

# L'analyse forensique de la mémoire volatile (RAM) avec Linux

GCSE | BOUSSETA HATIM |BOUSSETA.HATIM@ETU.UAE.AC.MA



|12/01/2024| Encadré par : Dr. Manar Kassou

0

## Table des matières

### Partie 1 : Introduction

1. L'analyse forensique (Digital forensics).
2. la mémoire volatile .

## **Partie 2 : Les étapes principales pour analyser une mémoire volatile dans un cadre forensique sur Linux :**

1. Acquisition de la mémoire volatile .
2. Analyse de la mémoire volatile (RAM) .

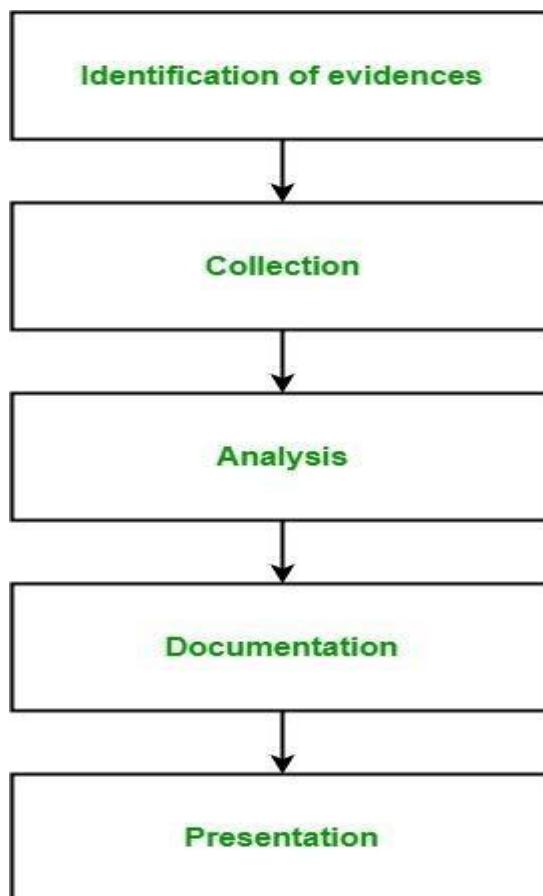
## **Partie 3 : Conclusion**

1. Vérifications et tests .
2. Investigation pratique.

# **PARTIE 1 :**

## **1- L'analyse forensique (Digital forensics)**

En **Cybersécurité**, la **forensique numérique** concerne l'investigation des systèmes informatiques, réseaux, et appareils électroniques pour identifier des activités suspectes, collecter des preuves exploitables, et analyser les causes d'incidents de sécurité. Elle utilise des outils spécialisés pour examiner disques, mémoire, logs et réseaux, tout en garantissant l'intégrité des données. Le **cycle de la digital forensics** se compose de plusieurs phases essentielles, organisées pour garantir une enquête efficace et la validité des preuves :



Le cycle de la digital forensics garantit une enquête traçable et juridiquement valide pour préserver, analyser et présenter des preuves numériques.

Les investigations numériques sont régies par des principes clés qui garantissent l'intégrité et la légalité du processus. Voici les principes fondamentaux à respecter :

- La **légalité** : respect des lois et réglementations en vigueur.
- L'**authenticité** : vérification de la provenance des données  
La **préservation** : prévention de l'altération des données.
- L'**analyse** : interprétation approfondie des données collectées.
- La **documentation** : traçabilité de toutes les actions entreprises.
- La **présentation** : communication claire des résultats de l'enquête.

Le **cadre forensique** désigne l'ensemble des étapes, normes, et bonnes pratiques suivies lors d'une enquête numérique pour garantir l'intégrité des preuves !

La **digital forensics** est une branche vaste, mais ces notions essentielles offrent un aperçu fondamental de son processus, qui va bien au-delà de ces concepts de base.

## 2- LA MEMOIRE VOLATILE :

Une mémoire volatile est un type de mémoire informatique qui nécessite une alimentation électrique pour conserver les données. Lorsque l'alimentation est coupée, les données sont effacées. Les mémoires volatiles sont généralement rapides et utilisées pour stocker temporairement des données en cours d'utilisation, comme la mémoire vive (RAM).

La mémoire volatile est très importante car elle contient des données temporaires, comme les processus en cours, les connexions réseau actives et les traces de logiciels malveillants .

L' analyse de cet mémoire nous permet de découvrir des comportements anormaux, comme des connexions réseau suspectes ou des commandes exécutées par le malware.

Donc quelles méthodes spécifiques peuvent être utilisées pour extraire et analyser efficacement ces données provenant de la RAM et d'autres sources dans un cadre d'enquête forensique ?

## PARTIE 2 :

### LES ETAPES PRINCIPALES POUR ANALYSER UNE MEMOIRE VOLATILE DANS UN CADRE FORENSIQUE SUR LINUX :

#### 1-Acquisition de la mémoire volatile :

= Capturer une image exacte du contenu de la RAM .

**Memory Sources :** /dev/crash , /proc/kcore , /dev/mem

**Pour l'extraction de la RAM, nous allons utiliser deux méthodes : Lime et avml**

Il est important de réaliser cette étape avec soin, car toute modification de la mémoire pendant l'acquisition peut altérer les preuves.

**Notes :** Pour garantir une installation et une configuration réussies de **Lime** et **avml**, j'ai proposé des étapes simples et détaillées à suivre. Pour **Lime**, il faut vérifier que la version du noyau Linux est compatible avec l'outil et s'assurer d'avoir les droits administratifs (root) pour l'utiliser. Si des erreurs surviennent, comme une

incompatibilité du module ou un problème de permissions, j'ai prévu une alternative avec **avml**, un outil facile à installer et à utiliser. En suivant mes étapes, vous pouvez éviter les problèmes courants, comme le manque d'espace disque pour l'image de la RAM ou la corruption des fichiers extraits. Grâce à cette méthode, même en cas d'échec avec Lime, **avml** prend le relais pour garantir une extraction de la RAM fiable et efficace.

**Notes :** Avec **avml**, des problèmes peuvent également survenir, comme l'absence de dépendances nécessaires ou une mauvaise configuration de l'exécutable. Par exemple, si avml n'a pas les permissions d'exécution, une erreur comme **Command not found** ou **Permission denied** peut apparaître. Lors de l'extraction de la RAM, un espace disque insuffisant peut poser problème, car le fichier généré est souvent très volumineux. Cela peut provoquer l'erreur **No space left on device**. Enfin, des interruptions comme une panne de courant ou un problème réseau peuvent corrompre le fichier extrait, rendant l'image inutilisable (**File corrupted**).

Le premier outil qu'on peut utiliser :

### LiME (Linux Memory Extractor) :

LiME (Linux Memory Extractor) est un outil open-source utilisé pour extraire la mémoire vive (RAM) d'un système Linux en temps réel .

Configuration :

1 - Assurez-vous d'avoir un accès root ou des privilèges sudo : 2

```
[root@parrot]~[~/home]
└─#git clone https://github.com/504ensicsLabs/LiME.git
Cloning into 'LiME'...
remote: Enumerating objects: 389, done.
remote: Counting objects: 100% (40/40), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 389 (delta 20), reused 28 (delta 14), pack-reused 349 (from 1)
Receiving objects: 100% (389/389), 1.62 MiB | 540.00 KiB/s, done.
Resolving deltas: 100% (209/209), done.
```

Cela crée une copie locale complète du dépôt, y compris tous les fichiers, l'historique des versions et les branches.

La commande « git » est utilisée pour initialiser un nouveau dépôt Git dans un répertoire local.

3 –Compilation : Compiler signifie transformer du code source écrit dans un langage de programmation (**comme C , C++ ou python**) en un fichier exécutable compréhensible par l'ordinateur.

Dans notre cas on doit compiler LiME, c-a-d transformer son code source en un fichier ([lime.ko](#)) que le noyau Linux peut utiliser pour capturer la mémoire. Cela permet à l'ordinateur de comprendre et d'exécuter les instructions écrites par les développeurs.

Un module noyau est une extension qui ajoute des fonctionnalités au noyau du système d'exploitation ([le cœur de Linux](#)), sans avoir à redémarrer l'ordinateur. Dans le cas de LiME, ce module permet au noyau d'interagir avec la mémoire vive et de capturer son contenu dans un fichier.

L'extension .ko signifie Kernel Object (objet noyau).

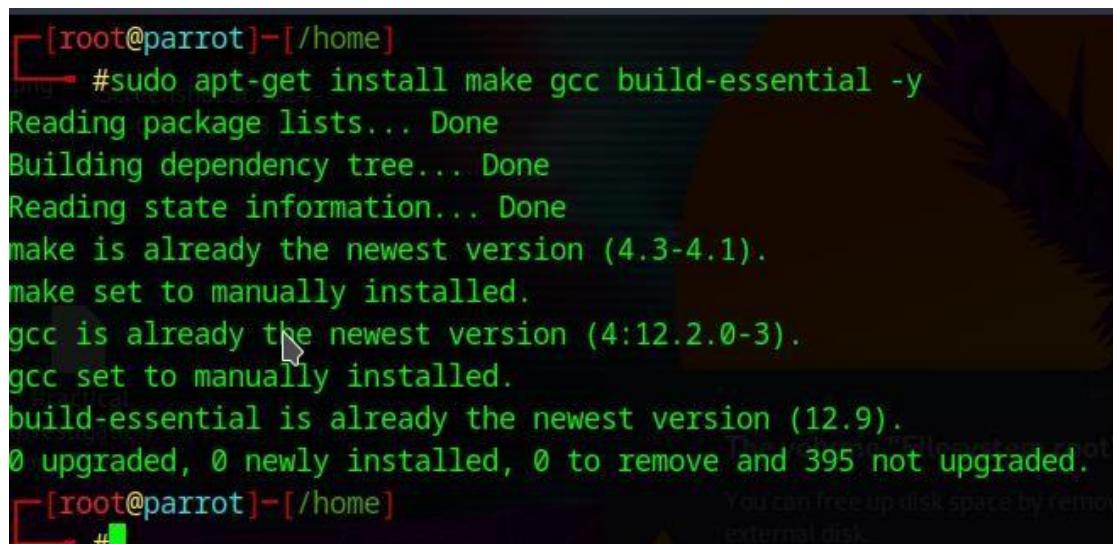
C'est un type de fichier utilisé sous Linux pour les modules noyau. Ces modules sont des programmes ou des morceaux de code qui peuvent être chargés ou déchargés dynamiquement dans le noyau du système d'exploitation pour lui ajouter des fonctionnalités, comme :

Gérer un matériel spécifique (exemple : pilotes).

Ajouter des capacités (comme l'acquisition de mémoire avec LiME).

Un fichier .ko est donc un module compilé, prêt à être intégré au noyau à l'aide de commandes comme insmod (charger) ou rmmod (décharger).

**Installation des outils nécessaires pour compiler des programmes sous Linux.) :**



```
[root@parrot]~[/home]
└─# sudo apt-get install make gcc build-essential -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
make is already the newest version (4.3-4.1).
make set to manually installed.
gcc is already the newest version (4:12.2.0-3).
gcc set to manually installed.
build-essential is already the newest version (12.9).
0 upgraded, 0 newly installed, 0 to remove and 395 not upgraded.
[root@parrot]~[/home]
└─#
```

**make** : Un outil qui automatise la compilation des programmes à partir de fichiers sources en utilisant un fichier Makefile.

**gcc** : Le compilateur GNU C/C++, utilisé pour transformer le code source en fichiers exécutables ou modules (.ko par exemple).

**build-essential** : Un méta-paquet qui installe des outils de développement essentiels, comme les bibliothèques et dépendances pour compiler correctement. (Un métapaquet est un paquet sous Linux qui ne contient pas de fichiers ou de logiciels spécifiques, mais plutôt une liste d'autres paquets à installer. Il permet d'installer plusieurs paquets nécessaires pour une tâche particulière en une seule commande.)

**-y** : Automatise la confirmation de l'installation, évitant d'avoir à répondre "oui" manuellement.

4- Quand tu utilises LiME (Linux Memory Extractor), il crée un module qui se connecte au noyau de Linux pour extraire la mémoire du système. Le noyau est le cœur du système d'exploitation, celui qui gère le matériel et les ressources de l'ordinateur.

La commande `uname -r` te donne la version du noyau qui est en cours d'utilisation.

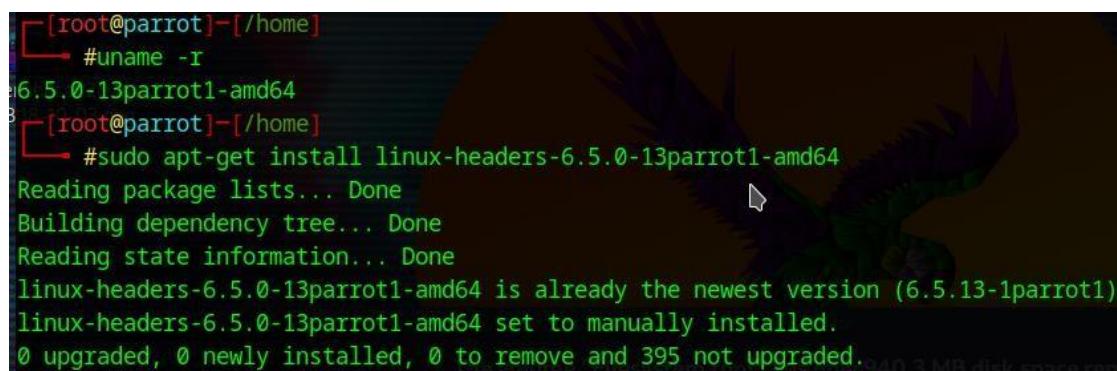
-Pourquoi est-ce important ?

LiME doit être compatible avec cette version du noyau. Si tu utilises une version spécifique du noyau, LiME doit être adapté à cette version pour bien fonctionner.

-Installation des outils : Pour compiler LiME (si nécessaire), tu as besoin des en-têtes du noyau (des fichiers nécessaires pour compiler les modules). Ces en-têtes doivent correspondre exactement à la version du noyau. Sinon, la compilation échouera.

Si ces en-têtes ne sont pas installés, la compilation échouera.

```
sudo apt-get install linux-headers-$(uname -r)
```



```
[root@parrot]~[/home]
└─#uname -r
6.5.0-13parrot1-amd64
3 [root@parrot]~[/home]
└─#sudo apt-get install linux-headers-6.5.0-13parrot1-amd64
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
linux-headers-6.5.0-13parrot1-amd64 is already the newest version (6.5.13-1parrot1)
linux-headers-6.5.0-13parrot1-amd64 set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 395 not upgraded.
```

Lors de l'installation de LiME, plusieurs erreurs peuvent survenir, principalement liées à la compatibilité du module avec la version du noyau. Une erreur fréquente est l'incompatibilité entre le module LiME et la version du noyau utilisé, ce qui nécessite de compiler LiME spécifiquement pour cette version. De plus, si les entêtes du noyau (fichiers nécessaires pour la compilation des modules) ne sont pas

installés, la compilation échouera. L'absence des en-têtes peut entraîner des messages d'erreur tels que "linux/headers not found". Il est aussi possible de rencontrer des problèmes liés à l'absence de certaines dépendances nécessaires à la compilation, ou à des permissions insuffisantes pour accéder aux fichiers nécessaires. Pour éviter cela, il est crucial d'installer les en-têtes correspondants à la version du noyau avec la commande sudo apt-get install linux-headers-\$(uname -r) et de s'assurer que toutes les dépendances sont présentes avant de lancer la compilation.

#### 5 – make :

```
[root@parrot]~[/home]
└─#cd ./LiME
[root@parrot]~[/home/LiME]
└─#ll
total 28K
drwxr-xr-x 1 root root 56 Dec 4 14:16 doc
-rw-r--r-- 1 root root 18K Dec 4 14:16 LICENSE
-rw-r--r-- 1 root root 4.7K Dec 4 14:16 README.md
drwxr-xr-x 1 root root 122 Dec 4 14:16 src
[root@parrot]~[/home/LiME]
└─#cd ./src
[root@parrot]~[/home/LiME/src]
└─#ll
total 44K
-rw-r--r-- 1 root root 2.5K Dec 4 14:16 deflate.c
-rw-r--r-- 1 root root 2.5K Dec 4 14:16 disk.c
-rw-r--r-- 1 root root 5.7K Dec 4 14:16 hash.c
-rw-r--r-- 1 root root 2.5K Dec 4 14:16 lime.h
-rw-r--r-- 1 root root 9.6K Dec 4 14:16 main.c
-rw-r--r-- 1 root root 1.8K Dec 4 14:16 Makefile
-rw-r--r-- 1 root root 1.7K Dec 4 14:16 Makefile.sample
-rw-r--r-- 1 root root 3.7K Dec 4 14:16 tcp.c
[root@parrot]~[/home/LiME/src]
└─#make
make -C /lib/modules/6.5.0-13parrot1-amd64/build M="/home/LiME/src" modules
make[1]: Entering directory '/usr/src/linux-headers-6.5.0-13parrot1-amd64'
  CC [M] /home/LiME/src/tcp.o
  CC [M] /home/LiME/src/disk.o
  CC [M] /home/LiME/src/main.o
```

6- La commande `insmod` est utilisée sous Linux pour insérer un module dans le noyau. `insmod [option] filename [args]`

Dans le cas de **LiME**, la commande `insmod` est utilisée pour charger le module du noyau LiME afin de commencer l'extraction de la mémoire.

Raw est la désignation générique d'un type de fichier d'images numériques issues d'appareils photo numériques ou de scanners

La forme brute (ou format raw) fait référence à des données qui sont stockées sans aucune modification, compression, ou formatage supplémentaire.



```
[root@parrot]~[/home/LiME/src]
[root@parrot]# insmod ./lime-6.5.13parrot1-amd64.ko "home/izu format = raw"
```

Et comme ça on peut capturer une ‘Dump memory’ avec LiME .

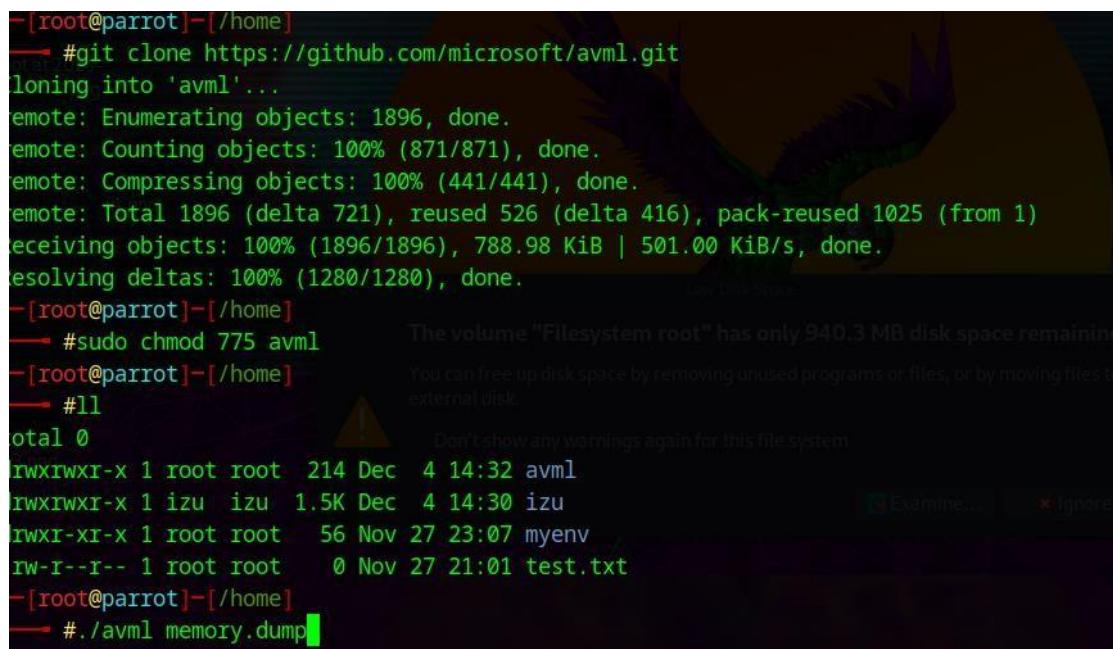
LiME a une configuration très compliquée car il nécessite une compilation avant l'extraction.

**!!!! L'outil AVML est effectivement une bonne alternative, surtout pour ceux qui recherchent un outil simple et rapide à utiliser, sans avoir à passer par la compilation d'un module comme avec LiME.**

### (microsoft) AVML (Acquire Volatile Memory for Linux)

AVML est un outil d'acquisition de mémoire volatile pour les systèmes x86\_64, écrit en Rust, et conçu pour être utilisé comme un binaire statique. Cela signifie que tu peux l'exécuter directement sans avoir à le compiler sur la machine cible, ce qui le rend plus facile à utiliser et plus portable.

Configuration : capture directe de la mémoire RAM avec [ ./avml nom\_du\_dump]



```
[root@parrot]~[/home]
[root@parrot]# git clone https://github.com/microsoft/avml.git
Cloning into 'avml'...
remote: Enumerating objects: 1896, done.
remote: Counting objects: 100% (871/871), done.
remote: Compressing objects: 100% (441/441), done.
remote: Total 1896 (delta 721), reused 526 (delta 416), pack-reused 1025 (from 1)
Receiving objects: 100% (1896/1896), 788.98 KiB | 501.00 KiB/s, done.
Resolving deltas: 100% (1280/1280), done.
[root@parrot]~[/home]
[root@parrot]# sudo chmod 775 avml
[root@parrot]~[/home]
[root@parrot]# ll
total 0
-rwxrwxr-x 1 root root 214 Dec 4 14:32 avml
-rwxrwxr-x 1 izu izu 1.5K Dec 4 14:30 izu
-rw-r--r-- 1 root root 56 Nov 27 23:07 myenv
-rw-r--r-- 1 root root 0 Nov 27 21:01 test.txt
[root@parrot]~[/home]
[root@parrot]# ./avml memory.dump
```

./ indique que l'on exécute un fichier dans le répertoire courant.

avml est le nom de l'exécutable, qui a été soit téléchargé sous forme de binaire précompilé, soit compilé à partir du code source (écrit en Rust).

L'un des principaux avantages de Volatility est qu'il ne nécessite pas d'accès au système d'exploitation ou aux fichiers de disque du système cible, car il fonctionne directement sur l'image mémoire capturée.

**ATTENTION !** Pendant l'acquisition, évite d'effectuer des actions sur la machine qui pourraient altérer les données de la mémoire ou interférer avec l'extraction.

## 2-ANALYSE DE LA MEMOIRE VOLATILE (RAM) :

**Volatility** est le framework le plus utilisé au monde pour extraire des « artifacts »(empreintes ou traces numériques) à partir d'échantillons de mémoire volatile (RAM).

L'un des principaux avantages de Volatility est qu'il ne nécessite pas d'accès au système d'exploitation ou aux fichiers de disque du système cible, car il fonctionne directement sur l'image mémoire capturée.

Volatility prend en charge divers formats de mémoire et de plateformes, ce qui en fait un outil polyvalent pour effectuer des analyses forensiques sur des systèmes Windows, Linux et macOS.

Deux versions principales : **Volatility 2** et **Volatility 3**.

Volatility 3 est plus modulaire et performante. +plugins

**Les éléments qu'on doit avoir pour analyser avec volatility3 :**

1 – **memory dump** .

2– **plugins** : un module qui exécute une tâche particulière pour analyser une image mémoire et en extraire des informations pertinentes.

Un plugin peut être utilisé pour lister les processus , rechercher des fichiers ouverts en mémoire , ou encore extraire des connexions réseau actives .

**!! Note : Les plugins dépendent de la mémoire capturée et non pas le système d'exploitation utilisé pour analyser.**

Exemple de plugins : Windows.pslist , Linux.netstat , mac.filescan ....

- La configuration de volatility3 :

### 1-Pré-requis :

Volatility3 nécessite Python 3.7 ou une version plus récente. Vous pouvez vérifier votre version de Python avec la commande :

```
python3 --version
```

Si vous n'avez pas la bonne version :

```
sudo apt-get install python3-pip python3-dev libpython3-dev
```

**pip** est un gestionnaire de paquets pour Python. Il permet d'installer des bibliothèques et des modules Python à partir du Python Package Index (PyPI). **python3-pip** est la version de pip pour Python 3.

Ce paquet fournit les en-têtes et fichiers nécessaires pour développer des extensions Python. Il est souvent requis pour compiler et installer certains paquets Python qui nécessitent des bibliothèques C sous-jacentes.

Ce paquet contient les fichiers nécessaires pour compiler des modules d'extension Python en C pour Python 3. Il inclut des bibliothèques et des en-têtes de développement essentiels.

**vol.py** est généralement un fichier qui fait partie de Volatility, un framework de mémoire volatile largement utilisé pour effectuer des analyses forensiques sur des images de mémoire (RAM) extraites. Ce fichier est un script Python qui permet d'interagir avec l'outil Volatility pour extraire et analyser des artefacts numériques provenant de la mémoire volatile d'un système compromis.

Exécution de programmes Python : Vous pouvez exécuter des scripts Python 3 en utilisant la commande **python3** dans un terminal ou une ligne de commande.

## 2- Télécharger Volatility3 :

```
git clone https://github.com/volatilityfoundation/volatility3.git cd  
./volatility3/volatility3
```

Après , on doit installer les tableaux de symboles (extraire ce fichier dans le dossier ./volatility3/volatility3/symbols)

Les tables de symboles pour différents systèmes d'exploitation sont souvent nécessaires lors de l'analyse de la mémoire ou des enquêtes forensiques. Ces tables fournissent les mappages nécessaires entre les adresses mémoire et les symboles utilisés par le noyau, ce qui est crucial lors de l'analyse des vidages de mémoire ou de la mémoire en direct pour identifier des objets, des fonctions et d'autres structures de données importantes.

**volatilityfoundation / volatility3** (Public)

**About**

Volatility 3.0 development

[volatilityfoundation.org/](https://volatilityfoundation.org/)

- python
- ram
- memory
- incident-response
- malware
- forensics
- volatility
- volatility-framework
- digital-investigation

Readme

View license

Cite this repository

Activity

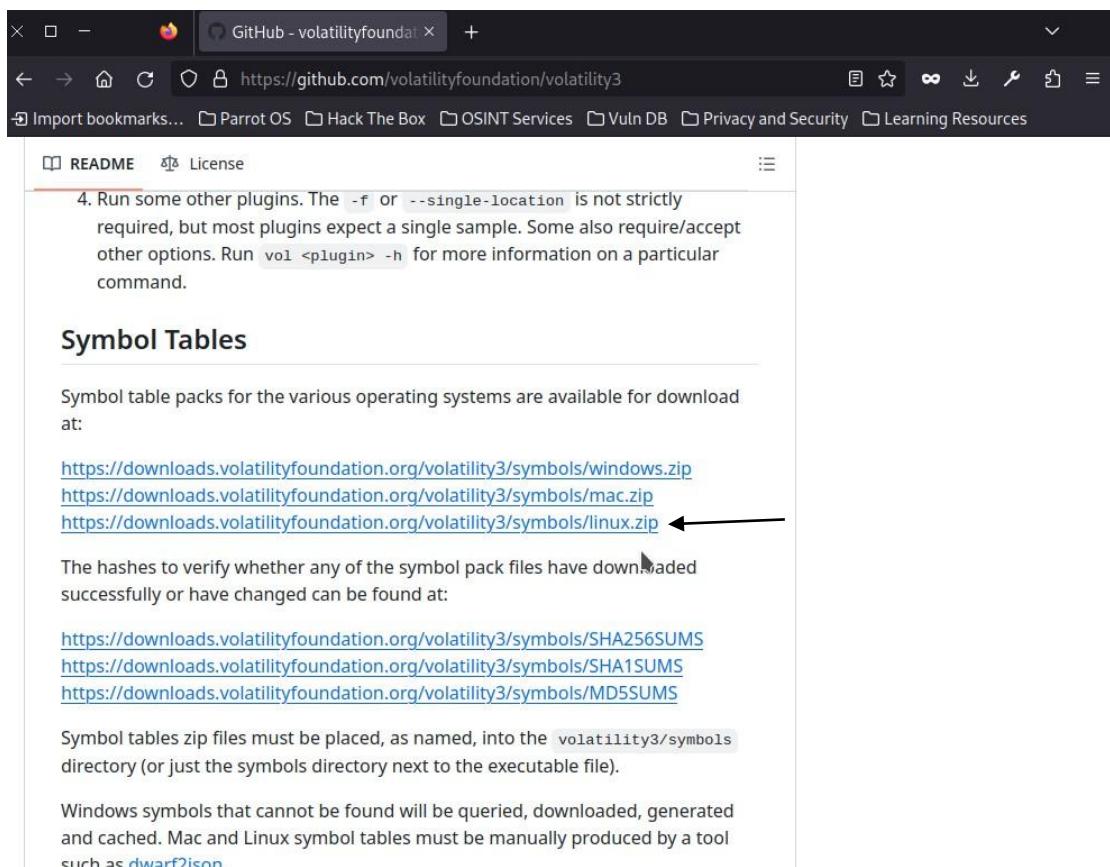
Custom properties

2.7k stars

58 watching

463 forks

```
[root@parrot]~[~/home]
└─#git clone https://github.com/volatilityfoundation/volatility3.git
Cloning into 'volatility3'...
remote: Enumerating objects: 37862, done.
remote: Counting objects: 100% (5523/5523), done.
remote: Compressing objects: 100% (939/939), done.
remote: Total 37862 (delta 5056), reused 4772 (delta 4581), pack-reused 32339 (from 1)
Receiving objects: 100% (37862/37862), 7.39 MiB | 1.02 MiB/s, done.
Resolving deltas: 100% (29053/29053), done.
[root@parrot]~[~/home]
└─#ls
avml izu myenv test.txt volatility3
[root@parrot]~[~/home]
└─#cd ./volatility3
[root@parrot]~[~/home/volatility3]
└─#ls
API_CHANGES.md development LICENSE.txt pyproject.toml test vol.py volshell.spec
CITATION.cff doc MANIFEST.in README.md volatility3 volshell.py vol.spec
[root@parrot]~[~/home/volatility3]
└─#cd ./volatility3
[root@parrot]~[~/home/volatility3/volatility3]
└─#ls
cli framework __init__.py plugins schemas symbols
```



4. Run some other plugins. The `-f` or `--single-location` is not strictly required, but most plugins expect a single sample. Some also require/accept other options. Run `vol <plugin> -h` for more information on a particular command.

## Symbol Tables

Symbol table packs for the various operating systems are available for download at:

<https://downloads.volatilityfoundation.org/volatility3/symbols/windows.zip>  
<https://downloads.volatilityfoundation.org/volatility3/symbols/mac.zip>  
<https://downloads.volatilityfoundation.org/volatility3/symbols/linux.zip>

The hashes to verify whether any of the symbol pack files have downloaded successfully or have changed can be found at:

<https://downloads.volatilityfoundation.org/volatility3/symbols/SHA256SUMS>  
<https://downloads.volatilityfoundation.org/volatility3/symbols/SHA1SUMS>  
<https://downloads.volatilityfoundation.org/volatility3/symbols/MD5SUMS>

Symbol tables zip files must be placed, as named, into the `volatility3/symbols` directory (or just the symbols directory next to the executable file).

Windows symbols that cannot be found will be queried, downloaded, generated and cached. Mac and Linux symbol tables must be manually produced by a tool such as [dwarf2json](#).



```
[root@parrot]~[/home/volatility3/volatility3/symbols]
└─#cd ..
[root@parrot]~[/home/volatility3/volatility3]
└─#ls
cli framework __init__.py plugins schemas symbols
[root@parrot]~[/home/volatility3/volatility3]
└─#mv '/home/izu/Downloads/linux' symbols
```

### 3. Installer les dépendances de Volatility3 :

```
python3 -m venv venv && . venv/bin/activate
pip install -e .[dev]
```

### 3-1. `python3 -m venv venv`

`python3` : Utilise la version 3 de Python. `-m` :

Exécute un module Python comme un script. `venv` : Il s'agit du module qui permet de créer des environnements virtuels en Python.

`venv` (après `-m venv`) : C'est le nom du répertoire où l'environnement virtuel sera créé. Il contiendra une installation isolée de Python, permettant d'installer des dépendances spécifiques à un projet sans affecter le système global.

Cette commande crée donc un environnement virtuel Python dans le répertoire `venv`. L'environnement virtuel permet de travailler sur un projet spécifique sans interférer avec les autres projets Python ou l'installation globale de Python.

### 3-2. `&& . venv/bin/activate`

`&&` : Cela signifie que la deuxième commande sera exécutée seulement si la première a réussi. C'est une manière de chaîner les commandes.

`. venv/bin/activate` : C'est un script qui active l'environnement virtuel Python que vous venez de créer.

Le `.` (point) ou `source` permet d'exécuter le script dans le shell actuel.

`venv/bin/activate` : Le chemin vers le fichier `activate` dans l'environnement virtuel créé. Ce script configure le terminal pour utiliser les bibliothèques Python installées dans l'environnement virtuel, au lieu des bibliothèques globales.

Après l'activation de l'environnement virtuel, le prompt du terminal changera pour indiquer que vous êtes maintenant dans l'environnement virtuel (généralement, le nom de l'environnement apparaît avant le prompt, par exemple `(venv)`).

### 3-4-. `pip install -e .[dev]`

`pip` : Le gestionnaire de paquets Python. `install` :

L'option pour installer des paquets avec `pip`.

`-e` : Cela signifie "installation en mode édition", ce qui signifie que vous installez un paquet en développement à partir du répertoire actuel. Cela vous permet de faire des modifications dans le code source du projet sans avoir à réinstaller le paquet à chaque modification.

`. :` Cela fait référence au répertoire actuel, donc cela installe le projet Python à partir du répertoire où la commande est exécutée.

[dev] : C'est une extra option qui indique à pip d'installer également les dépendances spécifiées dans un groupe dev du fichier setup.py du projet. Ces dépendances sont généralement utilisées pour le développement (par exemple, des outils de test, des linters, etc.).

Résumé :

```
python3 -m venv venv : Crée un environnement virtuel Python.
```

```
&& . venv/bin/activate : Active cet environnement virtuel.
```

```
pip install -e .[dev] : Installe le projet en mode édition avec ses dépendances de développement.
```

The screenshot shows a terminal window with a dark background and green text. It displays the following commands:

```
[root@parrot]~/home/volatility3]
ls
API_CHANGES.md development LICENSE.txt pyproject.toml test vol.py volshell.spec
CITATION.cff doc MANIFEST.in README.md volatility3 volshell.py vol.spec
[root@parrot]~/home/volatility3]
#python3 -m venv venv && . venv/bin/activate
pip install -e .[dev]
```

The terminal window is part of a desktop environment, as evidenced by the window title bar which reads "File Edit View Go Applications Help" and the desktop icons visible at the bottom.

Fin de configuration .

## PARTIE 3 :

### 1 - VERIFICATION ET TESTS :

-Pour lister les options et les plugins :

```
Python3 vol.py -h
```

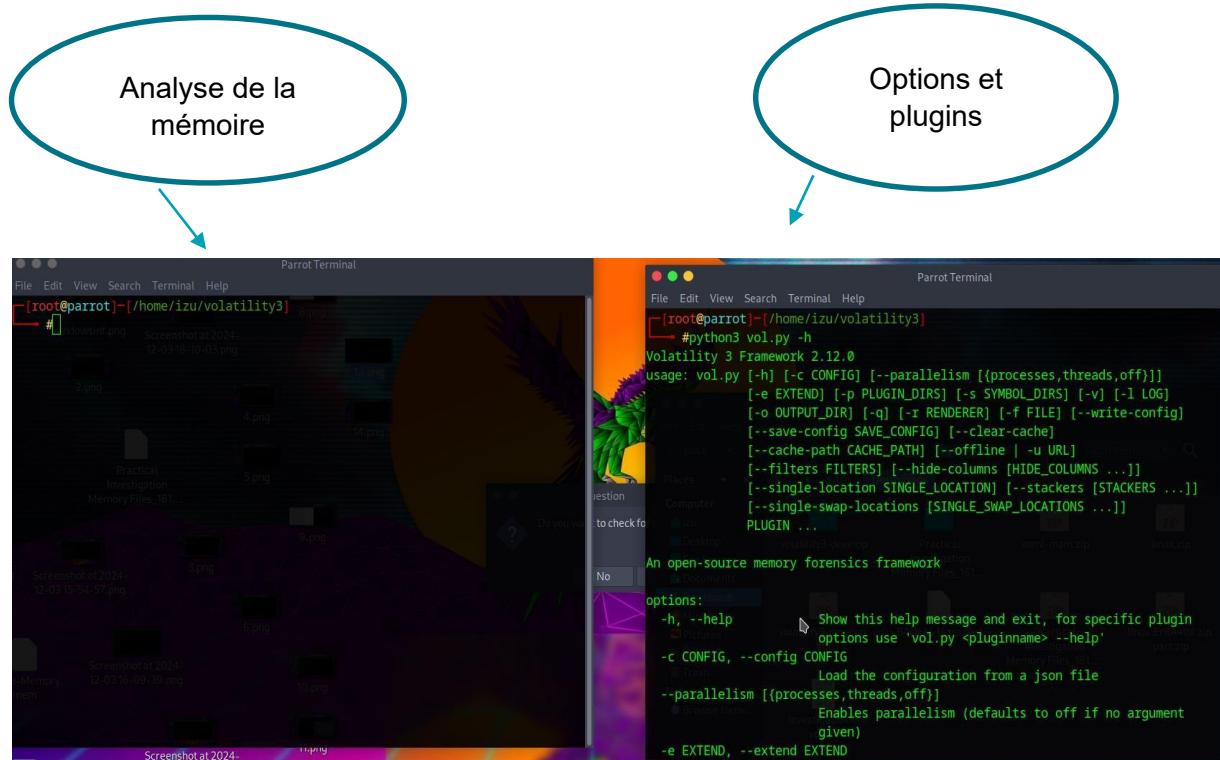
-Pour avoir des informations sur la mémoire :

```
Python3 -f memory.dump OS.info
```

OS (operation system) = Windows , linux , mac

-Pour analyser la mémoire :

Python3 vol.py -f memory.dump plugin



Lors de l'utilisation de **Volatility 3**, on commence généralement par spécifier des options essentielles pour l'analyse, telles que l'image mémoire à analyser et le profil du système. Voici quelques-unes des options courantes :

- f : Spécifie le chemin vers l'image mémoire à analyser (fichier de dump mémoire).
- p : Utilisé pour définir un profil système spécifique, comme le système d'exploitation (Windows, Linux, macOS).
- v : Active le mode verbex pour afficher des informations détaillées pendant l'exécution des commandes.

Note : Le message qui indique que Volatility a détecté l'absence des fichiers de métadonnées (comme VMSS ou VMSN), mais il a tout de même tenté l'analyse du fichier VMEM. Cela est possible car certains outils, notamment Volatility, peuvent fonctionner en mode "brut" pour extraire des informations même sans métadonnées.

Si le profil du système n'est pas détecté automatiquement, précisez-le manuellement

```
vol -f Investigation-1.vmem --profile=<nom_du_profil> windows.pslist
```

The terminal window displays the following volatility commands and their outputs:

```
[root@parrot]# python3 vol.py -f '/home/izu/Desktop/Investigation-1.vmem' windows.skeleton_key_check
Volatility 3 Framework 2.12.0
WARNING: volatility3.framework.layers.vmware: No metadata file found alongside VMEM file. A VMSS or VMSN file may be required to correctly process a VMEM file. These should be placed in the same directory with the same file name, e.g. Investigation-1.vmem and Investigation-1.vmsn.
Progress: 100.00      PDB scanning finished
PID      Process Skeleton Key Found      rc4HmacInitialize      rc4HmacDecrypt

[root@parrot]# python3 vol.py -f '/home/izu/Desktop/Investigation-1.vmem' windows.pslist
Volatility 3 Framework 2.12.0
WARNING: volatility3.framework.layers.vmware: No metadata file found alongside VMEM file. A VMSS or VMSN file may be required to correctly process a VMEM file. These should be placed in the same directory with the same file name, e.g. Investigation-1.vmem and Investigation-1.vmsn.
Progress: 100.00      PDB scanning finished
Offset PID      TID      StartAddress      CreateTime      ExitTime
```

On the right side of the terminal, there is a large list of volatility modules with their descriptions:

- windows.registry.hivelist.HiveList**: Lists the registry hives present in a particular memory image.
- windows.registry.hivescan.HiveScan**: Scans for registry hives present in a particular windows memory image.
- windows.registry.printkey.PrintKey**: Lists the registry keys under a hive or specific key value.
- windows.registry.userassist.UserAssist**: Prints userassist registry keys and information.
- windows.scheduled\_tasks.ScheduledTasks**: Decodes scheduled task information from the Windows registry, including information about triggers, actions, run times, and creation times.
- windows.sessions.Sessions**: Lists processes with session information extracted from Environmental Variables.
- windows.shimcacheMem.ShimcacheMem**: Reads Shimcache entries from the abcache.sys AVL tree.
- windows.skeleton\_key\_check.Skeleton\_Key\_Check**: Looks for signs of Skeleton Key malware.
- windows.ssdt.SSDT**: Lists the system call table.
- windows.statistics.Statistics**: Lists statistics about the memory space.
- windows.strings.Strings**: Reads output from the strings command and indicates which process(es) each string belongs to.
- windows.suspicious\_threads.SuspiciousThreads**: Lists suspicious userland process threads.
- windows.svcdiff.SvcDiff**: Compares services found through list walking versus scanning to find rootkits.
- windows.svclist.Svclist**: Lists services contained with the services.exe double-linked list of services.
- windows.svscan.SvcScan**: Scans for windows services.
- windows.symlinkscan.SymlinkScan**: Scans for links present in a particular windows memory image.
- windows.thrdscan.ThrdScan**: Scans for windows threads.
- windows.threads.Threads**: Lists process threads.
- windows.timers.Timers**: Prints kernel timers and associated module DPCs.
- windows.truecrypt.Passphrase**:

```
[root@parrot:~/home/izu/volatility3]
└─# python3 vol.py -f '/home/izu/Desktop/Investigation-1.vmem' windows.info
WARNING: volatility3.framework.layers.vmware: No metadata file found alongside VMEM file. A VMSS or VMSN file may be required to correctly process a VMEM file. These should be in the same directory with the same file name, e.g. Investigation-1.vmem and Investigation-1.vmss.
Progress: 100.00          PDB scanning finished
Variable      Value
Kernel Base    0x804d7000
DTB     0xfee000
Symbols file:///home/izu/volatility3/volatility3/symbols/windows/ntkrnlpa.pdb/305FB31AE7E4ACAABA750AA241FF331-1.json.xz
is64Bit False
IsPAE True
layer_name    0 WindowsIntelPAE
memory_layer   1 FileLayer
KdDebuggerDataBlock 0x80545ae0
NTBuildLab    2600.xpsp.080413-2111
CSOVersion    3
KdVersionBlock 0x80545ab8
Major/Minor    15.2600
MachineType   332
KdVersionBlock 0x80545ab8
PE MajorOperatingSystemVersion 5
PE MinorOperatingSystemVersion 1
PE Machine    332
PE TimeStamp   Sun Apr 13 18:31:06 2008
[root@parrot:~/home/izu/volatility3]
```

**NOTE :** L'analyse de la mémoire se fait sur windows comme linux , mais l'extraction de la mémoire est différente .

## 2 - INVESTIGATION PRATIQUE :

Votre SOC vous a informé qu'ils ont récupéré un dump mémoire , suspecté d'avoir été compromis par un « trojan Horse Malware » bancaire se faisant passer pour un document Adobe. Votre mission est d'utiliser vos connaissances en renseignement sur les menaces et en ingénierie inverse pour effectuer une analyse forensique de la mémoire sur l'hôte infecté.

**Les chevaux de Troie sont des programmes qui prétendent remplir une fonction mais en remplissent en réalité une autre, généralement malveillante**

Ils ont souvent utilisés pour :

Voler des informations personnelles (comme des identifiants de connexion ou des informations bancaires).

Installer d'autres malwares sur le système, comme des ransomwares ou des logiciels espions.

Permettre l'accès à distance à un ordinateur ou un réseau pour mener des actions malveillantes, telles que l'exfiltration de données ou le contrôle du système.

(mémoire = Investigation-1.vmem) qui est déjà extractée .

**Questions :**

Quelle est la version de la machine hôte ?

À quelle heure le fichier mémoire a-t-il été acquis?

Quel processus peut être considéré comme suspect?

Quel est le processus parent du processus suspect?

Quel est le PID du processus suspect ?

Quel **user-agent** a été utilisé par l'adversaire ?

La banque Chase faisait-elle partie des domaines suspects trouvés ?

### Python3 –f memory.dump windows.info

```
File Edit View Search Terminal Help
[root@parrot]:~/home/izu/volatility3]
#python3 vol.py -f '/home/izu/Desktop/Investigation-1.vmem' windows.info
Volatility 3 Framework 2.12.0
WARNING: volatility.framework.layers.vmware: No metadata file found alongside VMEM file. A VMSS or VMSN file may be required to correctly process a VMEM file. These should be in the same directory with the same file name, e.g. Investigation-1.vmem and Investigation-1.vmss.
Progress: 100.00          PDB scanning finished
Variable      Value
Kernel Base    0x804d7000
DTB     0x2fe000
Symbols file:///home/izu/volatility3/volatility3/symbols/windows/ntkrnlpa.pdb/30B5FB31AE7E4ACAA8A750AA241FF331-1.json.xz
Is64Bit False
IsPAE  True
layer_name     0 WindowsIntelPAE
memory_layer   1 FileLayer
KdDebuggerDataBlock  0x80545ae0
NTBuildLab    2600.xpsp.080413-2111
CSVersion     3
KdVersionBlock 0x80545ab8
Major/Minor    15.2600
MachineType   332
KeNumberProcessors 1
SystemTime    2012-07-22 02:45:00+00:00
NtSystemRoot  C:\WINDOWS
NtProductType NtProductInNt
NtMajorVersion 5
NtMinorVersion 1
PE MajorOperatingSystemVersion 5
PE MinorOperatingSystemVersion 1
PE Machine    332
PE TimeStamp    Sun Apr 13 18:31:06 2008
[...]
[root@parrot]:~/home/izu/volatility3]
```

2600.xpsp.080413-2111

2012-07-22 02:45:08

### Python3 –f memory.dump windows.psscan

```

File Edit View Search Terminal Help
[root@parrot]~/home/izu/volatility3]
#python3 vol.py -f '/home/izu/Desktop/Investigation-1.vmem' windows.psscan
Volatility 3 Framework 2.12.0
WARNING volatility3.framework.layers.vmware: No metadata file found alongside VMEM file. A VMSS or VMSN file may be required to correctly process a VMEM file. These should be placed in the same directory with the same file name, e.g. Investigation-1.vmem and Investigation-1.vmss.
Progress: 100.00          PDB scanning finished
PID  Proc  PPID  ImageFileName  Offset(V)  Threads Handles SessionId  Wow64  CreateTime      ExitTime      File output
1640  svhost.exe  0x2029ab8  9    226  0    False  2012-07-22 02:42:33.000000 UTC  N/A  Disabled
664   lsass.exe   0x202a3b8  24   330  0    False  2012-07-22 02:42:32.000000 UTC  N/A  Disabled
652   services.exe 0x202ab28  16   243  0    False  2012-07-22 02:42:32.000000 UTC  N/A  Disabled
1484  reader_sl.exe 0x207bda0  5    39   0    False  2012-07-22 02:42:36.000000 UTC  N/A  Disabled
1512  spoolsv.exe  0x20b17b8  14   113  0    False  2012-07-22 02:42:36.000000 UTC  N/A  Disabled
1588  wuauctl.exe 0x225bda0  5    132  0    False  2012-07-22 02:44:01.000000 UTC  N/A  Disabled
788   alg.exe     0x22e8da0  7    104  0    False  2012-07-22 02:43:01.000000 UTC  N/A  Disabled
1484  explorer.exe 0x23dea70  17   415  0    False  2012-07-22 02:42:36.000000 UTC  N/A  Disabled
1056  svhost.exe   0x23fdfa0  5    60   0    False  2012-07-22 02:42:33.000000 UTC  N/A  Disabled
1136  wuauctl.exe 0x23fcfa0  8    173  0    False  2012-07-22 02:43:46.000000 UTC  N/A  Disabled
1220  svhost.exe   0x2495650  15   197  0    False  2012-07-22 02:42:35.000000 UTC  N/A  Disabled
608   winlogon.exe 0x2498700  23   519  0    False  2012-07-22 02:42:32.000000 UTC  N/A  Disabled
584   csrss.exe    0x24a0598  9    326  0    False  2012-07-22 02:42:32.000000 UTC  N/A  Disabled
368   smss.exe     0x24f1020  3    19   N/A  False  2012-07-22 02:42:31.000000 UTC  N/A  Disabled
1004  svhost.exe   0x25001d0  64   1118 0    False  2012-07-22 02:42:33.000000 UTC  N/A  Disabled
824   svhost.exe   0x2511360  20   194  0    False  2012-07-22 02:42:33.000000 UTC  N/A  Disabled
4    System        0x25c89c8  53   240  N/A  False  N/A   Disabled

```

reader\_sl.exe

1640

1484

python3 vol.py -f /home/izu/Desktop/Investigation-1.vmem -o /home/izu windows.memmap -pid 1640 --dump

```

File Edit View Search Terminal Help
[root@parrot]~/home/izu/volatility3]
#python3 vol.py -f '/home/izu/Desktop/Investigation-1.vmem' -o /home/izu windows.memmap --pid 1640 --dump
Volatility 3 Framework 2.12.0
WARNING volatility3.framework.layers.vmware: No metadata file found alongside VMEM file. A VMSS or VMSN file may be required to correctly process a VMEM file. These should be placed in the same directory with the same file name, e.g. Investigation-1.vmem and Investigation-1.vmss.
Progress: 100.00          PDB scanning finished
Virtual Physical      Size  Offset in File  File output
0x3d00000 0xd7fa000 0x1000 0x0 pid.1640.dmp
0x3d1000 0xd87b000 0x1000 0x1000 pid.1640.dmp
0x3d2000 0xd6fc000 0x1000 0x2000 pid.1640.dmp
0x3d3000 0xd77d000 0x2000 0x3000 pid.1640.dmp
0x3d5000 0xd67f000 0x1000 0x5000 pid.1640.dmp
0x3d6000 0xdb00000 0x1000 0x6000 pid.1640.dmp
0x3d7000 0xd781000 0x1000 0x7000 pid.1640.dmp
0x3d8000 0xd7c2000 0x1000 0x8000 pid.1640.dmp
0x3d9000 0xd843000 0x1000 0x9000 pid.1640.dmp
0x3da000 0xd744000 0x1000 0xa000 pid.1640.dmp
0x3db000 0xd785000 0x1000 0xb000 pid.1640.dmp
0x3dc000 0xd706000 0x1000 0xc000 pid.1640.dmp
0x3dd000 0xd807000 0x1000 0xd000 pid.1640.dmp
0x3de000 0xd748000 0x1000 0xe000 pid.1640.dmp
0x3df000 0xd809000 0x1000 0xf000 pid.1640.dmp
0x3e0000 0xd810000 0x1000 0x1000 pid.1640.dmp

```

**--dump : Dumping Memory Regions in Volatility :** Lorsque **Volatility** effectue une analyse de la mémoire, il peut localiser des régions spécifiques de mémoire (par exemple, des parties de la mémoire d'un processus, de la mémoire noyau, ou de la mémoire DLL) et "dump" ces régions dans des fichiers séparés pour une analyse détaillée.

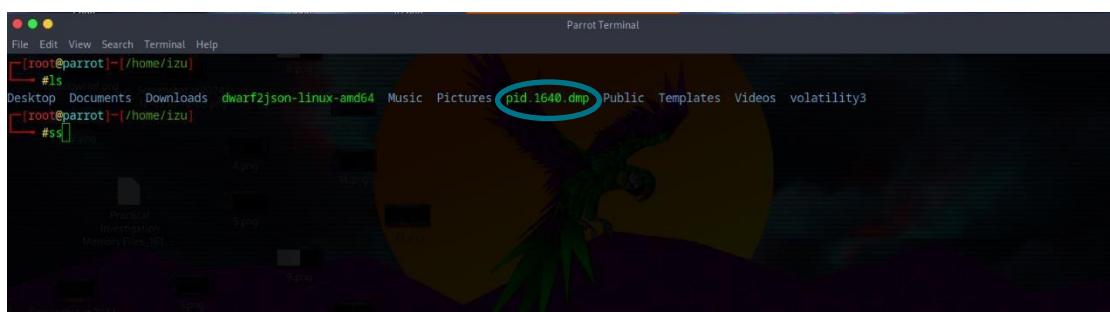
Ainsi, le "dump" dans **Volatility** signifie l'extraction et l'enregistrement de données spécifiques de l'image mémoire dans des fichiers pouvant être analysés isolément.

-o : L'emplacement de l'output .

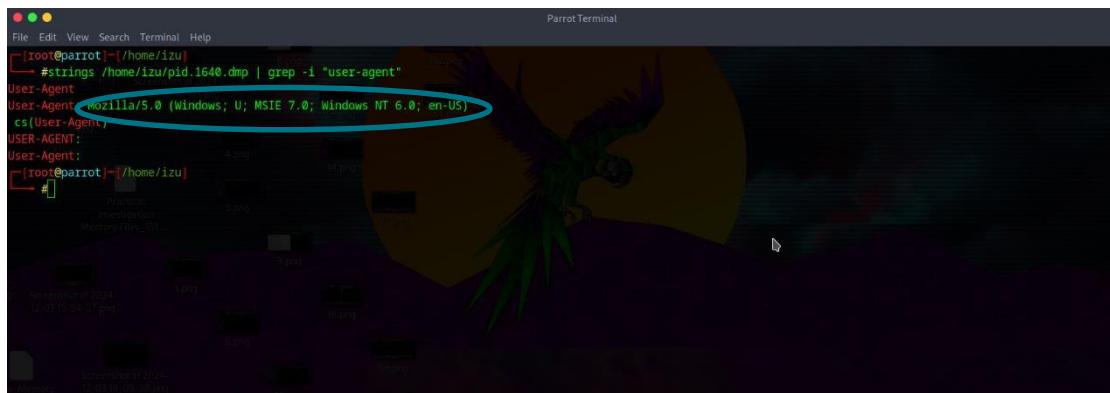
Windows.memmap : extraire et afficher la cartographie de la mémoire d'un processus ou d'un système entier .

Il fournit des informations détaillées sur les différentes régions de mémoire, y compris les adresses, tailles et types de mémoire.

L'option **--pid** dans **Volatility** est utilisée pour spécifier un **identifiant de processus (PID)**, ce qui permet de cibler l'analyse de la mémoire pour un processus particulier ;



Strings /home/izu/pid.1640.dmp | grep -i user-agent



La commande **strings** est un utilitaire utilisé pour extraire du texte lisible par l'humain à partir de fichiers binaires, y compris des dumps mémoire .

Un **user-agent** est une chaîne de texte qu'un navigateur web ou une application envoie à un serveur web pour s'identifier. Cette chaîne contient généralement des informations sur le navigateur, sa version, le système d'exploitation et parfois le type de dispositif ou l'application effectuant la requête.

**-i** permet donc de ne pas tenir compte de la différence entre les lettres majuscules et minuscules lors de la recherche.

```
Strings /home/izu/pid.1640.dmp | grep -i chase
```

On peut déduire que l'attaquant ait envoyé à la victime un lien vers une fausse page bancaire où la victime a téléchargé le document depuis le domaine !!!!!

**FIN .**