National College of Ireland

Hybrid Detection of Cross-Site Scripting (XSS) Vulnerabilities in Web Applications

MSc Industrial Internship
MSc Cyber Security

Yogesh Anandhakumar
Student ID: x23167998

School of Computing
National College of Ireland

Supervisor: Kamil Mahajan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Yogesh Anandhakumar
Student ID:	x23167998
Programme:	MSc Cyber Security **Year:** 2023-2024
Module:	MSc Industrial Internship
Supervisor:	Kamil Mahajan
Submission Due Date:	02/09/2024
Project Title:	Hybrid Detection of Cross-Site Scripting (XSS) Vulnerability in Web Applications
Word Count:	6247 **Page Count:** 19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Yogesh Anandhakumar

Date: 02/09/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Hybrid Detection of Cross-Site Scripting (XSS) Vulnerability in Web Applications

Yogesh Anandhakumar
x23167998

Abstract

Cross-Site Scripting (XSS) is a vulnerability for web applications and their security, by which attackers can inject malicious JavaScript into the system, leading to account takeover, session hijacking, or cookie stealing, among other issues. To address this, the research introduces a novel hybrid XSS detection approach and its implementation in the form of a tool named XSSFind. This tool integrates Static Application Security Testing with Dynamic Application Security Testing methodologies through white-box and black-box testing techniques to improve the discovery rate of cross-site scripting vulnerabilities. This work was motivated by existing tools that rely either on static or dynamic analysis but cannot provide comprehensive coverage. XSSFind offers the strength of both static and dynamic approaches, detecting vulnerabilities at the code level and runtime level-wise, thereby enabling a comprehensive security assessment. Results show that these combined approaches have given required results in terms of finding actual XSS vulnerabilities, making them a promising addition to the web security community. The findings further suggest directions for future improvement, such as extending payload libraries and detection models based on machine learning.

1 Introduction

Web application have been essential for daily life purposes such as online shopping, banking, and other services(Garcia-Alfaro and Navarro-Arribas 2007). Modern enterprise systems require their web applications for service delivery and user data management, but since most of these applications involves private information, the whole operation has to be as secure as possible(Garcia-Alfaro and Navarro-Arribas 2007). Rising dependence on web applications brought along a larger attack surface in the form of security vulnerabilities. Of these, XSS vulnerabilities are among the most dangerous and widespread (Hannousse, Yahiouche and Nait-Hamoud, 2024). According to Open Web Application Security Project (OWASP), in the case of XSS vulnerability, attackers can use malicious JavaScript on the behalf of victim in the browser of the victim. XSS vulnerabilities (Mahmoud et al. 2017) enable attackers to execute malicious scripts into web pages the victim has access to, and these codes are run in browsers, enabling many attacks such as data theft, session hijacking, or even changing the content of the affected page[1]. The newest version of OWASP Top 10[2] has seen XSS vulnerabilities rise from #A07 in the year 2017 to a higher position at #A03, as they present both high severity and prevalence over recent years. Although with solutions available, many web applications are still vulnerable to XSS vulnerabilities because the detection level is not advanced enough to detect these vulnerabilities[3]. Therefore, a unified tool that addresses all forms of XSS vulnerabilities can be highly advantageous. The aim of this research is to develop a generic tool that merges Static Application Security Testing (SAST) and Dynamic Application

[1] https://owasp.org/Top10/A03_2021-Injection/

[2] https://owasp.org/Top10/

[3] https://www.bleepingcomputer.com/news/security/high-severity-gitlab-flaw-lets-attackers-take-over-accounts/

Security Testing (DAST) to improve the detection of XSS vulnerabilities by leveraging strengths of both code-level and runtime-level analysis. SAST can identify issues within the source code without execution, while DAST involves interacting with the application in its operational or runtime environment.

1.1 Research Question

The research question posed in this thesis is -

"How does the integration of static & dynamic analysis security techniques enhance the detection of XSS vulnerabilities in web applications?"

1.2 Research Objectives:

The goal of this research has proposed a hybrid XSS detection tool that merges the strength of both SAST and DAST. The research objectives are as follows:

- Undertake a literature review to identify the implementations of static and dynamic analysis.
- Develop a hybrid method that integrates static and dynamic analysis to detect XSS vulnerabilities more effectively.
- Develop a tool to perform hybrid analysis and compare the tool against existing solutions, such as Snyk[4] for SAST and Burp Suite[5] for DAST, to assess its accuracy, and false positive rate.
- Design a user-friendly interface that simplifies the scanning process and generates clear and detailed reports.

This research contributes towards enhancing web application security by offering an effective solution for detecting XSS vulnerabilities.

2 Related Work

XSS vulnerability detection in web security has been a domain of wide research. Since then, different techniques of protecting against these attacks have been proposed, each presenting its strengths and weaknesses. This section critically reviews the existing literature of approaches to detecting XSS, where the main focus will be on static analysis (SAST), dynamic analysis (DAST), and techniques that use both approaches.

2.1 XSS Detection in Web Applications

Cross-Site Scripting vulnerability is a kind of Injection vulnerability that allows the attacker to inject their malicious code or payloads in a website that has XSS vulnerabilities(OWASP - XSS 2024). There are three types of XSS vulnerabilities: DOM-based XSS, Reflected XSS, and Stored XSS.

DOM-based XSS occurs when the JavaScript code of the web page directly uses the user input without a proper check. This allows the attacker to embed their own malicious code within a payload and transmit it to the unsuspecting victim. Once the victim opens it, the attacker's

[4] https://snyk.io/

[5] https://portswigger.net/burp

malicious code will execute directly in the user's browser, possibly carrying out harmful actions like session hijacking, redirects to malicious web pages, *etc*(Portswigger, 2024). In reflected XSS, an attacker sends the payload inside a URL to a victim; upon clicking by the victim, the payload is executed. This has also been called non-persistent XSS (Nidecki, 2024). Stored XSS is where the attacker crafts a payload and sends it to the Web application; then, the Web application stores it in its database. Then the payload will execute for everyone who comes on that particular payload injected webpage of the web application (TrendMicro, 2023).

Detection of XSS in Web applications is basically identifying vulnerabilities that can be used to inject the malicious scripts in the web pages, which can sometimes lead to session hacking and data theft (Cui, Cui and Hu, 2020). Tools like "XSS-Check" were developed by Jasmine Devi and George (2017) that would take the URL and file and inject payloads to discover the XSS vulnerability. The work on the study of XSS vulnerability was done in great detail by Di Lucca et al. (2004) showing the lacunas that might bring in the attack. Research has concluded that this XSS vulnerability is a critical issue and although many existing methods are here, still a lot of improvement is needed. According to Sarmah, U., Bhattacharyya, D.K., Kalita, J.K. (2018), a lot of improvements have come, and tools are created, but still a gap exists because emerging web vulnerabilities are getting complex. According to Liu et al. (2019) in their survey, they have differentiated the different types of methods to find XSS vulnerability. Broadly they can be classified as static, dynamic, and hybrid analysis techniques. Baranwal (2012) compared and analysed tools that can be used to mitigate SQL injection and XSS attacks. It is also concluded as that there is still a lot more improvement required because the web changes every day. Baranwal (2012) further explains that the tools being used are not 100% accurate; the false positive incomplete coverage of the attacks are some of the existing problems that need to be addressed.

2.2 Static Analysis of XSS Vulnerabilities in Web Applications

SAST is a method of security analysis used to secure the web applications, by analysing it's source code without running the application. Wassermann and Su (2008) have mentioned that the XSS vulnerability has been the high reporting 21.5% of vulnerability in 2006. They proposed a tool which bridged the gap on false positives with static analysis. It focuses on both unchecked and inadequate checked inputs. Jingyu et al. (2021) have used a subsequence matching algorithm that identifies the common subsequence input parameters and generates the data which detects the XSS attacks. It has mainly focused on the DOM-based XSS, Reflected XSS and Stored XSS. Jovanovic, Kruegel and Kirda (2006) created a tool called Pixy which is a static analysis tool for XSS vulnerability specifically for PHP applications. It used context-sensitive data flow analysis, flow sensitive, and intraprocedural to identify taint-style vulnerabilities with SQL injection. However, it does not support object-oriented of PHP. Talib and Doh (2021) have done a review on the SAST tools. In this paper, the authors identified seven tools such as PHPCS, PhsSAFE, Pixy, RIPS, VisualCodeGrepper (VCG), Web Application Protection (WAP), and YASCA. They came across accuracy that is as high as 96.8% to 0.6%.

2.3 Dynamic Analysis of XSS Vulnerabilities in Web Applications

DAST is also a method of security analysis used to secure the web applications, by testing the running web application manually or using automated tools. Singh et al. (2015) have proposed a tool which finds vulnerabilities in web application for SQLI or XSS. It uses payloads to perform the attack on the loopholes and analyse the response, this tool is effective on JavaScript inputs. The study performed by Abikoye et al. (2020) was on SQLI and XSS attack, they proposed a tool that uses the Knuth-Morris-Pratt (KMP) string machine

algorithm. The system was developed using a PHP and Apache XAMPP server where it is comparing the user input with the known attack pattern to find the vulnerability. In the research of Stock et al. (2014), they have researched about the tools for client-side Cross-Site Scripting (XSS) filters, which includes Chrome's XSS Auditor and other tools which relies on the string-based matching technique; it comes to light that it mostly bypasses the non-trivial injection contexts and because of that attackers can exploit the system. They proposed a model using dynamic taint tracking with the combination of taint-aware parsers. It keeps on tracking the characters within the browser for the detection whenever an attacker tries to control the inputs through executed code. This mitigates the parsing of malicious inputs at runtime. Gupta, Singh and Dixit (2017) have used the Intrusion Detection System -Snort to find the XSS attack in the web application. The system uses network packets and through signature-based it identifies the malicious patterns and detect the XSS attack.

2.4 Hybrid Analysis of XSS Vulnerabilities in Web Applications

Choi et al. (2017) have introduced a hybrid system with static string analysis and dynamic browser rendering. They used PhantomJS, a headless browser to find the XSS vulnerability. The system successfully identified 55 unique XSS vulnerabilities and gave low false positives. In the research conducted by Mubaiwa and Mukosera (2022) they have specifically targeted SQL injection and XSS with the use of white box and black box testing techniques. In this model they have used a crawler, fuzzing component with report generating, which has given high accuracy then the scanners like Vegam and Arachini. This tool needs to be optimized in terms of scan time.

2.5 Research Gap Identified

The research has been intensive within the scope of web security for the development of a system by both SAST and DAST in detecting XSS vulnerabilities. Literature available suggests that SAST is one of the ways to ensure that the vulnerability could be determined at the earlier stage in the software life cycle process through source code analysis. Research on SAST has shown it to be implemented effectively in the discovery of some of the most common security vulnerabilities, such as injection attacks and insecure deserialization. *etc*. DAST, on the other hand, is known to complement well with SAST and is effective for the discovery of runtime vulnerabilities through attack simulations on a live running application. Findings from such studies indicates that a combination of both SAST and DAST methods can give more comprehensive security testing than is achieved individually[6]. The current project extended this awareness and also the company which I worked as an AppSec intern also researching to develop a comprehensive tool of both approaches, with an intention to enhance the detection of XSS vulnerabilities in Python-based web applications.

3 Research Methodology

This section provides a detailed description of the research methodology employed to develop and evaluate the proposed hybrid Cross-Site Scripting (XSS) detection tool, XSSFind. The methodology combines both static and dynamic techniques in the enhancement of the accuracy of detecting XSS vulnerabilities. This methodology was used in the detection of the XSS vulnerabilities in web applications.

[6] https://circleci.com/blog/sast-vs-dast-when-to-use-them/

3.1 Research Procedure

Developing a hybrid XSS detection tool that combines Static Application Security Testing (SAST) technique and Dynamic Application Security Testing (DAST) technique, was the aim of this research. There are lots of tools available in the market performing either SAST or DAST. For doing the SAST, there are tools like Snyk, Fortify SCA, SonarQube, and many more to do the source code analysis and give a result of vulnerability report. Many tools are developed for DAST, such as Burp Suite, Acunetix, Nikto, among others. None of them provides hybrid methodology, which means SAST + DAST. So, in this research proposed tool development of a comprehensive hybrid detection of XSS vulnerability in web applications. Therefore, to show the practical work of hybrid methodology, a tool using python was developed, which does SAST, DAST, and hybrid analysis. Furthermore, to compare the effectiveness of the hybrid method, this research has utilized Burp Suite, a web application scanner for the comparison between DAST analyses and Snyk, a source code analysis tool for the comparison between SAST analyses.

3.2 Proposed Tool Development & Detection methods:

Python and HTML were used in the development of the hybrid XSS detection tool - XSSFind. Continuous testing and improvement were done during the development process of XSSFind to enhance the tool's detecting capabilities and report generation capabilities.

- **Static Analysis (SAST):** The SAST tool (sast.py) was created using Python to analyse the source code of web applications to detect possible XSS vulnerabilities without executing the code.
- **Dynamic Analysis (DAST):** The DAST tool (dast.py) was created using BeautifulSoup, concurrent.futures, requests, selenium library in it, so it could automate web browser interactions, *etc.*, to mimic like real user behaviour.
- **Hybrid Analysis:** The hybrid analysis tool (hybrid_tool.py), was created by combining the sast.py and dast.py, where it could do static analysis security texting (SAST) and dynamic analysis security testing (DAST).
- **Web Interface:** The web interface should be user friendly and options to select the mode of analysis.

3.3 Workflow Description

The workflow of the XSSFind tool involved the following steps:

- **Initialization:** The user runs the app.py, a web UI will run in the local host.
- **Input Submission:** The user submits either the source code of a web application or a URL or both for performing an analysis.
- **Static Analysis:** The SAST tool analyses the source code for detecting vulnerabilities in the source code using regular expressions and patterns or string matching.
- **Dynamic Analysis:** The DAST tool simulates real-world user interactions with the web application, injecting test payloads to identify XSS vulnerabilities, which happens during the runtime of the target web application.
- **Result Integration:** The hybrid tool combines the findings from both the static and dynamic analysis and provides a comprehensive vulnerability report.
- **Report Generation:** The final report is automatically generated and saved in the tool directory, which explains the detected vulnerabilities with providing recommendations for mitigation.

4 Design Specification

This section outlines the design and architecture of the hybrid XSS detection tool - XSSFind, developed as part of this research. The design focuses on integrating Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) techniques to create a robust solution capable of detecting a wide range of Cross-Site Scripting (XSS) vulnerabilities. The design specification covers the system architecture, component interactions, algorithmic approaches, user interface considerations, and system requirements.

4.1 Proposed System Architecture

The XSSFind tool's architecture was designed to support the integration of static (SAST) and dynamic (DAST) analysis techniques. This tool was developed with separate codes for Hybrid method, SAST method and DAST method, in order to generate a detailed report after each analysis phase.

- **Initiation:** Once app.py is executed, a flask application runs at localhost at port 5000, a user interface is there in the web UI, where user inputs like source code or URL can be given.
- **Input:** The tool accepts the source code of a web application for SAST or a URL for DAST, or both source code and URL for Hybrid.
- **Processing:** The static, dynamic, and hybrid tools process their inputs independently, after analysis it generates detailed reports.
- **Output:** The sast.py and dast.py give the actual results. The hybrid tool merges the results of SAST & DAST analyses, producing a comprehensive report of the detected vulnerabilities.

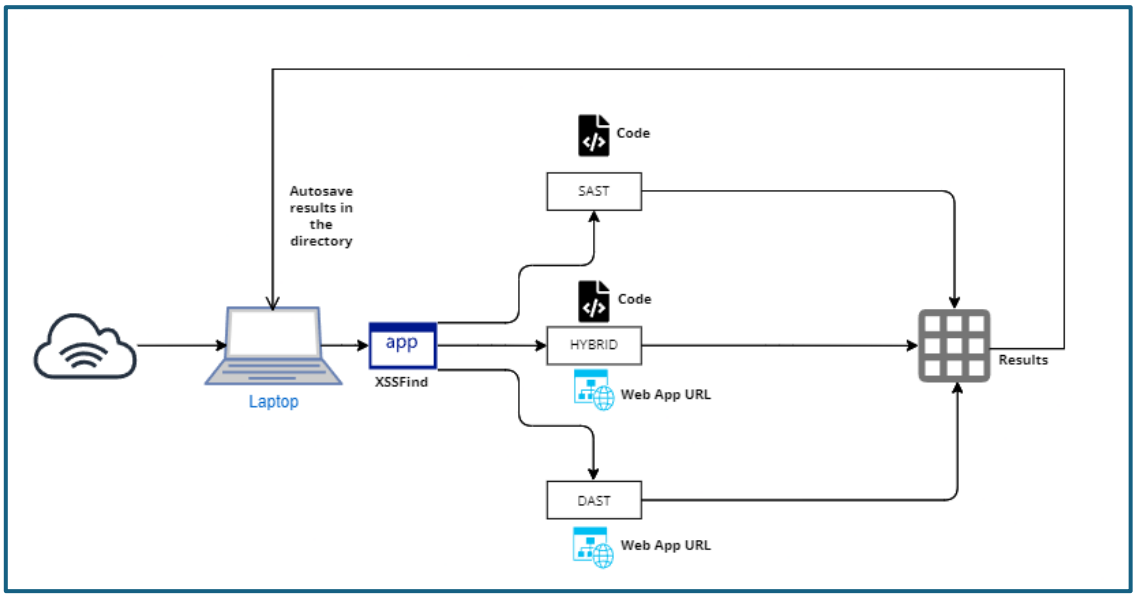

Figure 1: XSSFind Architecture[7]

Figure 1 above is the architecture of the hybrid method proposed for this research, developed using the "online.visual-paradigm" website.

[7] https://online.visual-paradigm.com/

4.1.1 SAST – XSSFind

This tool uses regular expressions matching and string analysis techniques to identify common XSS patterns, such as Unsanitized User Input, DOM Manipulation issues, Stored User input, *etc*. Then finally gives a detailed report of detected vulnerabilities. Figure 2 illustrates the different patterns used to match and identify XSS within the sast.py code.

```
import os
import re
import csv

# Define patterns for various XSS vulnerabilities
patterns = {
    "Unsanitized User Input": r"request\.(GET|POST)\[.*\]",
    "Direct Output of User Input": r"((.*request\.(GET|POST)\[.*\].*))",
    "Use of Unsafe Functions": r"\b(eval|setTimeout|setInterval|innerHTML|outerHTML|document\.write)\b",
    "Missing Output Encoding": r"response\.write\(.*request\.(GET|POST)\[.*\]\)",
    "DOM Manipulation Issues": r"document\.getElementById\(\w+\)\.(innerHTML|outerHTML|value) =.*request\.(GET|POST)\[.*\]",
    "Lack of Content Security Policy (CSP)": r"Content-Security-Policy",
    "Insecure Event Handlers": r"on(click|mouseover|mouseout|keydown|keyup|focus|blur) *= *['\"].*request\.(GET|POST)\[.*\].*['\"]",
    "Reflected User Input": r"response\.write\(.*request\.(GET|POST)\[.*\]\)",
    "Stored User Input": r"db\.insert\(.+request\.(GET|POST)\[.*\]",
    "Misuse of JavaScript Libraries": r"\b(eval|Function)\(.*\)",
    "Vulnerable Third-Party Components": r"script src=.*\b(known_vuln_lib)\b",
    "Insecure Data Handling in Cookies": r"document\.cookie *= *request\.(GET|POST)\[.*\]",
    "Improper URL Construction": r"(href|src) *= *['\"].*request\.(GET|POST)\[.*\].*['\"]",
    "Insecure Handling of File Uploads": r"request\.FILES\['.*'\]",
    "JSONP Callback Issues": r"\bcallback *= *request\.(GET|POST)\[.*\]"
}

def scan_file_for_xss(file_path):
    """Scan a file for potential XSS vulnerabilities based on predefined patterns."""
    vulnerabilities = []

    with open(file_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()
        for i, line in enumerate(lines):
            for vuln_type, pattern in patterns.items():
                if re.search(pattern, line):
                    vulnerabilities.append((i + 1, line.strip(), vuln_type))
```

Figure 2: XSSFind – sast.py

4.1.2 Dynamic – XSSFind

Once the target URL is given and the tool starts crawling the website and injects the malicious payloads into different entry points or input fields, forms are present on the website to detect the XSS vulnerability during running of website. Then finally provides a detailed report of detected vulnerabilities. Below Figure 3, represents the dast.py code in the tool repo.

```
import os
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse, parse_qs, urlencode
import concurrent.futures
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.common.exceptions import WebDriverException
import csv
import time

class AdvancedXSSDAST:
    def __init__(self, target_url):
        self.target_url = target_url
        self.visited_urls = set()
        self.crawled_urls = []  # List to store crawled URLs
        self.vulnerabilities = []
        self.init_browser()
        self.xss_payloads = {
            'html': ["<script>alert('XSS')</script>", "<img src='x' onerror='alert(1)'>"],
            'attribute': ['" onmouseover="alert(1)"'],
            'js': ["');alert(1);//", "';alert(1)//"],
            # Add more payloads if necessary
        }

    def init_browser(self):
        chrome_options = Options()
        chrome_options.add_argument("--headless")
        chrome_options.add_argument("--disable-gpu")
        chrome_options.add_argument("--no-sandbox")
        self.driver = webdriver.Chrome(options=chrome_options)

    def run(self):
        self.crawl(self.target_url)
        self.generate_report("dast_results.csv")
        self.driver.quit()
```

Figure 3: XSSFind – dast.py

4.1.3 Hybrid – XSSFind

It is developed by combining the sast.py and dast.py, where it does static analysis security texting (SAST) and dynamic analysis security testing (DAST). Then provides a comprehensive detailed report of detected vulnerabilities, which consists of both SAST & DAST vulnerabilities. Below Figure 4 represents the hybrid_tool.py code of the tool repo.

```python
import os
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.common.exceptions import WebDriverException
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin
import concurrent.futures
import ast
import csv
import time

class HybridXSSDetectionTool:
    def __init__(self, mode, target_url=None, code_directory=None):
        self.mode = mode
        self.target_url = target_url
        self.code_directory = code_directory
        self.vulnerabilities = []
        self.crawled_urls = []

    def dast_init(self):
        chrome_options = Options()
        chrome_options.add_argument("--headless")
        chrome_options.add_argument("--disable-gpu")
        chrome_options.add_argument("--no-sandbox")
        self.driver = webdriver.Chrome(options=chrome_options)
        self.xss_payloads = {
            'html': ["<script>alert('XSS')</script>", "<img src='x' onerror='alert(1)'>"],
            'attribute': ['" onmouseover="alert(1)"'],
            'js': ["');alert(1);//", "');alert(1)//"],
            # Add more payloads if necessary
```

Figure 4 XSSFind – hybrid_tool.py

4.1.4 Web Interface – XSSFind

Web Interface is created using Flask, and it is built separately in a file named – app.py. Once this tool is executed, a flask application will be running in the localhost at port 5000, there the testing inputs has to be given, like source code for SAST, URL for DAST, and source code & URL for hybrid. After analysis phase, a detailed comprehensive report of SAST and DAST vulnerabilities will be automatically generated in the same folder with a timestamp for future usage Users can customize the analysis process by selecting specific modules (SAST, DAST, or both). Below Figure 5 represents the app.py code of the tool repo and Figure 6 represents the Web UI of the Tool.

```python
from flask import Flask, render_template, request, redirect, url_for, send_file, Response
import os
import time
import zipfile
import glob
from werkzeug.utils import secure_filename
from hybrid_tool import HybridXSSDetectionTool
import progress_tracker

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads/'

# Ensure the upload folder exists
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

def extract_zip_to_directory(zip_path, extract_to):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_to)
```

Figure 5: XSSFind – app.py

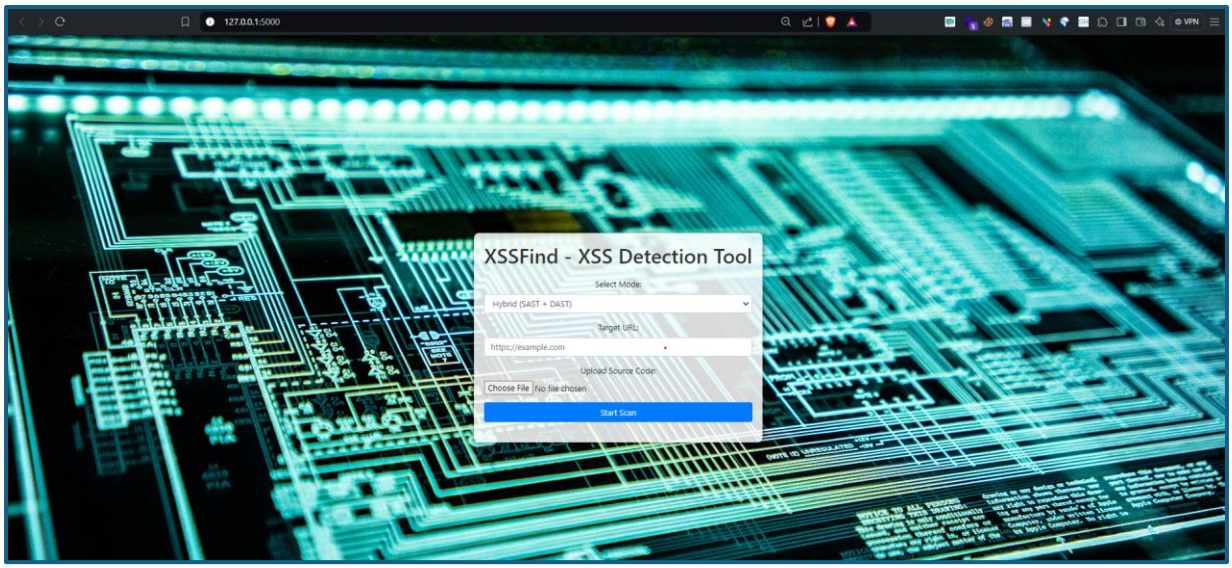

Figure 6: Web Interface

4.2 System Requirements

4.2.1 Hardware Requirements

The proposed tool is designed to run on standard personal computers. The minimum hardware requirements are:

- CPU: Intel i5 or equivalent
- RAM: 8 GB
- Storage: 500 MB of free disk space
- Network: Internet connection required for dynamic analysis and accessing online vulnerable web applications.

4.2.2 Software Requirements

The tool requires the following software:

- Operating System: Windows 10/11, Linux, MacOS
- Python: Version 3.7 or higher
- Framework: Flask 3.0
- Web Browser: Google Chrome or Mozilla Firefox (for dynamic analysis)
- Web Server: Apache or Nginx (optional for hosting web applications during testing)

4.2.3 Libraries/Modules:

The development of XSSFind relied on several Python libraries and modules, including:

- requests: For sending HTTP requests to web applications.
- BeautifulSoup: For parsing HTML content during dynamic analysis.
- selenium: For automating browser interactions.
- pandas: For handling and analysing tabular data.
- csv: For generating and processing reports.

4.2.4 Development Tools

The following development tools were used:

- Integrated Development Environment (IDE): PyCharm or Visual Studio Code.
- Version Control: Git for version management and collaboration.
- Testing Environment: VirtualBox with Kali Linux to test in Linux environments. Also, a Web browser (Chrome or Firefox) for testing the web interface.

5 Implementation

Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) methodologies were integrated into the development of the proposed hybrid methodology - XSSFind, resulting in a powerful hybrid XSS detection tool that supports several platforms. Various tools and environments were used during the development and testing process to guarantee its efficiency, compatibility, and ease of use.

5.1 Development Environment

- **Visual Code Studio:** An Integrated Development Environment (IDE) which is popular and adaptable, Visual Studio Code, was used to build the XSSFind tool. Visual Studio Code was selected because it supports python, has a lot of extensions to help while development, inbuilt terminals – Python, Command Prompt, PowerShell, Bash, *etc*. Also, it helps during development, debugging, and testing processes, like if any errors made in coding, it would highlight the error, so it reduces the coding errors.
- **Python:** The primary programming language used to develop XSSFind, chose for its ease of use, large number of libraries, and excellent community support. Python's large ecosystem of libraries made it easy for efficient development of the hybrid method. Key libraries included **requests** for HTTP interactions, **BeautifulSoup** for parsing HTML, and **selenium** for browser automation during dynamic analysis, *etc*.
- **Virtual Environments:** Through the entire development process of hybrid tool, virtual environments have been used, to manage all the dependencies required for the hybrid tool and make sure the tool operates properly in every kind of systems, like windows, Linux. This usage also helped to easily isolate and manage dependent Python packages required by XSSFind.
- **Version Control:** GitHub was used for systematic tracking of changes made in any of the code or file in the repository, maintain versions, and collaboration with the developers of the internship company if needed. GitHub's platform also provided an efficient means of sharing and deploying the tool. Below Figure 7 represents the tool directory which is uploaded to GitHub.

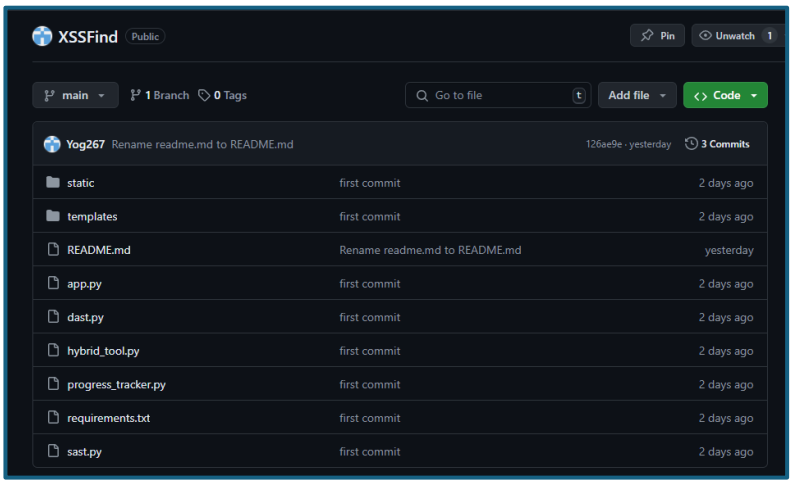

Figure 7: GitHub Page and XSSFind Folder Structure (Anandhakumar 2024)

5.2 Development Platforms

- **Windows Development:** The XSSFind was developed in a Windows 11 machine, using Visual Studio Code. This choice also ensured compatibility of the tool with a significant user base.

- **Linux Compatibility Testing:** To make sure that XSSFind tool functions effectively in a Linux environment, the tool was tested on Kali Linux, a Debian-based distribution. Kali Linux is a popular operating system used by cyber security professionals, penetration testers and security researchers. Also, Kali Linux has many varieties of cyber security tools. This compatibility testing helped in understanding the multi-platform compatibility of the XSSFind tool.

5.3 Code Structure and Management – code function explain

The codebase for XSSFind was structured to separate the static and dynamic analysis components while allowing for easy integration via the hybrid analysis module.

- **Static Analysis (sast.py):** This tool was implemented with a focus on pattern matching and data flow analysis. It scans the source code for potential XSS vulnerabilities, focusing on common patterns such as unescaped user inputs. It imports modules like **csv** for data export, **re** for regular expressions, **os** for managing file. It has given few patterns of XSS Vulnerabilities like Unsanitized user input, DOM manipulation, *etc*., as input , so the tool will use string analysis and pattern matching technique. The function scan_file_for_xss(file_path) checks line by line of a file using the patterns, if any pattern detected, it would report as findings with its line number, type of vulnerability. The function scan_directory_for_xss(directory), an extension of previous function, which repeats the same process over all the files in the directory.
- **Dynamic Analysis (dast.py):** This tool is implemented to detect XSS vulnerabilities in the web applications, during runtime. This tool imports modules like **requests** for making web requests, **bs4** for parsing html, **selenium** for automating browser interactions, *etc*. The class **AdvancedXSSDAST** starts with a target URL, a headless chrome browser, preconfigured XSS payloads. Then the **crawl( )** is used to visit all the URLs in the web application and inject the payloads, wherever forms or entry points available. After crawling and injecting, all the data, such as crawled URLs and XSS vulnerabilities are stored and generated as two different csv reports – crawled URLs & dast results.
- **Hybrid Integration (hybrid_tool.py):** This tool is implemented to detect XSS vulnerabilities in hybrid mode (sast + dast). It imports modules like, **requests** for making web requests, beautifulsoup (**bs4)** for parsing html, **selenium** for automating browser interactions, Abstract Syntax Tree (ast) for parsing and analysing source codes. The class **HybridXSSDetectionTool** starts with a mode sast + dast, where target URL and target web app source code is given, then the sast does pattern matching and generates report and automatically stores in the tool folder. Then dast starts crawling all the URL inside the web app, then injects the payloads, wherever forms or entry points available. After crawling and injecting, all the data, such as crawled URLs and XSS vulnerabilities are stored, combined with the sast results and generated as two different csv reports – crawled URLs & hybrid results.

5.4 Testing and Validation

Testing and Validation are the important steps in the implementation process, making sure that the XSSFind will work and detect XSS vulnerabilities in different environments and scenarios.

- **Multi-Platform Testing:** After its development on Windows, the tool was tested on Kali Linux to check whether it is working properly and checking any performance issues while working in the Linux environment. This testing proved that the XSSFind tool's functionality remained same regardless of the operating system, showcasing its multi-platform compatibility.

- **Validation on Real-World Applications:** XSSFind was tested on a vulnerable web application owned by Acunetix, a popular security scanner organization. This test proved the tool's ability in detecting XSS vulnerabilities and its ability to merge static and dynamic analysis results after hybrid analysis.

5.5 Deployment Considerations

XSSFind was designed to be easily deployed and used, by using Python's cross-platform feature. Any laptop or computer running the Python version 3.7 or above can install and use this tool. The tool has to be installed in the user system, install the requirements.txt using *pip*, which has all the requires dependency packages. After installing those, the tool is ready to go for testing web apps for detecting XSS Vulnerabilities.

6 Evaluation

The evaluation of the XSSFind tool was conducted through a series of experiments designed to assess its effectiveness, efficiency, and accuracy in detecting Cross-Site Scripting (XSS) vulnerabilities. The evaluation was performed by comparing the tool's performance with industry-standard tools such as Burp Suite for Dynamic Application Security Testing (DAST) and Snyk for Static Application Security Testing (SAST). The evaluation focused on several key metrics: accuracy and false positive rate.

6.1 Test Case 1: Static Analysis

- **Objective:** Ensure XSSFind tool analyses the XSS Vulnerability of the Simple Hospital Management System from Code Astro, and a custom malicious Python file, which is created and saved inside the same code repo. Also, the custom malicious file is created to detect a XSS vulnerability in the line 23.

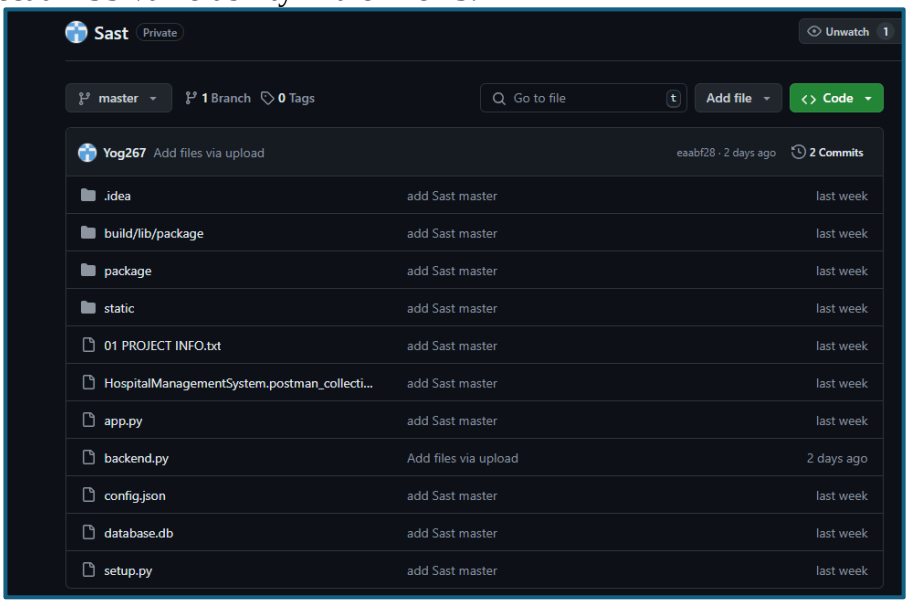

Figure 8: Source code file repo for SAST

6.2 Test Case 2: Dynamic Analysis

- **Objective:** Ensure XSSFind analyses the XSS Vulnerability of the vulnerable web application – http://testphp.vulnweb.com .

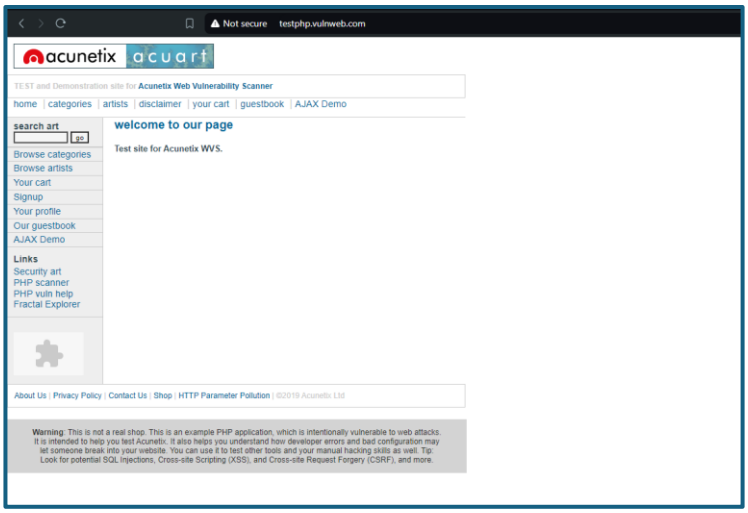

Figure 9: Web App for DAST.

6.3 Test Case 3: Hybrid Analysis

- **Objective:** Analyse XSSFind's Hybrid method against Snyk & Burp Suite by testing the Simple Hospital Management System from Code Astro, & a self-created malicious Python file created & saved inside the same code repo and also the vulnerable web application – http://testphp.vulnweb.com.

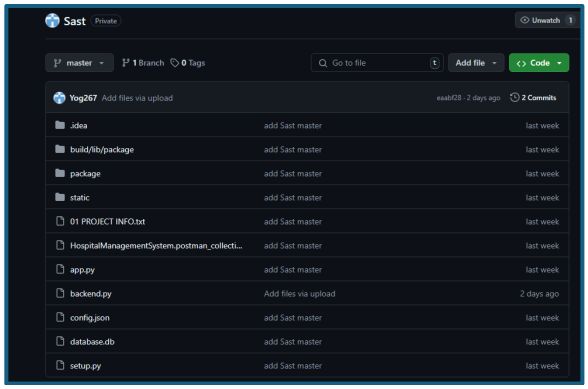

Figure 10: Source code file repo for SAST

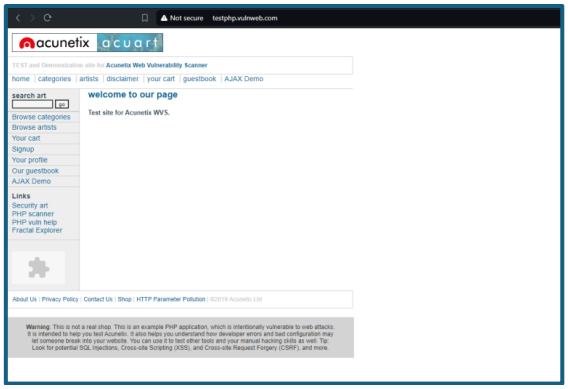

Figure 11: Web App for DAST

6.4 Discussion

The evaluation results below indicate that XSSFind is a powerful tool for detecting XSS vulnerabilities, particularly when leveraging the hybrid analysis approach. The tool's ability to combine the strengths of SAST and DAST allows it to provide a more comprehensive security assessment than either method alone. However, the trade-offs between accuracy and false positives. So, this research satisfies the research question.

6.4.1 Snyk vs XSSFind

- **Outcome**: XSSFind detected 1 vulnerability, while Snyk identified 4, which is indicating high accuracy and no false positives of XSSFind when compared to Snyk, which did not detect the actual vulnerability in line 23.

Table 1: XSSFind SAST Results

File/URL	Context	Line Number	Payload	Vulnerability Description	Mitigation
uploads/extracted\SimpleHospitalMS-Python\backend.py	SAST	23		Use of potentially unsafe function: eval	Avoid using unsafe functions like `eval` or `exec`. Consider using safer alternatives. For more information, visit: https://owasp.org/www-community/attacks/xss/

Figure 12: Snyk SAST Results

6.4.2 Burp Suite vs XSSFind

- **Outcome:** XSSFind detected 27 runtime vulnerabilities where Burp Suite detected 19 runtime vulnerabilities.

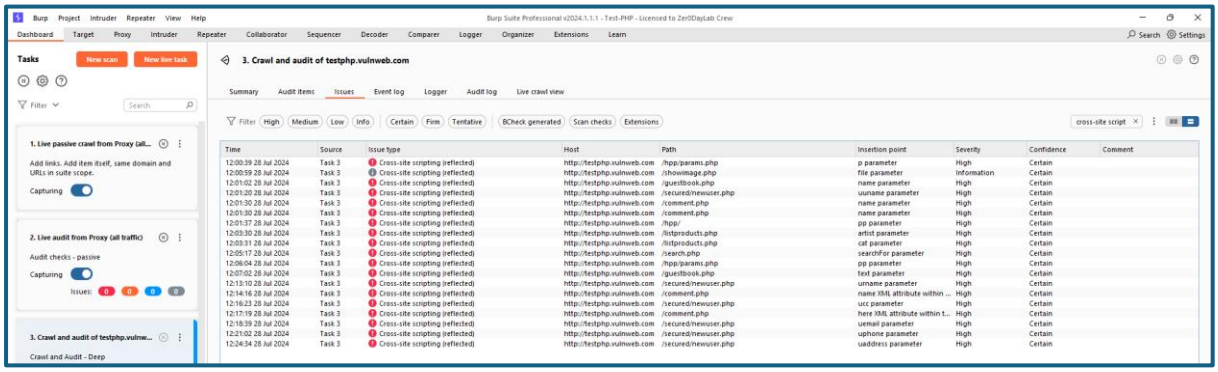

Figure 13: Burp Suite DAST Results

File/URL	Conte	Line Numb	Payload	Vulnerability Description	Mitigation
http://testphp.vulnweb.com/	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/index.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/categories.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/artists.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/disclaimer.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/cart.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/guestbook.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/guestbook.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/login.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/signup.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/userinfo.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/artists.php?artist=1	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?artist=1	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=1	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=2	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=3	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=4	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=5	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=6	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/artists.php?artist=2	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?artist=2	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=7	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/artists.php?artist=3	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?artist=3	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?cat=1	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?cat=2	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?cat=3	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?cat=4	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/

Figure 14: XSSFind DAST Results

6.4.3 Burp Suite & Snyk Vs XSSFind Hybrid:

- **Outcome:** XSSFind detected 27 runtime vulnerabilities where Burp Suite detected 19 runtime vulnerabilities and also it detected 1 vulnerability, while Snyk identified 4, indicating high accuracy with no false positive detections where Snyk didn't detect the vulnerability created in the custom malicious file and gave 4 false positives.

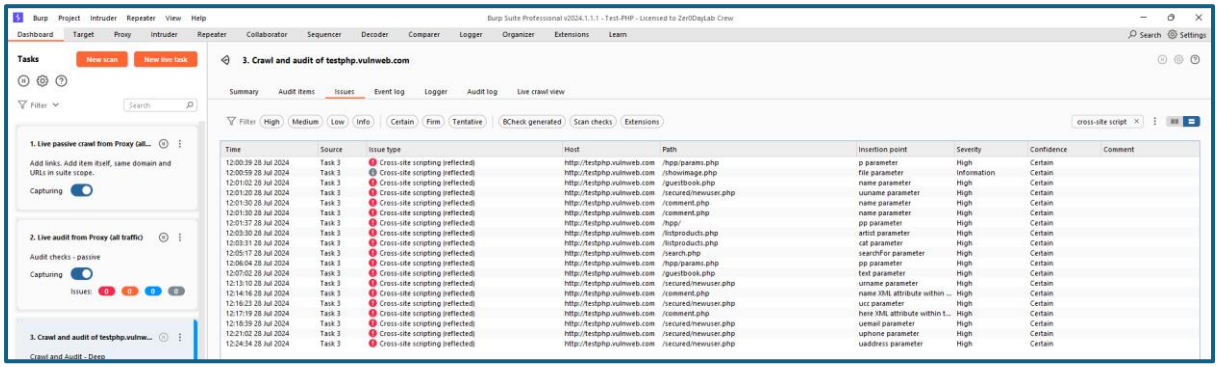

Figure 15: Burp Suite DAST Results

Figure 16: Snyk SAST Results

File/URL	Conte	Line Numb	Payload	Vulnerability Description	Mitigation
uploads/extracted\SimpleHospitalMS-Python\backend.py	SAST	23		Use of potentially unsafe function: eval	Avoid using unsafe functions like `eval` or `exec`. Consider using safer alternatives. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/index.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/categories.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/artists.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/disclaimer.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/cart.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/guestbook.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/guestbook.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/login.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/signup.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/userinfo.php	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/artists.php?artist=1	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?artist=1	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=1	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=2	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=3	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=4	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=5	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=6	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/artists.php?artist=2	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?artist=2	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/product.php?pic=7	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/artists.php?artist=3	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?artist=3	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?cat=1	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?cat=2	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?cat=3	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/
http://testphp.vulnweb.com/listproducts.php?cat=4	html		<script>alert('XSS')</script>	XSS vulnerability detected	Ensure all user inputs are properly escaped before rendering in HTML. For more information, visit: https://owasp.org/www-community/attacks/xss/

Figure 17: XSSFind Hybrid Results

6.4.4 Strengths:

- When compared to the standalone tools like Burp Suite (DAST) and Snyk (SAST), the proposed hybrid analysis approach greatly reduces false positives and identifies more vulnerabilities than them.
- The user-friendly interface and comprehensive detailed reports enhance the tool's usability, may result in usage by a wide range of users.

6.4.5 Limitations:

- This static analysis in hybrid method is developed to analyse the source code of python web applications alone.
- The dynamic analysis in hybrid method may require optimization for quicker results and better performance under heavy loads & large-scale applications. Also, more powerful effective payloads need to be added.

7 Conclusion and Future Work

7.1 Conclusion

In this research, the proposed method for the development of a hybrid method to detect Cross-Site Scripting (XSS) vulnerabilities, which is named as XSSFind. It puts together both the SAST and DAST approach in detecting XSS vulnerabilities. According to the evaluation, XSSFind can manage to lesser false positives and enhance detection accuracy by integrating both static and dynamic techniques of analysis. On the other hand, the methodology of Hybrid Detection followed by XSSFind allows for even more in-depth penetration testing than what standalone tools like Snyk and Burp Suite do.

With these benefits, a few shortcomings were observed for XSSFind: only Python source codes are analysed in the static analysis and dast analysis also a bit slower. These findings of this study indicates that, while the hybrid approach could be effective, but still it needs to be improved, so that the performance can be increased, and multiple language applications can be tested.

One of the key contributions in this research is extending the capability of a detection tool, where XSSFind is a hybrid security testing tool implementing SAST and DAST to detect XSS vulnerabilities in web applications. More than that, the created hybrid methodology in XSSFind would give a complete and reliable result with lowered false positives and increased detection in comparison to the single solutions Burp Suite (DAST) and Snyk (SAST).

7.2 Future Work

In order to reduce false positives, future research may focus on improving the SAST method, by incorporating more advanced techniques like machine learning. In a study by Santithanmanan, Kirimasthong and Boongoen (2024) the machine learning approach has been used to detect the XSS attacks using numerous classifiers; they have used the algorithms such as Decision tree, Random Forest and Gradient Boosting. While comparing it with the traditional signature-based techniques it has shown effective results with less false positives. Mahiuob et al. (2021) proposed XGBXSS a framework for detection of XSS using Extreme Gradient Boosting (XSBoost) with a hybrid feature and optimization parameters. The results have shown a 99.59% accuracy of the tool and low false rate of 0.18%. It is seen that it is more effective in finding the zero-day XSS attack. Kumar and Ponsam (2023) have focused on a machine learning approach using CNN, LSTM, AdaBoost and Random Forest and Decision tree. The model with AdaBoost, Random Forest and Decision tree have performed very well with higher accuracy and precision. In comparison to the traditional models machine learning techniques are more accurate in finding the XSS attack. Bakır and Bakır (2024) have proposed an approach to detect the XSS attacks using Hybrid Semantic Embeddings with the combination of Universal Sentence encoder and Word2Vec. This method has used machine learning and deep learning and has achieved a good accuracy and F1 score making it effective for real-time XSS detection. These studies can help in improving the XSS detection by using ML. Also, it may focus on improving the DAST method to handle large-scale applications more efficiently and quickly, this could enhance the tool's scalability and performance. Also, may focus on integration of real-time monitoring and adaptive payload generation methods could further increase the detection capabilities of XSSFind, which makes it more adaptable to evolving web security threats.

References

Abikoye, O.C., Abubakar, A., Dokoro, A.H., Akande, O.N. and Kayode, A.A. 2020. A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm. *Eurasip Journal on Information Security* 2020(1), pp. 1–14. Available at: https://jis-eurasipjournals.springeropen.com/articles/10.1186/s13635-020-00113-y [Accessed: 1 August 2024].

Anandhakumar, Y. 2024. *GitHub - Yog267/XSSFind*. Available at: https://github.com/Yog267/XSSFind [Accessed: 1 August 2024].

Bakır, R. and Bakır, H. 2024. Swift Detection of XSS Attacks: Enhancing XSS Attack Detection by Leveraging Hybrid Semantic Embeddings and AI Techniques. *Arabian Journal for Science and Engineering*, pp. 1–17. Available at: https://link.springer.com/article/10.1007/s13369-024-09140-0 [Accessed: 1 August 2024].

Baranwal, A.K. 2012. Approaches to detect SQL injection and XSS in web applications. *The University of British Columbia*.

Choi, H., Hong, S., Cho, S. and Kim, Y.G. 2017. HXD: Hybrid XSS detection by using a headless browser. *Proceedings of the 2017 4th International Conference on Computer Applications and Information Processing Technology, CAIPT 2017* 2018-January, pp. 1–4. Available at: https://doi.org/10.1109/CAIPT.2017.8320672

Cui, Y., Cui, J. and Hu, J. 2020. A Survey on XSS Attack Detection and Prevention in Web Applications. *ACM International Conference Proceeding Series*, pp. 443–449. Available at: https://dl.acm.org/doi/10.1145/3383972.3384027 [Accessed: 1 August 2024].

Garcia-Alfaro, J. and Navarro-Arribas, G. 2007. Prevention of Cross-Site Scripting Attacks on Current Web Applications *. *Lecture Notes in Computer Science, vol 4804. Springer, Berlin, Heidelberg.* Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0b64d3fafef0fba27f8991c6fc0bde32c70b90dc [Accessed: 1 August 2024].

Gupta, K., Ranjan Singh, R. and Dixit, M. 2017. Cross site scripting (XSS) attack detection using intrustion detection system. *Proceedings of the 2017 International Conference on Intelligent Computing and Control Systems, ICICCS 2017* 2018-January, pp. 199–203. Available at: https://doi.org/10.1109/ICCONS.2017.8250709

Hannousse, A., Yahiouche, S. and Cherif Nait-Hamoud, M. 2024. Twenty-two years since revealing cross-site scripting attacks: A systematic mapping and a comprehensive survey. *Computer Science Review, Volume 52, 2024, 100634, ISSN 1574-0137,*. Available at: www.elsevier.com/locate/cosrev [Accessed: 1 August 2024].

Harish Kumar, J. and Godwin Ponsam, J.J. 2023. Cross Site Scripting (XSS) vulnerability detection using Machine Learning and Statistical Analysis. *2023 International Conference on Computer Communication and Informatics, ICCCI 2023*. Available at: https://doi.org/10.1109/ICCCI56745.2023.10128470

Invicti. 2024. *Reflected/Non-Persistent Cross-Site Scripting*. Available at: https://www.invicti.com/learn/reflected-xss-non-persistent-cross-site-scripting/ [Accessed: 1 August 2024].

Jasmine M S, Devi Kirthiga and George Geogen. 2017. *(PDF) Detecting XSS Based Web Application Vulnerabilities*. Available at: https://www.researchgate.net/publication/317166075_Detecting_XSS_Based_Web_Application_Vulnerabilities [Accessed: 25 August 2024].

Jingyu, Z., Hongchao, H., Shumin, H. and Huanruo, L. 2021. A XSS Attack Detection Method Based on Subsequence Matching Algorithm. *2021 IEEE International Conference on Artificial Intelligence and Industrial Design, AIID 2021*, pp. 83–86. Available at: https://doi.org/10.1109/AIID51893.2021.9456515

Jovanovic, N., Kruegel, C. and Kirda, E. 2006. Pixy: A static analysis tool for detecting web application vulnerabilities (Short paper). *Proceedings - IEEE Symposium on Security and Privacy* 2006, pp. 258–263. Available at: https://doi.org/10.1109/SP.2006.29

Liu, M., Zhang, B., Chen, W. and Zhang, X. 2019. A Survey of Exploitation and Detection Methods of XSS Vulnerabilities. *IEEE Access* 7, pp. 182004–182016. Available at: https://doi.org/10.1109/ACCESS.2019.2960449

Di Lucca, G.A., Fasolino, A.R., Mastroianni, M. and Tramontana, P. 2004. Identifying cross site scripting vulnerabilities in Web applications. *Proceedings - Sixth IEEE International Workshop on Web Site Evolution, WSE 2004*, pp. 71–80. https://doi.org/10.1109/WSE.2004.10013

Mahiuob, F., Mokbal, M., Dan, W., Xiaoxi, W., Wenbin, Z. and Lihua, F. 2021. XGBXSS: An Extreme Gradient Boosting Detection Framework for Cross-Site Scripting Attacks Based on Hybrid Feature Selection Approach and Parameters Optimization. *Journal of Information Security and Applications* 58. Available at: https://doi.org/10.1016/j.jisa.2021.102813 [Accessed: 1 August 2024].

Mahmoud, S.K., Alfonse, M., Roushdy, M.I. and Salem, A.B.M. 2017. A comparative analysis of Cross Site Scripting (XSS) detecting and defensive techniques. *2017 IEEE 8th International Conference on Intelligent*

Computing and Information Systems, ICICIS 2017 2018-January, pp. 36–42. Available at: https://doi.org/10.1109/INTELCIS.2017.8260024

Mubaiwa, T.G. and Mukosera, M. 2022. A HYBRID APPROACH TO DETECT SECURITY VULNERABILITIES IN WEB APPLICATIONS. *International Journal of Computer Science and Mobile Computing* 11(2), pp. 89–98. Available at: https://doi.org/10.47760/ijcsmc.2022.v11i02.011 [Accessed: 1 August 2024].

OWASP - XSS. 2024. *Cross Site Scripting (XSS) | OWASP Foundation*. Available at: https://owasp.org/www-community/attacks/xss/#:~:text=Cross%2DSite%20Scripting%20(XSS),to%20a%20different%20end%20user [Accessed: 1 August 2024].

portswigger. 2024. *What is DOM-based XSS (cross-site scripting)? Tutorial & Examples | Web Security Academy*. Available at: https://portswigger.net/web-security/cross-site-scripting/dom-based [Accessed: 1 August 2024].

Santithanmanan, K., Kirimasthong, K. and Boongoen, T. 2024. Machine Learning Based XSS Attacks Detection Method. *Advances in Computational Intelligence Systems. UKCI 2023*, pp. 418–429. Available at: https://link.springer.com/chapter/10.1007/978-3-031-47508-5_33 [Accessed: 1 August 2024].

Sarmah, U., Bhattacharyya, D.K. and Kalita, J.K. 2018. A survey of detection methods for XSS attacks. *Journal of Network and Computer Applications*. Available at: https://doi.org/10.1016/j.jnca.2018.06.004 [Accessed: 1 August 2024].

Singh, P., Thevar, K., Shetty, P. and Shaikh, B. 2015. Detection of SQL Injection and XSS Vulnerability in Web Application. *International Journal of Engineering and Applied Sciences (IJEAS) ISSN: 2394-3661, Volume-2, Issue-3, March 2015*. Available at: https://media.neliti.com/media/publications/257981-detection-of-sql-injection-and-xss-vulne-3e81bb03.pdf [Accessed: 1 August 2024].

Stock, B., Lekies, S., Mueller, T., Spiegel, P. and Johns, M. 2014. Precise Client-side Protection against DOM-based Cross-Site Scripting. *Proceedings of the 23rd USENIX Security Symposium. August 20–22, 2014*. Available at: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/stock [Accessed: 1 August 2024].

Talib, N.A.A. and Doh, K.-G. 2021. Static Analysis Tools Against Cross-site Scripting Vulnerabilities in Web Applications : An Analysis. *Journal of Software Assessment and Valuation* 17(2), pp. 125–142. http://dx.doi.org/10.29056/jsav.2021.12.14

Trendmicro. 2024. *3 Types of Cross-Site Scripting (XSS) Attacks | Trend Micro (IE)*. Available at: https://www.trendmicro.com/en_ie/research/23/e/cross-site-scripting-xss-attacks.html [Accessed: 1 August 2024].

Wassermann, G. and Su, Z. 2008. Static detection of cross-site scripting vulnerabilities. *Proceedings - International Conference on Software Engineering*, pp. 171–180. Available at: https://dl.acm.org/doi/10.1145/1368088.1368112 [Accessed: 1 August 2024].