Final Project Report

Automated Script - Cross Site Scripting Vulnerability Scanner

Viharika Deverapally

Sowmika Lolla

Chirag Jain

Department of Information and Technology, Arizona State University

IFT 520: Advance Information Security

Dr. Jim Helm

November 29th, 2022

Contents

Abstract 3

Introduction 4

- **Problem Statement** 6
- **Significance of Study** 8
- **Objectives** 9

Design Process 10

- **Requirements** 10
- **Scope** 10

Development Process 11

- **Tools** 12
- **Technical Description of Project** 14

Testing and Results 21

Summary and Conclusions 26

References 27

Appendix 28

Abstract

This project is all about a web-based cross-site scripting vulnerability scanner designed primarily for penetration testers, security workers inside a company, and freelancing testers who find vulnerabilities (XSS) in online applications. The scanner will be able to examine and detect a few forms of cross-site scripting vulnerabilities in any form of online application, whether HTTP or HTTPS and will inform us whether the web page is weak to cross-site scripting vulnerability or not.

Taking into consideration that not every website is vulnerable to XSS, this code only examines a few parameters. We put in enough work and study for this project to comprehend and test a few parameters. Most of the parameters must be verified for a precise outcome.

It will consist of a script built entirely in Python 3 and compatible with several operating systems, making it available for every OS user. The terminal output has been cleanly programmed to generate information that can be readily comprehended, reducing ambiguity during the evaluation process, as opposed to other scanners that produce data that is difficult to grasp and requires time to go through the lines and find the problem presented. With the success of this project, such incidents that might result in the loss of personal user information being compromised could be successfully addressed by the usage of this scanner.

Introduction

Web applications offer users a variety of services and typically have excellent levels of usability. Web apps have been a popular target for cybercriminals since users are regularly required to submit private information in order to complete delicate online operations (such as financial transactions). Cross-site scripting (XSS) cyberattacks against web applications have significantly risen in recent years in this area.

Recent trends in data framework security demonstrate a significant growth of Cross-Site Scripting (XSS) flaws. A determined aggressor could exploit knowledge of XSS flaws to access a control framework organization due to the intermixing of controlling framework technology and data framework innovation. In XSS, harmful Web programming instructions are posted to a publicly accessible place online against the wishes of the area's owners. These instructions take advantage of functionality built into Web applications or other scriptable apps (like email clients), which view a site or manage Internet routes that cause content execution when the malicious website is viewed or the malicious link is clicked.

Although XSS can be done quickly, a successful attempt to attack a control system using this vector would take time and effort, depending on the aggressor's knowledge and motivation.

By taking advantage of vulnerabilities in web applications, XSS allows an attacker to run arbitrary code uninvited by the web application. In this way, a user who is not paying attention can become a victim of identity theft, electronic fraud, or other types of cybercrime.

The three main, clearly distinct types of XSS attacks are reflected-XSS, stored-XSS, and DOM-based XSS. These methods vary from one another in how they are able to introduce dangerous code into the application and how this code is carried out.

Attacks that use stored and reflected XSS take advantage of holes in online applications. These attacks insert script code into an HTTP request, typically as a web form parameter or input. Because the script is part of the answer to the HTTP request, it is executed immediately in the victim's browser in reflected attacks. The aim of stored-XSS attacks, in contrast, is to inject the malicious or unintended scripts in a persistent manner. In this approach, a vulnerability can be exploited simply once, and the injected script will run each time the corresponding web page is accessed.

This project, "Online applications XSS vulnerability scanner," will be a tool that will be very useful in the area of cyber security since it will help develop simpler and more adaptable ways to address cross-site scripting vulnerabilities. Ethical hackers will be able to locate the exact location of an XSS vulnerability on a website and detect it promptly. With such a tool, testers will also find it simpler to alert web developers to address the vulnerability, saving time and simplifying testing and routine maintenance.

Problem Statement

A fraction of the web apps are produced by less professional counseling software engineers who have little to no knowledge of security, irrespective of the fact that the web foundations is created by professional developers who have security concerns in their minds. This exposed the web application's several flaws and opened doors for online criminals to increase unauthorized access to sensitive data.

Cross-site scripting attacks have predominated as a result of attackers taking advantage of this flaw to obtain sensitive information from websites all over the world. The Aurelia framework's default HTML sanitizer was vulnerable to XSS attacks after a vulnerability that might provide attackers an advantage for subsequent exploitation was discovered.

Attackers could choose from a wide range of different components to tie, insert malicious code, and launch an XSS attack because the HTML sanitizer will by default only handle script elements during the sanitization interaction. This is a typical illustration of a platform that people use in their daily lives without realizing that they are vulnerable to assaults and that their personal data may be disclosed out or even misused by others.

The primary goal of building this scanner was to create a tool that could scan and identify XSS vulnerabilities in web applications and clearly identify the source of the vulnerability, improving the security of websites that could be targeted by cyberattacks.

Each stage of the scanning process can now be completed concurrently with the others thanks to this design choice.

The following is a list of how the scanner works:

1. The web application is crawled by a website parser agent.
2. The preceding step's information about the various web forms is utilized to create a collection of potential injection locations.
3. The collection of injection points listed by the parser agent is read by a script injector agent.
4. Additionally, the script injector agent chooses various attack vectors from the payloads provided
5. Each of the application's possible points of attack is subjected to the chosen set of attack vectors.
6. A conducted attack list is a list of the attacks that were actually carried out.
7. The list of attacks that need to be validated is given to the verificator agent.
8. The verification agent searches the web application for each assault by crawling it.
9. A report on the scanning process' findings is created and saved

Web parser:

An agent that explores websites looks for probable injection points where stored-XSS attacks could be launched. Since it spreads around the website through following the hyperlinks it finds while carefully obtaining data from the sites it visits, this parsing technique is akin to that utilized by web crawlers and spiders. It differs from a typical web crawler, though, in two ways:

- It just examines the links that point to the scanned website using no external connections.

- Information is retrieved via web forms. Online forms have in fact been chosen as the point of entry for stored-XSS attacks because they are the main tool offered by web applications for entering new material that is regularly retained.

- Websites that transmit user information to other websites via online forms are also excluded. Links with parameters were frequently used to query the web application's database rather than to contribute new information, therefore they are not kept as entry points.

Significance of Study

Due to the nature of websites, they are exposed to various threats. The protection of the site and its underlying components is very important in order to maintain a secure and reliable web application. The vulnerability known as XSS allows arbitrary code execution to be performed on a site's JavaScript code. Attacks can be performed depending on the sensitivity of the data that's being handled by the site. There are many risks that are associated if a website is compromised with XSS such as session hijacking, identity theft, denial of service attacks and financial fraud. We need to secure our websites to protect any organization from these risks. A Web application scanner is an essential component of an organization's security strategy to prevent would-be hackers from accessing its data. Despite the numerous steps that companies have taken to improve the security of their web applications, they are still vulnerable to attacks due to the lack of proper security measures. This vulnerability has been exploited by hackers to 5 gain access to corporate data. Having a dedicated web application scanner is also important to prevent stealing of cookies and sensitive data leakage. Here we will be using web scraping methodology to build a scanner using various functions in python. We will face many challenges in this process, and we will learn something new to overcome them.

Objectives

The main goal is to create a web-based XSS vulnerability scanner which can parse through every link on a website and accurately identify XSS vulnerabilities, preventing attackers from taking advantage of the defect and putting user data at risk.

1. Creating Python code that can scan through a webpage and filter out all the forms and links that are vulnerable to XSS is one of the project's specific goals, to name just a few.

2. Effectively creating a script that can swiftly recognize the various XSS vulnerabilities, hence enhancing its use scoop.

3. incorporating Python modules into my code to aid in deep scanning that will be helpful in finding any hidden vulnerabilities

Design Process

Requirements

For the good functioning of this project, we need the following hardware and software requirements

Software Requirements:

- Windows
- Python 3.8 or above
- Requests and Beautiful Soup Library for web scraping
- Tkinter library to design UI in python

Hardware Requirements:

- Core i5 10th gen processor
- 8GB RAM
- 256GB Hard Disk

Scope

The tool was able to successfully identify and fix the XSS vulnerability. It can also be used to detect other related issues such as cross-site request forgery and SQL injection. Modifying the code according to the demand will help improve the detection of XSS and other vulnerabilities. This tool can also be used to scan more websites for these issues. The architecture of the proposed system is flexible and adaptable, which makes it ideal for analyzing any web app. It can be used by web managers and developers to enhance the security of their websites. It can also be utilized by external entities to perform web security audits. It can be used to detect and prevent XSS attacks.

Development Process

Existing Work

Various techniques have been studied to detect and prevent XSS vulnerabilities. Some of these include implementing secure programming guides and ensuring that the developers follow proper encoding procedures. The best practices for mitigating XSS vulnerabilities can be found in the cheat sheet published by Wyk and Graff. In order to automatically clean untrusted input, various techniques have been proposed using template languages. Unfortunately, these methods are not widely used in legacy web applications due to their technological limitations. One of the most common methods that is used to clean untrusted input is by implementing an abstract data type. This method, which is very expensive, comes with a 25% run-time overhead. Another technique known as ScriptGard is a run-time method that is similar to the ones used for Java and WASP for SQL injection. These methods are generally used to clean large scale legacy systems. They can also be performed on a path-sensitive basis using binary code instrumentation. However, they require a runtime component to perform their operations. This method can be very expensive to implement due to the changes that need to be made to existing web applications. This approach is generally used to clean web applications that are built using various web languages such as Java, HTML, and CSS. Unlike static analysis, which is commonly used to identify XSS vulnerabilities, our method does not require a runtime component. One of the main disadvantages of this method is that it does not evaluate the effectiveness of the sanitization functions.

Instead, dynamic analysis techniques are commonly used to analyze the response of an application to a certain type of error. They aim to find out which errors are most likely to occur in the context-sensitive environment of the application. In the case of black box testing, various algorithms have been investigated. In a paper, Duchene and colleagues proposed a method that is known as data flow aware fuzzing. This technique uses a state-aware crawler to collect data about an application's operations and then infers the data flow. The advantage of this method is that it can provide a more accurate view of stored XSS vulnerabilities. Due to the nature of crawling-based inference, it can lead to high false negatives when it comes to identifying hidden XSS vulnerabilities. In contrast, source code analysis can be used to extract all execution paths from the data flow.

In a paper, the authors of the proposed black-box vulnerability scanner proposed a method that can increase the test coverage by leveraging the activities of the user. However, this method can lead to false negatives due to the lack of confidence in the effectiveness of the method. For instance, a blog site can allow users to use HTML tags as input. However, if they are not allowed to use HTML encoding functions, they can't prevent unwanted JavaScript programs from running in the body context. There are also heuristic filters that try to block these programs. However, they are not always reliable because many patches have been issued.

Tools

- Visual Studio Code
- IDLE
- Requests and Beautiful Soup Library for web scraping
- Tkinter library to design UI in python
- Lucid Chart

Visual Studio Code:

The powerful and lightweight Visual Studio Code is a free source code editor that can run on your desktop or on the web. It can be used for Windows, macOS, Raspberry Pi, and Linux. It has built-in support for various programming languages, such as JavaScript, Node.js, and TypeScript. It also has a variety of extensions for other languages, such as C++, Java, Python, and Go, as well as for environments, such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure. The goal of Visual Studio Code is to be lightweight and start quickly. It features a variety of powerful editing tools and features, such as multi-cursor editing, code completion, graphical debugging, and refactoring. Some of these were adapted from the technology used in Visual Studio.

IDLE:

IDLE is a part of the Python learning environment. It makes it easy for programmers to write code. It can be used to create, modify, and execute various types of scripts. With IDLE, programmers can create fully-featured Python scripts that feature smart indent, autocompletion, and syntax highlighting. It also has a debugger that has a variety of features, such as stepping and breakpoints.

Beautiful Soap Library:

The Beautiful Soup library is a Python program that can be used to pull data from XML and HTML files. It can create a tree from the source code and extract data in a more readable and hierarchical manner.

Tkinter Library:

The Tkinter framework is the de facto way to create Graphical User Interfaces in Python. It's the only framework that's built into the standard library of the platform. The Tkinter framework is a thin layer that acts as an interface to the Tk toolkit. The toolkit is a collection of graphical control elements, which are commonly used to build application interfaces.

Lucid Chart:

The Lucid suite is the only visual collaboration tool that enables teams to visualize and build the future. It’s cloud-based and allows them to work from anywhere. Lucidchart is an intelligent diagramming app that helps teams align their insights and clarify complex concepts. Lucidchart makes it easy to align with your teammates and visualize complex ideas. Teams can innovate 38% faster with Lucidchart, and they can eliminate around 2.5 meetings per week. It also lets you create diagrams, org charts, and flowcharts.

Technical Description of Project

System Design:

The system design stage is a process that involves overcoming any issues that can arise between the current system and the arrangement area. For instance, in the case of how to execute, this stage involves changing the record to an organization that can perform the operation.The various sub-exercises that occur during this stage are designed to accomplish the main objective of system improvement.

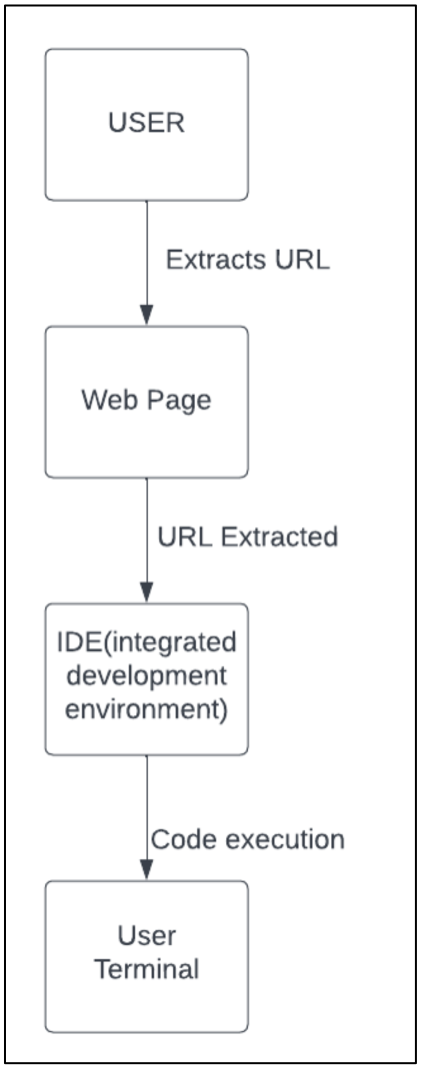
Logical Design:

A logical design is a unique plan that you can utilize to develop a variety of effective applications. Although you don't have the control over the execution details yet, you can still manage the kinds of data that you need to get the most out of your strategy. The process of designing a logical design involves orchestrating information into a series of connections that are referred to as substances and qualities.

Block Diagram:

A block diagram is a representation of a system's principal functions and parts. It shows the relationships between the various blocks by connecting them using lines.

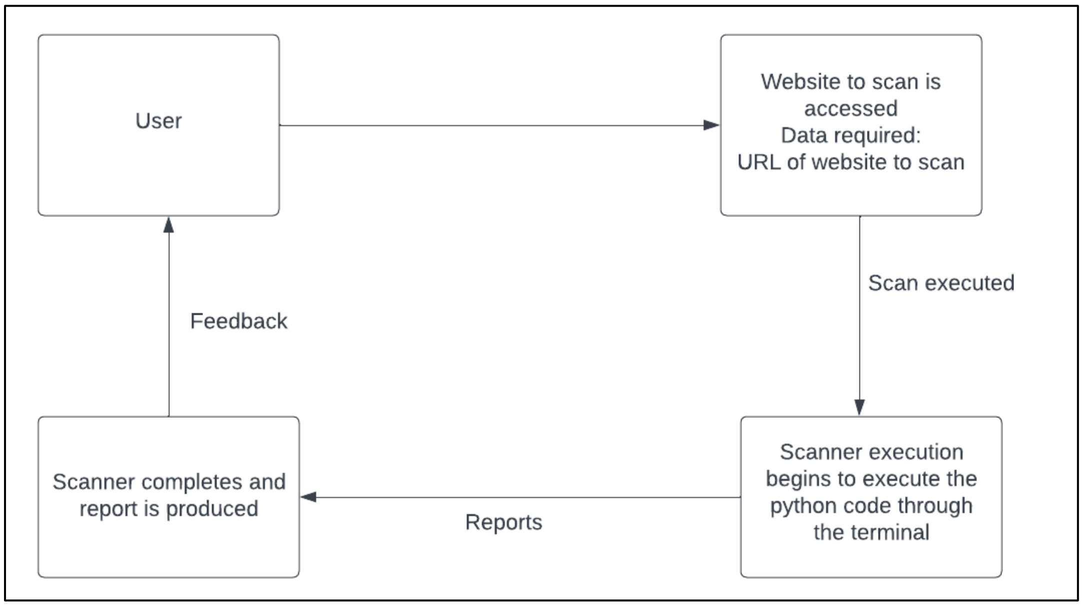
1. **USER**– The person who is interacting with a scanner is referred to as a user. They can be either an IT employee or a penetration tester.
2. **WEB PAGE**- This indicates that the website has been scanned using a scanner. This is the case with the trial website that was used for penetration testing.
3. **IDE** (Integrated Development Environment) - The scanner's environment is built on top of the Visual Studio IDE. This is the environment where the code is written.
4. **USER TERMINAL**- The user terminal is the place where the scanner runs. To successfully run the program, one has to access the project's directory and open the python file, which contains the main code.



Block Diagram

Data Flow Diagram:

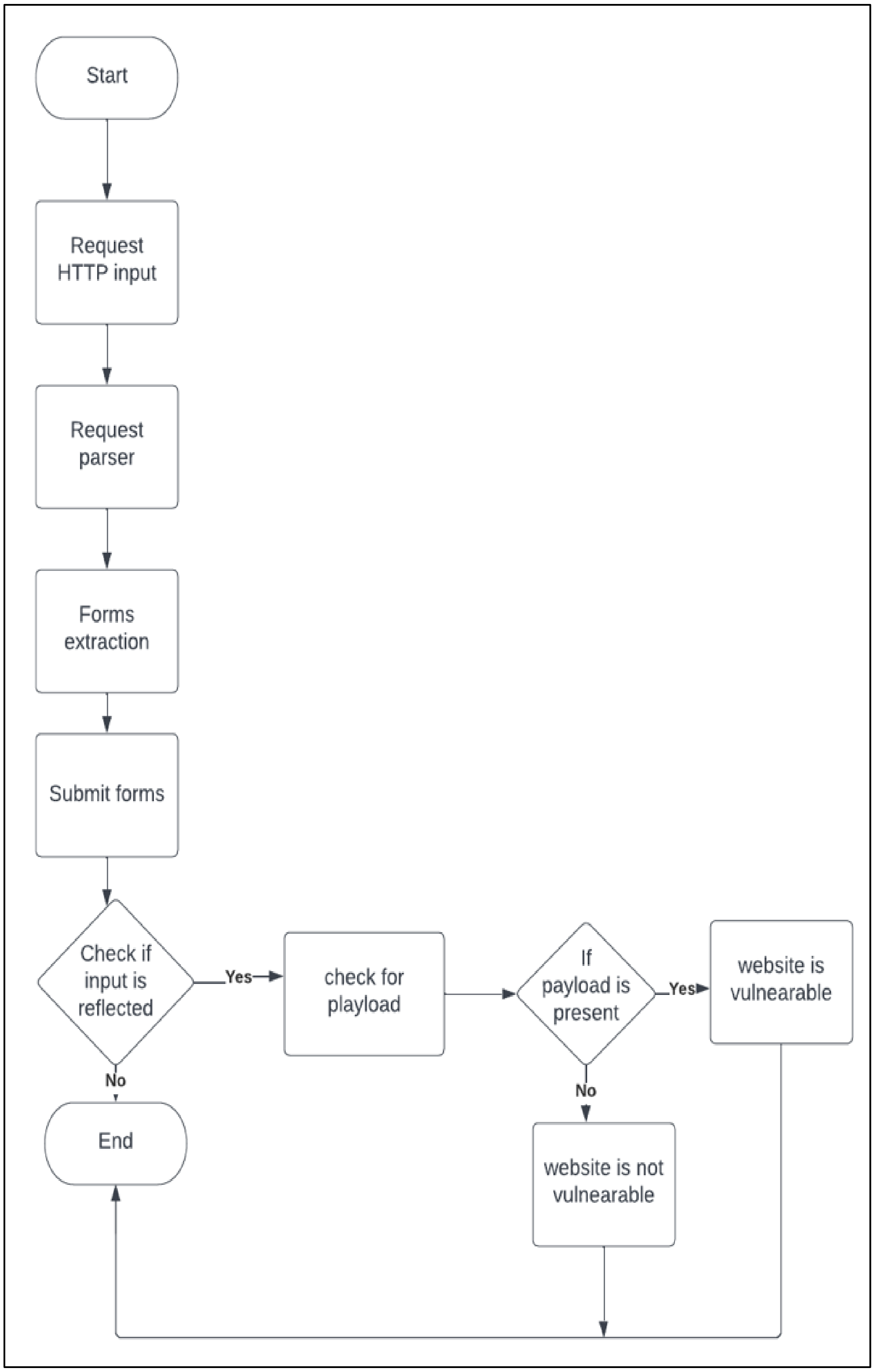
A data flow diagram is a representation of the flow of information that occurs in a system or process. The figure below shows the representation of the data flow by the scanner at the beginning and the end.


User
Website to scan is accessed
Data required:
URL of website to scan
Scan executed
Scanner execution begins to execute the python code through the terminal
Reports
Scanner completes and report is produced
Feedback

Data flow diagram

Flow chart:

Flow charts are typically graphical representations of steps. They show the various boxes of steps in a row or chart, and their order is connected by arrows. The figure below shows the steps taken by a scanner to identify a cross-site scripting vulnerability in a website.


Start
Request HTTP input
Request parser
Forms extraction
Submit forms
Check if input is reflected
Yes
No
End
check for playload
If payload is present
Yes
No
website is vulnearable
website is not vulnearable

Implementation:

In this project, we will develop an automated tool that will detect and report XSS on web apps. It will be used to modify and record flags and bug bounties. Since these kinds of vulnerabilities are commonly exploited in online forms and inputs, we will use JavaScript code to fill out the forms that we encounter. The code for this web app vulnerability scanner was written using python, which was compatible with the various modules I used for its development. The version of python that was used was 3.

The Beautifulsoup library is a python program that can be used to extract data from HTML and XML files. I was able to fetch my links from my scanner using the Urllib module, which is a component of the python framework. It provides a variety of functions and classes that allow me to perform various actions related to the web page's URL.

The **first step** here is to do the web scraping. We would be extracting forms from web pages and filling them out using the Beautiful Soup libraries and requests. Due to the vulnerability in web forms, we have created a function that will allow us to access all of them from the HTML content of our web page. This is done by taking a look at the various forms that are available on our web page.

Web scraping: Web scraping is a process that involves collecting data and content from the internet. This method usually involves saving the data in a local file so that it can be analyzed and manipulated later. When people refer to web scraping, they usually mean software applications. These programs visit websites and, in order to extract useful information, are programmed to go through the pages of the site. Web scraping bots can quickly gather large amounts of data due to how they can automate the process. The digital age has given us the opportunity to store and analyze vast amounts of information. Another popular open-source framework for web scraping is Beautiful soup, which is written in Python. It can create a tree that is capable of handling HTML data. It has multiple features that allow users to search, modify, and navigate through the tree.

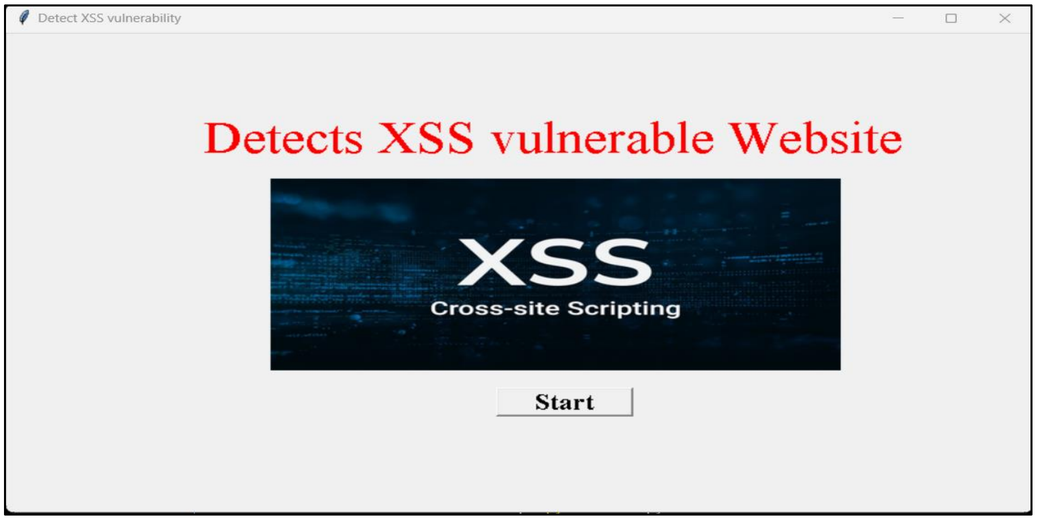
The program runs through all of the forms and submits them, as well as searching for input fields using JavaScript code. So, here we have implemented a function to submit the form. We will be providing form_data which is the output of get_form_data() function as an argument to the form_submit function as it contains all the various details of the form. This method accepts the original HTML form's URL and the value of every search field or text input. After getting the required information related to the forms, we need to submit the form using the requests.get() and requests.post() methods.

Now the next step is to find all the vulnerabilities through scanning. Here we have implemented a scan function namely xss_scanner(url). This program grabs all the HTML forms that are displayed in a web page and then prints the number of them that it has detected. It then iterates over the forms and returns the value of all the text and the input fields. If the Javascript code is successfully executed and injected into the web page, then it's a clear indication that the page is vulnerable to XSS.

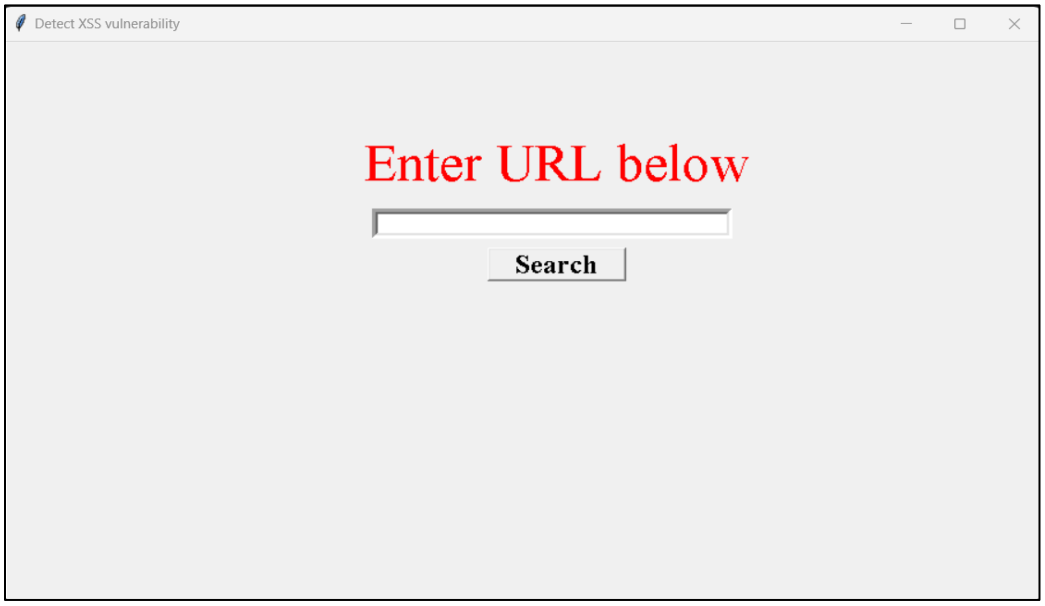
At last, we have implemented a UI for our tool. Here we used the Tkinter library to create the windows, add labels, buttons and images and implement more functionalities. The PIL library provides a variety of image processing tools. It can be used to create and save images. Compared to other frameworks,Tkinter is incredibly lightweight and easy to use. This makes it a great choice for building cross-platform applications, especially for those that require a modern sheen. The top priority is to create something that's both functional and beautiful.

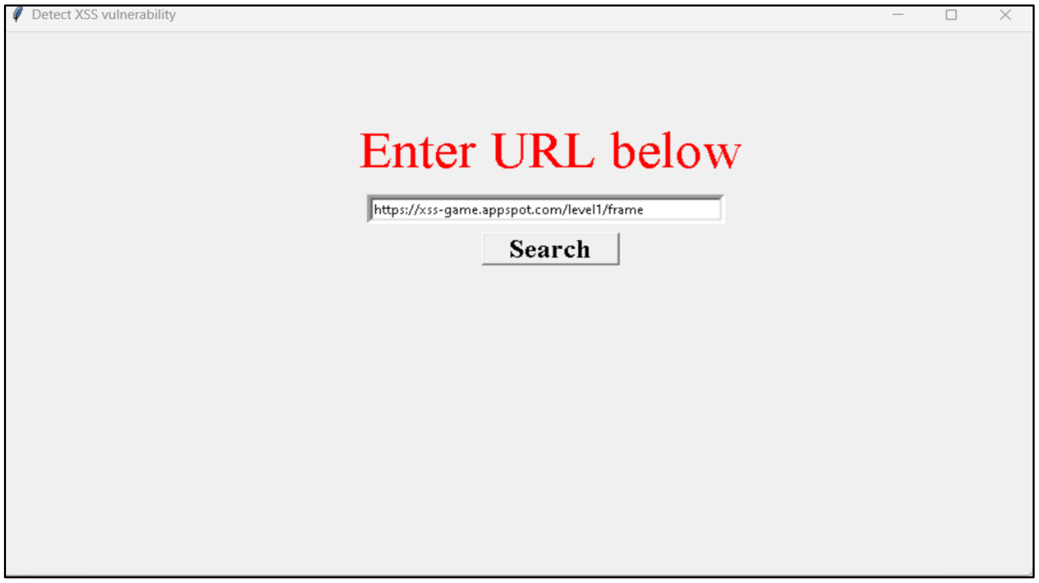
Testing and Results

1. This is the initial window of the scanner through which the user can start the application.

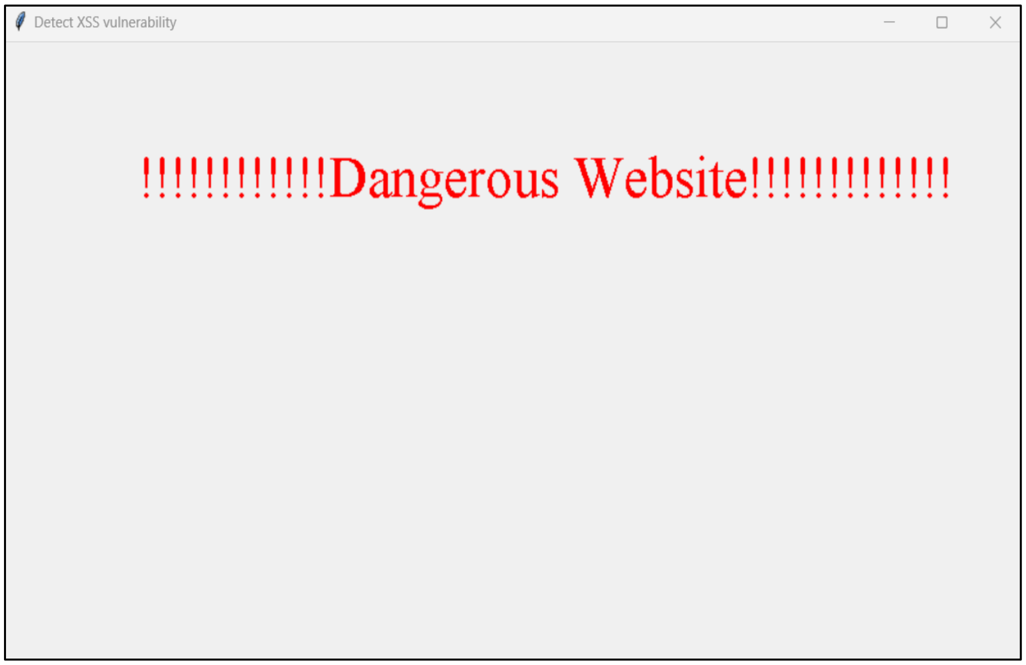


2. In the second window the user can give the input to the scanner which will be the url of the website of which the user wants to test the website for vulnerability



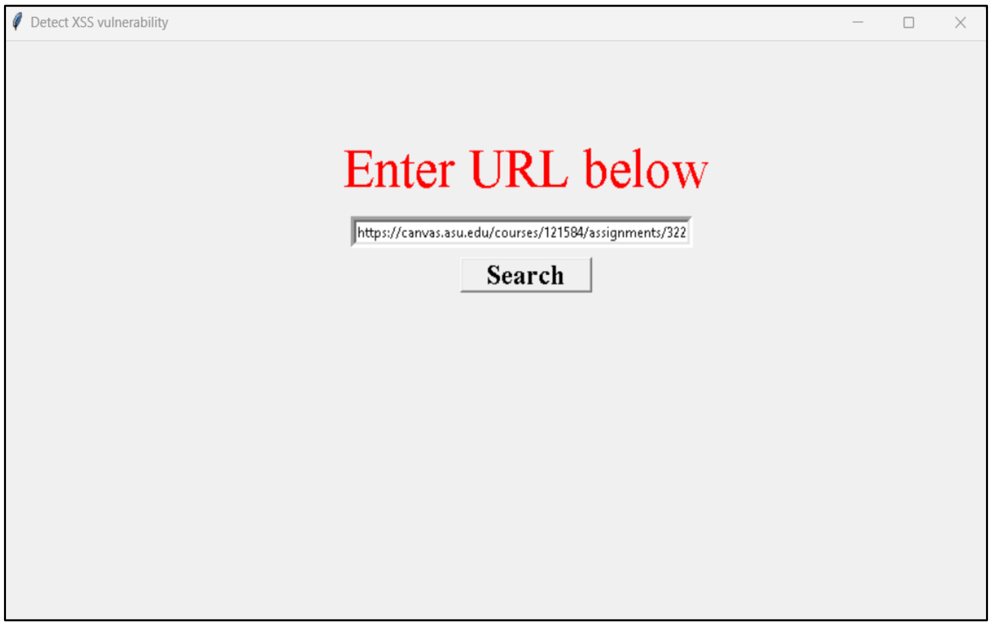


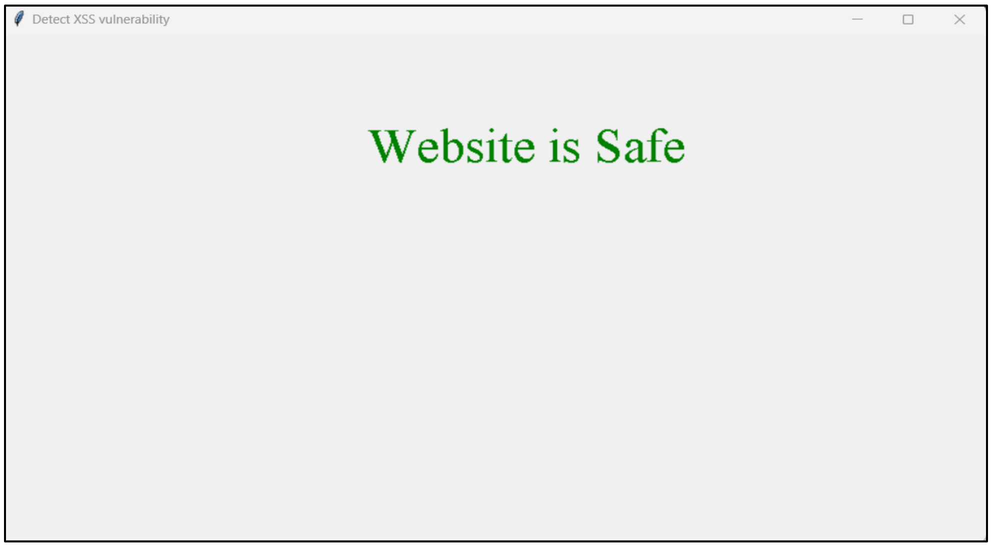
3. After the user gives the input, the scanner will start scanning the website for vulnerability using the payload and if the condition is satisfied and vulnerability is detected, then it will be displayed in the output screen as shown in the below image.



!!!!!!!!!!!!Dangerous Website!!!!!!!!!!!!!

4. After the user gives the input, the scanner will start scanning the website for vulnerability using the payload and if the condition is not satisfied, then there is no vulnerability and it will be displayed in the output screen as shown in the below image.

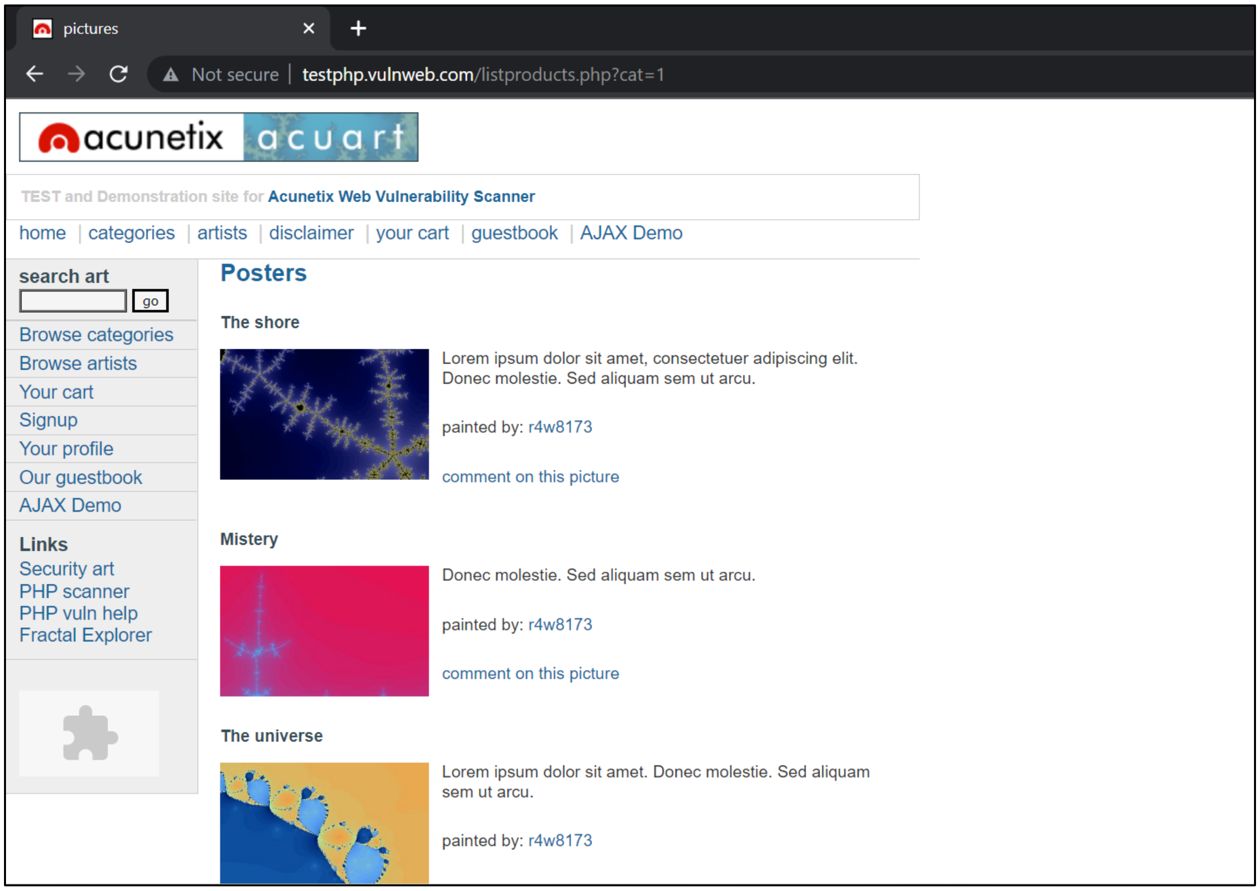




Website is Safe

To see if this script that we generated is working or not, we have manually injected malicious javascript payload to “http://testphp.vulnweb.com/listproducts.php?cat=1” website and observed that after executing the script a popup has been displayed proving that this website is vulnerable to XSS.

Original website that we are targeting is displayed below:


pictures
Not secure
testphp.vulnweb.com/listproducts.php?cat=1
acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go
Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo
Links
Security art
PHP scanner
PHP vuln help
Fractal Explorer

Posters

The shore

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie. Sed aliquam sem ut arcu.

painted by: r4w8173

comment on this picture

Mistery

Donec molestie. Sed aliquam sem ut arcu.

painted by: r4w8173

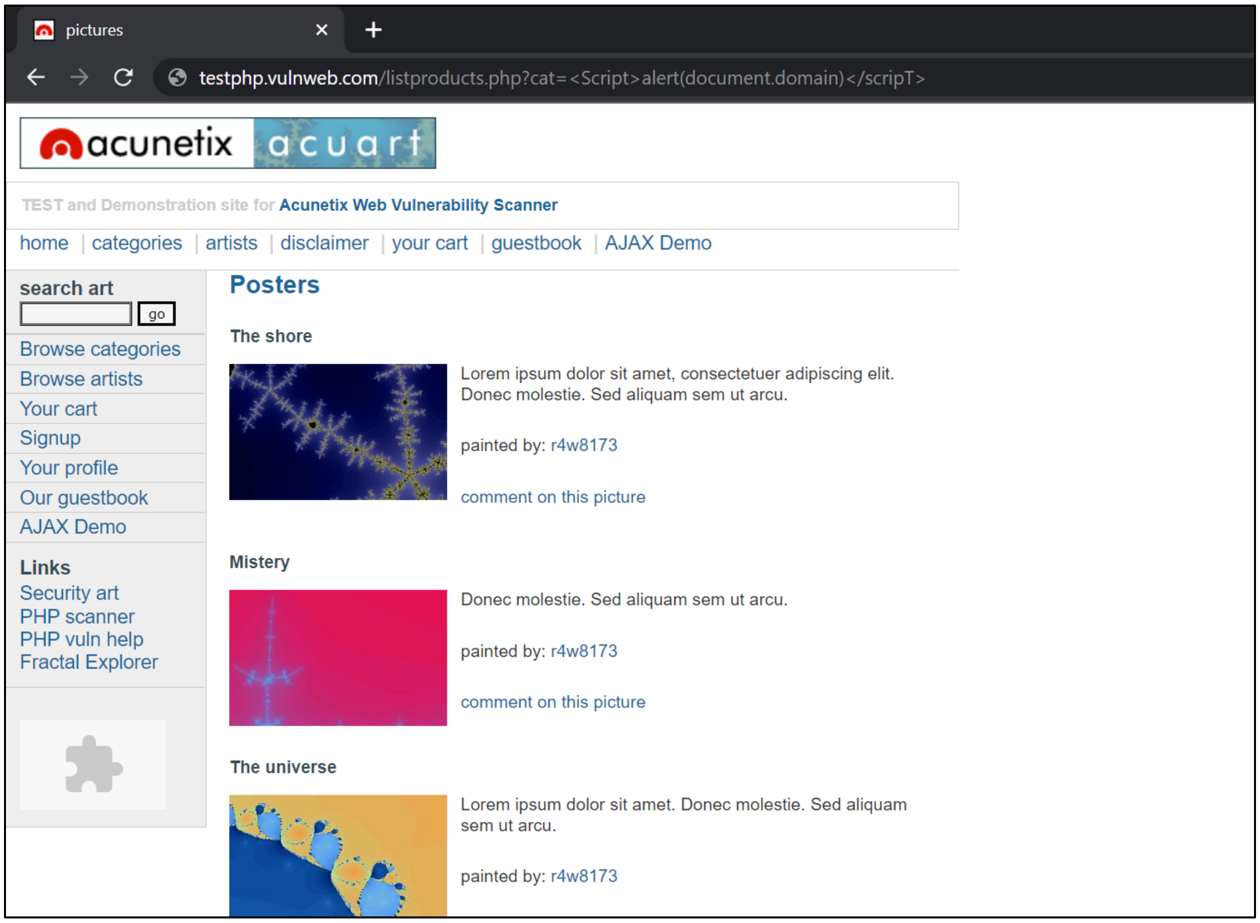
comment on this picture

The universe

Lorem ipsum dolor sit amet. Donec molestie. Sed aliquam sem ut arcu.

painted by: r4w8173

Injection java script payload "<Script>alert(document.domain)</scripT>" as shown below


pictures
testphp.vulnweb.com/listproducts.php?cat=<Script>alert(document.domain)</scripT>
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go
Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo
Links
Security art
PHP scanner
PHP vuln help
Fractal Explorer

Posters

The shore

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec molestie. Sed aliquam sem ut arcu.

painted by: r4w8173

comment on this picture

Mistery

Donec molestie. Sed aliquam sem ut arcu.

painted by: r4w8173

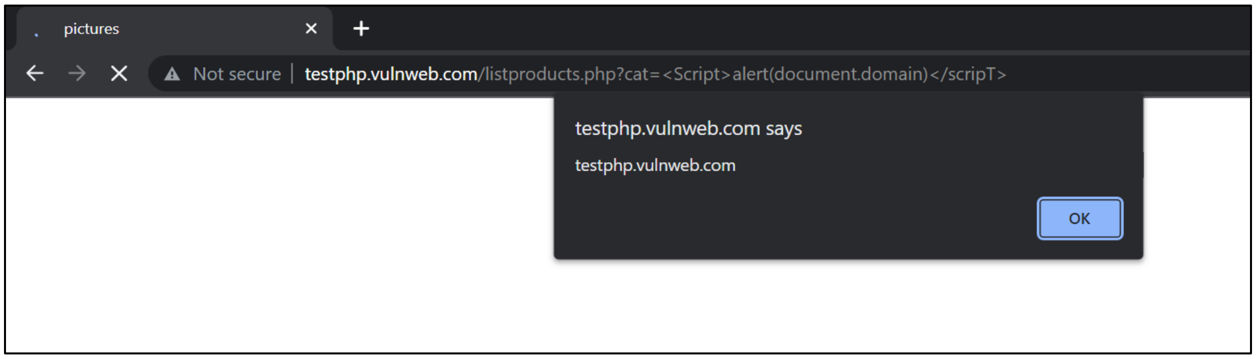
comment on this picture

The universe

Lorem ipsum dolor sit amet. Donec molestie. Sed aliquam sem ut arcu.

painted by: r4w8173

Hit enter and observe that the payload is executed, and the website is vulnerable. The injected javascript payload returns the domain of the website as a popup.


pictures
Not secure | testphp.vulnweb.com/listproducts.php?cat=<Script>alert(document.domain)</scripT>
OK

testphp.vulnweb.com says

testphp.vulnweb.com

Summary and Conclusions

The rise of the internet has created a virtual world where people can access faster services at any time which led to the proliferation of security concerns. Due to the increasing number of people and organizations using the internet, there are many security issues that need to be addressed and one among them is the XSS attack. Despite the various security measures that are available to prevent XSS, it still remains a threat until people become more aware of their privacy and security. In this project, we developed a web scanning tool that can help developers and testers identify and prevent XSS vulnerabilities in their web applications. A vulnerability scanner is a promising tool that can help prevent the spread of XSS vulnerabilities in web applications. A framework for implementing a vulnerability scanner was presented. The tool can be used by security professionals to identify and prevent web systems from being exploited by vulnerabilities.

References

Mack, J., Hu, Y.-H. (Frank), & Hoppa, M. (2019). A Study of Existing Cross-Site Scripting Detection and Prevention Techniques Using XAMPP and VirtualBox. Virginia Journal of Science, 70(3). *https://doi.org/10.25778/bx6k-2285*

Mohammadi, M., Chu, B., & Lipford, H. (n.d.). Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing. *https://arxiv.org/ftp/arxiv/papers/1804/1804.00755.pdf*

Galán, E., Alcaide, A., Orfila, A., & Blasco, J. (n.d.). A Multi-agent Scanner to Detect Stored-XSS Vulnerabilities. Retrieved November 30, 2022, from *https://core.ac.uk/download/pdf/30043269.pdf*

payloadbox/xss-payload-list. (2022, November 29). GitHub. *https://github.com/payloadbox/xss-payload-list/blob/master/Intruder/xss-payload-list.txt*

Li, C., Wang, Y., Miao, C., & Huang, C. (2020). Cross-Site Scripting Guardian: A Static XSS Detector Based on Data Stream Input-Output Association Mining. Applied Sciences, 10(14), 4740. *https://doi.org/10.3390/app10144740*

Appendix

SOURCE CODE

```
import requests

from bs4 import BeautifulSoup as bs

from urllib.parse import urljoin

import sys

from tkinter import *

from PIL import ImageTk, Image

import threading


main_windowScreen = Tk()


main_windowScreen.geometry("900x500")

main_windowScreen.title("Detect XSS vulnerability")



Label(main_windowScreen, text= "Detect XSS vulnerable Website", fg='red',font=("Times New
Roman",35)).place(x=30,y=80,relwidth=1)


#this takes image to add it on title page.

img = ImageTk.Image.open(r"D:\Downloads\xss.jpeg")

imag = img.resize((500,200), Image.ANTIALIAS)

cover_image= ImageTk.PhotoImage(imag)
```

```
imglabel = Label(main_windowScreen, image = cover_image).place(x=230,y=150)


#this function is responsible for generating UI pop up
def on_click():



    other_window = Toplevel(main_windowScreen)
    other_window.geometry("900x500")
    other_window.title("Detect XSS vulnerability")
    Label(other_window, text= "Enter URL below", fg='red',font=("Times New
Roman",35)).place(x=30,y=80,relwidth=1)

    url2=StringVar()
    url1 = Entry(other_window, width = 50, borderwidth =
5,textvariable=url2).place(x=320,y=150)

    def on_click1():
        output = Toplevel(other_window)
        output.geometry("900x500")
        output.title("Detect XSS vulnerability")
        out = Toplevel(other_window)
        out.geometry("900x500")
        out.title("Detect XSS vulnerability")
```

```
#requests module is used to send requests to paarticular url and then html content is parsed.
#below function fetches form details (params) from web pages.
def extract_forms(url):

    b_s_ = bs(requests.get(url).content, "html.parser")
    return b_s_.find_all("form")

#once it fetches html content from above function below function is responsible for getting attributes like action type, method like post or get..etc. returns them as soup objects (data_to_dictionary).
def get_form_data(form):

    data = {}

    f_attr_get_actn = form.attrs.get("action")
    action = f_attr_get_actn.lower()



    f_attr_get_m =form.attrs.get("method", "get")
    method = f_attr_get_m.lower()
```

```
            inpts = get_input_data(form)


            data_to_dictionary(action, data, inpts, method)
            return data


        def data_to_dictionary(action, data, inputs, method):
            data["action"] = action
            data["method"] = method
            data["inputs"] = inputs


    #below function creates list called all_inputs and then once it finds input type as text and name
it returns results to all_inputs.
        def get_input_data(form):
            all_inpts = []
            for tag_inpts in form.find_all("input"):
                inpt = tag_inpts.attrs.get("type", "text")
                inpt_name_ = tag_inpts.attrs.get("name")
                all_inpts.append({"type": inpt, "name": inpt_name_})
            return all_inpts


    #below is function to submit form


        def form_submit(form_data, url, value):
```

```
    dest_url = urljoin(url, form_data["action"])

    data = replace_values(form_data, value)

    return form_method(data, form_data, dest_url)

  def replace_values(form_data, value):
    inpts = form_data["inputs"]
    data = {}
    for inpt in inpts:

      if inpt["type"] == "search" or inpt["type"] == "text" or inpt["type"] == "email" or
inpt["type"] == "url":
        inpt["value"] = value
      inpt_name_ = inpt.get("name")
      inpt_val = inpt.get("value")

      if inpt_name_ and inpt_val:

        data[inpt_name_] = inpt_val
    return data
```

```
def form_method(data, form_data, target_url):
    if form_data["method"] == "post":
        return requests.post(target_url, data=data)
    else:
        return requests.get(target_url, params=data)
```

# this function checks no_of_forms on the target website for example like detected 1 form on target website to test for xss.

```
def xss_scanner(url):

    no_of_forms = extract_forms(url)
    print(f"[+] Detected {len(no_of_forms)} form(s) on URL: {url}.")
    with open('payloads.txt', 'r') as f:
        js_script = [line.strip() for line in f]
    #js_script = "<Script>alert(document.domain)</scripT> "

    is_vulnerable = False
    Label(output, text= "Website is Safe",
              fg='green',
              font=("Times New Roman",35)).place(x=30,y=80,relwidth=1)



    return detect_vuln(no_of_forms, is_vulnerable, js_script, url)
```

#this function triggers get_all_forms() and then if js_script gets reflected in response forms then it classifies it as malicious site other wise as safe website.

```
def detect_vuln(forms_, xss_vulnerable, js_script, url_):
    for form in forms_:
        form_data = get_form_data(form)
        for i in js_script:
            content = form_submit(form_data, url_, i).content.decode()
            if i in content:
                xss_vulnerable = True
                print(f"[+] XSS Detected on {url_}")
                print(f"[+] Form data:")
                print(form_data)
                print("Is XSS Vulnerable: ")
                Label(output, text= "!!!!!!!!!!!!Dangerous Website!!!!!!!!!!!!!",
                      fg='red',
                      font=("Times New Roman",35)).place(x=30,y=80,relwidth=1)



    return xss_vulnerable




if __name__ == "__main__":
```

```
            url=url2.get()
        #url_ = input("Enter the website URl to detect vulnerability: ");
            print(xss_scanner(url))
        other_window.mainloop()

    button = Button(other_window, text="Search",font=("Times New Roman",18,'bold'),
command=on_click1).place(x=420,y=185,width=120,height=30)



# https://xss-game.appspot.com/level1/frame
button = Button(main_windowScreen, text="Start",font=("Times New Roman",18,'bold'),
command=threading.Thread(target=on_click).start).place(x=430,y=370,width=120,height=30)

main_windowScreen.mainloop()
```