Task 1: Web Application Security Testing

➢ Objective of the Task

The objective of this task was to gain a foundational understanding of web application security by analyzing an intentionally vulnerable application and identifying common vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). Through hands-on practice, the aim was to understand how attackers exploit web application weaknesses, how these vulnerabilities are detected, and what measures can be implemented to prevent such attacks. This task also developed my skills in vulnerability identification using industry tools.

➢ Tools and Methods Used

To accomplish this task, the following tools and methodologies were employed:

- WebGoat: An intentionally vulnerable web application used for training in web security.
- OWASP ZAP (Zed Attack Proxy): A powerful open-source tool for finding security vulnerabilities in web applications.
- Localhost Environment: WebGoat and ZAP were installed and run on the local machine (Windows).
- Manual Testing Techniques: Basic exploitation methods like SQL payload insertion, HTML/JavaScript injection, and CSRF token observation.

➢ Steps Taken to Complete the Task

1. Setup

1.1 Environment Details :

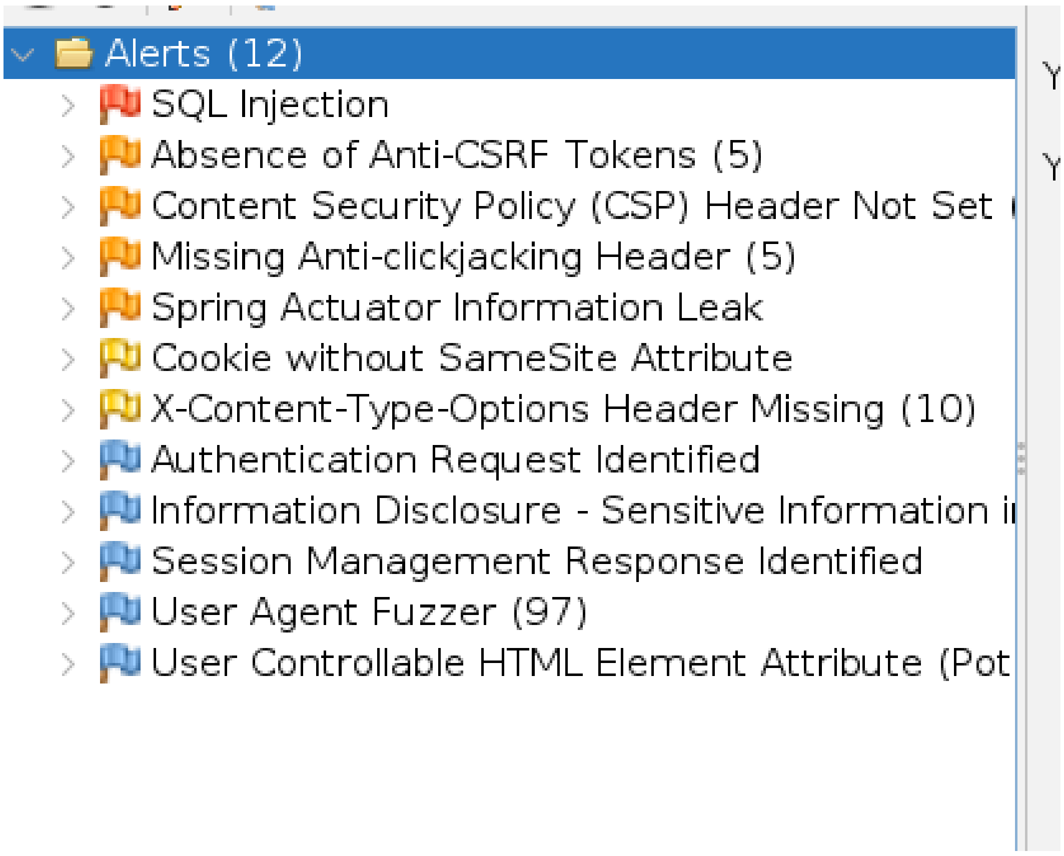
- Operating System: Kali Linux
- Web Application Used: WebGoat 2025
- Browser: Firefox
- Proxy Tool: OWASP ZAP
- Accessed via: http://127.0.0.1:8080/WebGoat

1.2 Configuration

WebGoat was launched using the command `java -jar webgoat-2025.3.jar` and tested using OWASP ZAP by proxying browser traffic.

2. Vulnerability Analysis Using OWASP ZAP

OWASP ZAP was used to perform both passive and active scanning of the WebGoat application. Key vulnerabilities were detected and are listed below.



Alerts (12)
- SQL Injection
- Absence of Anti-CSRF Tokens (5)
- Content Security Policy (CSP) Header Not Set
- Missing Anti-clickjacking Header (5)
- Spring Actuator Information Leak
- Cookie without SameSite Attribute
- X-Content-Type-Options Header Missing (10)
- Authentication Request Identified
- Information Disclosure - Sensitive Information i
- Session Management Response Identified
- User Agent Fuzzer (97)
- User Controllable HTML Element Attribute (Pot

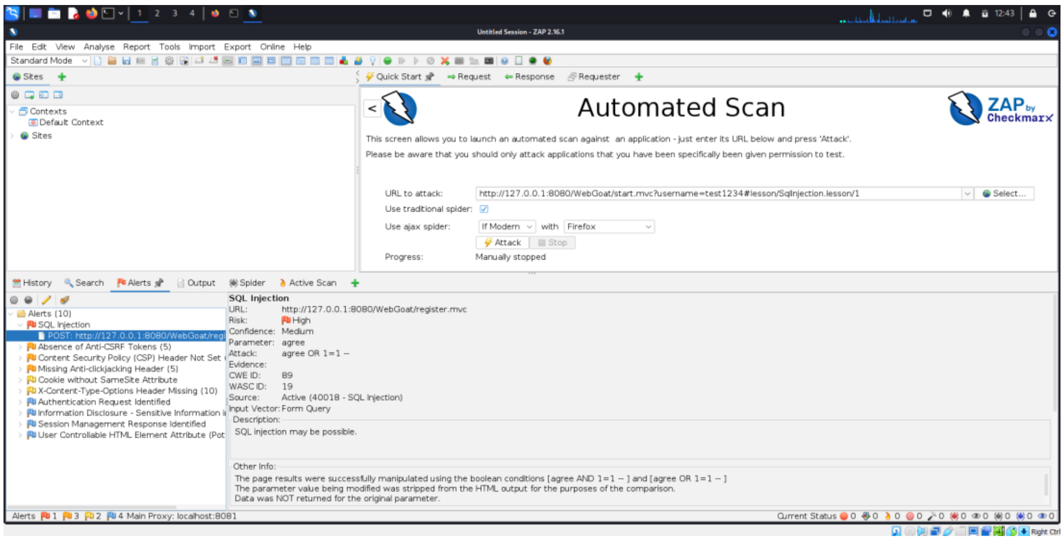
Figure 1: OWASP ZAP vulnerability alerts summary

3. Identified Vulnerabilities

3.1 SQL Injection

- **Location:** /register endpoint
- **Payload Used:**`agree OR 1=1 --`
- **Risk Level:** High

  SQL injection was successfully detected using boolean-based queries. The application did not sanitize user input.
- **Mitigation:** Use prepared statements and validate user input.



This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://127.0.0.1:8080/WebGoat/start.mvc?username=test1234#lesson/SqlInjection.lesson/1
Use traditional spider: ✓
Use ajax spider: If Modern with Firefox
Attack Stop
Progress: Manually stopped

Alerts (10)
- SQL Injection
  - POST: http://127.0.0.1:8080/WebGoat/regi
- Absence of Anti-CSRF Tokens (5)
- Content Security Policy (CSP) Header Not Set
- Missing Anti-clickjacking Header (5)
- Cookie without SameSite Attribute
- X-Content-Type-Options Header Missing (10)
- Authentication Request Identified
- Information Disclosure - Sensitive Information i
- Session Management Response Identified
- User Controllable HTML Element Attribute (Pot

SQL Injection
URL: http://127.0.0.1:8080/WebGoat/register.mvc
Risk: High
Confidence: Medium
Parameter: agree
Attack: agree OR 1=1 --
Evidence:
CWE ID: 89
WASC ID: 19
Source: Active (40018 - SQL Injection)
Input Vector: Form Query
Description:
SQL injection may be possible.

Other Info:
The page results were successfully manipulated using the boolean conditions [agree AND 1=1 --] and [agree OR 1=1 --]
The parameter value being modified was stripped from the HTML output for the purposes of the comparison.
Data was NOT returned for the original parameter.

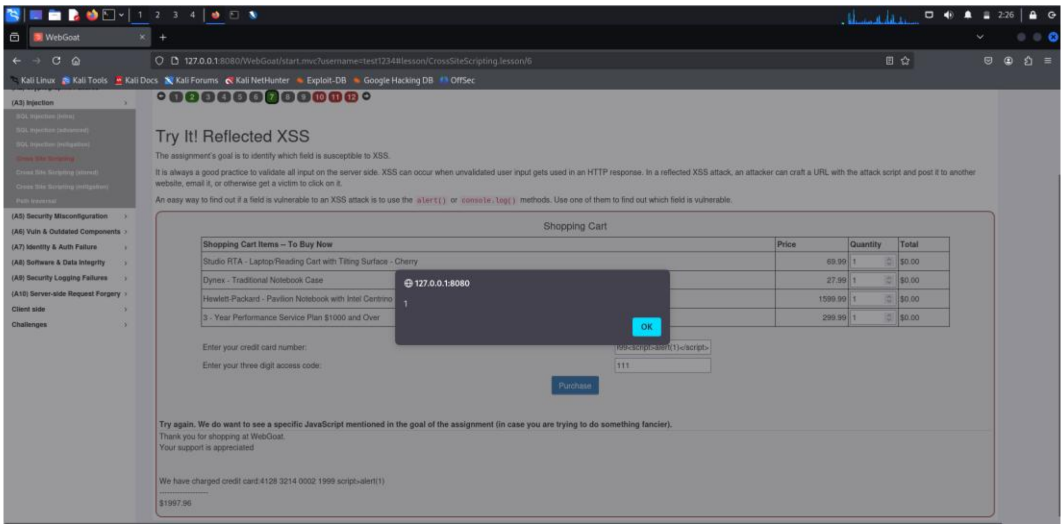
Alerts 1 3 2 4 Main Proxy: localhost:8081
Current Status 0 0 0 0 0 0 0 0 0 0

Figure 2: SQL Injection detection via OWASP ZAP

3.2 Reflected Cross-Site Scripting (XSS)

- **Location:** Shopping cart form
- **Payload Used:** `<script>alert(1)</script>`
- **Risk Level:** High

  XSS vulnerability was confirmed through manual input of a JavaScript payload.
- **Mitigation:** Encode output, use CSP, sanitize inputs.


WebGoat
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
(A3) Injection
(A5) Security Misconfiguration
(A6) Vuln & Outdated Components
(A7) Identity & Auth Failure
(A8) Software & Data Integrity
(A9) Security Logging Failures
(A10) Server-side Request Forgery
Client side
Challenges

127.0.0.1:8080/WebGoat/start.mvc?username=test1234#lesson/CrossSiteScripting.lesson/6

Try It! Reflected XSS

The assignment's goal is to identify which field is susceptible to XSS.

It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input gets used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

An easy way to find out if a field is vulnerable to an XSS attack is to use the `alert()` or `console.log()` methods. Use one of them to find out which field is vulnerable.

Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	$0.00
Dynex - Traditional Notebook Case	27.99	1	$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	$0.00
3 - Year Performance Service Plan $1000 and Over	299.99	1	$0.00

127.0.0.1:8080
1
OK

Enter your credit card number: 199<script>alert(1)</script>
Enter your three digit access code: 111
Purchase

Try again. We do want to see a specific JavaScript mentioned in the goal of the assignment (in case you are trying to do something fancier).
Thank you for shopping at WebGoat.
Your support is appreciated

We have charged credit card:4128 3214 0002 1999 script>alert(1)

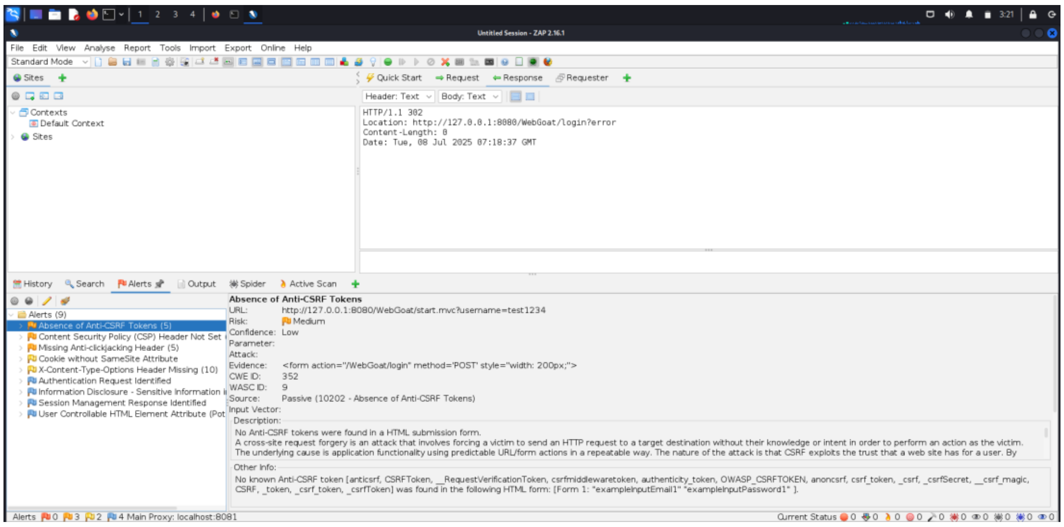
$1997.96

Figure 3: Successful Reflected XSS attack

3.3 Cross-Site Request Forgery (CSRF)

- **Location:** Login and other POST-based forms
- **Risk Level:** Medium

  ZAP found missing Anti-CSRF tokens in POST requests.
- **Mitigation:** Use CSRF tokens, validate Origin headers, set SameSite cookies.



```
HTTP/1.1 302
Location: http://127.0.0.1:8080/WebGoat/login?error
Content-Length: 0
Date: Tue, 08 Jul 2025 07:18:37 GMT
```

Alerts (9)
- Absence of Anti-CSRF Tokens (5)
- Content Security Policy (CSP) Header Not Set
- Missing Anti-clickjacking Header (5)
- Cookie without SameSite Attribute
- X-Content-Type-Options Header Missing (10)
- Authentication Request Identified
- Information Disclosure - Sensitive Information i
- Session Management Response Identified
- User Controllable HTML Element Attribute (Pot

Absence of Anti-CSRF Tokens

URL:	http://127.0.0.1:8080/WebGoat/start.mvc?username=test1234
Risk:	Medium
Confidence:	Low
Parameter:	
Attack:	
Evidence:	<form action="/WebGoat/login" method='POST' style="width: 200px;">
CWE ID:	352
WASC ID:	9
Source:	Passive (10202 - Absence of Anti-CSRF Tokens)
Input Vector:	

Description:
No Anti-CSRF tokens were found in a HTML submission form.
A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By

Other Info:
No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token, _csrfToken] was found in the following HTML form: [Form 1: "exampleInputEmail1" "exampleInputPassword1"].

Alerts 0 3 2 4 Main Proxy: localhost:8081 Current Status 0 0 0 0 0 0 0 0 0 0

Figure 4: CSRF Token Missing alert

3.4 Missing Security Headers

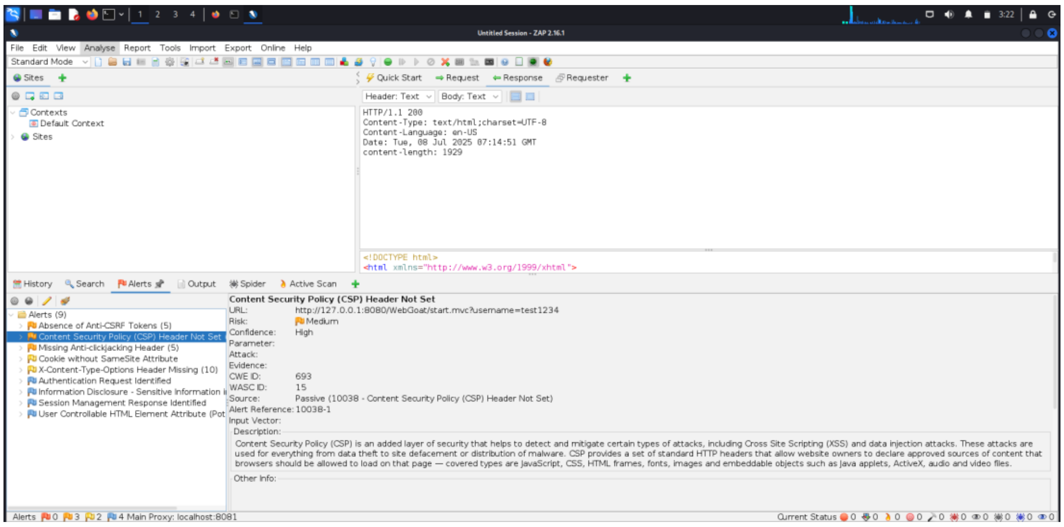
OWASP ZAP reported the absence of key headers such as:

- Content Security Policy (CSP)

- X-Content-Type-Options

- Anti-clickjacking headers

- **Risk Level:** Medium to High
- **Mitigation:** Add required headers in HTTP responses to restrict unsafe behaviors.



```
HTTP/1.1 200
Content-Type: text/html;charset=UTF-8
Content-Language: en-US
Date: Tue, 08 Jul 2025 07:14:51 GMT
content-length: 1929
```

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
```

Alerts (9)
- Absence of Anti-CSRF Tokens (5)
- Content Security Policy (CSP) Header Not Set
- Missing Anti-clickjacking Header (5)
- Cookie without SameSite Attribute
- X-Content-Type-Options Header Missing (10)
- Authentication Request Identified
- Information Disclosure - Sensitive Information i
- Session Management Response Identified
- User Controllable HTML Element Attribute (Pot

Content Security Policy (CSP) Header Not Set

URL:	http://127.0.0.1:8080/WebGoat/start.mvc?username=test1234
Risk:	Medium
Confidence:	High
Parameter:	
Attack:	
Evidence:	
CWE ID:	693
WASC ID:	15
Source:	Passive (10038 - Content Security Policy (CSP) Header Not Set)
Alert Reference:	10038-1
Input Vector:	

Description:
Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

Other Info:

Alerts 0 3 2 4 Main Proxy: localhost:8081 Current Status 0 0 0 0 0 0 0 0 0 0

Figure 5: CSP Header not set alert in OWASP ZAP

Conclusion :

This internship project has provided me with a valuable and practical foundation in the field of **cybersecurity**, specifically focusing on **network security** and **web application security**. Through hands-on tasks and the use of real-world tools like **Wireshark**, **OWASP ZAP**, and **WebGoat**, I gained critical insights into how digital systems are vulnerable to various types of attacks and how to protect them through proactive security measures.

In Task 1, I developed an understanding of common **network threats**, such as **viruses, worms, and phishing attacks,** and implemented basic yet effective security configurations in a small network environment. Monitoring network traffic using **Wireshark** helped me identify normal vs. suspicious activity and reinforced the importance of firewall and encryption settings.

In Task 2, I explored the inner workings of **web vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF)** using **OWASP ZAP** and **WebGoat**. I not only learned how these vulnerabilities can be exploited but also how developers and security professionals can mitigate them through best practices in coding and application design.

This experience enhanced my technical knowledge, problem-solving skills, and security awareness. It also strengthened my interest in pursuing a career in cybersecurity and motivated me to continue learning about secure coding, penetration testing, and vulnerability assessment. The skills gained through this project will be a strong foundation for future academic and professional endeavors in the cybersecurity domain.