Análisis forense de un volcado de memoria volátil (RAM) utilizando Volatility.

Daniel López Gala

12 de Diciembre de 2022

1. Introducción.

Se trata de realizar el análisis de memoria RAM de un *dump* de un sistema con Ubuntu cuya extracción se realizará con LiMe, un módulo LKM que permite adquisiciones de memoria volátil de Linux y sistemas basados en Linux, como Android.

El análisis se realizará utilizando Volatility, un framework con herramientas forenses para la extracción de evidencias digitales a partir de muestras de memoria volátil (RAM).

2. Instalación de Volatility.

La versión 2.6.1 de Volatility se puede descargar desde este enlace. Una vez descargada y descomprimida, podremos ver los siguientes archivos.

Figura 1: Volatility 2.6.1

Como se indica en el repositorio, Volatility funciona con Python 2, ya que existe una nueva versión de Volatility llamada Volatility 3 que utiliza Python 3, pero es incompatible con muchos módulos de análisis y da bastantes problemas en general, por lo que se sigue utilizando la versión antigua. Por tanto, debemos instalar Python 2 tal y como se indica aquí.

```
sudo apt install python
```

```
sudo apt install curl
curl https://bootstrap.pypa.io/pip/2.7/get-pip.py --output get-pip.py
sudo python2 get-pip.py
pip2 --version
```

Continuamos con el proceso de instalación de Volatility ejecutando *sudo python2 get-pip.py* para obtener pip de Python 2.

Figura 2: Instalación de get-pip.py

A continuación instalamos las dependencias necesarias de Volatility.

```
pip2 install --upgrade setuptools
sudo apt-get install python-dev
```

```
pip2 install pycrypto
pip2 install distorm3
```

Figura 3: Instalación de las dependencias de Volatility.

Ahora tenemos dos opciones, dirigirnos al directorio de Volatility y ejecutar *./vol.py [comando]* para utilizar la herramienta, o realizar una instalación completa para poder usarlo desde cualquier directorio y poder eliminar el repositorio. Optamos por la segunda opción.

```
sudo python setup.py install
```

3. Creación de un volcado LiMe.

Es posible extraer la memoria de forma local con un comando del tipo *sudo insmod lime-3.16.0-77-generic.ko "path=/home/marcos/Evidencias/MemLub1404 format=raw"*, o de forma remota con *sudo insmod LiME/src/lime-3.16.0-77-generic.ko "path=tcp:4444 format=raw"*.

Esto crearía un puerto de escucha desde el que se podría descargar el volcado de memoria a través de una máquina expuesta a la misma red. En el artículo citado hay un oneliner que permite obtener los hashes adecuados para adjuntarlo en un informe pericial.

```
time ncat 192.168.1.33 && cp MemLubuntu1404 MemLubuntu1404.bak && ls -l
↪  && sha1sum MemLubuntu1404 MemLubuntu1404.bak > HashMemLubuntu.txt &&
↪  cat HashMemLubuntu.txt
```

El primer paso es descargar la herramienta en el sistema.

```
git clone https://github.com/504ensicsLabs/LiME.git
```

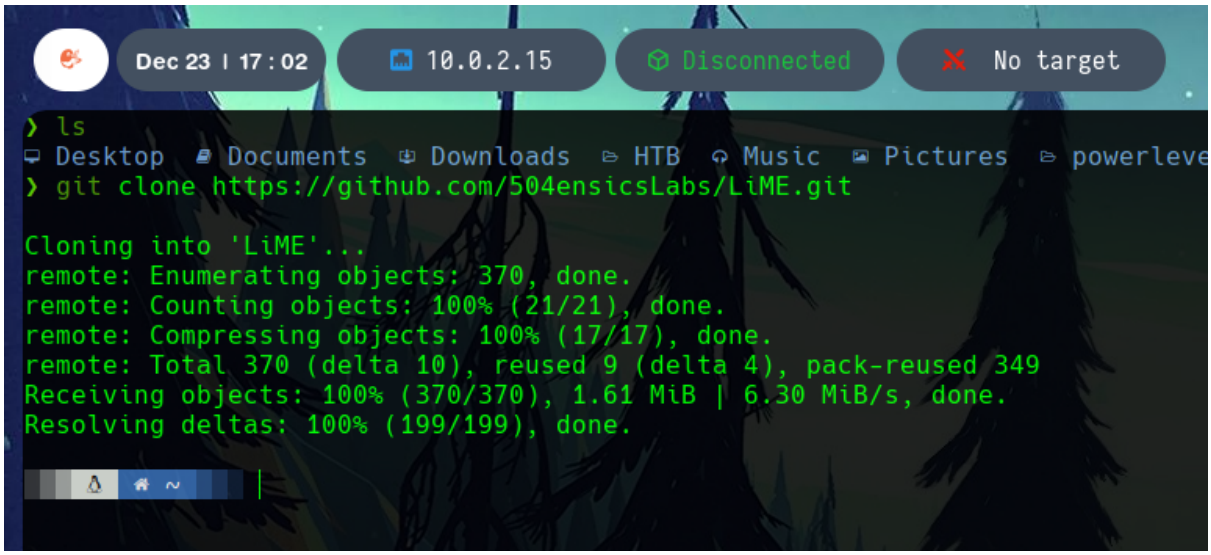

Figura 4: Instalación de LiMe.

A continuación, simplemente accediendo al directorio src de la herramienta y compilando podremos generar el módulo de nuestro sistema, que se corresponde al archivo con formato .ko generado, como podemos ver en la figura 5.

```
cd LiME/src
make
```

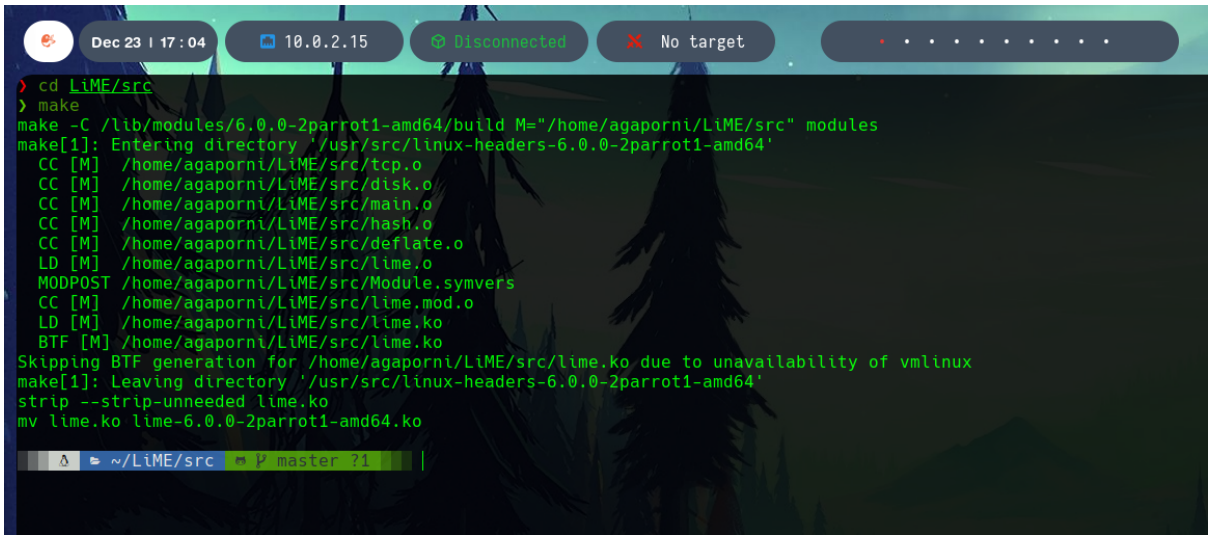

Figura 5: Extracción del volcado de memoria.

4. Análisis del volcado.

Ahora comenzaremos con el análisis de un volcado de memoria. Simularemos un ejemplo práctico e intentaremos obtener información relevante a partir de un volcado LiMe como el que acabamos de crear.

4.1. Situación inicial.

Se nos ofrece un fichero .zip que contiene un volcado de memoria de un sistema Linux.

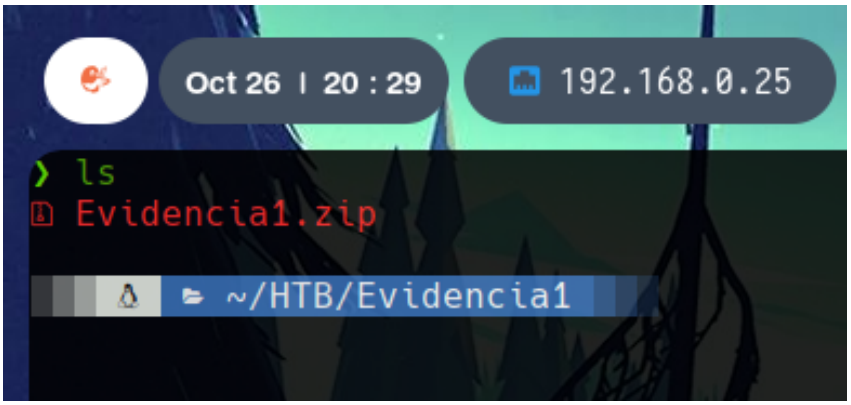

Figura 6: Evidencia.

En este comprimido nos encontramos el volcado de la evidencia en formato .lime y un archivo module.dwarf junto al directorio boot que contiene el archivo System.map del sistema de donde se ha extraído la evidencia. Esto será necesario para la creación del profile.

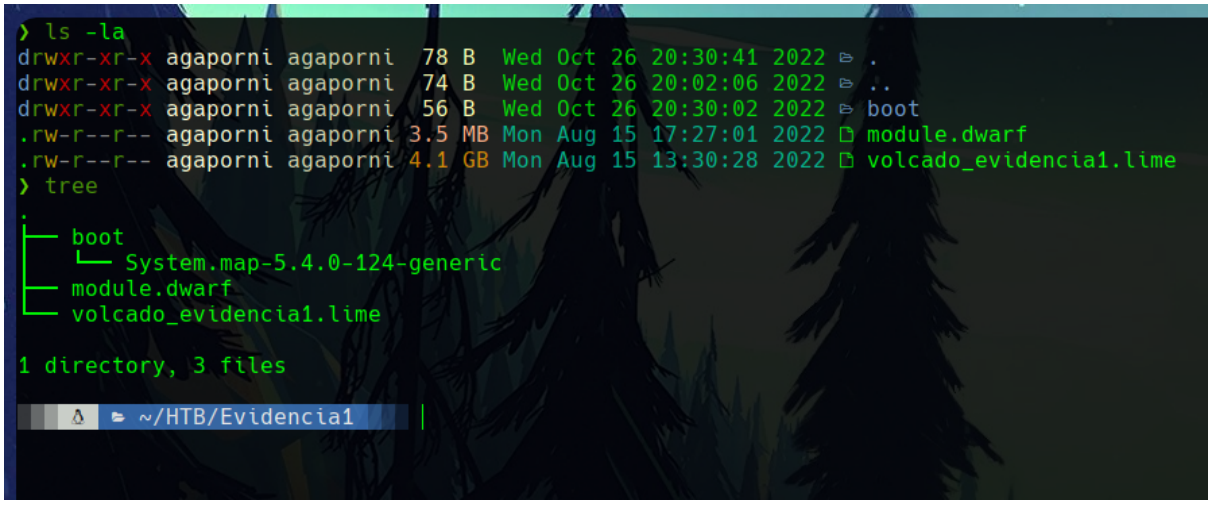

Figura 7: Contenido de la evidencia.

4.2. Creación del perfil.

A continuación tenemos que crear un perfil de volatility con el System.map y el archivo module.dwarf. Esto puede hacerse con el comando zip de la siguiente manera.

```
zip profile-5.8.0-38-generic System.map-5.8.0-38-generic module.dwarf
```

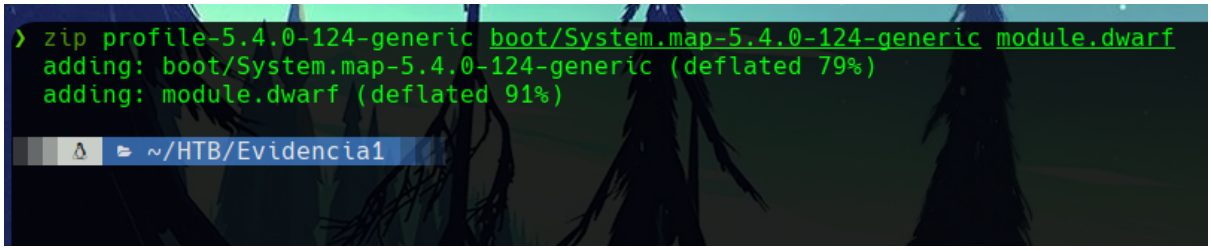

Figura 8: Creación del perfil.

Ahora hay que copiar el profile a la carpeta *volatility/plugins/overlays/linux/*.

En nuestro caso, como hemos instalado volatility con el setup, la ruta de los plugins estará en el directorio /usr/local/lib/python2.7/dist-packages/, concretamente en el paquete */usr/local/lib/python2.7/dist-packages/volatility-2.6.1-py2.7.egg/volatility/plugins/overlays/linux*.

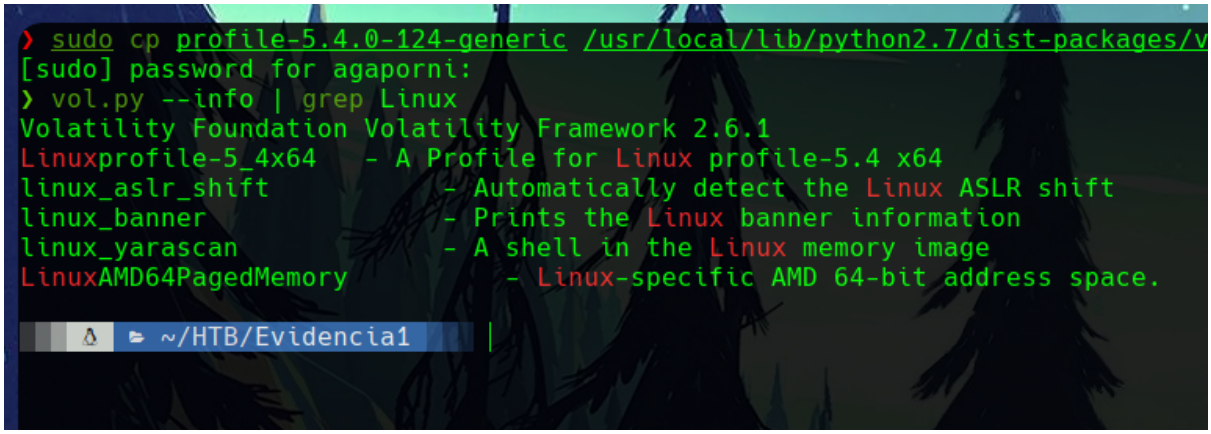

Figura 9: Instalación del perfil.

Podemos verificar que el perfil está correctamente instalado utilizando Volatility con el comando –info y filtrando por Linux.

```
vol.py --info | grep Linux
```

Este comando muestra los perfiles de Linux que están instalados en la herramienta. Por defecto, Volatility contiene algunos perfiles de las distribuciones de Linux y los kernels más frecuentes, pero en casos como este es necesario añadirlo manualmente.

4.3. Procesos en ejecución.

Para utilizar la herramienta harán falta, por norma general, 3 parámetros. El primero, el archivo del volcado .lime, seguido del identificador del perfil que estamos usando, y por último el plugin que queremos aplicar para analizar la imagen.

```
vol.py -f volcado_evidencia1.lime --profile=Linuxprofile-5_4x64 linux_psscan
```

Para identificar los procesos de un sistema se puede utilizar el plugin *pslist*, que recorre la lista que apunta a *PsActiveProcessHead*. El plugin *pstree* los enumera con la misma técnica pero los imprime con estructura de árbol. Por último, podemos utilizar *psscan* que además de lo anterior muestra el ID del offset, procesos inactivos, u otros ocultados. Es decir, realiza un escaneo en el volcado.

En nuestro ejemplo, utilizando *psscan* podemos identificar los siguientes procesos.

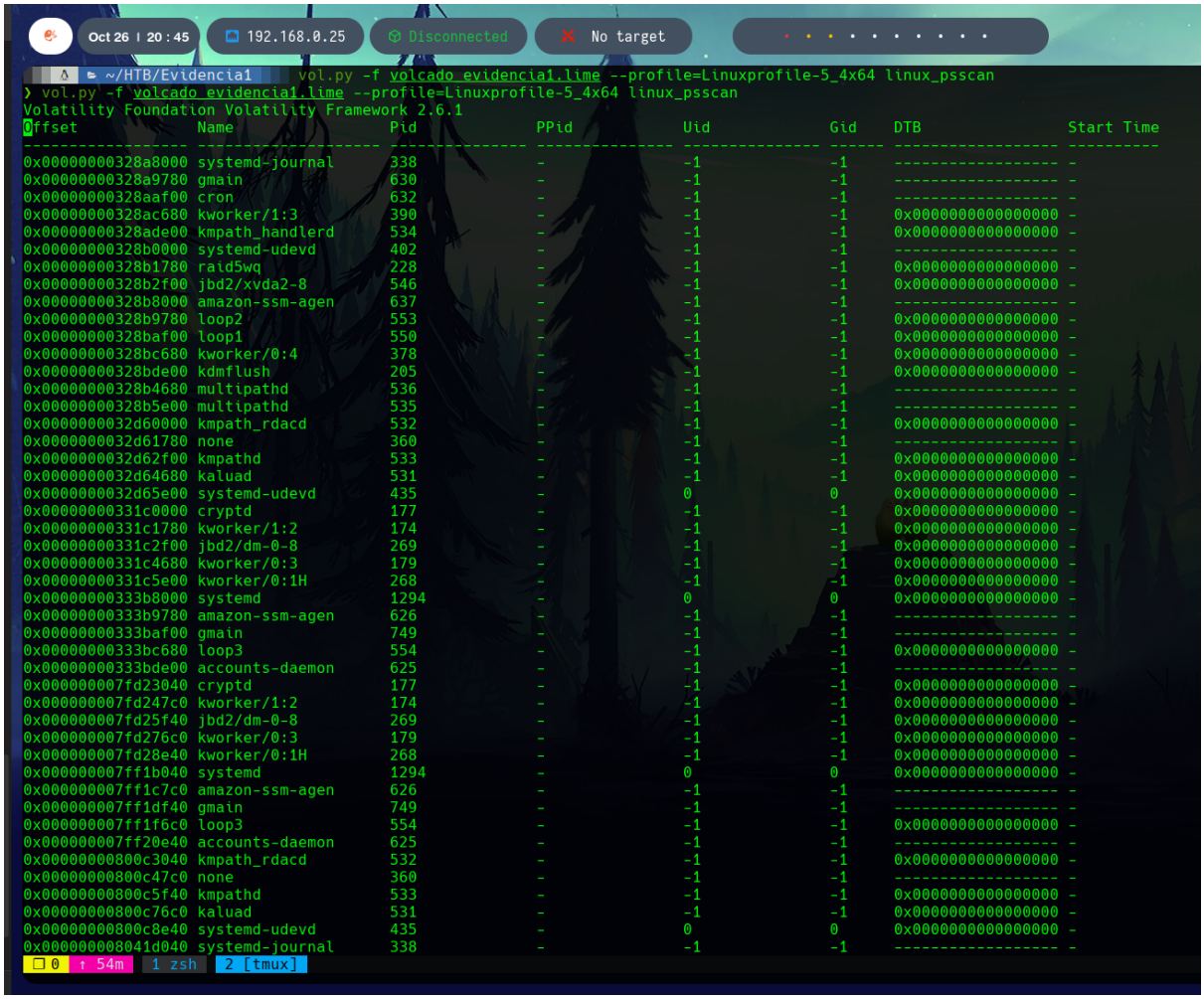

Figura 10: Procesos en ejecución.

4.4. Análisis de sockets.

Podemos utilizar algunas herramientas para recuperar información sobre las conexiones del sistema. En concreto, podemos usar *netscan* para realizar un escaneo TCP y UDP en la propia máquina. Esto nos permite identificar qué servicios pueden estar haciendo uso de determinadas conexiones. El comando sería el siguiente

```
vol.py -f volcado_evidencia1.lime --profile=Linuxprofile-5_4x64 linux_netscan
```

Y como podemos ver, existe una comunicación relativa al puerto 8080 que muy seguramente se trate de un servidor web.

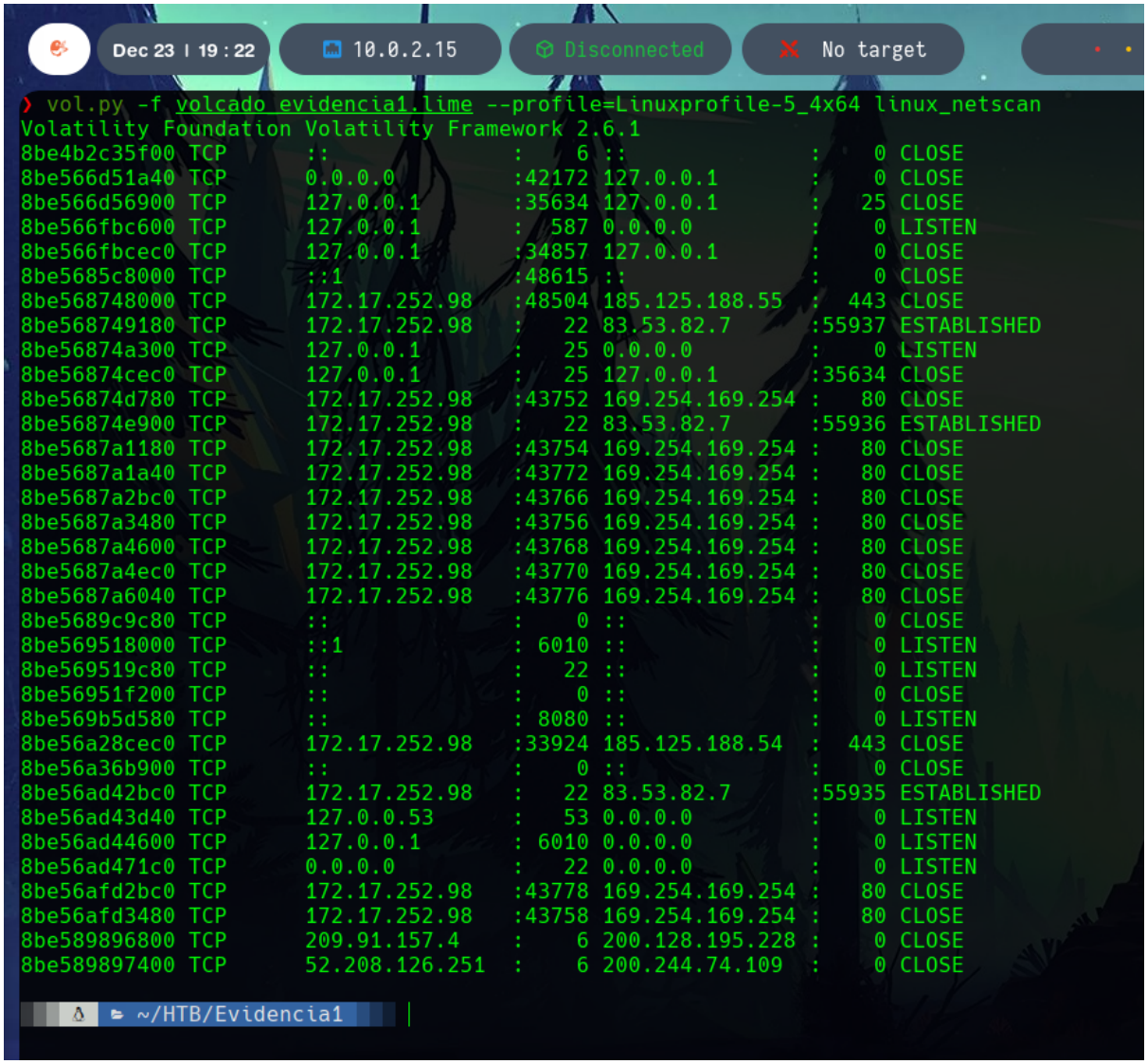

Figura 11: Sockets.

4.5. Recuperar un archivo.

Ahora podemos explorar el directorio */var* para tratar de encontrar información relevante acerca del servidor web que podría estar alojándose en este sistema. Para ello, se puede utilizar el plugin *linux_enumerate_files* y filtrar por el directorio que estamos buscando de la siguiente manera.

```
vol.py -f volcado_evidencia1.lime --profile=Linuxprofile-5_4x64
↪  linux_enumerate_files | grep /var
```

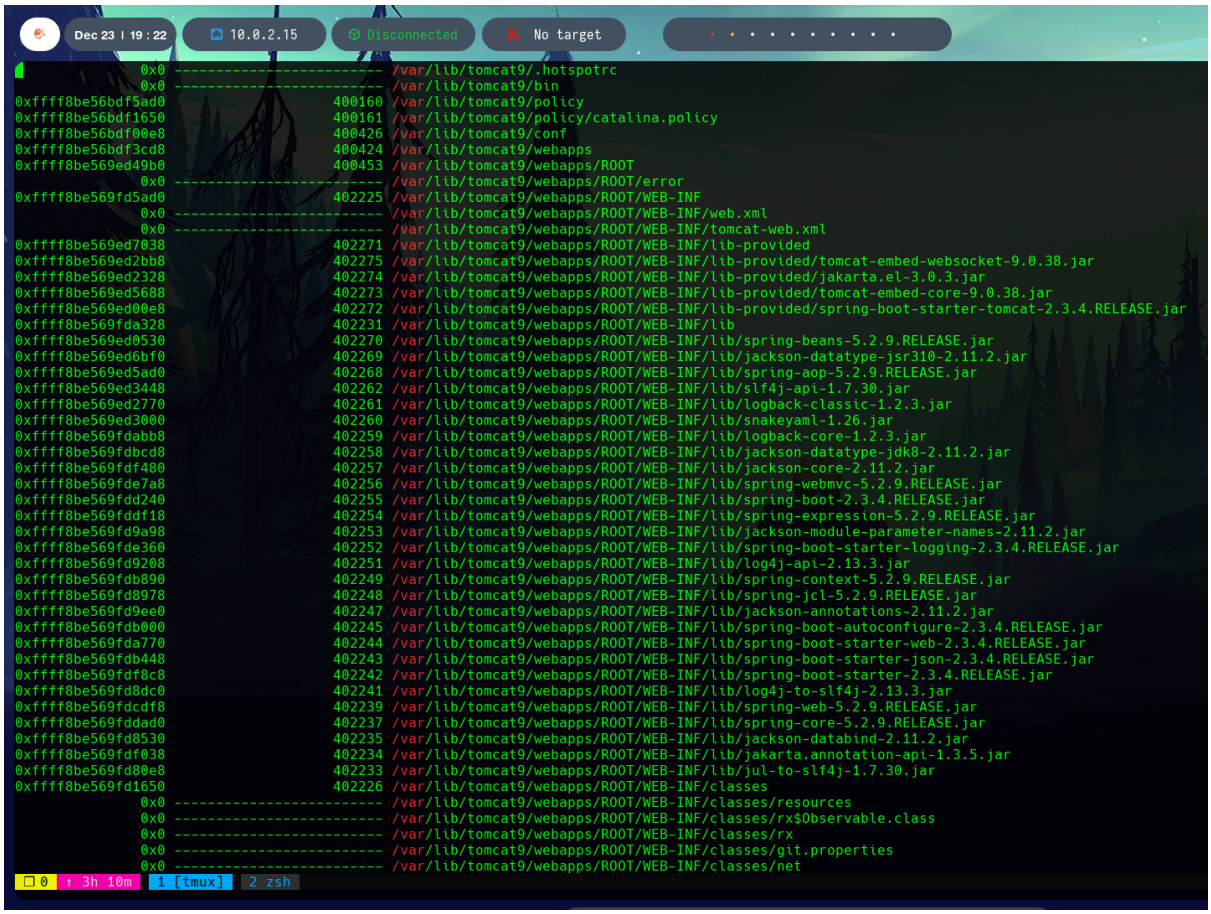

Figura 12: Directorio /var.

Gracias a este reconocimiento, podemos ver el directorio /var/lib/tomcat9, que se corresponde a un servidor web Tomcat. Podemos intentar recuperar algún archivo relevante del servidor.

Como sabemos que la máquina hosteaba un servidor de Tomcat, buscaremos el inode del archivo *login.js* en el volcado utilizando el plugin *linux_find_file* y la ruta del archivo *login.js*.

```
vol.py -f volcado_evidencia1.lime --profile=Linuxprofile-5_4x64
↪  linux_find_file -F
↪  "/var/lib/tomcat9/webapps/ROOT/WEB-INF/classes/static/login.js"
```

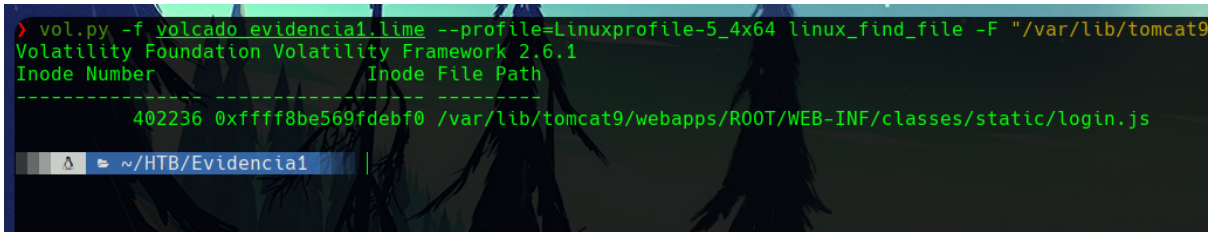

Figura 13: Inode del archivo login.js del servidor Tomcat.

Como se puede ver, obtenemos el número de inodo y su dirección en hexadecimal. Esto nos permite utilizar el mismo plugin con la flag -i para recuperar el contenido del archivo, y la flag -O para indicar dónde queremos que lo guarde.

```
vol.py -f volcado_evidencia1.lime --profile=Linuxprofile-5_4x64
↪  linux_find_file -i 0xffff8be569fdebf0 -O login.js
```

No obstante, al ejecutarlo obtuve el siguiente error.

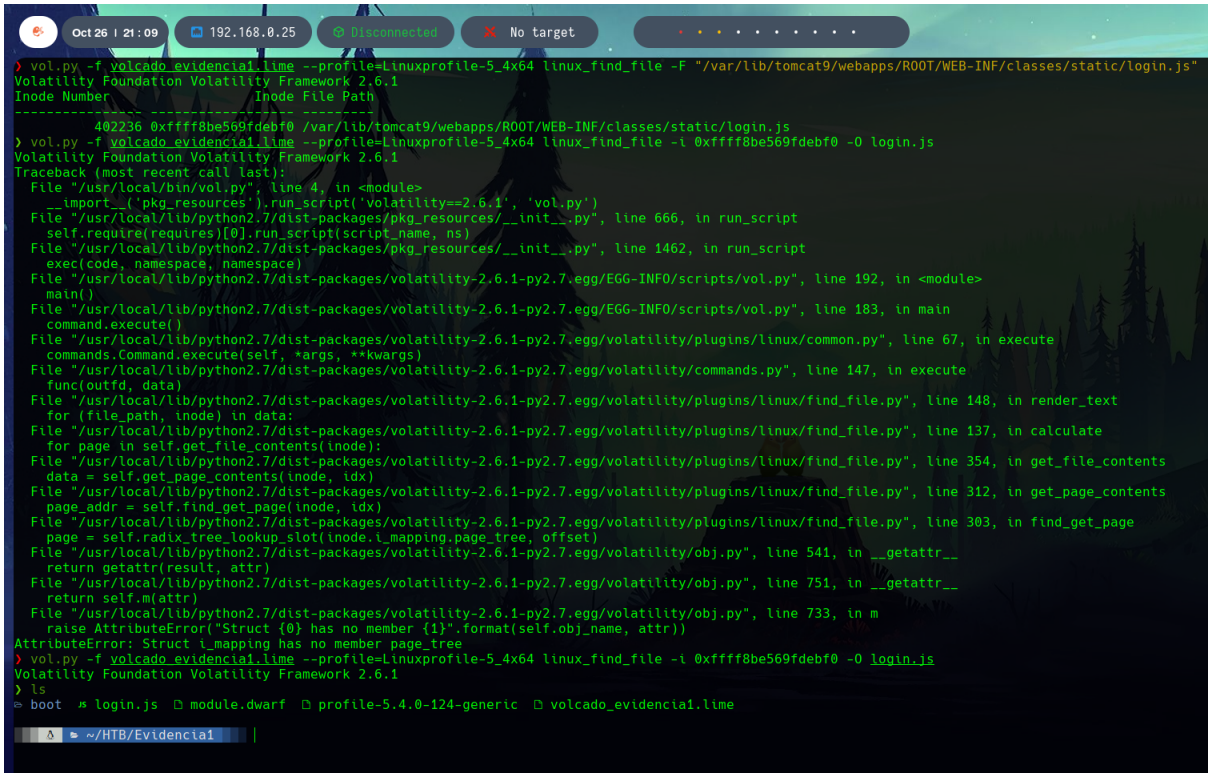

Figura 14: Error y solución en Volatility.

Pude solucionarlo modificando el script *linux_find_file.py* a mano. En el repositorio de Volatility se encuentra el script actualizado. En la imagen anterior se ve al final la ejecución del mismo comando, pero sin ningún error.

Ahora podemos leer el archivo, aunque viene en hexadecimal porque proviene directamente del volcado.

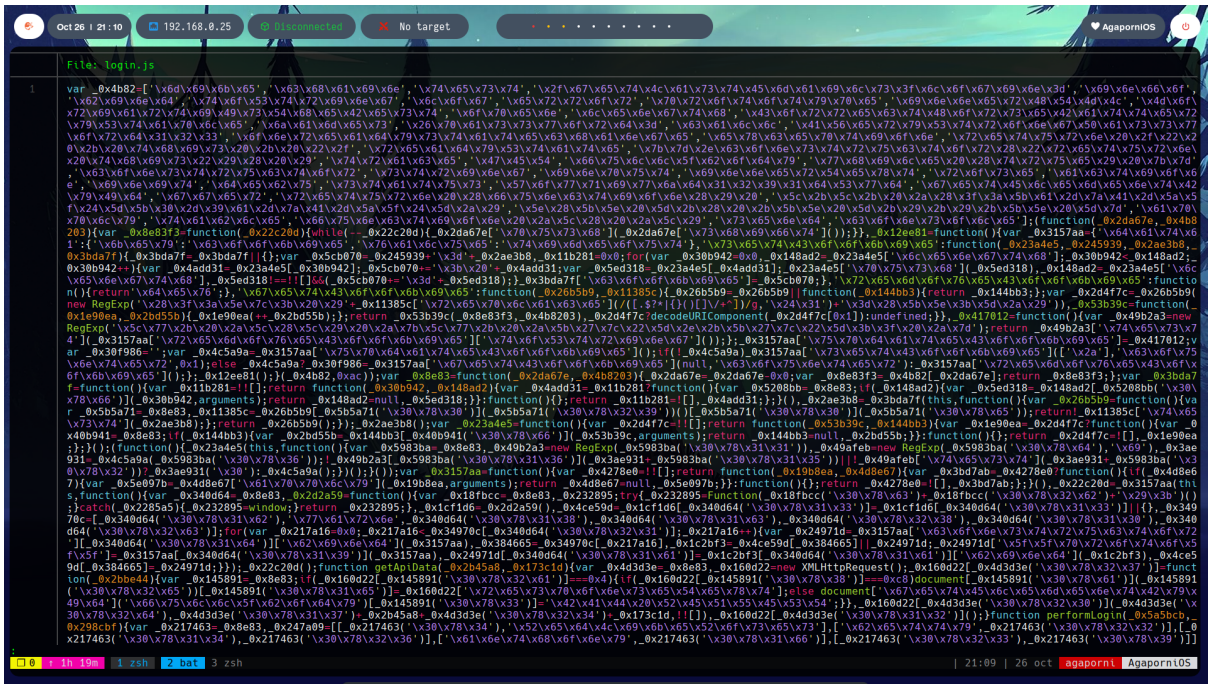

Figura 15: Archivo login.js

Es sencillo deducir que en la variable declarada al comienzo del archivo se almacenará toda la información relativa al login, pues contiene las variables necesarias del propio archivo js. Con un editor de texto podemos convertir correctamente el vector para quedarnos con las strings en hexadecimal.

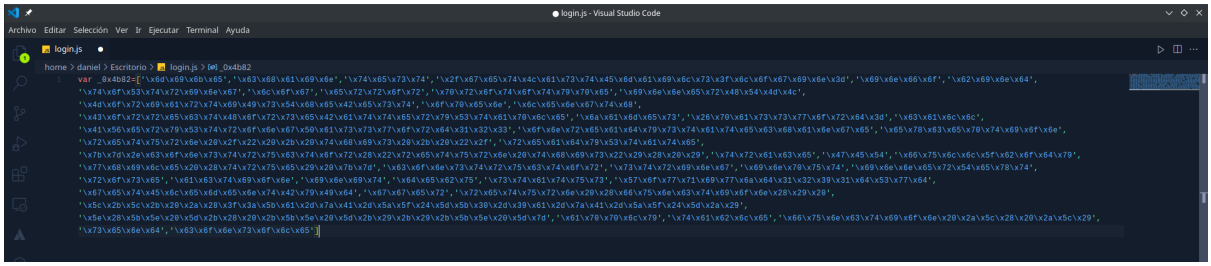

Figura 16: Strings en raw.

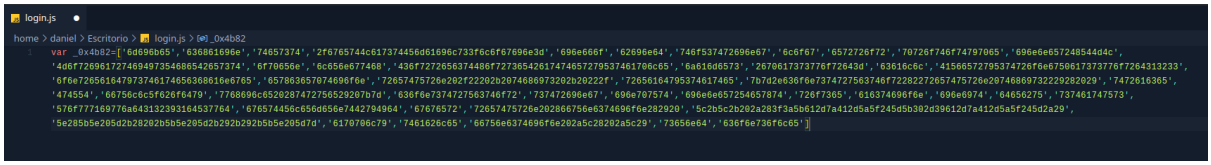

Figura 17: Strings en hexadecimal.

Hay muchas maneras de procesarlas, pero ya que tenía el VS code abierto y la sintaxis del vector es la misma que en Python, con un pequeño bucle podemos convertir todas las variables en hexadecimal a una string normal para leer su contenido. El script es el siguiente.

```
for each in _0x4b82:
    print(bytes.fromhex(each).decode("utf-8"))
```

Y el resultado, como podemos ver, desvela que una de las variables es la contraseña del servidor web Tomcat.

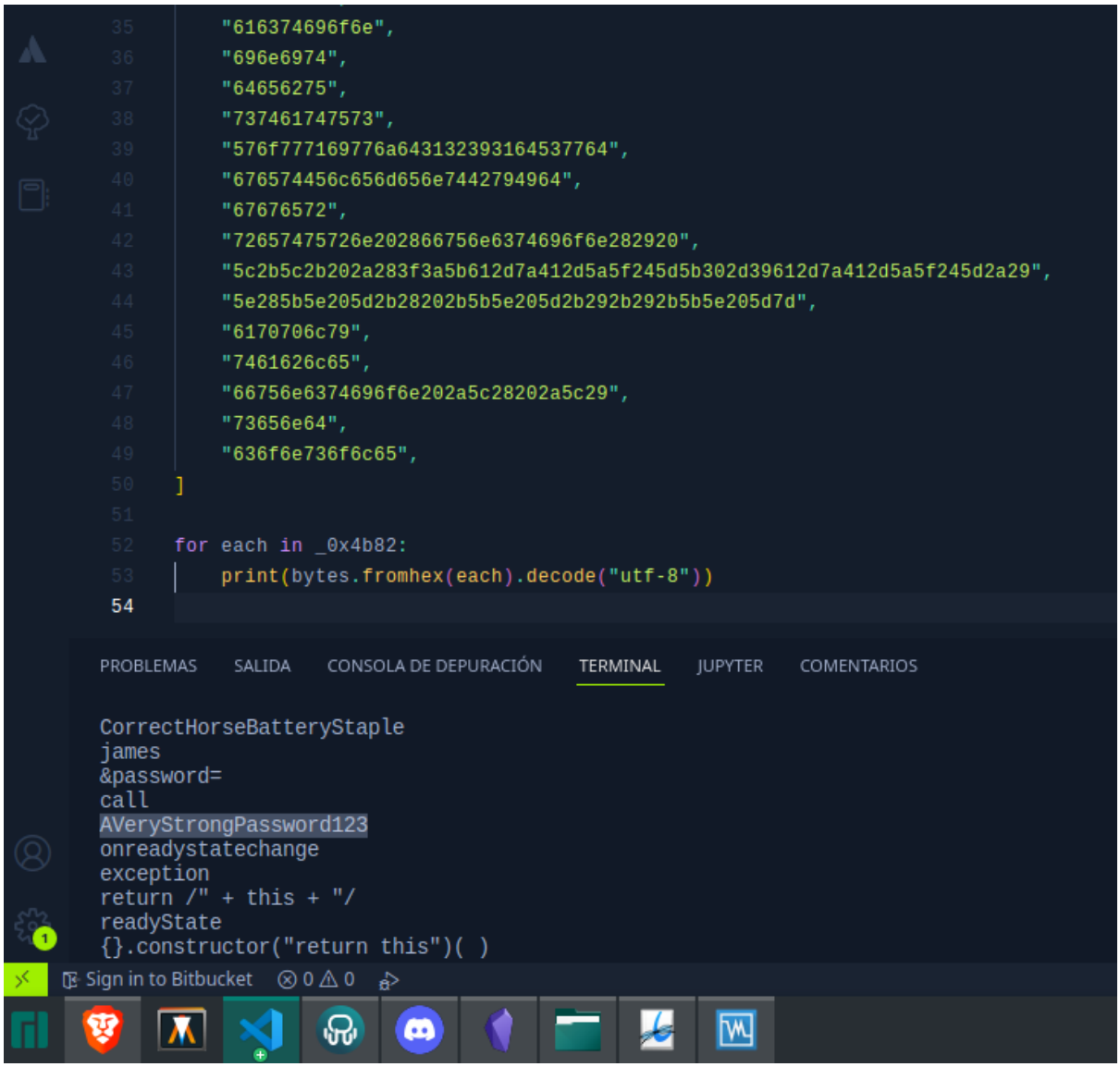

Figura 18: Contraseña de Tomcat.

Obviamente, esto es sólo un ejemplo didáctico de lo que se puede hacer con Volatility. Es una herramienta muy compleja y muy utilizada en análisis forense, ya que permite extraer pruebas periciales que sirven para demostrar qué ocurre o que ocurrió en un determinado sistema informático.