**Resolución del reto “Coquette”**

TheGrue - CTF Team

/home/jhonnthedickers/hacking/writeups/this.docs

20/02/2024

# **Nick de los autores***:* Dreysanox, Znati.

Localidad: Madrid, Madrid (España)

Temas: CTF, Hack0n, MISC

Contacto: jhonnthedickers@gmail.com

# **Índice**

[**Índice 2**](#_jndyhq1cnps4)

[**Enunciado del Reto 2**](#_4d34og8)

[**Resolución 3**](#_2s8eyo1)

[Entendiendo el árbol de Merkle: 3](#_26in1rg)

[Idea de resolución: 3](#_e8wtsnpoaso5)

[Desarrollando el script: 4](#_p5gx7jna1h0)

[Función de concatenación: 4](#_lnxbz9)

[Función de hasheado: 5](#_pl4r0ugtgnwt)

[Función calcular hash: 5](#_5gs5e088i7r)

[Implementación Final: 6](#_c1y8aek2uzun)

[**Flag: 7**](#_iei8aqfmk384)

# 

# 

# 

# 

# **Enunciado del Reto**

?????

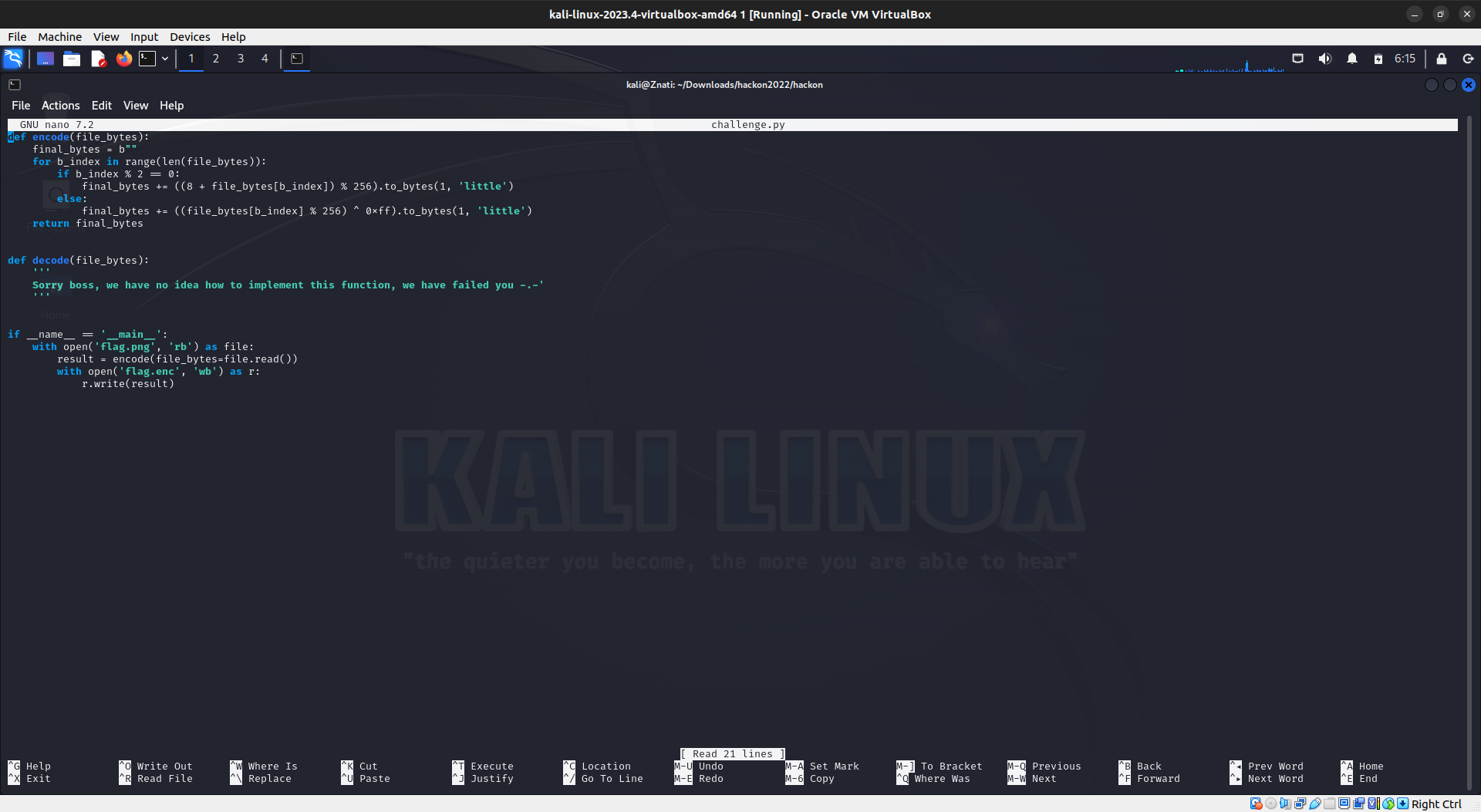
**Mensaje**: “Sorry boss, we have no idea how to implement this function, we have failed you -.-”

**Ficheros:** Flag.enc y Challenge.py

*Palabras clave:* Crypto, Árbol de Merkle, Hash, Sha-256, Python.

**Challenge.py:**

| def encode(file\_bytes):  final\_bytes = b""  for b\_index in range(len(file\_bytes)):  if b\_index % 2 == 0:  final\_bytes += ((8 + file\_bytes[b\_index]) % 256).to\_bytes(1, 'little')  else:  final\_bytes += ((file\_bytes[b\_index] % 256) ^ 0xff).to\_bytes(1, 'little')  return final\_bytes  def decode(file\_bytes):  '''  Sorry boss, we have no idea how to implement this function, we have failed you -.-'  '''  if \_\_name\_\_ == '\_\_main\_\_':  with open('flag.png', 'rb') as file:  result = encode(file\_bytes=file.read())  with open('flag.enc', 'wb') as r:  r.write(result) |
| --- |

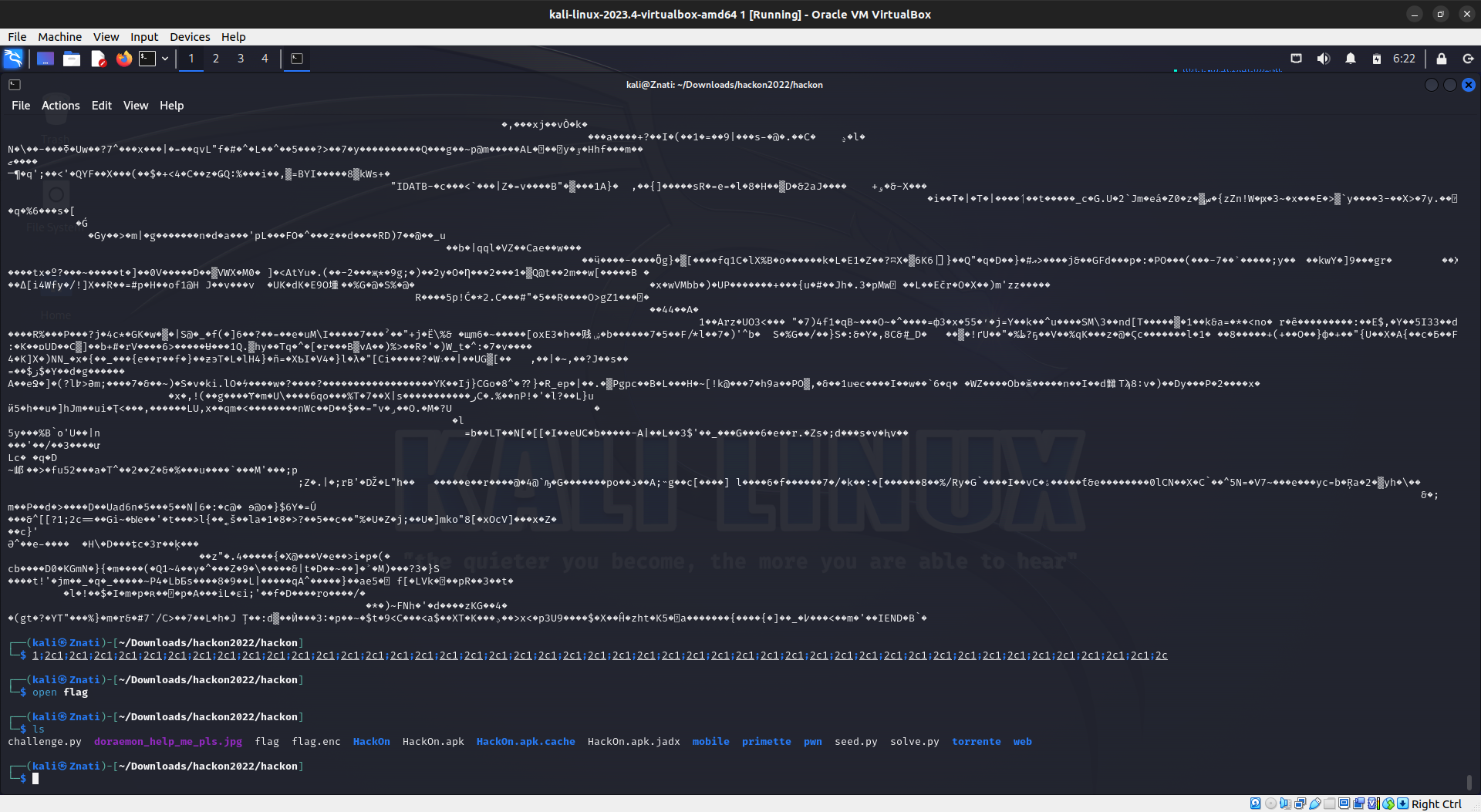


# **Resolución**

## 

## **Entendiendo el código:**

En el main vemos que se abre un fichero “*flag.png*” y le aplica la funcion “*encode()*” leyendolo byte a byte, su resultado será “*flag.enc*”, fichero el cual nos dan en el reto y al abrirlo no sacamos nada en claro.



Vemos que en la funcion *“decode()*” pone “Sorry boss, we have no idea how to implement this function, we have failed you”, asi que lo mas seguro es que se tenga que implementar esta funcion para obtener la imagen inicial.

“*encode()*” en primer lugar se crea una variable donde irá almacenando los bytes finales, a los bytes pares les suma 8 y el resultado es modulo 256, a los bytes impares solo les hace un XOR con 0xff módulo 256.

## **Idea de resolución:**

Entonces para implementar la funcion “*decode()*” necesitamos dos partes, una para los bytes pares y otra para los bytes impares. La parte de los impares es facil, tenemos que tener en cuenta que los valores que nos llegan nunca superarán el valor de 256, por lo que el módulo es redundante. el byte mas grande que nos puede llegar es “*FF*” (1111 1111 en binario) su valor en decimal es 255, por otro lado para revertir un XOR basta con saber sus propiedades, si A ⊕ B = C, entonces C ⊕ B = A. Por lo que el código se quedaría igual.

En los pares siguiendo la misma lógica, no nos puede llegar un byte superior a 256, pero como le estamos sumando 8 a todos los que llegan (255 + 8), podríamos tener alguna modificación con el módulo, asi que lo tendremos en cuenta.

## **Desarrollando el script:**

Entonces empezamos con una variable para guardar los bytes decodeados,

final\_bytes = b””

Ahora segun nos vayan llegando los bytes, seleccionaremos los pares e impares, en los pares nos vale con restar el 8 y hacer el modulo otra vez, es importante poner el modulo, ya que sino nos pueden quedar valores negativos y nos daria error. En los pares como dijimos antes no hay ningun cambio

def decode(file\_bytes):

final\_bytes = b””

for b\_index in range(len(file\_bytes):

half\_array.append(array[i] + array[i + 1])

return half\_array

### ***Función de concatenación:***

Esta función coge posiciones del array de dos en dos, las concatena y las sustituye en orden en el array original, por lo que nuestro array resultante es la mitad de grande:

def concatenar\_strings(array):

half\_array = []

for i in range(0, len(array), 2):

half\_array.append(array[i] + array[i + 1])

return half\_array

### 

### ***Función de hasheado:***

Para realizar esta función primero desarrolle una función que me sustituya cada posición del array por el resultado de pasarle el contenido de esa posición a otra función (en este caso una función que hashe):

def sustituir con hash(array):

hash\_array = [calcular\_hash(string) for string in array]

return hash\_array

### ***Función calcular hash:***

Primero tenemos que averiguar qué tipo de función hash han utilizado en su árbol de Merkle, que conseguimos metiendo el hash en el identificador de hashes de deCode.fr, que nos dice que el hash de tipo Sha-256.

Para calcular el hash de un string utilizaremos la librería hashlib. Implementamos la siguiente función como viene especificado en la documentación de la librería:

from hashlib import \*

def calcular\_hash(data):

sha256 = hashlib.sha256()

sha256.update(data)

return sha256.hexdigest()

### 

### 

### **Implementación Final:**

Ahora solo nos queda implementar el bucle que recorre la lista, nos calcula la raíz con cada palabra y compara con la raíz proporcionada para ver si la palabra es correcta.

Primero declaramos el array de wordlist, donde metemos todas las posibles contraseñas de rockyou.

wordlist = []

with open("/usr/share/wordlists/rockyou.txt", "r", encoding='latin1') as file:

for line in file:

wordlist.append(line.strip())

Utilizando este bucle podemos meter cada línea de la lista en cada posición del array, dejando un string compuesto de los carácteres ascii de la password.

Luego declaramos el siguiente bucle:

for password in wordlist:

temp\_mensaje = []

temp\_mensaje = mensaje.copy()

temp\_mensaje = temp\_mensaje.append(password)

temp\_mensajemensaje = sustituir\_con\_hash(mensaje)

while len(mensaje) > 1:

temp\_mensajemensaje = concatenar\_strings(temp\_mensajemensaje)

print(mensaje)

temp\_mensajemensaje = sustituir\_con\_hash(temp\_mensaje)

print(mensaje)

if temp\_mensaje[0] == targetHash:

print(password)

break

Copiamos el mensaje original en un array temporal y le añadimos la palabra de wordlist como último nodo. Calculamos sus hashes para dejar los nodos preparados para el resto del proceso.

Por último solo nos queda iterar entre concatenar el contenido de los nodos hijos, y calcular el hash del resultado, hasta que solo quede un nodo, el nodo raíz. Conseguimos esto con el bucle while

Antes de pasar a la siguiente palabra de la wordlist, comprobamos si el hash raíz es el correcto, en cuyo caso printeamos la palabra que hemos utilizado para calcularlo.

### **Flag:**HackOn{thereisnopassword}