Web Application Penetration Testing Framework

1. Tools & Technologies

- **Python 3**
- **OWASP ZAP** (Zed Attack Proxy) – API-based vulnerability scanner
- **SQLmap** – Automated SQL injection testing tool

2. Features

- Spider and actively scan websites using OWASP ZAP
- Detect SQL Injection vulnerabilities using SQLmap
- Save scan reports in JSON, CSV, and HTML formats
- Highlight XSS and other common vulnerabilities from ZAP alerts

3. Procedure

1. Install ZAP (CLI version):
   Download and install from: https://www.zaproxy.org/download/
   Environment

2. Set up Python:

   ```
   python3 -m venv zapenv
   source zapenv/bin/activate
   ```

3. Running the Framework

   Step 1: Start ZAP Daemon
   ```
   zaproxy -daemon -config api.key=changeme -port 8090
   ```

   Make sure the port matches the one in your script (ZAP_URL = http://localhost:8090).

   Step 2: Run the Python Tool :

   ```
   python zap_sqlmap_tool.py
   ```

Menu Options

The CLI menu will prompt the following actions:

--- Web Pentesting Menu ---
1. Spider Website (ZAP)
2. Active Scan (ZAP)
3. Show ZAP Alerts

4. SQL Injection Test (SQLmap)
5. Exit
6. Save ZAP Alerts to JSON
7. Save ZAP Alerts to CSV
8. Save ZAP Report to HTML

Vulnerability Types Detected

- **SQL Injection** (via SQLmap)
- **Cross-Site Scripting (XSS)**
- **Security misconfigurations**
- **Input validation errors**
- Other common OWASP Top 10 issues
- **Output Reports**

File	Description
zap_alerts.json	JSON dump of all ZAP alerts
zap_alerts.csv	CSV version of ZAP alerts
zap_report.html	Formatted HTML report with ZAP alert details
sqlmap_output.txt	Raw output from SQLmap
sqlmap_report.html	HTML formatted SQLmap output

Web-pentesting-framework

```
├── zap_sqlmap_tool.py
├── requirements.txt
├── zap_alerts.json  (optional)
├── sqlmap_output.txt  (optional)
├── zap_report.html    (optional)
├── sqlmap_report.html (optional)
```

Code:

```
from zapv2 import ZAPv2
import subprocess
import time
import json
import csv
import os
import re

# ZAP setup
API_KEY = 'changeme'
ZAP_URL = 'http://localhost:8090'
```

```
zap = ZAPv2(apikey=API_KEY, proxies={'http': ZAP_URL, 'https': ZAP_URL})

# URL Validation Function
def is_valid_url(url):
    regex = re.compile(
        r'^(?:http|ftp)s?://' # http:// or https://
        r'(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+(?:[A-Z]{2,6}\.?|[A-Z0-9-]{2,}\.))' # domain...
        r'(?::\d+)?' # optional port
        r'(?:/?|[/?]\S+)$', re.IGNORECASE)
    return re.match(regex, url) is not None

# ---------- ZAP FUNCTIONS ----------

def spider_target(url):
    print(f"[+] Spidering target: {url}")
    zap.spider.scan(url)
    time.sleep(2)
    while int(zap.spider.status()) < 100:
print(f"Spider progress: {zap.spider.status()}%")
        time.sleep(1)
    print("[+] Spidering complete.")

def active_scan(url):
    print(f"[+] Starting active scan: {url}")
    scan_id = zap.ascan.scan(url)
    time.sleep(2)

    while int(zap.ascan.status(scan_id)) < 100:
        print(f"Scan progress: {zap.ascan.status(scan_id)}%")
        time.sleep(2)

    print("[+] Active scan complete.")


def show_alerts():
    print("\n[!] ZAP Alerts:")
    alerts = zap.core.alerts()
    for alert in alerts:
        print(f"- {alert['alert']} on {alert['url']} (Risk: {alert['risk']})")

def save_alerts_to_json():
    alerts = zap.core.alerts()
    with open("zap_alerts.json", "w") as f:
        json.dump(alerts, f, indent=4)
```

```
    print("[+] Alerts saved to zap_alerts.json")

def save_alerts_to_csv():
    alerts = zap.core.alerts()
if not alerts:
        print("[!] No alerts found to save.")
        return

    with open("zap_alerts.csv", "w", newline='') as csvfile:
        fieldnames = ["alert", "risk", "url", "description", "solution"]
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()
        for alert in alerts:
            writer.writerow({
                "alert": alert.get("alert", ""),
                "risk": alert.get("risk", ""),
                "url": alert.get("url", ""),
                "description": alert.get("description", ""),
                "solution": alert.get("solution", "")
            })
    print("[+] Alerts saved to zap_alerts.csv")

def save_alerts_to_html():
    alerts = zap.core.alerts()
    if not alerts:
        print("[!] No alerts found to save.")
        return

    with open("zap_report.html", "w") as f:
        f.write("<html><head><title>ZAP Report</title></head><body>")
        f.write("<h1>ZAP Scan Alerts</h1>")
        for alert in alerts:
            f.write(f"<h2>{alert.get('alert')}</h2>")
f.write(f"<p><strong>Risk:</strong> {alert.get('risk')}</p>")
            f.write(f"<p><strong>URL:</strong> {alert.get('url')}</p>")
            f.write(f"<p><strong>Description:</strong> {alert.get('description')}</p>")
            f.write(f"<p><strong>Solution:</strong> {alert.get('solution')}</p>")
            f.write("<hr>")
        f.write("</body></html>")
    print("[+] ZAP HTML report saved to zap_report.html")


# ---------- SQLMAP FUNCTION ----------
```

```
def run_sqlmap(url):
    if not is_valid_url(url):
        print("[!] Invalid URL format.")
        return
    print(f"[+] Running SQLmap on {url} ...\n")
    output_file = "sqlmap_output.txt"
    with open(output_file, "w") as f:
        subprocess.run([
            "sqlmap",
            "-u", url,
            "--forms",
            "--crawl=2",
            "--level=3",
            "--risk=2",
            "--random-agent",
            "--batch",
"--timeout=30"  # Timeout after 30 seconds for each request
        ], stdout=f, stderr=subprocess.STDOUT)
    print(f"[+] SQLmap output saved to {output_file}")
    generate_sqlmap_html_report(output_file)

# ---------- SQLMAP HTML REPORT ----------

def generate_sqlmap_html_report(log_path):
    if not os.path.exists(log_path):
        print("[!] No SQLmap output file found.")
        return

    with open(log_path, "r") as f:
        content = f.read()

    with open("sqlmap_report.html", "w") as f:
        f.write("<h1>SQLmap Scan Report</h1>\n")
        f.write("<pre style='background:#f4f4f4;padding:1em;border-radius:5px;'>\n")
        f.write(content)
        f.write("</pre>")
    print("[+] SQLmap HTML report saved to sqlmap_report.html")

# ---------- MAIN MENU ----------

def main():
    while True:
        print("\n--- Web Pentesting Menu ---")
```

```
        print("1. Spider Website (ZAP)")
        print("2. Active Scan (ZAP)")
        print("3. Show ZAP Alerts")
        print("4. SQL Injection Test (SQLmap)")
        print("5. Exit")
        print("6. Save ZAP Alerts to JSON")
        print("7. Save ZAP Alerts to CSV")
        print("8. Save ZAP Report to HTML")
        choice = input("Select an option: ")

        if choice == "5":
            print("Exiting.")
            break

        elif choice in ["1", "2", "4"]:
            url = input("Enter target URL: ").strip()

        if choice == "1":
            spider_target(url)
        elif choice == "2":
            active_scan(url)
        elif choice == "3":
            show_alerts()
        elif choice == "4":
            run_sqlmap(url)
        elif choice == "6":
            save_alerts_to_json()
        elif choice == "7":
            save_alerts_to_csv()
        elif choice == "8":
            save_alerts_to_html()
else:
            print("[!] Invalid option.")

if __name__ == "__main__":
    main()
```

4. Screen shots

```
┌──(saranya㉿kali)-[~]
└─$ cd web-pentest-framework


┌──(saranya㉿kali)-[~/web-pentest-framework]
└─$ python3 -m venv zapenv

┌──(saranya㉿kali)-[~/web-pentest-framework]
└─$ source zapenv/bin/activate

┌──(zapenv)─(saranya㉿kali)-[~/web-pentest-framework]
└─$ python zap_sqlmap_tool.py
python: can't open file '/home/saranya/web-pentest-framework/zap_sqlmap_tool.py': [Errno 2] No such file or directory

┌──(zapenv)─(saranya㉿kali)-[~/web-pentest-framework]
└─$ ls
main.py  main.py.save  sqlmap_output.txt  sqlmap_report.html  zapenv

┌──(zapenv)─(saranya㉿kali)-[~/web-pentest-framework]
└─$ python main.py

── Web Pentesting Menu ──
1. Spider Website (ZAP)
2. Active Scan (ZAP)
3. Show ZAP Alerts
4. SQL Injection Test (SQLmap)
5. Exit
6. Save ZAP Alerts to JSON
7. Save ZAP Alerts to CSV
8. Save ZAP Report to HTML
Select an option:
```

```
── Web Pentesting Menu ──
1. Spider Website (ZAP)
2. Active Scan (ZAP)
3. Show ZAP Alerts
4. SQL Injection Test (SQLmap)
5. Exit
6. Save ZAP Alerts to JSON
7. Save ZAP Alerts to CSV
8. Save ZAP Report to HTML
Select an option: 3

[!] ZAP Alerts:
- Content Security Policy (CSP) Header Not Set on http://testphp.vulnweb.com/robots.txt (Risk: Medium)
- Missing Anti-clickjacking Header on http://testphp.vulnweb.com (Risk: Medium)
- Server Leaks Version Information via "Server" HTTP Response Header Field on http://testphp.vulnweb.com/robots.txt (Risk: Low)
- Content Security Policy (CSP) Header Not Set on http://testphp.vulnweb.com/sitemap.xml (Risk: Medium)
- Server Leaks Version Information via "Server" HTTP Response Header Field on http://testphp.vulnweb.com/sitemap.xml (Risk: Low)
- Charset Mismatch (Header Versus Meta Content-Type Charset) on http://testphp.vulnweb.com (Risk: Informational)
- Content Security Policy (CSP) Header Not Set on http://testphp.vulnweb.com (Risk: Medium)
- Absence of Anti-CSRF Tokens on http://testphp.vulnweb.com (Risk: Medium)
- Missing Anti-clickjacking Header on http://testphp.vulnweb.com/categories.php (Risk: Medium)
- Server Leaks Version Information via "Server" HTTP Response Header Field on http://testphp.vulnweb.com (Risk: Low)
- X-Content-Type-Options Header Missing on http://testphp.vulnweb.com (Risk: Low)
- Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) on http://testphp.vulnweb.com (Risk: Low)
- Charset Mismatch (Header Versus Meta Content-Type Charset) on http://testphp.vulnweb.com/categories.php (Risk: Informational)
- Content Security Policy (CSP) Header Not Set on http://testphp.vulnweb.com/categories.php (Risk: Medium)
- Absence of Anti-CSRF Tokens on http://testphp.vulnweb.com/categories.php (Risk: Medium)
- Missing Anti-clickjacking Header on http://testphp.vulnweb.com/index.php (Risk: Medium)
- Charset Mismatch (Header Versus Meta Content-Type Charset) on http://testphp.vulnweb.com/index.php (Risk: Informational)
- Content Security Policy (CSP) Header Not Set on http://testphp.vulnweb.com/index.php (Risk: Medium)
- Server Leaks Version Information via "Server" HTTP Response Header Field on http://testphp.vulnweb.com/categories.php (Risk: Low)
- Absence of Anti-CSRF Tokens on http://testphp.vulnweb.com/index.php (Risk: Medium)
- X-Content-Type-Options Header Missing on http://testphp.vulnweb.com/categories.php (Risk: Low)
```

```
── Web Pentesting Menu ──
1. Spider Website (ZAP)
2. Active Scan (ZAP)
3. Show ZAP Alerts
4. SQL Injection Test (SQLmap)
5. Exit
6. Save ZAP Alerts to JSON
7. Save ZAP Alerts to CSV
8. Save ZAP Report to HTML
Select an option: 4
Enter target URL:  http://testphp.vulnweb.com
[+] Running SQLmap on http://testphp.vulnweb.com ...

[+] SQLmap output saved to sqlmap_output.txt
[+] SQLmap HTML report saved to sqlmap_report.html
```

```
── Web Pentesting Menu ──
1. Spider Website (ZAP)
2. Active Scan (ZAP)
3. Show ZAP Alerts
4. SQL Injection Test (SQLmap)
5. Exit
6. Save ZAP Alerts to JSON
7. Save ZAP Alerts to CSV
8. Save ZAP Report to HTML
Select an option: 1
Enter target URL: http://example.com
[+] Spidering target: http://example.com
[+] Spidering complete.
```

```
Scan progress: 2%
Scan progress: 2%
Scan progress: 2%
Scan progress: 2%
Scan progress: 3%
Scan progress: 3%
Scan progress: 3%
Scan progress: 3%
Scan progress: 3%
Scan progress: 3%
Scan progress: 3%
Scan progress: 4%
Scan progress: 4%
Scan progress: 4%
Scan progress: 4%
Scan progress: 4%
Scan progress: 4%
Scan progress: 6%
Scan progress: 9%
Scan progress: 9%
Scan progress: 10%
Scan progress: 10%
Scan progress: 10%
Scan progress: 11%
Scan progress: 11%
Scan progress: 11%
Scan progress: 12%
Scan progress: 12%
Scan progress: 12%
Scan progress: 12%
Scan progress: 12%
Scan progress: 12%
Scan progress: 12%
```

5. Summary

- Use this tool only on applications **you have permission to test**. Unauthorized scanning is **illegal** and unethical.
- It is a Python-based automated tool for scanning web applications using:
  OWASP ZAP – for spidering, active scanning, and XSS detection.
  SQLmap – for SQL Injection testing.