Crypto - Chantaje en la oficina

Autor:

Alejandro González Momblán (alias Yisus_Christ) -
alejandro.gonzalezm02@estudiante.uam.es

Introducción:

Se ha resuelto el problema y descifrado la flag "flag{L0_sI3Nt0_s0l0_3r4_Un4_Br0M4}" a través de una conexión remota SSH a la dirección especificada y un poco de observación en los ficheros y rastro de comandos bash empleados anteriormente.

Indice

1. Conexión al equipo remoto

En primer lugar debemos establecer una conexión al equipo remoto en la dirección IP 52.59.205.171 a través del puerto 2222 y como se nos indica que el usuario es 'office' podemos ejecutar la siguiente instrucción:

```
ssh office@52.59.205.171 -p 2222
```

Se nos mostrará un mensaje de aviso sobre la seguridad en la conexión respecto a la autenticidad del host (se indica además el código hash del fingerprint del host), aceptamos para guardar dicho host en una lista de equipos permitidos y posteriormente se nos pedirá la contraseña del equipo (en este caso '0ffic3').

2. Investigación de comandos y ficheros

Una vez hemos establecido la conexión remota si hacemos un 'ls' en la ruta por defecto veremos un fichero con nombre *mensaje.txt* que nos avisa del problema ante el que nos encontramos. Si echamos un vistazo a los últimos comandos bash introducidos por este trabajador molesto veremos algunos que nos llaman la atención:

```
cd Templates/template_184
```

Esto nos permite saber que el usuario hizo todas las operaciones desde el directorio */home/office/Templates/template_184*

```
python3 encrypt.py -e 'topSecret.txt'
rm topSecret.txt
```

Después de desarrollar un programa en python con nombre *encrypt.py* el usuario cifró un fichero llamado 'topSecret.txt', que posteriormente eliminó para no dejar rastro del fichero original.

```
mv topSecret.txt.enc ../template_99/
```

Por último, el fichero resultante cifrado fue movido al directorio */home/office/Templates/template_99*, y si intentamos ver su contenido, naturalmente no será posible ya que está cifrado.

El contenido ilegible de dicho fichero es el siguiente:

```
gAAAAABhtx_9A82MyqEEO3CGqcLSVsAgJL1HNIHbvb1FEKraVdaB1xYKU51tuhZjSn2jL_bzr6B
qngVPzGjb49hG1_I1Yu0qXACuAo41S4BG3Vyi-XfZ7WrvmriEdILIazCmmxwQFBrp
```

3. Análisis del fichero *encript.py*

Desde un primer momento nos damos cuenta de que no va a resultar fácil recuperar el fichero 'topSecret.txt' eliminado, por lo que la opción más recomendable es tratar de descifrar el fichero encriptado que ha dejado el usuario en la ruta anteriormente mencionada.

Para poder hacer esto primero examinamos el fichero *encript.py*:

```
from cryptography.fernet import Fernet
from sys import argv, exit, exc_info
import base64, hashlib, os

error = ("\nSorry, bad usage\n")

if (len(argv) < 2):
    print(error)
    exit(1)


def create_message():
    mensaje = "Tu secreto ha sido encriptado. Ahora harás lo que yo quiera.
La próxima vez deberias currarte más la contraseña de tu usuario. JAJAJA."
    with open ("mensaje.txt", "w") as file:
        file.write(mensaje)
```

```
def encrypt(filename, key):
    f = Fernet(key)
    with open(filename, "rb") as file:
        # read the encrypted data
        file_data = file.read()
    # encrypt data
    encrypted_data = f.encrypt(file_data)
    # write the encrypted file
    with open(filename + ".enc", "wb") as file:
        file.write(encrypted_data)


def decrypt(filename, key):
    if (key == key_64.decode("utf-8")):
        f = Fernet(key)
        with open(filename, "rb") as file:
            # read the encrypted data
            encrypted_data = file.read()
        # decrypt data
        decrypted_data = f.decrypt(encrypted_data)
        base = os.path.splitext(filename)[0]
--

def encrypt(filename, key):
    f = Fernet(key)
    with open(filename, "rb") as file:
        # read the encrypted data
        file_data = file.read()
    # encrypt data
    encrypted_data = f.encrypt(file_data)
    # write the encrypted file
    with open(filename + ".enc", "wb") as file:
        file.write(encrypted_data)


def decrypt(filename, key):
    if (key == key_64.decode("utf-8")):
        f = Fernet(key)
        with open(filename, "rb") as file:
            # read the encrypted data
            encrypted_data = file.read()
        # decrypt data
        decrypted_data = f.decrypt(encrypted_data)
        base = os.path.splitext(filename)[0]
        print("\n\n", decrypted_data.decode("utf-8"), "\n")

    else:
        print("\nIncorrect key\n")


my_password = 'sorry'.encode()
key = hashlib.md5(my_password).hexdigest()
key_64 = base64.urlsafe_b64encode(key.encode("utf-8"))
```

```
def main(argv):
    if (argv[1] == '-e'):
        encrypt(argv[2], key_64)
        create_message()
    elif (argv[1] == '-d'):
        if len(argv) == 4:
            decrypt(argv[2], argv[3])
        else:
            print(error)
            exit(1)
    else:
        print(error)
        exit(1)

if (__name__ == '__main__'):
    main(argv)
```

Podemos ver que el grueso de la funcionalidad de este programa son las funciones de cifrado y descifrado de un mensaje en concreto, las explicamos a continuación:

- ***create_message()***

  Esta función guarda un mensaje de texto específico en un fichero con nombre 'mensaje.txt'. Con esto ya entendemos como se ha generado lo que encontrábamos la primera vez que hacíamos la conexión SSH.

- ***encrypt(filename, key)***

  Esta función recibe un fichero y una clave y cifra el contenido del fichero en otro distinto (con el mismo nombre pero con extensión 'enc') siguiendo un esquema de cifrado simétrico a través del módulo (Fernet)[https://cryptography.io/en/latest/fernet/]. Vemos ahora como el usuario ha podido ocultar el mensaje original.

- ***decrypt(filename, key)***

  Esta función es muy similar a la anterior, también recibe un fichero y una clave pero esta vez descifra el contenido y lo muestra por pantalla. Es importante destacar que solo funciona si la clave introducida por el usuario es correcta.

Por último, si analizamos la función *main()* nos damos cuenta de que para realizar el proceso de descifrado necesitamos introducir 3 argumentos adicionales a nuestro programa, por lo que el comando que debemos introducir será de la manera:

```
python3 encrypt.py -d 'topSecret.txt.enc' '<secret_key>'
```

4. Solución final

La resolución del problema es muy sencilla una vez hemos analizado con detalle el fichero *encrypt.py*, ya que encontramos diversas pistas por el código:

Por un lado, encontramos las siguientes lineas definidas antes de la función main():

```
my_password = 'sorry'.encode()
key = hashlib.md5(my_password).hexdigest()
key_64 = base64.urlsafe_b64encode(key.encode("utf-8"))
```

Vemos que lo que está ocurriendo es que se toma la contraseña 'sorry', se calcula su hash a través de MD5 y posteriormente se codifica el texto con la base UTF-8, para un correcto procesado como texto plano. Sin embargo, si probamos a ejecutar la función de descifrado con esta contraseña no funciona y se muestra el mensaje *"Incorrect key"*.

Por otro lado si vemos de nuevo la función decrypt() y nos fijamos en la primera linea:

```
if (key == key_64.decode("utf-8")):
```

Nos damos cuenta de que se está comprobando si la contraseña introducida por el usuario coincide con la contraseña 'cifrada' por el programa en base UTF-8. Por lo tanto, no es de extrañar que la contraseña 'sorry' no funcione, ya que no se está comparando el texto en claro sino el resultado hasheado.

Es decir, lo que debemos introducir como contraseña es el mismo hash de la contraseña 'sorry' calculado por el programa. Si hacemos las mismas 3 operaciones anteriores por nuestra cuenta obtenemos el valor de hash **NzNjZjBlMzg4OTcxZWU0ZWMzNGU4ZGFlZGQwZDM2Y2M=** que es justo lo que el programa espera para la comprobación. Por lo tanto, ejecutamos la siguiente instrucción desde la ruta */home/office/Templates*:

```
python3 template_184/encrypt.py -d 'template_99/topSecret.txt.enc'
'NzNjZjBlMzg4OTcxZWU0ZWMzNGU4ZGFlZGQwZDM2Y2M='
```

y obtenemos como resultado por pantalla el mensaje con la solución:
flag{L0_sI3Nt0_s0l0_3r4_Un4_Br0M4}