**Resolución del reto “Crypto Tree”**

TheGrue - CTF Team

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

20/02/2024

# **Nick de los autores***:* Dreysanox, Astharot15.

Localidad: Madrid, Madrid (España)

Temas: CTF, Hack0n, Crypto

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**

Mi tío me quería pasar su contraseña por correo pero el mensaje se ha corrompido y justo hemos perdido la parte donde venía la contraseña. Aún así me pasó la raíz de Merkle del mensaje para asegurarse de que el mensaje no había sido modificado en el trayecto.

Su contraseña ya se filtró en una lista muy conocida hace mucho tiempo.

¿Puedes ayudarme a recuperar la contraseña?

**Mensaje**: “The password that I use is the same as in the Google account it is ???”

**Raíz de Merkle:** "30c085686aa4b1d76ac1c72dfefab6f4a02f5e3865acd76f868b6d5781d2efc8"

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

# **Resolución**

## 

## **Entendiendo el árbol de Merkle:**

Un árbol de Merkle (en inglés: Merkle hash tree) es una estructura de datos en árbol en el que cada nodo viene identificado por un valor hash. El tipo de hasheado que se utilice es irrelevante siempre y cuando se mantenga en todo el proceso de generación del árbol.

El valor hash de cada nodo se calcula con la concatenación de los hashes hijos, normalmente siendo el árbol de tipo binario, pero no siempre. El nodo raíz nos proporciona el hash raíz de merkle, que es el hash que se nos proporciona en este reto.

Los primeros nodos son calculados hasheado los artículos originales de los que queramos hacer el aŕbol de Merkle.

Como todo nodo depende del anterior la única forma de conseguir la raíz de Merkle es conociendo todos los nodos originales. Esto es lo que usaremos para resolver el reto.

## **Idea de resolución:**

Como en el enunciado nos dicen que la contraseña se encuentra en una lista antigua muy famosa, podemos deducir que la contraseña se encuentra en la lista rockyou.txt.

Para conseguir averiguar cual de todas las posibles contraseñas es la que buscamos, escribiremos un script que cicle por toda la lista calculando la respectiva raíz de Merkle del mensaje con la password de la lista, y que compare el resultado con la raíz que nos proporcionan.

## **Desarrollando el script:**

Después de entender cómo funciona el árbol de Merkle nos damos cuenta qué necesitamos dos funciones; una que nos concatena el valor de dos nodos, y otra que calcule el hash de un nodo. Ciclando estas funciones tantas veces como sea necesario, conseguiremos la raíz de Merkle de los nodos qué le introducimos.

Yo he elegido meter cada palabra como posición de un array, por lo que en todas las funciones trabajaré tratando las posiciones del array como un nodo:

mensaje = ["The","password","that","I","use","is","the","same","as","in","the","Google","account","it","is"]

### ***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}