**NAME: GUNA SATYA MURARI**

**Email : murariguna3939@gmail.com**

**NUMBER: +919392103939**

**DOMAIN: CYBERSECURITY**

**Project Goals**

* Develop a tool that can scan web applications for common vulnerabilities.
* Understand and implement various techniques for vulnerability detection.
* Gain hands-on experience with web security best practices.

**Sorce code : (webvelscanner.py)**

import requests

from bs4 import BeautifulSoup

from urllib.parse import urljoin, urlparse

# Common payloads

SQL\_PAYLOADS = ["' OR '1'='1", "'; DROP TABLE users; --"]

XSS\_PAYLOADS = ['<script>alert(1)</script>', '" onmouseover="alert(1)"']

VULNS\_FOUND = []

def is\_valid\_url(url):

    return url.startswith("http")

def get\_forms(url):

    try:

        res = requests.get(url, timeout=5)

        soup = BeautifulSoup(res.text, "html.parser")

        return soup.find\_all("form")

    except:

        return []

def get\_form\_details(form):

    details = {}

    try:

        action = form.attrs.get("action", "").strip()

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

        inputs = []

        for input\_tag in form.find\_all("input"):

            input\_type = input\_tag.attrs.get("type", "text")

            input\_name = input\_tag.attrs.get("name")

            inputs.append({"type": input\_type, "name": input\_name})

        details['action'] = action

        details['method'] = method

        details['inputs'] = inputs

    except:

        pass

    return details

def submit\_form(form\_details, base\_url, payload):

    url = urljoin(base\_url, form\_details['action'])

    data = {}

    for input in form\_details["inputs"]:

        if input["type"] == "text" or input["type"] == "search":

            data[input["name"]] = payload

    try:

        if form\_details["method"] == "post":

            return requests.post(url, data=data)

        else:

            return requests.get(url, params=data)

    except:

        return None

def scan\_sql\_injection(url):

    print("[\*] Scanning for SQL Injection...")

    forms = get\_forms(url)

    for form in forms:

        form\_details = get\_form\_details(form)

        for payload in SQL\_PAYLOADS:

            response = submit\_form(form\_details, url, payload)

            if response and ("sql" in response.text.lower() or "syntax" in response.text.lower()):

                print(f"[!] SQL Injection vulnerability found on {url}")

                VULNS\_FOUND.append(("SQL Injection", url, payload))

                break

def scan\_xss(url):

    print("[\*] Scanning for XSS...")

    forms = get\_forms(url)

    for form in forms:

        form\_details = get\_form\_details(form)

        for payload in XSS\_PAYLOADS:

            response = submit\_form(form\_details, url, payload)

            if response and payload in response.text:

                print(f"[!] XSS vulnerability found on {url}")

                VULNS\_FOUND.append(("XSS", url, payload))

                break

def check\_security\_headers(url):

    print("[\*] Checking for security headers...")

    try:

        res = requests.get(url)

        missing = []

        required = [

            "X-Content-Type-Options",

            "X-Frame-Options",

            "Content-Security-Policy",

            "Strict-Transport-Security"

        ]

        for header in required:

            if header not in res.headers:

                missing.append(header)

        if missing:

            print(f"[!] Missing security headers: {', '.join(missing)}")

            VULNS\_FOUND.append(("Missing Headers", url, ", ".join(missing)))

    except:

        pass

def generate\_report():

    print("\n====== Vulnerability Report ======")

    if not VULNS\_FOUND:

        print("No critical vulnerabilities detected.")

    else:

        for vuln in VULNS\_FOUND:

            print(f"\nType: {vuln[0]}\nURL: {vuln[1]}\nPayload/Details: {vuln[2]}")

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

    target = input("Enter target URL (e.g., http://example.com): ").strip()

    if not is\_valid\_url(target):

        print("Invalid URL. Please include http:// or https://")

    else:

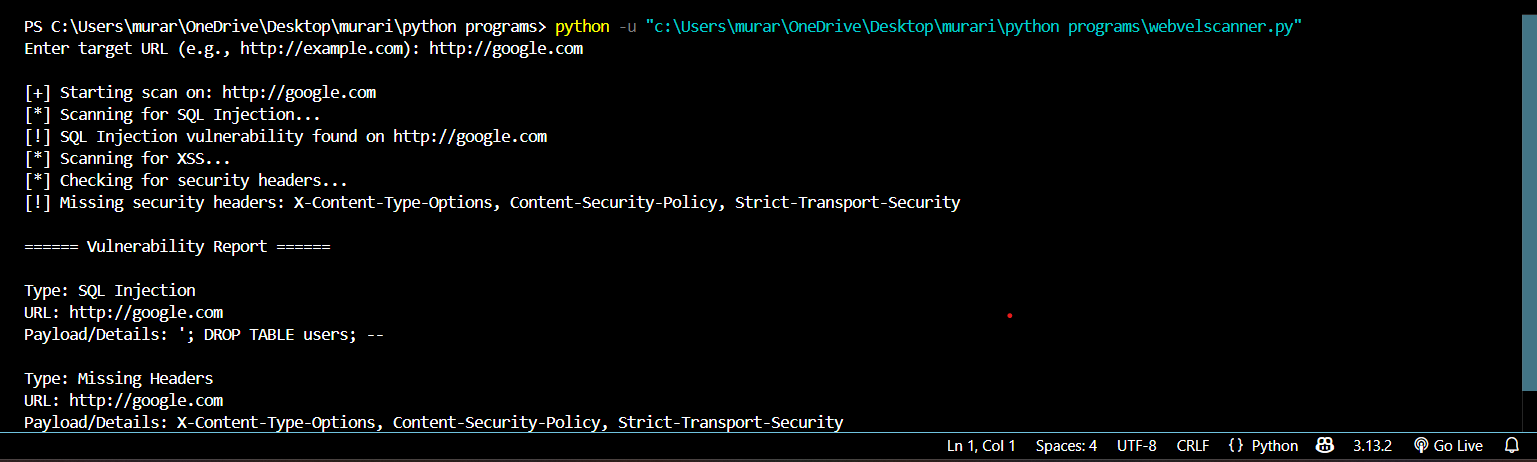
        print(f"\n[+] Starting scan on: {target}")

        scan\_sql\_injection(target)

        scan\_xss(target)

        check\_security\_headers(target)

        generate\_report()

**SAMPLE REPORT:**

**Features**

1. SQL Injection (SQLi) Detection

* Description: Identifies potential SQL injection vulnerabilities by attempting to inject common SQL payloads into input fields or query parameters.
* Example Payloads:
  + ' OR '1'='1
  + ' OR 1=1 --
  + '; DROP TABLE users; --

2. Cross-Site Scripting (XSS) Detection

* Description: Detects XSS vulnerabilities by injecting script-based payloads into input fields and query parameters.
* Example Payloads:
  + <script>alert("XSS")</script>
  + " onmouseover="alert(1)"
  + <img src="x" onerror="alert(1)">

3. Insecure Server Configuration Check

* Description: Analyzes server responses to check for missing security headers and misconfigurations that could lead to vulnerabilities.
* Headers Checked:
  + X-Content-Type-Options
  + X-Frame-Options
  + Strict-Transport-Security
  + Content-Security-Policy

4. Crawl Internal Pages

* Description: Automatically crawls the target web application to identify internal links and scan each for vulnerabilities.
* Depth of Crawling: You can configure the depth of crawling to control how many levels deep the tool should crawl.

5. Report Generation

* Description: After scanning, the tool generates an HTML report that highlights the identified vulnerabilities and provides actionable recommendations for fixing them.
* Report Format: The report includes:
  + Vulnerability Type (e.g., SQLi, XSS, missing headers)
  + Target URL where the vulnerability was found
  + Payload Used for testing the vulnerability

**Setup**

**Prerequisites**

Before setting up the scanner, ensure you have the following:

1. Python 3.6+: The scanner is compatible with Python 3.6 and above.
2. Pip (Python's package manager)

Installing Dependencies

To install the required dependencies, run the following command:

pip install requests beautifulsoup4

* requests: Used for sending HTTP requests to web applications.
* beautifulsoup4: Used for parsing and navigating HTML documents.

**Usage**

Running the Scanner

1. Clone or Download the repository that contains the web\_vuln\_scanner.py script.
2. Navigate to the directory where the script is located.
3. Run the script by executing the following command in your terminal:

python web\_vuln\_scanner.py

1. When prompted, enter the target URL of the web application you want to scan (e.g., http://example.com).