TASK 2: [INTRODUCTION TO WEB APPLICATION SECURITY]

1.Introduction:

In the contemporary internet era, web-based applications are an indispensable part of nearly all commerce activities. However, as their value increases, so it is essential to comprehend and counteract the threats that can undermine their effectiveness. In this blog post, we will explore the different types of web application flaws with the help of a simple, yet helpful example application. This introduction is hands-on and meant to prepare you with basic skills for practicing web security and identifying vulnerabilities. Using tools such as OWASP ZAP and WebGoat you will be able to see how the weaknesses are exploited with the intent of helping you improve your defense. Get ready this course is an eye opener on application security and security measures that will assist you in the care of your treasures.

2. Setting up WebGoat.

a. Objective:

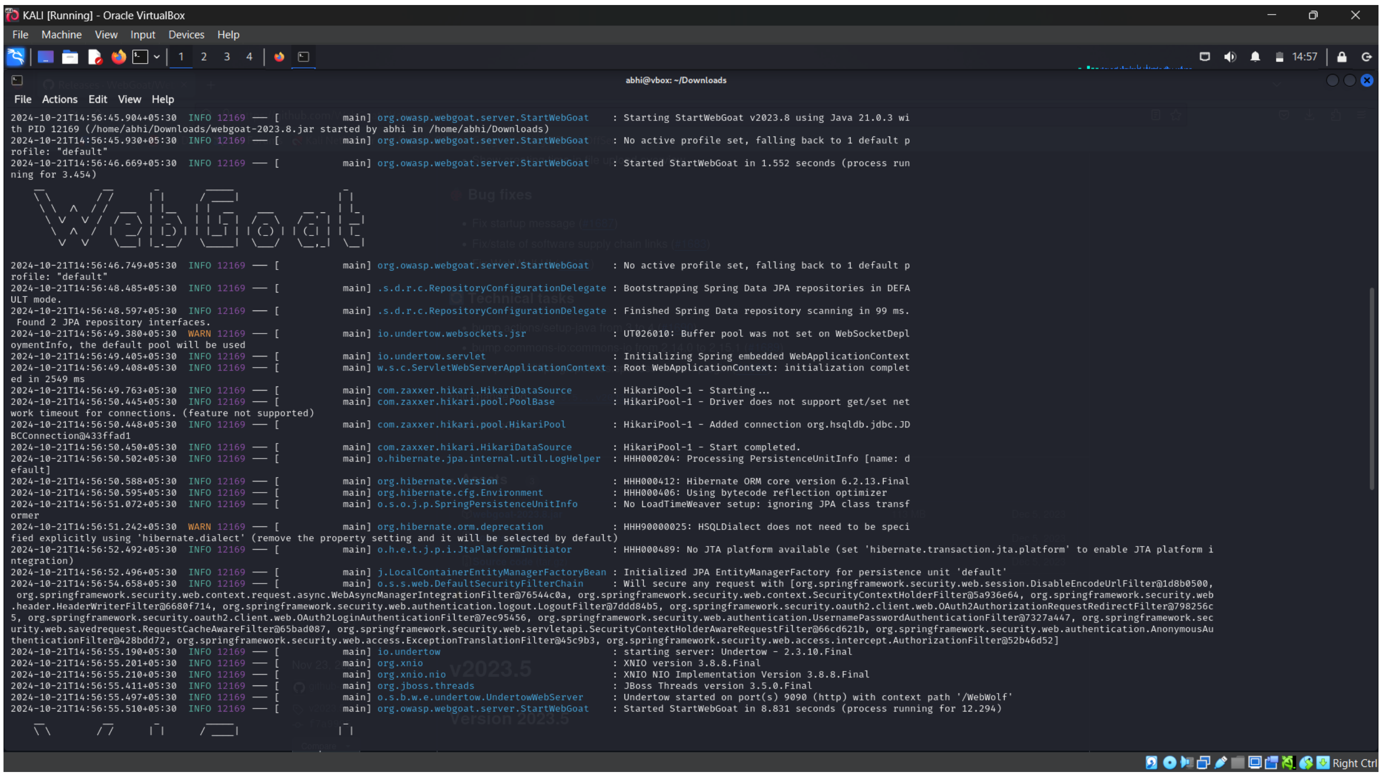
WebGoat was chosen as the primary learning tool because it is designed to teach common vulnerabilities in web applications. It is a deliberately vulnerable web application provided by OWASP, making it a perfect platform for practicing real-world attack scenarios in a controlled environment.

b. Steps for Installation:

1. Installed WebGoat on a virtual machine running [insert your OS].

2. Set up the virtual environment and downloaded WebGoat.

3. Configured the WebGoat server and accessed it via `localhost` in the browser.



```
2024-10-21T14:56:45.904+05:30  INFO 12169 --- [           main] org.owasp.webgoat.server.StartWebGoat    : Starting StartWebGoat v2023.8 using Java 21.0.3 wi
th PID 12169 (/home/abhi/Downloads/webgoat-2023.8.jar started by abhi in /home/abhi/Downloads)
2024-10-21T14:56:45.930+05:30  INFO 12169 --- [           main] org.owasp.webgoat.server.StartWebGoat    : No active profile set, falling back to 1 default p
rofile: "default"
2024-10-21T14:56:46.669+05:30  INFO 12169 --- [           main] org.owasp.webgoat.server.StartWebGoat    : Started StartWebGoat in 1.552 seconds (process run
ning for 3.454)

2024-10-21T14:56:46.749+05:30  INFO 12169 --- [           main] org.owasp.webgoat.server.StartWebGoat    : No active profile set, falling back to 1 default p
rofile: "default"
2024-10-21T14:56:48.485+05:30  INFO 12169 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFA
ULT mode.
2024-10-21T14:56:48.597+05:30  INFO 12169 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 99 ms.
 Found 2 JPA repository interfaces.
2024-10-21T14:56:49.380+05:30  WARN 12169 --- [           main] io.undertow.websockets.jsr               : UT026010: Buffer pool was not set on WebSocketDepl
oymentInfo, the default pool will be used
2024-10-21T14:56:49.405+05:30  INFO 12169 --- [           main] io.undertow.servlet                      : Initializing Spring embedded WebApplicationContext
2024-10-21T14:56:49.408+05:30  INFO 12169 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complet
ed in 2549 ms
2024-10-21T14:56:49.763+05:30  INFO 12169 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-10-21T14:56:50.445+05:30  INFO 12169 --- [           main] com.zaxxer.hikari.pool.PoolBase          : HikariPool-1 - Driver does not support get/set net
work timeout for connections. (feature not supported)
2024-10-21T14:56:50.448+05:30  INFO 12169 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection org.hsqldb.jdbc.JD
BCConnection@433ffad1
2024-10-21T14:56:50.450+05:30  INFO 12169 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-10-21T14:56:50.502+05:30  INFO 12169 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: d
efault]
2024-10-21T14:56:50.588+05:30  INFO 12169 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.2.13.Final
2024-10-21T14:56:50.595+05:30  INFO 12169 --- [           main] org.hibernate.cfg.Environment            : HHH000406: Using bytecode reflection optimizer
2024-10-21T14:56:51.072+05:30  INFO 12169 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transf
ormer
2024-10-21T14:56:51.242+05:30  WARN 12169 --- [           main] org.hibernate.orm.deprecation            : HHH90000025: HSQLDialect does not need to be speci
fied explicitly using 'hibernate.dialect' (remove the property setting and it will be selected by default)
2024-10-21T14:56:52.492+05:30  INFO 12169 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform i
ntegration)
2024-10-21T14:56:52.496+05:30  INFO 12169 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-10-21T14:56:54.658+05:30  INFO 12169 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.springframework.security.web.session.DisableEncodeUrlFilter@1d8b0500,
 org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@76544c0a, org.springframework.security.web.context.SecurityContextHolderFilter@5a936e64, org.springframework.security.web
.header.HeaderWriterFilter@6680f714, org.springframework.security.web.authentication.logout.LogoutFilter@7ddd84b5, org.springframework.security.oauth2.client.web.OAuth2AuthorizationRequestRedirectFilter@798256c
5, org.springframework.security.oauth2.client.web.OAuth2LoginAuthenticationFilter@7ec95456, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@7327a447, org.springframework.sec
urity.web.savedrequest.RequestCacheAwareFilter@65bad087, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@66cd621b, org.springframework.security.web.authentication.AnonymousAu
thenticationFilter@428bdd72, org.springframework.security.web.access.ExceptionTranslationFilter@45c9b3, org.springframework.security.web.access.intercept.AuthorizationFilter@52b46d52]
2024-10-21T14:56:55.190+05:30  INFO 12169 --- [           main] io.undertow                              : starting server: Undertow - 2.3.10.Final
2024-10-21T14:56:55.201+05:30  INFO 12169 --- [           main] org.xnio                                 : XNIO version 3.8.8.Final
2024-10-21T14:56:55.210+05:30  INFO 12169 --- [           main] org.xnio.nio                             : XNIO NIO Implementation Version 3.8.8.Final
2024-10-21T14:56:55.411+05:30  INFO 12169 --- [           main] org.jboss.threads                        : JBoss Threads version 3.5.0.Final
2024-10-21T14:56:55.497+05:30  INFO 12169 --- [           main] o.s.b.w.e.undertow.UndertowWebServer     : Undertow started on port(s) 9090 (http) with context path '/WebWolf'
2024-10-21T14:56:55.510+05:30  INFO 12169 --- [           main] org.owasp.webgoat.server.StartWebGoat    : Started StartWebGoat in 8.831 seconds (process running for 12.294)
```

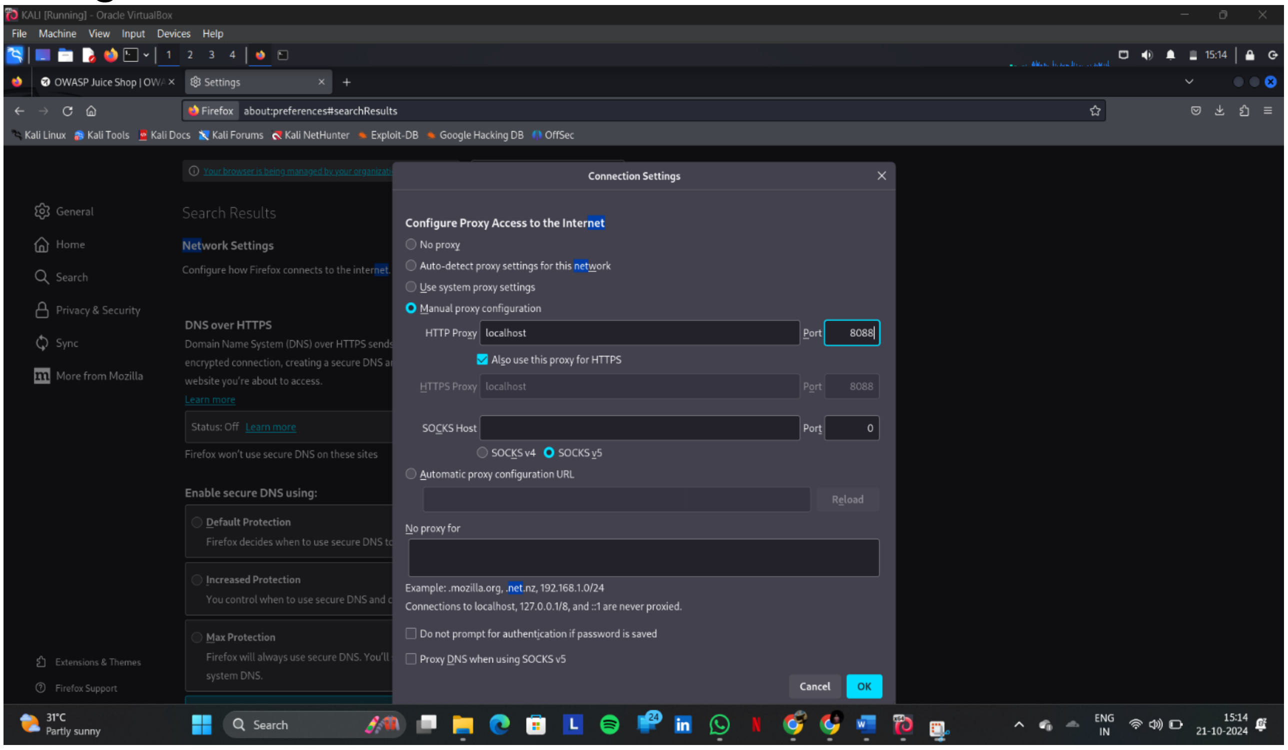
The WebGoat application was successfully installed and accessible through a browser, allowing me to begin vulnerability testing and analysis.

3. OWASP ZAP Vulnerability Scanning.

a. Objective 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 opensource tool widely used for finding security vulnerabilities in web applications.

b. Scanning Process 1. Loaded OWASP ZAP and pointed it to the OWASP JUICE SHOP web application URL. 2. Performed a full scan, allowing OWASP ZAP to detect vulnerabilities. 3. Focused on vulnerabilities like SQL Injection and Cross-Site Scripting, both common and dangerous in modern web applications.

➢ **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: 8088

[✓] Also use this proxy for HTTPS

HTTPS Proxy: localhost Port: 8088

SOCKS Host: Port: 0

SOCKS v4 [✓ SOCKS v5]

Automatic proxy configuration URL

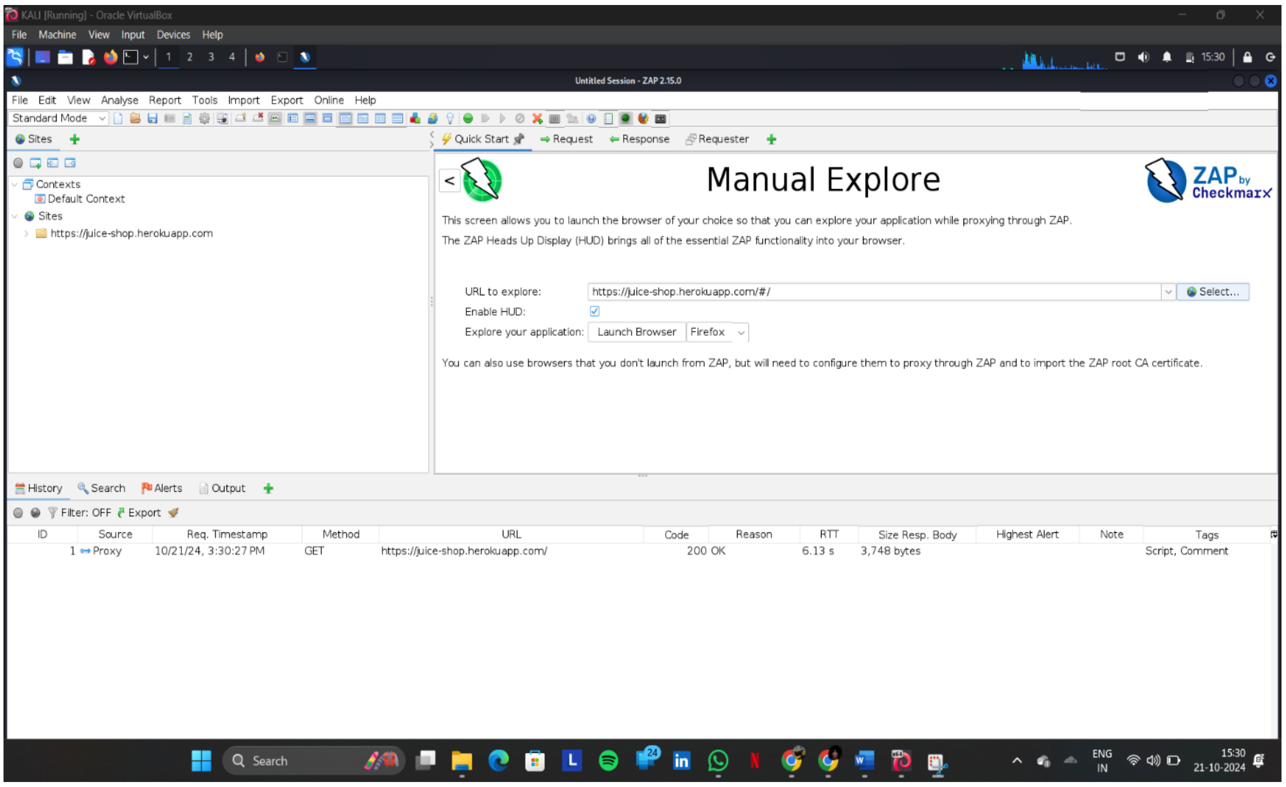
No proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24
Connections to localhost, 127.0.0.1/8, and ::1 are never proxied.

[ ] Do not prompt for authentication if password is saved

[ ] Proxy DNS when using SOCKS v5

➢ **Step 2: Opening Zap and pasting Owasp Juice shop URL.**



Untitled Session - ZAP 2.15.0

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.

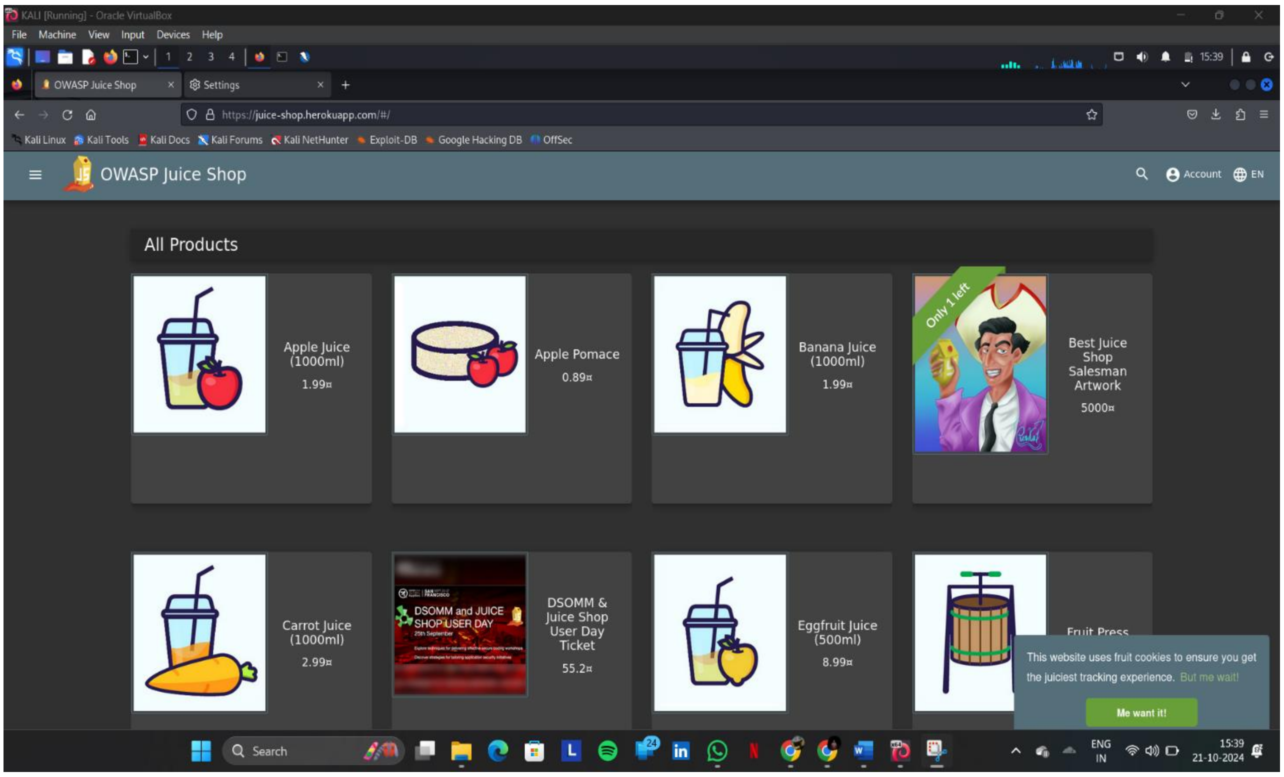
URL to explore: https://juice-shop.herokuapp.com/#/
Enable HUD: [✓]
Explore your application: Launch Browser Firefox

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.

Sites: Contexts / Default Context / Sites / https://juice-shop.herokuapp.com

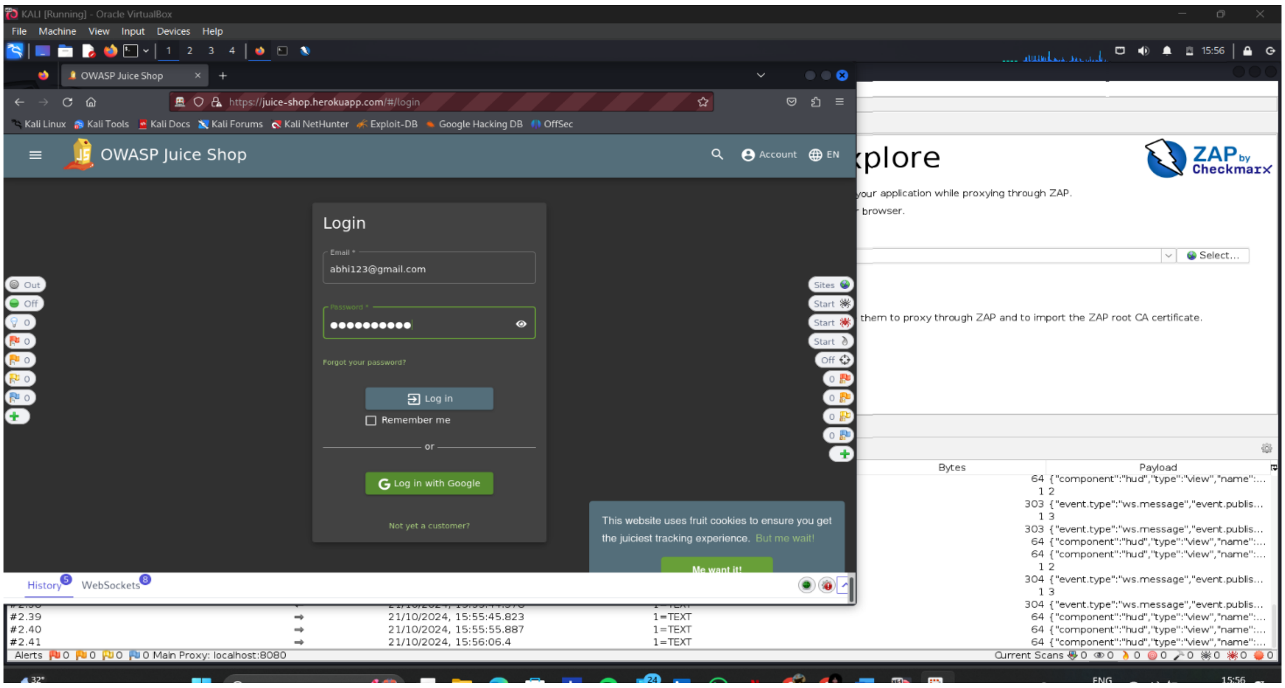
ID	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert	Note	Tags
1	Proxy	10/21/24, 3:30:27 PM	GET	https://juice-shop.herokuapp.com/	200	OK	6.13 s	3,748 bytes			Script, Comment

KALI [Running] - Oracle VirtualBox
File Edit View Analyse Report Tools Import Export Online Help
Standard Mode
Quick Start
Request
Response
Requester
Select...
History Search Alerts Output
Filter: OFF Export
15:30 21-10-2024


KALI [Running] - Oracle VirtualBox
File Machine View Input Devices Help
OWASP Juice Shop
Settings
https://juice-shop.herokuapp.com/#/
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Account EN
All Products
Apple Juice (1000ml) 1.99¤
Apple Pomace 0.89¤
Banana Juice (1000ml) 1.99¤
Only 1 left
Best Juice Shop Salesman Artwork 5000¤
Carrot Juice (1000ml) 2.99¤
DSOMM and JUICE SHOP USER DAY
DSOMM & Juice Shop User Day Ticket 55.2¤
Eggfruit Juice (500ml) 8.99¤
Fruit Press
Me want it!
Search
ENG IN
15:39 21-10-2024

This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!

➢ **Step 3: Logging in and starting spidering the website.**


KALI [Running] - Oracle VirtualBox
File Machine View Input Devices Help
OWASP Juice Shop
https://juice-shop.herokuapp.com/#/login
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Account EN
Login
Forgot your password?
Log in
Remember me
or
Log in with Google
Not yet a customer?
Me want it!
History WebSockets
xplore
ZAP by Checkmarx
Select...
Bytes Payload
Alerts Main Proxy: localhost:8080
Current Scans

Email *: abhi123@gmail.com

Password *: ●●●●●●●●●●●

This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!

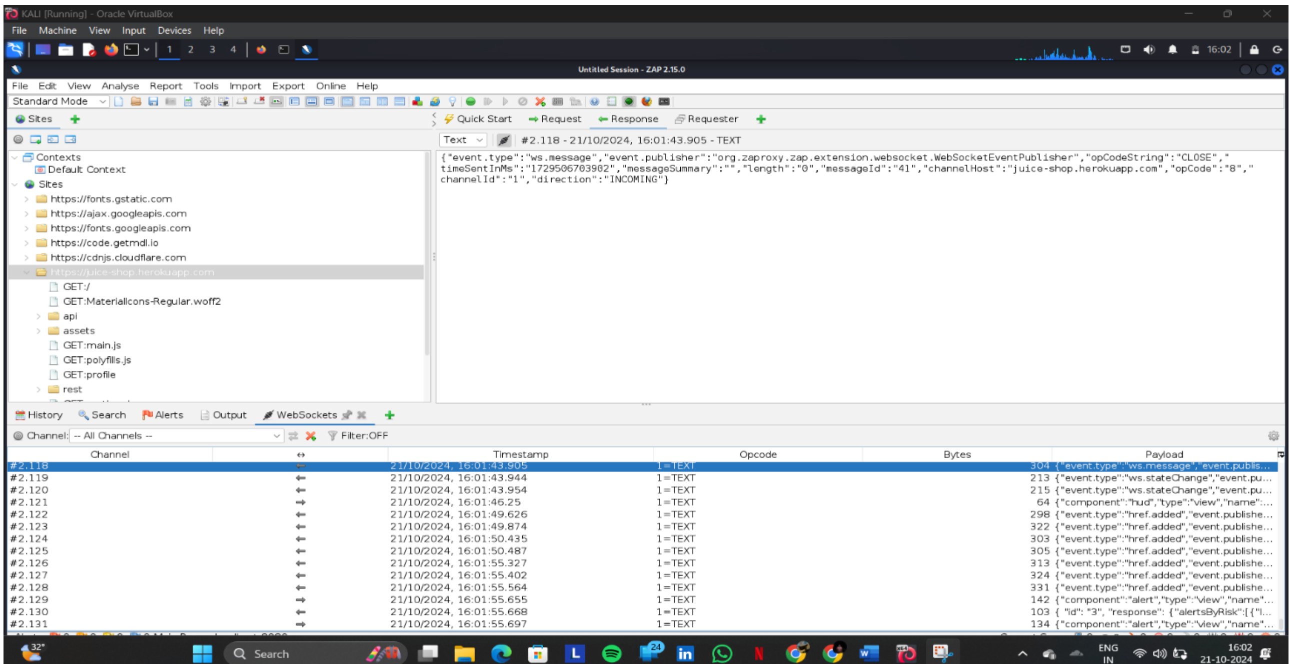
your application while proxying through ZAP.
browser.
them to proxy through ZAP and to import the ZAP root CA certificate.

```
#2.39   →   21/10/2024, 15:55:45.823   1=TEXT   64 {"component":"hud","type":"view","name":...
#2.40   →   21/10/2024, 15:55:55.887   1=TEXT   64 {"component":"hud","type":"view","name":...
#2.41   →   21/10/2024, 15:56:06.4     1=TEXT   64 {"component":"hud","type":"view","name":...
```

login with your credentials

Username :- abhi123@gmail.com.

Password :-a.b.c@2005.



Untitled Session - ZAP 2.15.0

Sites
- Contexts
  - Default Context
- Sites
  - https://fonts.gstatic.com
  - https://ajax.googleapis.com
  - https://fonts.googleapis.com
  - https://code.getmdl.io
  - https://cdnjs.cloudflare.com
  - https://juice-shop.herokuapp.com
    - GET:/
    - GET:MaterialIcons-Regular.woff2
    - api
    - assets
    - GET:main.js
    - GET:polyfills.js
    - GET:profile
    - rest

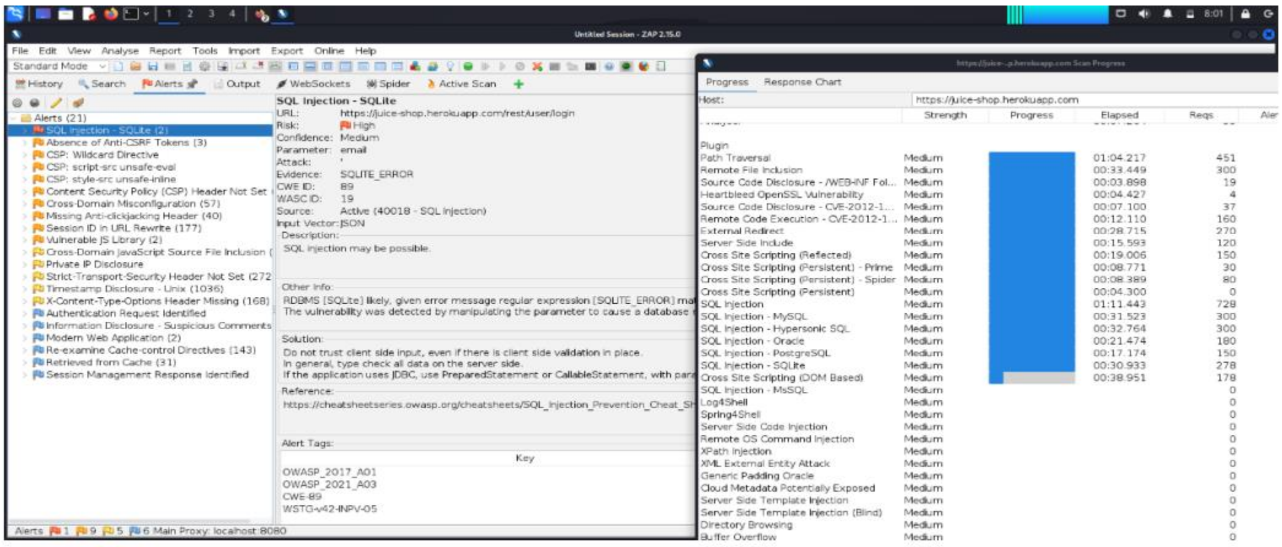
#2.118 - 21/10/2024, 16:01:43.905 - TEXT

```
{"event.type":"ws.message","event.publisher":"org.zaproxy.zap.extension.websocket.WebSocketEventPublisher","opCodeString":"CLOSE","
timeSentInMs":"1729506703902","messageSummary":"","length":"0","messageId":"41","channelHost":"juice-shop.herokuapp.com","opCode":"8","
channelId":"1","direction":"INCOMING"}
```

Channel: -- All Channels -- Filter:OFF

Channel	↔	Timestamp	Opcode	Bytes	Payload
#2.118	←	21/10/2024, 16:01:43.905	1=TEXT	304	{"event.type":"ws.message","event.publis...
#2.119	←	21/10/2024, 16:01:43.944	1=TEXT	213	{"event.type":"ws.stateChange","event.pu...
#2.120	←	21/10/2024, 16:01:43.954	1=TEXT	215	{"event.type":"ws.stateChange","event.pu...
#2.121	→	21/10/2024, 16:01:46.25	1=TEXT	64	{"component":"hud","type":"view","name":...
#2.122	←	21/10/2024, 16:01:49.626	1=TEXT	298	{"event.type":"href.added","event.publishe...
#2.123	←	21/10/2024, 16:01:49.874	1=TEXT	322	{"event.type":"href.added","event.publishe...
#2.124	←	21/10/2024, 16:01:50.435	1=TEXT	303	{"event.type":"href.added","event.publishe...
#2.125	←	21/10/2024, 16:01:50.487	1=TEXT	305	{"event.type":"href.added","event.publishe...
#2.126	←	21/10/2024, 16:01:55.327	1=TEXT	313	{"event.type":"href.added","event.publishe...
#2.127	←	21/10/2024, 16:01:55.402	1=TEXT	324	{"event.type":"href.added","event.publishe...
#2.128	←	21/10/2024, 16:01:55.564	1=TEXT	331	{"event.type":"href.added","event.publishe...
#2.129	→	21/10/2024, 16:01:55.655	1=TEXT	142	{"component":"alert","type":"view","name"...
#2.130	←	21/10/2024, 16:01:55.668	1=TEXT	103	{ "id": "3", "response": {"alertsByRisk":[{"l...
#2.131	→	21/10/2024, 16:01:55.697	1=TEXT	134	{"component":"alert","type":"view","name"...

16:02 21-10-2024



Untitled Session - ZAP 2.15.0

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

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 par

Reference:
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sh

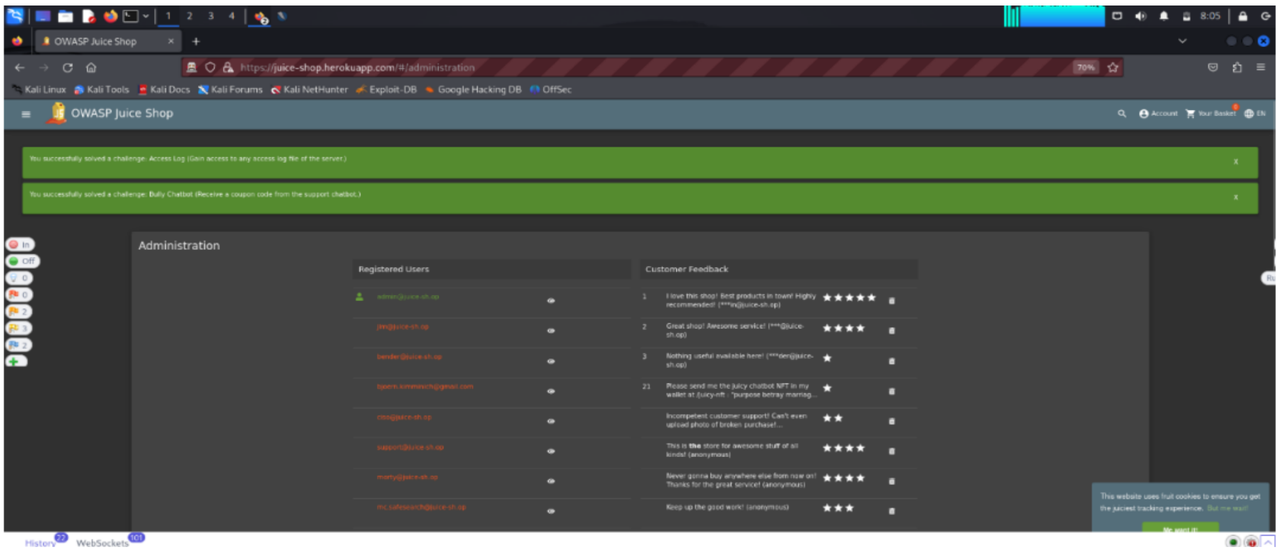
Alert Tags:

Key
OWASP_2017_A01
OWASP_2021_A03
CWE-89
WSTG-v42-INPV-05

Alerts 1 9 5 6 Main Proxy: localhost:8080

Host: https://juice-shop.herokuapp.com

Plugin	Strength	Progress	Elapsed	Reqs
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



OWASP Juice Shop — https://juice-shop.herokuapp.com/#/administration

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)	★★★

c. Vulnerabilities Identified :

-SQl Injection(SQLi): A technique where attackers manipulate SQL queries to gain unauthorized access to database information.

-Cross-Site Scripting(XSS): A vulnerability that allows attackers to inject malicious scripts into web pages, potentially leading to session hijacking or defacement.

4. Cross-Site Scripting (XSS) Vulnerability Analysis.

a. Objective :

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.

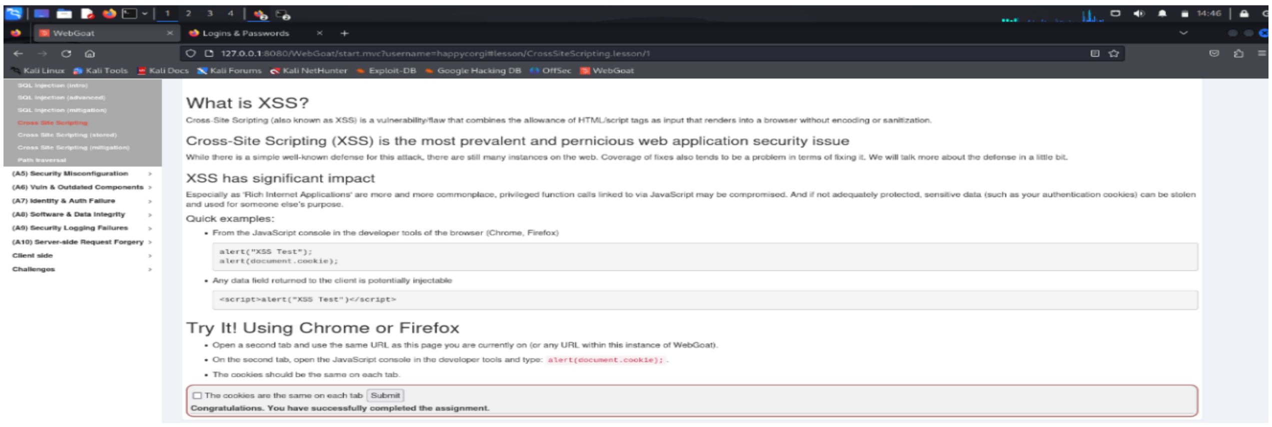
b. Exploitation Attempt :

I attempted to inject a simple script such as:

html

“><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
127.0.0.1:8080/WebGoat/start.mvc?username=happycorgi#lesson/CrossSiteScripting.lesson/1
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
WebGoat
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
14:46

What is XSS?

Cross-Site Scripting (also known as XSS) is a vulnerability/flaw that combines the allowance of HTML/script tags as input that renders into a browser without encoding or sanitization.

Cross-Site Scripting (XSS) is the most prevalent and pernicious web application security issue

While there is a simple well-known defense for this attack, there are still many instances on the web. Coverage of fixes also tends to be a problem in terms of fixing it. We will talk more about the defense in a little bit.

XSS has significant impact

Especially as 'Rich Internet Applications' are more and more commonplace, privileged function calls linked to via JavaScript may be compromised. And if not adequately protected, sensitive data (such as your authentication cookies) can be stolen and used for someone else's purpose.

Quick examples:

- From the JavaScript console in the developer tools of the browser (Chrome, Firefox)

```
alert("XSS Test");
alert(document.cookie);
```

- Any data field returned to the client is potentially injectable

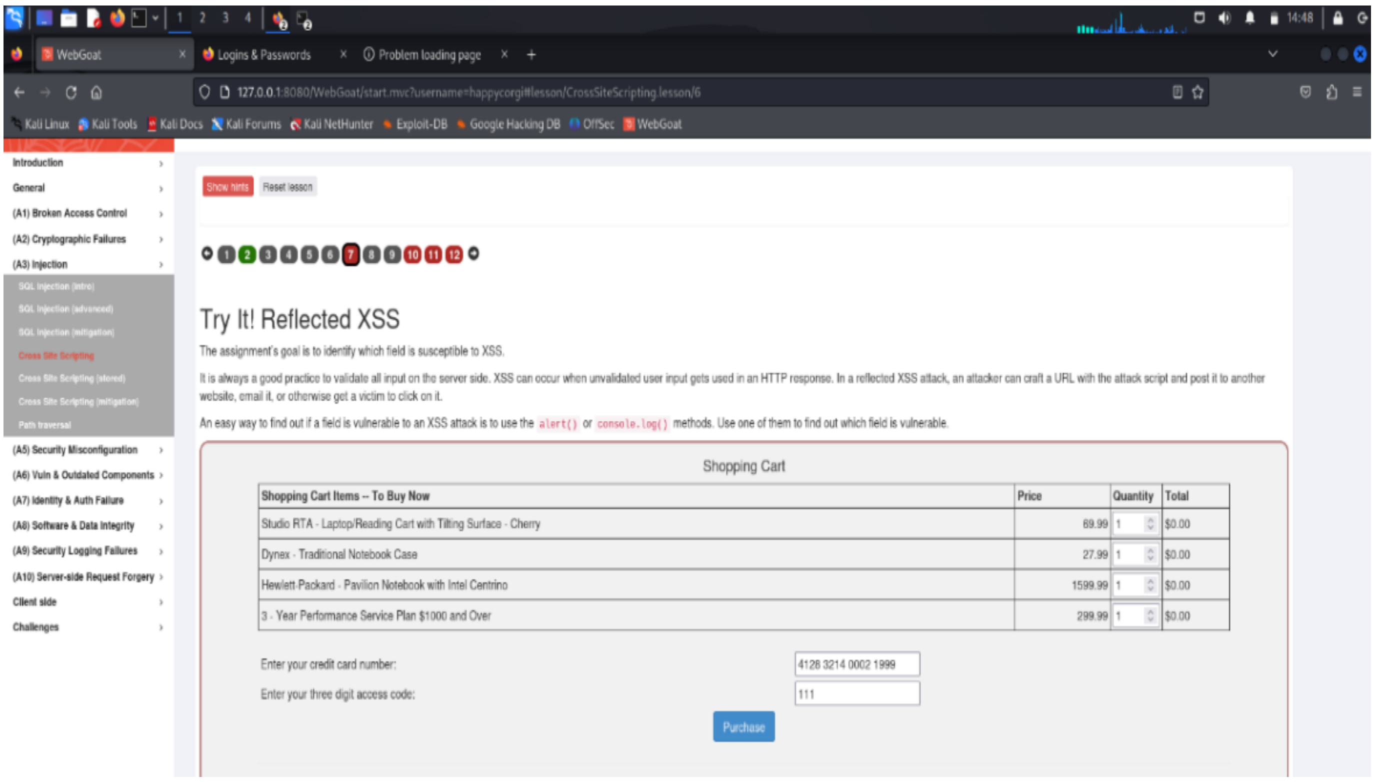
```
<script>alert("XSS Test")</script>
```

Try It! Using Chrome or Firefox

- Open a second tab and use the same URL as this page you are currently on (or any URL within this instance of WebGoat).
- On the second tab, open the JavaScript console in the developer tools and type: `alert(document.cookie);` .
- The cookies should be the same on each tab.

☐ The cookies are the same on each tab Submit

Congratulations. You have successfully completed the assignment.



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.

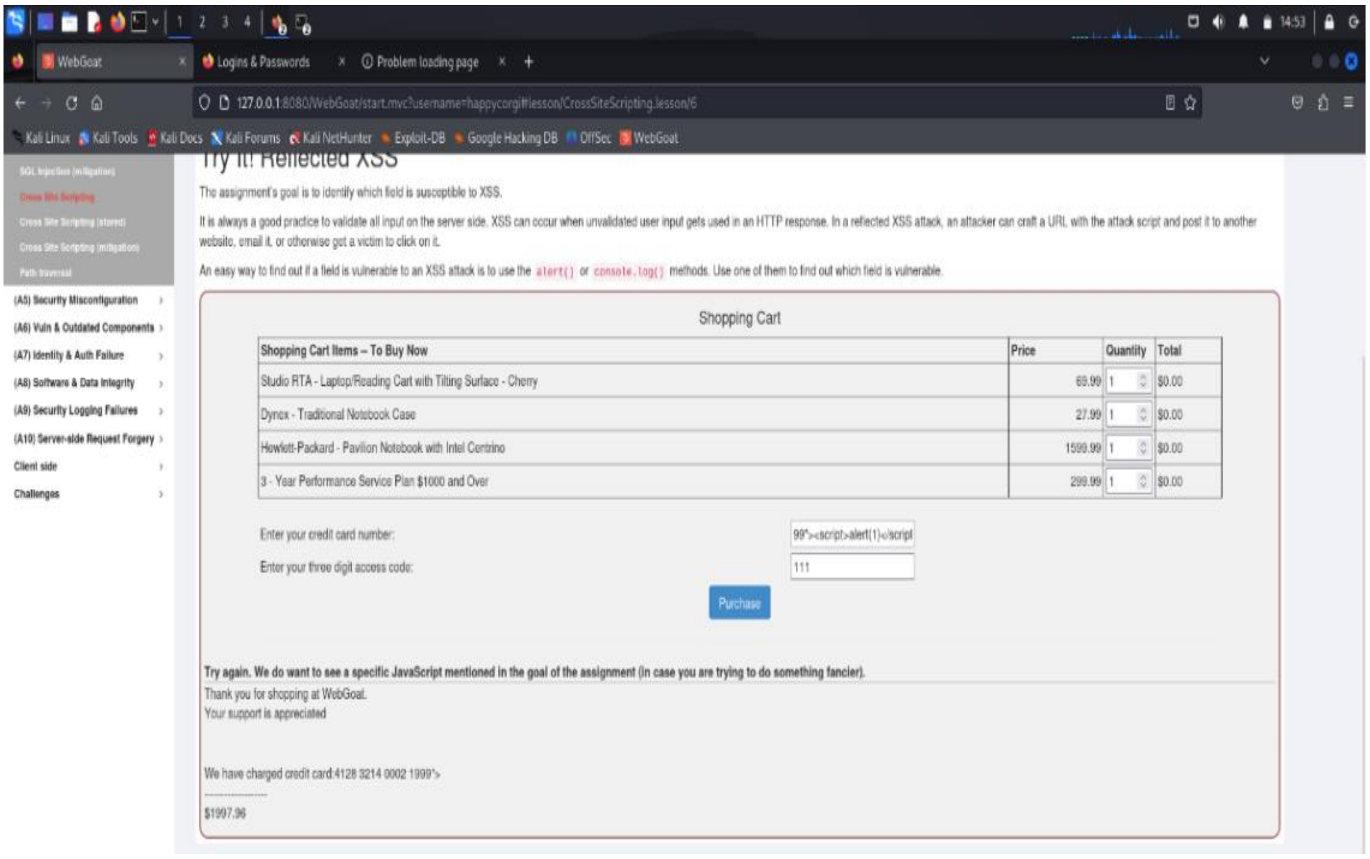
It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input gets used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on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



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

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

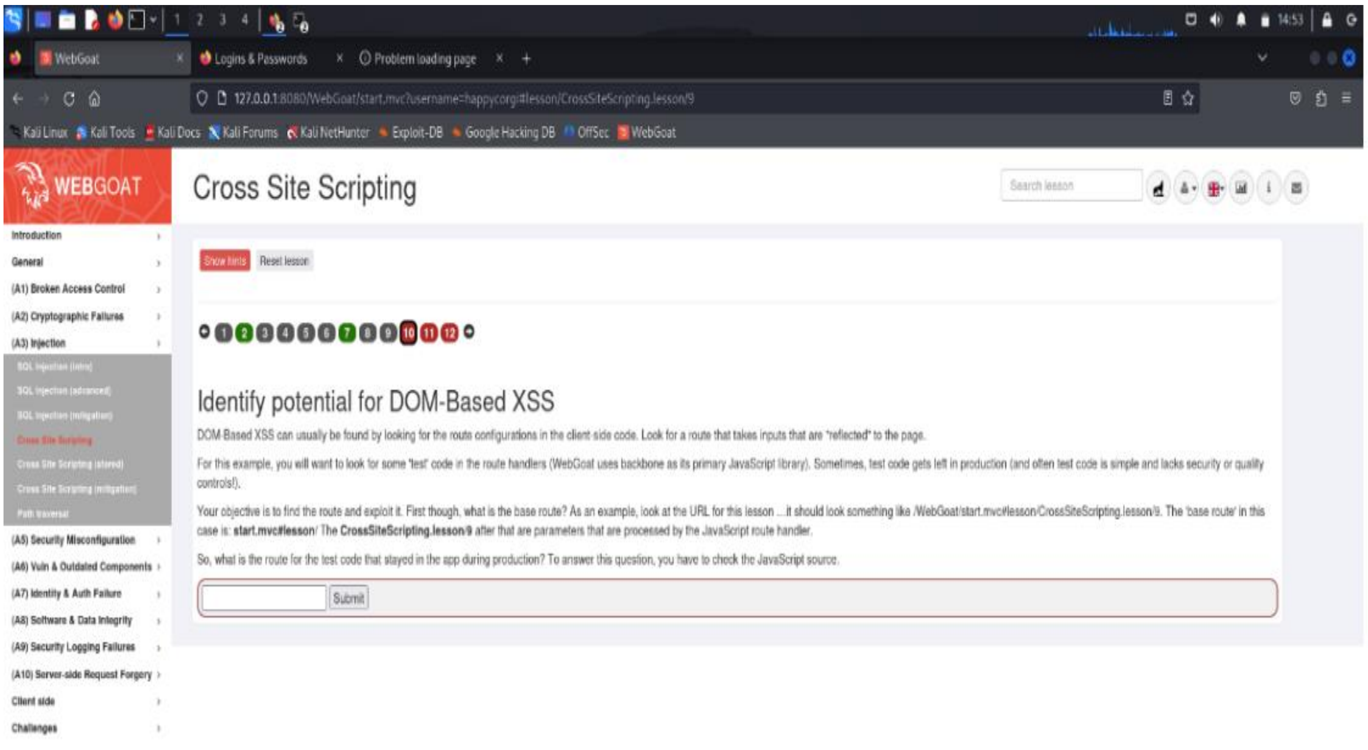
Enter your credit card number:	99"><script>alert(1)</script
Enter your three digit access code:	111

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


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

Cross Site Scripting

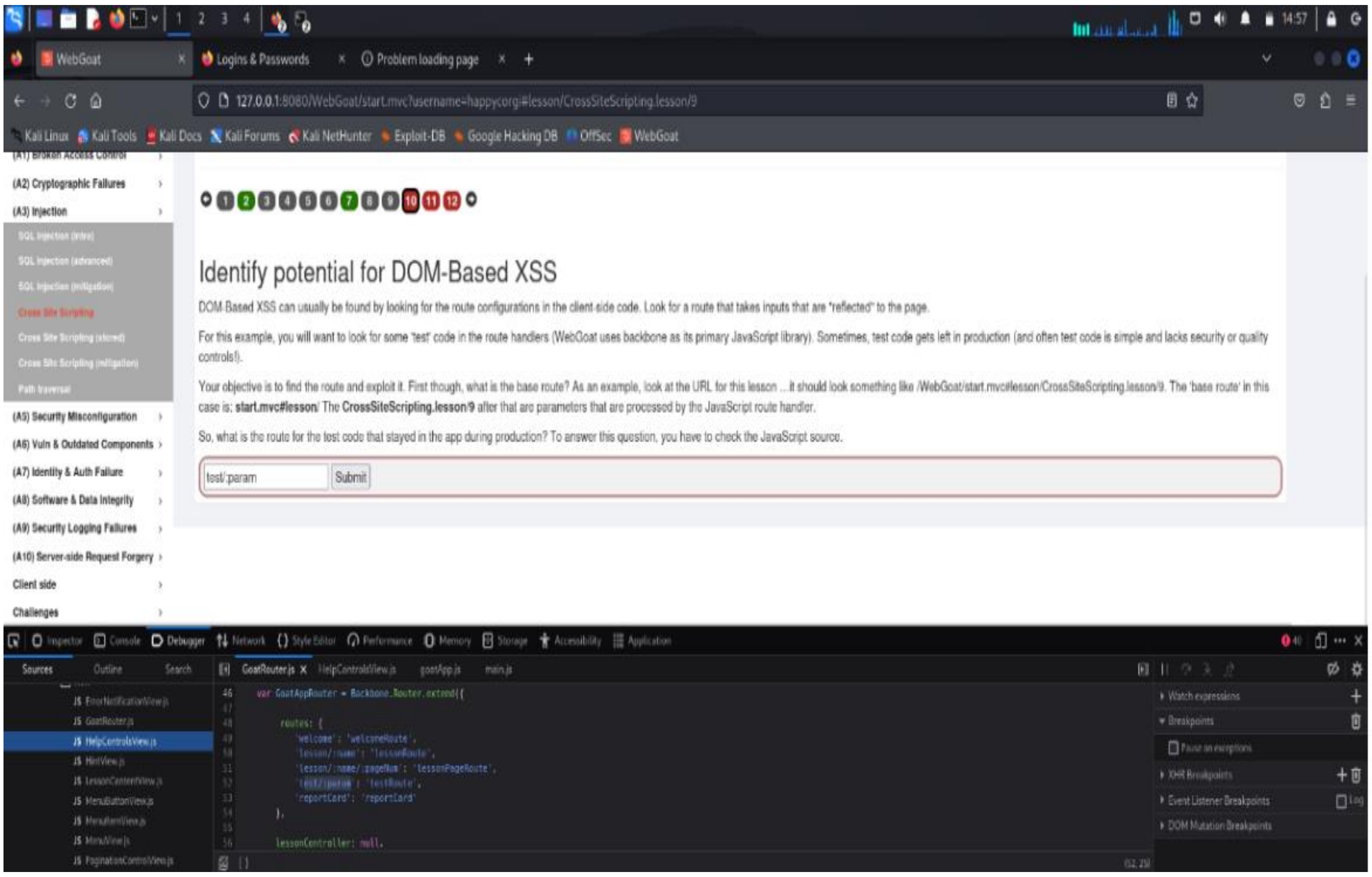
Identify potential for DOM-Based XSS

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.


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.

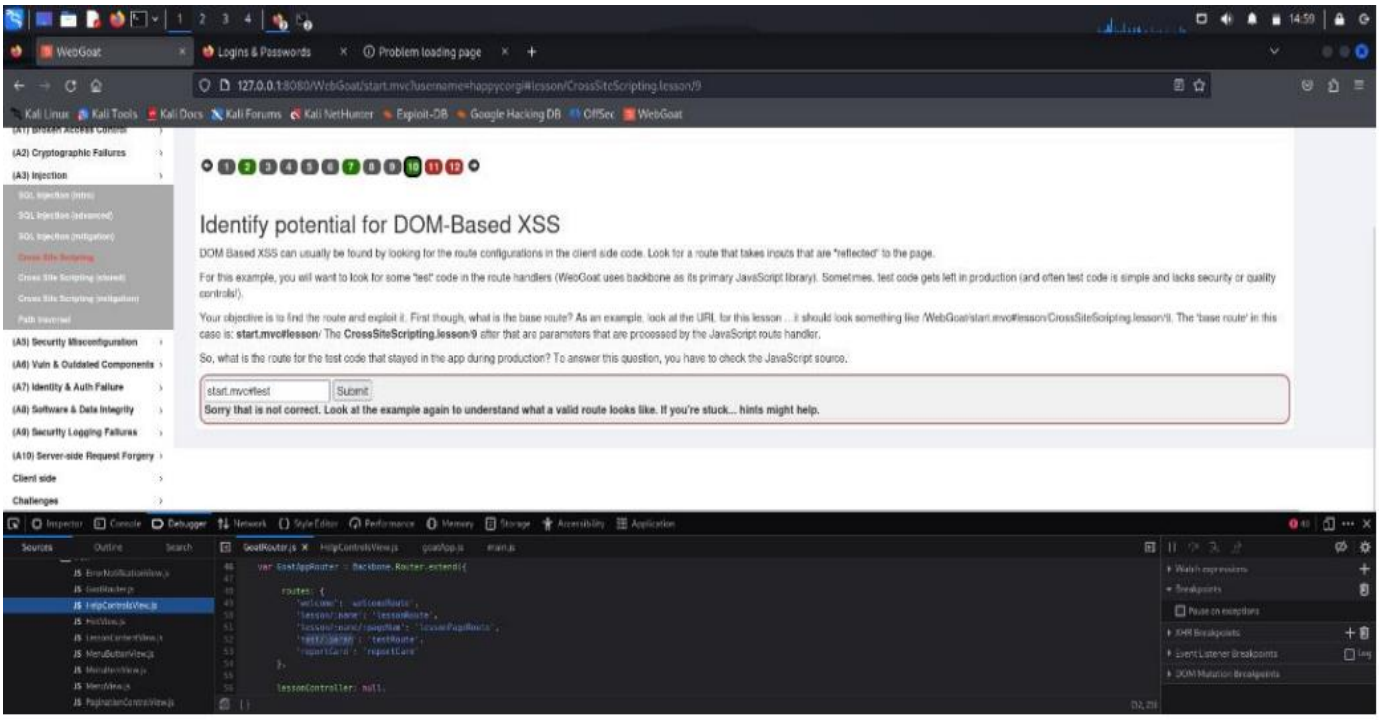
test/:param

GoatRouter.js

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

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

    lessonController: null,
```


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.

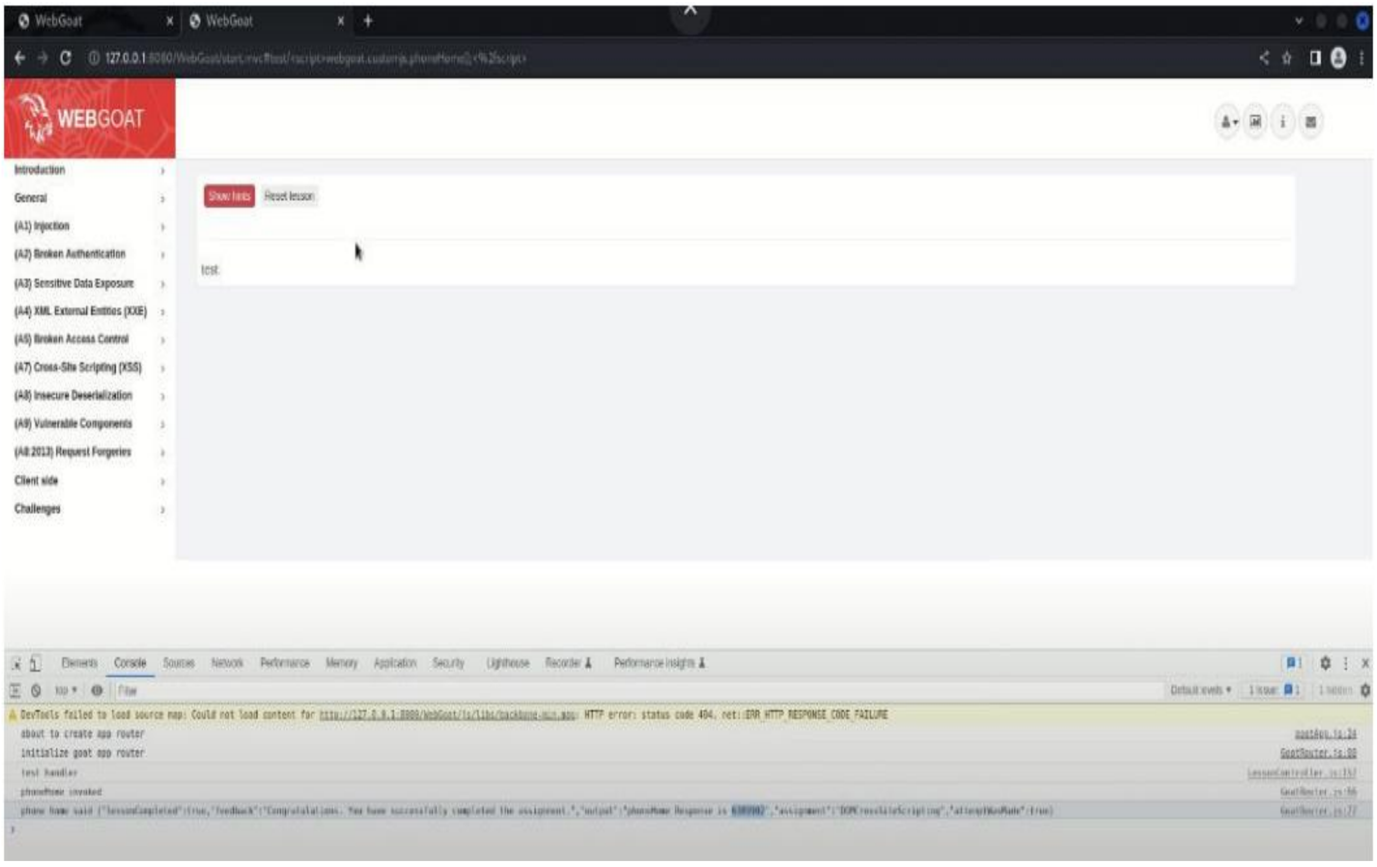
start.mvc#test Submit

Sorry that is not correct. Look at the example again to understand what a valid route looks like. If you're stuck... hints might help.

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

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

    lessonController: null,
```


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

Show hints   Reset lesson

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:34
initialize goat app router                                                            GoatRouter.js:80
test handler                                                                          LessonController.js:157
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
```

c. Result :

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.

d. Mitigation for Cross-Site Scripting (XSS)

To mitigate XSS:

-Output Encoding: Ensure that all user input is properly encoded before displaying it in the browser to prevent script execution.

- Content Security Policy (CSP): Implement CSP headers that restrict which scripts can run on a web page.

- Input Validation: Validate and sanitize user input to reject dangerous characters and scripts.

- Escape User Input: Ensure that any data inserted into HTML, JavaScript, or SQL queries is properly escaped to prevent unintended execution.

5. OWASP ZAP Scan Report.

Summary :Report

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 the website.

6. Reflection and Learning.

This project came with a practical component that dealt with web application vulnerabilities and their life cycle starting from identification, exploitation, and mitigation. I appreciated the need to secure the input fields and sanitize the user data so as to shield web applications from common threats such as SQL Injection, XSS and others.

Undertaking this internship impressed upon me a lot of knowledge concerning vulnerability scanning and web application security in general and in particular the same provided me with the much needed hands on experience that is very useful to me going forward in my career in cyberspace security.

7. Conclusion:

I can say that the contribution I have made in the course of the internship is of great importance as it has increased my knowledge about the various types of web application vulnerabilities and the process of overcoming them. OWASP ZAP and Web Goat were especially useful when it came to the practical aspect of dealing with the vulnerabilities that exist out there.

It is very important to protect such applications from threats that are aimed at exploiting vulnerabilities such as SQL Injection or Cross-Site Scripting as this affects how secure a web application is maintained.