Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO TUTELADO SSI
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Memory Forensics

A Coruña, decembro de 2020.

Resumo

No seguinte documento vaise detallar o funcionamento e os principais conceptos teóricos do método ***Memory forensics*** centrándonos no sistema operativo **Windows**. Tamén se probará a ferramenta *Volatility* co fin de entender os conceptos teóricos vistos na primera parte do documento.

A maiores incluimos unha serie de apéndices vendo diferentes aspectos como son a proba dunha ferramenta de análise forense, a instalación de *Volatility* en **Ubuntu** e un caso de estudio para extraer a información dun disco cifrado mediante **bitlocker**.

Apoiámonos nas ideas dos libros *The art of Memory Forensics* [1] e *Practical Malware Analysis* [2].

Índice Xeral

Índice de Figuras

Capítulo 1

Introdución

1.1 Memory Forensics

O término de *Memory Forensics* correspondese ao estudo ou investigación de ataques levados a cabo nun ordenador os cales solo deixan o seu rastro na memoria volatil (RAM) do noso ordenador.

Calquera software que faga algo que cause dano a un usuario, a computadora ou a rede pode considerarse malware. O análise de este é o arte de diseccionar malware para comprender como funciona, como identificalo e como evitalo ou eliminalo.

Cada función realizada por un sistema operativo ou aplicación deixa a su huella na memoria do ordenador, como que procesos se estaban executando, conexións de rede abertas e comandos executados recentemente. Pois para executar calquera programa, primeiro debe cargarse en memoria, o que fai que sexa fundamental para os forenses identificar o distintos ataques.

1.2 Diferentes técnicas

Cando se realiza un análise dun *malware* moitas veces solo teremos o *malware executable* polo que non será moi lexible para ás persoas. Para darlle sentido podemos empregar un serie de ferramentas. Hay dous enfoques posibles para esté análise: ***Análise dinámico*** e ***Análise estático***.

1.2.1 Análise estático

Este tipo de análise implica examinar o *malware* pero sen chegar a executalo. Dentro deste podemos diferenciar:

- *Análise estático básico*: Neste tipo de análise solo podremos confirmar se un archivo é

malicioso e proporcionar información sobre a sua funcionalidade. Este tipo de análise é sencillo e rápido, pero a veces pode ser ineficaz pois pode pasar por alto comportamentos de software máis sotisficados.

- *Análise estático avanzado*: Consiste en aplicar inxeñeria inversa (*reverse-engineering*)[3] aos compoñentes internos do *malware*. É decir cargando o executable nun desensamblador e mirando as instruccións do programa para descubrir cal é a súa función. As instruccións son executadas polo **CPU** polo que o *análise estático básico* dinos exactamente que fai o programa.

1.2.2 Análise dinámico

A difencia do anterior neste caso si que se vai executar o *malware* para comprender o seu funcionamento.

- *Análise dinámico básico*: Implica executar o *malware* e observar o seu comportamento no sistema para así eliminar a infección, producir firmas efectivas ou ambas. Sin embargo, antes de executar o *malware* de forma segura debemos de configurar un entorno que permita estudar o funcionamento deste, como é o caso das coñecidas *sandboxes*[4], sen dañar o noso sistema.

- *Análise dinámico avanzado*: Emprega un depurador para examinar o resultado interno dun executable que está correndo. Este tipo de técnica é moi útil cando se intenta obter información que é dificil de conseguir coas outras técnicas.

1.3 Clasificación segundo a sua volatilidade

Entendemos por volatilidade dos datos, a certa información que ten a propiedade de cambiar e transformarse de forma constante no tempo e é pouco predecible, incluso poidendo chegar a desaparecer. Distinguimos entón dous tipos:

1. **Volátiles**: Son aqueles datos que se perderán ou se modificarán si se apaga ou se reinicia o sistema. Son por exemplo rexistros e contidos da caché, contidos da memoria, estado das conexións da rede, tablas de rutas e estado dos procesos en execución.

2. **Non volátiles**: Estes datos a diferencia dos anteriores persisten no tempo independentemente de se apagamos ou reniciamos a máquina, son por exemplo o contido do sistema de arquivos e dos discos duros, metadatos dos arquivos e contido de outros dispositivos de almacenamento.

1.4 Análise en quente e analise en frío

Podemos diferenciar dous tipos de análise a hora de chegar a unha escena.

1.4.1 Equipo encendido, análise en quente

Co equipo que se quere analizar encendido pódense obter datos e evidencias volátiles que non sería posible obter se o estado deste fora o de apagado. A principal vantaxe deste tipo de análise radica na posibilidade de atopar evidencias de ataques que se encontran activos nese momento, datos de moita importancia para poder chegar a identificar a orixe do ataque.

Aquí nace a importancia de realizar volcados ou copias da memoria do sistema como veremos máis adiante, **Memory Acquisition**(2(.
Cabe destacar tamén a importancia neste tipo de análise de recoller información do estado da rede. Este análise da rede tense que facer en primeiro lugar co fin de desconectalo da mesma o mais rápido posible para evitar unha perda de información debido a unha actuación remota dende fora do noso ordenador.

1.4.2 Equipo apagado, análise en frío

A diferencia do análise en quente este tipo de análise pode chegar a realizarse dunha forma máis profunda pero co incoveniente de que podemos perder ou deixar de obter probas e recursos de tipos volátil.

Realízase partindo dunha copia exacta dun disco estraída a partires dun equipo en estado apagado. Trátase dun tipo de análise menos intrusivo que o anterior e evitando modificar a información que contén o disco traballando sobre unha copia exacta do mesmo. Isto pódese realizar mediante hardware (clonadoras) ou mediante software.

Capítulo 2

Adquisición da memoria

A adquisición da memoria (captura, volcado, muestreo) implica copiar o contido da memoria volátil a un almacenamento non volátil. Este é un dos pasos máis importantes pois é onde imos coller toda a información que a posterior imos a analizar, e é un dos primeiros pasos que temos que realizar para o proceso forense da memoria.

2.1 Qué é a memoria volatil

Os datos volátiles son todos os datos que se almacenan temporalmente nun dispositivo informático mentras se está a executar e se o dispositivo se apaga estes se perderían.

Existen archivos temporalmente de caché, RAM e archivos do sistema. Por exemplo, si se está traballando calqueira archivo de texto sen guardalo na memoria persistente do ordenador, existe a posibilidade de perder o archivo no caso de que o sistema de cerre. Os datos volátiles conterían as últimas accións non gardadas realizadas no documento.

2.2 Visión xeral

Para adquirir esa memoria pódense empregar unha serie de ferramentas e unha serie de capacidades para adaptar o uso de estas técnicas dependendo de cada caso e dos entornos no que te atopas. A seguinte figura 2.1 amosanos un árbore de decisión moi simple baseado na maioria dos casos que nos podemos atopar a hora de volcar a memoria.

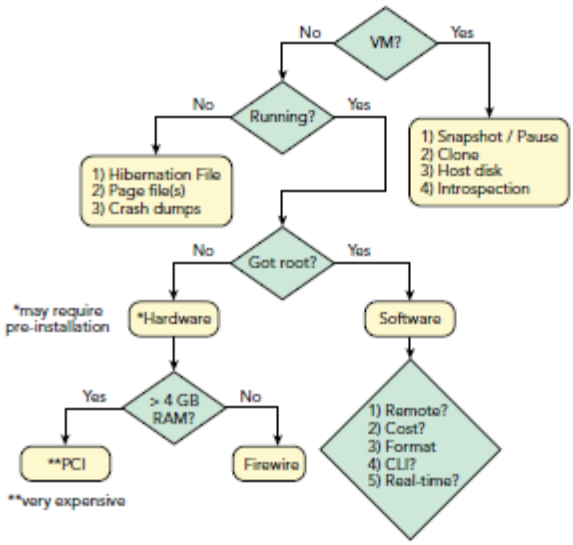


Figura 2.1: Diagrama Memory Acquisition

Como podemos ver, o primeiro que temos que ter en conta é si nos atopamos nunha máquina virtual, pois de ser así, esta xa nos proporciona varias ferramentas. O **hipervisor** [5] permitenos pausar, suspender, tomar unha instantánea ou usar unha introspección.

En cambio se o sistema é un ordenador común ou un servidor debemos determinar primeiro de nada se o programa está correndo. Se non o está, o estado actual da memoria non é volatil, pero cabe a posibilidade de que os datos recentes se escribiran en dispositivos de almacenamento como o disco duro. En cambio se o sistema atópase en execución temos a oportunidade de obter o estado actual da memoria volátil, pero para isto necesitarase permisos de adminitrador *root*. Nestes casos podemos empregar unha utilidade basada en software, do contrario, as opcións non son tan sinxelas.

Para adquisición da memoria mediante hardware esta faise mediante **DMA** (Direct Memory Acces)[6] mediante unha teconoloxía como *Firewire*, *Thunderbolt*, *ExpressCard* ou *PCI*. A principal desventaxa deste método é que a máquina destino ten que estar equipada cun dispositivo deste tipo ou que admita dispostivos de intercambio en quente, pois senón débese apagar o equipo para instalar os adaptadores hardware requeridos destruindo así a memoria volatil. Ademais desto ten limitacións de tamaño pois *Firewire* solo pode coller os 4 primeiros *GB* da memoria RAM e por último que os dispositivos *PCI*[7] son extremadamente raros, polo que tamén son bastante costosos.

2.3 Memory Dump Formats

Existen diferentes tipos de formatos a hora de recoller a información da memoria:

1. ***Raw Memory Dump***: O volcado de memoria sen formato é o formato máis empregado entre as ferramentas de análise. Ten como característica principal que non contén encabezados, metadatos ou valores máxicos para a identificación do tipo de arquivo.
   Moitas veces rechee calquera rango de memoria que se omitiu intencionadamente ou que a ferramenta de adquisición de memoria non foi capaz de leer, provocando así unha capacidade de ter unha boa integridade espacial.

2. ***Windows Crash Dump***: Este tipo solo é propio de **Windows**. **Windows** contén unha función de depuración do kernel e aplicacións que se poden empregar para xerar *crash-dumps* e que foron diseñados para propositos de debugueado.
   Son similares aos anteriores, pois estes non se encontran comprimidos e sempre van a empezar por as estructuras *_DMP_HEADER* ou *_DMP_HEADER64*.

3. ***Windows Hibernation File***: Tamén solo é propia de **Windows** e sempre teñen o seguinte nome *hiberfil.sys*.
   Este arquivo conten unha copia comprimida da memoria que o sistema volcou no disco durante o proceso de hibernación. Este formato presenta unha gran desventaxa, pois antes do proceso de hibernación, a configuración se existe en *DHCP* [8] liberase e calquera das conexións que estivesen activas rematanse, polo que os datos da rede poden estar incompletos e resultar así incorrecto o seu posterior análise.

4. ***Virtual Machine Memory***: As máquinas virtuais *VMware* e outras tecnoloxías de virtualización teñen unha función para facer instantáneas dunha instacia dunha máquina virtual. Cando esto ocurre, todo o estado da execución do sistema gárdase na máquina *host*, incluída unha instantánea da *RAM* (que se empregará de novo ao reanudar a *VM*). Normalmente en *VMware* toda a memoria da *VM* gárdase íntegra nun arquivo *.vmem*.

5. ***Pagefiles***: Tamén unicamente para sistemas **Windows**, aunque non é unha copia da *RAM*, asemellase moito, pois **Windows** emprega o arquivo *pagefile.sys* no disco como espacio de intercambio ou para almacenar temporalmente algunhas páxinas de memoria cando un sistema asigna máis memoria que a que cabe na memoria *RAM* física.

   Windows pode chegar a ter ata 16 *page files* diferentes polo que a veces poder ser complicado atopar a que nos interese. Tamén o análise de estes *page files* non é tan sinxelo coma os anteriores, pois estes son como unha especie de rompecabezas que complica o seu análise.

2.4 Convertir Memory Dumps

Como ben enumeramos antes, existen diferentes tipos de *Memory Dumps* que van ser empregados para os diferentes *frameworks* de análise. A maioría destes programas só van a admitir un ou dous tipos de arquivos polo que a veces nos vemos na obriga de convertir o ficheiro a outro formato, na maioría das veces ao formato *RAW*.

Existen diferentes ferramentas para facer estes cambios, nós non imos necesitar empregar ningunha delas, pois o framework *Volatility* pode admitir calqueira tipo de *Memory Dump*. Ainda así algunhas das máis comúns son *MoonSols Windows Memory Toolkit, VMware vmss2core, Volatility raw2dmp,...*.

2.5 Ferramentas software

Todas as ferramentas de aquisición baseadas en software siguen un protocolo similar para adquirir a memoria. Estas ferramentas funcionan cargando un módulo do kernel que mapea a dirección física no espazo de direccions virtual dunha tarea que se está a executar no sistema. Neste punto poden acceder aos datos dende ese espazo de direccións virtuais e escribilos no almacenamento non volatil seleccionado.

2.5.1 Listaxe de ferramentas

Existen unha gran variedade de ferramentas, pero nós solo imos probar o funcionamento de algunhas delas, pois todas operan dunha maneria parecida. Neste apartado solo se enumeran, mentres que as probas estarán a continuación 2.6.

1. *Magnet RAM Capture* [9]: É unha ferramenta gratuita diseña para capturar a memoria física do ordenador en formato **RAW** (*.dmp,.raw ou .bin*). Os sistemas operativos compatibles son **Windows XP, Vista, 7, 8, 10, 2003, 2008, 2012 (32 e 64 bits)**.

   Ten unha opción bastante boa, pois é que si nós estamos correndo a ferramenta nun USB formateado con *FAT32* [10] e o tamaño de memoria *RAM* que estamos capturando é maior que **4GB** podemos usar a opción de *segmentation feature*.

2. *Belkasoft Live RAM Capturer* [11]: Esta ferramenta ten como principal atractivo de que é capaz de volcar a memoria cando incluso existen mecanismos agresivos anti-depuración e anti-dumping.
   É compatible con **Windows XP, Vista, Windows 7 e 8, 2003 e 2008 Server**.

3. *MoonSols DumpIt* [12]: É unha fusión de *win32dd* e *win64dd* combinados nun executable. Con el pódese obter un volcado de memoria *RAM* con formato *RAW*. A versión

gratuita ten unhas funcionalidades moi básicas pero se queremos cambiar o tipo de formato de saída ou habilitar a encriptación mediante *RC4* debemos comprar a versión de pago.

2.6 Proba das ferramentas

Vamos a probar as ferramentas sobre o sistema operativo **Windows** e en concreto a versión **Windows 7**, que se instalou nunha máquina virtual de **4GB** de *RAM*, **32GB** de disco duro e un adaptador de tipo *NAT* para a conexión a internet.

2.6.1 *Magnet RAM Capture*

Para poder descargalo simplemente acudimos a seguinte páxina na que solo fai falla cubrir un cuestionario. Unha vez descargado o executable o seu funcionamento é moi sinxelo. Simplemente debemos executalo como administrador e amosaranos unha pantalla como a seguinte.

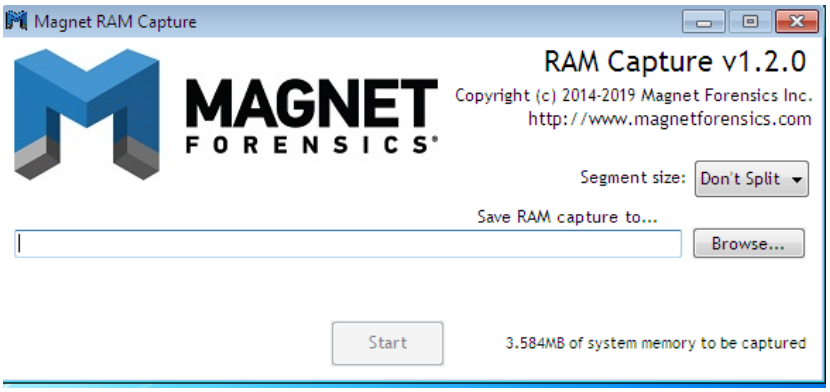


Figura 2.2: Memory Dump - Magent Ram Capture

Na que simplemente elexiremos onde queremos gardalo e se queremos a opción de segmentación como ben comentábamos anteriormente.

2.6.2 *Belkasoft Live RAM Capturer*

Descargamos a ferramenta dende o seguinte enlace. Unha vez obtido o executable é tan sinxelo de usar coma o anterior, simplemente debemos seleccionar onde queremos gardar o arquivo *.mem* e comezará un volcado da nosa memoria.

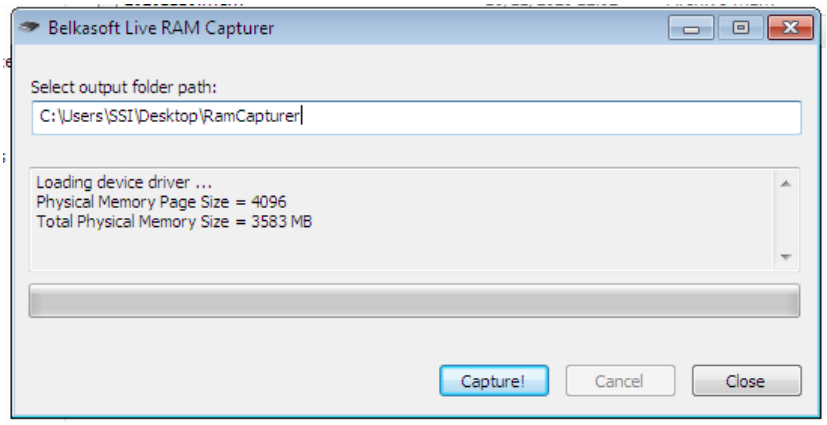


Figura 2.3: Memory Dump - Belkasoft

2.6.3 *MoonSols DumpIt*

Descargamos o executable dende o seguinte repositorio. De novo o seu funcionamento volve a ser moi básico, simplemente debemos executar o *.exe* en modo administrador e a continuación pulsar a tecla **y**, e automáticamente crearase unha copia da nosa memoria *RAM*.

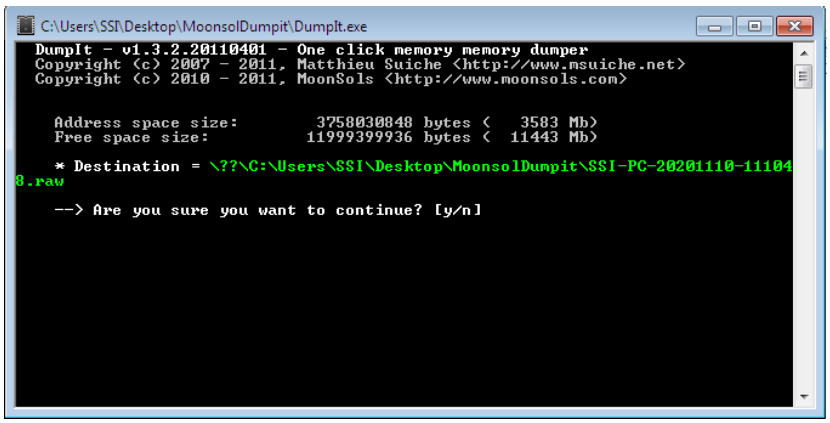


Figura 2.4: Memory Dump - Dumpit

2.7 Outros contidos volátiles

Aparte da memoria *RAM* dun ordenador existen outros compoñentes volátiles que se han de gardar para así poder analizalos con posterioridade. Existen ferramentas internas ou nativas dos sistemas (**Windows**) para conseguir algún destes datos pero é convinte utilizar ferramentas de terceiros empregadas en medios extraíbles protexidos contra a escritura para

asegurar a integridade das evidencias tomadas xa que calqueira proceso que corramos na memoria do equipo obxetivo de análise pode provocar unha sobre-escritura dos datos e unha modificación da mesma.
Algunha destas ferramentas de terceiros son por exemplo OSForensics ou Sysinternals Suite.

2.7.1 Hora e fecha do sistema

Windows proporciona ferramentas para obter a fecha ou a hora directamente dende o intérprete de comandos. Estes datos é importante obtelos para establecer posteriormente unha liña temporal do caso de estudio ou se polo contrario a fecha e hora establecidas son incorrectas intentar de averiguar o porqué.
Para obter a fecha simplemente executamos o comando: **# date /t** e para obter a hora: **# time /t**.

2.7.2 Histórico do intérprete de comandos

Podemos consultar os últimos comandos escritos na consola de **Windows** (*CMD*) empregando o seguinte comando: **# doskey /history**

2.7.3 Pagefiles

Cando a *RAM* dos sistemas énchese debido o seu uso, **Windows** move algun dato dende esta *RAM* ao noso disco duro hubicandoo na **pagefile**. Este arquivo é unha forma de memoria virtual.
Esta lectura de datos é mais lenta que se lesemos directamente da memoria *RAM*, pero sirvenos como unha memoria de respaldo, en lugar de tirar datos potencialmente importantes ou que os programas o bloqueen, almacenámolos no disco duro, por iso a hora de realizar un análise forense é importante obter esta **pagefile**.
A forma máis básica de obtelas é directamente copiando *C:\pagefile.sys*. Pero isto a veces é complicado de facer, pois o sistema está continuamente empregando estes ficheiros o que dificulta a súa copia directa, e é por isto que existen ferramentas que xa nos realizan este traballo automáticamente como é por exemplo **OSForensics**.

2.7.4 Árbol de directorios e ficheiros

Pódense obter os seguintes listados:

- Listado en base a fecha de modificación: **# dir /t:w /a /s /o:d c:**
- Listado en base o último acceso: **# dir /t:a /a /s /o:d c:**
- Listado en base a fecha de creación: **# dir /t:c /a /s /o:d c:**

Capítulo 3

Volatility framework

Volatility[13] é unha colección de ferramentas, implementadas en *Python*. É utilizado principalmente para extraer artefactos dixitais dende volcados de memoria *RAM*. **Volatility** soporta volcados de memoria dende as principais versións de **Windows** de 32 e 64 bits ademais das principais distribucións de **Linux** e **MAC OSX**.

Volatility é capaz de analizar formatos de arquivos *raw*, *crash dumps*, *archivos de hibernación*, estados gardados de virtualizadores como *VMware*, *LIME*, *EWF*,.... Dispón ademais dunha serie de algoritmos rápidos e eficientes que lle permiten analizar os volcados de memoria *RAM* de sistemas de gran volume sen un consumo de recursos excesivo en moi pouco tempo.

Este framework contén unha gran cantidade de plugins que extenden a funcionalidade de **Volatility** añadindolle novas e potentes características. Aínda así xa incorpora infinidade de módulos ou ferramentas que se poden ver directamente no git da ferramenta.

Na súa versión orixinal non incluia unha *GUI*, pois todas as ferramentas accedianse mediante a consola de comandos facendo isto algo complicado para os que non estiveran afeitos. Por iso lanzaron unha nova versión que incorpora un interfaz gráfica de usuario, Volatility Workbench.

3.1 Instalación de *Volatility*

Para a instalación podemos recurrir a dúas maneiras, a primeira é descargando directamente *Standalone Windows Executable*. Esta é a maneira mais rápida e sencilla de empezar a empregar *Volatility* en **Windows**. A outra maneira, que nos eleximos, é instalar o que eles chaman *Source Code Packages*. Esta maneira é a mais versátil pois é común para os tres sistemas operativos.

Tanto para a primeira coma para a segunda maneira requerimos a instalación do interprete de *Python* [14], no noso caso a versión **2.7.18**. Para a súa instalación solo necesitamos descar-

gar a versión correspondente dende esta ligazón e seguir os pasos do executable. Unha vez realizada a instalación debemos incluir o *PATH* do executable de python para poder acceder a el dende calqueira carpeta.

Unha vez instalado **Python** procedemos a instalación de *Volatility* dende o seguinte enlace, a versión que nos escollimos foi a **2.4** e o seu respectivo *Source pythonCode*. Unha vez descargado debemos executar *setup.py install* e unha serie de módulos *third-party* a maiores.

1. **Distorm3** [15]: É un desensamblador moi potente, podemos descargalo dende o repositorio oficial e se o instalamos en **Windows** tan so fai falla executar un *.exe*.

2. **Yara** [16]: Permite identificar e clasificar o *Malware*. Como o anterior solo temos que executar un *.exe* descargandoo dende o seguinte enlace se o instalamos en **Windows**.

3. **pyCrypto**: É o quit de ferramentas de criptografía de *Python*, para descargalo temos que facer dous pasos previos:

   - Instalar Visual C++ Compiler for Python 2.7.
   - Instalar **pip**[17] se non está instalado xa na nosa versión de **Python**.
   - Instalar **pyCrypto** mediante o seguinte comando *pip install pycrypto*.

3.2 Primeiros pasos

Mais adiante veremos un funcionamento máis en profundidade desta ferramenta, pero agora ímonos centrarnos nos principais aspectos.

En canto aos comandos básicos en *Volatility* a estructura mais básica componse do *path* do volcado de memoria a analizar, o nome do profile e a continuación o *plugin* a executar (opcionalmente parámetros).

python vol.py –f <FILENAME> –profile=<PROFILE> <PLUGIN> [ARGS]

Para ensinar o menu de opcións simplemente pasamoslle a opción **-h ou –help** así mesmo tamén existe a opción de mostrar as opcións dun *plugin* específico introducindo o nome e **-h** ou **-help**.

3.2.1 Seleccionando o Profile

Unha das principais opcións que vamos usar moi frecuentemente é **–profile**. Esta opción dille a **Volatility** de que tipo de sistema provén o arquivo de volcado de memoria, para así poder coñecer que estructura de datos, algoritmos e símbolos vai a empregar.

Internamente **Volatility** ten asignado como *profile* por defecto **WinXPSP2x86** que se corresponde a un *Windows XP SP2* de 32 bits. Asique se estamos analizando a memoria desde sistema operativo non fai falla incluir esta opción, do contrario debémola incluir.

En moitos casos non sabemos que sistema estamos a analizar, para solucionar isto, *Volatility* inclúe dous plugins para determinalo, que son *imageinfo* e *kdbgscan*. *Imageinfo* proporciona un resumo de alto nivel da mostra (memory dump) que se está a analizar. Moitas veces varios *profiles* son suxeridos, esto débese porque varios sistemas operativos comparten moitas características que poden facer que sexan moi similares. Un exemplo do uso deste *plugin* podémolo ver na seguinte imaxen.

```
C:\Users\SSI\Desktop\volatility-2.6 (1)\volatility-master>python vol.py -f 20201
110.mem imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86
                     AS Layer1 : FileAddressSpace (C:\Users\SSI\Desktop\volatili
ty-2.6 (1)\volatility-master\20201110.mem)
                      PAE type : No PAE
                           DTB : 0x185000L
                          KDBG : 0x26cbc28L
          Number of Processors : 2
     Image Type (Service Pack) : 27648
       KPCR for CPU 1308821511 : 0x750eb360L
        KPCR for CPU 554306591 : 0x750eb330L
             KUSER_SHARED_DATA : 0xffdf0000L
```

Figura 3.1: Plugin Imageinfo

Para ter unha idea mais clara do *profile* que estamos a analizar podemos empregar o *plugin kdbgscan*. Este atopa e analiza características do *kernel debugger data block* (_KDDEBUGGER_DATA64). Esta estructura de datos está típicamente aloxada dentro do *NT kernel module*, contén varios campos como son un string, valores númericos que indican o maior e menor número de compilación e o *service pack*[18] do sistema operativo. Un exemplo da saída da execución do *plugin* é a seguinte.

```
C:\Users\SSI\Desktop\volatility-2.6 (1)\volatility-master>python vol.py -f 20201
110.mem --profile=Win7SP0x86 kdbgscan
Volatility Foundation Volatility Framework 2.6
**************************************************
Instantiating KDBG using: C:\Users\SSI\Desktop\volatility-2.6 (1)\volatility-mas
ter\20201110.mem Win7SP0x86 (6.1.7600 32bit)
Offset (P)                    : 0x26cbc28
KDBG owner tag check          : True
Profile suggestion (KDBGHeader): Win7SP1x86_23418
Version64                     : 0x26cbc00 (Major: 15, Minor: 7601)
PsActiveProcessHead           : 0x82744f18
PsLoadedModuleList            : 0x8274c850
KernelBase                    : 0x82602000

**************************************************
Instantiating KDBG using: C:\Users\SSI\Desktop\volatility-2.6 (1)\volatility-mas
ter\20201110.mem Win7SP0x86 (6.1.7600 32bit)
Offset (P)                    : 0x26cbc28
KDBG owner tag check          : True
Profile suggestion (KDBGHeader): Win7SP1x86
Version64                     : 0x26cbc00 (Major: 15, Minor: 7601)
PsActiveProcessHead           : 0x82744f18
PsLoadedModuleList            : 0x8274c850
KernelBase                    : 0x82602000
```

Figura 3.2: Plugin kdbgscan

Algunhas veces cando executamos este plugin encontrámonos con varias estructuras de datos (diferentes *offsets*). Isto débese porque existe máis dunha estructura *_KDDEBUGGER_DATA64* residente na memoria física. Esto pode pasar si o sistema de destino non se reiniciou dende a aplicación dun parche de actualización ou se a máquina se reiniciou tan rápido que non se vaciou todo o contido da *RAM*.
Se nos atopamos con isto temos que fixarnos nos atributos *PsActiveProcessHead* e *PsLoaded-ModuleList*. Se ambos dous teñen 0 procesos e 0 módulos non é a estructura correcta.

Capítulo 4

Volatility en Windows

Neste capítulo vamos probar a ferramenta **Volatility** en **Windows**, vendo os pincipais usos que lle podemos dar a abordando as diferentes seccións máis importantes que nos podemos encontrar no noso sistema.

4.1 Analizando os procesos

Volatility ofrecenos varios comandos que podemos empregar para extraer información acerca dos procesos. Estes comandos son:

- **pslist**: Este comando atopa e recorre a lista de procesos dobremente enlazados e imprime un resumen dos resultados. A única desventaxe que ten é que non pode amosar procesos ocultos ou terminados.
- **pstree**: Toma a saída de **pslist** e a formatea para que podamos ver a relación que poida haber entre os procesos pai e os procesos fillo.
- **psscan**: A diferencia co primeiro comando, este busca obxetos *_EPROCESS*[19]. Este tipo de estructura forma o que se chama a cadea de obxetos de proceso ou tamen chamada, lista drobemente enlazada. Grazas a este tipo de búsqueda permitenos atopar procesos rematados ou que están ocultos.
- **psxview**: Localiza procesos empregando *alternate process listings*, para que logo poida facer referencias cruzadas de diferentes fontes de información e revelar así discrepancias que poden resultar maliciosas.

Un exemplo do comando **pslist** é o seguinte:

```
SSI@SSI-PC MINGW32 ~/Desktop/volatility-2.6 (1)/volatility-master
$ python vol.py -f memoryDump.raw --profile=Win7SP1x86_23418 pslist
Volatility Foundation Volatility Framework 2.6
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------------------------------
0x84eca870 System                    4      0     75      535 ------      0 2020-11-10 10:31:00 UTC+0000
0x85ea6d40 smss.exe                252      4      2       29 ------      0 2020-11-10 10:31:00 UTC+0000
0x86764778 csrss.exe               320    312      8      375      0      0 2020-11-10 10:31:02 UTC+0000
0x86aa2030 wininit.exe             356    312      3       75      0      0 2020-11-10 10:31:03 UTC+0000
0x86aa2428 csrss.exe               368    348     11      370      1      0 2020-11-10 10:31:03 UTC+0000
0x86aaf030 winlogon.exe            408    348      5      115      1      0 2020-11-10 10:31:03 UTC+0000
0x85f9c030 services.exe            452    356      8      191      0      0 2020-11-10 10:31:03 UTC+0000
0x85ffb030 lsass.exe               460    356      6      554      0      0 2020-11-10 10:31:03 UTC+0000
0x86adbd40 lsm.exe                 468    356     10      141      0      0 2020-11-10 10:31:03 UTC+0000
0x85d88030 svchost.exe             572    452     10      352      0      0 2020-11-10 10:31:04 UTC+0000
0x85dda480 svchost.exe             648    452      7      262      0      0 2020-11-10 10:31:04 UTC+0000
0x86d69030 svchost.exe             756    452     22      491      0      0 2020-11-10 10:31:04 UTC+0000
0x86d85030 svchost.exe             824    452     13      307      0      0 2020-11-10 10:31:04 UTC+0000
0x86d89948 svchost.exe             856    452     40     2182      0      0 2020-11-10 10:31:04 UTC+0000
0x86c2dab0 audiodg.exe             924    756      6      129      0      0 2020-11-10 10:31:04 UTC+0000
0x86eae030 svchost.exe             972    452     17      437      0      0 2020-11-10 10:31:04 UTC+0000
0x86c56aa8 svchost.exe            1068    452     17      498      0      0 2020-11-10 10:31:04 UTC+0000
0x86ef2a20 spoolsv.exe            1204    452     12      268      0      0 2020-11-10 10:31:05 UTC+0000
0x86f0c3c8 svchost.exe            1240    452     19      316      0      0 2020-11-10 10:31:05 UTC+0000
0x86f3e800 svchost.exe            1340    452     15      243      0      0 2020-11-10 10:31:05 UTC+0000
0x86f4d8b0 dwm.exe                1788    824      3       72      1      0 2020-11-10 10:31:08 UTC+0000
0x86ff1180 explorer.exe           1824   1780     27      786      1      0 2020-11-10 10:31:09 UTC+0000
0x87002030 taskhost.exe           1832    452      9      165      1      0 2020-11-10 10:31:09 UTC+0000
0x87089030 SearchIndexer.         1280    452     14      682      0      0 2020-11-10 10:31:15 UTC+0000
0x870cad40 wmpnetwk.exe           1872    452      9      205      0      0 2020-11-10 10:31:15 UTC+0000
0x871c4030 GoogleCrashHan         3336   3068      5       91      0      0 2020-11-10 10:32:13 UTC+0000
0x84f54918 sppsvc.exe             2316    452      4      142      0      0 2020-11-10 10:33:05 UTC+0000
0x8527db48 svchost.exe            3148    452     10      302      0      0 2020-11-10 10:33:05 UTC+0000
0x864df848 chrome.exe              780   1824     35     1056      1      0 2020-11-10 10:33:45 UTC+0000
0x8524a920 chrome.exe             2936    780      9       98      1      0 2020-11-10 10:33:45 UTC+0000
0x872775d0 chrome.exe              564    780     16      440      1      0 2020-11-10 10:33:46 UTC+0000
0x86f9e030 chrome.exe             2896    780      8      133      1      0 2020-11-10 10:33:46 UTC+0000
0x8505b4c8 chrome.exe             2436    780      8      190      1      0 2020-11-10 10:33:48 UTC+0000
0x8724f030 chrome.exe              984    780     13      253      1      0 2020-11-10 10:33:52 UTC+0000
0x872164b8 chrome.exe              288    780     14      178      1      0 2020-11-10 10:33:54 UTC+0000
0x851c3d40 chrome.exe             3092    780     12      170      1      0 2020-11-10 10:33:57 UTC+0000
0x8710e428 TrustedInstall         2328    452      4      116      0      0 2020-11-10 10:33:59 UTC+0000
0x850b5d40 WmiPrvSE.exe           3028    572      7      115      0      0 2020-11-10 10:35:05 UTC+0000
0x85072ad0 chrome.exe             2080    780      0 --------      1      0 2020-11-10 10:35:30 UTC+0000   2020-11-10 10:35:52 UTC+0000
0x87191b60 chrome.exe             2192    780      0 --------      1      0 2020-11-10 10:35:32 UTC+0000   2020-11-10 10:35:52 UTC+0000
0x8723e658 chrome.exe             2164    780     12      180      1      0 2020-11-10 10:35:37 UTC+0000
0x871d4520 chrome.exe             1936    780     10      172      1      0 2020-11-10 10:35:37 UTC+0000
0x85402548 chrome.exe             3676    780      0 --------      1      0 2020-11-10 10:35:39 UTC+0000   2020-11-10 10:35:54 UTC+0000
0x870ff3f8 chrome.exe             2656    780      0 --------      1      0 2020-11-10 10:35:43 UTC+0000   2020-11-10 10:35:59 UTC+0000
0x852691b0 chrome.exe             3632    780     12      159      1      0 2020-11-10 10:35:45 UTC+0000
0x870d2c68 SearchProtocol         1428   1280      8      283      0      0 2020-11-10 10:35:48 UTC+0000
0x850749a8 SearchFilterHo         2384   1280      5       95      0      0 2020-11-10 10:35:49 UTC+0000
0x850ac560 MRCv120.exe            3476   1824     15      301      1      0 2020-11-10 10:35:59 UTC+0000
0x868a93b0 software_repor          612    780      7      145      1      0 2020-11-10 10:36:19 UTC+0000
0x850ab560 software_repor         2900    612      7 163...33      1      0 2020-11-10 10:36:20 UTC+0000
0x850ae7c0 software_repor         2376    612      1       90 ------      0 2020-11-10 10:36:22 UTC+0000
```

Figura 4.1: Pslist

Como podemos ver a primeira columna *Offset(V)* amosanos a dirección virtual (na memoria kernel) da estructura *_EPROCESS*. A seguinte columna corresponde o nome do proceso, o seu *PID*, o *Parent PID*, número de threads que creou, número de punterios (*handles*), *Session id* e a fecha de creación e de eliminación.

O *Session id* dun proceso pódenos indicar de que sistema operativo se trata. Antes de *Windows XP* asignabaselle como *Session id* 0 a todos os procesos que son corridos polo primeiro usuario loggeado, pero a partir de **Windows Vista**, *Session* 0 está reservada unicamente para servicios e outras aplicacións que non as inicia o usuario. Os usuarios que están loggeados no sistema, as súas aplicacions deben correr na *Session* 1 ou máis altas.

A maiores os procesos que teñen 0 *threads*, 0 *handles* e/ou un *exit time* significa que ese proceso non está activo. Tamén destacar que os procesos *System* e *smss.exe* nunca van ter un *Session id* asociado, pois *System* empeza antes de que a sesión sexa establecida e *smss.exe* é o manager das sesións.

A continuación vamos ver o comando **pstree**. Este plugin permítenos extender o noso coñecemento acerca da orixe dos procesos, pois proporcionanos unha interpretación visual entre as relacións entre procesos pais e fillos. Emprega o mesmo mecanismo que pslist polo que non vamos poder ver os procesos ocultos e *unliked*.
Como vemos na seguinte imaxen os fillos, están indexados hacia a dereita e precedidos de puntos.

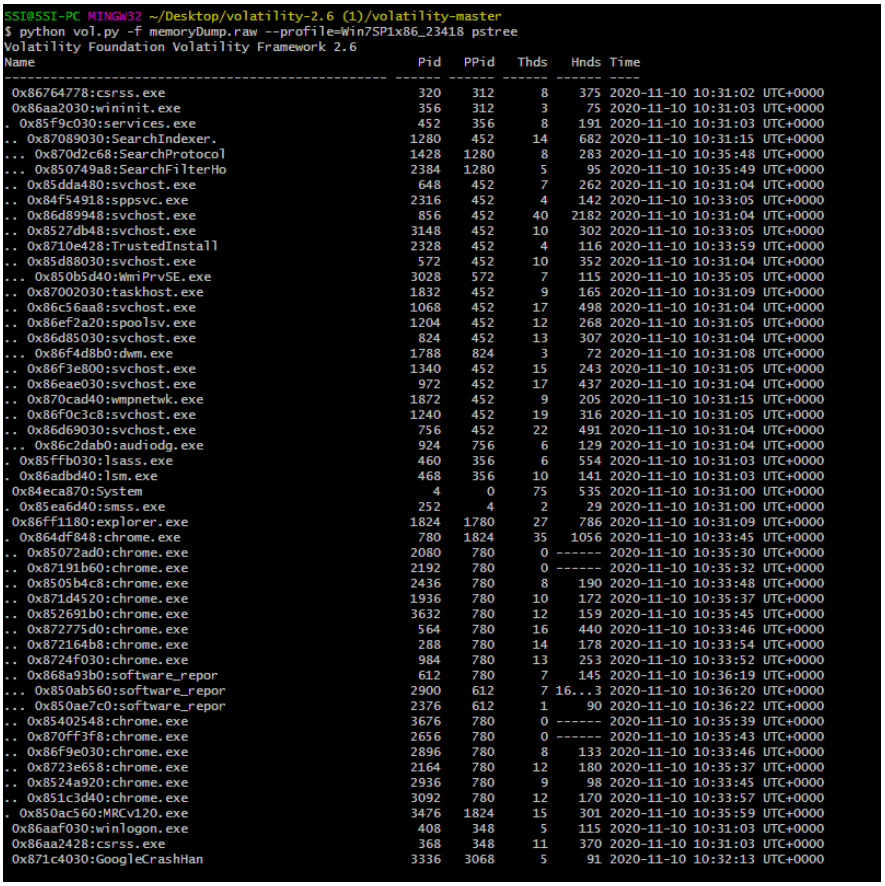


Figura 4.2: Pstree

Vendo o árbore de procesos, podemos determinar mais facilmente os posibles eventos que ocurriron durante o ataque. Podemos ver como *chrome.exe* co PID 780 empezou a partires de *explorer.exe* co PID 1824. Esto ocurre sempre, pois sempre que iniciamos unha aplicación a través do *start menu* ou clickando no icono do escritorio o proceso pai sempre vai ser *Windows Explorer*.

Outra forma de ver esta árbore de relacións dunha maneira máis gráfica é mediante o plugin **psscan** a través do render de gráficos **dot**[20]. A diferencia dos anteriores este si que nos amosa os procesos acabados ou ocultos. Podemos xerar o gráfico mediante:

```
SSI@SSI-PC MINGW32 ~/Desktop/volatility-2.6 (1)/volatility-master
$ python vol.py psscan -f SSI-PC-20201110-110930.raw --profile=Win7SP1x86 --output=dot --output-file=processes1.dot
Volatility Foundation Volatility Framework 2.6
Outputting to: processes1.dot
```

Figura 4.3: Psscan

O resultado gárdasenos nun ficheiro *.dot* que o podemos visualizar con múltiples ferramentas. Nós eleximos Graphviz a que simplemente temos que instalar e executar o seguinte comando para obter un pdf co resultado, **# dot -Tpdf processes1.dot -o graph1.pdf**.

Unha parte do resultado sería o seguinte, onde podemos ver os procesos comentados anteriormente.

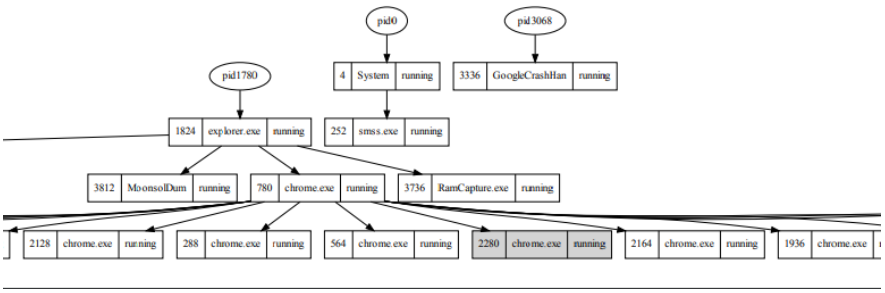


Figura 4.4: Diagrama do árbore de procesos

Continuando co plugin **psxview** que enumera os procesos en 7 maneiras diferentes, a *linked list* de procesos activos e seis novos métodos. Este novos métodos son coñecidos co nome de *alternate process listings* e son os seguintes:

- **Process object scanning**: Tamén coñecido como *Pool Scanning* que consiste en localizar obxetos do kernel, o que nos da a opción de localizar procesos que foron liberados ou ocultados. Esta técnica require dunha búsqueda exhaustiva da memoria física.
- **Thread scanning**: Todo proceso ten polo menos un *thread* activo. Polo tanto podemos buscalos polos obxetos *_ETHREAD* e logo mapealos de novo ao seu proceso propiedade.
- **CSRSS handle table**: O proceso *csrss.exe* está involucrado na creación de cada proceso e *thread*.
- **PspCid table**: É unha tabla especial ubicada na memoria do kernel que almacena unha referencia a todos os procesos activos.

- **Session processes**: *_EPROCESS* asocia a todos os procesos que pertencen a mesma sesión de inicio dun usuario.
- **Desktop threads**: Este tipo de estructuras almacenan unha lista de todos os *threads* adxuntos a cada escritorio, e pódese asignar cada un deses fíos ao seu proceso orixe.

Como vemos, a seguinte imaxen mostra sete columnas indicando con **true** ou **false** se atopou ese proceso empregando cada un dos anteriores métodos mencionados.

```
SSI@SSI-PC MINGW32 ~/Desktop/volatility-2.6 (1)/volatility-master
$ python vol.py -f SSI-PC-20201110-110930.raw --profile=Win7SP1x86 psxview
Volatility Foundation Volatility Framework 2.6
Offset(P)  Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
---------- -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
0xdf1da480 svchost.exe             648 True   True   True     True   True  True    True
0xddf3e800 svchost.exe            1340 True   True   True     True   True  True    True
0xde291a10 chrome.exe             3612 True   True   True     True   True  True    True
0xde57e558 chrome.exe             1040 True   True   True     True   True  True    True
0xddff1180 explorer.exe           1824 True   True   True     True   True  True    True
0xddf0c3c8 svchost.exe            1240 True   True   True     True   True  True    True
0x05154918 sppsvc.exe             2316 True   True   True     True   True  True    True
0xddc89030 SearchIndexer.         1280 True   True   True     True   True  True    True
0xddef2a20 spoolsv.exe            1204 True   True   True     True   True  True    True
0xddeae030 svchost.exe             972 True   True   True     True   True  True    True
0xdf188030 svchost.exe             572 True   True   True     True   True  True    True
0xde056aa8 svchost.exe            1068 True   True   True     True   True  True    True
0xdef9c030 services.exe            452 True   True   True     True   True  True    False
0xdfa7db48 svchost.exe            3148 True   True   True     True   True  True    False
0xde3a1d40 SearchFilterHo          748 True   True   True     True   True  True    True
0xddf9e030 SearchProtocol         3720 True   True   True     True   True  True    True
0xddccad40 wmpnetwk.exe           1872 True   True   True     True   True  True    True
0xddce0780 audiodg.exe            2636 True   True   True     True   True  True    True
0xdfcbe030 chrome.exe             3716 True   True   True     True   True  True    True
0xde189948 svchost.exe             856 True   True   True     True   True  True    True
0xde2a2030 wininit.exe             356 True   True   True     True   True  True    True
0xddca07b8 chrome.exe             2412 True   True   True     True   True  True    True
0x05168d40 chrome.exe             4036 True   True   True     True   True  True    True
0xdddc4030 GoogleCrashHan         3336 True   True   True     True   True  True    True
0xde2af030 winlogon.exe            408 True   True   True     True   True  True    True
0xde185030 svchost.exe             824 True   True   True     True   True  True    True
0xde6aad40 chrome.exe             1256 True   True   True     True   True  True    True
0xddaef030 MoonsolDum             3812 True   True   True     True   True  True    True
0xdddd98e0 chrome.exe             2540 True   True   True     True   True  True    True
0xde169030 svchost.exe             756 True   True   True     True   True  True    True
0xddf4d8b0 dwm.exe                1788 True   True   True     True   True  True    True
0xdeffb030 lsass.exe               460 True   True   True     True   True  True    False
0xde4d8168 conhost.exe             928 True   True   True     True   True  True    True
0xde5cc030 WmiPrvSE.exe           2240 True   True   True     True   True  True    True
0xde2dbd40 lsm.exe                 468 True   True   True     True   True  True    False
0xddc02030 taskhost.exe           1832 True   True   True     True   True  True    True
0xde2a2428 csrss.exe               368 True   True   True     True   False True    True
0xdeea6d40 smss.exe                252 True   True   True     True   False False   False
0x050ca870 System                    4 True   True   True     True   False False   False
0xde764778 csrss.exe               320 True   True   True     True   False True    True
0x4cdb7800 svchost.exe            1340 False  True   False    False  False False   False
0x458c68b0 dwm.exe                1788 False  True   False    False  False False   False
0x12111030 svchost.exe             572 False  True   False    False  False False   False
0x4163a030 chrome.exe              144 False  True   False    False  False False   False    2020-11-10 11:01:57 UTC+0000
0x49f17030 chrome.exe             2896 False  True   False    False  False False   False
0x3e568848 chrome.exe              780 False  True   False    False  False False   False
0x28184030 lsass.exe               460 False  True   False    False  False False   False
```

Figura 4.5: Psxview

No anterior exemplo incluímos o flag *–aply-rules*, que nos indica cun **Okey** aquelas columnas que non se atopou o proceso pero que non é sospeitoso.

A maiores existen outros plugins para traballos con procesos. Un deles é **getsids** que devolve a asociación que existe entre un proceso e a conta dun usuario. Un exemplo do uso deste plugin é o seguinte.

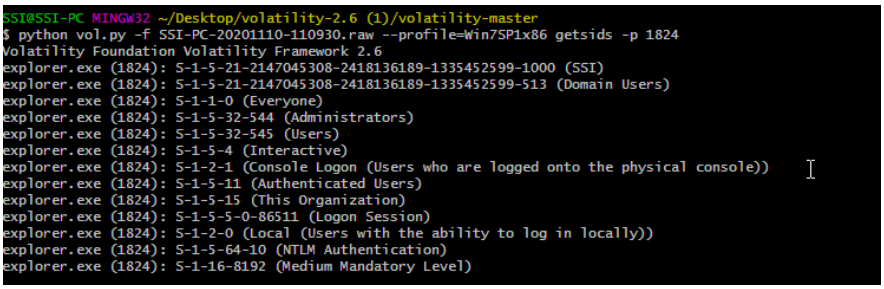


Figura 4.6: Getsids

Podemos ver como o proceso *explorer.exe* foi iniciado polo usario con SID (S-1-S-21-2147045308-[snip]-1000). Este proceso tamén é membro dos administradores do sistema polo que se un atacante se une ao sistema e pode executar este proceso vai poder navegar libremente por todo o sistema.

Outro aspecto importante dos procesos son os privilexios. Un privilexio é un permiso para realizar unha tarefa específica, como depurar un proceso, apagar o sistema, cambiar a zona horaria....
Un administrador do sistema pode indicar que privillexios están presentes configurandoos na *Local Security Policy* (LSP). Un privilexio pode habilitarse das seguintes maneiras: hablitandoo de forma predeterminada mediante a *LSP*, mediante herencia do proceso pai, ou que un proceso habilite explicitamente un privilexio.
Dende o que realmente nos importa, a perspectiva forense, tennos que preocupar os privilexios que foron habilitados da última forma en especial os seguintes:

- **SeBackupPrivilege**: Este privilexio otorga acceso de lectura a calqueira arquivo no sistema, polo que os atacantes poden incluso ata copiar arquivos bloqueados.
- **SeDebugPrivilege**: Otorga a capacidade de leer e escribir noutro proceso. Practicamente todo o *malware* que realiza *code injection* adquire este tipo de privilexio.
- **SeLoadDriverPrivilege**: Capacidade de cargar e descargar controladores do kernel.
- **SeChangeNotifyPrivilege**: Notifica cando se cambian arquivos ou directorios. Os atacantes poden empregar isto para saber se un dos seus arquivos de configuración ou executables foi eliminado por un antivirus ou polos administradores do sistema.
- **SeShutdownPrivilege**: Permite apagar ou reiniciar o sistema.

Para analizar estes privilexios acerca dun proceso temos o plugin **privs**.

```
SSI@SSI-PC MINGW32 ~/Desktop/volatility-2.6 (1)/volatility-master
$ python vol.py -f SSI-PC-20201110-110930.raw  --profile=Win7SP1x86 privs -p 1824
Volatility Foundation Volatility Framework 2.6
Pid      Process          Value  Privilege                         Attributes               Description
-------- ---------------- ------ --------------------------------- ------------------------ -----------
    1824 explorer.exe          2 SeCreateTokenPrivilege                                     Create a token object
    1824 explorer.exe          3 SeAssignPrimaryTokenPrivilege                              Replace a process-level token
    1824 explorer.exe          4 SeLockMemoryPrivilege                                      Lock pages in memory
    1824 explorer.exe          5 SeIncreaseQuotaPrivilege                                   Increase quotas
    1824 explorer.exe          6 SeMachineAccountPrivilege                                  Add workstations to the domain
    1824 explorer.exe          7 SeTcbPrivilege                                             Act as part of the operating system
    1824 explorer.exe          8 SeSecurityPrivilege                                        Manage auditing and security log
    1824 explorer.exe          9 SeTakeOwnershipPrivilege                                   Take ownership of files/objects
    1824 explorer.exe         10 SeLoadDriverPrivilege                                      Load and unload device drivers
    1824 explorer.exe         11 SeSystemProfilePrivilege                                   Profile system performance
    1824 explorer.exe         12 SeSystemtimePrivilege                                      Change the system time
    1824 explorer.exe         13 SeProfileSingleProcessPrivilege                            Profile a single process
    1824 explorer.exe         14 SeIncreaseBasePriorityPrivilege                            Increase scheduling priority
    1824 explorer.exe         15 SeCreatePagefilePrivilege                                  Create a pagefile
    1824 explorer.exe         16 SeCreatePermanentPrivilege                                 Create permanent shared objects
    1824 explorer.exe         17 SeBackupPrivilege                                          Backup files and directories
    1824 explorer.exe         18 SeRestorePrivilege                                         Restore files and directories
    1824 explorer.exe         19 SeShutdownPrivilege            Present                     Shut down the system
    1824 explorer.exe         20 SeDebugPrivilege                                           Debug programs
    1824 explorer.exe         21 SeAuditPrivilege                                           Generate security audits
    1824 explorer.exe         22 SeSystemEnvironmentPrivilege                               Edit firmware environment values
    1824 explorer.exe         23 SeChangeNotifyPrivilege        Present,Enabled,Default     Receive notifications of changes to files or directories
    1824 explorer.exe         24 SeRemoteShutdownPrivilege                                  Force shutdown from a remote system
    1824 explorer.exe         25 SeUndockPrivilege              Present                     Remove computer from docking station
    1824 explorer.exe         26 SeSyncAgentPrivilege                                       Synch directory service data
    1824 explorer.exe         27 SeEnableDelegationPrivilege                                Enable user accounts to be trusted for delegation
    1824 explorer.exe         28 SeManageVolumePrivilege                                    Manage the files on a volume
    1824 explorer.exe         29 SeImpersonatePrivilege                                     Impersonate a client after authentication
    1824 explorer.exe         30 SeCreateGlobalPrivilege                                    Create global objects
    1824 explorer.exe         31 SeTrustedCredManAccessPrivilege                            Access Credential Manager as a trusted caller
    1824 explorer.exe         32 SeRelabelPrivilege                                         Modify the mandatory integrity level of an object
    1824 explorer.exe         33 SeIncreaseWorkingSetPrivilege  Present                     Allocate more memory for user applications
    1824 explorer.exe         34 SeTimeZonePrivilege            Present                     Adjust the time zone of the computer's internal clock
    1824 explorer.exe         35 SeCreateSymbolicLinkPrivilege                              Required to create a symbolic link
```

Figura 4.7: Privs

Como vemos existen varios privilexios no proceso *explorer.exe*. Estes poden ir precedidos dos atribuos *Present*, *Enabled*, *Default*. O que a nós nos interesaría sería ver aqueles privilexios que estean *Enabled* pero que non fosen creados por defecto, *Default*.

4.2 Analizando os events logs

Os rexistros de eventos conteñen unha gran cantidade de información forense e son un elemento básico en calqueira investigación. Conteñen detalles sobre errores da aplicación, inicios de sesión, cambios na política do *firewall* e outros eventos que han ocurrido no sistema.

Depende de que sistema operativo **Windows** esteamos a analizar, estes arquivos vanse gardar en formatos completamente distintos. No noso caso, **Windows 7**, están contidos nun formato binario *XML*, en arquivos coa extensión *.evtx*. Podemos atopar máis de 60 rexistros no disco en *%systemroot%\system32 \winevt \Logs*. Para poder extraer os rexitros da memoria, vamos ter que empregar o plugin **dumpfiles** e logo analizalos con unha ferramenta externa a **Volatility**.

Se executamos o plugin buscando por esa extensión obtemos a seguinte saída, e os ficheiros gardaranse no directorio *output*.

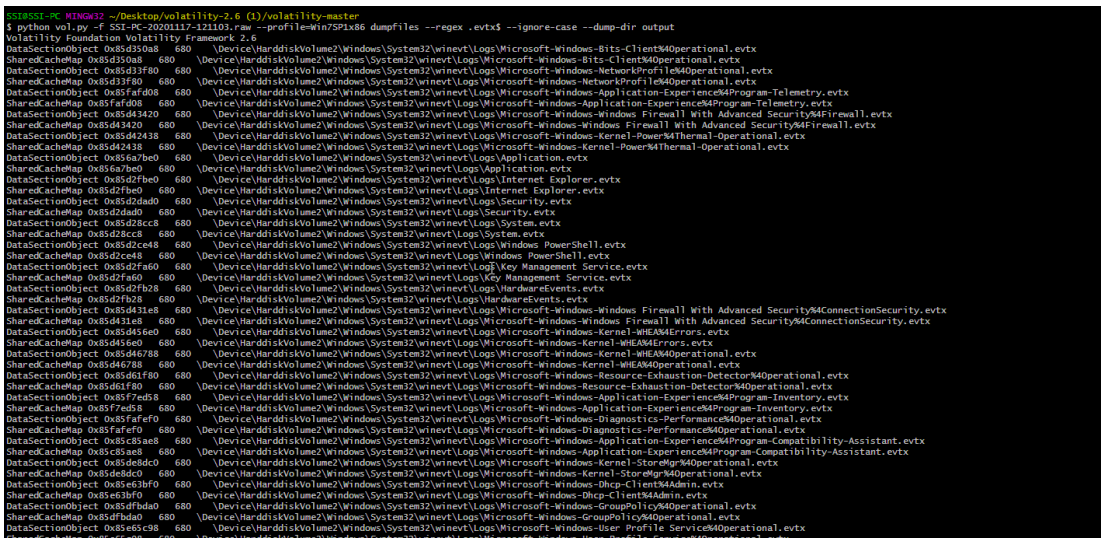


Figura 4.8: Dumpfiles

Dentro destes ficheiros podemos ver varios campos de interese como son o *Provider name* que nos di dende que log foi extraida a información, o *Event ID* para comprobar en internet que evento ocurriu, *TimeCreated* que indica cando foi xerado o evento, ...

4.3 Extracción dos password hash

Como en moitos sistemas, **Windows** protexe as contrasinais dos usuarios gardándoas empregando *hashes* de esas contrasinais. Antes de proceder a extraer os *hashes* debemos coñecer os tres conceptos mais básicos que emprega **Windows**:

- **LM hash**: é unha anticuada técnica que se empregou ata **Windows Vista** pero que pode ser habilitada se se quere. Deixouse de empregar porque as contrasinais non distinguian de maiúsculas e minúsculas, a lonxitude do contrasinal era de 14 caracteres como máximo e porque dividía o texto en dúas mitades de 7 caracteres antes de codificalos por separado e volvendoos a concatenar. Asique se a nosa contrasinal tiña menos de 7 caracteres era moi fácil de descifrar.
- **NT hash**: é a forma mais recente. Primeiro codifica o contrasinal usando *UTF-16-LE* e logo emprega o algoritmo hash *MD-4*.
- **SAM database file**: é a database que alberga os *hashes* dos contrasinais do usuario. Vense empregando dende **Windows XP**

Podemos volcar os *hashes* das contrasinais gardadas empregando o plugin *hashdump*. Este recolle a información acerca dos contidos *SYSTEM* e *SAM*.

A saída do plugin hashdump é a seguinte.

```
SSI@SSI-PC MINGW32 ~/Desktop/volatility-2.6 (1)/volatility-master
$ python vol.py -f SSI-PC-20201117-121103.raw  --profile=Win7SP1x86 hashdump
Volatility Foundation Volatility Framework 2.6
Administrador:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Invitado:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
SSI:1000:aad3b435b51404eeaad3b435b51404ee:e6ae2d8498c47ba47fe07b85eff78e03:::
```

Figura 4.9: Hashdump

Como vemos, obtemos os *hashes* das contrasinais de administrador e do usuario *SSI*. Para intentar descifralas recurrimos a ferramenta Jhon the Ripper e a un diccionario de palabras creado previamente para intentar descifrala mediante a forza bruta. Como resultado da execución da ferramenta obtemos a seguinte saída, coa contrasinal descifrada **fic2020**:

```
PS C:\Users\andre\Downloads\john-1.9.0-jumbo-1-win64\run> .\john.exe --rules --wordlist=pass.txt --format=nt .\hashes.tx
t
Using default input encoding: UTF-8
Loaded 2 password hashes with no different salts (NT [MD4 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=12
Press 'q' or Ctrl-C to abort, almost any other key for status
fic2020          (SSI)
1g 0:00:00:00 DONE (2020-11-18 12:53) 55.55g/s 16777p/s 16777c/s 27444C/s Lukitas5..Oiing
Warning: passwords printed above might not be all those cracked
Use the "--show --format=NT" options to display all of the cracked passwords reliably
Session completed
```

Figura 4.10: Jhon The Ripper

4.4 Networking

As dúas entidades máis importantes dos compoñentes do término *networking* en **Windows** son os **sockets** e as **conexións**.

Os **sockets** [21] definen puntos finais para as comunicacións. As aplicacións crean **sockets** de cliente para iniciar conexións a servidores, e ao mesmo tempo, crean **sockets** de servidor para escoitar nunha interfaz as conexións entrantes. Estes **sockets** pódense crear de tres maneiras distintas en **Windows**, directamente dende o user mode chamando a función *socket*, indirectamente dende o *user mode* chamando a funcións en librerias como *WinINet* ou directamente dende o modo *kernel*.

Despois de que unha aplicación chame a función **socket**, este aínda non está preparado para o seu uso. Os servidores deben suministrar a dirección local e o porto cando chamamos a *bind* (establecer o protocolo a empregar) e *listen*. Do mesmo modo, os clientes tamen teñen que proporcionar a dirección e o porto remotos cando chamamos a *connect* (non é obligatorio chamar a *bind*). O *socket* non vai funcionar ata non coñezca a IP e o porto.

En memory forensics os principais aspectos que nos van a interesar con respecto o **Networking** son os seguintes:

1. **Identificar oíntes non autorizados**: diferenciar entre os *sockets* de servidores lexítimos dos que empregan para aceptar conexións entrantes de atacantes.

2. **Revelar conexións remotas sospeitosas**.

3. **Ubicar sistemas con tarxetas de rede en modo *promiscuous***: este tipo de modo nas tarxetas de rede permite detectar máquinas que están na súa rede que poden estar intentando facer *sniff* cara ou dende outros sistemas, ou intentando realizar ataques estilo do *man-in-the-middle*.

4. **Detectar portos ocultos en sistemas activos**: moitos *rootkits* filtran portos e direccións IP enlazando APIs en sistemas activos.

5. **Reconstruir o historial do navegadores**: se se eliminan do disco ainda existe a posibilidade de poder atopalos na caché da memoria.

Para probar estas ferramentas vímonos na obriga de crear unha nova máquina virtual co sistema operativo **Windows XP** pois por exemplo o plugin **sockets** solo funciona para esta versión e para **Windows 2003 Server**. Con isto e con unhas das ferramentas vistas na sección de adquisición da memoria 2 obtemos un volcado de memoria co nome de *windowsXp.raw*.

Con este novo ficheiro podemos executar o plugin **sockets** co novo profile *WinXPSP2x64* e obtemos a seguinte resposta.

```
SSI@SSI-PC MINGW32 ~/Desktop/volatility-2.6 (1)/volatility-master
$ python vol.py -f windowsXp.raw --profile=WinXPSP2x64 sockets
Volatility Foundation Volatility Framework 2.6
Offset(V)              PID   Port  Proto Protocol        Address         Create Time
------------------ -------- ------ ------ --------------- --------------- -----------
0xfffffadfce646010      776   1027     17 UDP             0.0.0.0         2020-11-20 16:46:36 UTC+0000
0xfffffadfce0c4940      380    500     17 UDP             0.0.0.0         2020-11-20 16:45:47 UTC+0000
0xfffffadfce172da0        4    445      6 TCP             0.0.0.0         2020-11-20 16:45:36 UTC+0000
0xfffffadfce528a30      688    135      6 TCP             0.0.0.0         2020-11-20 16:45:38 UTC+0000
0xfffffadfce6e1c00      776   1036     17 UDP             0.0.0.0         2020-11-20 16:52:39 UTC+0000
0xfffffadfce6a15d0      820   1900     17 UDP             10.0.2.15       2020-11-20 16:45:47 UTC+0000
0xfffffadfce12e580      380      0    255 Reserved        0.0.0.0         2020-11-20 16:45:47 UTC+0000
0xfffffadfce30d3f0        4    139      6 TCP             10.0.2.15       2020-11-20 16:45:40 UTC+0000
0xfffffadfce121da0      776   1037     17 UDP             0.0.0.0         2020-11-20 16:52:39 UTC+0000
0xfffffadfce2e31e0        4    137     17 UDP             10.0.2.15       2020-11-20 16:45:40 UTC+0000
0xfffffadfce6b6430     1512   1026      6 TCP             127.0.0.1       2020-11-20 16:45:51 UTC+0000
0xfffffadfce4b0da0      776   1053     17 UDP             0.0.0.0         2020-11-20 16:52:39 UTC+0000
0xfffffadfce12fa40      380   1025      6 TCP             0.0.0.0         2020-11-20 16:45:51 UTC+0000
0xfffffadfce30a710      820   1900     17 UDP             127.0.0.1       2020-11-20 16:45:47 UTC+0000
0xfffffadfce17f6c0      380   4500     17 UDP             0.0.0.0         2020-11-20 16:45:47 UTC+0000
0xfffffadfce1737a0        4    445     17 UDP             0.0.0.0         2020-11-20 16:45:36 UTC+0000
0xfffffadfce25a430        4    138     17 UDP             10.0.2.15       2020-11-20 16:45:40 UTC+0000
```

Figura 4.11: Sockets - WindowsXp

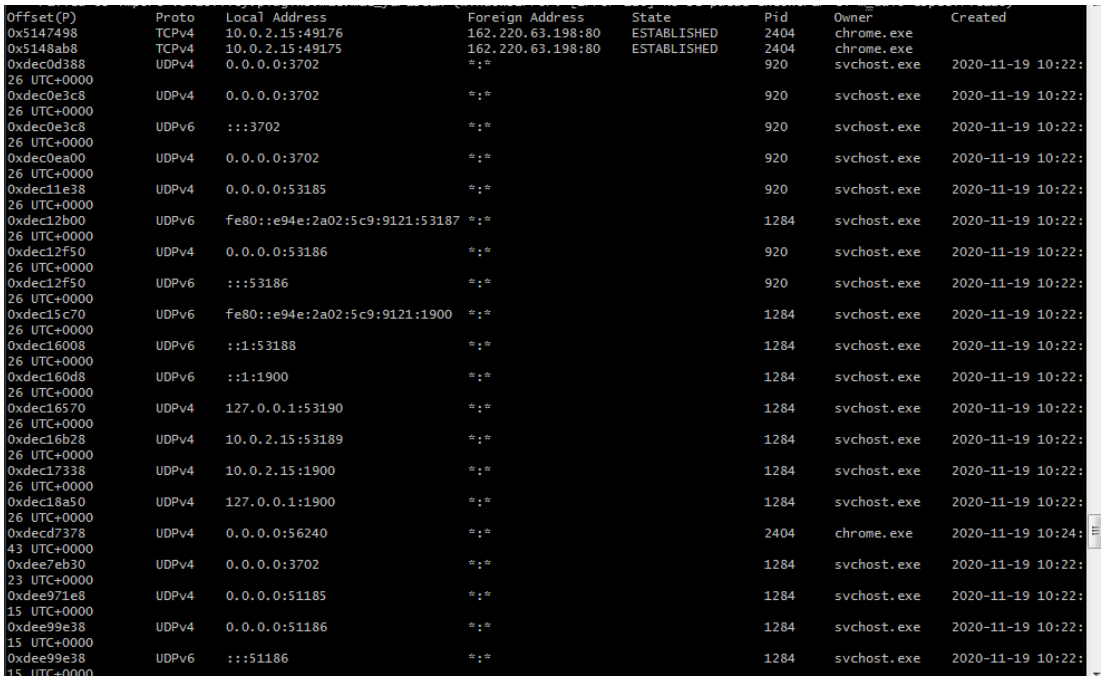
Na saída podemos ver varios campos, como son o ID do proceso propietario, o porto, o protocolo, a dirección IP e o momento de creación.
Para poder identificiar conexións e **sockets** que estean inactivos podemos recurrir ao plugins **sockscan** e **connscan**.
Este último plugin pode encontrar conexións anteriores que xa han rematado a maiores das activas, pódese ver na seguinte imaxe:

```
Offset(P)          Local Address             Remote Address            Pid
------------------ ------------------------- ------------------------- ---
0x00000000063bb6c0 10.0.2.15:1148            216.58.211.35:443         1852
0x00000000063d6d90 10.0.2.15:1163            142.250.74.238:443        1852
0x000000000678a500 10.0.2.15:1141            216.58.209.66:443         1852
0x0000000006a4b420 10.0.2.15:1064            8.241.105.126:80          1852
0x0000000006bcf500 10.0.2.15:1060            8.238.65.254:80           1852
0x0000000006c81a20 10.0.2.15:1140            216.58.209.66:443         1852
0x0000000006f054c0 10.0.2.15:1122            216.58.211.35:80          1852
0x0000000006f22d90 10.0.2.15:1065            216.58.211.237:443        1852
0x000000000701a8a0 10.0.2.15:1161            8.247.222.126:80          1852
0x000000000703fd90 10.0.2.15:1123            172.217.168.164:443       1852
```

Figura 4.12: Connscan - WindowsXp

Un plugin moi coñecido, non solo en **Volatility**, é **Netscan**. Iste emprégase en versións de **Windows Vista** e superiores. Mostra diversa información como puntos finais TCP, ointes TCP, puntos finais e ointes UDP. Permite distinguir entre *IPv4* e *IPv6*, imprime a IP local e remota (se corresponde), o porto local e remoto (se corresponde), a hora que se creou o *socket* ou cando se estableceu a conexión, así coma o seu estado actual (soamente TCP).

Netscan emprega pool tag scanning para localizar as estructuras *_TCP_ENDPOINT*, *_TCP_LISTENER* e *_UDP_ENDPOINT* na memoria.
Un exemplo da saída deste plugin sería:



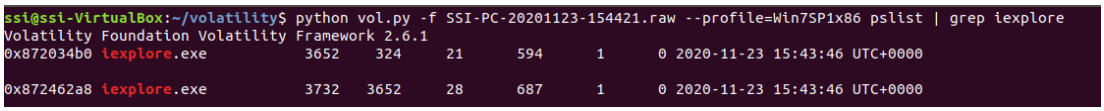
Offset(P)	Proto	Local Address	Foreign Address	State	Pid	Owner	Created
0x5147498	TCPv4	10.0.2.15:49176	162.220.63.198:80	ESTABLISHED	2404	chrome.exe	
0x5148ab8	TCPv4	10.0.2.15:49175	162.220.63.198:80	ESTABLISHED	2404	chrome.exe	
0xdec0d388	UDPv4	0.0.0.0:3702	*:*		920	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec0e3c8	UDPv4	0.0.0.0:3702	*:*		920	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec0e3c8	UDPv6	:::3702	*:*		920	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec0ea00	UDPv4	0.0.0.0:3702	*:*		920	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec11e38	UDPv4	0.0.0.0:53185	*:*		920	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec12b00	UDPv6	fe80::e94e:2a02:5c9:9121:53187	*:*		1284	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec12f50	UDPv4	0.0.0.0:53186	*:*		920	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec12f50	UDPv6	:::53186	*:*		920	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec15c70	UDPv6	fe80::e94e:2a02:5c9:9121:1900	*:*		1284	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec16008	UDPv6	::1:53188	*:*		1284	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec160d8	UDPv6	::1:1900	*:*		1284	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec16570	UDPv4	127.0.0.1:53190	*:*		1284	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec16b28	UDPv4	10.0.2.15:53189	*:*		1284	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec17338	UDPv4	10.0.2.15:1900	*:*		1284	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdec18a50	UDPv4	127.0.0.1:1900	*:*		1284	svchost.exe	2020-11-19 10:22:
26 UTC+0000							
0xdecd7378	UDPv4	0.0.0.0:56240	*:*		2404	chrome.exe	2020-11-19 10:24:
43 UTC+0000							
0xdee7eb30	UDPv4	0.0.0.0:3702	*:*		1284	svchost.exe	2020-11-19 10:22:
23 UTC+0000							
0xdee971e8	UDPv4	0.0.0.0:51185	*:*		1284	svchost.exe	2020-11-19 10:22:
15 UTC+0000							
0xdee99e38	UDPv4	0.0.0.0:51186	*:*		1284	svchost.exe	2020-11-19 10:22:
15 UTC+0000							
0xdee99e38	UDPv6	:::51186	*:*		1284	svchost.exe	2020-11-19 10:22:

Figura 4.13: Netscan

Como ben deciamos grazas a **Volatility** podemos coñecer o historial do noso navegador aínda se o borramos do disco. Para isto vamos botar man dun plugin que xa vimos anteriormente **pslist** e de **yarascan**.

Nos tivemos problemas para instalar correctamente **yarascan** en **Windows 7** polo que recurrimos a instalación de **Volatility** e os seus diferentes plugins nunha máquina virtual **Ubuntu 18.04**, pódese ver esta instalación no anexo C.

O primeiro que temos que facer e buscar os procesos do navegador, no noso caso buscamos polo proceso do programa **Internet Explorer** de **Windows**.



```
ssi@ssi-VirtualBox:~/volatility$ python vol.py -f SSI-PC-20201123-154421.raw --profile=Win7SP1x86 pslist | grep iexplore
Volatility Foundation Volatility Framework 2.6.1
0x872034b0 iexplore.exe           3652    324     21      594      1      0 2020-11-23 15:43:46 UTC+0000

0x872462a8 iexplore.exe           3732   3652     28      687      1      0 2020-11-23 15:43:46 UTC+0000
```

Figura 4.14: Pslist - Internet History

Agora que coñecemos os **PID** (3652,3732), podemos usar o plugin *yarascan* para obter unha idea inicial de onde se pode ubicar o arquivo *index.dat* (contén moita información acerca do progama *Internet Explorer*). A firma deste arquivo contén a palabra *Client UrlCache* polo que podemos empregar isto para iniciar a búsqueda.

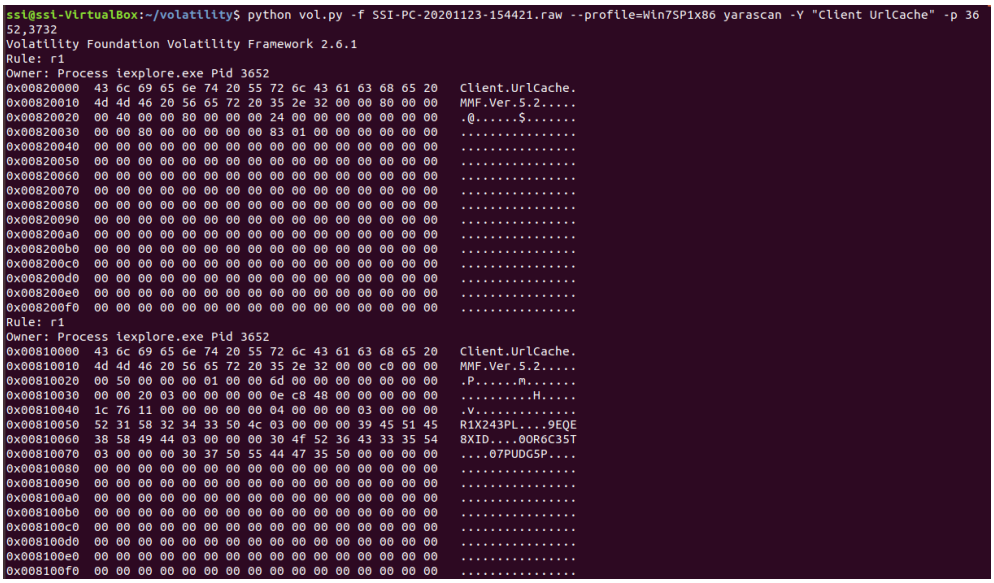


Figura 4.15: yarascan - Client UrlCache

Podemos ver que a firma *Client UrlCache* atópase en máis dunha ubicación dentro da memoria do proceso con PID 3652. Sen embargo para buscar simplemente entradas no historial non é necesario analizar o arquivo *index.dat*. Basta con buscar simplemente o historial individual de rexistros que empecen con *URL*,*LEAK* ou *REDR* (LEAK: termino de Microsoft para un error, REDR: redireccións) como mostra a seguinte imaxen:

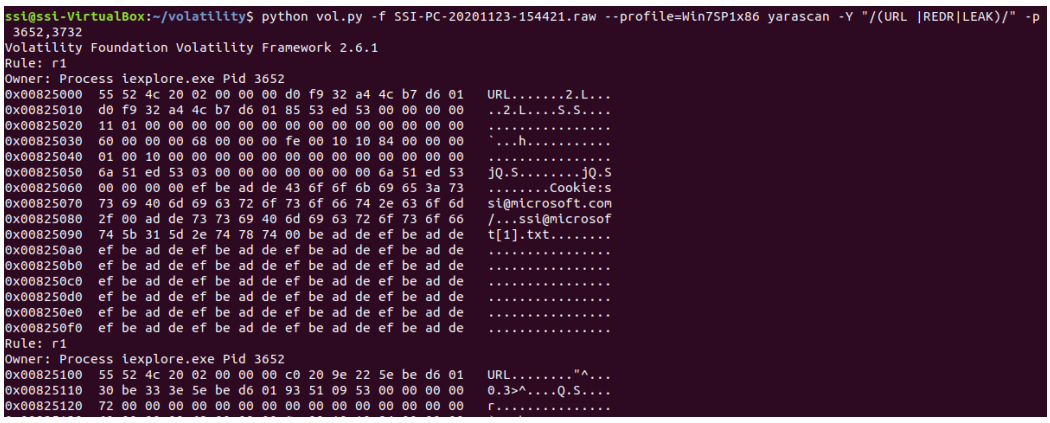


Figura 4.16: yarascan - URL REDR LEAK

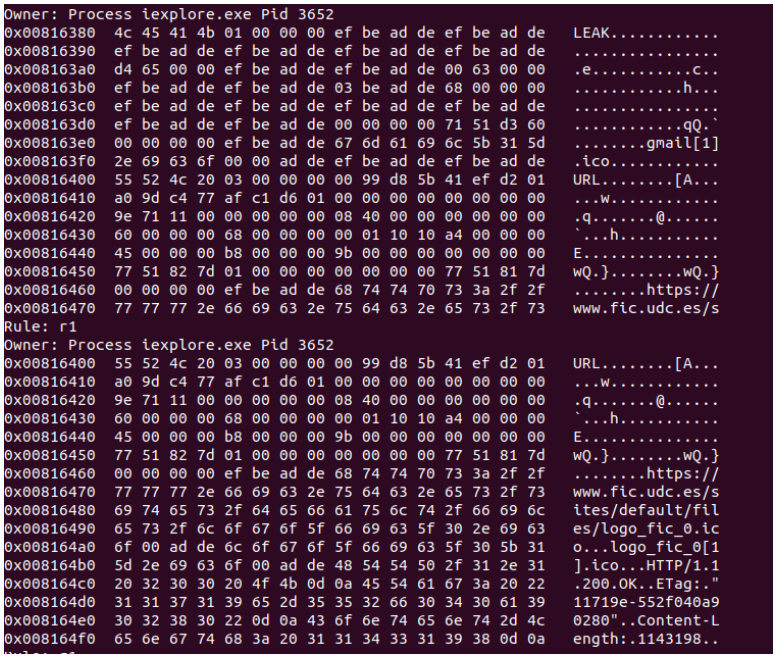


Figura 4.17: yarascan

Podemos ver no exemplo anterior que atopamos a dirección *https://www.fic.udc.es*.

Temos outra opción de poder ver a información dos navegadores que é empregar o plugin **iehistory**. Este plugin devolve un formato de saída diferente cunha mellor automatización e análise dos resultados. Tamén podemos devolver un formato en *CSV* pasándolle a opción *–output=csv*.

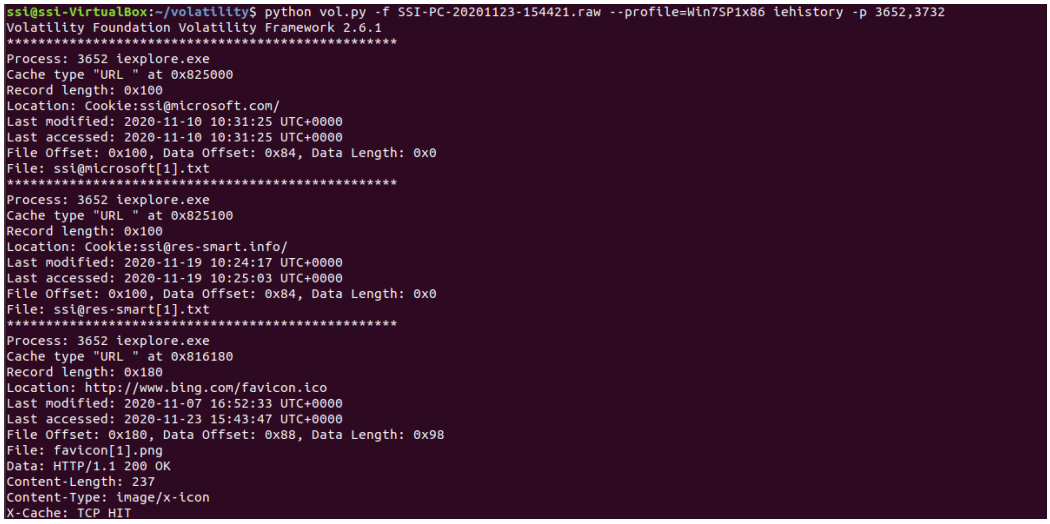


Figura 4.18: iehistory

Por último para rematar esta parte de **Networking** vamos a ver como podemos obter o ficheiro **hosts** dun volcado de memoria. Este ficheiro é o encargado de almacenar nomes de *host* coas súas correspondentes **IPs**.
Para iso primeiro temos que ver onde está ubicado esté arquivo na memoria obtendo o seu desprazamento, por iso empregamos o plugin **filescan**.

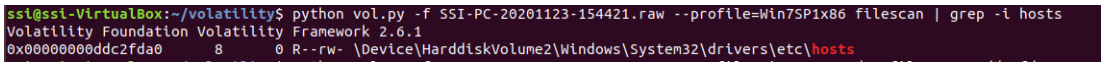


Figura 4.19: filescan - hosts

A continuación temos que extrar este arquivo do volcado, para iso podemos empregar o plugin **dumpfiles**, indicando o directorio no que o queremos gardar (*outdir*).

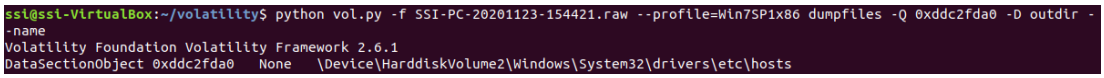


Figura 4.20: dumpfiles - hosts

Por último e co comando nativo de **Ubuntu**, *strings*, que mostra as cadenas de caracteres imprimibles que hai nos ficheiros, podemos ver o contido do ficheiro *hosts* e así poder coñecer as resolucións DNS.

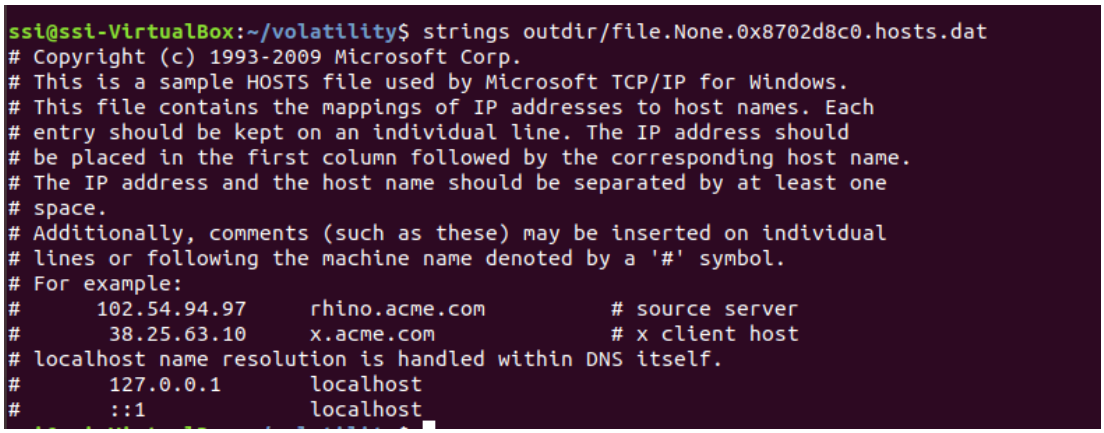


Figura 4.21: strings - hosts

4.5 Windows GUI

O subsistema da interfaz gráfica de usuario (*GUI*) de **Windows** é responsable de administrar a entrada do usuario, como os movementos do ratón e pulsacións de teclas. Ademais dibuxa a superficie de visualización, presenta ventás, botóns e menús, ademais de proporcionar o aislamento necesario para admitir varios usuarios simultáneos. Por ter esta importancia, moitos *malware* modifican a memoria da *GUI* para infectar o noso equipo.

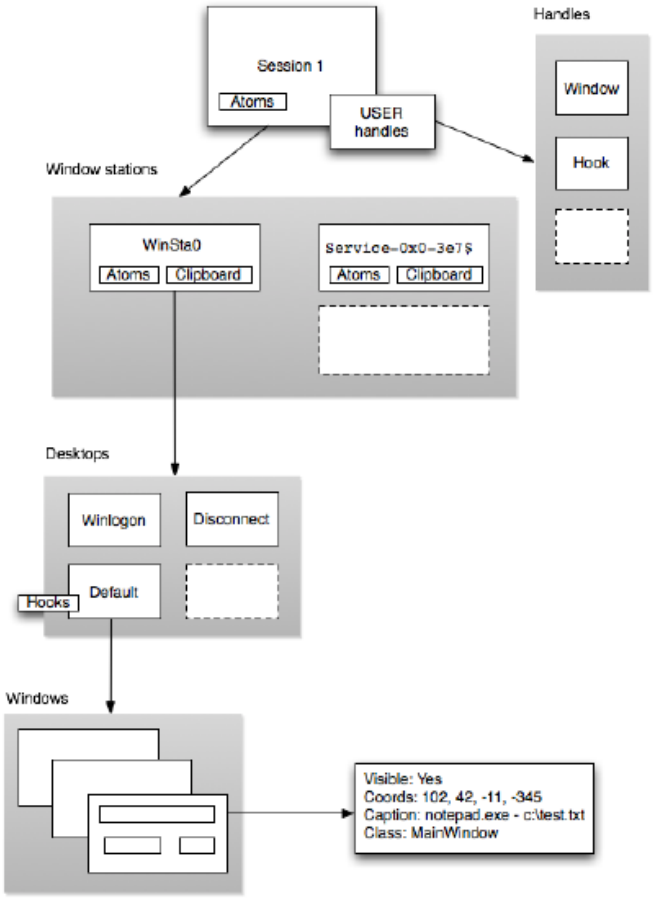


Figura 4.22: GUI - schema

Como vemos na figura anterior, o contedor *Session* é a capa máis externa, representa o entorno do inicio de sesión. Estas sesións teñen un ID único e creanse cando os usuarios inician sesión, polo que cada sesión está ligada a un usuario en particular e os recursos vinculados a sesión pódense atribuir a accións realizadas polo usuario. Estes recursos inclúen unha tabla atómica (grupo de cadenas compartidas globalmente entre as aplicacións da sesión), unha ou máis estacións de ventá e unha tabla de identificadores para os obxetos USER (*handle table*).

As aplicacións que requiren a entrada do usuario execútanse nunha *Windows Station* como

é o caso de *WinSta0*. Os servizos que non requiren desta participación do usuario (servizos en segundo plano) executan *non-interactive windows stations*. Cada unha destas estacións contén unha tabla *Atom*, un portapapeis (*clipboard*) e un ou máis escritorios.

Un escritorio contén os obxetos da interfaz de usuario, como ventás, menús, ...
Estas ventás poden ser visibles ou invisibles, teñen un conxunto de coordenadas de pantalla, un procedemento de ventá (a función que se executa cando os mensaxes da ventá son recibidos), unha lenda ou un título opcional e unha clase asociada.

Ao analizar as ventás podemos determinar que atacantes ou víctimas estaban vendoas no momento do volcado de memoria, ou que aplicacións GUI se habían executado no pasado.

O primeiro plugin que imos a ver é **sessions**. Este plugin separa os procesos segundo a súa propia sesión, pero non implica que poidamos vinculalas cada sesión a un usuario diferente. No seguinte exemplo vemos como existen duas sesións con IDs 0 e 1. A 0 son os procesos creados polo propio sistema e cos do ID 1 son os creados polo usuario.

```
ssi@ssi-VirtualBox:~/volatility$ python vol.py -f SSI-PC-20201123-154421.raw --profile=Win7SP1x86 sessions
Volatility Foundation Volatility Framework 2.6.1
**************************************************
Session(V): 8fa66000 ID: 0 Processes: 23
PagedPoolStart: 80000000 PagedPoolEnd ffbfffff
 Process: 336 csrss.exe 2020-11-23 15:42:40 UTC+0000
 Process: 396 wininit.exe 2020-11-23 15:42:41 UTC+0000
 Process: 492 services.exe 2020-11-23 15:42:42 UTC+0000
 Process: 500 lsass.exe 2020-11-23 15:42:42 UTC+0000
 Process: 508 lsm.exe 2020-11-23 15:42:42 UTC+0000
 Process: 620 svchost.exe 2020-11-23 15:42:43 UTC+0000
 Process: 684 VBoxService.ex 2020-11-23 15:42:44 UTC+0000
 Process: 740 svchost.exe 2020-11-23 15:42:44 UTC+0000
 Process: 832 svchost.exe 2020-11-23 15:42:44 UTC+0000
 Process: 896 svchost.exe 2020-11-23 15:42:44 UTC+0000
 Process: 924 svchost.exe 2020-11-23 15:42:44 UTC+0000
 Process: 1024 audiodg.exe 2020-11-23 15:42:44 UTC+0000
 Process: 1068 svchost.exe 2020-11-23 15:42:45 UTC+0000
 Process: 1168 svchost.exe 2020-11-23 15:42:45 UTC+0000
 Process: 1312 spoolsv.exe 2020-11-23 15:42:46 UTC+0000
 Process: 1340 svchost.exe 2020-11-23 15:42:46 UTC+0000
 Process: 1436 svchost.exe 2020-11-23 15:42:46 UTC+0000
 Process: 268 taskeng.exe 2020-11-23 15:43:14 UTC+0000
 Process: 328 GoogleCrashHan 2020-11-23 15:43:16 UTC+0000
 Process: 1004 SearchIndexer. 2020-11-23 15:43:21 UTC+0000
 Process: 212 wmpnetwk.exe 2020-11-23 15:43:21 UTC+0000
 Process: 2608 WmiPrvSE.exe 2020-11-23 15:43:24 UTC+0000
 Process: 4012 SearchFilterHo 2020-11-23 15:43:56 UTC+0000
 Image: 0x86753118, Address 940a0000, Name: win32k.sys
 Image: 0x84f724b0, Address 94300000, Name: TSDDD.dll
**************************************************
Session(V): 8faf7000 ID: 1 Processes: 21
PagedPoolStart: 80000000 PagedPoolEnd ffbfffff
 Process: 388 csrss.exe 2020-11-23 15:42:41 UTC+0000
 Process: 432 winlogon.exe 2020-11-23 15:42:41 UTC+0000
 Process: 352 dwm.exe 2020-11-23 15:43:14 UTC+0000
 Process: 324 explorer.exe 2020-11-23 15:43:14 UTC+0000
 Process: 480 taskhost.exe 2020-11-23 15:43:14 UTC+0000
 Process: 648 VBoxTray.exe 2020-11-23 15:43:15 UTC+0000
 Process: 2108 chrome.exe 2020-11-23 15:43:22 UTC+0000
 Process: 2140 chrome.exe 2020-11-23 15:43:22 UTC+0000
 Process: 2380 chrome.exe 2020-11-23 15:43:23 UTC+0000
 Process: 2448 chrome.exe 2020-11-23 15:43:23 UTC+0000
 Process: 2844 chrome.exe 2020-11-23 15:43:25 UTC+0000
 Process: 3092 chrome.exe 2020-11-23 15:43:30 UTC+0000
 Process: 3456 chrome.exe 2020-11-23 15:43:42 UTC+0000
 Process: 3504 chrome.exe 2020-11-23 15:43:42 UTC+0000
 Process: 3652 iexplore.exe 2020-11-23 15:43:46 UTC+0000
 Process: 3732 iexplore.exe 2020-11-23 15:43:46 UTC+0000
 Process: 3992 SearchProtocol 2020-11-23 15:43:56 UTC+0000
```

Figura 4.23: sessions

Como ven comentabamos antes, as *Windows Stations* actuan como límites de seguridade para procesos e escritorios. Dende un pusto de vista forense, analizando estas estacións podemos detectar aplicacións que realizan **snooping** ao portapapeis ou tamén podemos determinar o uso do portapapeis como a súa frecuencia de uso, a disponibilidade de formatos....

Grazas ao comando **wndscan** podemos analizar a frecuencia de uso do *clipboard* (ferramenta a cal nos permite almacenar temporalmente información de calqueira tipo).
Na seguinte imaxen podemos ver como a *window station*, **WinSta0**, para a sesión 1 o usuario copiou 4 elementos ao portapaeis empregando os formatos descritos na casilla *Formats*.

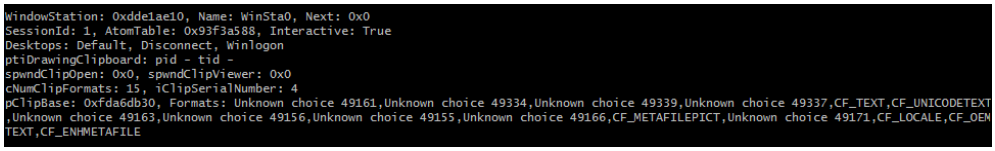


Figura 4.24: wndscan

Por outra banda, moitos *malwares* intentan mirar as operacións do portapapeis para intentar roubar por exemplo credenciais. Hai duas formas de acceder a estes datos. Mediante a API *setClipboardData* (empregase a hora de copiar un elemento ao *clipboard*) ou mediante a función *GetClipboardData* que é o contrario que a anterior, as aplicacións chaman a esta en resposta a operacións de pegado.

Microsoft sen embargo recomenda acceder aos datos tan pronto como se copian ao portapapeles mediante un visor de portapapeis chamado *SetClipboarViewer* ou un listener de formato chamado *AddClipboarFortmatListener*. Estas funcións permiten que as aplicacións reciban notificacións a través de mensaxes *WM_DRAWCLIPBOARD* sempre que cambie o contido do portapapeis e logo elas xa poden abrilo e consultar os datos cando elas decidan.

Podemos entón saber que aplicacións teñen acceso ao clipboard coñecendo aquelas que empregan o API para acceder a este (*CLIPBRDWNDCLASS*), esto podémolo coñecer co plugin **wintree** que o explicaremos máis adiante.

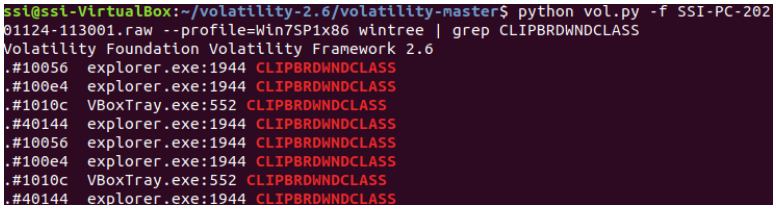


Figura 4.25: wintree - clipboard

Pasamos agora aos escritorios (**Desktops**). Un escritorio é un contedor para ventás de aplicacións e obxetos da interfaz de usuario. O *Malware* pode empregar estos **Desktops** para por exemplo lanzar aplicacións en escritorios alternativos, polo que o usuario non é capaz de velas, ou que o *ransomware* bloquea aos usuarios do seu propio escritorio e ata que se pague unha determinada tarifa non é capaz de volver.

Para ver os escritorios activos e a información acerca destes podemos empregar o plugin **deskscan**. Este plugin busca estacións ventá de **Windows** e logo recorre a lista de escritorios *rpdeskList*. O resultado é o seguinte:

```
ssi@ssi-VirtualBox:~/volatility-2.6/volatility-master$ python vol.py -f SSI-PC-20201124-113001.raw --profile=Win7SP1x86 deskscan
Volatility Foundation Volatility Framework 2.6
**************************************************
Desktop: 0xdddbe048, Name: msswindowstation\mssrestricteddesk, Next: 0x0
SessionId: 0, DesktopInfo: 0xfe600578, fsHooks: 0
spwnd: 0xfe600618, Windows: 14
Heap: 0xfe600000, Size: 0x80000, Base: 0xfe600000, Limit: 0xfe680000
**************************************************
Desktop: 0xdde086c8, Name: WinSta0\Default, Next: 0x86e28170
SessionId: 0, DesktopInfo: 0xfea00578, fsHooks: 32
spwnd: 0xfea00618, Windows: 14
Heap: 0xfea00000, Size: 0xc00000, Base: 0xfea00000, Limit: 0xff600000
 1324 (spoolsv.exe 1312 parent 488)
 1996 (spoolsv.exe 1312 parent 488)
 1976 (spoolsv.exe 1312 parent 488)
 1952 (spoolsv.exe 1312 parent 488)
 1336 (spoolsv.exe 1312 parent 488)
 1316 (spoolsv.exe 1312 parent 488)
 472 (csrss.exe 332 parent 316)
 464 (csrss.exe 332 parent 316)
 388 (wininit.exe 384 parent 316)
**********************************************
Desktop: 0xdde28170, Name: WinSta0\Disconnect, Next: 0x86e29f78
SessionId: 0, DesktopInfo: 0xff660578, fsHooks: 0
spwnd: 0xff660618, Windows: 1
Heap: 0xff660000, Size: 0x10000, Base: 0xff660000, Limit: 0xff670000
**********************************************
Desktop: 0xdde29f78, Name: WinSta0\Winlogon, Next: 0x0
SessionId: 0, DesktopInfo: 0xff640578, fsHooks: 0
spwnd: 0xff640618, Windows: 2
Heap: 0xff640000, Size: 0x20000, Base: 0xff640000, Limit: 0xff660000
**********************************************
Desktop: 0xdde3fd98, Name: Service-0x0-3e7$\Default, Next: 0x0
SessionId: 0, DesktopInfo: 0xff670578, fsHooks: 0
spwnd: 0xff670618, Windows: 4
Heap: 0xff670000, Size: 0x80000, Base: 0xff670000, Limit: 0xff6f0000
 3288 (svchost.exe 936 parent 488)
 2780 (SearchIndexer. 2508 parent 488)
 1520 (svchost.exe 892 parent 488)
 2572 (SearchIndexer. 2508 parent 488)
```

Figura 4.26: deskscan

Nesta saída vemos varios tipos de ecritorio, os máis importantes son, **Winlogon** que representa o login de inicio de sesión, que introducimos ao iniciar sesión en **Windows**. Se o usuario e contrasinal son correctos pásase ao escritorio **Default**. Os únicos subprocesos neste escritorio pertencen a *winlogon.exe* polo que se se apreciasen outros procesos poderíanos indicar que se nos está a intentar a roubar as credenciais de inicio de sesión. O outro é o escritorio **Default** que ten como característica fundamental que a cantidade de ventás neste escritorio é moito maior que nas demais.

As **Windows** como ben deciamos antes son contedores para botóns, barras de desprazamento, áreas de texto/edición. Desempeñan un papel tan importante na interfaz do usuario que non sorprende que o *malware* encontrase numerosas formas de explotar estas ventás.

Alguns dos usos destas ventás son por exemplo empregalas para detectar ferramentas de seguridade/monitoreo, deshabilitar antivirus, monitorear USB, simular interaccións do usuario como clicks do ratón ou pulsación de teclas....

Co plugin **wintree** podemos ver as relacións que existen entre as ventás pais e fillos nun escritorio.

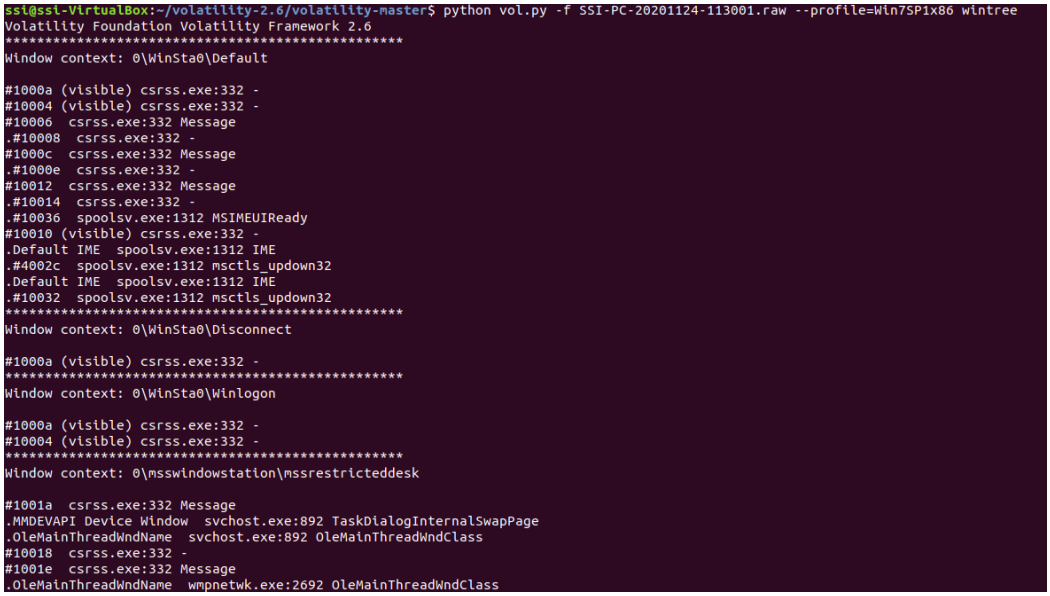


Figura 4.27: wintree

Outro plugin importante é **screenshot** que enumera as pantallas para cada escritorio. Toma as coordenadas de cada ventana e debuxa os rectángulos con *Python Imaging Library* (PIL). Polo tanto para empregar este plugin temos que instalar esta librería mediante o comando # **pip install pil**. Unha vez feito isto podemos executar o seguinte.

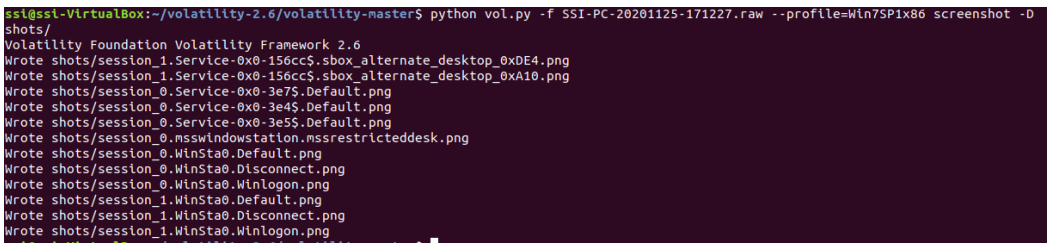


Figura 4.28: screenshot

É decir no directorio *shots* imos xerar unhas capturas de como sería a disposición das ventás nos diferentes escritorios. Nas seguintes imaxes podemos comparar como se ve o noso escritorio no momento do volcado de memoria e a continuación a representación obtida polo plugin **screenshot**.

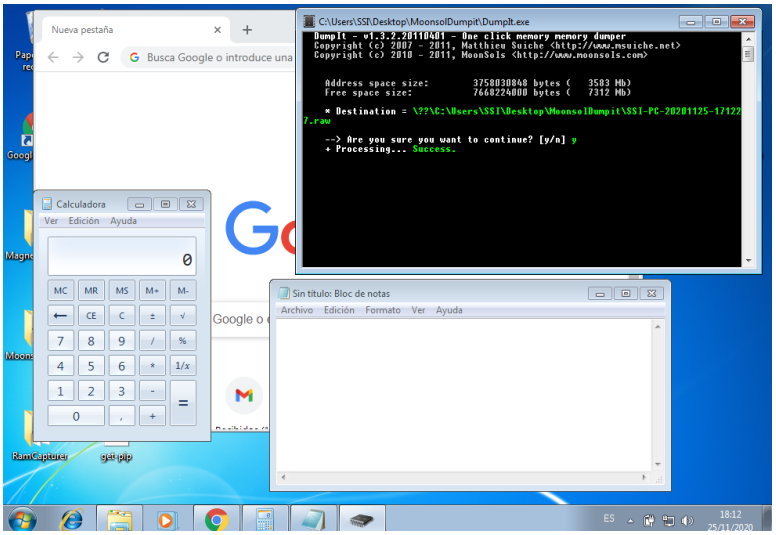


Figura 4.29: Disposición ventanas Windows

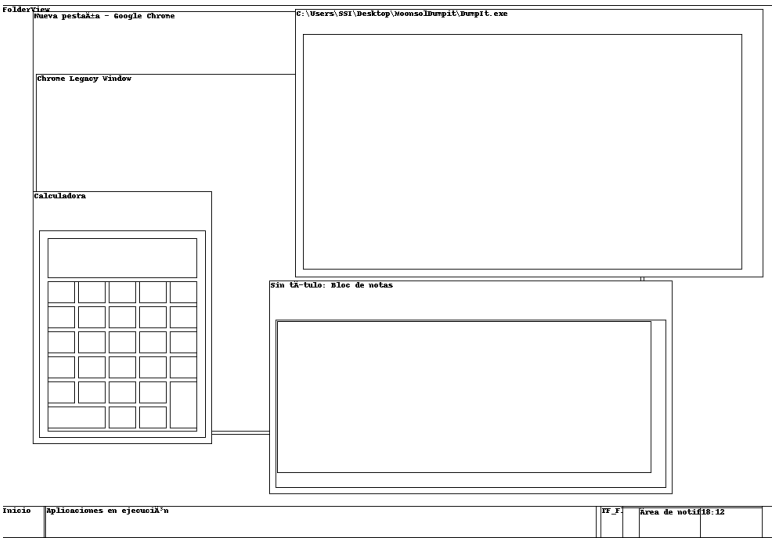


Figura 4.30: Representacion - screenshot

Podemos extraer tamén a información do *Clipboard*. Podemos extraer o contido dos datos do portapapeis da *RAM*, que en algúns casos pode conter información confidencial como contrasinais, nomes de usuario e demais información.

```
ssi@ssi-VirtualBox:~/volatility-2.6/volatility-master$ python vol.py -f SSI-PC-20201124-113001.raw --profile=Win7SP1x86 clipboa
rd
Volatility Foundation Volatility Framework 2.6
Session    WindowStation Format                Handle Object     Data
---------- ------------- ----------------- ---------- ---------- ------------------------------------------------
         1 WinSta0       0xc009L             0x60185 0xfe8405e8
         1 WinSta0       0xc0b6L           0x2601b5 0xfe705c48
         1 WinSta0       0xc0bbL             0xa01c1 0xfe6e5c48
         1 WinSta0       0xc0b9L             0x501d1 0xffb8d298
         1 WinSta0       CF_TEXT             0x80171 0xffa19178 122312414141
         1 WinSta0       CF_UNICODETEXT    0x1901b7 0xfdf17168 122312414141
         1 WinSta0       0xc00bL             0xd01d3 0xfe6df5f0
         1 WinSta0       0xc004L             0xf0189 0xfe6745f0
         1 WinSta0       0xc003L             0x901d5 0xfdf5d950
         1 WinSta0       0xc00eL             0xb01bd 0xfe77e358
         1 WinSta0       CF_METAFILEPICT   0xc1509aa ----------
         1 WinSta0       0xc013L             0x101d7 0xfe6ed0e0
         1 WinSta0       CF_LOCALE           0xe01a3 0xfdf928f0
         1 WinSta0       CF_OEMTEXT              0x1 ----------
         1 WinSta0       CF_ENHMETAFILE          0x3 ----------
```

Figura 4.31: Clipboard

Podemos ver que o usuario con sesión 1 colocou unha cadea *UNICODE* e *TEXT* no portapapeis con valor *122312414141*. Para os demais valores, que están escritos en hexadecimal podemos leelo pasandolle **-v** ao plugin **clipboard**.

4.6 Disk artifacts in Memory

Nesta sección vámonos centrar en analizar o sistema de arquivos *NFTS* de **Windows**. Os usuarios e o sistema operativo están constantemente leendo, escribindo e elimininando arquivos, polo que estas accións deixan rastro na memoria. Esto resúltanos útil para facer un primeiro análise forense do disco só mirando a memoria *RAM* que é notablemente menor en tamaño que a memoria completa dun disco.

En **NTFS** almacénase todo como un ficheiro. Isto inclúe ficheiros especiais de metadatos empregado para organizar e trackear outros ficheiros. **MFT** é un ficheiro especial situado na raiz do sistema de ficheiros, que almacena información crítica sobre todos os demais ficheiros da partición. O **MFT** contén unha entrada para cada ficheiro e directorio do sistema de ficheiros. Cada entrada, que ten un tamaño máximo de 1024 bytes, contén información como o nome, o tipo e as localizacións no disco onde se poden atopar os seus datos. Tamen se inclúen marcas de tempo que conteñen os momentos de creación, modificación ou de acceso de ese ficheiro.

Mediante o plugin **mftparser** podemos extraer entradas **MFT** da memoria. Podemos ver a execución de ese plugin onde gardamos a sua saída en *mftverbose.txt*.

```
ssi@ssi-VirtualBox:~/volatility-2.6/volatility-master$ python vol.py -f SSI-PC-20201124-113001.raw --profile=Win7SP1x86 mftparser --ou
tput-file=mftverbose.txt
Volatility Foundation Volatility Framework 2.6
Outputting to: mftverbose.txt
Scanning for MFT entries and building directory, this can take a while
```

Figura 4.32: mftparser

Neste novo ficheiro creado podemos ver os diferentes ficheiros e arquivos do sistema xunto con información adicional acerca deles como é a fecha de creación, a fecha de modificación, cando se accedeu por última vez e a ruta completa de onde está gardado no sistema.

```
$FILE_NAME
Creation                       Modified                       MFT Altered                    Access Date                    Name/Path
------------------------------ ------------------------------ ------------------------------ ------------------------------ ---------
2020-11-24 11:30:03 UTC+0000 2020-11-24 11:30:03 UTC+0000   2020-11-24 11:30:03 UTC+0000   2020-11-24 11:30:03 UTC+0000
Users\SSI\Desktop\MOONSO~1\SSI-PC-20201124-113001.raw

$DATA
```

Figura 4.33: mftverbose - mftparser

4.7 Command History

A diferencia do *bash shell* en UNIX, o shell de **Windows** non ten a capacidade de rexistrar os comandos introducidos nun arquivo histórico. Esto imposibilita analizar as actividades de usuarios non autorizados en función do uso do shell de comandos. Pero grazas a **Memory Forensics** somos capaces de:

- Recuperar comandos de shells rematados, incluso despois de que o proceso rematase.
- Extraer búferes de entrada e saída da consola, é dicir, podemos imprimir a resposta que deu o sistema ante determinados comandos.
- Podemos enumerar e traducir **alias**. Os *alias* permiten asignarnos un string a outro string, por exemplo, podemos asignarlle *abc* ao comando *c:_Windows_Malware.exe –port=8000 –host=1.2.3.4*. Cando executamos na terminal *abc* en realidade estamos executando o *target* string que configuramos.
- Reconstruir a actividade do usuario.

Cmd.exe é unha aplicación de consola (unha aplicación non GUI que se executa no escritorio), aínda que si que ten algunhas funcionalidades de GUI como minimizar o tamaño da ventana, copiar e pegar do *clipboard*.
Antes de **Windows 7** desto encargábase o proceso *csrss.exe* que se executa con privilexios do sistema, sen embargo, esto era aproveitado por un exploit chamado *Malicious Window Abuse*.
A partir de **Windows 7**, microsoft introduxo o proceso de host da consola (**conhost.exe**) que asume as mismas reponsabilidades que **csrss.exe** pero execútase cos permisos do usuario que iniciou a shell de comandos.

En definitiva, os comandos ingresados por *cmd.exe* son procesados por *csrss.exe* ou *conhost.exe*, según a plataforma de destino. En outras palabras *cmd.exe* é solo o cliente nunha arquitectura cliente-servidor.

O plugin **cmdscan** vai atopar todas as instancias do historial de comandos nas páxinas de memoria en propiedad de *csrss.exe* ou *conhost.exe*.

```
ssi@ssi-VirtualBox:~/volatility-2.6/volatility-master$ python vol.py -f SSI-PC-20201204-094659.raw --profile=Win7SP1x86 cmdscan
Volatility Foundation Volatility Framework 2.6
**************************************************
CommandProcess: conhost.exe Pid: 2204
CommandHistory: 0x3f1ce0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 5 LastAdded: 4 LastDisplayed: 4
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
Cmd #0 @ 0x3ecc78: systeminfo
Cmd #1 @ 0x3ecc98: ipconfig
Cmd #2 @ 0x3eccb8: cd Desktop
Cmd #3 @ 0x3f1dd8: cd MoonsolDumpit
Cmd #4 @ 0x3eccd8: DumpIt.exe
Cmd #8 @ 0x310034: >@1
Cmd #9 @ 0x310038: 1
Cmd #18 @ 0x31002d: ?
Cmd #19 @ 0x300030: ??
Cmd #36 @ 0x3c00c4: >???<???<
Cmd #37 @ 0x3ec790: =?<???????>
**************************************************
CommandProcess: conhost.exe Pid: 2692
CommandHistory: 0x271828 Application: DumpIt.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
Cmd #36 @ 0x2400c4: '?'?$???$
Cmd #37 @ 0x26d360: &?&?&?
```

Figura 4.34: cmdscan

Vemos que temos dous procesos *conhost.exe*. No primeiro contamos 5 comandos introducidos *CommandCount*. Podemos apreciar que o máximo número de comandos que pode amosar son 50 *CommandCountMax*. Por último podemos ver os comandos que o usuario introduciu: *systeminfo*, *ipconfig*, *cd Desktop*, ...

O seguinte plugin **consoles** é moi parecido ao anterior, pero este amosa a saída que realizou o sistema aos comandos introducidos. Isto pódenos axudar a comprender moito mellor as accións que puido realizar un atacante. Comeza mostrando o historial de comandos, pero logo volca todo o buffer de pantalla.

```
ssi@ssi-VirtualBox:~/volatility-2.6/volatility-master$ python vol.py -f SSI-PC-20201204-094659.raw --profile=Win7SP1x86 consoles
Volatility Foundation Volatility Framework 2.6
**************************************************
ConsoleProcess: conhost.exe Pid: 2204
Console: 0x7c81c0 CommandHistorySize: 50
HistoryBufferCount: 3 HistoryBufferMax: 4
OriginalTitle: S?mbolo del sistema
Title: S?mbolo del sistema - DumpIt.exe
AttachedProcess: cmd.exe Pid: 2196 Handle: 0x5c
----
CommandHistory: 0x3f3050 Application: ipconfig.exe Flags:
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x0
----
CommandHistory: 0x3f2f58 Application: systeminfo.exe Flags:
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x0
----
CommandHistory: 0x3f1ce0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 5 LastAdded: 4 LastDisplayed: 4
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
```

Figura 4.35: consoles 1

```
CommandHistory: 0x3f1ce0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 5 LastAdded: 4 LastDisplayed: 4
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
Cmd #0 at 0x3ecc78: systeminfo
Cmd #1 at 0x3ecc98: ipconfig
Cmd #2 at 0x3eccb8: cd Desktop
Cmd #3 at 0x3f1dd8: cd MoonsolDumpit
Cmd #4 at 0x3eccd8: DumpIt.exe
----
Screen 0x3d6ac0 X:80 Y:300
Dump:
Microsoft Windows [Versi?n 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\SSI>systeminfo

Nombre de host:                           SSI-PC
Nombre del sistema operativo:             Microsoft Windows 7 Home Basic
Versi?n del sistema operativo:            6.1.7601 Service Pack 1 Compilaci?n 7
601
Fabricante del sistema operativo:         Microsoft Corporation
Configuraci?n del sistema operativo:      Estaci?n de trabajo independiente
Tipo de compilaci?n del sistema operativo: Multiprocessor Free
Propiedad de:                             SSI
Organizaci?n registrada:
Id. del producto:                         00346-339-0000007-85160
Fecha de instalaci?n original:            07/11/2020, 12:05:03
Tiempo de arranque del sistema:           04/12/2020, 10:45:52
Fabricante del sistema:                   innotek GmbH
Modelo el sistema:                        VirtualBox
Tipo de sistema:                          X86-based PC
```

Figura 4.36: consoles 2

Capítulo 5

Conclusións

Como acabamos de ver, o análise forense é un ámbito que resulta moi importante no ámbito da informática e que nos últimos anos está en auxe. Grazas a este podemos realizar unha identificación, preservación, análise e presentación de datos que poden ser presentados por exemplo nun proceso legal.

A maiores permítenos realizar una ánalise das consecuencias que produciu un ataque, averiguar quen foi o autor e detectar as debilidades que ten o sistema para arranxalas e poder evitar ataques futuros.

No noso traballo centrámonos máis nas partes prácticas, como poden ser a adquisición da memoria ou o uso de ferramentas para realizar os análises, pero outras partes como son a cadena de custodia ou a intregidade dos datos son partes moi importantes dentro deste ámbito que podería servir para posibles liñas futuras de investigación.

A maiores centrámonos solo nos sistemas operativos **Windows** máis concretamente en **Windows 7**. Por falta de extensión non poidemos abordar outros sistemas como serían **Linux**, **IOS** ou versións máis recentes de **Windows** polo que sería unha moi boa idea continuar analizando estes sistemas no futuro. Tamén fixemos especial énfasis na ferramenta **Volatility** que ofrece unha gran capacidade para o análise forense, pero no mercado atópase unha gran variedade de *frameworks*, moitos deles de pago.

Con isto decir que o ámbito do peritaxe informático é moi extenso e levaría moitos anos facerse un experto neste sector, pero grazas a este traballo puidémonos facer unha moi boa introducción neste tema e percatarnos da importancia que ten, o cal nos vindeiros anos creemos que vai tomar unha maior relevancia.

Apéndices

Apéndice A

Ferramenta OSForensics

Neste capitulo vamos facer unha breve introdución a ferramenta **OSForensics** e a ver, e probar, os aspectos que a nós nos pareceron máis interesantes das múltiples funcionalidades que esta aplicación pode ofrecer.

OSForensics[22] é unha ferramenta forense que nos permite extraer e analizar datos dixitais dunha maneria efeciente e moi sinxela. Esta aplicación inclue unha serie de ferramentas que en conxunto nos aporta casi todas as capacidades forenses dixitais, como son, a adquisición, extracción, análise, análise do correo electrónico, imaxenes de datos, restauración de imaxenes....

A descarga podemola facer dende o seguinte enlace, e a última versión (cando a probamos), a 8.0 admite os principais sistemas operativos **Windows** e **Windwos Server** exceptuando o *XP*. O único problema que presenta esta ferramenta é que é de pago, pero dannos a oportunidade de probala gratuitamente durante un período de 30 días e con algunhas limitacións.

Nós vamos a probar esta aplicación sobre un sistema operativo **Windows 10** de 64 bits.

A.1 Auto-adquisición de probas

A opción *Auto Triage* permitenos realizar varios analises simultáneos dunha maneira semi-automática dun sistema dado. A hora de comezar o análise solo bastará con proporcionarlle o disco ou extraible que queremos analizar, onde se van gardar os resultados e as áreas que desexamos buscar.

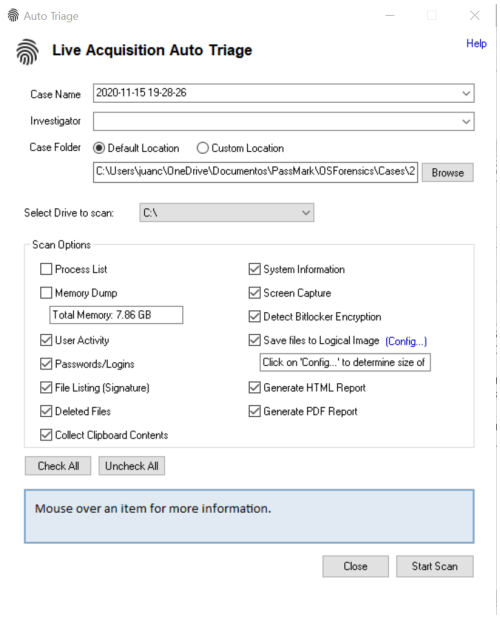


Figura A.1: OSForensics - Auto triage 1

Unha vez finalizado podemos ir área por área vendo os resultado obtidos (xa entraremos en detalle destes máis adiante) así coma un informe tanto en *html* ou *pdf* dos resultados finais.

Figura A.2: OSForensics - Auto triage 2

A.2 Forensic Imaging

Dentro desta área **OSForensics** permítenos crear unha *Logical Image* dos archivos volátiles mais comúns do noso sistema, como poden ser *hibefil.sys*, *pagefile.sys*, *usrClass.dat*....
A hora de gardalo temos duas opción, ou ben copiar directamente os arquivos a unha carpeta ou ven crear un *Virtual Disk Image* (.vhd) que podemos abrir por exemplo con *Oracle VM VirtualBox*.

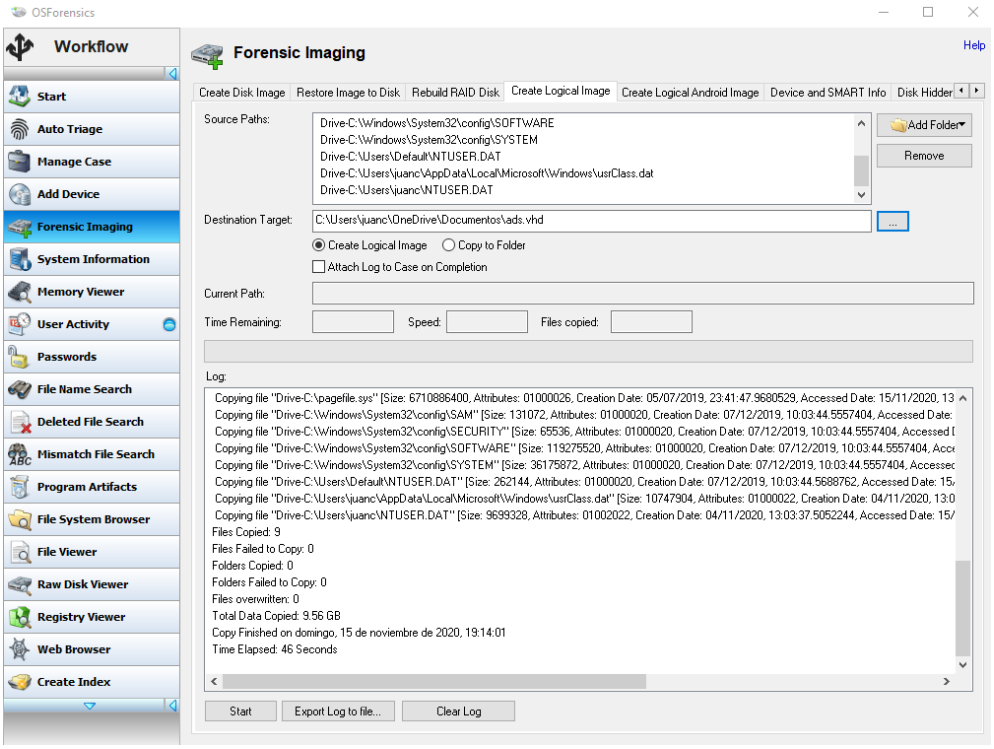


Figura A.3: OSForensics - Forensic Imaging

A.3 System Information

Esta sección permítenos saber con exactitud todas as características do sistema analizado. Existen varias listas de características pero a que nós nos interesa é a de *Basic System Information* que inclúe o nome do equipo, o sistema operativo, a información da CPU, memoria física, gráficos, USB, portos ou impresoras conectadas, diferente información da rede, información sobre unidades ópticas e físicas de almacenamento, información sobre volúmenes ou particións de disco e información da placa base.

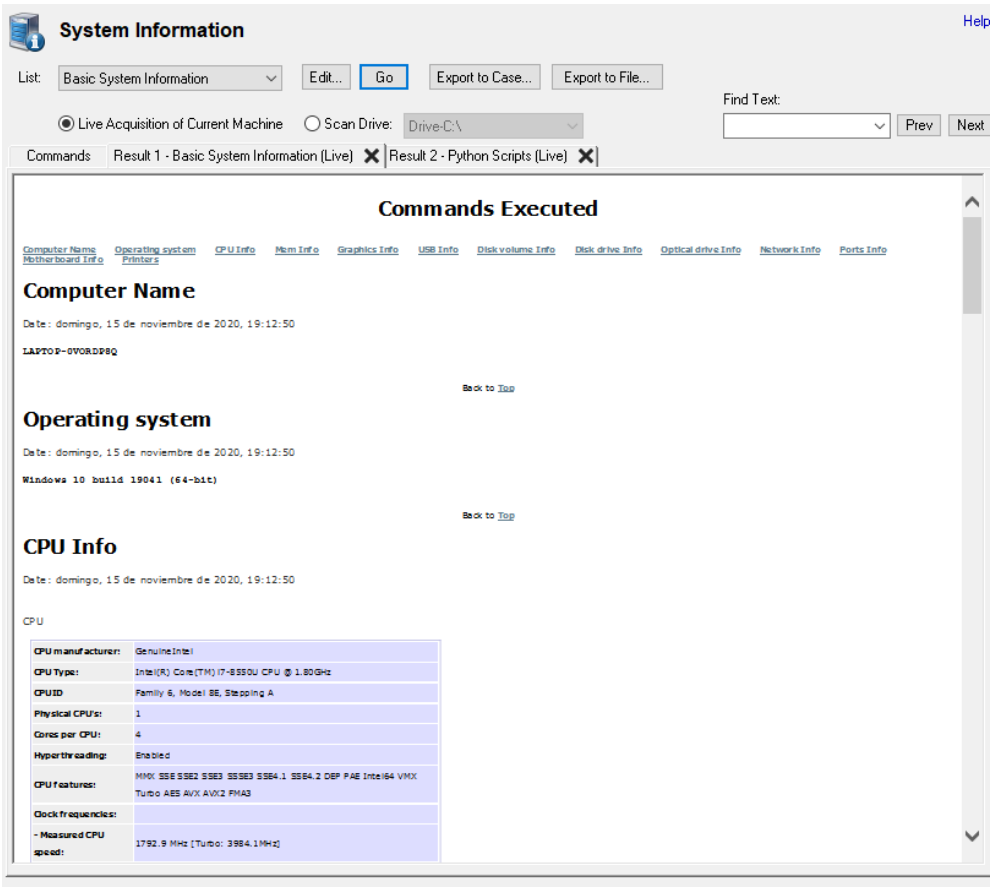


Figura A.4: OSForensics - System information

Como opción adicional podemos obter un informe final, dende a opción de *Export to file*, donde solo nos dará a opción de gardalo como *html*.

A.4 Memory Viewer

Nesta área podemos analizar a memoria volátil do noso sistema en tempo real. Permítenos visualizar todos os procesos que están correndo no noso sistema así como o seu PID, o momento no que se creou, o uso de CPU, os threads que creou, a descripcion, o nome do producto que o orixinou, o nome da compañia a versión e onde está gardado así como outros moitos parámetros.

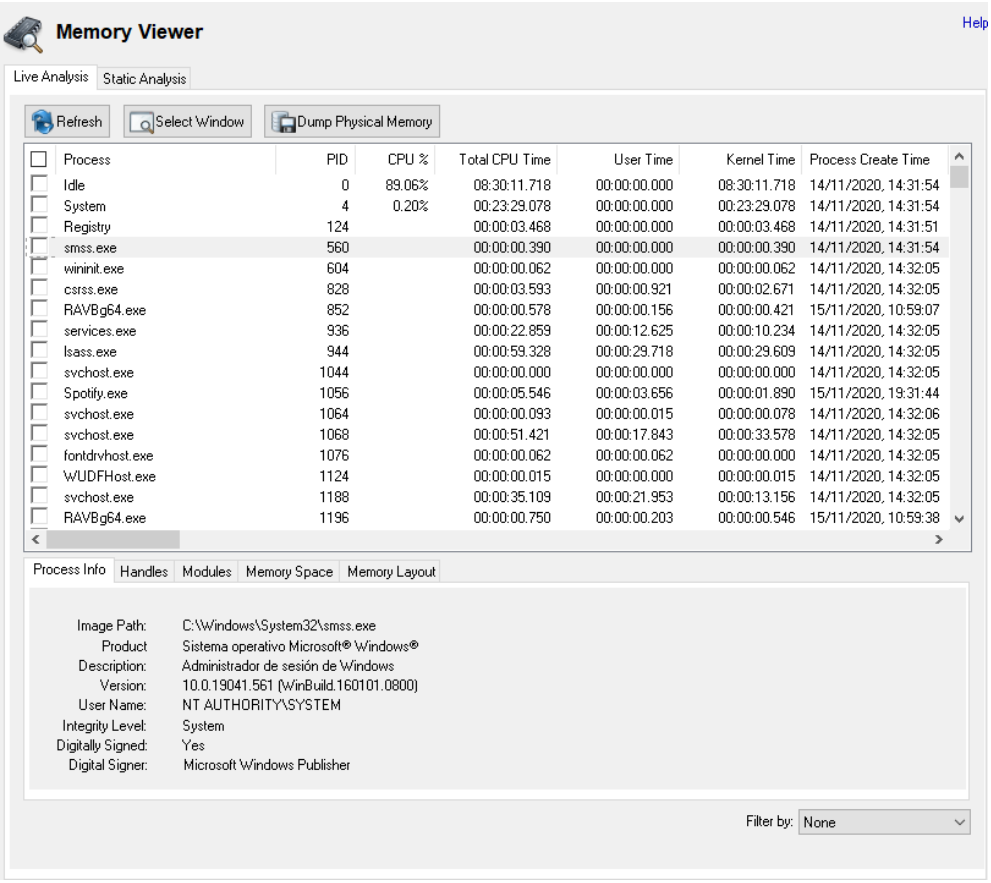


Figura A.5: OSForensics - Memory viewer

Tamén nos da a opción de analizar un volcado de memoria previamente gardado nun ficheiro ou facer un volcado da memoria *RAM* do sistema, nun arquivo con formato *.mem*.

A.5 User Activity

Permítenos coñecer as actividades recentes que realizou o usuario no sistema. Podemos obter información acerca das tarefas, dos eventos, dos arquivos recentemente usados, as últimas *WLAN* conectadas, cookies, descargas dende calquer navegador, o historial destes así como os favoritos gardados, os últimos programas instalados, as búsquedas no sistema de arquivos de **Windows** e outros moitos máis campos.

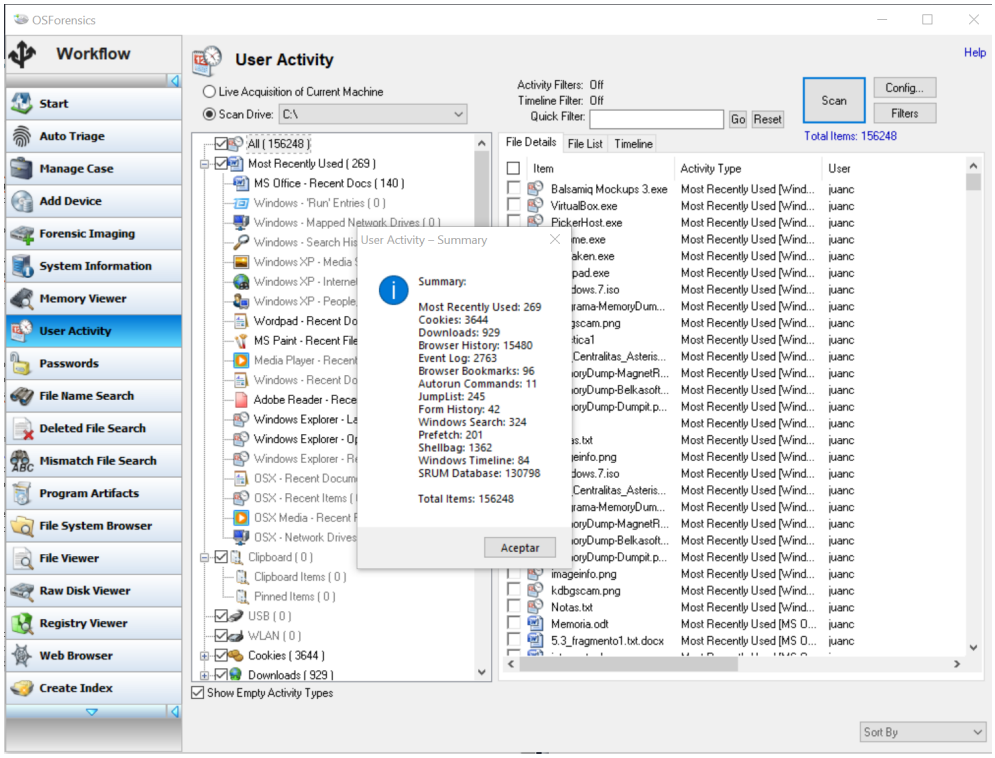


Figura A.6: OSForensics - User Activity

A.6 Passwords

Esta área permítenos obter os contrasinais gardados no noso sistema, e se as descoñecemos intentar atacar estas mediante *hashes* de estas a través da opción *Generate Rainbow Table*. Podemos recuperar as contrasinais de diferentes sitios como son contrasinais de WIFI, claves de producto para *Microsoft Office* ou *Visual Studio*, claves de **Windows**, de correos coma *Outlook*, de cifradores de coma *Bitlocker*[23], ...

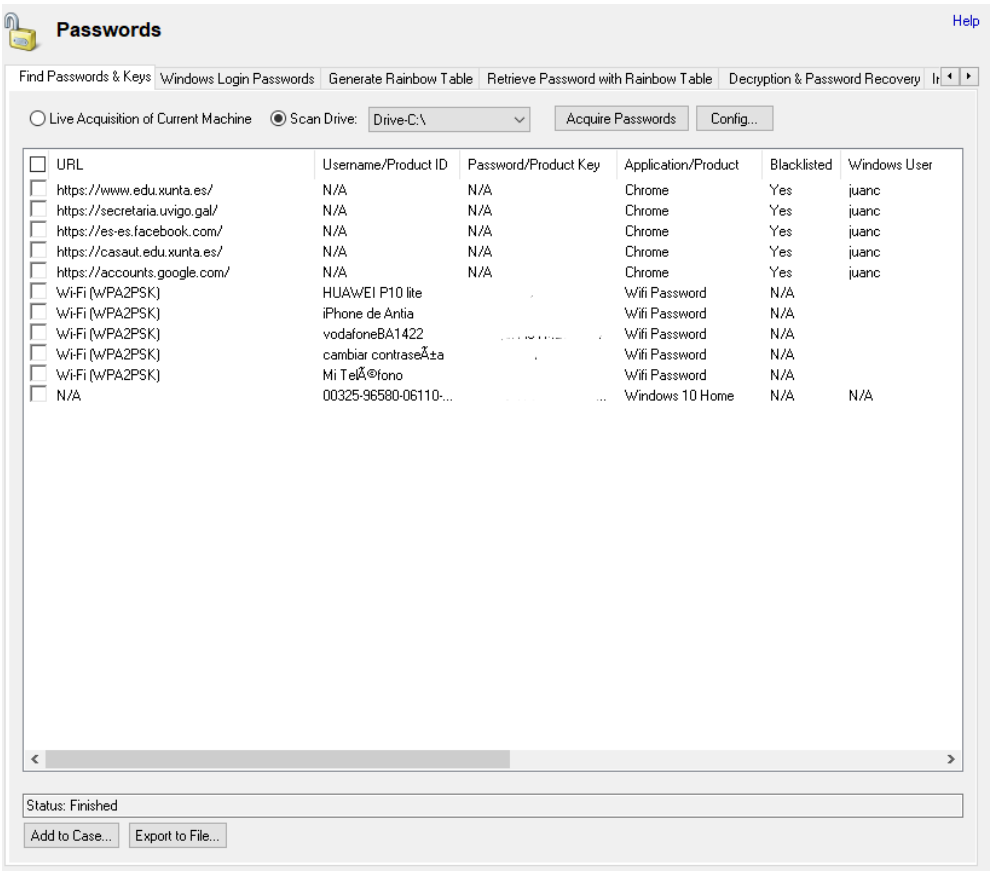


Figura A.7: OSForensics - Passwords

A.7 Deleted File Search

Permítenos recuperar e buscar archivos borrados incluso despois de borralos da papeleira de reciclaxe. Isto permitenos a opción de revisar arquivos que alguén quixo intentalos destruir con anterioridade.

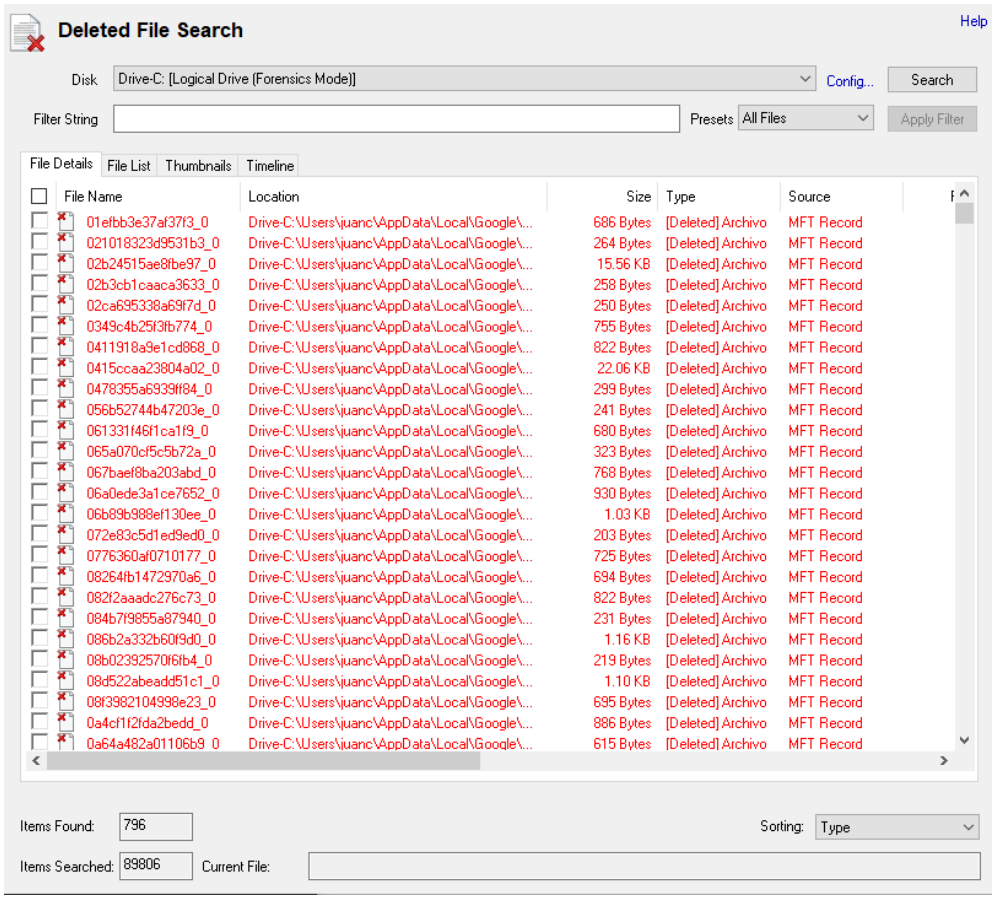


Figura A.8: OSForensics - Deleted File Search

A.8 Raw Disk Viewer

Grazas a este módulo vamos ter acceso a todos os sectores do disco seleccionado posibilitando así un análise de maior profundidade de todos os seus datos. Lee mais alá dos directorios e sistemas de arquivos, xa que accede o nivel máis baixo posible permitíndonos así poder atopar información sospeitosa que puidese estar oculta en sectores de disco sen asignar que non son normalmente accesibles polos mecanismos normais do sistema.

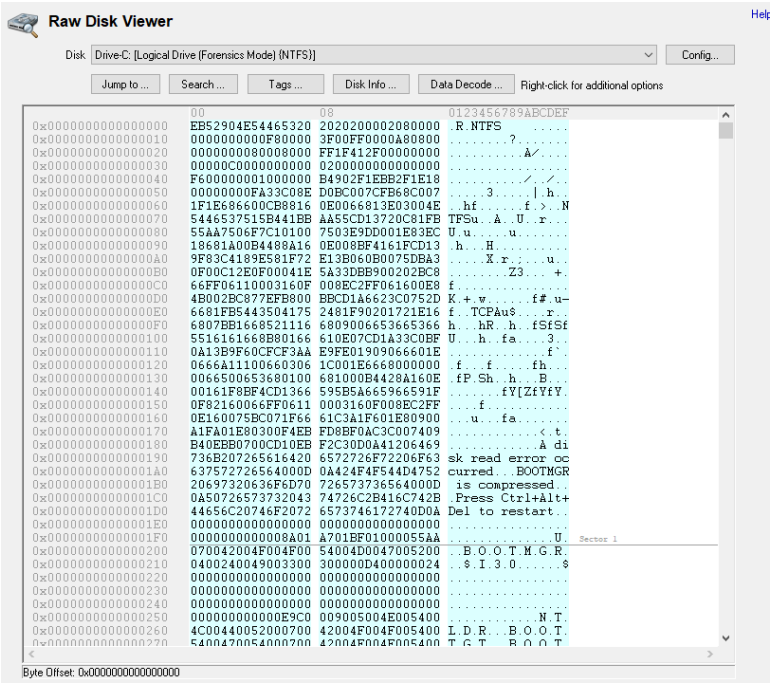


Figura A.9: OSForensics - Raw Disk Viewer

Podemos seleccionar a fracción de disco que queremos analizar mediante *Data Carving*[24] e visualizar na propia ferramenta a que arquivo corresponde ese sector do disco.

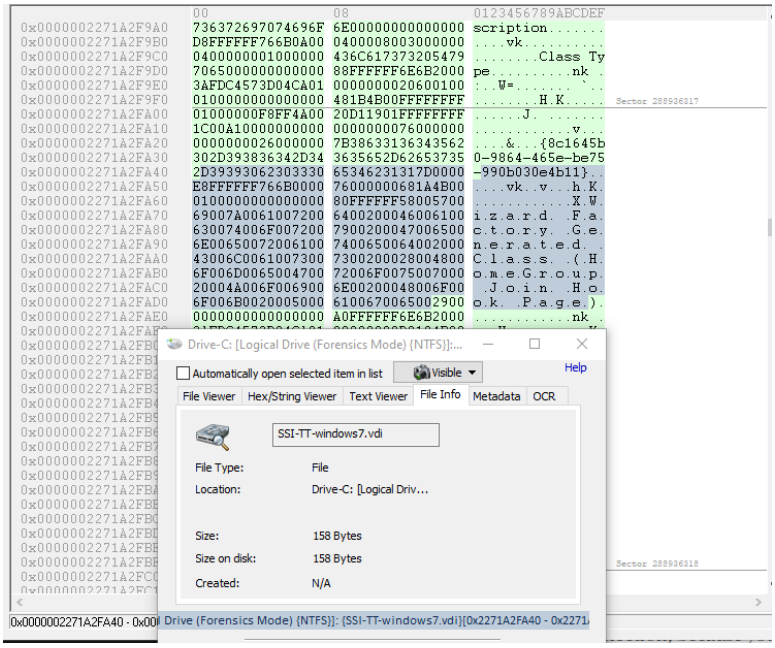


Figura A.10: OSForensics - Raw Disk Viewer 2

Apéndice B

Bitlocker

Bitlocker, [23], é unha carecterística de seguridade que permite ter mellor protección sobre os datos almacenados de un equipo. Está dispoñivel a partir de **Windows Vista** nas versión *Enteprise*, *Ultimate* e *Profesional*.
A diferenza de *EFS* (Encrypted File System), que encripta a nivel de archivos e carpetas, **Bitlocker**, encripta por unidade.
Bitlocker pode empregar un compoñente hardware chamado **TPM** (Módulo de Plataforma Segura) [25]. Este chip contén a tecnoloxía de cifrado de información para usuarios e é capaz de almacenar claves cifradas de datos de forma segura.
Para levar a cabo este caso de estudio creamos unha nova máquina virtual con **Windows 8.1 Profesional** nunha partición e **Ubuntu 18.04** noutra partición. A continuación vamos a detallar os pasos para levar a cabo a configuración e posta en marcha de **Bitlocker**.

O primeiro paso é seleccionar que método vamos a empregar apra desbloquear a unidade de inicio. Dánnos a posibilidade de desbloquear mediante unha unidade USB (deberemos introducilo cada vez que iniciamos o noso equipo) ou mediante un contrasinal. Nós eliximos esta última opción.

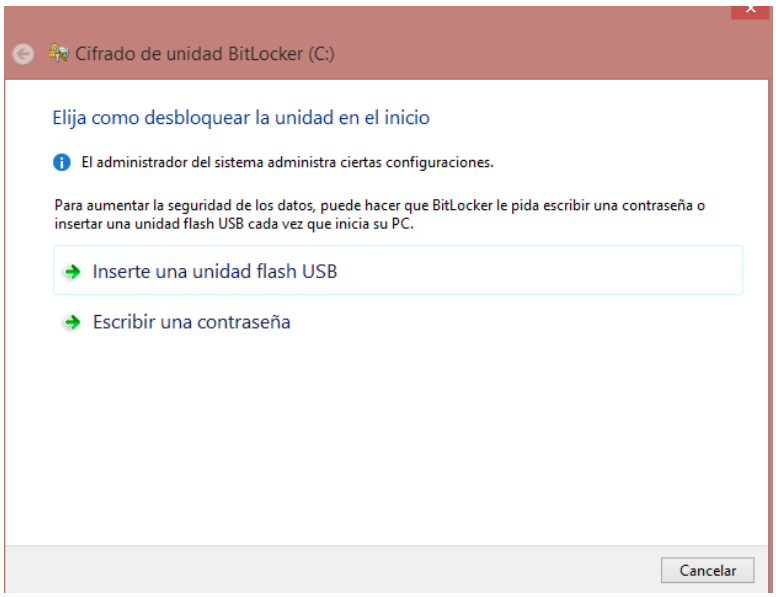


Figura B.1: Bitlocker - Install 1

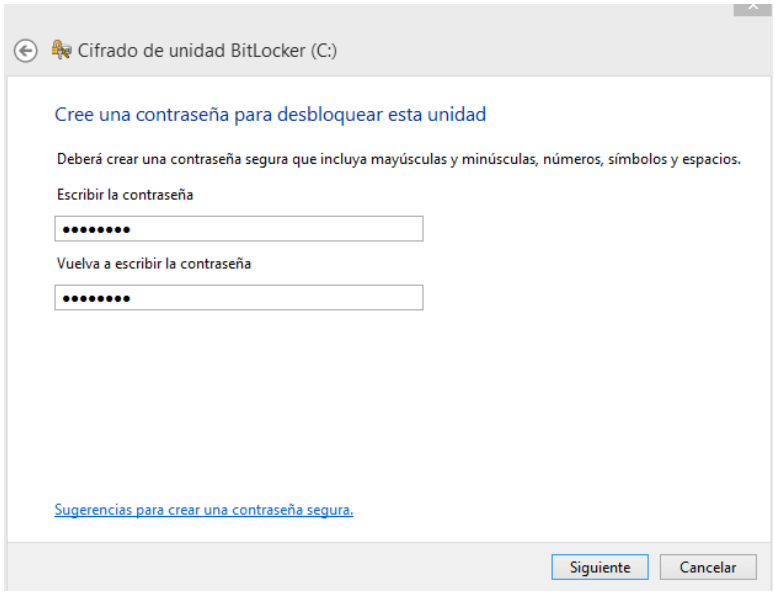


Figura B.2: Bitlocker - Install 2

No seguinte paso temos que elixir onde vamos a realizar a copia de seguridade da nosa clave de recuperación. O asistente para isto danos diferentes opcións como é gardar a clave en OneDrive (a que eliximos), utilizar un pen drive para dita clave, crear un ficheiro de texto onde é almacenada a clave ou imprimila directamente.

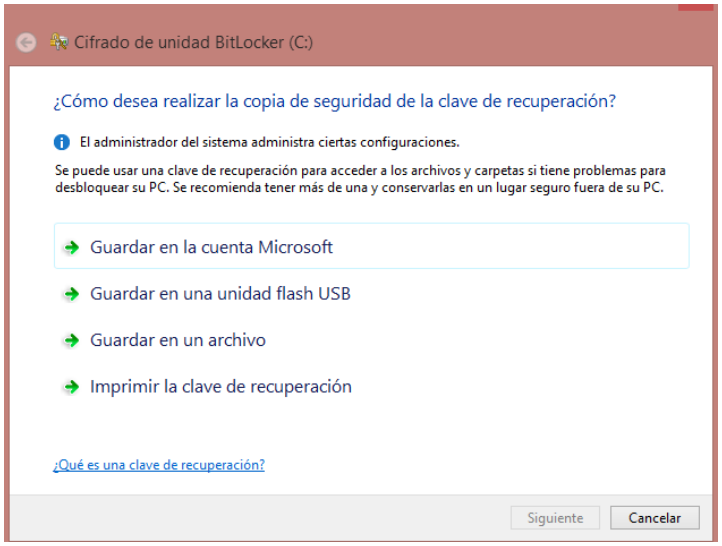


Figura B.3: Bitlocker - Install 3

A continuación teremos que seleccionar a cantidade de disco que vamos querer cifrar. Existen dúas posibilidades, cifrar toda a unidade ou cifrar solo a parte da unidade que estemos.

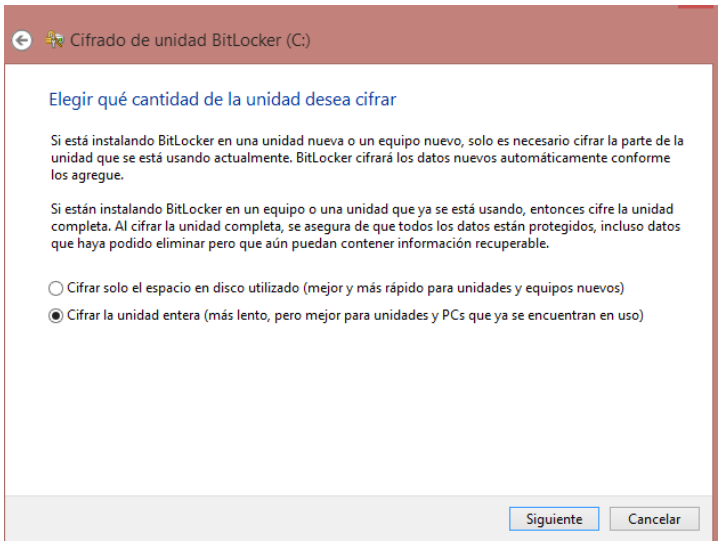


Figura B.4: Bitlocker - Install 4

Comenzará o cifrado e cando este termine pedirásenos reiniciar o equipo. No seguinte inicio xa se nos pedirá a clave para poder acceder ao noso sistema.

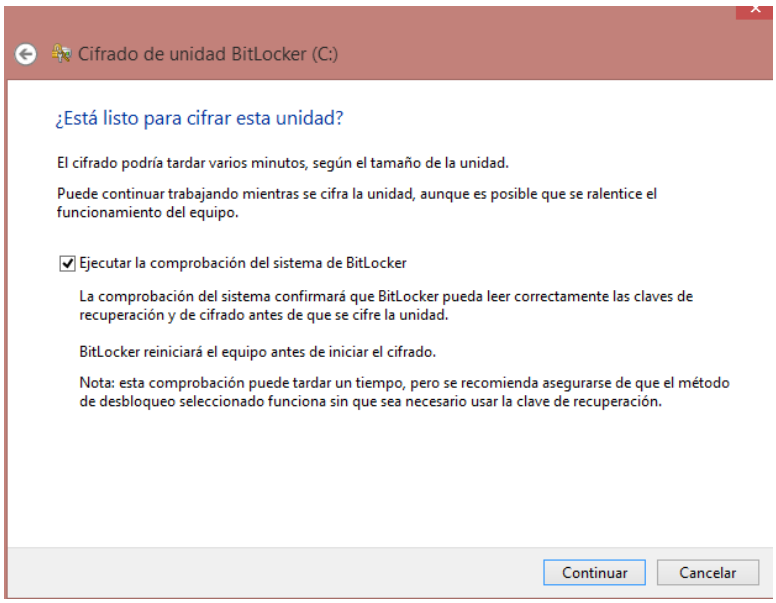


Figura B.5: Bitlocker - Install 5

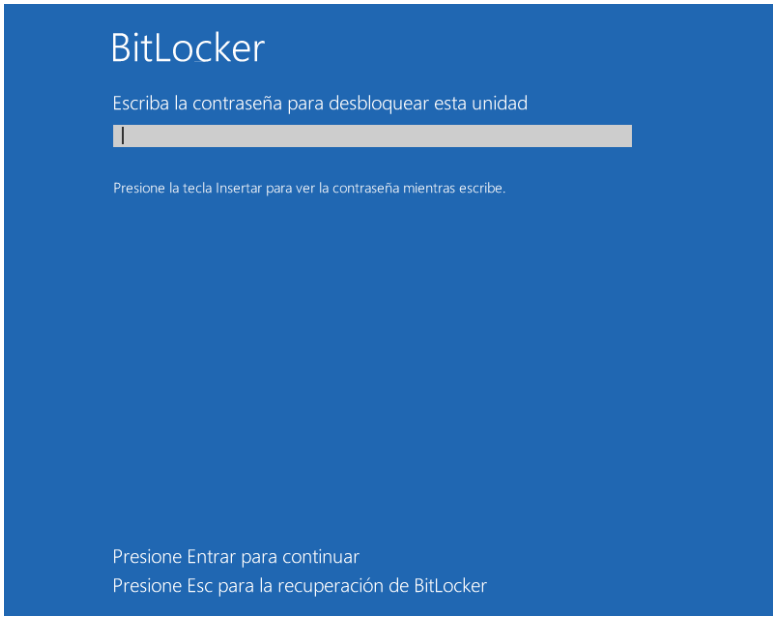


Figura B.6: Bitlocker - Inicio de disco

Se queremos facer un análisis forense de este disco, non vamos ser capaces de facelo sin ter a clave de desbloqueo, pero **Volatility** ofrece un plugin **Bitlocker** que é capaz de encontrar e devolver a clave de Bitlocker **FVEKS** (Full Volumen Encryption Keys). Esta só se emprega en versións de Windows Vista e Windows 7, mentras que nas versión máis recentes emprégase a API **CNGB** [26].

O plugin **Bitlocker**, podemos descargalo dende o seguinte enlace e gardalo na carpeta plugins da nosa carpeta **Volatility**.

O seguinte paso é descargar **libbde-utils**. Esta é unha librería que permite acceder a un disco que está cifrado mediante **Bitlocker**. Podemos empregar **bdeinfo** para determinar a información acerca do volumen que está encriptado, como se pode ver na seguinte imaxe a partición de **Windows** atópase na ubicación */dev/sda2*.

```
ssi@ssi-VirtualBox:~/Escritorio/volatility-master$ sudo bdeinfo /dev/sda2
bdeinfo 20170902

BitLocker Drive Encryption information:
        Encryption method               : AES-CBC 128-bit
        Volume identifier               : 4a16d94f-4b8b-4781-aa2b-ca0a931b60b8
        Creation time                   : Dec 07, 2020 16:26:52.692984100 UTC
        Description                     : JUANITO C: 07/12/2020
        Number of key protectors        : 2

Key protector 0:
        Identifier                      : 45d156a3-3ec7-4821-b48d-959d6f58a5c8
        Type                            : Password

Key protector 1:
        Identifier                      : 69227205-39b5-443b-92f3-1bf50ef1419c
        Type                            : Recovery password

Unable to unlock volume.
```

Figura B.7: Bitlocker - bdeinfo

A continuación facemos uso da ferramenta **Volatility** co correspondente plugin, **Bitlocker**, e vamos a extraer as claves FVEK que se atopan no volcado de memoria. Como o equipo que estamos a analizar é Windows 8, emprega a API CNG, vannos dar múltiples valores aloxados en diferentes ubiacións da memoria.

```
ssi@ssi-VirtualBox:~/volatility-2.6/volatility-master$ python vol.py -f JUANITO-20201207-163737.raw --profile=Win8SP1x64 bitlocker
Volatility Foundation Volatility Framework 2.6

Address : 0xe001bb4f5d60
Cipher  : AES-128
FVEK    : c23ff9d79fc05f0be46986336eaad9bb

Address : 0xe001bb4f65e0
Cipher  : AES-128
FVEK    : c23ff9d79fc05f0be46986336eaad9bb

Address : 0xe001bceaa560
Cipher  : AES-128
FVEK    : c23ff9d79fc05f0be46986336eaad9bb

Address : 0xe001bceaa800
Cipher  : AES-128
FVEK    : c23ff9d79fc05f0be46986336eaad9bb

Address : 0xf8021bd70560
Cipher  : AES-128
FVEK    : c23ff9d79fc05f0be46986336eaad9bb

Address : 0xf8021bd70800
Cipher  : AES-128
FVEK    : c23ff9d79fc05f0be46986336eaad9bb
```

Figura B.8: Bitlocker - volatility bitlocker

Creamos unha carpeta para montar o disco **BDE**, e facemos uso do comando **bdemount** para descifrar o disco. A este comando témoslle que pasar a clave FVEK que obtimos do plugin Bitlocker de Volatility.

```
ssi@ssi-VirtualBox:~/Escritorio/volatility-master$ sudo mkdir /mnt/bde
ssi@ssi-VirtualBox:~/Escritorio/volatility-master$ sudo bdemount -k c23ff9d79fc05f0be46986336eaad9bb /dev/sda2 /
mnt/bde
bdemount 20170902
```

Figura B.9: Bitlocker - bdemount

Finalmente, montamos e accedemos aos datos do disco.

```
ssi@ssi-VirtualBox:~/Escritorio/volatility-master$ sudo mkdir /mnt/WinHD
ssi@ssi-VirtualBox:~/Escritorio/volatility-master$ sudo mount -o loop,ro /mnt/bde/bde1 /mnt/WinHD
```

Figura B.10: Bitlocker - mount

```
ssi@ssi-VirtualBox:~/Escritorio/volatility-master$ sudo ls -l /mnt/WinHD/
total 1704381
lrwxrwxrwx 2 root root         24 dic  7 15:06 'Archivos de programa' -> '/mnt/WinHD/Program Files'
-rwxrwxrwx 1 root root     404250 nov 21  2014  bootmgr
-rwxrwxrwx 1 root root          1 jun 18  2013  BOOTNXT
lrwxrwxrwx 2 root root         16 ago 22  2013 'Documents and Settings' -> /mnt/WinHD/Users
-rwxrwxrwx 1 root root 1476395008 dic  7 19:08  pagefile.sys
drwxrwxrwx 1 root root          0 ago 22  2013  PerfLogs
drwxrwxrwx 1 root root       4096 dic  7 18:06  ProgramData
drwxrwxrwx 1 root root       4096 dic  7 17:58 'Program Files'
drwxrwxrwx 1 root root       4096 dic  7 18:06 'Program Files (x86)'
drwxrwxrwx 1 root root          0 dic  7 15:08 '$Recycle.Bin'
-rwxrwxrwx 1 root root  268435456 dic  7 19:08  swapfile.sys
-rwxrwxrwx 1 root root       1024 dic  7 18:43  SYSTAG.BIN
drwxrwxrwx 1 root root       4096 dic  7 18:50 'System Volume Information'
drwxrwxrwx 1 root root       4096 dic  7 15:07  Users
drwxrwxrwx 1 root root      24576 dic  7 15:07  Windows
```

Figura B.11: Bitlocker - ls -l /mnt/WinHD

Apéndice C

Instalación de Volatility en Ubuntu

Neste apendice número tres vamos realizar a instalación da ferramenta **Volatility** en **Ubuntu**, pois é empregada nalgúns casos durante o noso proxecto.
Para iso creamos unha nova máquina virtual, con un sistema operativo **Ubuntu 18.04.5** de 64 bits e 2gb de memoria ram.

A maiores da instalación de **Volatility**, imos instalar dous dos plugins mais empregados por esta ferramenta, que son **yara** e **distorm3**.

Empezamos primeiro de nada descargando do repositorio oficial de **Volatility** [27] a última versión dispoñible, que no noso caso é a **2.6.1**.

A continuación deberemos instalar na nosa máquina o linguaxe de programación **Python**. Nos eleximos a versión **2.7** pois lendo en diversos sitios chegamos a conclusión que é a que mellor funciona con **Volatility**. Para instalalo simplemente temos que executar:

```
ssi@ssi-VirtualBox:~/volatility-2.6/volatility-master$ sudo apt-get install python
[sudo] contraseña para ssi:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  libpython-stdlib python-minimal python2.7 python2.7-minimal
Paquetes sugeridos:
  python-doc python-tk python2.7-doc binfmt-support
Se instalarán los siguientes paquetes NUEVOS:
  libpython-stdlib python python-minimal python2.7 python2.7-minimal
0 actualizados, 5 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 176 kB/1.713 kB de archivos.
Se utilizarán 4.982 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n]
```

Figura C.1: Python 2.7 - Ubuntu

Unha vez instalado o seguinte paso é instalar **Pip**. Este é un sistema de xestión de paquetes utilizado para instalar e administrar paquetes de software escritos en **Python**. Existen diversas formas de instalalo xa sexa mediante o executable de **get-pip.py** ou directamente empregando a ferramenta avanzada de empaquetado ou mais ben coñecida como **apt** [28].

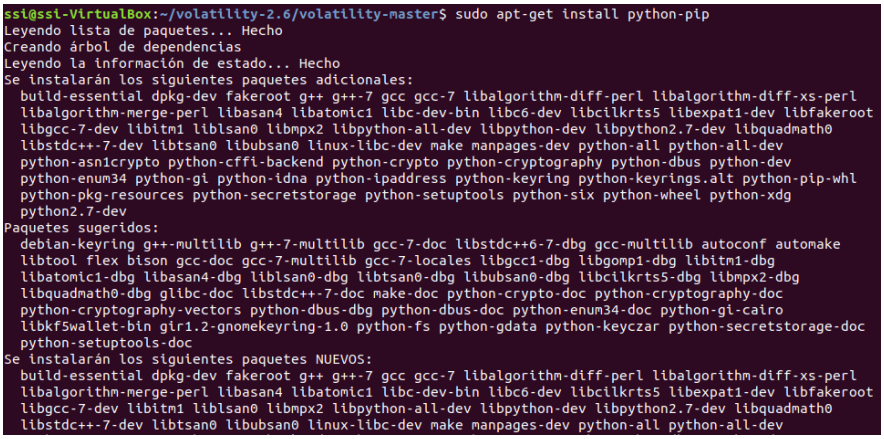


Figura C.2: Install pip - Ubuntu

Xa podemos entón instalar **Volatility**. Descomprimimos a carpeta que descargamos no primeiro paso e unha vez dentro dela podemos executar o instalador **setup.py** pasándolle a opción *install*.

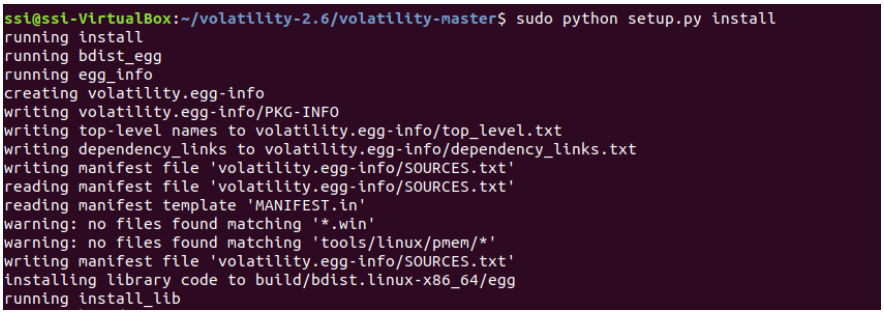


Figura C.3: setup.py install - Ubuntu

A continuación imos a instalar dous plugins para **Volatility**, os cales son **yara** e **distorm3**.
Para instalar **yara** imos acceder ao seu repositorio en *Github* [29] e descargar a última versión dispoñible, no noso caso é a **v4.0.2**.
Na documentación de yara xa nos dan unha serie de pasos para que nos sexa máis sinxela a súa instalación:

- Primeiro de nada deberemos instalar os paquetes **automake**, **libtool**, **make**, **gcc** e **pkg-config**.

```
ssi@ssi-VirtualBox:~/Descargas/yara-4.0.2$ sudo apt-get install automake libtool make gcc pkg-config
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
make ya está en su versión más reciente (4.1-9.1ubuntu1).
fijado make como instalado manualmente.
gcc ya está en su versión más reciente (4:7.4.0-1ubuntu2.3).
fijado gcc como instalado manualmente.
Se instalarán los siguientes paquetes adicionales:
  autoconf autotools-dev libltdl-dev libsigsegv2 m4
Paquetes sugeridos:
  autoconf-archive gnu-standards autoconf-doc libtool-doc gfortran
  | fortran95-compiler gcj-jdk m4-doc
Se instalarán los siguientes paquetes NUEVOS:
  autoconf automake autotools-dev libltdl-dev libsigsegv2 libtool m4
  pkg-config
0 actualizados, 8 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 1.483 kB de archivos.
Se utilizarán 6.380 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] S
```

Figura C.4: packages previuos yara - Ubuntu

- Logo deberemos executar o ficheiro .sh **./bootstrap.sh**.

```
ssi@ssi-VirtualBox:~/Descargas/yara-4.0.2$ sudo ./bootstrap.sh
libtoolize: putting auxiliary files in AC_CONFIG_AUX_DIR, 'build-aux'.
libtoolize: copying file 'build-aux/ltmain.sh'
libtoolize: putting macros in AC_CONFIG_MACRO_DIRS, 'm4'.
libtoolize: copying file 'm4/libtool.m4'
libtoolize: copying file 'm4/ltoptions.m4'
libtoolize: copying file 'm4/ltsugar.m4'
libtoolize: copying file 'm4/ltversion.m4'
libtoolize: copying file 'm4/lt~obsolete.m4'
configure.ac:20: installing 'build-aux/ar-lib'
configure.ac:20: installing 'build-aux/compile'
configure.ac:38: installing 'build-aux/config.guess'
configure.ac:38: installing 'build-aux/config.sub'
configure.ac:8: installing 'build-aux/install-sh'
configure.ac:8: installing 'build-aux/missing'
Makefile.am: installing 'build-aux/depcomp'
parallel-tests: installing 'build-aux/test-driver'
configure.ac: installing 'build-aux/ylwrap'
```

Figura C.5: bootstrap.sh - Ubuntu

- A continuación deberemos compilar **yara**.

```
ssi@ssi-VirtualBox:~/Descargas/yara-4.0.2$ sudo ./configure
checking whether make supports nested variables... yes
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking for style of include used by make... GNU
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether gcc understands -c and -o together... yes
checking dependency style of gcc... gcc3
checking for ar... ar
checking the archiver (ar) interface... ar
checking for gcc... (cached) gcc
checking whether we are using the GNU C compiler... (cached) yes
checking whether gcc accepts -g... (cached) yes
```

Figura C.6: configure.sh - Ubuntu

- Por último deberemos crear o instalador e executalo mediante a utilidade de **make** [30].

Figura C.7: make yara - Ubuntu

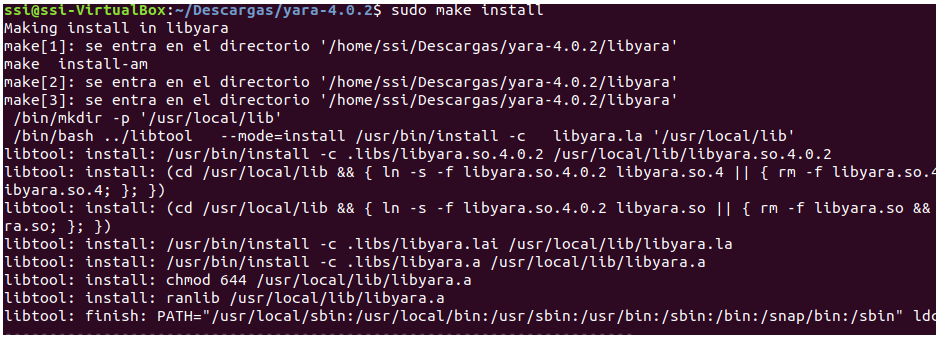


Figura C.8: make install yara - Ubuntu

- Podemos facer unha comprobación de que todo está correctamente instalado a través do comando **make check**.

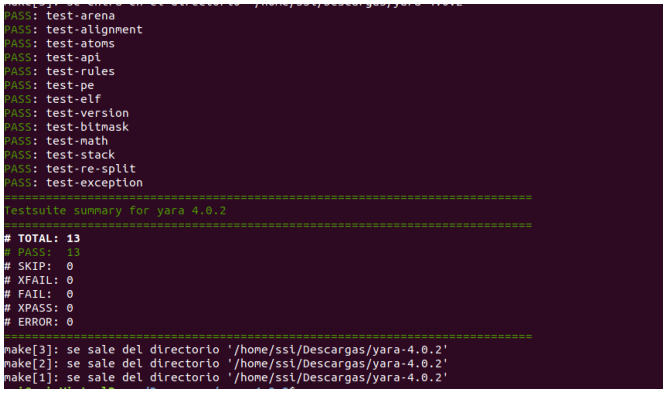


Figura C.9: make check yara - Ubuntu

Por último imos instalar o plugin **distorm3**. En ubuntu este acometido realízase moi sinxelo pois podemos instalalo directamente con **pip**. Nós imos instalar a versión **3.4.4** pois é a que mellor funciona con **Python 2.7**.

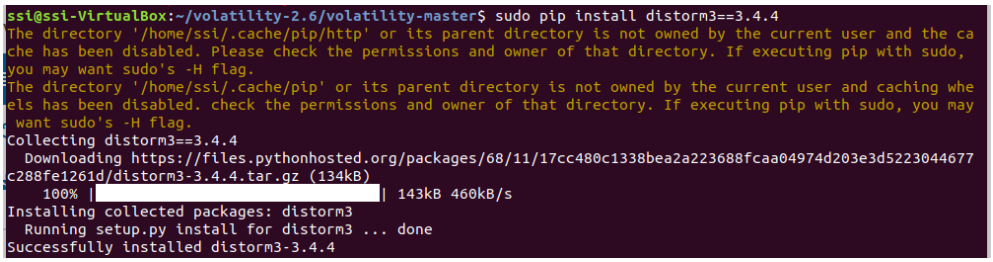


Figura C.10: pip install distorm3 - Ubuntu

Podemos comprobar que a instalación de **Volatility** resultou correcta executando **vol.py** e abrindo a opción de axuda.

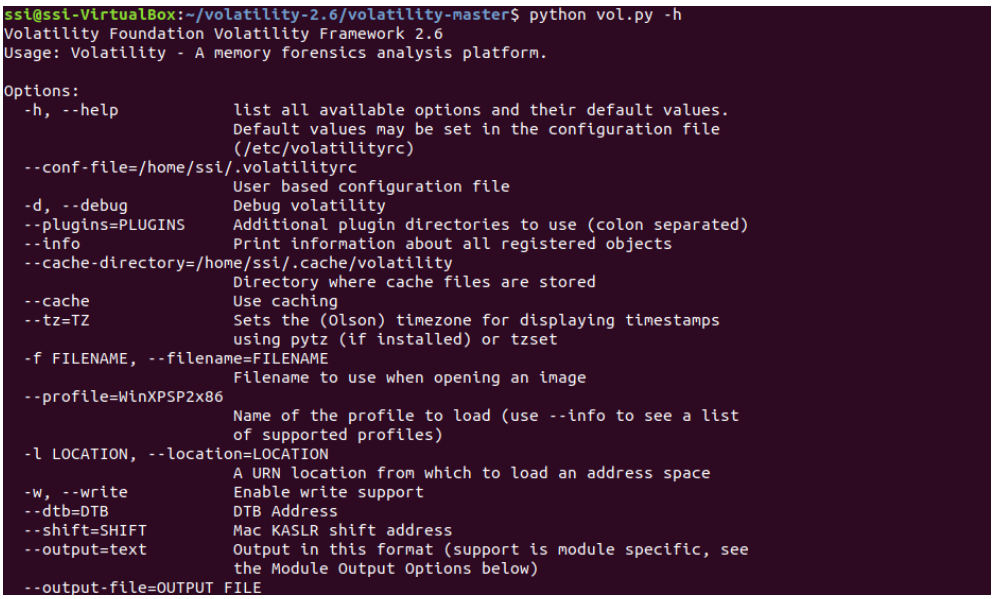


Figura C.11: vol.py -h - Ubuntu

Bibliografía

[1] M. H. Ligh, A. Case, J. Levy, and A. Walters, *The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory.* John Wiley & Sons, 2014.

[2] M. Sikorski and A. Honig, *Practical Malware Analysis.* no starch press, 2012.

[3] "Reverse-enginering," https://en.wikipedia.org/wiki/Reverse_engineering, accessed November 07, 2020.

[4] "Sandbox," https://en.wikipedia.org/wiki/Sandbox_(computer_security), accessed November 07, 2020.

[5] "Hipervisor vm," https://es.wikipedia.org/wiki/Hipervisor, accessed December 08, 2020.

[6] "Direct memory access," https://en.wikipedia.org/wiki/Direct_memory_access, accessed November 07, 2020.

[7] "Peripheral component interconnect," https://en.wikipedia.org/wiki/Peripheral_Component_Interconnect, accessed November 07, 2020.

[8] "Dynamic host configuration protocolt," https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol, accessed November 07, 2020.

[9] "Magnet ram capture," https://www.magnetforensics.com/resources/magnet-ram-capture/, accessed November 07, 2020.

[10] "Fat32," https://es.wikipedia.org/wiki/Tabla_de_asignaci%C3%B3n_de_archivos, accessed December 08, 2020.

[11] "Belkasoft live ram capturer," https://belkasoft.com/ram-capturer, accessed November 07, 2020.

[12] "Moonsols dumpit," https://github.com/thimbleweed/All-In-USB/tree/master/utilities/DumpIt, accessed November 07, 2020.

[13] "Volatility," https://www.volatilityfoundation.org/, accessed November 07, 2020.

[14] "Python," https://www.python.org/, accessed November 07, 2020.

[15] "Distorm3," https://github.com/gdabah/distorm/, accessed November 07, 2020.

[16] "Yara plugin," https://yara.readthedocs.io/en/stable/index.html, accessed December 08, 2020.

[17] "Pip," https://pip.pypa.io/en/stable/, accessed November 07, 2020.

[18] "Service pack," https://es.wikipedia.org/wiki/Service_Pack, accessed November 07, 2020.

[19] "Eprocess," https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/eprocess, accessed November 07, 2020.

[20] "Dot," https://es.wikipedia.org/wiki/DOT, accessed December 08, 2020.

[21] "Winsock," https://en.wikipedia.org/wiki/Winsock, accessed December 08, 2020.

[22] "Osforensics," https://www.osforensics.com/, accessed November 07, 2020.

[23] "Bitlocker," https://docs.microsoft.com/es-es/windows/security/information-protection/bitlocker/bitlocker-overview, accessed December 08, 2020.

[24] "Data carving o file carving," https://en.wikipedia.org/wiki/File_carving, accessed November 07, 2020.

[25] "Tpm," https://www.profesionalreview.com/2018/11/10/tpm/, accessed December 08, 2020.

[26] "Cngb," https://docs.microsoft.com/en-us/windows/win32/seccng/cng-portal, accessed December 08, 2020.

[27] "Repositorio volatility releases," https://github.com/volatilityfoundation/volatility/releases, accessed December 08, 2020.

[28] "Advanced packaging tool," https://help.ubuntu.com/kubuntu/desktopguide/es/apt-get.html, accessed December 08, 2020.

[29] "Repositorio yara plugin," https://yara.readthedocs.io/en/stable/index.html, accessed December 08, 2020.

[30] "Ubuntu make," https://wiki.ubuntu.com/ubuntu-make, accessed December 08, 2020.