2024| BY: *Madhumitha .B*

THEREDUSER TASK 2 REPORT

COMPANY NAME : Thereduser.tech

COURSE :Cyber Intern

INTRODUCTION TO WEB APPLICATION SECURITY

➢Introduction

- During my cybersecurity internship, I explored web application security through hands-on learning.
- My primary objective was to understand vulnerabilities such as SQL Injection and Cross-Site Scripting (XSS) using OWASP ZAP and WebGoat, a deliberately vulnerable web application designed for learning purposes.
- I installed WebGoat on a virtual machine and used OWASP ZAP to scan for vulnerabilities, focusing on SQL Injection and XSS.
- Additionally, I attempted to manually exploit the SQL Injection vulnerabilities identified by OWASP ZAP.

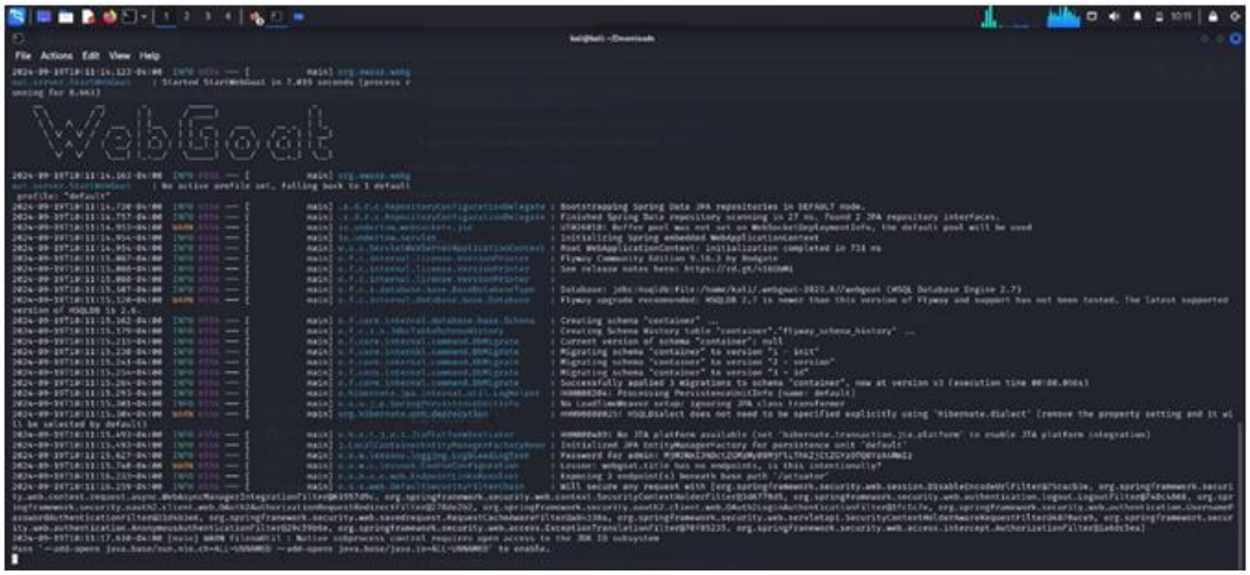
➢Setting up WebGoat

✓ *Objective*

- WebGoat was chosen as the primary learning tool because it is designed to teach common vulnerabilities in web applications.
- It is a deliberately vulnerable web application provided by OWASP, making it a perfect platform for practicing real-world attack scenarios in a controlled environment.

✓ *Steps for Installation*

- Installed WebGoat on a virtual machine running [insert your OS].
- Set up the virtual environment and downloaded WebGoat.
- Configured the WebGoat server and accessed it via `localhost` in the browser.



```
2024-09-19T10:11:14.123-04:00  INFO [illegible] --- [           main] org.owasp.webg
oat.server.StartWebGoat        : Started StartWebGoat in 7.[illegible] seconds (process r
unning for 8.[illegible])

 WebGoat

2024-09-19T10:11:14.162-04:00  INFO [illegible] --- [           main] org.owasp.webg
oat.server.StartWebGoat        : No active profile set, falling back to 1 default
 profile: "default"
[illegible]  : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
[illegible]  : Finished Spring Data repository scanning in 27 ms. Found 2 JPA repository interfaces.
[illegible]  : UT026010: Buffer pool was not set on WebSocketDeploymentInfo, the default pool will be used
[illegible]  : Initializing Spring embedded WebApplicationContext
[illegible]  : Root WebApplicationContext: initialization completed in 731 ms
[illegible]  : Flyway Community Edition 9.16.3 by Redgate
[illegible]  : See release notes here: https://rd.gt/[illegible]
[illegible]  : Database: jdbc:hsqldb:file:/home/kali/.webgoat-2023.8//webgoat (HSQL Database Engine 2.7)
[illegible]  : Flyway upgrade recommended: HSQLDB 2.7 is newer than this version of Flyway and support has not been tested. The latest supported
version of HSQLDB is 2.6.
[illegible]  : Creating schema "container" ...
[illegible]  : Creating Schema History table "container"."flyway_schema_history" ...
[illegible]  : Current version of schema "container": null
[illegible]  : Migrating schema "container" to version "1 - init"
[illegible]  : Migrating schema "container" to version "2 - version"
[illegible]  : Migrating schema "container" to version "3 - id"
[illegible]  : Successfully applied 3 migrations to schema "container", now at version v3 (execution time 00:00.[illegible])
[illegible]  : HHH000204: Processing PersistenceUnitInfo [name: default]
[illegible]  : No LoadTimeWeaver setup: ignoring JPA class transformer
[illegible]  : HHH90000025: HSQLDialect does not need to be specified explicitly using 'hibernate.dialect' (remove the property setting and it wi
ll be selected by default)
[illegible]  : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
[illegible]  : Initialized JPA EntityManagerFactory for persistence unit 'default'
[illegible]  : Password for admin: [illegible]
[illegible]  : Lesson: webgoat.title has no endpoints, is this intentionally?
[illegible]  : Exposing 3 endpoint(s) beneath base path '/actuator'
[illegible]  : Will secure any request with [org.springframework.security.web.session.DisableEncodeUrlFilter@[illegible], org.springframework.securi
ty.web.context.request.async.WebAsyncManagerIntegrationFilter@[illegible], org.springframework.security.web.context.SecurityContextHolderFilter@[illegible], [illegible]
[illegible]
2024-09-19T10:11:17.[illegible] [main] WARN [illegible] : Native subprocess control requires open access to the JDK IO subsystem
Pass '--add-opens java.base/sun.nio.ch=ALL-UNNAMED --add-opens java.base/java.io=ALL-UNNAMED' to enable.
```

✓ ***Result***

- The WebGoat application was successfully installed and accessible through a browser, allowing me to begin vulnerability testing and analysis.

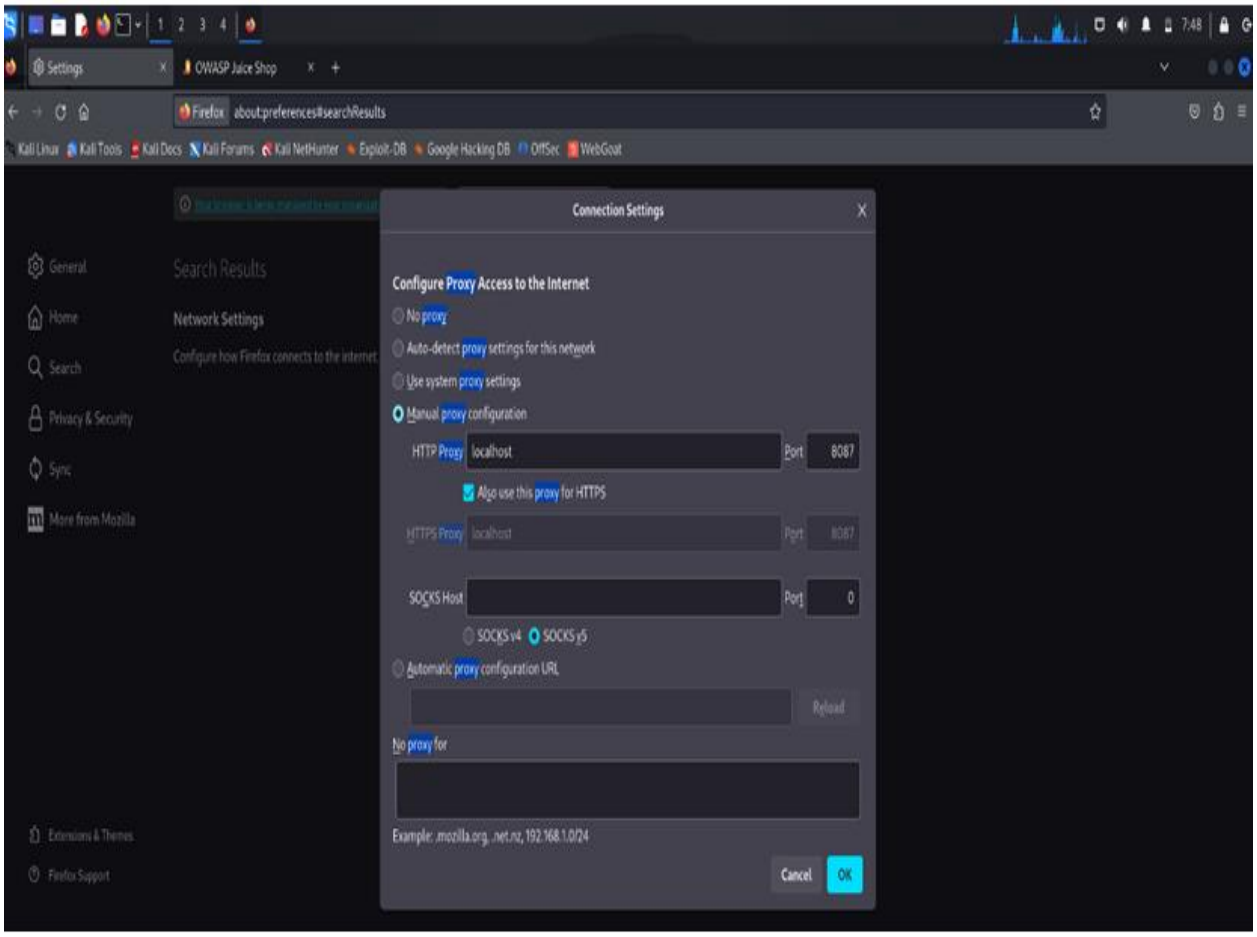
➢OWASP ZAP Vulnerability Scanning

✓ ***Objective,process:***

- The primary objective of using OWASP ZAP was to scan the OWASP JUICE SHOP application for potential vulnerabilities, particularly focusing on SQL Injection and XSS attacks.
- OWASP ZAP is an open source tool widely used for finding security vulnerabilities in web applications.
- Loaded OWASP ZAP and pointed it to the OWASP JUICE SHOP web application URL.
- Performed a full scan, allowing OWASP ZAP to detect vulnerabilities.
- Focused on vulnerabilities like SQL Injection and Cross-Site Scripting, both common and dangerous in modern web applications.

Let see step by step set up

Step 1: Changing browser proxy settings to manual configuration.



Connection Settings

Configure Proxy Access to the Internet

No proxy
Auto-detect proxy settings for this network
Use system proxy settings
[✓ Manual proxy configuration]

HTTP Proxy: localhost Port: 8087

[✓] Also use this proxy for HTTPS

HTTPS Proxy: localhost Port: 8087

SOCKS Host: Port: 0

SOCKS v4 [✓ SOCKS v5]

Automatic proxy configuration URL

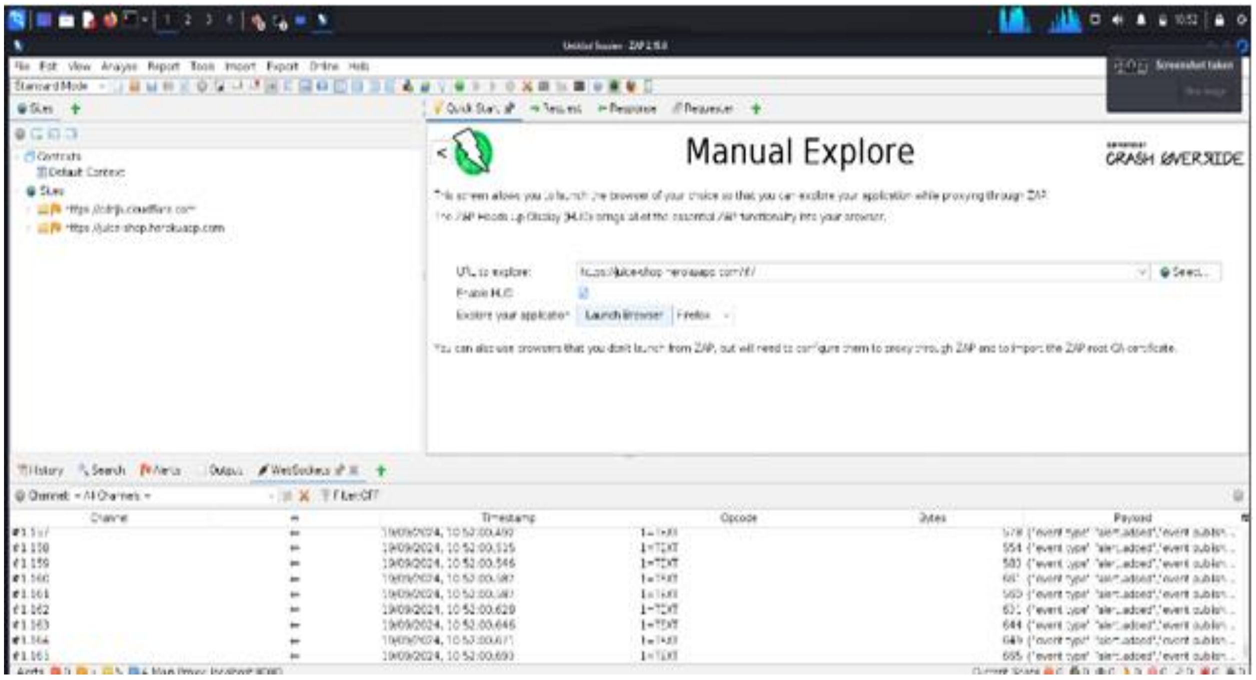
Reload

No proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24

Cancel OK

Step 2: Opening Zap



Manual Explore

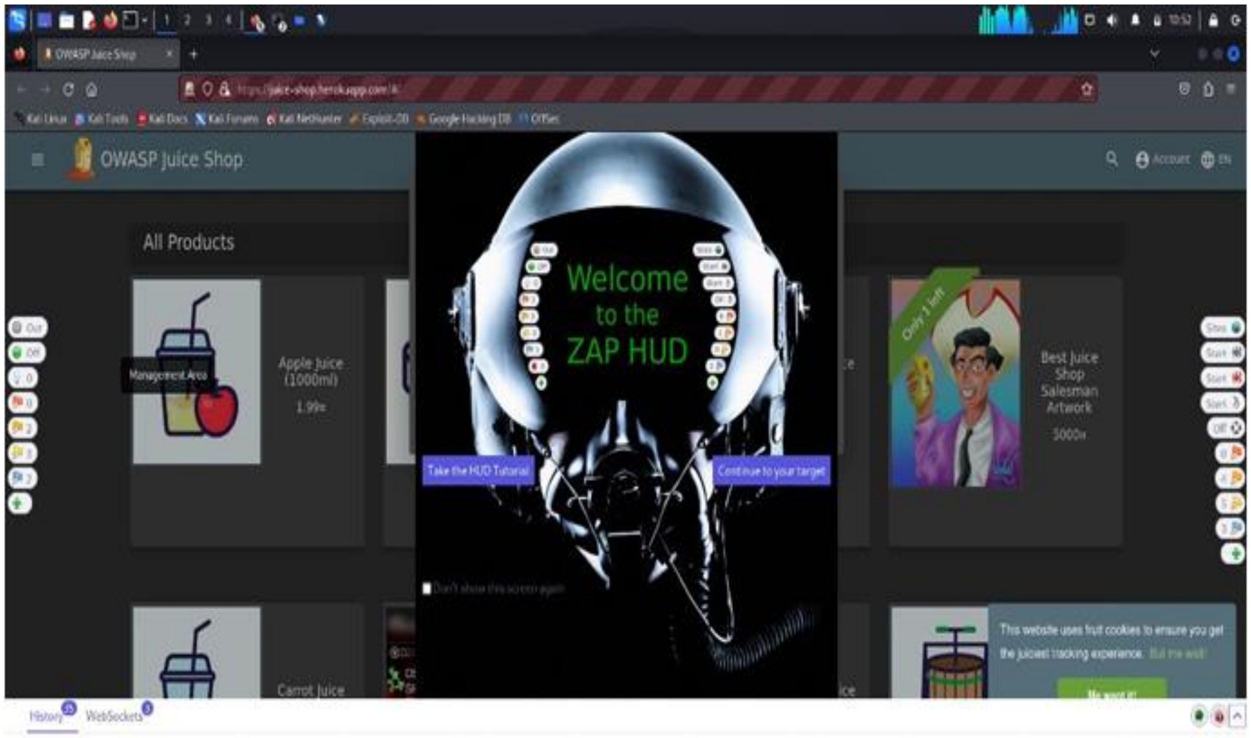
This screen allows you to launch the browser of your choice so that you can explore your application while proxying through ZAP.
The ZAP Heads Up Display (HUD) brings all of the essential ZAP functionality into your browser.

URL to explore: https://juice-shop.herokuapp.com/#/
Enable HUD: ✓
Explore your application: Launch Browser Firefox

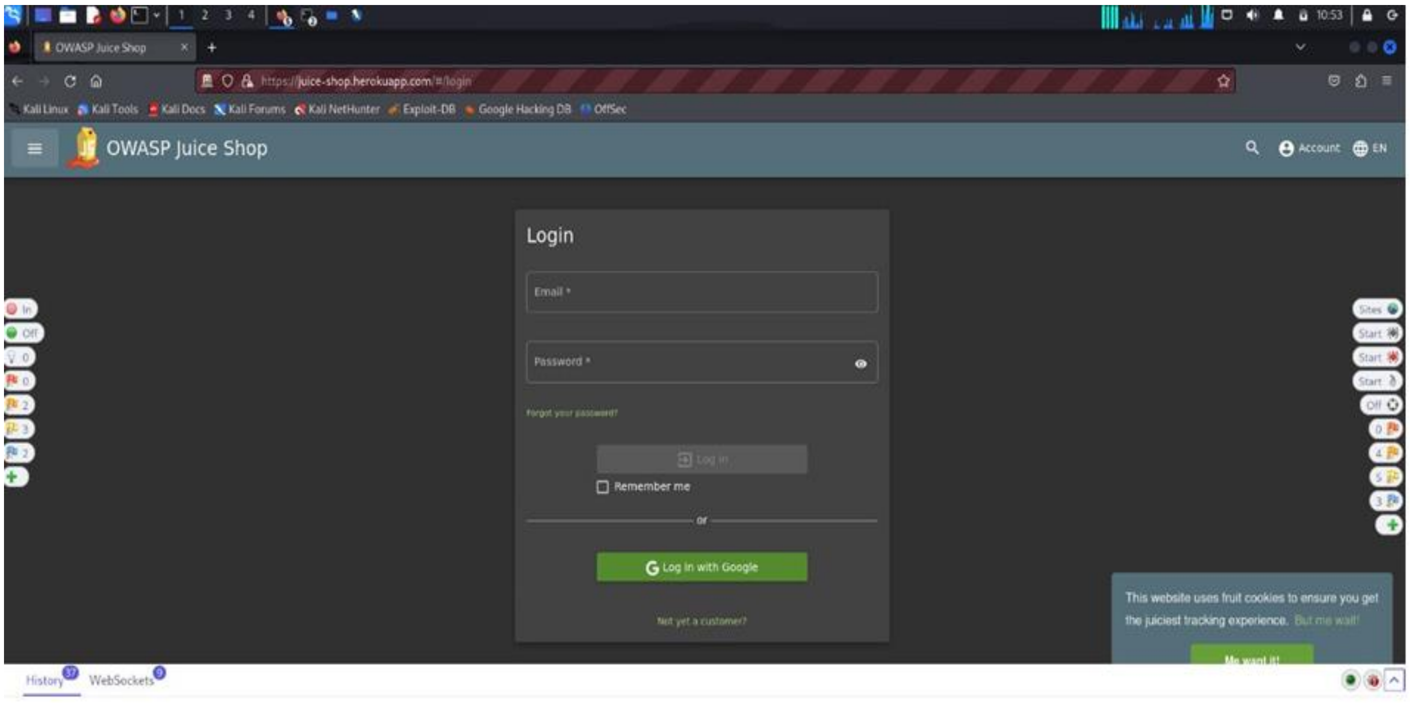
You can also use browsers that you don't launch from ZAP, but will need to configure them to proxy through ZAP and to import the ZAP root CA certificate.

```
Channel      Timestamp                Opcode   Bytes  Payload
#1.157       19/09/2024, 10:52:00.492  1=TEXT   578    {"event-type":"alert.added","event-publish...
#1.158       19/09/2024, 10:52:00.535  1=TEXT   554    {"event-type":"alert.added","event-publish...
#1.159       19/09/2024, 10:52:00.546  1=TEXT   580    {"event-type":"alert.added","event-publish...
#1.160       19/09/2024, 10:52:00.587  1=TEXT   607    {"event-type":"alert.added","event-publish...
#1.161       19/09/2024, 10:52:00.587  1=TEXT   565    {"event-type":"alert.added","event-publish...
#1.162       19/09/2024, 10:52:00.628  1=TEXT   631    {"event-type":"alert.added","event-publish...
#1.163       19/09/2024, 10:52:00.646  1=TEXT   644    {"event-type":"alert.added","event-publish...
#1.164       19/09/2024, 10:52:00.671  1=TEXT   649    {"event-type":"alert.added","event-publish...
#1.165       19/09/2024, 10:52:00.693  1=TEXT   655    {"event-type":"alert.added","event-publish...
```

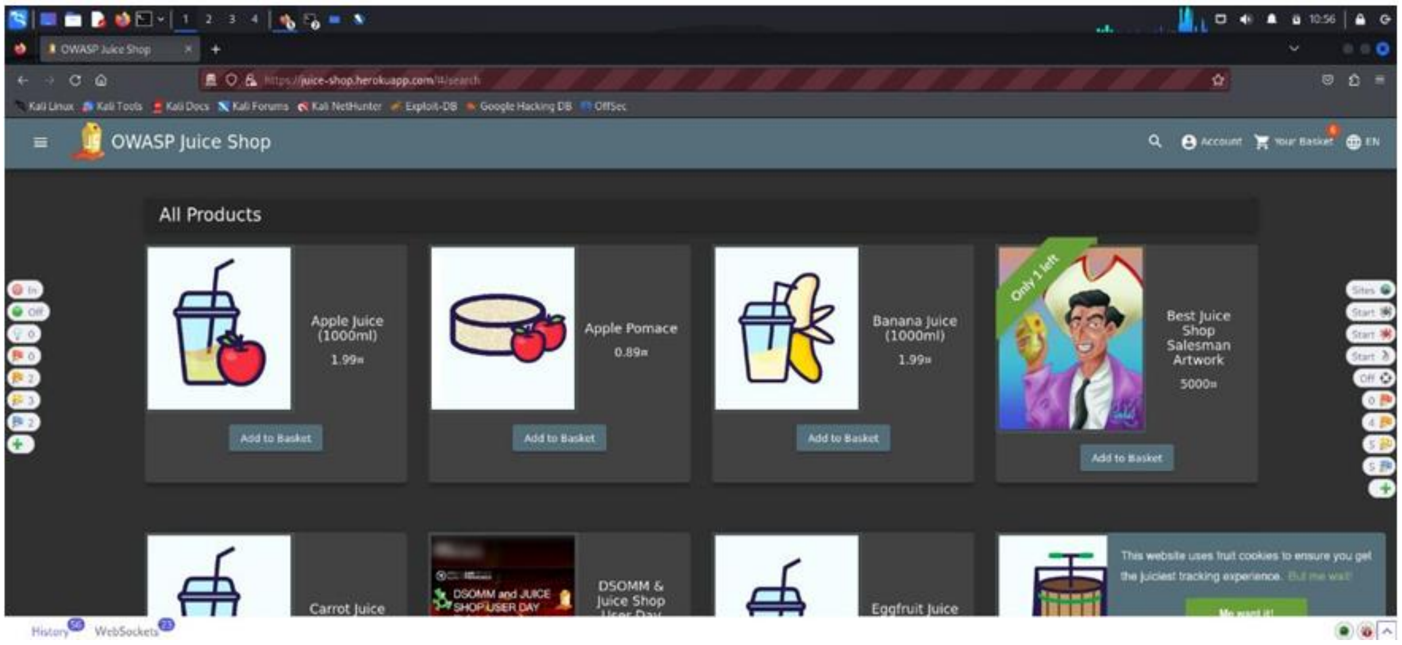
Step 3: pasting Owasp Juice shop URL



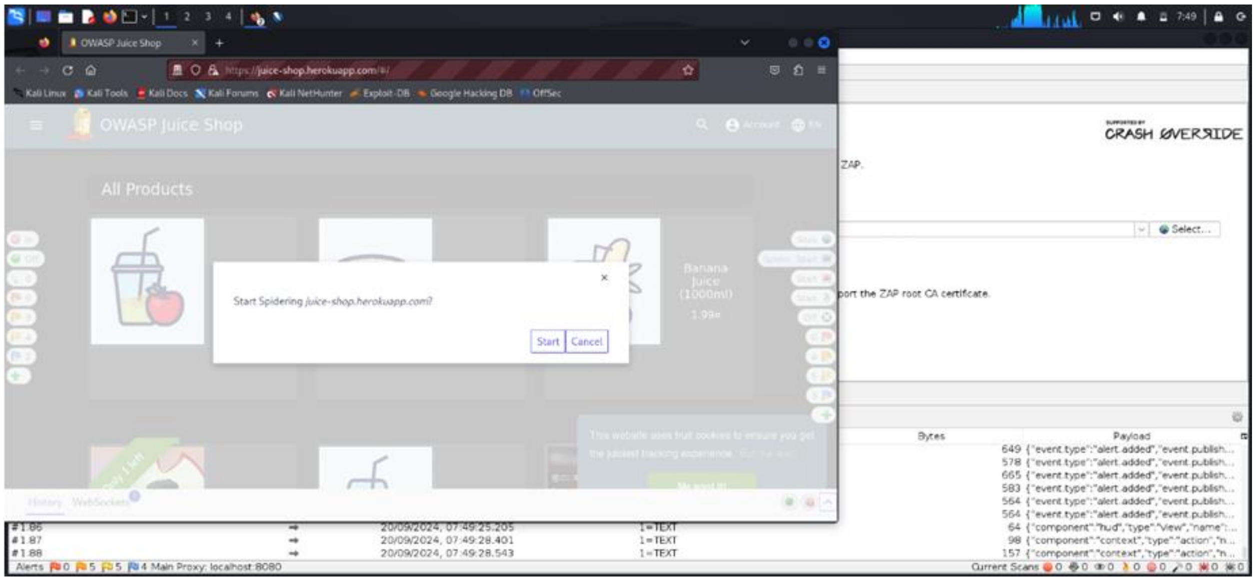
Step 4: Logging in and starting spidering the website.



step 5: After logging you will see the below page



Step 6: Start spidering juice shop which is showed below



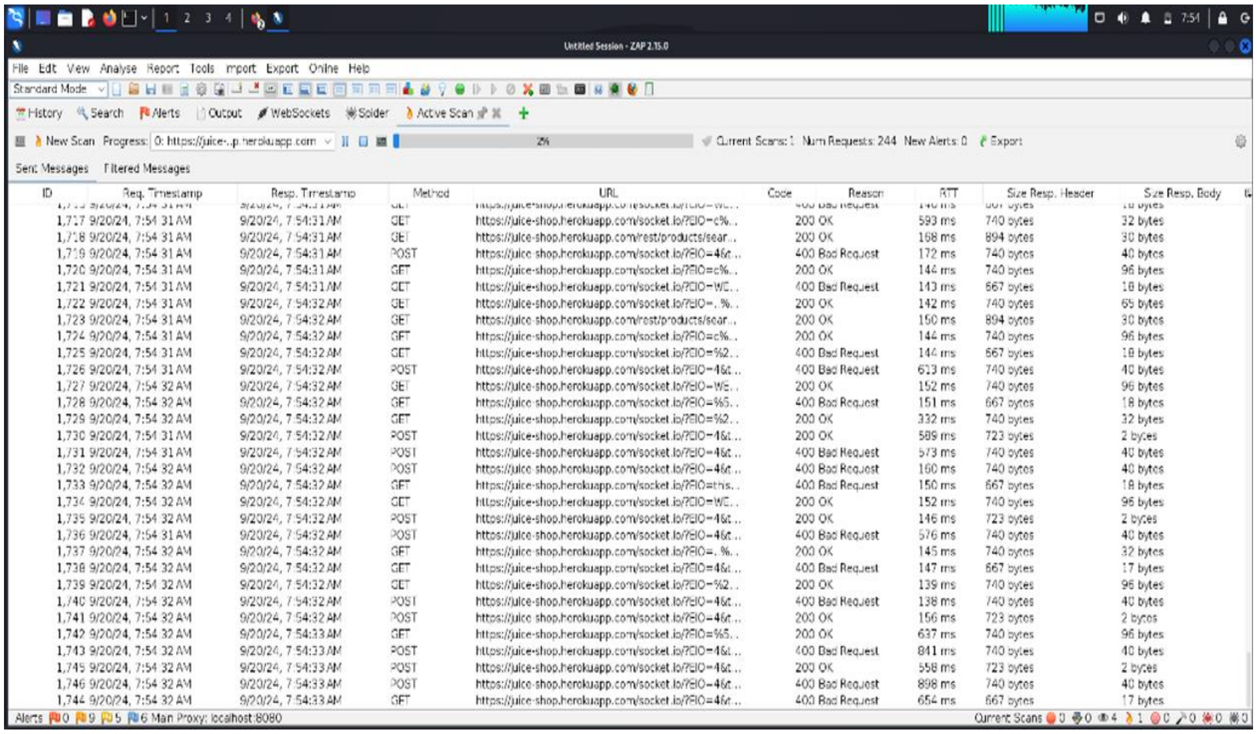
Start Spidering *juice-shop.herokuapp.com?*

Start Cancel

```
#186      ➡    20/09/2024, 07:49:25.205    1=TEXT
#187      ➡    20/09/2024, 07:49:28.401    1=TEXT
#188      ➡    20/09/2024, 07:49:28.543    1=TEXT
Alerts 0 5 5 4 Main Proxy: localhost:8080
```

```
Bytes  Payload
649 {"event.type":"alert.added","event.publish...
578 {"event.type":"alert.added","event.publish...
665 {"event.type":"alert.added","event.publish...
583 {"event.type":"alert.added","event.publish...
564 {"event.type":"alert.added","event.publish...
564 {"event.type":"alert.added","event.publish...
64 {"component":"hud","type":"view","name":...
98 {"component":"context","type":"action","n...
157 {"component":"context","type":"action","n...
Current Scans 0 0 0 0 0 0 0 0
```

Step 7: after spidering you will look for scanning alerts in ZAP



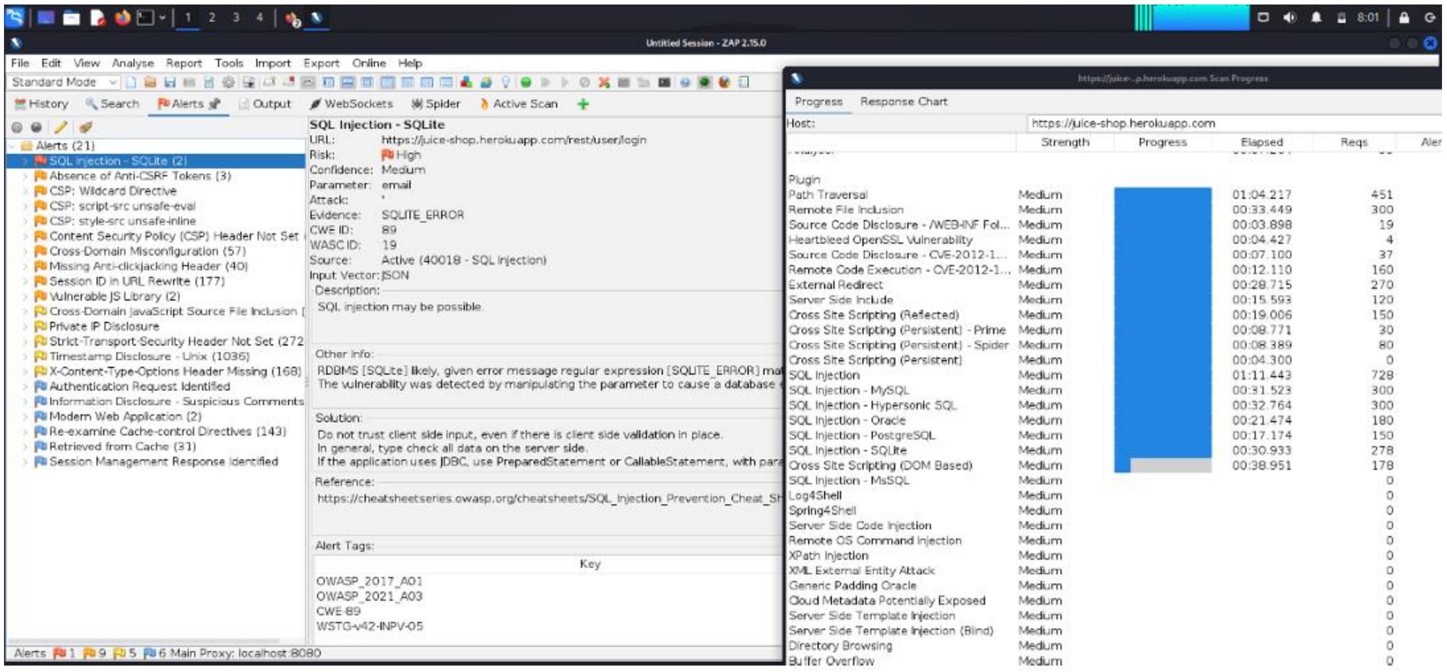
Untitled Session - ZAP 2.15.0

New Scan Progress: 0: https://juice-..p.herokuapp.com 2% Current Scans: 1 Num Requests: 244 New Alerts: 0 Export

ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
1,717	9/20/24, 7:54:31 AM	9/20/24, 7:54:31 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=c%...	200	OK	593 ms	740 bytes	32 bytes
1,718	9/20/24, 7:54:31 AM	9/20/24, 7:54:31 AM	GET	https://juice-shop.herokuapp.com/rest/products/sear...	200	OK	168 ms	894 bytes	30 bytes
1,719	9/20/24, 7:54:31 AM	9/20/24, 7:54:31 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	172 ms	740 bytes	40 bytes
1,720	9/20/24, 7:54:31 AM	9/20/24, 7:54:31 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=c%...	200	OK	144 ms	740 bytes	96 bytes
1,721	9/20/24, 7:54:31 AM	9/20/24, 7:54:31 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=WE...	400	Bad Request	143 ms	667 bytes	18 bytes
1,722	9/20/24, 7:54:31 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=.%...	200	OK	142 ms	740 bytes	65 bytes
1,723	9/20/24, 7:54:31 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/rest/products/sear...	200	OK	150 ms	894 bytes	30 bytes
1,724	9/20/24, 7:54:31 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=c%...	200	OK	144 ms	740 bytes	96 bytes
1,725	9/20/24, 7:54:31 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=%2...	400	Bad Request	144 ms	667 bytes	18 bytes
1,726	9/20/24, 7:54:31 AM	9/20/24, 7:54:32 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	613 ms	740 bytes	40 bytes
1,727	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=WE...	200	OK	152 ms	740 bytes	96 bytes
1,728	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=%5...	400	Bad Request	151 ms	667 bytes	18 bytes
1,729	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=%2...	200	OK	332 ms	740 bytes	32 bytes
1,730	9/20/24, 7:54:31 AM	9/20/24, 7:54:32 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	200	OK	589 ms	723 bytes	2 bytes
1,731	9/20/24, 7:54:31 AM	9/20/24, 7:54:32 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	573 ms	740 bytes	40 bytes
1,732	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	160 ms	740 bytes	40 bytes
1,733	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=this...	400	Bad Request	150 ms	667 bytes	18 bytes
1,734	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=WE...	200	OK	152 ms	740 bytes	96 bytes
1,735	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	200	OK	146 ms	723 bytes	2 bytes
1,736	9/20/24, 7:54:31 AM	9/20/24, 7:54:32 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	576 ms	740 bytes	40 bytes
1,737	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=.%...	200	OK	145 ms	740 bytes	32 bytes
1,738	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	147 ms	667 bytes	17 bytes
1,739	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=%2...	200	OK	139 ms	740 bytes	96 bytes
1,740	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	138 ms	740 bytes	40 bytes
1,741	9/20/24, 7:54:32 AM	9/20/24, 7:54:32 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	200	OK	156 ms	723 bytes	2 bytes
1,742	9/20/24, 7:54:32 AM	9/20/24, 7:54:33 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=%5...	200	OK	637 ms	740 bytes	96 bytes
1,743	9/20/24, 7:54:32 AM	9/20/24, 7:54:33 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	841 ms	740 bytes	40 bytes
1,745	9/20/24, 7:54:32 AM	9/20/24, 7:54:33 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	200	OK	558 ms	723 bytes	2 bytes
1,746	9/20/24, 7:54:32 AM	9/20/24, 7:54:33 AM	POST	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	898 ms	740 bytes	40 bytes
1,744	9/20/24, 7:54:32 AM	9/20/24, 7:54:33 AM	GET	https://juice-shop.herokuapp.com/socket.io/?EIO=4&t...	400	Bad Request	654 ms	667 bytes	17 bytes

Alerts 0 9 5 6 Main Proxy: localhost:8080   Current Scans 0 0 4 1 0 0 0 0

Step 8: after that u have to click SQl injection

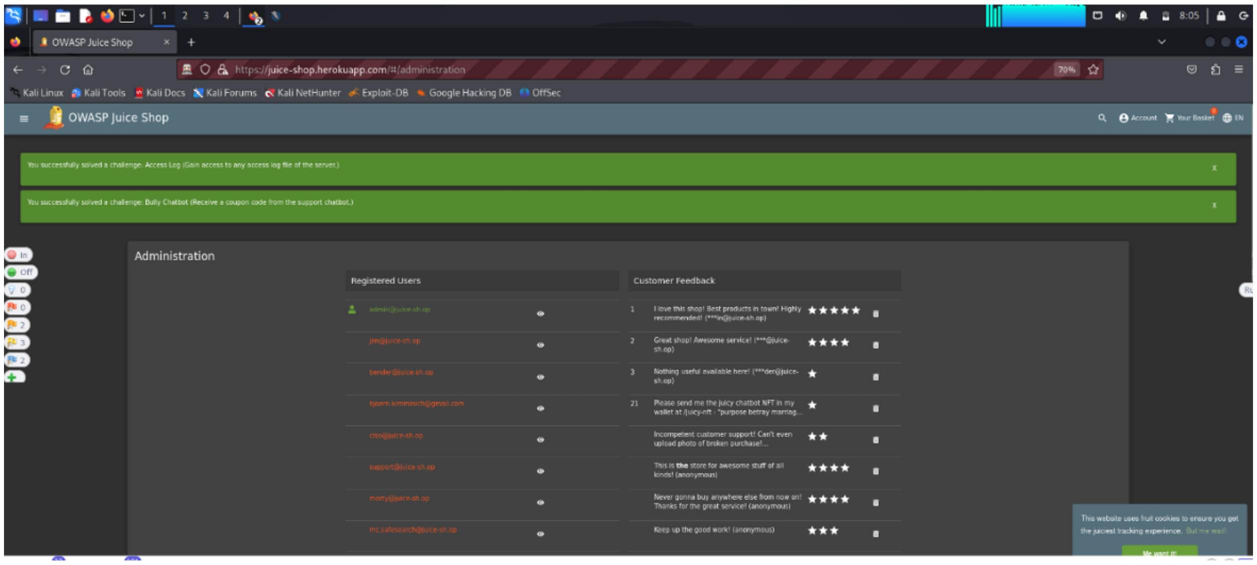

File Edit View Analyse Report Tools Import Export Online Help
Standard Mode
History Search Alerts Output WebSockets Spider Active Scan
Untitled Session - ZAP 2.15.0
https://juice-...p.herokuapp.com Scan Progress
Progress Response Chart

```
Alerts (21)
  SQL Injection - SQLite (2)
  Absence of Anti-CSRF Tokens (3)
  CSP: Wildcard Directive
  CSP: script-src unsafe-eval
  CSP: style-src unsafe-inline
  Content Security Policy (CSP) Header Not Set
  Cross-Domain Misconfiguration (57)
  Missing Anti-clickjacking Header (40)
  Session ID in URL Rewrite (177)
  Vulnerable JS Library (2)
  Cross-Domain JavaScript Source File Inclusion
  Private IP Disclosure
  Strict-Transport-Security Header Not Set (272
  Timestamp Disclosure - Unix (1036)
  X-Content-Type-Options Header Missing (168)
  Authentication Request Identified
  Information Disclosure - Suspicious Comments
  Modern Web Application (2)
  Re-examine Cache-control Directives (143)
  Retrieved from Cache (31)
  Session Management Response Identified
Alerts 1 9 5 6 Main Proxy: localhost:8080
```

```
SQL Injection - SQLite
URL:          https://juice-shop.herokuapp.com/rest/user/login
Risk:         High
Confidence:   Medium
Parameter:    email
Attack:       '
Evidence:     SQLITE_ERROR
CWE ID:       89
WASC ID:      19
Source:       Active (40018 - SQL Injection)
Input Vector: JSON
Description:
 SQL injection may be possible.
Other Info:
 RDBMS [SQLite] likely, given error message regular expression [SQLITE_ERROR] ma
 The vulnerability was detected by manipulating the parameter to cause a database
Solution:
 Do not trust client side input, even if there is client side validation in place.
 In general, type check all data on the server side.
 If the application uses JDBC, use PreparedStatement or CallableStatement, with para
Reference:
 https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sh
Alert Tags:
 Key
 OWASP_2017_A01
 OWASP_2021_A03
 CWE-89
 WSTG-v42-INPV-05
```

```
Host: https://juice-shop.herokuapp.com
                                    Strength   Progress   Elapsed     Reqs   Aler
Plugin
Path Traversal                      Medium                01:04.217    451
Remote File Inclusion               Medium                00:33.449    300
Source Code Disclosure - /WEB-INF Fol... Medium           00:03.898     19
Heartbleed OpenSSL Vulnerability    Medium                00:04.427      4
Source Code Disclosure - CVE-2012-1... Medium             00:07.100     37
Remote Code Execution - CVE-2012-1... Medium              00:12.110    160
External Redirect                   Medium                00:28.715    270
Server Side Include                 Medium                00:15.593    120
Cross Site Scripting (Reflected)    Medium                00:19.006    150
Cross Site Scripting (Persistent) - Prime Medium          00:08.771     30
Cross Site Scripting (Persistent) - Spider Medium         00:08.389     80
Cross Site Scripting (Persistent)   Medium                00:04.300      0
SQL Injection                       Medium                01:11.443    728
SQL Injection - MySQL               Medium                00:31.523    300
SQL Injection - Hypersonic SQL      Medium                00:32.764    300
SQL Injection - Oracle              Medium                00:21.474    180
SQL Injection - PostgreSQL          Medium                00:17.174    150
SQL Injection - SQLite              Medium                00:30.933    278
Cross Site Scripting (DOM Based)    Medium                00:38.951    178
SQL Injection - MsSQL               Medium                               0
Log4Shell                           Medium                               0
Spring4Shell                        Medium                               0
Server Side Code Injection          Medium                               0
Remote OS Command Injection         Medium                               0
XPath Injection                     Medium                               0
XML External Entity Attack          Medium                               0
Generic Padding Oracle              Medium                               0
Cloud Metadata Potentially Exposed  Medium                               0
Server Side Template Injection      Medium                               0
Server Side Template Injection (Blind) Medium                            0
Directory Browsing                  Medium                               0
Buffer Overflow                     Medium                               0
```

Step9: you will see successfully solved challenges


OWASP Juice Shop
https://juice-shop.herokuapp.com/#/administration
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Account Your Basket EN

You successfully solved a challenge: Access Log (Gain access to any access log file of the server.)

You successfully solved a challenge: Bully Chatbot (Receive a coupon code from the support chatbot.)

Administration

Registered Users
admin@juice-sh.op
jim@juice-sh.op
bender@juice-sh.op
bjoern.kimminich@gmail.com
ciso@juice-sh.op
support@juice-sh.op
morty@juice-sh.op
mc.safesearch@juice-sh.op

Customer Feedback		
1	I love this shop! Best products in town! Highly recommended! (***in@juice-sh.op)	★★★★★
2	Great shop! Awesome service! (***@juice-sh.op)	★★★★
3	Nothing useful available here! (***der@juice-sh.op)	★
21	Please send me the juicy chatbot NFT in my wallet at /juicy-nft : "purpose betray marriag...	★
	Incompetent customer support! Can't even upload photo of broken purchase!...	★★
	This is **the** store for awesome stuff of all kinds! (anonymous)	★★★★
	Never gonna buy anywhere else from now on! Thanks for the great service! (anonymous)	★★★★
	Keep up the good work! (anonymous)	★★★

This website uses fruit cookies to ensure you get the juiciest tracking experience.

From the above process ,I identified 2 vulnerabilites

- ✓ **SQL injection**

- ✓ **Cross site scripting**

SQL Injection Vulnerability Analysis

I focused on SQL Injection vulnerabilities, identified as a critical issue by OWASP ZAP.

This vulnerability allows attackers to send malicious SQL commands through input fields to manipulate the database.

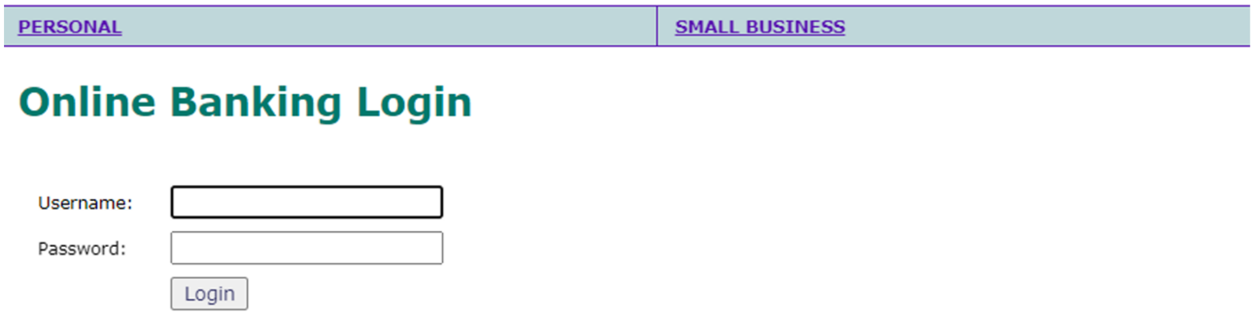
Exploitation Attempt

After identifying the SQL Injection vulnerability, I manually attempted to exploit it using SQL commands like:

sql ' OR 1=1 -- I

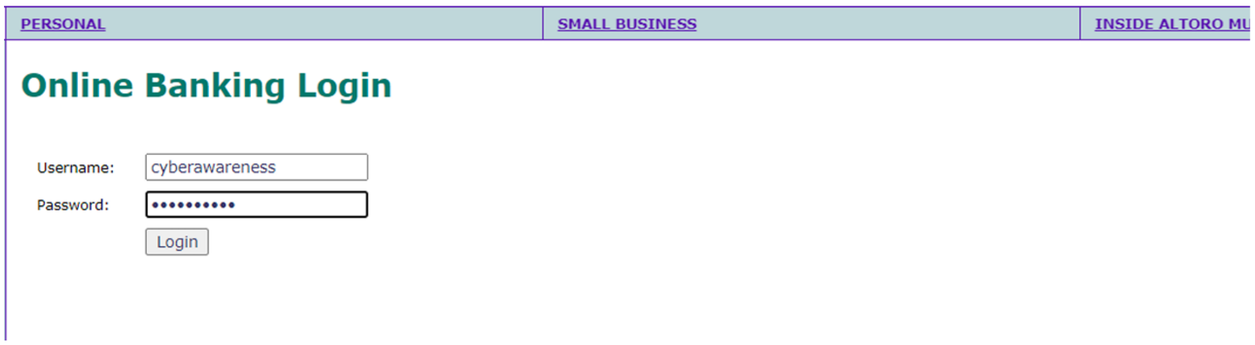
used this command in the WebGoat login page to bypass authentication, which resulted in successful exploitation, demonstrating the vulnerability.

And entering user name password



Online Banking Login

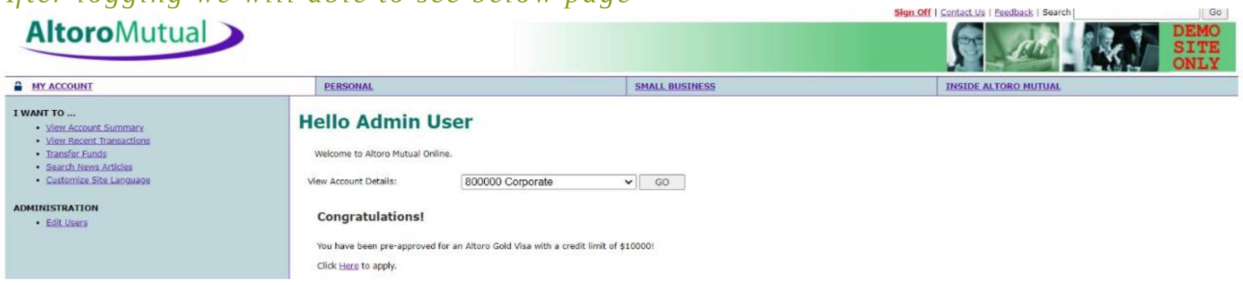
Username:	
Password:	



Online Banking Login

Username:	cyberawareness
Password:	••••••••••

After logging we will able to see below page



Hello Admin User

Welcome to Altoro Mutual Online.

View Account Details: 800000 Corporate

Congratulations!

You have been pre-approved for an Altoro Gold Visa with a credit limit of $10000!

Click Here to apply.

The SQL Injection attack successfully bypassed Altoro Mutual's authentication system, proving the presence of a vulnerability that could allow an attacker to gain unauthorized access.

Cross-Site Scripting (XSS) Vulnerability Analysis

Cross-Site Scripting (XSS) vulnerabilities were also identified during the OWASP ZAP scan.

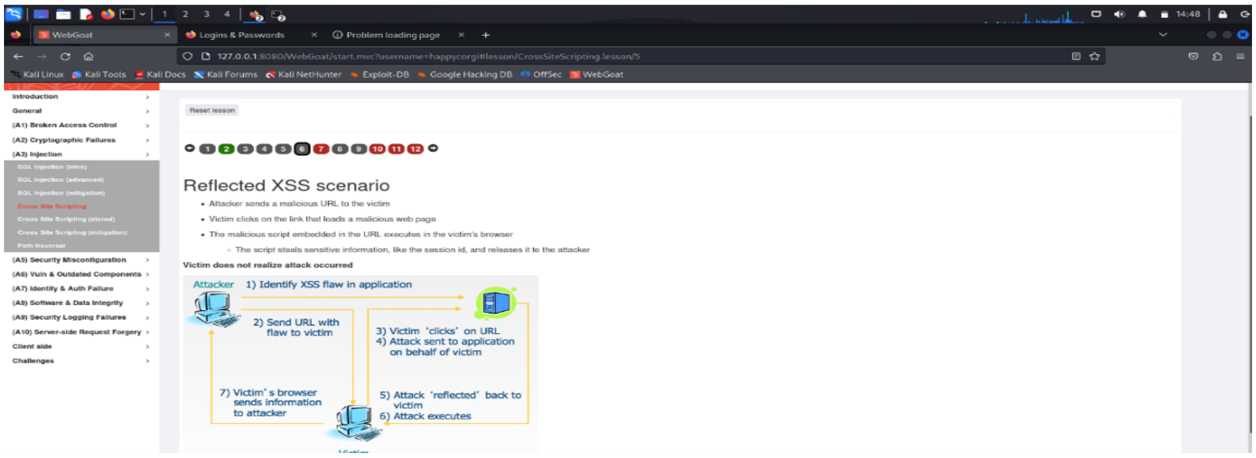
XSS allows attackers to inject malicious scripts into web pages viewed by other users, potentially leading to session hijacking, defacement, or data theft.

Exploitation Attempt

I attempted to inject a simple script such as

“><script>alert(1)</script>

into a WebGoat form field. The injected script was successfully executed in the victim's browser, confirming the presence of an XSS vulnerability.

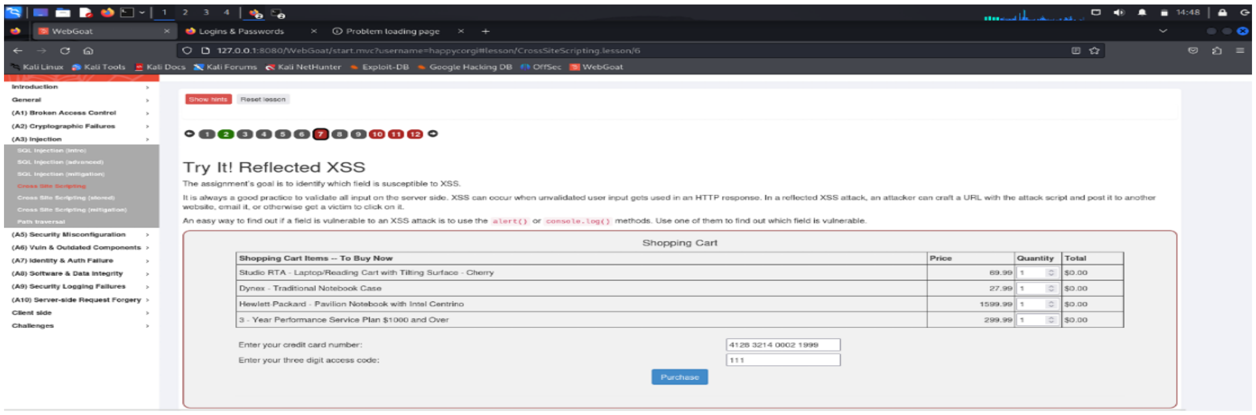

WebGoat
Logins & Passwords
Problem loading page
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
WebGoat
Introduction
General
(A1) Broken Access Control
(A2) Cryptographic Failures
(A3) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Cross Site Scripting
Cross Site Scripting (stored)
Cross Site Scripting (mitigation)
Path traversal
(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
Reset lesson
1 2 3 4 5 6 7 8 9 10 11 12
Attacker
1) Identify XSS flaw in application
2) Send URL with flaw to victim
3) Victim 'clicks' on URL
4) Attack sent to application on behalf of victim
5) Attack 'reflected' back to victim
6) Attack executes
7) Victim's browser sends information to attacker
Victim

```
127.0.0.1:8080/WebGoat/start.mvc?username=happycorgi#lesson/CrossSiteScripting.lesson/5
```

Reflected XSS scenario

- Attacker sends a malicious URL to the victim
- Victim clicks on the link that loads a malicious web page
- The malicious script embedded in the URL executes in the victim's browser
  - The script steals sensitive information, like the session id, and releases it to the attacker

Victim does not realize attack occurred



```
127.0.0.1:8080/WebGoat/start.mvc?username=happycorgi#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.

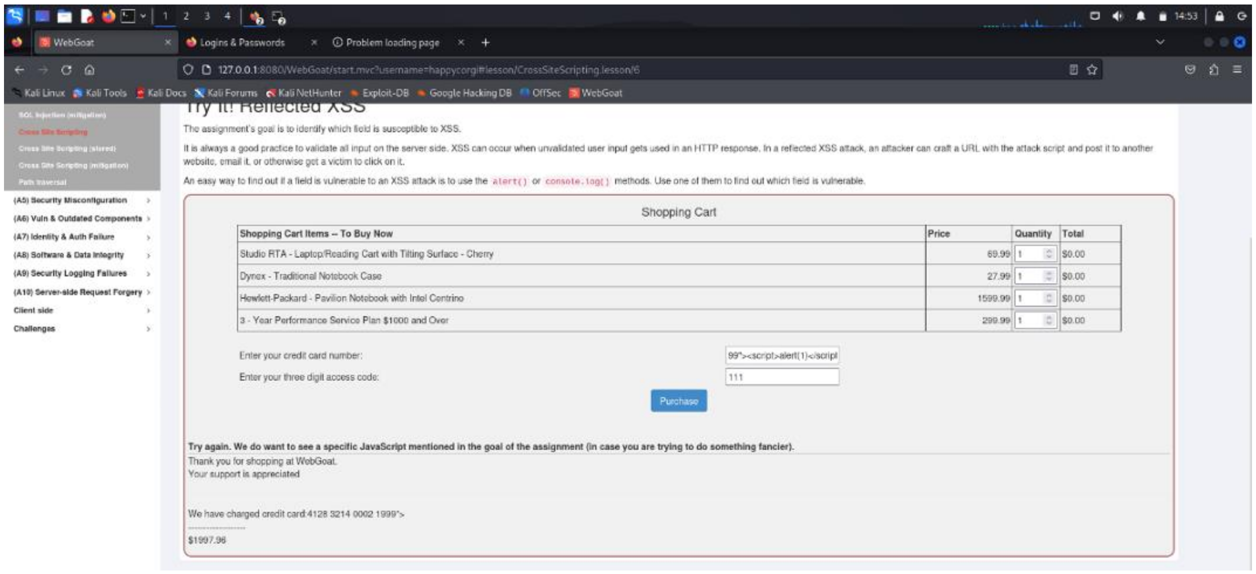
An easy way to find out if a field is vulnerable to an XSS attack is to use the `alert()` or `console.log()` methods. Use one of them to find out which field is vulnerable.

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

Enter your credit card number:	4128 3214 0002 1999
Enter your three digit access code:	111

After that we will able to see the purchase page ,showed below



127.0.0.1:8080/WebGoat/start.mvc?username=happycorgi#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

Enter your credit card number: 99"><script>alert(1)</script

Enter your three digit access code: 111

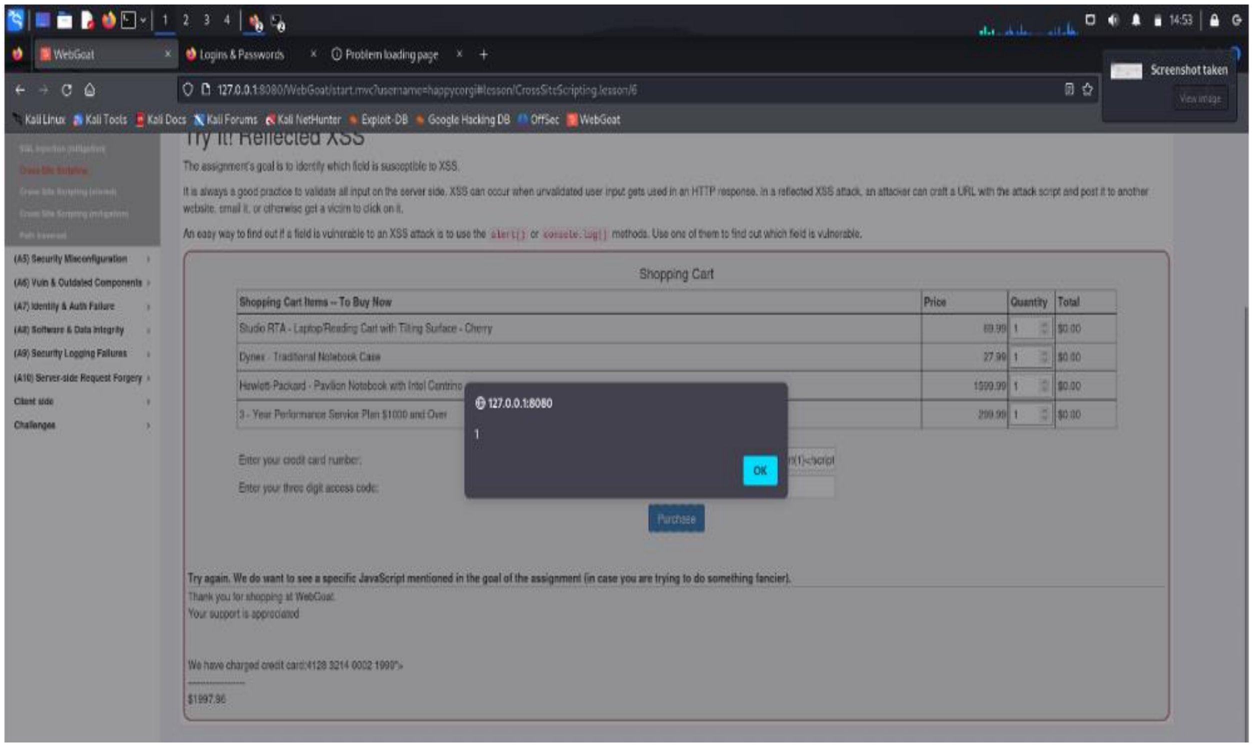
Try again. We do want to see a specific JavaScript mentioned in the goal of the assignment (in case you are trying to do something fancier).

Thank you for shopping at WebGoat.
Your support is appreciated

We have charged credit card:4128 3214 0002 1999">

$1997.96

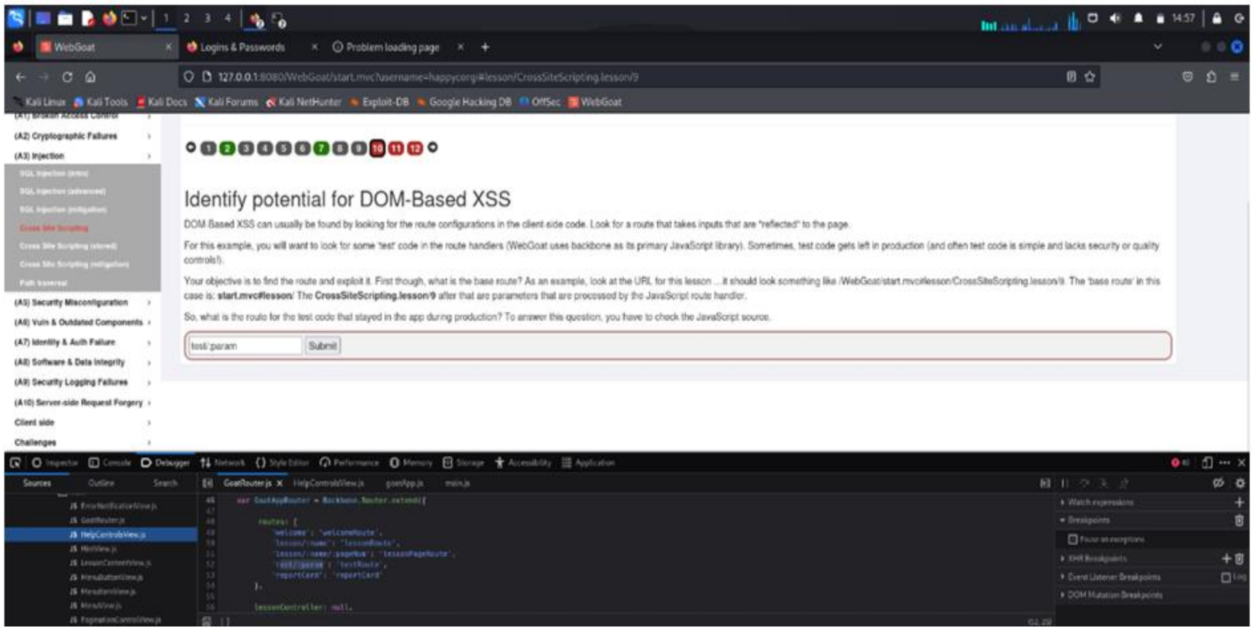
After entering card no and 3 digit code we will able to see some ip address like 198.32.32 like something



127.0.0.1:8080

1

After clicking okay go to web goat page



127.0.0.1:8080/WebGoat/start.mvc?username=happycorgi#lesson/CrossSiteScripting.lesson/9

Identify potential for DOM-Based XSS

DOM-Based XSS can usually be found by looking for the route configurations in the client side code. Look for a route that takes inputs that are "reflected" to the page.

For this example, you will want to look for some 'test' code in the route handlers (WebGoat uses backbone as its primary JavaScript library). Sometimes, test code gets left in production (and often test code is simple and lacks security or quality controls!).

Your objective is to find the route and exploit it. First though, what is the base route? As an example, look at the URL for this lesson ...it should look something like /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9. The 'base route' in this case is: **start.mvc#lesson/** The **CrossSiteScripting.lesson/9** after that are parameters that are processed by the JavaScript route handler.

So, what is the route for the test code that stayed in the app during production? To answer this question, you have to check the JavaScript source.

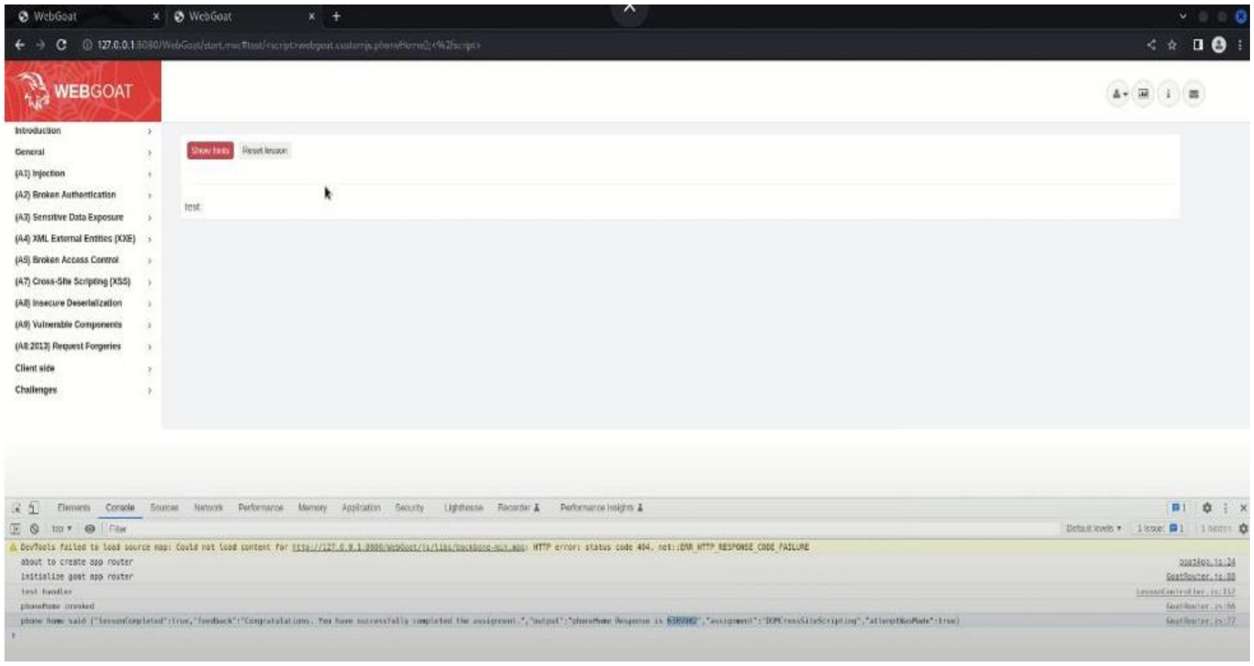
test/:param

```
var GoatAppRouter = Backbone.Router.extend({

    routes: {
        'welcome': 'welcomeRoute',
        'lesson/:name': 'lessonRoute',
        'lesson/:name/:pageNum': 'lessonPageRoute',
        'test/:param': 'testRoute',
        'reportCard': 'reportCard'
    },

    lessonController: null,
```

After clicking the button submit u will see the below page ,if u got any error ,fix it and u will see the output



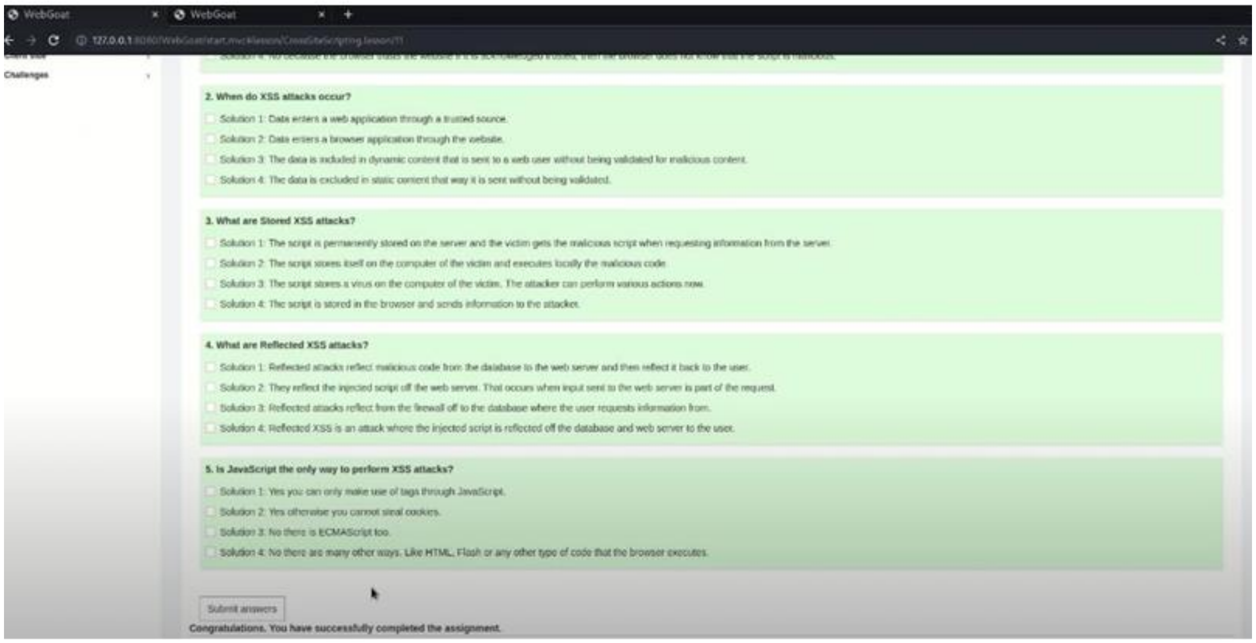
127.0.0.1:8080/WebGoat/start.mvc#test/<script>webgoat.customjs.phoneHome();<%2fscript>

test

```
⚠ DevTools failed to load source map: Could not load content for http://127.0.0.1:8080/WebGoat/js/libs/backbone-min.map: HTTP error: status code 404, net::ERR_HTTP_RESPONSE_CODE_FAILURE
about to create app router                                                                goatApp.js:24
initialize goat app router                                                             GoatRouter.js:88
test handler                                                                   LessonController.js:152
phoneHome invoked                                                                      GoatRouter.js:66
phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.","output":"phoneHome Response is [illegible]","assignment":"DOMCrossSiteScripting","attemptWasMade":true}   GoatRouter.js:77
```

WEBGOAT
Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Show hints
Reset lesson
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse

After that assignment session ,if you complete the answer ,you will be get as “congradulation, you have completed the assignment “


WebGoat
WebGoat
127.0.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/11
Challenges

2. When do XSS attacks occur?

- Solution 1: Data enters a web application through a trusted source.
- Solution 2: Data enters a browser application through the website.
- Solution 3: The data is included in dynamic content that is sent to a web user without being validated for malicious content.
- Solution 4: The data is excluded in static content that way it is sent without being validated.

3. What are Stored XSS attacks?

- Solution 1: The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.
- Solution 2: The script stores itself on the computer of the victim and executes locally the malicious code.
- Solution 3: The script stores a virus on the computer of the victim. The attacker can perform various actions now.
- Solution 4: The script is stored in the browser and sends information to the attacker.

4. What are Reflected XSS attacks?

- Solution 1: Reflected attacks reflect malicious code from the database to the web server and then reflect it back to the user.
- Solution 2: They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.
- Solution 3: Reflected attacks reflect from the firewall off to the database where the user requests information from.
- Solution 4: Reflected XSS is an attack where the injected script is reflected off the database and web server to the user.

5. Is JavaScript the only way to perform XSS attacks?

- Solution 1: Yes you can only make use of tags through JavaScript.
- Solution 2: Yes otherwise you cannot steal cookies.
- Solution 3: No there is ECMAScript too.
- Solution 4: No there are many other ways. Like HTML, Flash or any other type of code that the browser executes.

Submit answers

Congratulations. You have successfully completed the assignment.

The XSS vulnerability allowed the injection and execution of malicious scripts, demonstrating how attackers could exploit this flaw to compromise user sessions or execute other malicious actions.

Let see about mitigation /prevention

SQL Injection Prevention and Mitigation

1) Parameterized Queries/Prepared Statements
   - Use parameterized queries or prepared statements to prevent SQL Injection. This ensures that user inputs are treated as data, not executable code.
   - For example, in Java:

```
String query = "SELECT * FROM users WHERE username = ? AND password = ?";

PreparedStatement pstmt = connection.prepareStatement(query);

pstmt.setString(1, username);

pstmt.setString(2, password);
```
```
```

ORMs (Object-Relational Mappers)

- Use an ORM (like Hibernate for Java, Django ORM for Python) as it abstracts away SQL queries and reduces the likelihood of SQL injection vulnerabilities.

Input Validation and Escaping

- Validate all user inputs on the server side. For instance, only accept expected data formats (like digits for a user ID).
- Use escaping libraries (e.g., `htmlspecialchars` in PHP or `SqlParameter` in C#) when dynamic input is necessary.

Least Privilege for Database Accounts

- Ensure the database account has the minimum privileges necessary to perform its functions. This limits the damage in case of an injection attack.

Use Web Application Firewalls (WAFs)

- A WAF can help detect and block SQL Injection attempts. Configure it with specific rules for SQL injection patterns.

Cross-Site Scripting (XSS) Prevention and Mitigation

Input Sanitization and Encoding

- Sanitize all user inputs to remove or escape any potential HTML/JavaScript code. Use libraries like `ESAPI` for Java or `OWASP HTML Sanitizer`.
- Encode output based on the context (HTML, JavaScript, URL) to prevent untrusted data from being executed. For example, use HTML escaping for text in HTML content and JavaScript escaping for text in JavaScript.

Content Security Policy (CSP)

- Implement a strong CSP header in your application to restrict where JavaScript, CSS, and other resources can be loaded from. This can prevent many XSS attacks.

```http

Content-Security-Policy: default-src 'self';

```

HttpOnly and Secure Flags for Cookies

- Set the `HttpOnly` and `Secure` flags for cookies containing sensitive information, such as session tokens, to prevent access by client-side JavaScript.

Use of Security Libraries and Frameworks

- Use frameworks and libraries with built-in XSS protections. For instance, Angular and React have built-in sanitization for XSS vulnerabilities.

Regular Code Audits and Penetration Testing

- Conduct regular code audits and use automated tools to scan for XSS vulnerabilities. Additionally, WebGoat exercises can help test for potential issues and validate security measures.

Owasp zap report

Summary

- The OWASP ZAP scan identified multiple vulnerabilities within the OWASP JUICE SHOP application, with a particular focus on SQL Injection and Cross-Site Scripting.
- Both of these vulnerabilities present serious security risks if left unpatched in a real-world web application.
- SQL Injection could allow attackers to access sensitive database information or bypass authentication.
- XSS could lead to user session hijacking, data theft or defacement of website .

THE END