



udp UNIVERSIDAD
DIEGO PORTALES

UNIVERSIDAD DIEGO PORTALES

ESCUELA DE INFORMÁTICA & TELECOMUNICACIONES

Tarea3: Cifrado en producción

Autor:

Marcos Fantóval Castro

marcos.fantoval@mail.udp.cl

19.962.344-3

Profesor: Nicolás Boettcher Ayudante: Francisco Lara

26 de Mayo de 2021

Índice

1. ¿Qué tipo de Hash se utiliza en cada archivo?	2
2. ¿A que atribuiría la diferencia de tiempo al crackear cada archivo?	4
3. Nuevo algoritmo Hash	5
4. Nuevo algoritmo Hash	5
5. Explicación funcionamiento de los códigos	6
5.1. Código del server	6
5.2. Código del cliente	8
6. Link del GitHub	11

1. ¿Qué tipo de Hash se utiliza en cada archivo?

En primer lugar, utilizando HashCat y la librería HashID de Python, se utilizan los comandos para poder encontrar los datos respectivos y se crea un código el cual abre los archivos y los analiza con la librería arrojando el resultado del algoritmo hash utilizado en cada uno, los cuales son:

- .-Archivo I: MD5
- .-Archivo II: MD5 (\$pass.\$salt)
- .-Archivo III: MD5 (\$pass.\$salt)
- .-Archivo IV: NTLM
- .-Archivo V: SHA-512Crypt

```

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: MD5
Hash.Target.....: C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\archivo_1.txt
Time.Started.....: Sun Jun 20 20:16:31 2021, (4 secs)
Time.Estimated...: Sun Jun 20 20:16:35 2021, (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\diccionario_2.dict)
Guess.Queue.....: 2/2 (100.00%)
Speed.#1.....: 15365 H/s (9.65ms) @ Accel:128 Loops:1 Thr:8 Vec:1
Speed.#3.....: 75392 H/s (4.16ms) @ Accel:1024 Loops:1 Thr:64 Vec:1
Speed.#*.....: 90757 H/s
Recovered.....: 1000/1000 (100.00%) Digests
Progress.....: 316096/316096 (100.00%)
Rejected.....: 0/316096 (0.00%)
Restore.Point...: 235307/316096 (74.44%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Restore.Sub.#3...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1...: prog123 -> Sexy420
Candidates.#3...: sexy45 -> zzzzzzzzzz
Hardware.Mon.#1..: N/A
Hardware.Mon.#3..: Temp: 48c Fan: 0% Util: 4% Core: 891MHz Mem:1000MHz Bus:4
Started: Sun Jun 20 20:16:10 2021
Stopped: Sun Jun 20 20:16:37 2021

```

Figura 1: HashCat del Archivo1

```

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: md5($pass.$salt)
Hash.Target.....: C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\archivo_2.txt
Time.Started.....: Sun Jun 20 23:22:15 2021, (3 secs)
Time.Estimated...: Sun Jun 20 23:22:18 2021, (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\diccionario_2.dict)
Guess.Queue.....: 2/2 (100.00%)
Speed.#1.....: 14423 H/s (7.98ms) @ Accel:64 Loops:1 Thr:8 Vec:1
Speed.#3.....: 104.8 kH/s (6.92ms) @ Accel:512 Loops:1 Thr:64 Vec:1
Speed.#*.....: 119.2 kH/s
Recovered.....: 1000/1000 (100.00%) Digests
Progress.....: 316096/316096 (100.00%)
Rejected.....: 0/316096 (0.00%)
Restore.Point...: 292651/316096 (92.58%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Restore.Sub.#3...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1...: tXNjh989 -> willdwp
Candidates.#3...: willdwal -> zzzzzzzzzz
Hardware.Mon.#1..: N/A
Hardware.Mon.#3..: Temp: 40c Fan: 0% Util: 0% Core: 891MHz Mem:1000MHz Bus:4
Started: Sun Jun 20 23:22:00 2021
Stopped: Sun Jun 20 23:22:20 2021

```

Figura 2: HashCat del Archivo2

```

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: md5($pass.$salt)
Hash.Target.....: C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\archivo_3.txt
Time.Started.....: Sun Jun 20 23:24:57 2021, (18 secs)
Time.Estimated.....: Sun Jun 20 23:25:15 2021, (0 secs)
Kernel.Feature.....: Pure Kernel
Guess.Base.....: File (C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\diccionario_2.dict)
Guess.Queue.....: 2/2 (100.00%)
Speed.#1.....: 1170.7 kH/s (7.48ms) @ Accel:64 Loops:1 Thr:8 Vec:1
Speed.#3.....: 10471.4 kH/s (5.10ms) @ Accel:512 Loops:1 Thr:64 Vec:1
Speed.#*.....: 11642.0 kH/s
Recovered.....: 1000/1000 (100.00%) Digests, 1000/1000 (100.00%) Salts
Progress.....: 316073686/316096000 (99.99%)
Rejected.....: 0/316073686 (0.00%)
Restore.Point.....: 233472/316096 (73.86%)
Restore.Sub.#1.....: Salt:997 Amplifier:0-1 Iteration:0-1
Restore.Sub.#3.....: Salt:999 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1.....: willdwa1 -> zzzzzzzzzz
Candidates.#3.....: Pooty77 -> willdup
Hardware.Mon.#1.....: N/A
Hardware.Mon.#3.....: Temp: 46c Fan: 0% Util: 0% Core: 891MHz Mem:1000MHz Bus:4

Started: Sun Jun 20 23:24:51 2021
Stopped: Sun Jun 20 23:25:16 2021
    
```

Figura 3: HashCat del Archivo3

```

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: NTLM
Hash.Target.....: C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\archivo_4.txt
Time.Started.....: Sun Jun 20 23:26:50 2021, (2 secs)
Time.Estimated.....: Sun Jun 20 23:26:52 2021, (0 secs)
Kernel.Feature.....: Pure Kernel
Guess.Base.....: File (C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\diccionario_2.dict)
Guess.Queue.....: 2/2 (100.00%)
Speed.#1.....: 28278 H/s (7.12ms) @ Accel:128 Loops:1 Thr:8 Vec:1
Speed.#3.....: 158.4 kH/s (5.26ms) @ Accel:512 Loops:1 Thr:64 Vec:1
Speed.#*.....: 186.7 kH/s
Recovered.....: 1000/1000 (100.00%) Digests
Progress.....: 316096/316096 (100.00%)
Rejected.....: 0/316096 (0.00%)
Restore.Point.....: 245760/316096 (77.75%)
Restore.Sub.#1.....: Salt:0 Amplifier:0-1 Iteration:0-1
Restore.Sub.#3.....: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1.....: Xhris -> zzzzzzzzzz
Candidates.#3.....: richguy -> xHQNXSQ3
Hardware.Mon.#1.....: N/A
Hardware.Mon.#3.....: Temp: 43c Fan: 0% Util: 4% Core: 891MHz Mem:1000MHz Bus:4

Started: Sun Jun 20 23:26:37 2021
Stopped: Sun Jun 20 23:26:54 2021
    
```

Figura 4: HashCat del Archivo4

```

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: sha512crypt $6$, SHA512 (Unix)
Hash.Target.....: C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\archivo_5.txt
Time.Started.....: Sun Jun 20 23:33:48 2021, (44 mins, 17 secs)
Time.Estimated.....: Mon Jun 21 00:18:05 2021, (0 secs)
Kernel.Feature.....: Pure Kernel
Guess.Base.....: File (C:\Users\Marcos F\Desktop\cripto\Tarea4\Archivos\diccionario_2.dict)
Guess.Queue.....: 2/2 (100.00%)
Speed.#1.....: 879 H/s (10.99ms) @ Accel:32 Loops:8 Thr:8 Vec:1
Speed.#3.....: 167 H/s (6.37ms) @ Accel:2 Loops:8 Thr:64 Vec:1
Speed.#*.....: 1046 H/s
Recovered.....: 20/20 (100.00%) Digests, 20/20 (100.00%) Salts
Progress.....: 5833728/6321920 (92.28%)
Rejected.....: 0/5833728 (0.00%)
Restore.Point.....: 285696/316096 (90.38%)
Restore.Sub.#1.....: Salt:12 Amplifier:0-1 Iteration:4992-5000
Restore.Sub.#3.....: Salt:12 Amplifier:0-1 Iteration:1536-1544
Candidate.Engine.: Device Generator
Candidates.#1.....: tmc1al -> txapela
Candidates.#3.....: uclky1e -> unclb111
Hardware.Mon.#1.....: N/A
Hardware.Mon.#3.....: Temp: 66c Fan: 0% Util: 1% Core: 891MHz Mem:1000MHz Bus:4

Started: Sun Jun 20 23:28:53 2021
Stopped: Mon Jun 21 00:18:06 2021
    
```

Figura 5: HashCat del Archivo5

2. ¿A que atribuiría la diferencia de tiempo al crackear cada archivo?

Luego de crackear los archivos con HashCat, se puede apreciar que los tiempos de crackeo son diferentes, en donde los archivos que estaban hasheados con MD5 fueron los de menor demora.

Lo anterior se puede atribuir a que MD5 actualmente se encuentra obsoleto debido a que se logró comprobar que existían colisiones entre sus hashes lo que pone en peligro la integridad de la información que se está hasheado. Esto junto con que MD5 fue desarrollado en 1991 en donde la capacidad de procesamiento y cómputo eran exponencialmente menores que las que se tienen hoy en día y esto se puede ver demostrado ya que aunque los archivos 2 y 3 presentan un salt lo que debería demorar o retrasar un poco el crackeo de los archivos, no lo logra ya que los tiempos son casi el mismo.

Aunque esto no solo se le puede atribuir al MD5 sino también al NTLM el cual tiene una base similar al MD5 por lo que presentan las mismas falencias y los tiempos de crackeo de ambos algoritmos es bastante similar.

¿Cuál cree que es el más seguro?

De los 5 archivos crackeados el quinto el cual fue hasheado con SHA-512Crypt es el más seguro porque hasta la fecha sigue siendo un tipo de hash válido aparte de que solo han logrado atribuirle un par de colisiones.

También según lo visto en clases los hashes del conjunto de SHA son de los más robustos en temas de seguridad por lo que su vigencia y uso están respaldados por las pruebas y tests que se le hacen

3. Nuevo algoritmo Hash

El nuevo algoritmo de hash que se utilizo es PBKDF2 en cual se define como una función de derivación clave con un costo computacional variable, que se utilizan para reducir las vulnerabilidades de los ataques de fuerza bruta.

El trabajo computacional adicional hace que el descifrado de contraseñas sea mucho más difícil y se conoce como extensión de claves. PBKDF2 es por diseño mucho mas lento que SHA, por lo que es mas adecuado para hash de contraseñas porque lleva mucho mas tiempo hacer un diccionario.

4. Nuevo algoritmo Hash

El cifrado asimétrico seleccionado es RSA que por nombre completo tiene Rivest, Shamir y Adleman. Específicamente con el formato PKCS8 que es considerado mas seguro ya que en el envío de la llave publica este no revela información de cual es el nombre del cifrado asimétrico a diferencia del formato PKCS1 que si lo hace.

En el caso del RSA la seguridad del cifrado se basa en el problema de factorización de números enteros y el funcionamiento se centra en el producto de dos números primos grandes seleccionados al azar lo que dificulta exageradamente el poder deducir los números específicos utilizados. Es por esto que este tipo de cifrado es el mas utilizado y que también se puede usar para cifrar para firmar de forma digital.

Este cifrado al igual que la mayoría de las cifras simétricos funciona de la siguiente manera en donde los usuarios requieren una llave pública y privada por lo que cuando se quiere enviar un mensaje el emisor llama a la llave pública del receptor cifrando el mensaje con esto y una vez que el receptor obtiene el mensaje entonces si fuera usando la llave privada.

La integridad del cifrado RSA se va a mantener robusta y constante mientras no sé hay una manera eficaz y de los de poder descomponer los números primos utilizados o sea números primos inmensos aunque según lo visto en clase se estima que a través de los avances tecnológicos los computadores cuánticos podrían llegar a destruir la robustez de di algoritmos de cifrado como estos dejándolos obsoletos.

5. Explicación funcionamiento de los códigos

Al inicio se importan las librerías:

```
import socket
import os
import sys
import subprocess
import binascii
import time
import asyncio
from Crypto.Protocol.KDF import PBKDF2
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
```

Figura 6: Librerías de los códigos

5.1. Código del server

El código del servidor inicia creando las llaves privada y pública correspondientes a la librería de rsa para seguir con la creación del socket que enviará la llave pública al cliente y muestra el siguiente resultado por consola

```
private_key = RSA.generate(1024)
public_key = private_key.publickey()
print (private_key)
print (public_key)

mysocket = socket.socket()
host = "127.0.0.1"
port = 5000
mysocket.bind((host, port))
mysocket.listen(5)
c, addr = mysocket.accept()
encrypt_str = (b"encrypted_message=")
```

Figura 7: Creación de las llaves

```
Private RSA key at 0x27CF483DFD0
Public RSA key at 0x27CF450F1F0
Conectado con el cliente.
Llave pública enviada al cliente...
Recibido: mensaje cifrado es 12a2ab3c88a60d6b07...
```

Figura 8: Respuesta de consola

Luego de enviar las claves el servidor queda en modo de espera mientras desde el lado del cliente recibe la llave pública y cifra los mensajes que se enviaran al servidor. Una vez que el servidor comienza a recibir los datos cifrados realiza el proceso de descifrarlos de la siguiente manera.

```
if data == b"Client: OK":
    print("Conectado con el cliente.")
    time.sleep(2)
    c.sendall(public_key.exportKey( passphrase=None, pkcs=8))
    print ("Llave pública enviada al cliente...")

elif encrypt_str in data:
    data = data.replace(encrypt_str, b'')
    data = binascii.b2a_hex(data)
    print ("Recibido: mensaje cifrado es " + data.decode())
    decryptor = PKCS1_OAEP.new(private_key)
    decrypted = decryptor.decrypt(bytes.fromhex(data.decode()))]
    c.send(b"Server: OK")
    print ("El mensaje descifrado es:\n " + decrypted.decode())

elif data == b"Quit": break
```

Figura 9: Código del server

Cuando el cliente empieza a enviar los mensajes cifrados y éstos llegan al servidor este los descifra utilizando la librería de rsa y obteniendo el contenido del cifrado una vez que realiza esto muestra el resultado por terminal de la siguiente manera.

```
El mensaje descifrado es:
8581c09b6c967bba4d515f94160c9de6
Recibido: mensaje cifrado es 3b5832725dee9a3d69bc027fc661a6219f22fbf091687eeb3cadaf609628bb2e69bbf5f0c4813eb984d3d3821ae
b07ad73fa171cb7594c45bd3e23d405215af8812673b6018c6b5ccd03429ad35f89bb71e302f775ed1a3f3a8049f29936e75336d4b5029cf930b1da6
278ad2818cc2eba2027fa44786c0172c4c4c6cf344673
El mensaje descifrado es:
a74cc7c4032601cb1c07d15f374d2726
Recibido: mensaje cifrado es 533d2e9c9c47acf65da709d3c3a74623a97aa36761c9b3a6a986105c93a8027c0ab2739c2126c05f6ad1120f19f
5c06cbf71f138428703157a57bcca4e20296299107c7ac9fc14b7822e9cd0a5469e653cdebfa2255adeafe7097041e4d99894f4817e67486fce67a1d
58667915681801933f8099f22eacda501db076add6dda
```

Figura 10: Respuesta de consola

Luego de todo lo anterior se finaliza guardando los mensajes descifrados en un archivo txt en la posición respectiva de cada mensaje enviado desde el lado del cliente.

5.2. Código del cliente

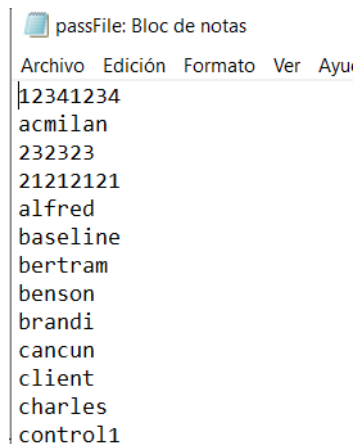
Al principio el código del cliente realiza la labor de identificar el tipo de hash mediante la librería HashID y los comandos de HashCat con los que se procede a crackear los hashes para obtener el texto plano. En esta parte del código también se crea un archivo llamado passFile donde se guarda el texto plano obtenido.

```
print('Los archivos pertenecen a los siguientes hashes:\n')
for i in [0,1,2]:
    b=i+1
    b = str(b)
    n= ('./Archivos/archivo_'+b+'.txt')
    with open(n) as f:
        hola = f.read().splitlines()
        hash1 = hola[0]

        comilla = ""
        hash2 = comilla + hash1 + comilla
        command = "python hash-id.py "
        command2 = hash1
        command3 = " | grep -A1 Possible | grep -v Hash | awk '{print $2}'"
        command4 = command + command2 + command3
        hashcat = subprocess.Popen(command4,shell=True, stdout=subprocess.PIPE,stderr=subprocess.STDOUT)
        a = hashcat.communicate()[0].decode()
        if i == 0:
            os.system('hashcat -a 0 -m 0 ./Archivos/archivo_'+b+'.txt ./Archivos/diccionario_1.dict ./Archivos/diccionario_2.dict')
            os.system("cat dictFull | " + "sed 's:/ /g' | " + "awk '{print$2}'" + " > passFile")
        else:
            print ("ERROR")

    print('archivo '+b+ ' = '+a.rstrip('\n'))
```

Figura 11: Identificado de los hash



```
passFile: Bloc de notas
Archivo Edición Formato Ver Ayu
12341234
acmilan
232323
21212121
alfred
baseline
bertram
benson
brandi
cancun
client
charles
control1
```

Figura 12: Contraseñas en texto plano

Una vez que quieras los 5 archivos obtienen todas las contraseñas en texto plano dentro del archivo PAC file y se procede a hacer charla con el nuevo algoritmo siendo éste RSA con el formato PBKBF2.

```

0 = open(['NewHash.txt', 'w'])
with open('passFile', 'r') as g:
    hola = g.read().splitlines()
    j = 0
    while True:
        if j < len(hola):
            hola1 = hola[j]
            x = PBKDF2(hola1, b'hola', 16, 1000, None, None)
            y = binascii.b2a_hex(x)

            print(y.decode())
            0.write(y.decode()+'\n')
            j = j+1
        else:
            break
0.close()
g.close()

```

Figura 13: Nuevo hasheo de contraseña

Luego se crea un archivo llamado NewHasH en el cual se guardan las contraseñas cacheadas con el algoritmo descifrado. Esto se realiza de manera iterativa contraseña por contraseña (Cada hash de las contraseñas tiene un largo fijo de 30 caracteres).

```

03c1ebcf4a6039628e4c79c4ac212cb
9f0ec70112ae3a746c32c2b62bbe139f
b67881ccb8bebe4a4a80fdc1c006a3a0
fec735fd58108c0bffa8f3a0547447c6
83425ecd55b8ce1066e0711e855502
6c5466a03b7ec21644917247a86d5e94
6cfa2cd614ef16f77d625fa4ac25affb
d245ca8eb51b8417f7113d7374d7c287
a0ab1e2b208202f56d52b460f874c791
c08ce4ed4ad319a89e09fac4a1dda18e
716b5ac96b4502f8ce05b9a03c6ec9df
06f2530757064679d9bf599a1a8c97eb
c2761c2efde23e608e7c59097ece603b
807d01f81ffc218f1ab562d7d8854cea
b639fb9e934e559317017ff75c73fab7
4956307cfffcc40a391f85c43679ec82e
fb11b410ef4bdfda6f66ca9af9353cd
fa6cdf57c83f90b94fb7254958461a18
1e73db0134a5014c4dc1e96a776f2f26
5f1159598a6503d60abd20b4cb075004
9bbb5f10a52dc3df8ab6bcab16ec1218
b80492f01d8d06778f7d4c9058f52d64
f93cffa74c4a931affcdc68b7dd19b6f
8202d6574f6b8adb35611e167eab6

```

Figura 14: Contraseñas hasheadas

Por último se procede a crear el socket para poder recibir y solicitar la llave pública del servidor con la que se cifrarán los mensajes que se enviarán al archivo NewHash que contiene las contraseñas.

```

server = socket.socket()
host = "127.0.0.1"
port = 5000
x = "NewHash.txt"
server.connect((host, port))

server.sendall(b"Client: OK")

server_string = server.recv(1024)

server_string = server_string.replace(b"llave publica=", b'')
server_string = server_string.replace(b"\r\n", b'')

server_public_key = RSA.importKey(server_string)
    
```

Figura 15: creacion del scket

Luego se abre recibo el archivo NewHash Para ir cifrando las contraseñas cacheadas con la llave pública que posteriormente serán enviadas al servidor para que éste pueda almacenar las en el archivo de texto correspondiente finalmente el cliente envía el mensaje de confirmación al servidor esto se hace por cada contraseña una vez terminado el ciclo se cierra el socket y finaliza el código.

```

with open(x, 'rb') as m:
    encryptor = PKCS1_OAEP.new(server_public_key)
    message = m.read().splitlines() #archivo a enviar
    l = 0
    while True:
        if l < len(message):
            time.sleep(0.2)
            hi = message[l]
            encrypted = encryptor.encrypt(hi)
            y = binascii.b2a_hex(encrypted)
            notificacion = b"encrypted_message="
            time.sleep(0.25)
            server.sendall(notificacion + encrypted)
            l=l+1
            print("Mensaje cifrado enviado...")
        else:
            m.close()
            break
    
```

Figura 16: Contraseñas hasheadas

```

0307d44c143ddc97c91071db02e17301
a10db60513431ccde6c3f323c0ce0a51
Mensaje cifrado enviado...
Mensaje cifrado enviado...
Mensaje cifrado enviado...
Mensaje cifrado enviado...
Mensaje cifrado enviado...
Mensaje cifrado enviado...
Mensaje cifrado enviado...
Mensaje cifrado enviado...
    
```

Figura 17: Contraseñas hasheadas

6. Link del GitHub

<https://github.com/MarcosFantoval2/CriptoT4>