

WEBSITE VULNERABILITY DETECTOR AND CODE QUALITY ANALYZER

A PROJECT REPORT

Submitted by

AKANSHA JHA (21BCY10228)
JANNAT KHAN (21BCY10093)
AYUSH GHOGRE (21BCY10063)
PRAKHAR MISHRA (21BCY10047)

*in partial fulfillment for the award of the degree
of*

BACHELOR OF TECHNOLOGY

in

CYBER SECURITY AND DIGITAL FORENSICS



VIT[®]
BHOPAL
www.vitbhopal.ac.in

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

VIT BHOPAL UNIVERSITY

KOTHRI KALAN, SEHORE

MADHYA PRADESH - 466114

**VIT BHOPAL UNIVERSITY, KOTHRI KALAN, SEHORE
MADHYA PRADESH – 466114**

BONAFIDE CERTIFICATE

Certified that this project report titled **“WEBSITE VULNERABILITY DETECTOR AND CODE QUALITY ANALYZER”** is the bonafide work of **“AKANSHA JHA (21BCY10228), JANNAT MATEENULLAH KHAN (21BCY10093), AYUSH GHOGRE (21BCY10063), PRAKHAR MISHRA (21BCY10047)”** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

PROGRAM CHAIR

Dr. R. Rakesh
School of Computer Science and Engineering
VIT BHOPAL UNIVERSITY

PROJECT GUIDE

Dr. S. Rajasoundaran (Supervisor)
School of Computer Science and Engineering
VIT BHOPAL UNIVERSITY

The Project Exhibition I Examination is held on 3rd October, 2022

ACKNOWLEDGEMENT

First and foremost, we would like to thank the Lord Almighty for His presence and immense blessings throughout the project work.

We wish to express our heartfelt gratitude to Dr. S. Poonkuntran, Head of the Department, School of Computer Science for much of his valuable support and encouragement in carrying out this work.

We wish to express our gratitude to Dr. R. Rakesh, Programme Chair, B.Tech CSE(Cyber) for much of his valuable support and encouragement in carrying out this work.

We would like to thank our internal guide Dr. S. Rajasoundaran, for continually guiding and actively participating in our project, giving valuable suggestions to complete the project work.

We would like to thank all the technical and teaching staff of the School of Computer Science, who extended directly or indirectly all support.

Last, but not least, we are deeply indebted to our parents who have been the greatest support while we worked day and night for the project to make it a success.

LIST OF ABBREVIATIONS

S. No.	ABBREVIATIONS	FULL FORM
1.	SQLIA	Structured Query Language Injection Attack
2.	XSS	Cross Site Scripting
3.	GUI	Graphical User Interface
4.	SYN	Synchronize
5.	SYS	System
6.	VSC	Visual Studio Code
8.	OS	Operating System
8.	CVE	Common vulnerabilities and exposure

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
3.3.4	LOOK AND FEEL REQUIREMENTS	19
4.1	METHODOLOGY AND GOAL	21
4.1	METHODOLOGY AND GOAL	22-23
4.3	SOFTWARE ARCHITECTURAL DESIGN	26
4.4	USER INTERFACE DESIGN	27
5.2	TECHNICAL CODING AND CODE SOLUTIONS	29
5.2	TECHNICAL CODING AND CODE SOLUTIONS	30
5.2	TECHNICAL CODING AND CODE SOLUTIONS	31
5.2	TECHNICAL CODING AND CODE SOLUTIONS	32
5.2	TECHNICAL CODING AND CODE SOLUTIONS	33
5.3	PROTOTYPE SUBMISSION	33
5.4	TEST AND VALIDATION	34
5.5	PERFORMANCE ANALYSIS	34-35

ABSTRACT

Website vulnerabilities are the flaws or weaknesses available in web-based applications. A website vulnerability is a software code flaw/ bug, system misconfiguration, or some other weakness in the website/ web application or its components and processes. Web application vulnerabilities enable attackers to gain unauthorized access to systems/ processes/ mission-critical assets of the organization. Having such access, attackers can orchestrate attacks, takeover applications, engage in privilege escalation to exfiltrate data, cause large-scale service disruption, and so on. It is reported (according to NTT application security) that on an average about 50% of web applications are vulnerable to attacks in 2021. Web Vulnerabilities can act as an open door opportunity for hackers. This paper focuses our attention on three most common vulnerabilities (SQLIA, XSS and Phishing) of web- application. We had also surveyed existing web vulnerabilities assessment tools advantages and disadvantages, and proposed a GUI-based mechanism for scanning them over any website, just by providing the url of the suspected website.

Keywords- Web Vulnerabilities, SQLIA, Phishing, XSS, GUI

TABLE OF CONTENTS

SR. NO.	TITLE	PAGE NO.
	List of Abbreviations	iv
	List of Figures	v
	Abstract	vi
1	CHAPTER-1: PROJECT DESCRIPTION AND OUTLINE 1.1 Introduction 1.2 Motivation for the work 1.3 Introduction to the project including techniques 1.4 Problem Statement 1.5 Objective of the work 1.6 Summary	10
2	CHAPTER-2: RELATED WORK INVESTIGATION 2.1 Introduction 2.2 Core area of the project 2.3 Existing Approaches/Methods 2.4 Pros and cons of the existing Approaches/Methods 2.5 Issues/observations from investigation 2.6 Summary	13
3	CHAPTER-3: REQUIREMENT ARTIFACTS 3.1 Introduction	17

	3.2 Hardware and Software requirements 3.3 Specific Project requirements 3.3.1 Data requirement 3.3.2 Functions requirement 3.3.3 Performance and security requirement 3.3.4 Look and Feel Requirements 3.4 Summary	
4	CHAPTER-4: DESIGN METHODOLOGY AND ITS NOVELTY 4.1 Methodology and goal 4.2 Functional modules design and analysis 4.3 Software Architectural designs 4.4 User Interface design 4.5 Summary	21
5	CHAPTER-5: TECHNICAL IMPLEMENTATION & ANALYSIS 5.1 Outline 5.2 Technical coding and code solutions 5.3 Prototype submission 5.4 Test and validation 5.5 Performance Analysis (Table) 5.6 Summary	29
6	CHAPTER-6: PROJECT OUTCOME AND APPLICABILITY 6.1 Outline 6.2 Significant project outcomes 6.3 Project applicability on Real-world applications 6.4 Inference	36

7	CHAPTER-7: CONCLUSIONS AND RECOMMENDATION 7.1 Outline 7.2 Limitation/Constraints of the System 7.3 Future Enhancements 7.4 Inference	38
	Appendix A	40
	References	42

CHAPTER – 1

PROJECT DESCRIPTION AND OUTLINE

1.1 INTRODUCTION

Websites experience multiple attacks per day. A website vulnerability is a weakness or misconfiguration in a website or web application code that allows an attacker to gain some level of control of the site, and possibly the hosting server. Most vulnerabilities are exploited through automated means, such as vulnerability scanners and botnets. Cybercriminals create specialized tools that scour the internet for certain platforms, like WordPress or Joomla, looking for common and publicized vulnerabilities. Once found, these vulnerabilities are then exploited to steal data, distribute malicious content, or inject defacement and spam content into the vulnerable site.

1.2 MOTIVATION FOR THE WORK

The motivation behind developing this website vulnerability detection tool is to use it against common and treacherous attempts on people either by luring or clickbaiting them into a fake website. According to its use in various ways to scan their system in order to keep their system safe, as well as, it can be used for finding the malicious attack types among the three common vulnerabilities namely XSS, SQLIA and phishing. The main goal of using the website vulnerability detection GUI is to scan the website links or URLs to know if there are any loopholes in the links. If there are any loopholes in their system the detector can detect them, and warn us against any malicious attack.

1.3 INTRODUCTION TO THE PROJECT INCLUDING TECHNIQUES

Website vulnerabilities are weaknesses or holes in a website or a web app that can be exploited by hackers and cyber attackers. A Web application scanner is an automated security program that searches for software vulnerabilities within Web applications. A Web application scanner first crawls the entire website, analyzing in-depth each file it finds, and displaying the entire

website structure. After this discovery stage, it performs an automatic audit for common security vulnerabilities by launching a series of Web attacks. Web application scanners check for vulnerabilities on the Web server, proxy server, Web application server and even on other Web services.

1.4 PROBLEM STATEMENT

Hackers have been successful in finding a gaping hole in the corporate security infrastructure, one of which organizations were previously unaware – Web applications. By design, Web applications are publicly available on the Internet, 24/7. This provides hackers with easy access and allows almost unlimited attempts to hack applications that have not been identified by webmasters as vulnerable through the use of a web application scanning solution. Organizations need a Web application scanning solution that can scan for security loopholes in Web-based applications to prevent would-be hackers from gaining unauthorized access to corporate information and data. Web applications are proving to be the weakest link in overall corporate security, even though companies have left no stone unturned in installing the better-known network security and anti-virus solutions. Quick to take advantage of this vulnerability, hackers have now begun to use Web applications as a platform for gaining access to corporate data; consequently the regular use of a web application scanner is essential.

1.5 OBJECTIVE OF THE WORK

Our objective is to make a website scanner that will protect us from various malicious threats. A website security scanner is a program that will help you find potential vulnerabilities in your site. There are many tools in this field but each tool has some cons along with pros. A vulnerability scan will check all pages and scripts in your system, looking for flaws like open ports. This leaves the website susceptible to hacking and attacks from hackers and malware. Website security scanning is definitely worth doing, especially if you own a website that holds sensitive data about customers like their credit card numbers or personal information.

1.6 SUMMARY

The basic idea is to scan any website in a very fast manner and save the time of the user. Our objective is to develop the field of security. Our tool can finish scanning any website in under a minute. Website detector proves to be useful in many cases, an authorized user can use this tool to see if there are any websites and URLs containing any kind of suspicious attack and to report this to the user in a matter of seconds.

CHAPTER – 2

RELATED WORK INVESTIGATION

2.1 INTRODUCTION

At the investigation phase of our project, we planned to investigate all kinds of available website vulnerabilities that may provide risk to web applications. During this process, we categorized top 13 web application security risks.

1. Sqlia
2. Broken Authentication
3. Sensitive data exposure
4. XML external entities(XEE)
5. Broken Access Control
6. Security misconfiguration
7. Cross-site scripting(XSS)
8. Insecure Deserialization
9. Vulnerable and Outdated components
10. Security logging and Monitoring Failures
11. Insecure Design
12. Software and Data Integrity Failures
13. Server side Request Forgery

2.2 CORE AREA OF THE PROJECT

Core area of the project is 3 most common Website vulnerabilities scanning, and our objective is to develop a vulnerability scanning tool with extra added features like – graphical user interface and efficiency (took less time). To enable the quick and effective scanning of the file/folder. Our aim is to develop a flawless algorithm for our tool which contains solutions to the most problems in existing tools.

2.3 EXISTING APPROACHES/METHODS

As we are developing a scanning tool, so are the existing tools which use different methods. For example, OpenVAS has a client-server architecture, in which on the server-side, all the processing is done in search of vulnerabilities, and storage of settings and scans is also

performed. The client-side provides an interface for the network administrator to configure the scan and view your reports.

2.4 PROS AND CONS OF THE EXISTING APPROACHES/METHODS

1. Nikto

Pros:-

- A free base program
- Thorough checks with the number of exploits in the standard scan match that sought by paid vulnerability managers
- External checks for Web applications
- Included in Kali Linux

Cons: -

- No GUI interface
- No development and support team
- No community forum
- Won't work without a paid vulnerability list

2. OpenVAS

Pros: -

- open-source vulnerability assessment tool
- popular and useful among SMEs
- built to be an-in-one scanner
- The scan engine is updated on regular basis

Cons: -

- Covers fewer CVEs as compared to Nessus
- Less operating system supportability
- Does not offer policy management

3.Nessus

Pros: -

- (CVE) Coverage of around 47,000
- Great customer support
- Facilitates group testing

- Offers real-time visibility
- Nessus has server-side compatibility with operating systems
- Built-in scan templates

Cons: -

- Nessus costs around \$2,790/year & is not viable for smaller companies
- It does not allow checking the local security policies of remote systems
- Network overload can be a drawback for Nessus
- Does not offer asset tagging and risk management

4.Netsparker

Pros: -

- easy to use
- customized for scanning any web application
- gives few false positive
- Many DevOps integration possibilities

Cons: -

- only for windows installed computers
- support only desktop or cloud version
- expensive
- very few vulnerabilities can be compared

5. Acunetix

Pros: -

- low rate of false positive
- good reporting option
- authenticated scans
- scan results are simple

Cons: -

- vulnerability identification speed is slow
- expensive
- requires more integration

2.5 ISSUES/OBSERVATIONS FROM INVESTIGATION

As we analyzed different types of tools, that too large in number, common findings were -

- A vulnerability scanning tool will not find nearly all vulnerabilities

- Constant updates required
- False positives
- Implications of vulnerability unclear

2.6 SUMMARY

There are quite a large number of tools available online, but they are quite expensive to use, giving false positives quite frequently. We are dedicated to understanding the basics of these scanners and removing available flaws making it more accessible, efficient, fast and easy to be used by anyone.

CHAPTER – 3

REQUIREMENT ARTIFACTS

3.1 INTRODUCTION

Our idea is to develop a website vulnerability detection tool which also analyzes code quality with GUI. It is user-friendly and it also provides the graphics-based interface that uses icons, buttons, prompt box, keyboard(to take input from the user) and a mouse (to click on the icon or press the button) to manage interactions with the system.To accomplish user-system interactions, we need hardware and software. By using our tool, the users would be able to give an url input to our system and detect if the vulnerabilities(i.e. SQLIA, XSS & Phishing Attacks) are present or not and the code quality is analyzed.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

We need to have -

- Laptop/Desktop/Tablet
- Any Python IDE
- It does not require any Internet Connection
- Windows 7 and above version
- Windows 7 & Windows 8:
 - ◆ 32-bit: 1GB RAM minimum, 2GB recommended
 - ◆ 64-bit: 2GB RAM minimum, 4GB recommended

3.3 SPECIFIC PROJECT REQUIREMENTS

3.3.1 DATA REQUIREMENT

- Our tool requires an input url.This url is then checked for any vulnerability.

- As Phishing detection is done by using Machine Learning Model(Logistic Regression), the dataset to train the models are required.

3.3.2 FUNCTIONS REQUIREMENT

- VS Code/ Any Python IDE:As we have used visual studio framework in our project, it is a very popular IDE for writing python. It combines the best developer experience with an obsessive focus on end-user performance. Our platform enables frontend teams to do their best work. Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

3.3.3 PERFORMANCE AND SECURITY REQUIREMENT

- PERFORMANCE REQUIREMENT:
 - ★ Speed :
 - XSS Detection- 05.819 s(approx)
 - SQLIA Detection-02.477 s(approx)
 - Phishing Detection-30.00 s (approx)
 - ★ Software's speed of response: 04.602 s(approx)
 - ★ Execution time: 10.421 s(approx)
 - ★ Storage Capacity: 12 kb
- SECURITY REQUIREMENT:
 - ★ Code Quality: Integrated and Validated Code

- ★ System Input: System won't accept any url with missing scheme.

3.3.4 LOOK AND FEEL REQUIREMENTS

Our tool is aesthetic and easy to use.

We have used Tkinter module to design our GUI.

Our interface:



3.4 SUMMARY:

This chapter covers all the hardware and software requirements used in our project. It also gives a brief description of performance and security requirements that our project satisfies. It is finally concluded with the aesthetic aspect of our project.

CHAPTER – 4

DESIGN METHODOLOGY AND ITS NOVELTY

4.1 METHODOLOGY AND GOAL

We started our project by categorizing top 13 security risks to web applications.

Then we researched about 3 severe web vulnerabilities- SQLIA, XSS and Phishing Attack.

Characteristics of these website vulnerabilities are listed below:

1. SQLIA

Severity	High
Description	SQL injection attack occurs when: <ul style="list-style-type: none">● An unintended data enters a program from an untrusted source.● The data is used to dynamically construct a SQL query.
Exploitability	Factor of 3.
Detectability	Score of 3.
Impact	Impact of 5
Mitigation	<ul style="list-style-type: none">● Input Validation & Sanitation● Stored Procedures & Parametrization

2. XSS

Severity	High
----------	------

Description	<p>Cross-Site Scripting (XSS) attacks occur when:</p> <ol style="list-style-type: none"> 1.Data enters a Web application through an untrusted source, most frequently a web request. 2.The data is included in dynamic content that is sent to a web user without being validated for malicious content.
Exploitability	Factor of 3.
Detectability	Score of 3.
Impact	Impact of 2
Mitigation	<ul style="list-style-type: none"> ● Input validation ● Encoding

3. Phishing Attack

Severity	High
Description	<p>Phishing is a type of social engineering attack. It occurs when :</p> <ul style="list-style-type: none"> ● An attacker, masquerading as a trusted entity, dupes a victim into opening an email, instant message, or text message. ● The recipient is then tricked into clicking a malicious link. ● It can lead to the installation of malware, the freezing of the system as part of a ransomware attack or the revealing of sensitive information.
Exploitability	Factor of 3.

Detectability	Score of 3.
Impact	Impact of 4
Mitigation	<ul style="list-style-type: none"> ● Spam & Web filters ● Converting HTMLemail into text only email messages

After understanding the characteristics of vulnerabilities and performing vulnerability assessment, the algorithms that showed how each of these vulnerabilities work were designed. Understanding the pattern these vulnerabilities follow, corresponding detection codes were written.

The goal of our project is to build a software that takes an url from user and detect if the vulnerabilities (SQLIA, XSS or Phishing) are present in that website.If any of these 3 vulnerability is found in the user-given url, the software gives the output “Vulnerability Detected,Code: Illegitimate”.This project will help the user to identify malicious websites and be more secure in cyberspace.

4.2 FUNCTIONAL MODULES DESIGN AND ANALYSIS

OS

The OS module in Python provides functions for interacting with the operating system. OS comes under Python’s standard utility modules. This module provides a portable way of using operating system-dependent functionality. The *Os* and *Os. path* modules include many functions to interact with the file system.

SYS

The sys module in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment. It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter. It lets us

access system-specific parameters and functions. import sys- First, we have to import the sys module in our program before running any functions.

TkInter

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. To create a GUI using TkInter is easy, for that we need to follow these steps-

- Import the Tkinter module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

requests:

The requests module allows us to send HTTP requests using Python. The HTTP request returns a Response Object with all the response data (content, encoding, status, etc).

pprint:

The pprint module provides a capability to “pretty-print” arbitrary Python data structures in a well-formatted and more readable way.

BeautifulSoup

Beautiful Soup is a Python library that is used for web scraping purposes to pull the data out of HTML and XML files. It creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner.

urllib.parse

The urllib.parse module defines functions that fall into two broad categories: URL parsing and URL quoting. This module defines a standard interface to break Uniform Resource Locator (URL) strings up in components (addressing scheme, network location, path etc.), to combine the components back into a URL string, and to convert a “relative URL” to an absolute URL given a “base URL.”

Pandas

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

Scikit-Learn

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. It is used for building machine learning models.

Natural Language Toolkit (NLTK):

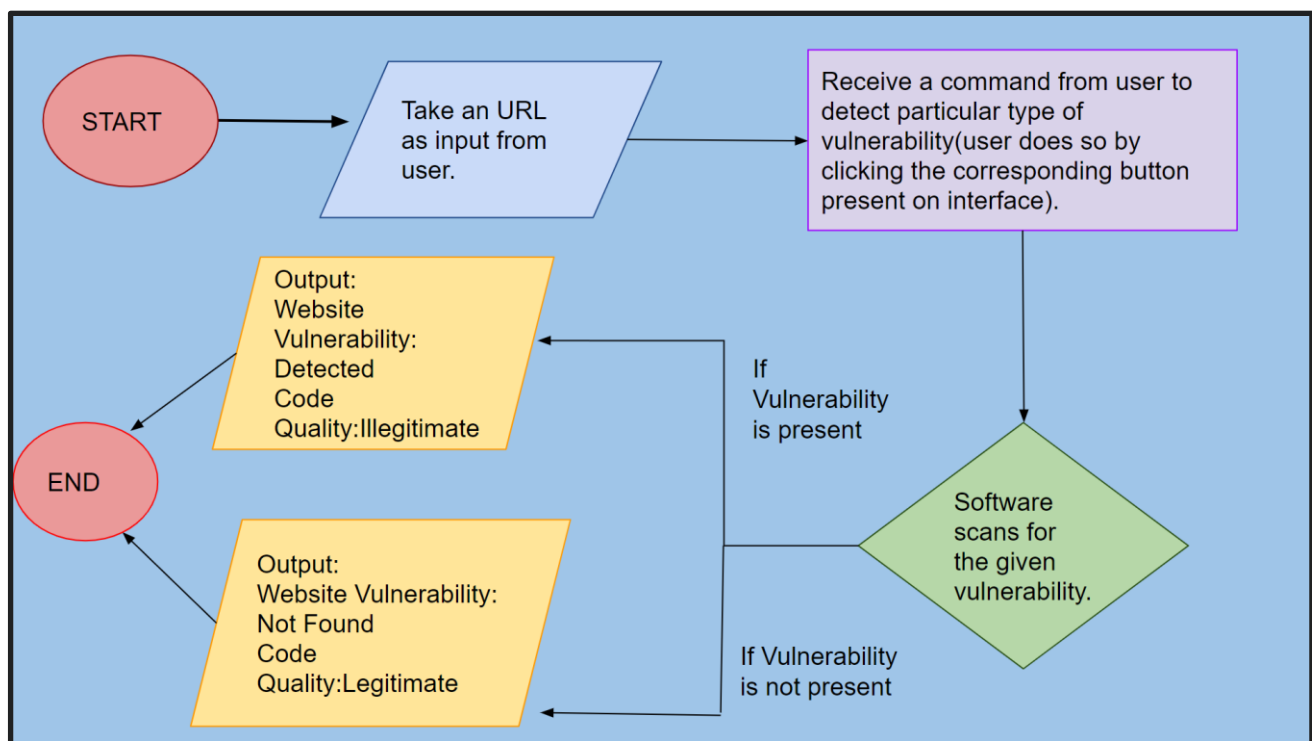
The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP).

It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning. It also includes graphical demonstrations and sample data sets as well as accompanied by a cook book and a book which explains the principles behind the underlying language processing tasks that NLTK supports.

Pickle

Pickle in Python is primarily used in serializing and deserializing a Python object structure. In other words, it's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network. The pickled byte stream can be used to re-create the original object hierarchy by unpickling the stream.

4.3 SOFTWARE ARCHITECTURAL DESIGN



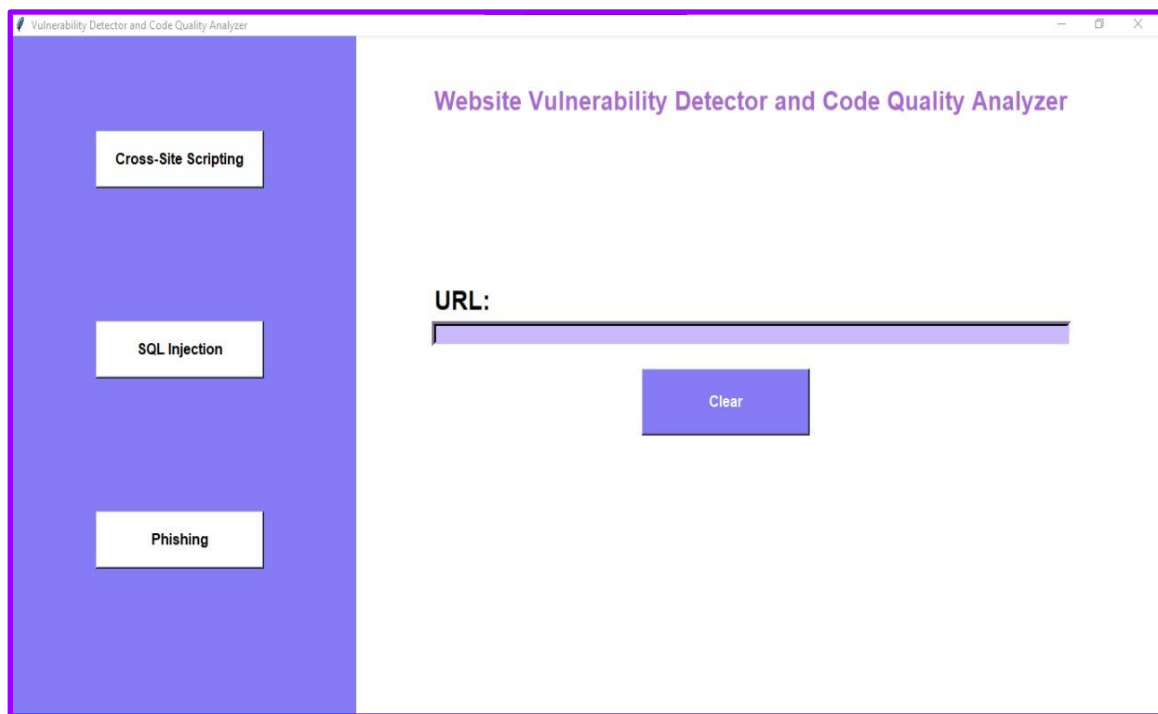
4.4 USER-INTERFACE DESIGN

Our user interface consists of:

1. **Name of the tool:** A heading “Website Vulnerability Detector and Code Quality Analyzer” is placed on the tool to identify it.
2. **Buttons(for Vulnerabilities):** The buttons for SQLIA,XSS & Phishing Attack are placed. After entering the url,the user has to press the button to analyze the particular vulnerability i.e. to analyze an url for phishing attack,the user has to press Phishing.

3. **Prompt Box:** A prompt box is placed below the command “URL” so that user can give url as input.
4. **CLEAR Button:** Clear button is used to clear the output once the tool gives the result.

INTERFACE:



As we can see our interface is:

1. User-friendly
2. Easy to use
3. Age-friendly
4. Aesthetic

4.5 SUMMARY

This chapter starts with stating our approach towards project and our final goal. Various modules used in the design of project are analyzed. The chapter is concluded by depicting our Software Architectural Design and User Interface Design.

CHAPTER – 5

TECHNICAL IMPLEMENTATION & ANALYSIS

5.1 OUTLINE

Our aim is to develop a flawless tool which is:

- User-friendly
- Invulnerable or resilient
- Compatible with windows OS
- Comfortable to view
- Age friendly
- Simple to use
- Does its job(i.e. vulnerability detection) with maximum accuracy

5.2 TECHNICAL CODING AND CODE SOLUTIONS

CODE DETAILS:

- Language used : Python
- Platform used: VS Code
- Snapshots of our code are given below:

```
Review-3 Project Exhibition.py > XSS
1  #importing libraries
2  import tkinter as tk
3  from tkinter import*
4  import os
5  from turtle import clear
6
7  #GUI
8  root = tk.Tk()
9  root.title('Vulnerability Detector and Code Quality Analyzer')
10 root.geometry("1366x768+40+40")
11 canvas = tk.Canvas(root, height = 3000, width = 3000, bg = "white")
12 canvas.grid()
13 myLabel1 = tk.Label(root, text="Website Vulnerability Detector and Code Quality Analyzer ", bg = "white", fg="#a869d0", font=('underline', 20,'bold') ).place(x=500, y=50)
14 frame = tk.Frame(root, bg= "#857bf5")
15 frame.place(relwidth=0.3, relheight=1.0)
16 Label1 = tk.Label(root, text="URL: ",bg="white",font=('underline', 20, 'bold')).place(x=500, y=260)
17 e=tk.Entry(root, width=125, bg="#cab9fb", borderwidth=5)
18 e.place(x=500, y=300)
19
20
```

```

Review-3 Project Exhibition.py > XSS
21 #XSS backend code starts
22 import requests
23 from pprint import pprint
24 from bs4 import BeautifulSoup as bs
25 from urllib.parse import urljoin
26 def get_all_forms(url):
27     """Given a `url`, it returns all forms from the HTML content"""
28     soup = bs(requests.get(url).content, "html.parser")
29     return soup.find_all("form")
30 def get_form_details(form):
31     """
32     This function extracts all possible useful information about an HTML `form`
33     """
34     details = {}
35     # get the form action (target url)
36     action = form.attrs.get("action", "").lower()
37     # get the form method (POST, GET, etc.)
38     method = form.attrs.get("method", "get").lower()
39     # get all the input details such as type and name
40     inputs = []
41     for input_tag in form.find_all("input"):
42         input_type = input_tag.attrs.get("type", "text")
43         input_name = input_tag.attrs.get("name")
44         inputs.append({"type": input_type, "name": input_name})
45     # put everything to the resulting dictionary
46     details["action"] = action
47     details["method"] = method
48     details["inputs"] = inputs
49     return details

```

```

Review-3 Project Exhibition.py > XSS
50 def submit_form(form_details, url, value):
51     """
52     Submits a form given in `form_details`
53     Params:
54         form_details (list): a dictionary that contain form information
55         url (str): the original URL that contain that form
56         value (str): this will be replaced to all text and search inputs
57     Returns the HTTP Response after form submission
58     """
59     # construct the full URL (if the url provided in action is relative)
60     target_url = urljoin(url, form_details["action"])
61     # get the inputs
62     inputs = form_details["inputs"]
63     data = {}
64     for input in inputs:
65         # replace all text and search values with `value`
66         if input["type"] == "text" or input["type"] == "search":
67             input["value"] = value
68         input_name = input.get("name")
69         input_value = input.get("value")
70         if input_name and input_value:
71             # if input name and value are not None,
72             # then add them to the data of form submission
73             data[input_name] = input_value
74
75     print(f"[+] Submitting malicious payload to {target_url}")
76     print(f"[+] Data: {data}")

```

```

Review-3 Project Exhibition.py > XSS
77     if form_details["method"] == "post":
78         return requests.post(target_url, data=data)
79     else:
80         # GET request
81         return requests.get(target_url, params=data)
82 def scan_xss(url):
83     """
84     Given a 'url', it prints all XSS vulnerable forms and
85     returns True if any is vulnerable, False otherwise
86     """
87     # get all the forms from the URL
88     forms = get_all_forms(url)
89     print(f"[+] Detected {len(forms)} forms on {url}.")
90     js_script = "<Script>alert('hi')</scripT>"
91     # returning value
92     is_vulnerable = False
93     # iterate over all forms
94     for form in forms:
95         form_details = get_form_details(form)
96         content = submit_form(form_details, url, js_script).content.decode()
97         if js_script in content:
98             print(f"[+] XSS Detected on {url}")
99             print(f"[*] Form details:")
100             pprint(form_details)
101             is_vulnerable = True
102             label2 = tk.Label(root, text="XSS Vulnerability:Detected \n Code Quality: Illegitimate",fg="Red", bg="white",font=('underline', 20, 'bold')).place(x=650, y=450)
103             # won't break because we want to print available vulnerable forms
104         else:
105             label3 = tk.Label(root, text="XSS Vulnerability:Not Found \n Code Quality: Legitimate",fg="Green", bg="white",font=('underline', 20, 'bold')).place(x=650, y=450)
106
107     return is_vulnerable
108 def XSS():
109     if __name__ == "__main__":
110         url = e.get()
111         print(scan_xss(url))
112
113 #XSS backend code ends

```

```

116 #SQLIA backend code starts
117 import requests
118 from bs4 import BeautifulSoup
119 from urllib.parse import urljoin
120 from pprint import pprint
121
122 # initialize an HTTP session & set the browser
123 s = requests.Session()
124 s.headers["User-Agent"] = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.106 Safari/537.36"
125
126 def get_all_forms(url):
127     """Given a 'url', it returns all forms from the HTML content"""
128     soup = BeautifulSoup(s.get(url).content, "html.parser")
129     return soup.find_all("form")
130

```

```

Review-3 Project Exhibition.py > ...
131
132 def get_form_details(form):
133     """
134     This function extracts all possible useful information about an HTML `form`
135     """
136     details = {}
137     # get the form action (target url)
138     try:
139         action = form.attrs.get("action").lower()
140     except:
141         action = None
142     # get the form method (POST, GET, etc.)
143     method = form.attrs.get("method", "get").lower()
144     # get all the input details such as type and name
145     inputs = []
146     for input_tag in form.find_all("input"):
147         input_type = input_tag.attrs.get("type", "text")
148         input_name = input_tag.attrs.get("name")
149         input_value = input_tag.attrs.get("value", "")
150         inputs.append({"type": input_type, "name": input_name, "value": input_value})
151     # put everything to the resulting dictionary
152     details["action"] = action
153     details["method"] = method
154     details["inputs"] = inputs
155     return details

```

```

Review-3 Project Exhibition.py > ...
156 def is_vulnerable(response):
157     """A simple boolean function that determines whether a page
158     is SQL Injection vulnerable from its `response`"""
159     errors = {
160         # MySQL
161         "you have an error in your sql syntax;",
162         "warning: mysql",
163         # SQL Server
164         "unclosed quotation mark after the character string",
165         # Oracle
166         "quoted string not properly terminated",
167     }
168     for error in errors:
169         # if you find one of these errors, return True
170         if error in response.content.decode().lower():
171             return True
172     # no error detected
173     return False

174 def scan_sql_injection(url):
175     # test on URL
176     for c in "\'\"":
177         # add quote/double quote character to the URL
178         new_url = f"{url}{c}"
179         print(f"[+] Trying: {new_url}")
180         # make the HTTP request
181         res = s.get(new_url)
182         if is_vulnerable(res):
183             # SQL Injection detected on the URL itself,
184             # no need to proceed for extracting forms and submitting them
185             print(f"[+] SQL Injection vulnerability detected, link: {new_url}")
186             label4 = tk.Label(root, text="SQL Injection Vulnerability:Detected \n Code Quality: Illegitimate",fg="Red", bg="white",font=('underline', 20, 'bold')).place(x=650, y=450)
187         else:
188             label5 = tk.Label(root, text="SQL Injection Vulnerability:Not Found \n Code Quality: Legitimate",fg="Green", bg="white",font=('underline', 20, 'bold')).place(x=600, y=450)
189     return
190
191 def SQLIA():
192     if __name__ == "__main__":
193         url = e.get()
194         scan_sql_injection(url)
195
196 #SQLIA backend code ends
197

198 #Phishing attack backend code starts
199 def Phishing():
200     #import libraries
201     import pandas as pd # use for data manipulation and analysis
202     from sklearn.linear_model import LogisticRegression # algo use to predict good or bad
203     from sklearn.model_selection import train_test_split # splitting the data between feature and target
204     from sklearn.metrics import confusion_matrix # gives info about actual and predict
205     from nltk.tokenize import RegexpTokenizer # regexp tokenizers use to split words from text
206     from sklearn.feature_extraction.text import CountVectorizer # create sparse matrix of words using regextokenizes
207     from sklearn.pipeline import make_pipeline # use for combining all preroessors techniues and algos
208     import pickle# use to dump model
209     import warnings # ignores pink warnings
210
211     warnings.filterwarnings('ignore')
212
213     #loading the dataset
214     phish_data = pd.read_csv('phishing_site_urls.csv')
215     pipeline_ls = make_pipeline(CountVectorizer(tokenizer = RegexpTokenizer(r'[A-Za-z]+').tokenize,stop_words='english'), LogisticRegression())
216
217     #training the datasets
218     trainX, testX, trainY, testY = train_test_split(phish_data.URL, phish_data.Label)
219     pipeline_ls.fit(trainX,trainY)
220     pipeline_ls.score(testX,testY)
221
222     con_mat = pd.DataFrame(confusion_matrix(pipeline_ls.predict(testX), testY),
223                             columns = ['Predicted:Bad', 'Predicted:Good'],
224                             index = ['Actual:Bad', 'Actual:Good'])
225
226
227     pickle.dump(pipeline_ls,open('phishing.pkl','wb'))
228     loaded_model = pickle.load(open('phishing.pkl','rb'))
229     x = e.get()
230     predict = [x]
231     result1=loaded_model.predict(predict)
232     print(result1)

```



```

print(result1)
if result1[0]=="bad":
    Label6 = tk.Label(root, text="Phishing Vulnerability:Detected \n Code Quality: Illegitimate ",fg="Red", bg="white",font=('underline', 20, 'bold')).place(x=650, y=450)
else:
    Label7 = tk.Label(root, text="Phishing Vulnerability:Not Found \n Code Quality: Legitimate ",fg="Green", bg="white",font=('underline', 20, 'bold')).place(x=650, y=450)
#Phishing attack backend code ends
def on_command():
    Label8 = tk.Label(root, text="
    return Label8
on_command =tk.Button(root, text = "Clear",padx= 50, pady = 10, fg="white",bg="#857bf5",font=('underline', 12, 'bold'),command=on_command)
on_command.place(x= 750, y = 350, height = 70, width = 200)

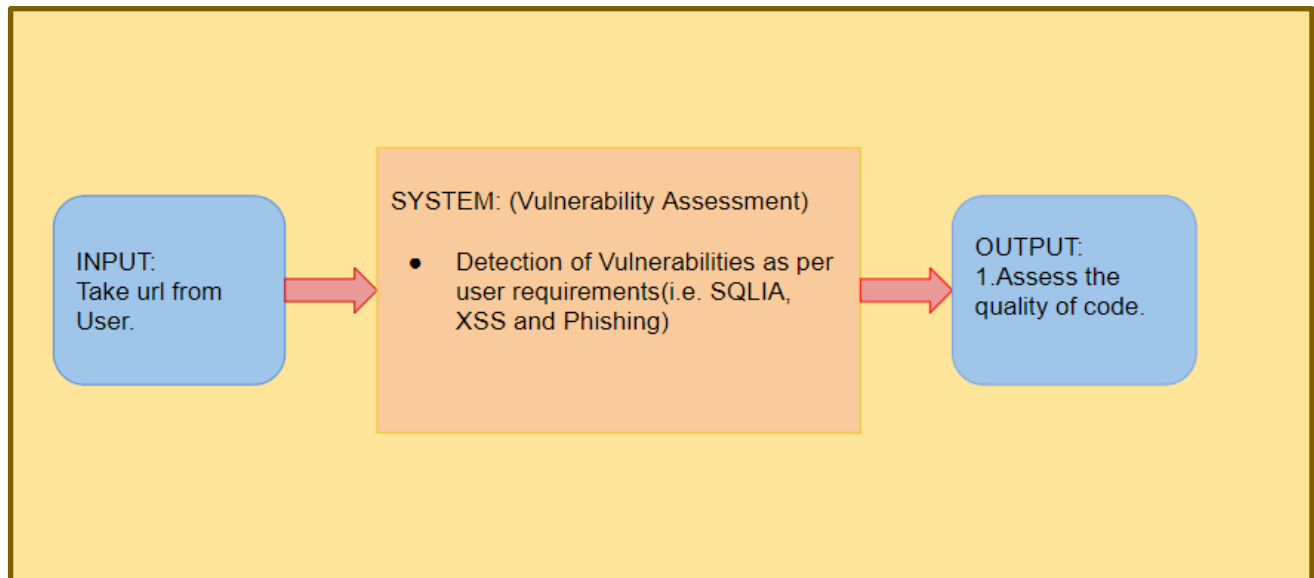
```

```

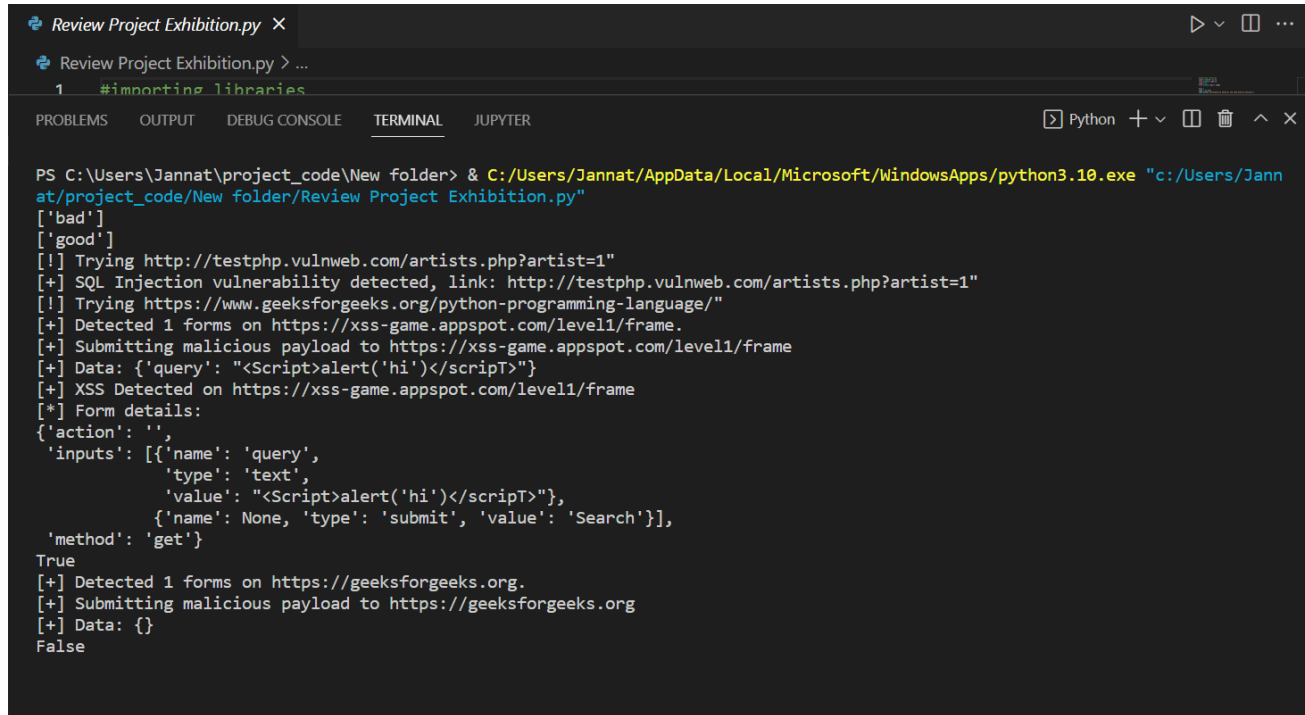
244
245 # GUI
246 XSS = tk.Button(frame, text="Cross-Site Scripting", padx=10,
247 | | | | | pady=5, fg="black", bg="white", font=('underline', 12, 'bold'), command=XSS)
248 XSS.place(x=100, y=100, height=60, width=200)
249
250 SQLIA = tk.Button(frame, text="SQL Injection", padx=26,
251 | | | | | pady=5, fg="black", bg="white", font=('underline', 12, 'bold'), command=SQLIA)
252 SQLIA.place(x=100, y=300, height=60, width=200)
253
254 Phishing = tk.Button(frame, text="Phishing", padx=38,
255 | | | | | pady=5, fg="black", bg="white", font=('underline', 12, 'bold'), command=Phishing)
256 Phishing.place(x=100, y=500, height=60, width=200)
257
258 root.mainloop()

```

5.3 PROTOTYPE SUBMISSION



5.4 TEST AND VALIDATION



```
PS C:\Users\Jannat\project_code\New folder> & C:/Users/Jannat/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/Jannat/project_code/New folder/Review Project Exhibition.py"
['bad']
['good']
[!] Trying http://testphp.vulnweb.com/artists.php?artist=1"
[+] SQL Injection vulnerability detected, link: http://testphp.vulnweb.com/artists.php?artist=1"
[!] Trying https://www.geeksforgeeks.org/python-programming-language/"
[+] Detected 1 forms on https://xss-game.appspot.com/level1/frame.
[+] Submitting malicious payload to https://xss-game.appspot.com/level1/frame
[+] Data: {'query': "<Script>alert('hi')</scripT>"}
[+] XSS Detected on https://xss-game.appspot.com/level1/frame
[*] Form details:
{'action': '',
 'inputs': [{'name': 'query',
               'type': 'text',
               'value': "<Script>alert('hi')</scripT>"},
             {'name': None, 'type': 'submit', 'value': 'Search'}],
 'method': 'get'}
True
[+] Detected 1 forms on https://geeksforgeeks.org.
[+] Submitting malicious payload to https://geeksforgeeks.org
[+] Data: {}
False
```

As we can see from the above image, our tool is working properly.

5.5 Performance Analysis

COMPARISON TABLE

<u>Sr. No</u>	<u>Type of Web Vulnerability detected.</u>	<u>URL (To be tested)</u>	<u>Is the resultant outcome similar to expected output?</u>	<u>Time taken for execution (seconds)</u>
1.	XSS	https://www.veenaworld.com/	Yes	2.07 s
2.	XSS	https://xss-game.appspot.com/level1/frame	Yes	2.10 s
3.	SQLIA	http://testphp.vulnweb.com/artists.php?artist=1	Yes	1.37 s
4.	SQLIA	https://www.geeksforgeeks.org/python-	Yes	0.91 s

		programming-language/		
5.	Phishing	https://youtube.com/watch?v=qI0TQJI3vdU	Yes	25.62 s
6.	Phishing	https://fazan-pacir.rs/temp/libraries/ipad	Yes	20.59 s

5.6 SUMMARY

This chapter describes the technical implementation and prototype of our project. Our tool is also validated and tested. The comparative analysis of our project shows that scanning of XSS and SQLIA takes less than 2 seconds. Phishing Attack, as built on ML model takes more time, that is around 30 s to 1 minute. Hence, our tool is effective and efficient.

CHAPTER – 6

PROJECT OUTCOME AND APPLICABILITY

6.1 OUTLINE

A Website vulnerability scanner first scans the entire website, analyzing in-depth each file it finds available, and displays the entire website structure. It scans your website application, following all links within the scope of your starting URLs, and attempts to exercise as many user inputs and event handlers as possible.

6.2 SIGNIFICANT PROJECT OUTCOMES

- Vulnerability and threat scanning
- low rate of false positive
- Analyzes code quality
- faster identification speed
- Improved Productivity.
- Available at low cost.

6.3 PROJECT APPLICABILITY ON REAL-WORLD APPLICATIONS

Imagine that you get an Email stating that your FaceBook account has been blocked due to some problems and you need to unlock your account providing necessary information and personal details such as your password. That Email directs you to a FaceBook lookalike and then it proceeds to steal your delicate information. To prevent these kinds of fraud and malicious acts you are provided this webpage vulnerability detector. You only need to copy and paste the URL for any website that you need assurance for and the detector will do the job for you in a matter of seconds. This enables us to check any website that may or may not be a threat to a user.

Our Website Vulnerability scanner usually performs these basic functions:

- Scanning known vulnerabilities such as XSS, SQLIA and Phishing.
- Website blacklisting detection.

- Security analytics to strengthen security.
- Distinguishes between legitimate and illegitimate websites.

6.4 INFERENCE

This chapter provides a high-level introduction to methods and tools of a web scanner. A website vulnerability detector is a kind of software used to prevent, scan and detect a malicious threat from a website being provided its URL. Website vulnerability scanners enable organizations to continuously identify vulnerabilities by crawling the website and its diverse parts, including web pages, third-party components, and software. We plan to develop a specification for website vulnerabilities scanners.

CHAPTER – 7

CONCLUSIONS AND RECOMMENDATION

7.1 OUTLINE

We defined website scanners and presented some vulnerabilities that this class of tools should detect. We plan to develop a specification for website scanners. Web applications have always played an important role in an organization being a gateway to organization's imperative information. The use of web applications has become conspicuously popular in recent years. This has led to the accumulation of valuable and sensitive data. This phenomena has led to attackers constantly looking for vulnerabilities, and has opened up a new battlefield for the developers.

7.2 LIMITATION/CONSTRAINTS OF THE SYSTEM

Most of the scanners detect vulnerabilities like SQL Injection, XSS, File inclusion, Path traversal, hidden pages & files and web server vulnerabilities. But sometimes, the web security scanner won't work in some cases. These are the few reasons why web security scanners fail to detect vulnerabilities.

- It lacks internet protection.
- It lacks real-time protection.
- Absence of dynamic features.
- Instruction is needed for every step.
- User interaction is mandatory while it is executing.

7.3 FUTURE ENHANCEMENTS

After understanding limitations of scanner, there is no efficient scanner to defense web application vulnerabilities, and although each scanner might limited to few attacks so in order to overcome scanners limitation and to prevent web application vulnerability an efficient mechanism must be introduced, overcome scanner limitations is future enhancement in future

proposing Hybrid framework , which will integrate existed scanner functionality and overcoming above mentioned limitations.

7.4 INFERENCE

The basic idea is to scan any website or and URL in a very fast manner and save the time of the user. Our objective is to develop a field of security for the users so as to keep their personal information secret and hidden from the hands of a hacker. Our tool can finish scanning a website for vulnerabilities in a minute and provide the user with a simple answer ‘Legitimate’ or ‘Illegitimate’. The three vulnerabilities that are covered in our project are SQL Injection Attacks as they are commonly used to manage and direct information on applications that hackers have come up with ways to slip their own SQL commands into the database. These commands may change, steal or delete data, and they may also allow the hacker access to the root system. Cross-Site Scripting (XSS) attacks are one of the most common ways an attacker can deploy a cross-site scripting attack is by injecting malicious code into an input field that would be automatically run when other visitors view the infected page. For example, they could embed a link to a malicious JavaScript in a comment on a blog.

APPENDIX A

Phishing

Phishing is a type of social engineering attack often used to steal user data, including login credentials and credit card numbers. It occurs when an attacker, masquerading as a trusted entity, dupes a victim into opening an email, instant message, or text message. The recipient is then tricked into clicking a malicious link, which can lead to the installation of malware, the freezing of the system as part of a ransomware attack or the revealing of sensitive information.

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a type of attack in which the attacker injects malicious scripts into web applications. An attacker uses XSS to send malicious script to an unsuspecting user and the end user's browser has no way to know that the script cannot be trusted and will execute the script. Unlike the other attacks, which involve two parties, the attacker and the target website, XSS involves three parties, attacker, client and target website. The attacker steals the cookies from the client and impersonates the client while attacking the target website. This is achieved by malicious JavaScript code running from the web browser of the client and with the access privileges of the client.

SQL Injection

An SQL Injection is a code injection technique where the attacker injects malignant code into the strings retrieving imperative information from the web server database. The attackers use SQL related keywords in their code which results in manipulation of users data, authentication bypass, denial of access which can further lead to destruction of the database. Some of the proposed solutions include such as avoiding connecting to a database with superuser access, avoiding using dynamic SQL queries, use encryption techniques or hash format for sensitive data. It is also advised

not to reveal more information in error messages instead use custom error messages for displaying minimal information.

GUI (Graphical User Interface)

A graphical user interface (GUI) is a type of user interface through which users interact with electronic devices via visual indicator representations. Graphical user interfaces would become the standard of user-centered design in software application programming, providing users the capability to intuitively operate computers and other electronic devices through the direct manipulation of graphical icons such as buttons, scroll bars, windows, tabs, menus, cursors, and the mouse pointing device. Many modern graphical user interfaces feature touch screen and voice-command interaction capabilities.

Scanners

Web applications have always played an important role in an organization being a gateway to organization's imperative information. Nowadays, hackers are always active and look for attacking data which leads to developing security testing applications or web security scanners. A web application security scanner is a software program which performs automatic black box testing on the web application and identifies security vulnerabilities. These scanners do not access source code instead they go for functional testing and find the vulnerabilities in web applications. These are widely used by security auditors and web application administrators because of their easy usability by performing tests and identifying vulnerabilities.

REFERENCES

1. <https://owasp.org/www-project-top-ten/>
2. <https://www.whitehatsec.com/>
3. <https://www.indusface.com/blog/what-is-a-website-vulnerability-and-how-can-it-be-exploited/>
4. <https://www.geeksforgeeks.org/how-to-build-a-sql-injection-scanner-in-python/>
5. <https://www.kaggle.com/datasets/taruntiwarihp/phishing-site-urls>
6. <https://www.thepythoncode.com/article/make-a-xss-vulnerability-scanner-in-python>
7. <https://www.geeksforgeeks.org/libraries-in-python/>
8. <https://docs.python.org/3/library/os.html#module-os>
9. <https://docs.python.org/3/library/sys.html#module-sys>