# CYBERSECURITY BCSE410L

**WEB APPLICATION SECURITY SCANNER**

A REPORT

submitted by

Harshavardhan Bommi (21BCE5571)

TO DR.SUBBULAKSHMI T

SLOT: E1

B. Tech. Computer Science and Engineering



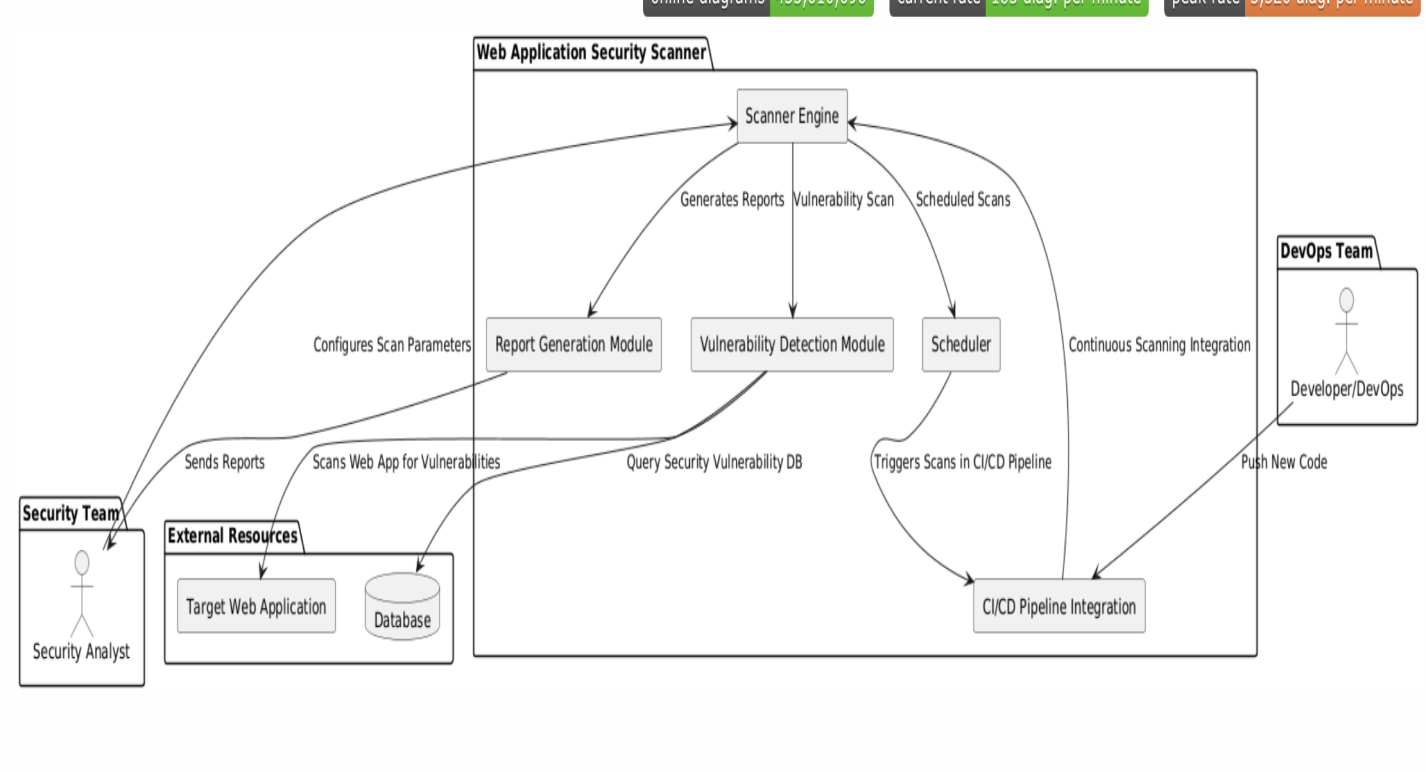
NAME : Harshavardhan Bommi (21BCE5571)

# INTRODUCTION

In Web applications form the backbone of all modern businesses, encompassing services that are critical to sensitive data handling and customer interaction. At the same time, however, web applications remain one of the prime targets for cyberattacks due to their ubiquitous nature and the vast attack surface they present. From the handling of user input to the management of authentication processes, web applications become a potential target for many forms of security threats if not properly secured.

Attackers frequently use a couple of the well-known vulnerabilities, like Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), SQL injection, and insecure endpoints, in trying to access or manipulate data or disrupt services without being authorized. Sometimes, security is sacrificed in favor of growing web application complexity.  
  
The Web Application Security Scanner will be designed to respond to these concerns with automation so as to find security vulnerabilities in web applications. The tool automatically conducts exhaustive, systematic testing for various kinds of cyberattacks. It thus enables the identification of common and high-risk vulnerabilities that can be exploited before they occur. It checks for issues like the OWASP Top 10, and it will also allow less common security flaws to be identified that could arise from specific application logics or improper configurations.  
  
One of the high-quality features of the scanner includes generating detailed vulnerability reports. These reports consider the nature of the detected vulnerabilities-inclusive of risk levels, potential impact, and recommended remediation ideas. This will enable development and security teams to address problems in a timely fashion, significantly reducing the risk of an effective attack. This solution will be flexible; it allows turning the actual scanning with regard to the user's needs about security from a fast check of vulnerability to an in-depth assessment of application security.  
  
Besides, the Web Application Security Scanner smoothly integrates into the CI/CD pipeline and supports continuous security testing within one software development cycle. It means that most of the vulnerabilities would be detected in due time and thus mitigated when on the initial stage before appearing in a production environment. The scanner automates security testing; respectively, the amount of manual efforts that would have been invested by security teams is reduced, enabling these teams to deal with other issues that might prove to be more complicated and further reduce general risk from web application vulnerabilities.

# ARCHITECTURE

****

**EXPLANATION**

The Python script below performs a security test of the web application for a few primary security vulnerabilities, like XSS, CSRF, SQL Injection, and Security headers missing. The script is designed to scan common endpoints such as /login, /admin, and /search and would alert the user in case any of the security issues get detected.

**Key Components:**

1. **Imports:**

* **requests:** Send HTTP GET and POST requests to simulate user interactions with the web application.
* **BeautifulSoup:** The library used for parsing HTML content in order to extract and manipulate form data.
* **urljoin, urlparse:** These functions are used to handle URL manipulations; these ensure that relative paths join correctly with a base URL and get parsed when needed.

1. **Variable Declaration**

* **base\_url:** This is the root URL of the website to be tested. For example, <http://fao.org.>
* **endpoints:** These are the common paths,/login,/admin,/search, etc., which will be joined to base\_url in order to form the complete URL for testing.
* **Payloads:**

1. **XSS Payloads:** It is the malicious JavaScript code that can be injected into vulnerable fields to check if the site is prone to XSS attacks.
2. **SQL Injection Payloads:** Most of the SQL queries used to exploit vulnerable web forms/input fields.
3. **CSRF Payloads:** Faux CSRF tokens to test if forms require valid anti-CSRF tokens.

* **Security Headers:** A set of key HTTP headers like Strict-Transport-Security and X-Frame-Options, which protect the website from some of the most common types of attacks, such as clickjacking, cross-site scripting, and HTTP downgrade attacks.

1. **Functions:**

* **test\_xss(url):** Makes a GET request with XSS payloads and checks if the malicious script is reflected in the response. In this case, when the payload is reflected in the response, there could be a possible vulnerability to XSS.
* **test\_csrf(url):** Parses all forms from the page and submits them with fake CSRF tokens. If the server does not require a valid CSRF token and accepts the request, the site may be vulnerable to CSRF attacks.
* **test\_sql\_injection(url):** Tests for SQL injection by sending common SQL payloads (like ' OR '1'='1 or ' AND 1=1 --) and checking the response for SQL errors or unusual behavior.
* **test\_security\_headers(url):** Checks the HTTP response for some essential security headers (like Content-Security-Policy). Flags those that are missing as potential risks.
* **test\_ssl\_cert(url):** Tests whether the URL uses HTTPS and checks the validity of its SSL certificate. If it can't find a valid certificate or if it finds only HTTP, it flags an issue.
* **test\_insecure\_endpoints(url):** Finds potentially insecure endpoints based on heuristics such as very low content length, 403 Forbidden status code, or 404 Not Found status code.
* **<test\_open\_redirects(url)>:** Finds form actions that redirect users to external domains and can be abused for open redirect attacks.

1. **MAIN:**

* <scan\_web\_app()>: Iterates over each endpoint in the list of endpoints, appending it to the base\_url, and applies all the security tests to the resulting URL: test\_xss, test\_csrf, test\_sql\_injection, etc. The results of scans are printed out to the console screen.

1. **EXECUTION BLOCK:**

* The script runs all the tests only if executed directly (if \_\_name\_\_ == "\_\_main\_\_"). This stops automatic execution of the scan if the script is imported elsewhere as a module.

# CODE

import requests

from bs4 import BeautifulSoup

from urllib.parse import urljoin, urlparse

# Configuration

base\_url = "http://example.com"

endpoints = [

"/",

"/login",

"/search",

"/profile",

"/admin"

]

# Payloads for testing

xss\_payloads = [

"<script>alert('XSS')</script>",

"<img src='x' onerror='alert(1)'>",

]

csrf\_payloads = [

{"key": "csrf\_token", "value": "dummy\_token"}

]

sql\_injection\_payloads = [

"' OR '1'='1",

"' AND 1=1 --",

]

def test\_xss(url):

for payload in xss\_payloads:

response = requests.get(url, params={'test': payload})

if payload in response.text:

print(f"[XSS] Potential XSS vulnerability detected at {url} with payload: {payload}")

def test\_csrf(url):

response = requests.get(url)

soup = BeautifulSoup(response.text, 'lxml')

forms = soup.find\_all('form')

for form in forms:

action = form.get('action')

if not action:

action = url

form\_url = urljoin(url, action)

form\_data = {input.get('name', ''): 'test' for input in form.find\_all('input')}

for token in csrf\_payloads:

form\_data[token["key"]] = token["value"]

response = requests.post(form\_url, data=form\_data)

if response.status\_code == 403:

print(f"[CSRF] Potential CSRF vulnerability detected at {url} with form data: {form\_data}")

def test\_sql\_injection(url):

for payload in sql\_injection\_payloads:

response = requests.get(url, params={'search': payload})

if "error" in response.text.lower():

print(f"[SQL Injection] Potential SQL Injection vulnerability detected at {url} with payload: {payload}")

def test\_insecure\_endpoints(url):

response = requests.get(url)

if response.status\_code == 200 and len(response.text) < 1000:

print(f"[Insecure Endpoint] Insecure endpoint detected at {url}: Low response content length")

elif response.status\_code == 403:

print(f"[Insecure Endpoint] Insecure endpoint detected at {url}: Access Forbidden")

elif response.status\_code == 404:

print(f"[Insecure Endpoint] Insecure endpoint detected at {url}: Not Found")

def scan\_web\_app():

for endpoint in endpoints:

url = urljoin(base\_url, endpoint)

print(f"\nScanning {url}...")

test\_xss(url)

test\_csrf(url)

test\_sql\_injection(url)

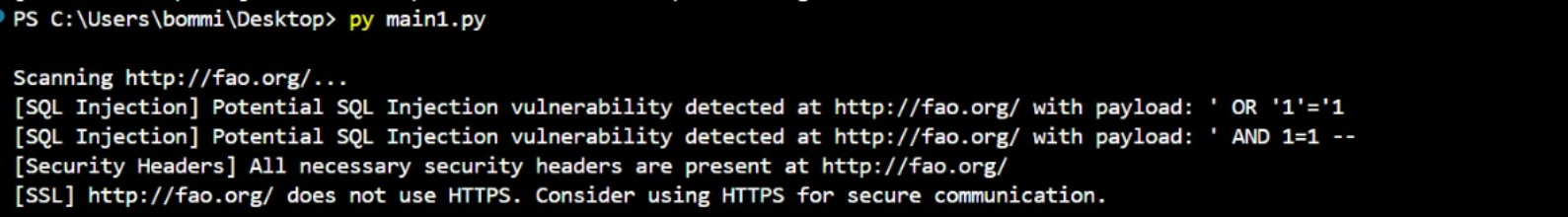
test\_insecure\_endpoints(url)

if \_name\_ == "\_main\_":

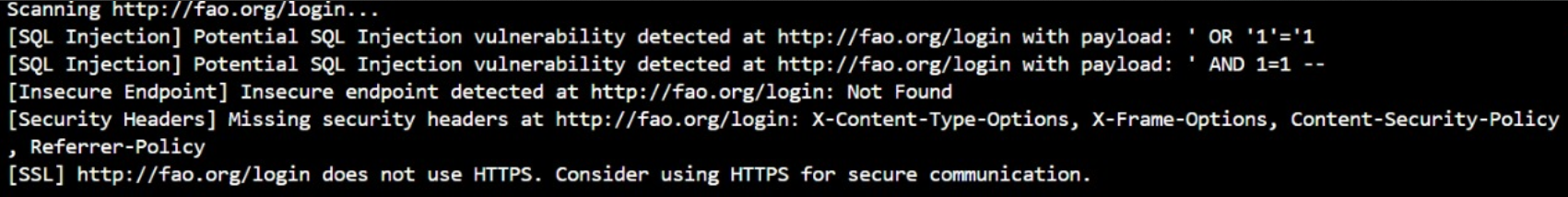
scan\_web\_app()}

# CODE SNIPPETS

For “/” webpage :



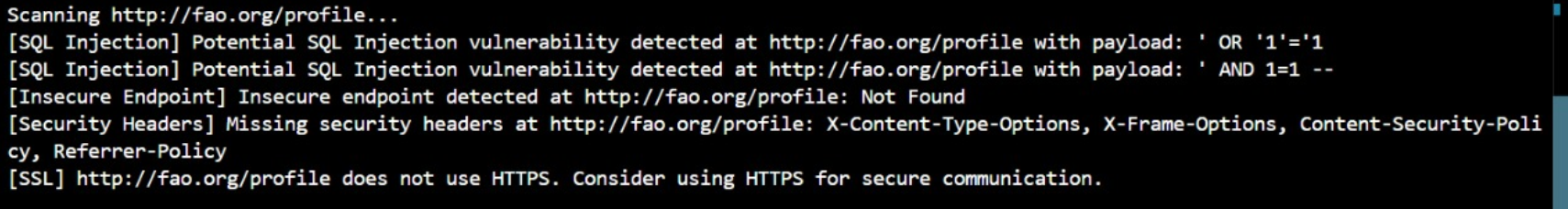
login :



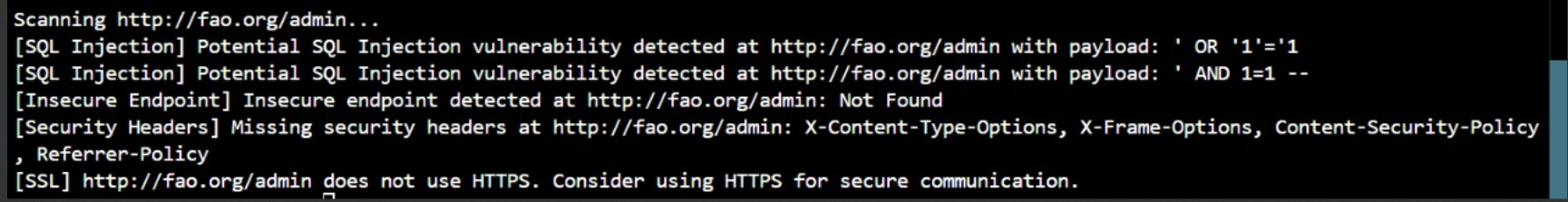
# Search :

# 

# profile :



# Admin :



# Conclusion

The Web Application Security Scanner performs a very important service concerning the security of web applications; it automates the identification process of various vulnerabilities, including cross-site scripting, cross-site request forgery, SQL injection, and insecure endpoints. Complete tests, detailed reports of vulnerabilities, with the possibility to integrate into the development chain, make sure the application will be constantly in the scope of security from the very beginning.

By knitting an increasingly complex web

application and ever-evolving cyber threats together, this forms the secret ingredient to helping development and security teams find and reduce risks as early as possible, keeping successful attacks lower. This will make the web application security scanner enhance the overall security posture of your organization by keeping web applications robust, resilient, and secure against potential vulnerabilities.