, for every empty form that presents in the URL, the code incorporates query. If the URL throws any error, it considers it as a vulnerability. Similarly, in the XSS Module instead of the query, the code incorporates client-side executable JavaScript code. For the phishing module, we created a database of 2000 values and split the dataset into two parts (training and testing). We have chosen certain feature extractions which help us in increasing the accuracy of detection. For each feature, we have created a new column in the existing dataset and these features act as decision trees in building the random forest model. As we know that random forest algorithm takes all these decision trees (extraction features) and merges them to get more accurate and stable results. We used Jupyter Notebook, Matplotlib, and NumPy packages for building the random forest model and incorporated this trained model in the GUI which is created in Spyder. The user just needs to run the GUI module and should have to provide the URL as an input and in the backend, all these modules run and provide the user with an output in the GUI.

Algorithm:

Step 1: Start.

Step 2: Targeted URL is given as an input to the GUI.

Step 3: In SQL Injection module, targeted URL is considered as argument.

Step 4: If forms are present on the website, it scrapes all the forms and stores in temporary variable. Predefined SQL errors are stored in another variable explicitly.

Step 5: Dumping query code (‘) in all the forms and all the buttons are clicked on the website. If forms are not present, then query code is injected into URL.

Step 6: If the website throws an error which are present in the predefined SQL error variable, it is considered as vulnerable else not vulnerable and output it stored in a new variable.

Step 7: In XSS module, targeted URL is considered as argument.

Step 8: Scraping of all the forms and stored in a temporary variable. Executable JavaScript is stored in another variable explicitly.

Step 9: Dumping the JavaScript stored explicitly into the forms and submit buttons on the website are clicked.

Step 10: If the website executes the JavaScript code and shows an output then, it is considered as vulnerable else not vulnerable, and output is stored in a new variable.

Step 11: In Phishing module, targeted URL is considered as argument.

Step 12: Predefined extraction features are applied, and each output is stored in different variables. These outputs are considered as decision trees.

Step 13: These decision trees are feed to a Random Forest classifier model which is created by using a dataset. This model gives an output, and it is stored in a new variable.

Step 14: The output of all the modules is shown in the output box of the GUI.

Step 15: Stop.

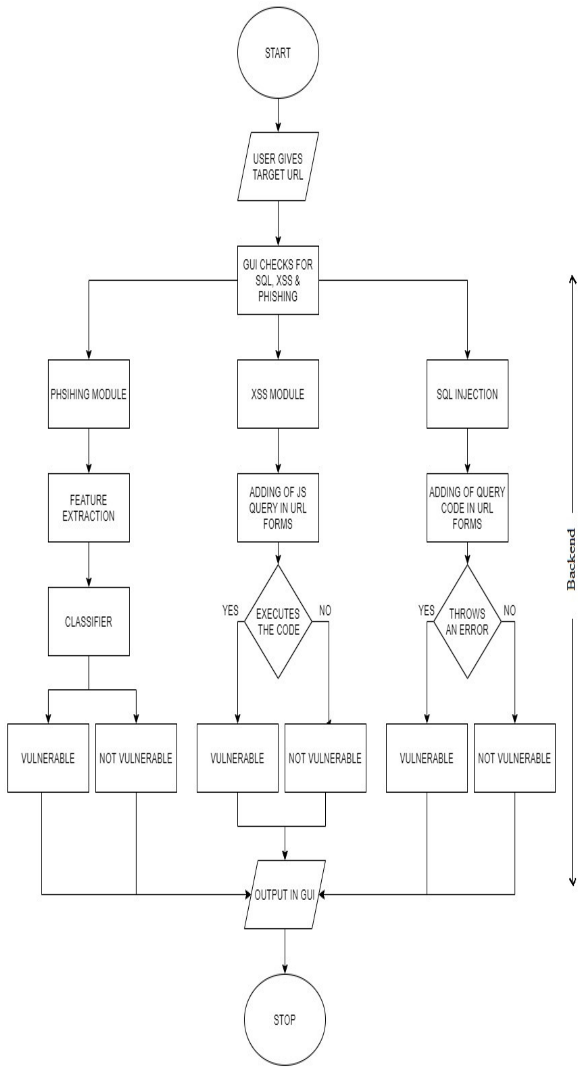


Fig 3.1: Flow diagram

4 .IMPLEMENTATION

4.1 Detection of SQL, XSS and Phishing vulnerabilities

In web applications, we have been able to automate the traditional testing and vulnerability detection method. It helps to overcome the human error problem and time limitations. We have taken and combined the most common vulnerabilities in three various sectors into one single application.

Four modules are involved :

1. Detection of SQL Injection (Server side)
2. Detection of XXS vulnerability (Client side)
3. Detection of Phishing vulnerability (Social engineering)
4. Building a Graphical Interface (GUI).

4.1.1 Detection of SQL Injection (Server side)

SQL injection arises when an application obtains some input from the mischievous user, uses it as an element of SQL comment to query the database. An attacker can inject SQL control-characters, Command keywords like single quote (‘), double quote (“), equal (=), comment (- -), etc., to modify the format of a query. Using these common control-characters with basic SQL commands like SELECT, FROM, DELETE, etc., facilitates the entry or retrieval of data elements from a database server. By attaching a malicious code in the application of a sql statement, a Successful attack is executed by an attacker. When malicious SQL-statements are executed against a database, an attacker can modify or access the database from blackened.

We have installed packages for extracting URL forms before starting this module. The packages Beautiful Soup, URL, Request and print have been used. We need to remove forms and text boxes from the web application for SQL Injection detection. So, we have written a function that automatically receives an input and gives extracted forms as output. We used the Beautiful Soup library to extract all form tags from HTML. This extraction takes place in the function get all forms (). A single form tag object argument is added in the other function getting form details () and only analyses applicable information. Those are mainly the target URL, GET, POST, and input field attributes.

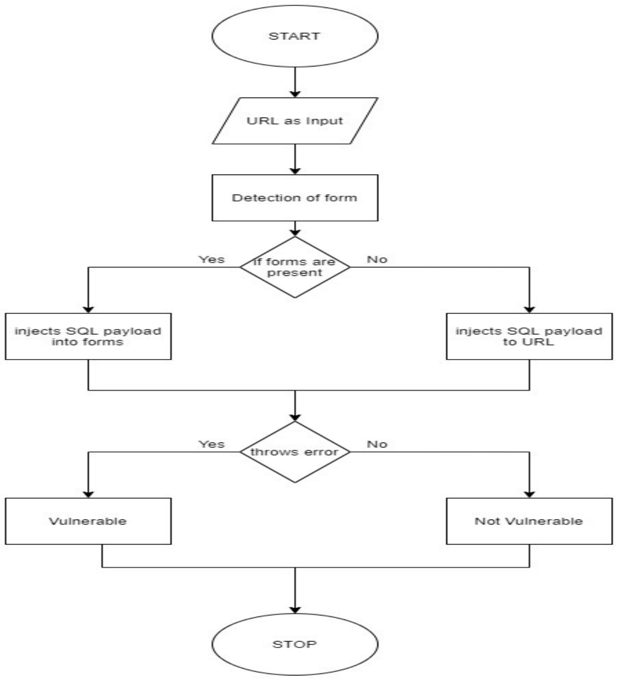
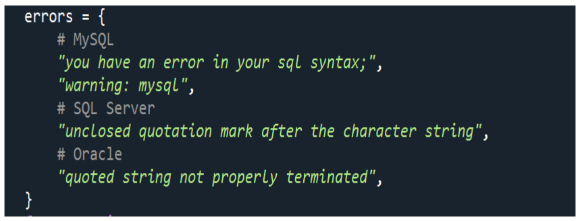


Fig 4.1.1: Flow Diagram of SQL Injection.

.



```
errors = {
    # MySQL
    "you have an error in your sql syntax;",
    "warning: mysql",
    # SQL Server
    "unclosed quotation mark after the character string",
    # Oracle
    "quoted string not properly terminated",
}
```

Fig 4.1.2. List of errors that are thrown by SQL vulnerable websites

We explicitly added all the error that occurs in SQL vulnerable websites. Now a new function is vulnerable (URL) added which takes URL as an argument and scrapes all the forms from the website and stores them in a temporary variable. It pushes the SQL payloads into all the forms using form details () function and reads the error thrown by the website. Then it compares the error with the explicitly stored errors. If both error statements match, then it is considered as SQL Injection vulnerable. But few websites do not have any forms but still, they are vulnerable to SQL Injection. For such kinds of URLs, this same function adds payload "single quote (')" at the end of the URL. Then they may throw the same errors and comparing the errors it detects the SQL Injection vulnerability.

4.2. Detection of XSS vulnerability (Client side)

An XSS vulnerability occurs if web applications take user data and dynamically add this data to websites without evaluating the information properly before. An attacker may execute random commands and random contents in the browser of the victim by using XSS vulnerabilities. An attacker manipulating the browser, or the web application of the victim is the result of a successful XSS attack. While XSS is being enabled through vulnerable web-based pages, users will become the victims of an XSS attack, not only the application itself. The potential for an XSS vulnerability is because in the context of the victim session, the malicious code executes allowing the attacker to bypass normal security limitations.

We have installed packages for extracting URL forms before starting this module. The packages BeautifulSoup, URL, Request and print have been used. We need to remove forms and text boxes from the web application for XSS vulnerability. So, we have written a function that automatically receives an input and gives extracted forms as output. We used the BeautifulSoup library to extract all form tags from HTML. This extraction takes place in the function get all forms (). A single form tag object argument is added in the other function getting form details () and only analyses applicable information. Those are mainly the target URL, GET, POST, and input field attributes.

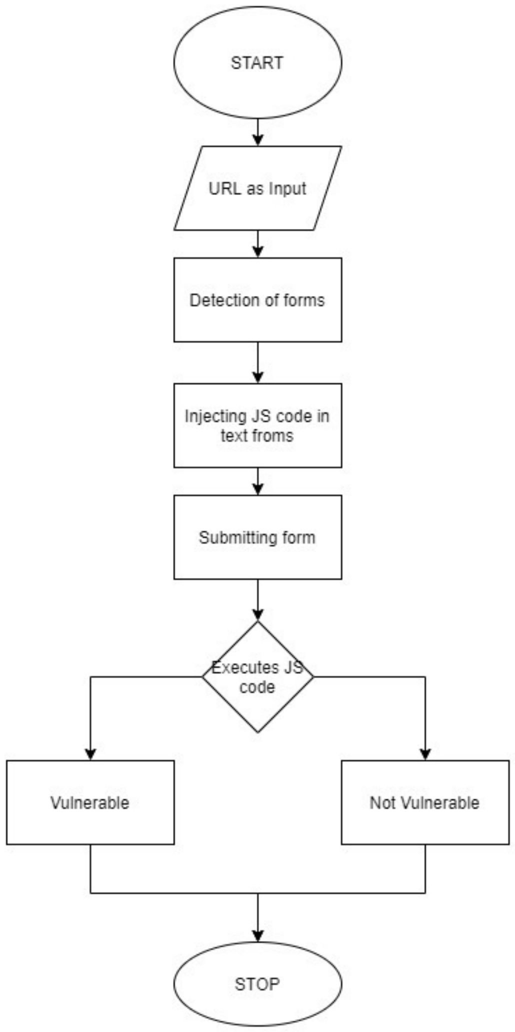
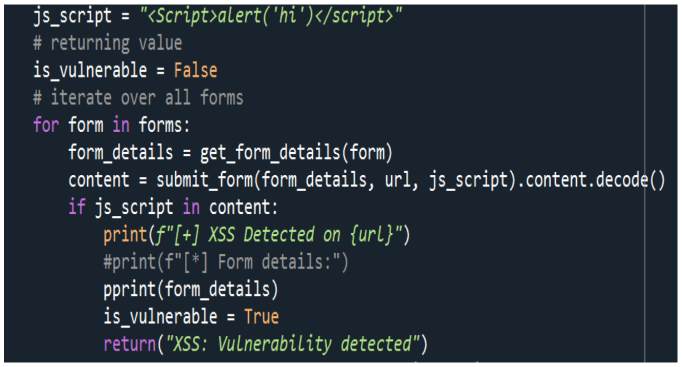


Fig 4.2: Flow chart of XSS vulnerability.

The function get form details collects all the details of forms (text, search, etc.) from the website. After gathering all the information based on the website structure the submission takes place either in the get or post method. The main function collects all HTML forms and then prints the detected range. It then evaluates all the forms, presents the forms with the

JavaScript code for the value of all text and search fields. The injection of Java-Script code is done by using the above get form details () function and submits the JavaScript code. If the injected JavaScript code is executed successfully and then it will be considered as vulnerable.



```
js_script = "<Script>alert('hi')</script>"
# returning value
is_vulnerable = False
# iterate over all forms
for form in forms:
    form_details = get_form_details(form)
    content = submit_form(form_details, url, js_script).content.decode()
    if js_script in content:
        print(f"[+] XSS Detected on {url}")
        #print(f"[*] Form details:")
        pprint(form_details)
        is_vulnerable = True
        return("XSS: Vulnerability detected")
```

Fig 4.2.1: Code of injecting JavaScript code into forms.

Generally, we can say that if a website executes the JavaScript snippet in the form and gives output to the user like "alert ()" statements. Then the website has XSS vulnerability.

4.3 Detection of Phishing vulnerability (Social Engineering)

Even though an attacker can send thousands of fraudulent messages, only a small proportion of the recipients fall into that scam, to steal significant information and money. Hackers design phishing pages to imitate a spoofed organization's actual websites. The websites appear legitimate using the same phrases, types, logos, and signatures. Further, Attackers usually try to get users attention to act by creating a sense of urgency. The user is less diligent and more susceptible to errors by applying this pressure. Ultimately, links within messages look like their legal colleagues, but typically include an incorrect domain name or additional subdomains. Example like My.edu/renewal URL could be changed as my.edurenewal.com. Similarity between these two web addresses create the illusion for the user that the link is safe, which lessens the awareness of the attack.

In this module, we have taken a dataset containing about 2000 web URLs. We divide the URL into two parts for training and testing in 80 and 20 percentage. 80 percentage as training dataset and 20 percentage as testing dataset. To detect a phishing website, we must consider the below few features.

1. Having a long URL that hides the suspicious part of the URL.
2. URL having "@" symbol.
3. Having a "//" symbol in the URL, which redirects the page to a malicious one.
4. Cast a prefix or suffix Separated by "-" to the Domain.
5. Sub-Domain and Multi Sub-Domains

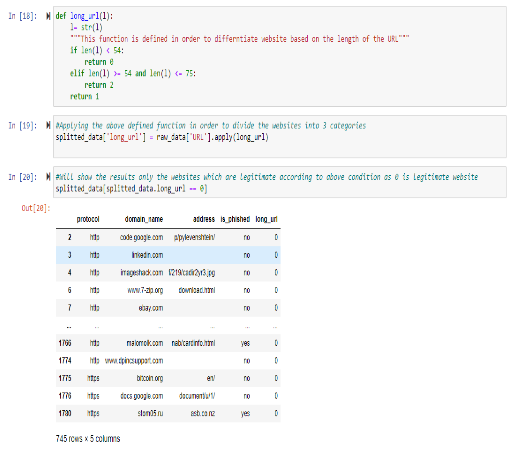
These are the features that can indicate whether a website is malicious. Going through a detailed explanation of the features.

1. The URL shall be classified as phishing if the URL is greater than or equal to 54 characters.

   0 --- indicates legitimate

   1 --- indicates Phishing

   2 --- indicates Suspicious



```
In [18]:  def long_url(l):
              l= str(l)
              """This function is defined in order to differntiate website based on the length of the URL"""
              if len(l) < 54:
                  return 0
              elif len(l) >= 54 and len(l) <= 75:
                  return 2
              return 1

In [19]:  #Applying the above defined function in order to divide the websites into 3 categories
          splitted_data['long_url'] = raw_data['URL'].apply(long_url)

In [20]:  #Will show the results only the websites which are legitimate according to above condition as 0 is legitimate website
          splitted_data[splitted_data.long_url == 0]

Out[20]:
```

	protocol	domain_name	address	is_phished	long_url
2	http	code.google.com	p/pylevenshtein/	no	0
3	http	linkedin.com		no	0
4	http	imageshack.com	f/219/cadir2yr3.jpg	no	0
6	http	www.7-zip.org	download.html	no	0
7	http	ebay.com		no	0
...	...	...	...	...	...
1766	http	malomolk.com	nab/cardinfo.html	yes	0
1774	http	www.dpincsupport.com		no	0
1775	https	bitcoin.org	en/	no	0
1776	https	docs.google.com	document/u/1/	no	0
1780	https	stom05.ru	asb.co.nz	yes	0

745 rows × 5 columns

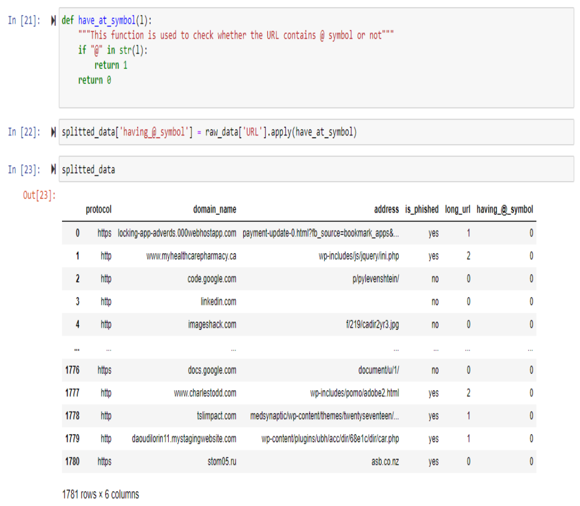
Fig 4.3: Output for testing feature - 1(URL)

2. Using the "@" symbol in the URL, the browser overlooks all the above "@" symbol, and the address usually follows the "@" symbol.

   IF {URL Having @ Symbol→ Phishing Otherwise→ Legitimate}

   0 --- indicates legitimate

   1 --- indicates Phishing



```
In [21]:  def have_at_symbol(l):
              """This function is used to check whether the URL contains @ symbol or not"""
              if "@" in str(l):
                  return 1
              return 0

In [22]:  splitted_data['having_@_symbol'] = raw_data['URL'].apply(have_at_symbol)

In [23]:  splitted_data

Out[23]:
```

	protocol	domain_name	address	is_phished	long_url	having_@_symbol
0	https	locking-app-adverds.000webhostapp.com	payment-update-0.html?fb_source=bookmark_apps&...	yes	1	0
1	http	www.myhealthcarepharmacy.ca	wp-includes/js/jquery/ini.php	yes	2	0
2	http	code.google.com	p/pylevenshtein/	no	0	0
3	http	linkedin.com		no	0	0
4	http	imageshack.com	f/219/cadir2yr3.jpg	no	0	0
...	...	...	...	...	...	...
1776	https	docs.google.com	document/u/1/	no	0	0
1777	http	www.charlestodd.com	wp-includes/pomo/adobe2.html	yes	2	0
1778	http	tslimpact.com	medsynaptic/wp-content/themes/twentyseventeen/...	yes	1	0
1779	http	daoudilorin11.mystagingwebsite.com	wp-content/plugins/ubh/acc/dir/68e1c/dir/car.php	yes	1	0
1780	https	stom05.ru	asb.co.nz	yes	0	0

1781 rows × 6 columns

Fig 4.3.1: Output for testing feature – 2(having '@' symbol)

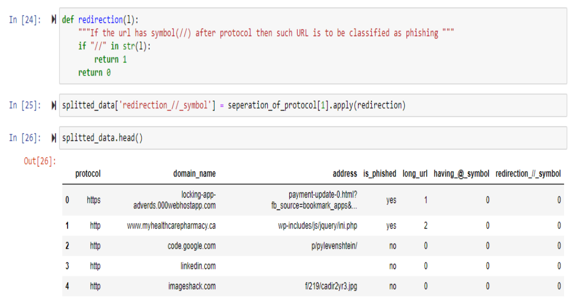
3. The presence of "//" within the URL means, the user will be redirected to another website. Example of such URLs : http://www.legitimate.com, //http://www.phishing.com". We examine the location where is the "//" present. If we find that if the URL starts with "HTTP", means that "//" should appear in the sixth position. However, if the URL employs "HTTPS" then the "//" should appear in the seventh position.

   IF {The Position of the Last Occurrence of "//" in the URL > 7→ Phishing

   otherwise→ Legitimate

   0 -- express legitimate

   1 -- express Phishing



```
In [24]:  def redirection(l):
              """If the url has symbol(//) after protocol then such URL is to be classified as phishing """
              if "//" in str(l):
                  return 1
              return 0

In [25]:  splitted_data['redirection_//_symbol'] = seperation_of_protocol[1].apply(redirection)

In [26]:  splitted_data.head()

Out[26]:
```

	protocol	domain_name	address	is_phished	long_url	having_@_symbol	redirection_//_symbol
0	https	locking-app-adverds.000webhostapp.com	payment-update-0.html?fb_source=bookmark_apps&...	yes	1	0	0
1	http	www.myhealthcarepharmacy.ca	wp-includes/js/jquery/ini.php	yes	2	0	0
2	http	code.google.com	p/pylevenshtein/	no	0	0	0
3	http	linkedin.com		no	0	0	0
4	http	imageshack.com	f/219/cadir2yr3.jpg	no	0	0	0

Fig 4.3.2: Output for testing the feature - 3 (URL containing “//” symbol)

4. In genuine URLs the hyphen (-) symbol was hardly used. For users to feel they deal with a genuine webpage; phishers are likely to add prefixes or suffixes divided by (-).

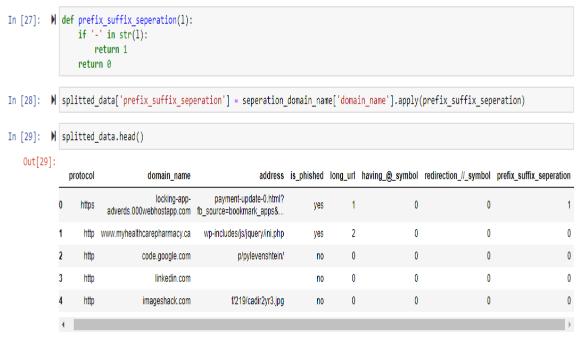
   For example, http://www.confirmpay-paypalmoney.com/.

   IF {Domain Name Part Includes (–) Symbol → Phishing

   Otherwise → Legitimate

   1 --> indicates phishing

   0 --> indicates legitimate



```
In [27]:  def prefix_suffix_seperation(l):
              if '-' in str(l):
                  return 1
              return 0

In [28]:  splitted_data['prefix_suffix_seperation'] = seperation_domain_name['domain_name'].apply(prefix_suffix_seperation)

In [29]:  splitted_data.head()

Out[29]:
```

	protocol	domain_name	address	is_phished	long_url	having_@_symbol	redirection_//_symbol	prefix_suffix_seperation
0	https	locking-app-adverds.000webhostapp.com	payment-update-0.html?fb_source=bookmark_apps&...	yes	1	0	0	1
1	http	www.myhealthcarepharmacy.ca	wp-includes/js/jquery/ini.php	yes	2	0	0	0
2	http	code.google.com	p/pylevenshtein/	no	0	0	0	0
3	http	linkedin.com		no	0	0	0	0
4	http	imageshack.com	f/219/cadir2yr3.jpg	no	0	0	0	0

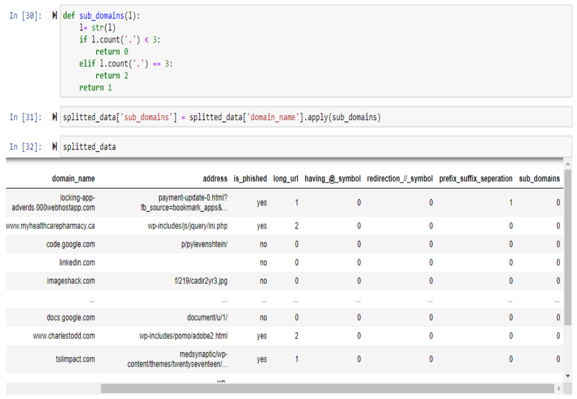
Fig 4.3.2: Output for testing feature - 4 (URL containing ‘-‘symbol)

5. There are two dots in the URL for a legitimate URL link, because "www." can be neglected. The URL is categorized as a "suspicious" if the number of points equaled three since it is divided by one subdomain. But if the points exceed three, they are categorized as "Phishy" because they will be subdomains.

   0 --- indicates legitimate

   1 --- indicates Phishing

   2 --- indicates Suspicious

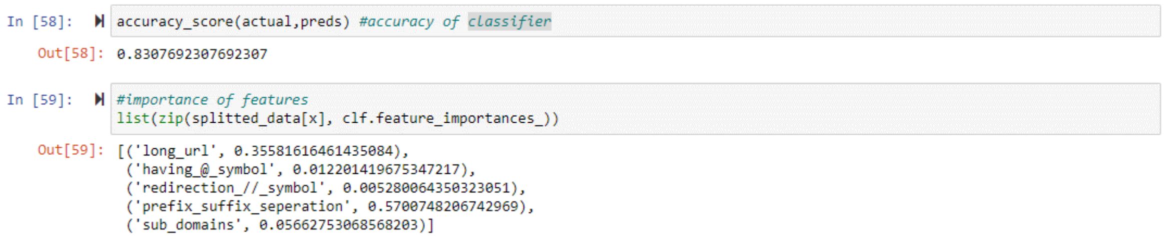


```
In [30]:  def sub_domains(l):
              l= str(l)
              if l.count('.') < 3:
                  return 0
              elif l.count('.') == 3:
                  return 2
              return 1

In [31]:  splitted_data['sub_domains'] = splitted_data['domain_name'].apply(sub_domains)

In [32]:  splitted_data
```

	domain_name	address	is_phished	long_url	having_@_symbol	redirection_//_symbol	prefix_suffix_seperation	sub_domains
	locking-app-adverds.000webhostapp.com	payment-update-0.html?fb_source=bookmark_apps&...	yes	1	0	0	1	0
	www.myhealthcarepharmacy.ca	wp-includes/js/jquery/ini.php	yes	2	0	0	0	0
	code.google.com	p/pylevenshtein/	no	0	0	0	0	0
	linkedin.com		no	0	0	0	0	0
	imageshack.com	f/219/cadir2yr3.jpg	no	0	0	0	0	0
	...	...	...	...	...	...	...	...
	docs.google.com	document/u/1/	no	0	0	0	0	0
	www.charlestodd.com	wp-includes/pomo/adobe2.html	yes	2	0	0	0	0
	tslimpact.com	medsynaptic/wp-content/themes/twentyseventeen/...	yes	1	0	0	0	0

Fig 4.3.3: Output of testing feature – 5 (URL containing two dots).

As we know, Random forests combines all the decision trees to produce a stable and precise result. Here these five features act as individual decision-making trees. So, we use these five features to get the outcome in the random forest algorithm. The results indicate that whether the URL is Phishing or not.



```
In [58]:  accuracy_score(actual,preds) #accuracy of classifier

Out[58]:  0.8307692307692307

In [59]:  #importance of features
          list(zip(splitted_data[x], clf.feature_importances_))

Out[59]:  [('long_url', 0.35581616461435084),
           ('having_@_symbol', 0.012201419675347217),
           ('redirection_//_symbol', 0.005280064350323051),
           ('prefix_suffix_seperation', 0.5700748206742969),
           ('sub_domains', 0.05662753068568203)]
```

Fig 4.3.4: Code for testing accuracy.

Using these decision trees, we tested this random forest classification model using the training dataset after the random forest model was built. The training data set is approximately 83 percent accurate. When a user gives a targeted URL as input this classifier takes this URL and comes to a decision after applying the extraction features whether the URL is phishing or not.

4.4 Building Graphical User Interface

GUI was developed using python and PyQt5 package. We title on the top using qt widgets module and we also incorporated QPixmap. For user input, we made an input box in the center using QLabel. A button is made to toggle the running of all the functions. Finally, an output box is created on the bottom using QLabel and for cursor elements we used QCursor.

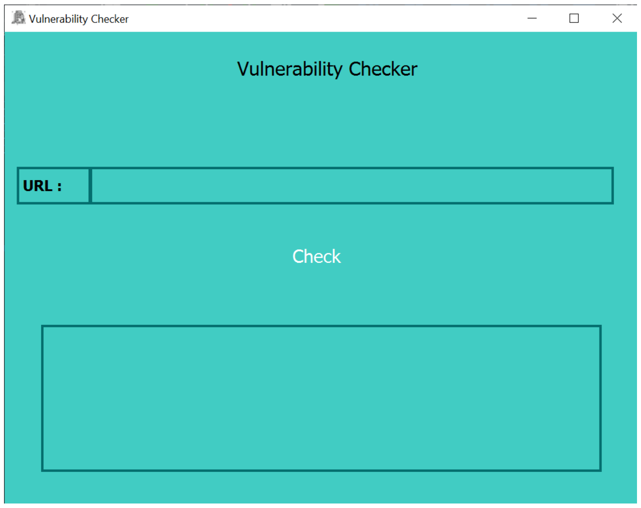


Fig 4.4: Structure of Graphical User Interface.

The main reason to create this GUI is to link all the three vulnerabilities function and get the output in a single place to improve the user experience. Whenever the user provides an URL as an input and the check button is clicked in the backend all three vulnerabilities detection modules are activated by taking this URL as an argument. The output of each module is stored in temporary variables. In the end, after we receive all the output from these modules, we print them in the output window of the GUI. Whenever the user gives an incorrect URL, a message is prompted saying please enter the correct URL. URLs without "HTTP" or which are not working are considered as incorrect URLs.

5.ENVIRONMENT

Python is a powerful programming language and a higher - level language. Python has straightforward declarations in English that allow us to know the code without much expertise of it. With its use of a considerable whitespace, Python's structure allows readability of the code. It is also followed by indents that do not prevent the code from being analyzed or debugged. The object-oriented approach is a clear overview of the logical and operational for users.

Interface - Spyder, Jupyter Notebook

Spyder is an IDE for integrated scientific programming used for Python Language, a cross-platform open-source environment. Spyder is integrated into the scientific Python Stack with many pronounced packages, as well as NumPy, SciPy, Pandas, IPython, Cython, and other open-source software. Spyder also includes Python Science. It is published under the MIT permit. Originally developed in 2009 by Pierre Raybaut, the group of scientific Python developers and the society have maintained and improved Spyder constantly since 2012. Spyder uses Qt for its GUI, which is engineered to use Pyqt or PySide Python bindings. QtPy a thin layer of abstraction developed by the Spyder anaconda, which is eventually adopted by numerous other packages.

The Jupyter Notebook is a free web-based application. It is an open source in which documents containing code, text or equation can be created and shared. Jupyter is a project that supports various programming languages by developing the Jupyter user interface.

Jupyter Notebooks is an IPython sub-project. It primarily supports Julia, Python and R, which is why it is called Jupyter. Together with the kernel IPython, Jupyter. It allows users to write their Python, R. code. There are about 100 additional kernels accessible for running the program in Jupyter.

Python Packages

BeautifulSoup:

BeautifulSoup is a HTML and XML parsing Python package. It generates a parse tree for parsed pages, which is helpful in web scraping, in process of extracting HTML data. BeautifulSoup was launched by Leonard Richardson, who continues to make contributions to this project and is also endorsed by Tidelift. Python 2.7 and Python 3 are available.

The Python library BeautifulSoup is designed for fast upturn, screen scraping projects. It is effective with three factors:

1. Beautiful Soup has some simple methods as well as Pythonic languages to navigate, browse, and edit the parsing tree: a document deconstructs and obtain toolkit. To type an application does not take a lot of code.
2. Beautiful Soup converts incoming and outgoing documents instantaneously to Unicode and UTF-8. You must not be thinking of coding except if you stipulate an encoding in the document, nor can be detected by Beautiful Soup. The actual encoding then only must be specified.
3. BeautifulSoup sits atop of popular Python parsers such as lxml and html5lib, which enable you to test various pars ing techniques or trade-speed for versatility.

Pprint

The pprint module allows arbitrarily chosen Python data structures to be 'pretty-printed' in a form that can be used as the interpreter's input. The interpretation cannot be loaded if the structures configured include objects that do not include essential Python types. This could be the case if objects like sockets, Files, classes, and other objects were included in the that cannot be represented as Python literals.

The formatted display holds items on a line if possible and tries to break them in multiple items if not within the width permitted. If you are to adjust the spacing constraint, build PrettyPrinter objects clearly and unambiguously.

NumPy

NumPy or Number Python is a library for certain array functions. It contains a n-dimensional array object useful for the whole array to apply algebra and different other mathematical formulas. The object NumPy can also be called the array NumPy. It stores information so that mathematical calculations are effectively carried out. In tabular format, NumPy Array store information, i.e., rows and columns. To use NumPy check if NumPy is or is not installed. If the NumPy package is not installed using the steps below. Go to the command prompt to setup NumPy. Enter and type the command below:

"pip install numpy"

Pandas

Pandas is the Python Software Library used for data Manipulation and Analysis of any programming language. It provides data-structures and operations for numerical tables and Time series Manipulation. This software is freely available under the BSD license of three clauses. The name derives from the term "panel data" which includes observations over multiple periods for the same people, an econometric term for data sets. The phrase "Python data analysis" itself is the reproduction of its name. Wes McKinney began building the AQR Capital pandas as a researcher between 2007 and 2010.

Sklearn

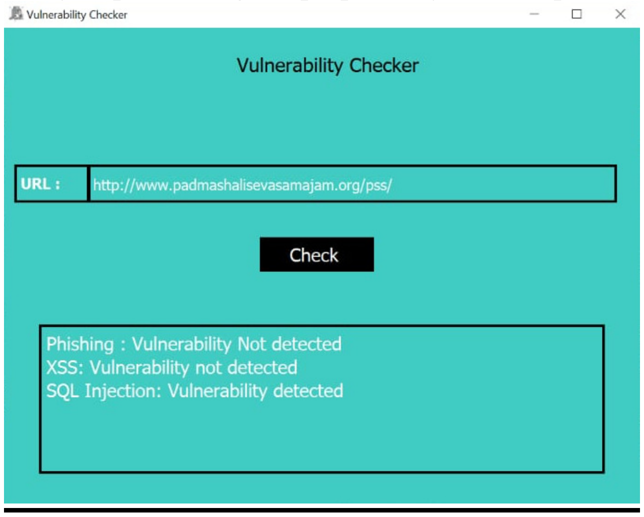
Scikit-learn is Python's free Machine Learning library. It includes different algorithms such as SVM, RF and k-Neighbors, and Python also supports numerical and scientific libraries. Python's leading machine learning library is Scikit-Learn, also called sklearn. While other packages are found that work better on certain tasks, the versatility of Scikit-Learn makes it the best starting point for most ML problems. It is also a fantastic beginner's library , it provides an interface of high levels for many tasks (e.g., preprocessing data, cross-validation, etc.). This allows you to practice and understand the overview of the entire machine learning workflow. The project was launched as a Google Summer Code project in 2007 by David Cornopean, and has contributed numerous volunteers ever since. For a list of core contributors, see the About us page. A team of volunteers is currently in charge of the project. Previously Scikit-learn was called Scikits.learn.

PyQt5

PyQt5, a set of Python bindings for the Qt Company's Qt application framework version 5. Qt, a collection of C++ libraries and development tools that include platform-independent abstractions for graphical user interfaces, threads, regular expressions, SQL databases, SVG, OpenGL, XML, user, networking,application settings, positioning and location services, short range communications (NFC and Bluetooth), web browsing, 3D animation, charts, 3D data visualization, and interacting with apps. As a series of Python modules, PyQt5 implements over 1000 of these classes. PyQt5 needs Python v3.5 or newer and runs on Windows, Linux, UNIX, Android, macOS, and iOS. PyQt5 also contains several utility programs.

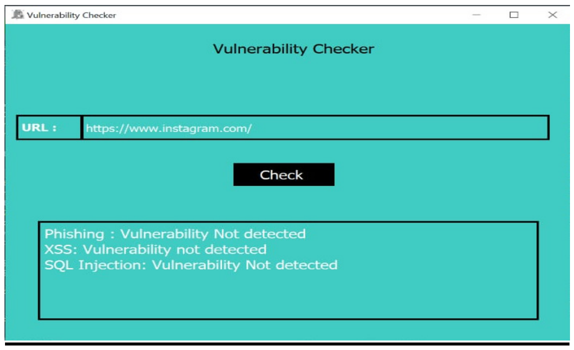
6. Results and Discussion

The results obtained by implementing the proposed system are presented in this section.



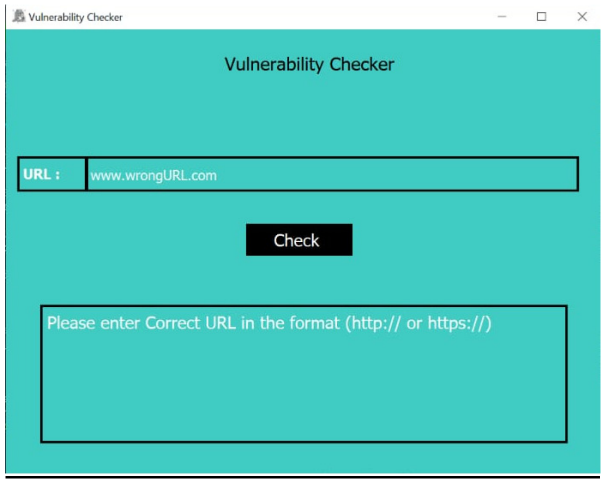
```
Phishing : Vulnerability Not detected
XSS: Vulnerability not detected
SQL Injection: Vulnerability detected
```

Fig 6.1: Testing SQL Injection functioning in the GUI



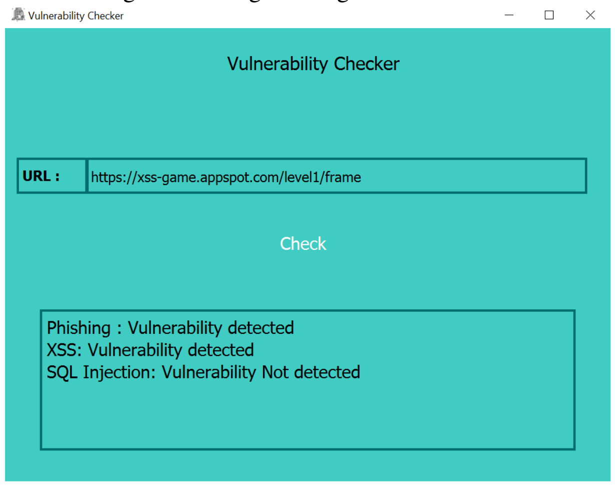
```
Phishing : Vulnerability Not detected
XSS: Vulnerability not detected
SQL Injection: Vulnerability Not detected
```

Fig 6.2: Testing a Genuine URL



Please enter Correct URL in the format (http:// or https://)

Fig 6.3: Testing a wrong format of URL.



```
Phishing : Vulnerability detected
XSS: Vulnerability detected
SQL Injection: Vulnerability Not detected
```

Fig 6.4: Testing Phishing and XSS vulnerabilities.

7.CONCLUSION and FUTURE SCOPE

With continuously evolving methods of hacking the detection of vulnerabilities becomes crucial. In this Paper we have created an GUI that can detect Phishing, Payload SQL injection and Reflected XSS vulnerability. The GUI can be run by anyone even without having prior knowledge of coding. It allows users to check any link just by copying it into the GUI and gives instant results. Random forest algorithms are used to detect phishing. Injecting SQL payloads and malicious scripts into the input fields of the given URL is the test of and detect the vulnerability of SQL Injections and Reflected XSS. When we get a particular output on load or script injection, the URL is classified as vulnerable to SQL injection or Reflected XSS. The GUI application can detect the given vulnerabilities in under 30 seconds with high accuracy and reliability.

We can improve the performance of the phishing module by using Deep learning techniques. Using larger datasets for the training and testing of random forest models can cultivate higher accuracy and precision.

To provide frontend extension by the development of web applications for smartphones and extensions for various browsers. We can also provide support for other operating systems like LINUX and MAC OS.

To facilitate detection of various types of Open Web Application Security Project (OWASP) vulnerabilities such as click-jacking, Remote File Inclusion (RFI), Local File Inclusion (LFI) etc.

REFERENCES

[1] W. Ali and S. Malebary, "Particle Swarm Optimization-Based Feature Weighting for Improving Intelligent Phishing Website Detection," in IEEE Access, vol. 8, pp. 116766-116780, 2020, doi: 10.1109/ACCESS.2020.3003569.

[2] Y. A. Alsariera, V. E. Adeyemo, A. O. Balogun and A. K. Alazzawi, "AI Meta-Learners and Extra-Trees Algorithm for the Detection of Phishing Websites," in IEEE Access, vol. 8, pp. 142532-142542, 2020, doi: 10.1109/ACCESS.2020.3013699.

[3] HR, M., MV, A., S, G. *et al.* Development of anti-phishing browser based on random forest and rule of extraction framework. *Cybersecur* **3,** 20 (2020). https://doi.org/10.1186/s42400-020-00059-1

[4] F. M. M. Mokbal, W. Dan, A. Imran, L. Jiuchuan, F. Akhtar and W. Xiaoxi, "MLPXSS: An Integrated XSS-Based Attack Detection Scheme in Web Applications Using Multilayer Perceptron Technique," in IEEE Access, vol. 7, pp. 100567-100580, 2019, doi: 10.1109/ACCESS.2019.2927417.

[5] Akrout et al.: An automated black box approach for web vulnerability identification and attack scenario generation. Journal of the Brazilian Computer Society 2014 20:4.

[6] Z. Dou, I. Khalil, A. Khreishah, A. Al-Fuqaha and M. Guizani, "Systematization of Knowledge (SoK): A Systematic Review of Software-Based Web Phishing Detection," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2797-2819, Fourthquarter 2017, doi: 10.1109/COMST.2017.2752087.

[7] Bakare K. Ayeni, Junaidu B. Sahalu, Kolawole R. Adeyanju, "Detecting Cross-Site Scripting in Web Applications Using Fuzzy Inference System", *Journal of Computer Networks and Communications*, vol. 2018, Article ID 8159548, 10 pages, 2018. https://doi.org/10.1155/2018/8159548

[8] A. A.A. and P. K., "Towards the Detection of Phishing Attacks," 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184), Tirunelveli, India, 2020, pp. 337-343, doi: 10.1109/ICOEI48184.2020.9142967.

[9] Stephen W. Boyd and Angelos D. Keromytis Department of Computer Science Columbia University {swb48,angelos}@cs.columbia.edu

[10] K. Kamtuo and C. Soomlek, "Machine Learning for SQL injection prevention on server-side scripting," 2016 International Computer Science and Engineering Conference (ICSEC), Chiang Mai, Thailand, 2016, pp.

[11] M. Liu, B. Zhang, W. Chen and X. Zhang, "A Survey of Exploitation and Detection Methods of XSS Vulnerabilities," in *IEEE Access*, vol. 7, pp. 182004-182016, 2019, doi: 10.1109/ACCESS.2019.2960449.

[12] A. El Aassal, S. Baki, A. Das and R. M. Verma, "An In-Depth Benchmarking and Evaluation of Phishing Detection Research for Security Needs," in *IEEE Access*, vol. 8, pp. 22170-22192, 2020, doi: 10.1109/ACCESS.2020.2969780.

[13] X. Liu and J. Fu, "SPWalk: Similar Property Oriented Feature Learning for Phishing Detection," in *IEEE Access*, vol. 8, pp. 87031-87045, 2020, doi: 10.1109/ACCESS.2020.2992381.

[14] Parvathapuram Pavan Kumar, T. Jaya, V. Rajendran,SI-BBA – A novel phishing website detection based on Swarm intelligence with deep learning,Materials Today: Proceedings,2021,ISSN 2214-7853 ,\,https://doi.org/10.1016/j.matpr.2021.07.178.

[15] Grega Vrbančič, Iztok Fister, Vili Podgorelec,Datasets for phishing websites detection,Data in Brief,Volume 33,2020,106438,ISSN2352-3409,https://doi.org/10.1016/j.dib.2020.106438.

[16] Priya Saravanan, Selvakumar Subramanian,A Framework for Detecting Phishing Websites using GA based Feature Selection and ARTMAP based Website Classification, Procedia Computer Science,
Volume 171, 2020, Pages 1083-1092, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2020.04.116.