Web Application Vulnerability Scanner

- **Objective:** Build a scanner to detect common web app vulnerabilities like XSS, SQLi, CSRF.
- **Tools:** Python, requests, BeautifulSoup, OWASP top 10 checklist, Flask.
- **Mini Guide:**
    a) Use requests and BeautifulSoup to crawl input fields and URLs.

    b) Inject payloads for XSS, SQLi, etc., and analyze responses.

    c) Use regex or pattern matching for vulnerability detection.

    d) Create a Flask UI to manage scans and view results.

    e) Log each vulnerability with evidence and severity.

- **Deliverables:** Python-based scanner with web interface and detailed reports.

Introduction

Web applications are the backbone of today's digital ecosystem, serving millions of users daily. However, their complexity and exposure to the internet make them attractive targets for attackers. According to OWASP (Open Web Application Security Project), the most common risks include:

- SQL Injection (SQLi): Malicious SQL queries injected into input fields.
- Cross-Site Scripting (XSS): Execution of untrusted scripts in users' browsers.
- Cross-Site Request Forgery (CSRF): Unauthorized actions performed on behalf of an authenticated user.
- Insecure Headers and Cookies: Missing security configurations that allow data theft or session hijacking.

Many existing vulnerability scanners are enterprise-grade and complex. Students and beginners need an educational, lightweight scanner to learn about these risks. This project bridges that gap by implementing a simplified, modular, and extensible scanner.

Tools & Technologies

- Programming Language: Python 3
- Libraries:
- requests → HTTP communication
- BeautifulSoup, lxml → HTML parsing
- Flask (optional) → Web-based UI
- report lab → PDF report generation
- Development Environment: Visual Studio Code / PyCharm.
- Testing Targets: example.com (safe), testphp.vulnweb.com (deliberately vulnerable)

Implementation Details

Crawler Example:

```
def crawl(start_url, max_pages=20):
    seen = set()
    queue = [start_url]
    results = []
    while queue and len(results) < max_pages:
        url = queue.pop(0)
        if url in seen: continue
        seen.add(url)
        resp = requests.get(url, timeout=5)
        results.append((url, resp))
        soup = BeautifulSoup(resp.text, "lxml")
        for a in soup.find_all("a", href=True):
            next_url = urljoin(url, a["href"])
            if start_url in next_url:
                queue.append(next_url)
    return results
```

Passive Checks:

- Missing Content-Security-Policy
- Missing X-Frame-Options
- Cookies without HttpOnly / Secure flags
- Mixed HTTP/HTTPS content

Active Checks (Labs Only):

- Inject "<script>alert(1)</script>" into parameters → Check if reflected.
- Inject "' OR '1'='1" into inputs → Look for SQL error messages.

Results & Findings

Case 1 – Safe Website (example.com):

- Missing CSP header.
- Cookies had no security flags.

Case 2 – Vulnerable Lab (testphp.vulnweb.com):

- Reflected XSS confirmed in search field.
- SQLi payload triggered error message.
- Forms missing CSRF tokens.

References

- OWASP Top 10 (2021) – https://owasp.org/Top10
- Requests Library Documentation – https://docs.python-requests.org/
- BeautifulSoup Documentation – https://www.crummy.com/software/BeautifulSoup/
- Ethical Hacking & Cybersecurity Best Practices