

Técnicas y herramientas para el análisis de debilidades en volcados de memoria RAM de sistemas basados en Linux.

Juan Ramón Betancor Olivares.

Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones (MISTIC).

Seguridad empresarial.

Víctor Méndez Muñoz.

Víctor García Font.

01/06/2020



Esta obra está sujeta a una licencia de Reconocimiento-
NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Técnicas y herramientas para el análisis de debilidades en volcados de memoria RAM de sistemas basados en Linux.</i>
Nombre del autor:	<i>Juan Ramón Betancor Olivares</i>
Nombre del consultor/a:	<i>Víctor Méndez Muñoz</i>
Nombre del PRA:	<i>Víctor García Font</i>
Fecha de entrega (mm/aaaa):	06/2020
Titulación:	<i>Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones</i>
Área del Trabajo Final:	<i>Seguridad empresarial</i>
Idioma del trabajo:	<i>Castellano</i>
Palabras clave	<i>Seguridad empresarial, volcados de memoria, análisis forense.</i>
Resumen del Trabajo (máximo 250 palabras): <i>Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.</i>	
<p>La sociedad en los últimos años ha migrado de lo analógico a lo digital, depositando en el ciberespacio gran parte de nuestras vidas, desde cuentas de banco hasta números de teléfonos. Las organizaciones y empresas al ver las posibilidades que el mundo digital ofrece, a lo largo de los años también han dado el paso al ciberespacio. Sin embargo, no son los únicos a los que la tecnología ha llamado la atención, los criminales también se han visto atraídos debido al mundo de posibilidades y anonimato que se les ofrece. Desde los años 90 hasta nuestros días, el cibercrimen ha ido creciendo de forma imparable, así como las medidas de seguridad y prevención hacia dicho fenómeno, siendo una de estas el análisis forense.</p> <p>En el presente trabajo se analizarán las metodologías y técnicas existentes para análisis de volcados de memoria, así como la implementación de estas herramientas sobre un volcado de memoria de un sistema IoT estándar con el objetivo de identificar las debilidades que este presenta.</p> <p>Posteriormente se generará un informe con los datos obtenidos durante el proceso de análisis de tal manera que nos permita ver de manera clara si existe problemas y debilidades en cuanto a diseño e implementación.</p> <p>Finalmente, teniendo las conclusiones presentes, se propondrá y analizará diferentes alternativas y soluciones a nivel de análisis y diseño a los problemas encontrados de tal manera se puedan mitigar y evitar que estos ocurran en un futuro.</p>	

Abstract (in English, 250 words or less):

In recent years, society has migrated from analog to digital, depositing much of our lives in cyberspace, from bank accounts to phone numbers. Organizations and companies seeing the possibilities that the digital world offers, over the years have also taken the step to cyberspace. However, they are not the only ones to whom technology has attracted attention, criminals have also been attracted due to the world of possibilities and anonymity offered to them. From the 90s to the present day, cybercrime has been growing unstoppably, as well as security and prevention measures towards this phenomenon, one of these being forensic analysis.

In the present work we will analyze the existing methodologies and techniques for memory dumps analysis, as well as the implementation of these tools on a memory dump of a standard IoT system with the objective of identifying the weaknesses that it presents.

Then, a report will be generated with the data obtained during the analysis process in a way that allows us to see clearly if there are problems and weaknesses in terms of design and implementation.

Finally, having the conclusions, different alternatives and solutions at the level of analysis and design will be proposed and analyzed to the problems encountered in such a way that they can be mitigated and prevented from occurring in the future.

Índice

1. Introducción.....	9
a. Problema por resolver: contexto y justificación del trabajo.....	9
b. Objetivos del Trabajo	11
c. Enfoque y metodología seguida.....	12
d. Listado de tareas	13
e. Planificación temporal del trabajo	14
f. Estado del arte	16
g. Breve resumen de productos obtenidos.....	18
h. Breve descripción de los capítulos de la memoria.....	19
Conceptos generales.....	20
Conceptos de seguridad informática.....	20
Sistemas distribuidos	20
Iot (Internet of Things)	20
Fileless malware.....	21
Memoria principal.....	22
Memoria secundaria	23
Dispositivos/unidades de almacenamiento y soportes/medios de almacenamiento	23
Volcados de memoria.....	24
Proceso de carga en la memoria RAM	24
Procesos informáticos.....	24
El núcleo del sistema GNU/Linux	25
Análisis forense.....	27
Metodología.....	27
Evidencia digital.....	27
Perspectiva de tres roles	28
Estudio de herramientas de análisis de volcados de memoria kernel.....	29
Herramientas para generar volcados de memoria.....	29
FTK Imager.....	30
LiME (Linux Memory Extractor)	30
Volatilitux	30
Herramientas adicionales.....	31
Volatility	31
Características	32

Redline.....	33
AccessData Forensic Toolkit (FTK).....	33
Menciones adicionales.....	34
Comparación entre herramientas.....	35
Preparando el entorno de trabajo.	36
Creación del perfil de Volatility e instalación.....	39
Análisis del volcado de memoria.....	40
Seleccionando el perfil de análisis.	42
Visualización de procesos en ejecución	43
Analizando los procesos.....	45
Memoria de los procesos.....	47
Objetos y memoria del kernel.....	48
Visualización de la red.....	52
Información del sistema.....	55
Detección de rootkits.....	59
Breve resumen de los resultados obtenidos.....	61
Conclusiones	63
Glosario	66
Bibliografía.	68
Anexos.....	71
Sistema operativo utilizado.....	71
Instalación de FTK Imager y ejemplo de uso.....	71
Instalación de LiME	72
Instalación de Volatility.....	73
Ejemplos de uso de Volatility	74
PSLIST Plugin Volatility	75
Redline de FireEye, instalación y ejemplo de uso.	76
Resultados del análisis de prueba utilizando Redline de FireEye.	79
Simulando arquitectura ARM con el sistema operativo Raspbian y QEMU	79
Simulando Raspbian Desktop OS sobre un entorno Windows 10.	81
Generando el volcado de memoria de Raspbian con LiME.	82
Generando el perfil/profile de Volatility del entorno Raspbian 4.19 e instalación en Volatility.	82
Visualización de procesos en ejecución	84

Visualización de los hilos de ejecución.....	85
Utilización del plugin malfind sobre el volcado de memoria	86
Realizando el vaciado de los procesos con el plugin procdump	87
Analizando los procesos	88
Mapeo de memoria de los distintos procesos	91
Memoria y objetos del kernel Linux.....	94
Networking.....	95
Información del sistema	96
Detección de rootkits	98

Lista de Figuras

Ilustración 1. Aumento de incidentes de ciberseguridad 2013-2018. CCN	10
Ilustración 2. Planificación temporal del proyecto	14
Ilustración 3. Diagrama de Gantt con la planificación temporal.....	15
Ilustración 4. Funciones básicas de un núcleo.	26
Ilustración 5. Propiedades del volcado de memoria.....	38
Ilustración 6. Vulnerabilidades que afectan al kernel 4.19 de Linux.	51
Ilustración 7. MDS Tool para la búsqueda de vulnerabilidades.	56
Ilustración 8. Funcionamiento de hostnecml.	71
Ilustración 9. FTK Imager, realizando el volcado.....	71
Ilustración 10. Clonando el repositorio de LiME.	72
Ilustración 11. Versión de python instalada.....	73
Ilustración 12. Clonando el repositorio de Volatility.....	73
Ilustración 13. Instalación de Volatility	73
Ilustración 14. Volatility imageinfo plugin	74
Ilustración 15. Volatility pslit plugin	75
Ilustración 16. FTK Imager para la generación del volcado.....	75
Ilustración 17. PSLIST Plugin de Volatility	75
Ilustración 18. Principales funcionalidades de Redline.....	76
Ilustración 19. Perfiles de investigación de Redline.....	77
Ilustración 20. Procesos en Redline	77
Ilustración 21. Panel de resultados del análisis.....	78
Ilustración 22. Filtros de Redline.....	78
Ilustración 23. Detalles y pestañas asociadas a un proceso	78
Ilustración 24. Raspbian emulado en QEMU.	80
Ilustración 25. Habilitar PAE en VirtualBox.	81
Ilustración 26.. Raspbian Desktop instalado en VirtualBox.	81
Ilustración 27. Generando el volcado de memoria con LiME.	82
Ilustración 28. Generando el fichero module.dwarf.....	83
Ilustración 29. Adquisición del fichero System.map.	83
Ilustración 30. Perfil instalado correctamente.....	84
Ilustración 31. Listado de procesos del volcado.	84
Ilustración 32. Relación de procesos padre-hijo del proceso systemd.	85
Ilustración 33. Listado de hilos de ejecución del volcado.	86
Ilustración 34. Utilización del plugin malfind.	86
Ilustración 35. Vaciado de todos los procesos activos.....	87
Ilustración 36. Cantidad de líneas del volcado del proceso sudo.	88
Ilustración 37. Uso de la herramienta strings para el análisis de ficheros binarios.....	88
Ilustración 38. Strings y grep para el filtrado de procesos.....	89
Ilustración 39. Resultado de la ejecución del script.sh.	89
Ilustración 40. Usando la herramienta foremost.	90
Ilustración 41. Contenido fichero audit.txt.	90
Ilustración 42. Ejemplo de detección de malware utilizando Virustotal.	91
Ilustración 43. Ejecución del plugin memmap.	92
Ilustración 44. Plugin linux_proc_maps.	92

Ilustración 45. Utilización del plugin linux_dump_map.	93
Ilustración 46. Visualización del historial de instrucciones ejecutadas a través del bash.	93
Ilustración 47. Plugin linux_lsmod.	94
Ilustración 48. Sistema de ficheros tmpfs.	95
Ilustración 49. Visualización de la tabla ARP.	95
Ilustración 50. Visualización de las interfaces de red con el plugin linux_ifconfig.	96
Ilustración 51. nmap y netstat sobre el volcado.	96
Ilustración 52. Visualización de la información de la CPU.	97
Ilustración 53. Plugin dmesg.	97
Ilustración 54. Direcciones reservadas para dispositivos de entrada y salida.	97
Ilustración 55. Plugin linux_mount.	98
Ilustración 56. Plugin linux_check_ainfo para la detección de rootkits.	98
Ilustración 57. linux_check_tty plugin.	99
Ilustración 58. Búsqueda de rootkits utilizando técnicas DKOM.	99
Ilustración 59. Utilización del plugin linux_check_fop.	99
Ilustración 60. Uso del plugin linux_check_idt.	100
Ilustración 61. Ejemplo del plugin linux_check_syscall.	100
Ilustración 62. Utilizando el plugin linux_check_modules.	100

1. Introducción.

a. Problema por resolver: contexto y justificación del trabajo

La evolución tecnológica de los últimos años ha acarreado con el inevitable paso del mundo analógico al mundo digital, gracias a, entre otros aspectos, las facilidades que la tecnología nos aporta en el día a día. Por una parte, la sociedad dispone de redes sociales donde comparten información con otros usuarios, disponen de banca electrónica, realizan transacciones por internet, etc. En definitiva, disponen de innumerables servicios disponibles que nos facilitan la vida. Por otro lado, las empresas y organizaciones debido a dichas ventajas que la tecnología ofrece también han dado el paso al ciberespacio, depositando ahí todos los datos de los usuarios y trabajadores, servicios, correos electrónicos, etc. En definitiva, se podría afirmar sin temor a equivocarnos que dependemos, hoy más que nunca, de la tecnología.

Todas las actividades que llevamos a cabo en internet se podría decir que tienen un denominador común. Las fotos que se suben a las redes sociales, así como las nóminas de los trabajadores de las empresas que se almacenan en la nube, son, en definitiva, datos e información. ¿Qué ocurriría si se dieran un mal uso o no se tuviesen las medidas de seguridad y precauciones necesarias sobre los sistemas que almacenan esta información? ¿Qué ocurriría si dichos dispositivos o sistemas disponen de fallos o errores a niveles de diseño que puedan acarrear pérdidas? Indudablemente puede desembocar en un problema bastante grave debido a que corremos el riesgo a perder toda esa información que nos pertenece, e incluso algo peor, que esta pueda acabar en manos no deseadas. Por lo tanto, también podemos decir otra verdad y es que dependemos de los sistemas informáticos para que toda la información esté bien protegida.

Uno de los principales activos que gestionan las empresas es sin lugar a duda los datos. Los datos, que se suponía que eran la nueva mina de oro con la digitalización, también son munición para la ciberdelincuencia, un negocio cada vez más lucrativo y devastador que, desde los años 90, no ha hecho más que aumentar sin control.

Quest Diagnostics, un laboratorio clínico estadounidense que forma parte de las 500 mayores empresas de ese país ha informado de que los datos de 11,9 millones de pacientes (incluidas tarjetas de crédito y cuentas bancarias) estuvieron ocho meses expuestos por el error de seguridad de un proveedor. Travelex Holdings, una empresa con sede en Londres, suspendió sus servicios en 30 países por otro agujero en sus sistemas. No hay que ir muy lejos, El Instituto Nacional de Ciberseguridad (INCIBE) gestiona más de 100.000 incidentes al año de empresas y particulares, de los que unos 700 corresponden a operadores estratégicos (desde eléctricas hasta empresas de telecomunicaciones, puertos...). Estos ejemplos citados son tan sólo una pequeña parte de todos los ciberataques que suceden diariamente sin distinción de lo compleja o grande que sea la organización afectada, desde pequeñas pymes hasta multinacionales y organizaciones de prestigio internacional.

Incidentes de ciberseguridad

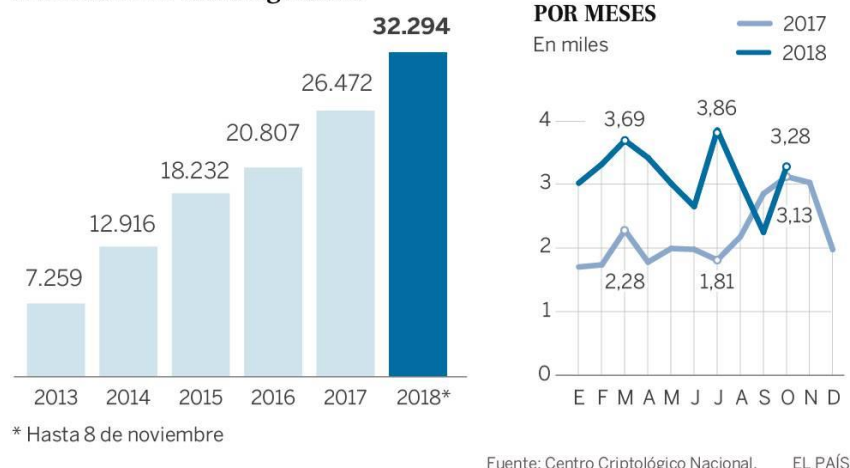


Ilustración 1. Aumento de incidentes de ciberseguridad 2013-2018. CCN

Debido al aumento de los incidentes de ciberseguridad en los últimos años se ha puesto el foco en cómo prevenir estos ciberataques y cómo reducir su impacto (tanto económico como reputacional y de fuga de información confidencial), sin duda es de lo más que se habla. Sin embargo, es más desconocida la pata que subyace bajo todos los cibercrímenes: su análisis forense. Y es que, al igual que sucede con los asesinatos y otros delitos físicos, cuando se produce un incidente informático de cierta entidad, es esencial analizar las causas del ataque y su origen, para poder evitar nuevas réplicas y, por supuesto, evitar ataques futuros.

Para ello, los analistas informáticos emplean distintas técnicas y metodologías de análisis forense, así como software especializado que permiten a la Policía o a los equipos de seguridad de las empresas conocer todos los detalles del atacante y su “*modus operandi*” (patrones de comportamiento, vectores de entrada, etc.). Por lo tanto, teniendo en cuenta la importancia de los soportes tecnológicos que almacenan la información (en cualquier ámbito, desde empresas y organizaciones a equipos del hogar) con relación a la fiabilidad que estos presentan y, por otro lado, las diferentes técnicas y herramientas de análisis forense, se pretende en este TFM tratar estos dos temas de manera conjunta, es decir, la finalidad de este TFM será precisamente la exposición de una de estas técnicas de peritaje o análisis forense sobre un soporte de almacenamiento con el objetivo de ver los posibles problemas o errores que estos puedan tener.

Linux es el kernel más usado en sistemas empujados, dispositivos cuya disponibilidad pública convierte el kernel en el punto crítico de muchos de los sistemas de seguridad integrada (validación de firmware, contraseñas, semillas de cifrado, raíces hardware, etc.). Diferentes configuraciones de placa son susceptibles ante ataques de lectura de memoria RAM en caliente [6], ya sea en bus SCI (ataques JTAG tipo BusPirate¹) o buses de RAM conectados a la CPU (ataques con electrónica Logic Analyzer Module²). En este TFM partiremos de volcados de memoria característicos de sistemas embebidos y estudiaremos alguna de las herramientas de análisis de memoria del kernel, para identificar las posibles debilidades y analizar medidas de mitigación a nivel de análisis y diseño.

¹ “Bus Pirate”, https://en.wikipedia.org/wiki/Bus_Pirate

² “Logic analyzer”, https://en.wikipedia.org/wiki/Logic_analyzer

b. Objetivos del Trabajo

El objetivo principal será el análisis de volcados de memoria típicos de IoT en sistemas Linux para identificar las posibles debilidades y analizar medidas de mitigación a nivel de análisis y diseño, utilizando para ello herramientas de análisis de memoria del kernel. Para conseguir este objetivo, se seguirá la vertiente de arquitectura software, es decir, se analizará y estudiará las diferentes herramientas y técnicas para el análisis de volcados de memoria disponibles en Linux y, posteriormente, se trataría de evaluar un volcado de memoria en concreto con la herramienta escogida, con el fin de identificar debilidades y proponer el diseño o alternativas existentes para los problemas detectados.

Por lo tanto, los objetivos se podrían dividir con relación al tipo o el nivel al que van dirigidos. Podemos diferenciar los siguientes:

1. Objetivos a nivel de investigación y estudios.
 - a. Investigación sobre las diferentes herramientas y técnicas para análisis de volcados de memoria Linux con el fin de escoger las herramientas que más nos convenga para el caso a tratar. Se tratará de buscar las características de cada uno, ventajas e inconvenientes y se propondrá ejemplos prácticos de uso de cada herramienta analizada.
 - b. Una vez realizada la investigación previa de las herramientas disponibles, convendría ampliar la información de la herramienta escogida, con el fin de acceder a otros estudios y datos sobre la aplicación de dicha herramienta en un entorno de prueba.
 - c. Ampliar el conocimiento buscando información y definiciones de conceptos esenciales para el entendimiento del proyecto, desde conceptos más generales de seguridad informática hasta conceptos específicos de análisis forense y peritaje, incluyendo metodologías de trabajo.
 - d. Conocer el estado del arte y la situación general de las herramientas y técnicas de análisis de memoria de kernel y dispositivos IoT.
2. Objetivos a nivel de implementación o desarrollo.
 - a. Saber aplicar las herramientas y técnicas de análisis de volcados de memoria del kernel sobre un caso práctico de manera correcta con la finalidad de observar cómo se comporta con la muestra y la información y datos que estas herramientas puedan generar.
 - b. Saber generar un informe de resultados en vista a los valores obtenidos.
 - c. Identificar los posibles errores y problemas tanto a nivel de diseño como a nivel de implementación.
 - d. Proponer medidas de mitigación frente a los problemas detectados durante el análisis y en vista a los datos obtenidos.
3. Objetivos a nivel académico/entregas.
 - a. Disponer de las diferentes entregas parciales en tiempo y forma.
 - b. Desarrollar una memoria de TFM rica en información, así como en calidad escrita y orden.
 - c. Generar un PPT y video defensa que sintetice de manera clara todo el proyecto.

c. Enfoque y metodología seguida

El enfoque del proyecto consiste en realizar un análisis de un volcado de memoria en sistemas Linux, previamente estudiando las diferentes herramientas existentes y seleccionando las más apropiada para el análisis teniendo en cuenta el sistema utilizado, con el fin de conocer la fiabilidad o errores en dichos soportes y cómo solucionarlos. Por lo tanto, podemos dividir el proyecto en dos partes, una primera parte teórica y otra segunda parte práctica:

- La primera parte teórica es referente a la investigación. Consiste en hacer un estudio del arte de las herramientas de análisis de volcados de memoria ya existentes. Además, se profundizará en diversos conceptos de seguridad y de análisis forense para establecer una base de conocimiento común, definiendo una serie de conceptos esenciales para el entendimiento de la memoria. Se apoyará esta base de conocimiento con la elaboración de un glosario, con los términos más utilizados a lo largo de la memoria. Dicha parte teórica mayoritariamente estará terminada con la segunda entrega parcial.
- La segunda parte consiste en seleccionar las herramientas más apropiadas analizadas en el estudio previo y realizar con ellas un análisis de un volcado de memoria estándar de tipo IoT. Seguidamente, se generará un informe de resultados y se expondrá, en la medida de lo posible, medidas de seguridad/soluciones a los problemas encontrados durante el análisis. Esta segunda parte corresponderá a la mayor parte del proyecto siendo mayoritariamente de enfoque práctico. Todos los datos generados durante el análisis a través de las herramientas se incluirán en un anexo al final de la memoria para tenerlos accesibles en todo momento. Esta parte se tendrá terminada según la planificación en la tercera entrega parcial del proyecto, dejando los flecos finales para la entrega de la memoria final.

Considerando la división en una parte teórica y en una parte práctica, conviene disponer de la base de conocimientos adquirida previo inicio de la parte práctica, por lo que inicialmente se empezará por conseguir los objetivos relacionados con el estudio y la investigación antes de adentrarse al caso práctico.

La estrategia a la hora de realizar la memoria será ir redactando la memoria cada vez que se cumplan los objetivos, es decir, a la par de la investigación y del análisis propiamente dicho. Esto nos permite llevar una mejor organización del proyecto, gestionar el tiempo de una mejor manera y, sobre todo, nos permite elaborar las diferentes entregas parciales a lo largo del proyecto, pudiendo conseguir los hitos/tareas a la par que se va redactando la memoria. Con respecto a la memoria, toda fuente a la que se ha accedido, así como los términos consultados (o los más destacados) se añadirán a la bibliografía y al glosario respectivamente al final de la memoria.

Respecto a la metodología utilizada durante el análisis del volcado de memoria propiamente dicho, se hará uso de la metodología general utilizada para el análisis forense [1]. Esta se basa principalmente en:

1. Adquisición: se obtienen copias de la información que se sospecha que puede estar vinculada con algún incidente o problema.
2. Preservación: en esta etapa se debe garantizar la información recopilada con el fin de que no se destruya o sea transformada. Es decir que nunca debe realizarse un análisis

sobre la muestra incautada, sino que deberá ser copiada y sobre la copia se deberá realizar la pericia.

3. Análisis: es la fase más técnica, donde se utilizan tanto hardware como software específicamente diseñados para el análisis forense.
4. Documentación: aquí ya debemos tener claro por nuestro análisis qué fue lo sucedido, e intentar poner énfasis en cuestiones críticas y relevantes a la causa. Debemos citar y adjuntar toda la información obtenida, estableciendo una relación lógica entre las pruebas obtenidas y las tareas realizadas, asegurando la repetibilidad de la investigación.
5. Presentación: es una exposición que nos detalla en mayor grado y precisión todo el análisis realizado, resaltando técnicas y resultados encontrados, poniendo énfasis en modo de observación y dejando de lado las opiniones.

d. Listado de tareas

Para poder superar los objetivos propuestos se propondrá una serie de tareas o hitos que nos permita guiar y organizar el desarrollo del proyecto. Como se ha comentado, existen objetivos ligados al estudio y objetivos ligados al diseño e implementación, por lo tanto, las tareas también se agruparán en distintos bloques. Un primer bloque relacionado con la planificación del proyecto, un segundo bloque relacionado con la investigación, un tercer bloque ligado a la planificación/implementación y un último bloque encargado de la preparación de la defensa del TFM, así como finalizar la memoria. Por lo tanto, podemos resumir las tareas de la siguiente manera:

1. Bloque1. Empezar a realizar la memoria del TFM estableciendo el Plan de Trabajo como punto de partida. Al finalizar el bloque 1 se procederá a la entrega “Plan de Trabajo”.
2. Bloque2. Tareas ligadas a la investigación. Una vez terminadas las tareas del segundo bloque, se procederá a la entrega parcial denominada “Entrega 2”.
 - a. Estudio de herramientas de análisis de volcados de memoria.
 - i. Explicación de las características de cada herramienta, funcionalidad de cada una de ellas y ejemplos prácticos de uso.
 - b. Adquisición y estudio de las muestras/volcado del caso práctico.
3. Bloque 3. Implementación. Una vez terminadas las tareas ligadas al bloque 3, se procederá a la entrega parcial denominada “Entrega 3”.
 - a. Implementación de las técnicas y herramientas al caso práctico a tratar.
 - b. Generación de un informe con los resultados obtenidos.
 - c. Proponer alternativas, diseños o soluciones para los problemas detectados durante el análisis.
4. Bloque 4. Presentación y defensa.
 - a. Finalización de la memoria TFM y entrega final.
 - b. Generación del video/PPT de la memoria del TFM.

Estas son las tareas principales por tratar. En el siguiente apartado se hará un mayor desglose de las tareas y se planificarán a lo largo del trimestre.

e. Planificación temporal del trabajo

Se dispone de un horizonte temporal aproximado de 3 meses para realizar el proyecto en su totalidad. Teniendo esto en cuenta y para mejorar la organización se dispondrá de una planificación dividida en 4 bloques, uno por entregable. Las tareas sombreadas en naranja en la siguiente planificación engloban cada bloque de trabajo, siendo el primer bloque la planificación del trabajo, el segundo lo referente al análisis, el tercero lo referente al diseño e implementación y el último bloque corresponde a la entrega de la memoria final. Las tareas en rojo son los hitos/entregas que se pretenden conseguir y que finalizan su correspondiente bloque.

Nombre	Duración	Inicio	Fin
Definición y planificación del TFM	7días?	24/02/2020	03/03/2020
Definir el contexto y justificación del trabajo	2días?	24/02/2020	25/02/2020
Definir los objetivos del TFM	2días?	24/02/2020	25/02/2020
Definir la metodología a seguir	2días?	25/02/2020	26/02/2020
Listar las tareas a desarrollar	2días?	26/02/2020	27/02/2020
Realizar la planificación temporal del TFM	2días?	26/02/2020	27/02/2020
Definir el estado del arte	2días?	27/02/2020	28/02/2020
Redactar el sumario de productos obtenidos	2días?	27/02/2020	28/02/2020
Elaboración de los resúmenes (inglés y español)	2días?	02/03/2020	03/03/2020
Entrega del plan de Trabajo	1día?	02/03/2020	02/03/2020
Análisis e investigación	19días?	04/03/2020	30/03/2020
Definición de conceptos de seguridad informática	3días?	04/03/2020	06/03/2020
Definición de conceptos de análisis forense	2días?	09/03/2020	10/03/2020
Estudio de herramientas de análisis de volcados (características, funcionalidad y ejemplos)	7días?	11/03/2020	19/03/2020
Indicar pros y contras de las herramientas analizadas	1día?	20/03/2020	20/03/2020
Creación del glosario de términos	2días?	23/03/2020	24/03/2020
Adquisición del volcado de memoria	1día?	25/03/2020	25/03/2020
Estudio de la muestra del volcado de memoria	2días?	26/03/2020	27/03/2020
Entrega 2. Análisis.	1día?	30/03/2020	30/03/2020
Diseño e implementación de las herramientas/técnicas de análisis sobre el volcado	20días?	31/03/2020	27/04/2020
Seleccionar las herramientas para el análisis de la muestra	1día?	31/03/2020	31/03/2020
Aplicación de las técnicas y herramientas sobre la muestra	5días?	01/04/2020	07/04/2020
Almacenar los datos y la información obtenida en los anexos	1día?	07/04/2020	07/04/2020
Generar un informe con los datos obtenidos durante y después del análisis	5días?	08/04/2020	14/04/2020
Identificar los problemas y debilidades en la muestra	3días?	15/04/2020	17/04/2020
Proponer medidas de mitigación a nivel de análisis y diseño	5días?	20/04/2020	24/04/2020
Entrega 3. Diseño e implementación.	1día?	27/04/2020	27/04/2020
Presentación y defensa del TFM	24días?	29/04/2020	01/06/2020
Concluir el TFM (bibliografía, anexos, glosarios, conclusiones)	23días?	29/04/2020	29/05/2020
Revisión de la memoria TFM	18días?	06/05/2020	29/05/2020
Entrega 4. Memoria Final	1día?	01/06/2020	01/06/2020
Generar el PPT y el video para la defensa	10días?	18/05/2020	29/05/2020
Defensa del TFM	1día?	01/06/2020	01/06/2020

Ilustración 2. Planificación temporal del proyecto

TÉCNICAS Y HERRAMIENTAS PARA EL ANÁLISIS DE DEBILIDADES EN VOLCADOS DE MEMORIA RAM DE SISTEMAS BASADOS EN LINUX

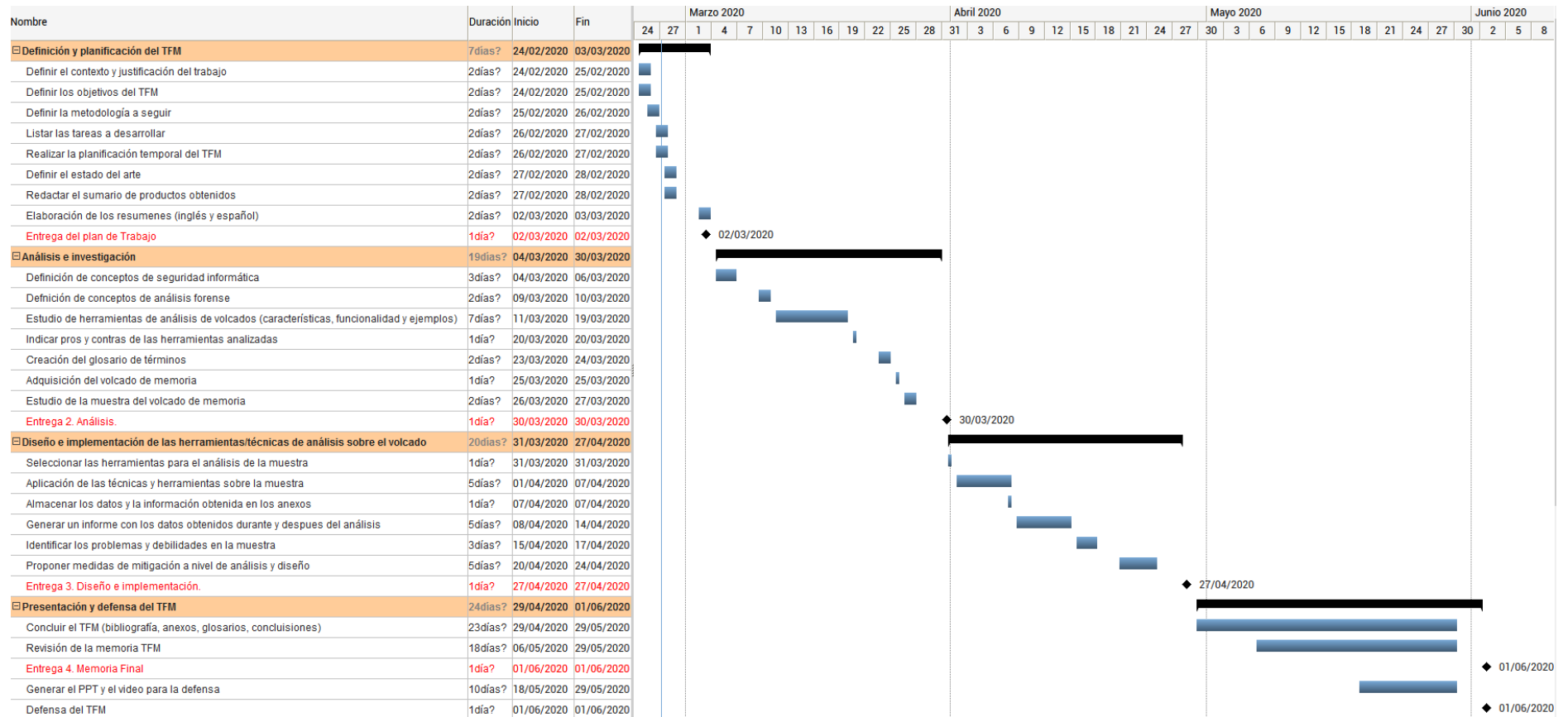


Ilustración 3. Diagrama de Gantt con la planificación temporal.

f. Estado del arte

Los cibercriminales mejoran sus técnicas a diario para lograr penetrar, incluso en los sistemas informáticos mejor protegidos, los cuales son custodiados fuertemente por una gran cantidad de controles administrativos, técnicos y por un equipo de administradores especialistas en respuesta y análisis de incidentes informáticos. En esta carrera, solo triunfarán aquellos con la capacidad para dominar los métodos, técnicas y herramientas más avanzadas para superar a los adversarios. Algunas de las armas más poderosas de los atacantes son los rootkits, que son malware que permite ocultar los procesos maliciosos, puertas traseras y archivos que se utilizan para tomar control de los sistemas comprometidos. Los rootkits consisten en uno o más programas y código que permiten mantener acceso permanente e indetectable en un ordenador. Los rootkits se pueden clasificar en dos grupos, los que van integrados en el núcleo/kernel y los que funcionan a nivel de aplicación:

- Los que actúan desde el kernel añaden o modifican una parte del código de dicho núcleo para ocultar el backdoor. Normalmente este procedimiento se complementa añadiendo nuevo código al kernel, ya sea mediante un controlador (driver) o un módulo, como los módulos del kernel de Linux o los dispositivos del sistema de Windows. Estos rootkits suelen *parhear* las llamadas al sistema con versiones que esconden información sobre el intruso. Son los más peligrosos, ya que su detección puede ser muy complicada.
- Los rootkits que actúan como aplicaciones pueden reemplazar los archivos ejecutables originales con versiones *crackeadas* que contengan algún troyano, o también pueden modificar el comportamiento de las aplicaciones existentes usando *hacks*, parches, código inyectado, etc.

Existen principalmente 2 opciones para detectar este tipo de malware: la primera es ejecutar programas especializados en la detección de rootkits, como Rootkit Hunter³ o Chkrootkit⁴ (para sistemas basados en UNIX). Algunos de estos programas, además de ser capaces de detectar rootkits conocidos, incorporan alguna funcionalidad genérica para detectar nuevas amenazas. Sin embargo, no existe la seguridad de que un nuevo rootkit utilice esas funcionalidades genéricas, por lo tanto, no hay garantía de que estas herramientas logren detectarlos. La segunda manera, que es de lo que va este TFM, es el análisis del volcado de memoria (además de ser la que ofrece una mayor posibilidad de detectar rootkits a nivel de kernel). Para explicar cómo funciona, conviene recordar algunos conceptos de arquitectura de computadores.

Un computador tiene dos tipos de memorias, una principal y una secundaria. La primera es la de mayor velocidad de lectura, intercambia datos constantemente con el procesador y es de menor tamaño que la memoria secundaria. Almacena la información de forma temporal mientras el ordenador se encuentre encendido y, en cuanto se apaga, la información se pierde (volátil). Volcar la memoria consiste en copiar el contenido de la memoria principal en un archivo, el cual puede ser analizado posteriormente para obtener información del estado del ordenador en el momento del volcado.

¿Por qué es tan importante incluir el análisis de volcado de memoria en una investigación forense computacional hacia atacantes informáticos? Porque este tipo de intrusos utilizan el

³ Rootkit Hunter, <https://es.wikipedia.org/wiki/Rkhunter>

⁴ Chkrootkit, <https://es.wikipedia.org/wiki/Chkrootkit>

cifrado y ofuscación para protegerse. Los más sofisticados, incluso utilizan herramientas que nunca escriben información en memoria secundaria. Sin importar qué tan perfeccionadas sean las técnicas empleadas por los atacantes informáticos, todo programa que se ejecuta en un ordenador, en algún momento se almacena en memoria principal. Por esa razón, es muy probable que el volcado de memoria contenga los programas utilizados por los atacantes informáticos o, al menos, rastros de ellos.

Desde hace años, los analistas forenses computacionales han capturado el contenido de la memoria. Su análisis consistía en la extracción de cadenas de texto para luego buscar direcciones IP o URLs que podían dar un gran contexto al investigador forense, pero no se lograba obtener otras estructuras de la memoria. En 2007 se creó Volatility, una herramienta para interpretar el contenido de la memoria, que va mucho más allá de las técnicas tradicionales de búsqueda de cadenas de texto. Este poderoso programa tiene la capacidad de interpretar las estructuras internas de memoria que almacena, entre otras cosas, la información de los procesos en ejecución y conexiones de red que estaban activas en el momento en que se capturó la memoria. Incluso proporciona información de conexiones de red y procesos ya finalizados para el momento en que se realizó la captura. Los cuales pueden ser indicios claves para resolver un caso. Otras herramientas útiles pueden ser:

- OSForensics. Es una suite que te permite conducir un escaneo a fondo del ordenador en busca de cualquier pieza de evidencia que pueda ofrecerte una pista, verificando todo, desde los archivos de email, archivos borrados, incluso el historial del navegador. Además, te permite organizar la evidencia creando casos separados, lo que te permitirá mantener todos los datos separados.
- Redline de Mandiant. La principal herramienta gratuita de seguridad informática de la empresa FireEye. Brinda herramientas de investigación de sistemas a los usuarios para encontrar signos de actividad maliciosa a través de la memoria y el análisis de archivos.

Los rootkits no son los únicos problemas que nos podemos encontrar, ni Linux el único sistema vulnerable, ni mucho menos. Por ejemplo, en el estudio llamado *“Acquisition and analysis of volatile memory from android devices”* por Joe Sylve y Andrew Caseb [28], presentan una metodología y un conjunto de herramientas para la adquisición y el análisis de la memoria física volátil de dispositivos Android. Según el estudio, es posible hacer un volcado completo de la memoria, incluido tarjetas SD, de dispositivos Android no sólo directamente desde el dispositivo, sino a nivel de red utilizando para ello un módulo del kernel llamado *dmd*⁵. Además, en el mismo estudio, presenta una nueva funcionalidad de Volatility para analizar dicho volcado de memoria.

Además de los estudios, existen libros completos dedicados a hablar sobre análisis forense en memorias. Por ejemplo, Michael Hale Ligh, Andrew Case y Jamie Levy presentan *“The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux and MAC memory”* [29] donde exponen cómo se comportan diferentes tipos de malware en los sistemas operativos más utilizado hoy en día, como son Linux, Windows y MAC. Un ejemplo puede ser el caso de los rootkits visto anteriormente. Hoy día el análisis forense sobre soportes de memoria RAM es un tema bastante tratado dentro de la comunidad científica y estudiantil, sobre todo relacionado con ciberataques y malware. Por ejemplo:

⁵ DMD (LiME), <https://github.com/504ensicsLabs/LiME>

- D. Paul Joseph et al. [30] establecen la importancia de los forenses en las investigaciones y frente a los ciberataques, en especial, en el análisis forense en memorias, los cuales pueden presentar pruebas importantes en casos jurídicos. Además, presentan herramientas de mitigación y casos de ejemplos, por ejemplo, el malware WannaCry.
- Amer Aljaedi et al. [31] comparan el método que tradicionalmente se solía usar para el análisis de memoria volátil que es la respuesta en tiempo real frente a la adquisición de imágenes/volcados de memoria de estos para su posterior análisis (el usado en este TFM), estableciendo las ventajas y desventajas que presentan ambos métodos, herramientas, etc.
- Por último, Steffen Logen et al. [32] en el que, al igual que en este TFM, implementa la herramienta Volatility para el análisis de volcados de memoria, utilizando para ello una extensión y un entorno gráfico que simplifica la utilización del framework.

En definitiva, no cabe duda de que, tanto los atacantes informáticos como los investigadores forenses digitales, tendrán que mejorar sus técnicas, métodos y herramientas constantemente, con el fin de mantener la ventaja en la guerra cibernética, la guerra de la era digital. El análisis de volcado de memoria será cada vez más importante para resolver con éxito los casos de intrusiones informáticas avanzadas.

g. Breve resumen de productos obtenidos

El TFM se divide en un conjunto de entregas parciales como se ha comentado en anteriores ocasiones y que, en su totalidad, se corresponderá con el resultado de la memoria completa, permitiendo de esta manera una mejor organización tanto práctica como temporal. Dichas entregas son las siguientes:

- 1ª Entrega. Plan de Trabajo. En esta primera entrega se incluirá la explicación detallada del problema a resolver, los objetivos del proyecto, la metodología utilizada, las tareas y la planificación a seguir durante el proyecto, estado del arte y principales secciones de la memoria.
- 2ª Entrega. Fase de estudio e investigación. En esta entrega se tratará la mayor parte de las tareas u objetivos teóricos, como son el estudio de las herramientas de análisis de volcados de memoria del kernel, el estudio del volcado a tratar en el experimento, definición y adquisición de las bases teóricas del proyecto, etc.
- 3ª Entrega. Implementación. Realización del análisis del volcado de memoria. En esta entrega se incluirá la parte práctica, en la cual se aplica la herramienta escogida al caso práctico del proyecto, se generará el informe de resultados y se propondrá las posibles soluciones a los problemas encontrados. En esta entrega también se añadirán los anexos con la salida de la implementación de las herramientas en la muestra a tratar.
- 4ª Entrega. Memoria final. Esta será la última entrega del proyecto. En ella se juntan las anteriores entregas. Además, se completarán las demás secciones como puede ser el glosario y la bibliografía.
- Adicionalmente también se hará la entrega del video para la defensa del proyecto. Para ello se creará una presentación PPT con una síntesis del TFM.

h. Breve descripción de los capítulos de la memoria

El proyecto se divide principalmente en 2 partes diferenciadas. Una primera parte de investigación, mayoritariamente teórica, y una segunda parte de implementación práctica. Para entender correctamente y de forma fluida la parte práctica será necesario disponer de una base de conocimiento previa. Por lo tanto, el primer capítulo de la memoria integrará los conocimientos necesarios para el correcto entendimiento. En este primer capítulo encontraremos esencialmente:

- Apartado con conceptos de “seguridad” y “análisis forense” básicos para el correcto entendimiento. Un ejemplo podría ser las diferencias entre memoria principal y secundaria en un sistema informático, esencial para entender los volcados de memoria y la necesidad de adquirir y preservar la información. También se podría explicar diferentes metodologías de trabajo, técnicas forenses y ampliar el estado del arte previamente visto indicando nuevos estudios interesantes que puedan aportar más valor al TFM.
- Apartado con el estudio de las principales herramientas de análisis de memoria kernel. Con especificación de las características de cada herramienta, puntos a favor y en contra de cada una, ejemplos prácticos, etc. La idea es disponer de información clara y concisa de manera objetiva para poder formarnos una conclusión a posteriori de cuál es la herramienta que nos conviene para el caso.
- Apartado con la comparación de herramientas entre sí. La idea principal es decidir qué herramienta o herramientas serán las escogidas para realizar el caso práctico dependiendo de las características que busquemos.
- Apartado con el estudio del volcado de memoria a tratar. Se hará un estudio sobre las muestras con la que se trabajará, explicando la composición de ellas, tamaño y características principales para tener en cuenta.

Una vez tengamos definido la base de conocimientos ya se podría dar paso a la segunda parte relacionada con la implementación práctica:

- Apartado con el análisis del volcado de memoria. En esta sección se aplicará las diferentes técnicas y herramientas anteriormente escogidas sobre el volcado de memoria de tipo IoT. Se indicarán los pasos a seguir, comandos utilizados, capturas de pantallas con la salida de los datos, etc.
- Apartado con los resultados obtenidos. Una vez realizado el análisis, se expondrá los resultados obtenidos, conclusiones, problemas identificados, etc.
- Apartado con las modificaciones/soluciones a los problemas encontrados. En esta sección se intentará dar soluciones a los problemas encontrados en la sección anterior de tal manera que se puedan mitigar para evitar fallos futuros.

Adicionalmente, se añadirá un último apartado de conclusiones. En dicho apartado se indicará si se han alcanzado los objetivos propuestos, problemas de la metodología seguida a lo largo del trabajo, problemas de implementación, etc. Adicionalmente se expondrá una visión futura sobre el estado del arte actual, propuestas de mejoras y en el caso de que no se hayan cumplido algún objetivo o alguna tarea, se indicará correctamente. Por último, destacaría la bibliografía donde se indicará todas las fuentes de información consultadas durante la realización del trabajo y el glosario con los principales conceptos del TFM y abreviaciones.

Conceptos generales.

Antes de comenzar con el estudio de las herramientas de análisis de volcados de memoria, es necesario establecer unos conocimientos generales (estado del arte) que nos permitan entender todo el proceso posterior de análisis. Además, se pretende introducir el concepto de análisis forense explicando términos, metodología de trabajo, objetivos, etc. con el objetivo de situar el análisis de soportes de información dentro de las etapas del análisis forense. Adicionalmente, conviene aclarar el porqué de las herramientas de análisis del volcado, es decir, por qué han sido desarrollados, qué necesidades satisfacen y qué función cumplen en el análisis forense.

Conceptos de seguridad informática.

Sistemas distribuidos

Un sistema distribuido se define como una colección de computadores separados físicamente y conectados entre sí por una red de comunicaciones (conjunto de equipos nodos y software conectados entre sí por medio de dispositivos físicos o inalámbricos para el transporte de datos, con la finalidad de compartir información, recursos y ofrecer servicios); cada máquina posee sus componentes de hardware y software que el programador percibe como un solo sistema (no necesita saber qué cosas están en qué máquinas). El programador accede a los componentes software de manera remota a través de un middleware (software que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él), entre los que destacan RPC⁶ y SOAP⁷. Los sistemas distribuidos presentan las siguientes características:

- Seguridad interna en el sistema distribuido.
- Se ejecuta en múltiples ordenadores.
- Interacción entre los equipos y dependencia de redes (LAN, MAN, WAN, etc.).
- Compatibilidad entre los dispositivos conectados.
- Transparencia (el uso de múltiples procesadores y el acceso remoto debe ser invisible).
- Diseño de software compatible con varios usuarios y sistemas operativos.

IoT (Internet of Things)

La definición de IoT podría ser la agrupación e interconexión de dispositivos a través de una red (bien sea privada o Internet, sin intervención humana), donde todos ellos podrían ser visibles e interactuar [26]. Respecto al tipo de dispositivos podrían ser cualquiera, desde sensores y dispositivos mecánicos hasta objetos cotidianos como pueden ser el frigorífico, el calzado o la ropa. El objetivo por tanto es una interacción de máquina a máquina, o lo que se conoce como una interacción M2M (*machine to machine*) o dispositivos M2M.

⁶ RPC, <https://es.wikipedia.org/wiki/XML-RPC>

⁷ SOAP, https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol

IoT. Tecnología emergente.

Internet ha evolucionado rápidamente y esto ha permitido que IoT sea ya una realidad y no sólo una visión de futuro. La fama de esta tecnología radica principalmente en todas las aplicaciones y posibilidades que nos proporciona tanto para mejorar la vida cotidiana de las personas como los entornos empresariales, donde ya se está implantando desde hace algún tiempo. Las aplicaciones son casi infinitas, pero se van a describir algunos ejemplos para dar visibilidad de alguna de ellas, tanto en la vida cotidiana como en el entorno empresarial:

- Supongamos el frigorífico de una casa, donde se conservan los alimentos que, a su vez, tienen una fecha de caducidad. En este escenario, se podría conectar el frigorífico a internet para que avisara al usuario a través de su teléfono móvil, por ejemplo, de cuando caducan los alimentos, si hay una bajada de temperatura por alguna avería, si algún alimento se ha está acabando o simplemente el consumo de electricidad.
- Otro escenario podría ser el de la domótica, dónde ya hay numerosos dispositivos que se conectan a Internet para facilitar la vida de los seres humanos, véase por ejemplo los dispositivos controlados por voz a los que se les solicita que reproduzca una canción desde un repositorio en internet, o los dispositivos y aplicaciones que permiten controlar todos los parámetros del agua de un acuario, o incluso los sistemas de alarmas de las casas que se conectan con las centrales. Los sistemas de seguridad que se conectan a la red para avisarte cuando alguien entra en tu casa o aquellos dispositivos que permiten encender la calefacción desde un teléfono móvil.
- Si se piensa en aplicaciones industriales, IoT es usado ya en muchas plantas de producción donde los dispositivos y sensores conectados a la red permiten analizar los datos y generar alarmas y mensajes que son enviados a los distintos usuarios para que tomen las acciones necesarias o incluso iniciar protocolos de actuación de forma automática, sin interacción humana, para corregir o tratar dichas alarmas.
- Otro ejemplo de aplicación sería en el sector ganadero dónde la monitorización biométrica y la geolocalización es un factor que ayuda a los ganaderos a que sus animales estén siempre controlados.

Fileless malware.

En el estado del arte [16] se introdujo una de las problemáticas que existe con la memoria RAM, que no es otra que la posibilidad de que exista malware oculto en los procesos del sistema dentro de la memoria RAM, como pueden ser los rootkits. Este tipo de malware se pueden denominar como *fileless malware*.

En cualquier estrategia de concienciación en ciberseguridad empresarial, los empleados tienen un mandamiento esencial: nunca hay que abrir archivos adjuntos de un email si no se está completamente convencido de que es seguro. En caso contrario, su descarga o apertura puede provocar una crisis de seguridad informática en toda la empresa.

Pero ¿qué pasa cuando el malware que inunda un ordenador no se inserta dentro de un archivo, sino mediante un proceso difícilmente detectable? Eso es precisamente lo que ocurre cuando las empresas se enfrentan al *fileless malware*.

¿Qué es el fileless malware?

El *fileless* malware, también llamado '*malware sin fichero*', se produce cuando el malware no entra en nuestro ordenador a través de un documento específico, sino que en realidad se instala dentro de la memoria RAM del propio equipo y se desarrolla con distintos procesos.

Una vez ejecutado, esta técnica para el cibercrimen tiene diversas formas de actuación: Anthrax⁸ afecta a los archivos del sistema, Phase Bot⁹ sirve como kit de configuración de malware para otros ciberdelincuentes y Poweliks¹⁰ altera los servidores para abrir nuevas puertas de infección, por poner algunos ejemplos. Con esta táctica, el *fileless malware* consigue no ser fácilmente detectado por el usuario en cuestión, además de escapar del control de las soluciones de ciberseguridad que no estén específicamente preparadas para detectar este tipo de intrusiones.

¿Cómo evitar el fileless malware?

El auge y progreso de este tipo de cibercrimen obliga a las empresas a tomar medidas para evitar nuevas infecciones. Algunas de las más recomendables son las siguientes:

1. Ser ciber-resiliente. Es el consejo más obvio, pero también el más importante: el cibercrimen readapta y reinventa sus estrategias a diario, así que cualquier compañía que quiera proteger su ciberseguridad empresarial debe ser ciber-resiliente y mantenerse al tanto de los nuevos tipos de ataques que puedan producirse.
2. Soluciones adaptadas. La mayor ventaja con la que juega el *fileless* malware es que, al no operar desde un archivo, sino en la memoria RAM, no es detectado por muchas de las soluciones de captación de vulnerabilidades.
3. Lenguajes de scripting. En muchas ocasiones el malware sin fichero se aprovecha de la existencia de herramientas que recurren a lenguajes de scripting como Powershell. En la medida de las posibilidades, las compañías deberán prescindir de ellos.
4. Cuidado con las macros. Las macros constituyen una de las herramientas más presentes en cualquier equipo, pero también una posible puerta de entrada para este tipo de cibercrimen. Al igual que con los lenguajes de scripting, no es obligatorio que las empresas prescindan de todo tipo de macros, pero sí que hagan un uso responsable y limitado de ellas.

Debido a todos los puntos nombrados anteriormente el uso de herramientas de análisis de volcados de memoria RAM, para descubrir los posibles procesos maliciosos, cobran una mayor importancia.

Memoria principal.

La memoria principal es la memoria del ordenador donde se almacenan temporalmente tanto los datos como los programas que la unidad central de procesamiento (CPU) está procesando o va a procesar en un determinado momento. Por su función, la MP debe ser inseparable del

⁸ Anthrax, <https://malware.wikia.org/wiki/Anthrax>

⁹ Phase Bot, <https://www.malwaretech.com/2014/12/phase-bot-fileless-rootki.html>

¹⁰ Poweliks, <https://www.vmray.com/cyber-security-blog/poweliks-fileless-malware-analysis/>

microprocesador o CPU, con quién se comunica a través del bus de datos y el bus de direcciones. El ancho del bus determina la capacidad que dispone el microprocesador para el direccionamiento de direcciones en memoria.

La MP es el núcleo del subsistema de memoria de un sistema informático, y posee una menor capacidad de almacenamiento que la memoria secundaria, pero una velocidad millones de veces superior. Al bloque de MP, suele llamarse RAM, por ser este el tipo de chips de memoria que conforman el bloque. Esta clase de memoria es volátil, es decir que cuando se corta la energía eléctrica, se borra toda la información que estuviera almacenada en ella [10].

Memoria secundaria

La memoria secundaria es el conjunto de dispositivos y soportes de almacenamiento de datos que conforman el subsistema de memoria del ordenador, junto con la memoria primaria o principal. La memoria secundaria es un tipo de almacenamiento masivo y permanente (no volátil) con mayor capacidad para almacenar datos e información que la memoria primaria que es volátil, aunque la memoria secundaria es de menor velocidad.

Deben diferenciarse los “dispositivos o unidades de almacenamiento” de los “soportes o medios de almacenamiento”, porque los primeros son los aparatos que leen o escriben los datos almacenados en los soportes [11].

Memoria principal	Memoria secundaria
Rápida en la transferencia de archivos	Lenta en la transferencia de archivos
Alto valor económico	Bajo valor económico en comparación con la memoria principal
Baja capacidad de almacenamiento y volátil	Gran capacidad de almacenamiento y no volátil.
Trabaja directamente con la CPU	No trabaja directamente con la CPU

Dispositivos/unidades de almacenamiento y soportes/medios de almacenamiento

Un dispositivo de almacenamiento de datos es un conjunto de componentes electrónicos habilitados para leer o grabar datos en el soporte de almacenamiento de datos de forma temporal o permanente. Realizan operaciones de alfabetización física y lógica de los medios donde se almacenan los archivos de un sistema informático [12].

Por otra parte, el soporte de almacenamiento de datos o medio de almacenamiento de datos es el material físico donde se almacenan los datos que pueden ser procesados por una computadora, un dispositivo electrónico, y un sistema informático, aunque este término también abarca el concepto de documento no necesariamente informatizados (generalmente en papel, piedra, madera, material fotosensible, material magnético, etc.). Ejemplos de soportes manejados por computadoras: los discos magnéticos (disquetes, discos duros), los discos ópticos (CD, DVD, Blu-ray), las cintas magnéticas, los discos magneto-ópticos (discos Zip, discos Jaz, SuperDisk), las tarjetas de memoria, etc. [13]

Volcados de memoria

En informática, un volcado de memoria (en inglés *core dump* o *memory dump*) es un registro no estructurado del contenido de la memoria en un momento concreto, generalmente utilizado para depurar un programa que ha finalizado su ejecución incorrectamente. Se puede referir a un archivo que contiene una imagen en memoria de un proceso determinado o un listado impreso de todo el contenido de la memoria [15].

Como se comentó anteriormente [Estado del arte], los volcados de memoria surgen como solución a los problemas que suponían el análisis en tiempo real (en caliente) de la memoria RAM tal y como explican Amer Aljaedi et al. en "[Comparative Analysis of Volatile Memory Forensics: Live Response vs. Memory Imaging](#)" (por ejemplo, sobrescribir evidencias).

Proceso de carga en la memoria RAM

Para ejecutar una aplicación hay que cargarla en memoria RAM, a partir de ese momento el programa se convierte en un proceso que actualiza sus datos e instrucciones (los envía o recibe) con la ayuda del procesador.

Con el propósito de mejorar las prestaciones del sistema, se debe tratar de reducir el tiempo de acceso a memoria. La diferencia entre la RAM y otros tipos de memoria de almacenamiento, como los disquetes o discos duros, es que la RAM es mucho más rápida y volátil. La RAM necesita "refrescarse" cada cierto tiempo para impedir que se pierda la información.

Las posiciones de memoria están organizadas en filas y en columnas. Cuando se quiere acceder a la RAM se debe empezar especificando la fila, después la columna y por último se debe indicar si se desea escribir o leer en esa posición. En ese momento la RAM coloca los datos de esa posición en la salida, si el acceso es de lectura o coge los datos y los almacena en la posición seleccionada, si el acceso es de escritura.

La cantidad de memoria RAM de un sistema afecta notablemente las prestaciones que da, fundamentalmente cuando se emplean sistemas operativos actuales. En general, y sobre todo cuando se ejecutan múltiples aplicaciones, puede ser que la demanda de memoria sea superior a la realmente existente. En esos casos el sistema operativo puede forzar al procesador a simular dicha memoria sobre el disco duro (se conoce como memoria virtual).

Procesos informáticos

En informática, un proceso se trata básicamente de un programa que entra en ejecución (o conexiones activas: UDP/TCP y puertos). Los procesos son una sucesión de instrucciones que pretenden llegar a un estado final o que persiguen realizar una tarea concreta. Lo más importante de este concepto, es de dónde sale un proceso o qué es realmente un programa y un sistema operativo.

El sistema operativo es el software básico de un ordenador, con éste, el usuario es capaz de interactuar a partir de un entorno gráfico o mediante entradas de texto en forma de instrucciones. El sistema operativo es capaz de ejecutar otros procesos dentro de sí mismo e incluso crearlos mediante código de programación y una compilación.

Por su parte, un programa es un algoritmo que genera una secuencia de instrucciones con las que podemos realizar una tarea concreta. Por supuesto los programas actuales no solo realizan una, sino muchas tareas gracias a tener muchos de estos algoritmos en su código de programación, cada uno de ellos para una función específica.

Un proceso también se puede dividir en distintas partes para ver cómo se ejecuta en nuestro ordenador. Dentro de él tenemos lo que llamamos instrucciones, que corresponden a cada uno de los pasos que debemos hacer para completar esa tarea. Además, para separar cada proceso, el procesador le asigna un contador de programa, para que cada uno esté separado y bien diferenciado de otro que incluso puede ser igual, por ejemplo, abrir dos veces el explorador. De esta forma cada proceso se guarda en distintos registros, con distintas variables y por supuesto en distinta región de la memoria RAM.

Es en este punto, es en donde aparece el concepto de hilos de procesamiento o *threads*. Como sabemos, los sistemas actuales permiten ejecutar varios programas de forma simultánea, y, en consecuencia, tendremos gran cantidad de procesos activos en el sistema, decimos que son multihilo. Cada proceso entonces se divide en uno o varios hilos de ejecución o subprocesos. Cada hilo, tiene sus propias instrucciones y un estado de ejecución, es decir, unos valores en los registros con los que el procesador sabe en qué fase se encuentran.

Las formas de empezar un proceso informático serán las siguientes:

- Que nosotros ejecutemos un programa o el propio ordenador.
- Que el sistema llame a los programas o procesos: se ejecutará el *boot loader* del disco duro y el sistema comenzará a cargar procesos en memoria. O bien el sistema le pida a un programa, por ejemplo, un controlador, ejecutarse.

Y también se pueden terminar:

- Terminar la rutina o el programa: dando un resultado final que considera correcto
- Finalizar de forma repentina por un error: la rutina puede estar mal programada y no dar el resultado esperado
- A partir de otro proceso o por nosotros mismos: nosotros mismos podemos ejecutar una tarea que elimine el que se está ejecutando
- Se puede bloquear: si espera una determinada respuesta y ésta no llega el proceso permanecerá bloqueado hasta que el sistema detecte que no puede continuar.
- Por un corte de alimentación

El núcleo del sistema GNU/Linux

El núcleo o kernel [14] es la parte básica de cualquier sistema operativo, y sobre él descansa el código de los servicios fundamentales para controlar el sistema completo. Básicamente, su estructura puede separarse en una serie de componentes, o módulos de gestión orientados a:

- Gestión de procesos: qué tareas se van a ejecutar, en qué orden y con qué prioridad.
- Intercomunicación de procesos y sincronización: ¿cómo se comunican las tareas entre sí?, ¿con qué diferentes mecanismos y cómo pueden sincronizarse los grupos de tareas?
- Gestión de entrada/salida (E/S): control de periféricos y gestión de recursos asociados.
- Gestión de memoria: optimización del uso de la memoria, sistema de paginación y memoria virtual.
- Gestión de ficheros: cómo el sistema controla y organiza los ficheros presentes en el sistema, y accede a los mismos.

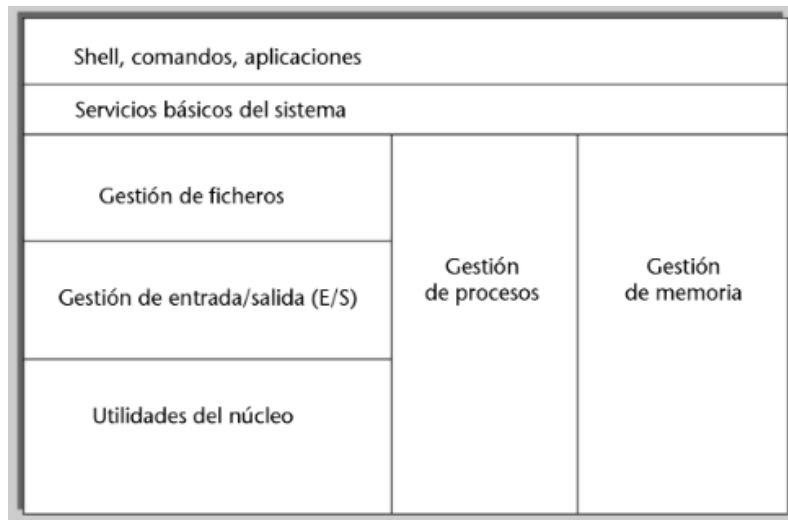


Ilustración 4. Funciones básicas de un núcleo.

En los sistemas privativos, el núcleo o kernel está perfectamente “oculto” bajo las capas del software del sistema operativo, y el usuario final no tiene una perspectiva clara de qué es ese núcleo ni tiene tampoco ninguna posibilidad de cambiarlo u optimizarlo, si no es por el uso de esotéricos editores de “registros” internos o programas especializados. Además, el núcleo suele ser único, es el que proporciona el fabricante, el cual se reserva el derecho de introducir las modificaciones que quiera y cuando quiera, así como tratar los errores que aparezcan en plazos no estipulados, mediante actualizaciones que nos ofrece como “parches” de errores.

En este caso, el núcleo Linux ofrece una solución de código abierto, con los consecuentes permisos de modificación, corrección, posibilidad de generación de nuevas versiones y actualizaciones de forma rápida, por parte de cualquiera que quiera y tenga los conocimientos adecuados para realizarlo. Esto permite a los usuarios críticos controlar mejor sus aplicaciones y el propio sistema, y poder montar sistemas con el propio sistema operativo “a la carta”, personalizado al gusto de cada uno. Al disponer del código fuente, se pueden aplicar mejoras y soluciones de forma inmediata, a diferencia del software privativo, donde debemos esperar a las actualizaciones del fabricante. Podemos, además, personalizar el núcleo tanto como necesitemos, requisito esencial en aplicaciones de alto rendimiento, críticas en el tiempo o en soluciones con sistemas empotrados (como dispositivos móviles).

Análisis forense.

El análisis forense digital se corresponde con un conjunto de técnicas destinadas a extraer información valiosa de discos, sin alterar el estado de estos. Esto permite buscar datos que son conocidos previamente, tratando de encontrar un patrón o comportamiento determinado, o descubrir información que se encontraba oculta. La informática forense tiene principalmente tres objetivos:

- La compensación de los daños causados por los intrusos o criminales.
- La persecución y procesamiento judicial de los criminales.
- La creación y aplicación de medidas para prevenir casos similares.

Estos objetivos se alcanzan de varias formas, siendo la principal la recopilación de evidencias.

Metodología

En el campo de la informática forense existen diversas etapas que definen la metodología a seguir en una investigación: identificación, preservación o adquisición, análisis y presentación de los resultados. Siguiendo el flujo lógico de actividades [7]:

1. Asegurar la escena se trata de la primera fase y no siempre se aplica, su objetivo es impedir que nadie pueda alterar algo.
2. Identificación de evidencias, se debe constatar qué evidencias deben recogerse para un análisis posterior, se deben identificar los dispositivos y sistemas a analizar y distinguir qué evidencias hay que destacar.
3. Adquisición de datos se trata de una fase crítica debido a la posibilidad de modificar por error alguna de las evidencias digitales. Un error de este estilo podría invalidar las pruebas en un posible proceso judicial.
4. Análisis de datos es la etapa en la que se busca información útil con relación a las evidencias, en este punto se estudian los ficheros donde puede estar la información eliminada, registros, logs del sistema, ficheros, etc.
5. Presentación e informe de resultados. Finalizado el análisis forense se debe redactar un informe pericial con conclusiones y justificación del sistema de trabajo empleado. El informe debe ser claro y conciso ya que puede terminar como prueba en los tribunales.

Evidencia digital

Los discos duros, las memorias USB y las impresoras (entre otros elementos) se pueden considerar evidencias en un proceso legal al igual que los volcados de memoria. Las evidencias digitales son las que se extraen de un medio informático. Estas evidencias comparten una serie de características que dificultan el ejercicio de la computación forense [8]: volatilidad, anonimato, facilidad de duplicación, alterabilidad y facilidad de eliminación

Estas evidencias se pueden dividir en tres categorías [8][9]:

- Registros almacenados en el equipo de tecnología informática (ej. imágenes y correos).
- Registros generados por equipos de tecnología informática (ej. transacciones, registros en eventos).
- Registros parcialmente generados y almacenados en los equipos de tecnología informática (ej. consultas en bases de datos).

Perspectiva de tres roles

En el análisis de un caso en el que sea necesario el cómputo forense, hay tres roles principales que son importantes y se deben tener en cuenta: el intruso, el administrador y la infraestructura de la seguridad informática, y el investigador:

- El intruso es aquel que ataca un sistema, hace cambios no autorizados, manipula contraseñas o cambia configuraciones, entre otras actividades que atentan contra la seguridad de un sistema.
- El administrador del sistema es el experto encargado de la configuración de este, de la infraestructura informática y de la seguridad del sistema. Estos administradores son los primeros en estar en contacto con la inseguridad de la información, ya sea por un atacante o por un fallo interno de los equipos. Al ser los arquitectos de la infraestructura y de la seguridad de la información del sistema, son quienes primero deberían reaccionar ante un ataque, y deben proporcionar su conocimiento de la infraestructura del sistema para apoyar el caso y poder resolverlo con mayor facilidad.
- El investigador es un nuevo profesional que actúa como perito, criminalista digital, o informático. Comprende y conoce las nuevas tecnologías de la información, y analiza la inseguridad informática emergente en los sistemas. El trabajo del informático es indagar en las evidencias, analizarlas y evaluarlas para poder decidir cómo estas evidencias pueden ayudar a resolver el caso. Por lo tanto, es ideal que un investigador conozca al menos sobre las siguientes áreas: justicia criminal, auditoría, administración y operación de tecnologías de Información.

Tal y como se ha comentado, el análisis forense digital se corresponde con un conjunto de técnicas destinadas a extraer información valiosa de discos para su posterior análisis. Dicho proceso de extracción, en caso de tratarse de la memoria principal, se denomina volcar la memoria. Volcar la memoria consiste en copiar el contenido de la memoria principal en un archivo, el cual puede ser analizado posteriormente para obtener información del estado del computador en el momento del volcado (es necesario realizar un volcado debido a la naturaleza volátil de la memoria principal como se ha visto anteriormente). Una vez obtenido el fichero con la información de la memoria, es momento de analizarlo. Inicialmente durante los primeros años del análisis forense, los informáticos extraían cadenas de texto para luego buscar direcciones IP o URLs que podían dar un gran contexto sobre los problemas que hayan surgidos o, en caso de ataque, acceder al foco de este. Pero debido a la complejidad que podría llegar a tener la estructura de los ficheros, era difícil acceder a esta valiosa información y el proceso a seguir era muy intrincado. Además, no se lograba obtener otras estructuras de la memoria que pudiesen ser también valiosas y en general no se aprovechaba de la mejor manera la información. ¿Cómo se podría solucionar esta problemática?

Para poder aprovechar la información de los volcados de memoria, así como mejorar/acelerar el proceso de adquisición de la información se ha desarrollado diferentes herramientas de análisis de volcado de memorias a lo largo de los años. Dichas herramientas cumplen la función básica de interpretar las estructuras internas de memoria para poder acceder a la información valiosa que estas almacenan.

Actualmente no sólo existen herramientas de análisis de memoria, si no también existen herramientas específicas para el análisis de discos duros, dispositivos móviles o correos electrónicos, al servicio de la informática forense. Se trata de herramientas especializadas en una capa concreta de la infraestructura en la que se ha podido realizar un ataque. En este TFM, se tratará de estudiar diferentes herramientas relacionadas con el análisis de vaciados de memoria de Linux Kernel (infraestructura de memoria). En el siguiente apartado se estudiarán varias de ellas, estableciendo características, funcionalidades, ejemplos prácticos en caso de que se puedan, etc.

Estudio de herramientas de análisis de volcados de memoria kernel

En este apartado se hará un estudio de las principales herramientas de análisis de volcados de memoria, estableciendo la historia, características, ejemplos prácticos en caso de que sean posibles, etc. Primeramente, antes de proceder con el estudio de las herramientas se tratará brevemente sobre herramientas para generar volcados de memoria presentes en el mercado. Dichas herramientas serán necesarias para posteriormente generar el volcado de memoria del sistema a analizar. Esto nos permitirá conocer de primera mano el proceso de adquisición y preservación de las evidencias de la memoria RAM y utilizaremos posteriormente estos volcados de memoria para realizar las pruebas de las herramientas de análisis.

Para realizar los ejemplos de las herramientas y el análisis posterior del volcado de memoria se utilizará un sistema virtualizado Linux, específicamente una distribución Kali, utilizado para Ethical Hacking¹¹ y un Windows 10 x64. La máquina virtual dispone de la siguiente configuración:

- Sistema operativo: Kali GNU/Linux Rolling
- Kernel: Linux 5.2.0-kali2-amd64 (arquitectura x86-64)

Herramientas para generar volcados de memoria.

Como se comentó anteriormente, antes de empezar con las herramientas de análisis necesitamos disponer de algunos volcados de memoria. Para ello, en este apartado se nombrarán algunas herramientas que nos solucionan dicho problema, estableciendo las principales características de cada herramienta, como descargarla y como generaríamos el volcado con cada una de ellas.

¹¹ Ethical Hacking, <https://www.eccouncil.org/ethical-hacking/>

FTK Imager

FTK Imager de AccessData es una herramienta para realizar réplicas y visualización previa de datos, la cual permite una evaluación rápida de evidencias electrónicas para posteriormente garantizar un análisis posterior con una herramienta forense como AccessData Forensic Toolkit. FTK Imager también puede crear copias perfectas (imágenes forenses) de datos de ordenadores sin realizar cambios en la evidencia original [18]. Como características a destacar:

- Aplicación gratuita.
- Existe una versión portable disponible.
- Dispone de una interfaz gráfica que permite realizar los volcados de memoria (y su visualización) mediante “*point and click*”.

En el anexo “Instalación de FTK Imager y ejemplo de uso.” se encuentra una guía de instalación junto a un ejemplo de uso, donde se muestra como generar el volcado de memoria de un sistema Windows.

LiME (Linux Memory Extractor)

LiME es una herramienta desarrollada por [504ensics Labs](#), de código abierto, que permite la adquisición de la memoria volátil de sistemas Linux y dispositivos basados en Linux, como Android, y que trabaja a nivel de kernel [19]. La herramienta permite la adquisición de memoria en el sistema de archivos del dispositivo o través de la red. LiME es único en el sentido de que es la primera herramienta que permite capturas de memoria completa de dispositivos Android. También minimiza su interacción entre el usuario y el espacio de los procesos del kernel durante la adquisición, lo que le permite producir capturas de memoria sin modificar el estado del dispositivo.

Existe una documentación oficial con las opciones de ejecución aportada por la comunidad de manera abierta y gratuita en el [directorio oficial](#). Todos los pasos de la instalación se explican detalladamente en el anexo “Instalación de LiME”, así como un ejemplo de uso práctico.

Volatilitux

Volatilitux es básicamente un framework basado en Python, equivalente a Volatility para sistemas Linux. Volatilitux admite las siguientes arquitecturas para volcados de memoria: ARM, x86 y x86 con PAE¹² habilitado. Además, detecta de manera automática o manual los *offsets* de la estructura del núcleo kernel y puede usarse como un módulo de Python para automatizar tareas e integrarse con otros proyectos. En el anexo “Instalación de Volatilitux.” se encuentra los pasos de instalación de la herramienta.

¹² PAE, https://es.wikipedia.org/wiki/Extensi%C3%B3n_de_direcci%C3%B3n_f%C3%ADstica

Herramientas adicionales

A continuación, se hará una mención breve a algunas herramientas para tener en cuenta:

- [Memfetch](https://github.com/citypw/lcamtuf-memfetch). Al igual que Volatilitux, es una herramienta simple para volcar toda la memoria de un proceso en ejecución, ya sea inmediatamente o cuando se descubre un error en ella. Para descargar dicha utilidad se puede acceder a su directorio GitHub: <https://github.com/citypw/lcamtuf-memfetch>
- [Crash utility from Red Hat, Inc.](#): es una herramienta independiente para investigar tanto los sistemas en funcionamiento como los volcados de memoria del kernel hechos con los paquetes de Red Hat netdump, diskdump o kdump. También se puede utilizar para el análisis forense de memoria.
- [Forensic Analysis Toolkit \(FATKit\)](#): un nuevo framework multiplataforma y modular diseñado para facilitar la extracción, análisis, agregación y visualización de datos forenses en varios niveles de abstracción y complejidad de datos.
- [Memdump](#). Este programa vuelca la memoria del sistema al *stream* de salida estándar, evitando los agujeros en los mapas de memoria. Por defecto, el programa vuelca el contenido de la memoria física (*/dev/mem*). Este software se distribuye bajo la Licencia Pública de IBM (IPL).
- [Dumplt](#). Esta herramienta permite capturar la memoria RAM (en entornos Windows de 32 y 64 bits) de una forma sencilla. Con tan solo un “*doble clic*” permite realizar la captura. Esta herramienta es muy similar a FTK Imager en cuanto a sencillez y complejidad.

Sin duda si estamos trabajando en entornos Windows, FTK Imager debido a su sencillez se convierte en la herramienta ideal para generar volcados de memoria. En caso de que trabajemos en entornos Linux, una de las herramientas más usadas, si no la que más, es LiME debido a la sencillez de uso y que se trata de una herramienta de código abierto (además es ideal para entornos móviles que corran Android). Una vez analizadas algunas de las herramientas existentes para generar volcados de memoria ya estamos listo para pasar a estudiar las herramientas de tratamiento y análisis de volcados de memoria.

Volatility

En 2007, la primera versión de Volatility se lanzó públicamente en Black Hat DC. El software se basó en años de investigación académica publicada sobre análisis avanzado de memoria y análisis forense. Hasta ese momento, las investigaciones digitales se habían centrado principalmente en encontrar “*contrabando*” dentro de las imágenes del disco duro. Volatility introdujo el poder analizar el estado en tiempo de ejecución de un sistema utilizando los datos encontrados en el almacenamiento volátil (RAM). También proporcionó una plataforma multiplataforma, modular y extensible para alentar a los desarrolladores la inclusión de nuevos módulos para mejorar el framework. Otro objetivo importante del proyecto era fomentar la colaboración, la innovación y la accesibilidad al conocimiento que había sido común dentro de las comunidades de *pentesting*.

Desde entonces, el análisis de memoria se ha convertido en uno de los temas más importantes para el futuro de las investigaciones digitales y Volatility se ha convertido en la plataforma forense de análisis de memoria más utilizada en el mundo. El proyecto cuenta con el apoyo de una de las comunidades más grandes y activas de la industria forense. Como resultado, la investigación construida sobre Volatility ha aparecido en las principales conferencias académicas y Volatility se ha utilizado en algunas de las investigaciones más críticas de la última década. Se ha convertido en una herramienta indispensable de investigación digital en la que confían investigadores policiales, militares, académicos y comerciales de todo el mundo [16].

El desarrollo de Volatility ahora está respaldado por *The Volatility Foundation*, una organización independiente sin fines de lucro. La fundación se estableció para promover el uso del análisis de memoria dentro de la comunidad forense, para defender la propiedad intelectual del proyecto (marcas registradas, licencias, etc.) y, finalmente, para ayudar a avanzar en la investigación innovadora de análisis de memoria. En este sentido, la fundación también se formó para ayudar a proteger los derechos de los desarrolladores que sacrifican su tiempo y recursos para hacer que la plataforma forense de memoria más avanzada del mundo sea gratuita y *open source*.

Características

Entre las principales características disponemos de lo siguiente [17]:

- Open source y está escrito en Python, por lo tanto, es compatible en cualquier sistema sobre el cual se disponga de Python, por ejemplo, Microsoft Windows, Mac OS X y Linux.
- Dispone además de una API extensible y programable, donde además de los propios módulos que ofrece el propio Volatility es posible la creación de módulos personalizados con el fin de llevar a cabo análisis de una forma automática, la creación de una interfaz web o GUI personalizada o simplemente explorar la memoria del kernel de una forma automatizada. Los analistas pueden agregar nuevos espacios de direcciones, complementos, estructuras de datos y superposiciones para soldar el marco de trabajo a las necesidades de un momento dado. Es posible explorar la [documentación propia de dicho módulo desde su página oficial](#) para conocer más a fondo sus componentes internos.
- Volatility proporciona capacidades que por defecto el propio depurador del kernel de Windows no permite, como podría ser hacer un *carving*¹³ de un histórico de comandos, buffer de entrada/salida de la consola, objetos de usuario y estructuras de datos relacionados con la red.
- Ofrece también una cobertura completa de formatos de archivo donde ofrece la posibilidad de analizar volcados sin procesar, por caída, archivos de hibernación, estados guardados de sistemas virtualizados como VMware o volcados del núcleo de VirtualBox, LiME (Linux Memory Extractor), etc.
- Dispone además de una serie de algoritmos rápidos y eficientes que le permiten analizar los volcados de memoria RAM de sistemas de gran volumen sin un consumo de recursos excesivo y en muy poco tiempo.

¹³ "File carving", https://en.wikipedia.org/wiki/File_carving

Todos los pasos de la instalación se explican detalladamente en el anexo “Instalación de Volatility”. Además, en el anexo “Ejemplos de uso de Volatility” se explican ejemplos básicos de uso de algunas de las funcionalidades que el framework ofrece para comprobar que el framework está correctamente instalado en la máquina de análisis forense. Posteriormente, durante la fase del análisis del volcado de memoria, se ampliará el conocimiento respecto a la cantidad de funcionalidades disponibles con ejemplos prácticos.

Redline

Redline es la principal herramienta gratuita de seguridad de *endpoints*¹⁴ de FireEye. Brinda herramientas de investigación a los usuarios para encontrar signos de actividad maliciosa a través de la memoria, el análisis de archivos y el desarrollo de un perfil de evaluación de amenazas.

Con Redline, puedes:

- Auditar y recopilar minuciosamente todos los procesos y controladores en ejecución desde la memoria, metadatos del sistema de archivos, datos de registro, registros de eventos, información de red, servicios, tareas e historial web.
- Analizar y ver los datos de auditoría importados, incluida la capacidad de filtrar resultados en un período de tiempo determinado.
- Agilizar el análisis de memoria con un flujo de trabajo probado para analizar malware en función de la prioridad relativa.
- Realizar análisis de Indicadores de Compromiso (IOC, *Indicators of Compromise*). Suministrado con un conjunto de IOC, Redline se configura automáticamente para recopilar los datos para realizar el análisis de IOC y una revisión de resultados.

Como características importantes a resaltar podemos indicar que es gratuito, disponible para entornos Windows, ofrece una interfaz gráfica que facilita la comprensión y es muy completo y configurable por lo que nos aporta una gran cantidad de información y de manera clara y ordenada. En el anexo “Redline de FireEye, instalación y ejemplo de uso.” Se muestra paso por paso las instrucciones necesarias para instalar el software en el equipo junto como un ejemplo de uso práctico. Además, en el anexo “Resultados del análisis de prueba utilizando Redline de FireEye.” se ofrece un directorio con el resultado completo del análisis utilizando Redline.

AccessData Forensic Toolkit (FTK)

Desarrollado por Access Data, FTK es una de las suites de software más admiradas disponibles para los profesionales forenses. FTK es una solución de investigación digital construida para proporcionar velocidad, estabilidad y facilidad de uso. FTK recopila datos de cualquier dispositivo o sistema digital que produzca, transmita o almacene datos; y realiza el análisis forense de los mismos. FTK es conocido por su interfaz intuitiva, su análisis de correo

¹⁴ Endpoints, <https://admware.es/que-es-un-endpoint/>

electrónico, las vistas de datos personalizables, su velocidad de procesamiento y su estabilidad. FTK ofrece:

- Base de datos de casos única y compartida: toda la evidencia digital se almacena en una base de datos de casos, lo que brinda a los equipos acceso a la evidencia más reciente del caso. Reduce el tiempo, el costo y la complejidad que tendría el crear múltiples conjuntos de datos.
- Velocidad y entorno confiable: dado que la evidencia se procesa por adelantado, no hay que esperar a que se ejecuten las búsquedas durante la fase de análisis. Proporciona procesamiento rápido, preciso y consistente mediante procesamiento distribuido.
- Soporte multiproceso-multinúcleo: utiliza el 100% de sus recursos de hardware y es más confiable en caso de problemas técnicos de hardware o software.
- Visualización: construya líneas de tiempo e ilustre las relaciones entre los elementos de interés en un caso. Con la visualización de archivos, email y redes sociales se pueden ver datos en múltiples formatos, incluidos cronogramas, gráficos de clúster y circulares, geolocalizaciones, etc. para determinar las relaciones y encontrar elementos clave.

FTK dispone de documentación oficial, videotutoriales y cursos para la comunidad, pero a diferencia de las opciones vistas anteriormente (Volatility y Redline) y de FTK Imager (software visto en el apartado “Herramientas para generar volcados de memoria.”, perteneciente a la misma compañía), no estamos frente a un programa gratuito, sino que es necesario comprar una licencia para disfrutar de sus funcionalidades sin límites. Además, es un software dedicado a dispositivos Windows. Como estamos frente a un software de pago, desgraciadamente no se podrá ofrecer ejemplos de uso ni el proceso de instalación debido al alto coste que conlleva adquirir la licencia de este tipo de software profesional y que, por ende, encarecería de manera drástica el desarrollo del TFM.

Menciones adicionales

Sin duda las herramientas antes vistas son las más utilizadas en el mercado debido a los completas que son y a la gran cantidad de información que se puede obtener de los volcados de memoria. Sin embargo, existen herramientas de menor complejidad que bien es cierto que no ofrecen tantas alternativas a la hora de obtener información de las imágenes pero que pueden dar solución a problemas específicos que no necesitan demasiados recursos:

- [The Sleuth Kit](#) es un conjunto de herramientas open source para el análisis de imágenes de discos. Inicialmente desarrollada para plataformas UNIX, esta suite actualmente se encuentra disponible también para OS X y Windows. Además, TSK cuenta con una interfaz gráfica conocida como Autopsy que agrupa todas sus herramientas y plugins.
- [Belkasoft Evidence Center](#) (BEC) es una solución forense todo en uno, que ayuda en la extracción y análisis de los dispositivos móviles, ordenadores, RAM, nubes y sistemas remotos en una sola herramienta. Es una herramienta de pago, pero dado su precio asequible (menos completo y caro que FTK), es una de las mejores opciones entre los productos disponibles en el mercado.

Comparación entre herramientas

En vista de los resultados obtenidos en el estudio de las herramientas del enunciado anterior, en esta sección se elegirá cual es la herramienta que se usará para el análisis de la muestra basándonos en las características de las herramientas analizadas previamente, haciendo hincapié en a qué tipo de sistema operativo van dirigidos, el coste de esta y la cantidad de funcionalidades que nos aporta.

Herramienta	Sistema operativo	Coste	Soportes	Funcionalidades
Volatility	Linux, Windows, MAC OS	Open source (gratuito)	Mayor documentación, videotutoriales, foros, etc.	Alta (diversidad de plugins) + posibilidad de UI
Redline	Windows	Gratuito	Documentación, videotutoriales, foros, etc.	Alta (dispone de entorno gráfico, filtros y plantillas)
FTK	Windows	Licencia de pago	Documentación, tutoriales, etc.	Alta (dispone de entorno gráfico)
TSK - Autopsy	Linux, OS X y Windows	Open source	Documentación, tutoriales, foros, etc.	Media-alta (entorno gráfico, plugins)
BEC	Windows	Licencia de pago (inferior a FTK)	Documentación, tutoriales, foros, etc.	Media-alta (entorno gráfico)

En vista a la tabla resumen, la mayoría de las herramientas disponen de soportes para ayudar a los usuarios al análisis, siendo Volatility la herramienta más utilizada y la que dispone de una mayor comunidad de usuarios. Asimismo, todas las herramientas disponen de bastantes funcionalidades, las mayorías de ellas disponen de entornos gráficos (menos Volatility, aunque es posible adquirirlo a través de un plugin) y filtros/plantillas para los análisis. Debido a que el volcado de memoria que se va a tratar en este TFM es de tipo Linux Kernel, considero importante que la herramienta elegida esté presente en este tipo de sistemas. Por otra parte, para que el coste de realizar este TFM sea el menor posible, se descartará desde un inicio todas las herramientas que necesiten adquirir una licencia para su uso.

Teniendo esto en cuenta nos quedan dos principales vertientes, Volatility o TSK-Autopsy. Aunque el proyecto TSK dispone del plugin Autopsy como entorno gráfico, debido a la gran cantidad de plugins, soportes en internet, comunidad de usuarios (es la herramienta más usada) y, en general, lo completa que es la herramienta, considero que la mejor opción para realizar el análisis es Volatility.

Más allá de qué herramienta nos conviene utilizar para este caso concreto (para este TFM), es un hecho que el mercado nos ofrece una gran variedad de herramientas que satisface las necesidades que tengamos entre manos. Los aspectos a tener en cuenta para la elección de las herramientas pueden ser las anteriores nombradas, como el sistema operativo que soporta la herramienta, el coste, los soportes y documentación para los usuarios, complejidad o

funcionalidades que ofrece la herramienta, etc. así como aspectos más técnicos como pueden ser qué interfaces de usuario soportan (UI¹⁵), qué tipo de imágenes soportan (*RAW*, *Crash dump*, *hibernation*, etc.), la arquitectura de CPU que soporta la herramienta (INTEL x86, AMD x64...) o limitaciones individuales (multilenguaje, limitaciones de tamaños de memoria, plugins, etc.). Teniendo en cuenta estos aspectos, existen herramientas en el mercado que pueden resultar útiles. Como hay muchos factores a la hora de elegir la mejor herramienta para según qué casos, de manera general haré una recomendación (herramienta más completa probada) basada en a qué tipo de sistema operativo va dirigido y, que sean de bajo coste (preferiblemente gratuito):

- Herramienta más completa (y para trabajos en sistemas Linux/MAC OS): Volatility. Debido a la cantidad de sistemas operativos que soporta, su característica open source, cantidad de formatos de volcados que soporta, cantidad de plugins disponibles y la comunidad que hay detrás involucrada con la herramienta recomiendo este framework para cualquier caso general.
- Herramienta para sistemas operativos Windows: Redline de Mandiant. Soporta la mayoría de las arquitecturas de Windows, dispone de un GUI bastante intuitivo, dispone de perfiles de investigación, soportan la mayoría de los volcados de memoria, etc.

Preparando el entorno de trabajo.

Una vez terminada la base teórica necesaria para el entendimiento del experimento práctico ya podemos adentrarnos en la sección práctica del TFM, en donde se procederá a la utilización de la herramienta elegida previamente, Volatility, sobre un volcado de memoria, con el objetivo de adquirir información útil que permita observar las posibles vulnerabilidades que este tiene, configuraciones, procesos, etc.

El punto de partida de todo análisis forense basado en análisis de memoria RAM consiste, como ya hemos comentado, en adquirir y preservar la información que estos contienen, de tal manera que en lugar de trabajar en caliente sobre la memoria RAM, trabajaremos sobre los denominados volcados de memoria o imágenes.

Como también he comentado, la idea es utilizar un volcado estándar de un sistema distribuido de tipo IoT. Con estándar me refiero a un volcado no comprometido de ninguna manera con software malicioso (al menos a propósito) ni configurado de una determinada manera para generar vulnerabilidades. La idea es intentar que sea un proceso lo más cercano posible a la realidad, es decir, simular un escenario real. Uno de los requisitos del internet de las cosas (IoT) es que los dispositivos deben ser pequeños y que los procesadores tienen que cambiar respecto a los que conocemos normalmente. No nos valen los procesadores, digamos, "clásicos" de ordenador, tiene que ser algo mucho más pequeño y de consumo menor. No importa que sean sencillos o poco potentes, lo que prima ante todo son esos dos puntos [53].

Los procesadores de smartphones y su evolución de los últimos años, con el formato SoC¹⁶ ya establecido, han ayudado mucho. Las soluciones de ARM¹⁷ cumplen con las expectativas: son

¹⁵ User interface (UI), https://es.wikipedia.org/wiki/Interfaz_de_usuario

¹⁶ SoC, https://es.wikipedia.org/wiki/System_on_a_chip

¹⁷ ARM, https://es.wikipedia.org/wiki/Arquitectura_ARM

pequeños y, aunque también poco potentes en comparación con otros chips del mercado, cubren con los requisitos planteados.

Un procesador ARM es un procesador que está basado en la arquitectura RISC (*Reduced Instruction Set Computer*). Esta arquitectura ha sido desarrollada por la empresa ARM. Los procesadores ARM están optimizados para realizar instrucciones mucho más sencillas, a muy bajo nivel, el procesador sólo tiene que seguir los pasos para conseguir que el código funcione. Gracias a esto, los procesadores ARM consumen mucho menos energía, lo que los convierte en los componentes idóneos para los SoC que requieren de un menor consumo.

Debido a las ventajas que los procesadores ARM ofrecen, son perfectos para sistemas IoT. Inicialmente mi idea era simular una Raspberry Pi con el sistema operativo Raspbian Lite para ARMv7 con el kernel en su versión 4.17. Para ello se haría uso del emulador de procesadores QEMU (u obtener directamente un volcado de una Raspberry PI real).

QEMU es un emulador de procesadores basado en la traducción dinámica de binarios (conversión del código binario de la arquitectura fuente en código entendible por la arquitectura huésped). QEMU también tiene capacidades de virtualización dentro de un sistema operativo, ya sea GNU/Linux, Windows, o cualquiera de los sistemas operativos admitidos; de hecho, es la forma más común de uso. Esta máquina virtual puede ejecutarse en cualquier tipo de Microprocesador o arquitectura (x86, x86-64, PowerPC, MIPS, SPARC, ARM, etc.). Está licenciado en parte con la LGPL y la GPL de GNU. El objetivo principal es emular un sistema operativo dentro de otro sin tener que particionar el disco duro, empleando para su ubicación cualquier directorio dentro de éste. El programa no dispone de GUI, pero existe otro programa llamado QEMU manager que puede hacer de interfaz gráfica si se utiliza QEMU desde Windows. También existe una versión para GNU/Linux llamada *qemu-launcher* [33].

El problema encontrado en simular una arquitectura de procesadores de ARM con QEMU viene en relación con el soporte que ofrece Volatility frente a este tipo de arquitecturas. Actualmente el framework soporta muy pocas funcionalidades/plugins con relación a este tipo de arquitecturas de procesadores (mayormente relacionado con Android) y el interés de la comunidad con relación a mejorar este aspecto actualmente no es muy alto, ya que son dispositivos que representan una minoría (no son *targets* principales de Volatility, al menos de momento). Bien es cierto que Volatility ofrece apoyo y perfiles a sistemas como Android, pero debido a como Raspbian (y otros sistemas operativos con soporte ARM) tienen definido el sistema estructural del kernel, hardware y software (ficheros compilados en C, por ejemplo, *overlays* y *blobs*¹⁸), no es posible actualmente filtrar la información de los volcados de memoria de este tipo de sistemas. Volatility utiliza la información almacenada en el fichero *System.map* de los sistemas Linux para generar los perfiles y este no se encuentra en Raspbian.

También cabe destacar que no existe perfiles de estructuras de kernel relacionada con ARM ni documentación para generar estas por cuenta propia en Volatility (debido a que día de hoy todavía no hay soporte). En cualquier caso, en el anexo denominado “Simulando arquitectura ARM con el sistema operativo Raspbian y QEMU” se establece los pasos necesarios para simular un sistema ARM con Raspbian en QEMU.

Debido a la problemática de ARM y Volatility he tenido que buscar alternativas. En la página oficial de Raspbian existe una versión de Raspbian de escritorio, que se puede emular en

¹⁸ Overlays/blobs, <https://www.raspberrypi.org/documentation/configuration/device-tree.md>

VirtualBox. Esta versión a diferencia de la versión de Raspberry Pi no cuenta con soporte para ARM, pero si cuenta con PAE y, en definitiva, la imagen que se generará será equivalente a todos los efectos, excepto en la definición del hardware, por lo que para nuestro objetivo no supondrá ningún inconveniente ya que este TFM va enfocado en el análisis forense de memoria RAM guiado a adquirir información para identificar debilidades con sus respectivas medidas de mitigación (cabe destacar que ambas versiones de Raspbian, con o sin ARM, están basadas en Debian). Por lo tanto, emularemos un sistema Raspbian 4.19 en un entorno Windows para la adquisición del volcado y la creación del perfil de Volatility, utilizando para ello el software de virtualización VirtualBox.

Adquisición del volcado de memoria.

Una vez tenemos el sistema Raspbian simulado [Simulando Raspbian Desktop OS sobre un entorno Windows 10.] estamos preparados para realizar el volcado de la máquina virtual. Tal y como se explicó anteriormente en el apartado “Herramientas para generar volcados de memoria.”, existe en el mercado una amplia variedad de herramientas que nos permiten generar volcados de memoria en sistemas Linux (recordad que Raspbian está basado en Debian). Para este caso particular utilizaré LiME para generar el volcado. En caso de no disponer de él, en el anexo “Instalación de LiME” se encuentra paso a paso el proceso de instalación de la herramienta.

Una vez instalado, procedemos a generar el volcado de memoria. En el anexo “Generando el volcado de memoria de Raspbian con LiME.” se encuentra el proceso completo. Ahora disponemos de un fichero llamado *raspbian.mem* de aproximadamente 1 GB de memoria con el volcado completo del sistema:

```
pi@osboxes:~/Desktop $ stat raspbian.mem
  File: raspbian.mem
  Size: 1073278016      Blocks: 2096256      IO Block: 4096   regular file
Device: 804h/2052d    Inode: 5505147       Links: 1
Access: (0444/-r--r--r--)  Uid: (   0/   root)   Gid: (   0/   root)
Access: 2020-04-13 08:37:47.530127000 -0900
Modify: 2020-04-12 01:53:13.065329277 -0900
Change: 2020-04-12 01:53:13.065329277 -0900
 Birth: -
```

Ilustración 5. Propiedades del volcado de memoria.

Normalmente si nos encontramos en un caso real de análisis forense sobre memoria RAM y, en general en un encargo profesional en la que interviene el análisis de soportes de memoria, no se puede modificar el estado de la máquina que se va a auditar (se prohíbe la instalación de programas en la máquina auditada). Esto es debido a que la máquina en sí mismo es una prueba que no debe de ser manipulada en ningún momento. En casos en los que intervienen, por ejemplo, discos duros, se debe de hacer lo que se denomina clonado de discos¹⁹, para generar una copia igual y trabajar sobre ella. En casos en los que intervienen análisis de memoria RAM,

¹⁹ Clonado de discos, <https://indalics.com/peritaje-informatico/clonado-forense-de-discos-duros>

el volcado de este se suele hacer de manera remota²⁰, sin intervenir directamente en la máquina. Como estamos en un entorno de pruebas, realizaremos el volcado de manera local, es decir, instalando los módulos necesarios (LiME) directamente en el sistema a auditar, pero es necesario indicar la importancia de realizar los volcados de manera remota (aplicable a la adquisición de los perfiles de Volatility y, en general, a cualquier acción que se necesite hacer sobre la máquina a auditar/analizar).

El fichero generado lo moveremos del sistema auditado a la máquina Kali donde realizaremos las pruebas de análisis, pero antes debemos de crear el perfil de Volatility en la máquina Raspbian para poder extraer la información del volcado de memoria.

Creación del perfil de Volatility e instalación.

Como he comentado anteriormente en varias ocasiones, necesitamos el perfil de Raspbian para filtrar y sacar la información del volcado de memoria. Con perfil o *profiles* se refiere a esencialmente un archivo .zip con información sobre las estructuras de datos del núcleo/kernel y los símbolos de depuración de este. Esto es lo que utiliza Volatility para localizar la información crítica y cómo interpretarla una vez encontrada. Por lo tanto, se deberá obtener un perfil que coincida con la versión del núcleo y arquitectura/sistema operativo del sistema que se desea analizar, por lo que debemos de crearlo sobre la máquina a la cual pertenece el volcado de memoria.

Y digo crear ya que inicialmente Volatility tan sólo dispone de perfiles sobre Windows (XP, Windows 7, 8, 10, etc.) accesibles desde la herramienta Volatility por defecto. Sin embargo, Volatility ofrece una serie de perfiles Linux preconstruidos sobre sistemas Linux y MAC de diferentes arquitecturas, accesibles desde [el repositorio oficial del framework](#). Estos perfiles (CentOS, Debian, Red Hat, etc.) pueden ser “instalados” en Volatility para analizar volcados de memoria de dichos sistemas. Sin embargo, entre estos perfiles preconstruidos no aparece ninguno para Raspbian 4.19. Inicialmente podemos pensar que, al estar basado en Debian, tal vez un perfil de Debian pueda cumplir la función y pueda filtrar y sacar la información del volcado sin necesidad de generar uno. Sin embargo, tras realizar pruebas (se instaló el perfil de Debian 8 x64), quedó confirmado que los perfiles Debian no servían para sacar la información del volcado tal y como queremos (error “*No suitable address space mapping found*”²¹).

Ante esta problemática, es decir, a la diversidad de sistemas operativos basados en Linux, arquitecturas y versiones, Volatility permite crear perfiles propios (en caso de que no existan ya) de sistemas Linux tal y como explican en el [repositorio oficial](#). De esta manera permiten crear perfiles para una mayor diversidad de sistemas operativos sin dependencia de los preconstruidos. Desde el propio repositorio nos avisan de la importancia de crear correctamente el perfil:

²⁰ Adquisición remota con LiME, <https://fwhibbit.es/volcado-de-memoria-ram-en-linux-lime>

²¹ *No suitable address space mapping found*, <https://github.com/volatilityfoundation/volatility/wiki/FAQ#no-address-space-mapping-no-valid-dtb-found>

“Con diferencia, el error más común con respecto al análisis forense de la memoria de Linux es crear un perfil para un sistema que no sea la máquina que desea analizar. Por ejemplo, no se puede crear un perfil para un sistema Debian 2.6.32 para analizar un volcado de memoria de Mandrake 2.6.32. Del mismo modo, no puede crear un perfil para el sistema SuSE 2.5.35 para analizar un volcado de memoria desde SuSE 2.6.42. Debe asegurarse de que el perfil que cree coincida con el sistema de destino en 1) distribución de Linux 2) versión exacta del kernel 3) arquitectura de CPU (32 bits, 64 bits, etc.)”.

Por lo tanto, crearemos un perfil para Raspbian 4.19, el cual es la versión del sistema a auditar. Para poder crear dichos perfiles es necesario disponer de Volatility instalado en el sistema. En el anexo “Instalación de Volatility” se explica detalladamente el proceso de instalación de la herramienta. Cabe recordar que, al estar en un entorno de pruebas, se instalará Volatility en el sistema auditado, sin embargo, no se podría instalar ningún software en la máquina a analizar en un entorno real, la creación del perfil en este caso se debería de hacer de manera remota o simulando un entorno similar al auditado para acceder a la estructura del kernel.

Para crear el *profile*, Volatility nos pide esencialmente 2 ficheros:

- *module.dwarf*: el fichero *module.dwarf* contiene información relacionada con la versión del kernel, arquitectura del sistema, mensajes de depuración, etc.
- *System.map*²²: es un fichero situado normalmente bajo el directorio */boot* en sistemas Linux que contiene esencialmente una tabla que relaciona los símbolos y en general la estructura del kernel con su correspondiente dirección de memoria.

El fichero *System.map* es el principal problema por el cual no es posible crear perfiles para procesadores ARM, ya que estos sistemas operativos no disponen de dicho fichero, ya que la estructura en estos sistemas está definida por *blobs files* y *overlays*. En el anexo “Generando el perfil/profile de Volatility del entorno Raspbian 4.19 e instalación en Volatility.”, se muestra paso por paso el proceso de generación del perfil de Volatility y su correspondiente instalación.

Considero importante acentuar que el perfil creado se ha instalado en el sistema que se utilizará para realizar las pruebas y el análisis en general [Sistema operativo utilizado], nunca en la máquina Raspbian, ya que esta es tan solo la “máquina auditada”. Por lo tanto, debemos de enviar tanto el volcado como el perfil a la distribución Kali Linux, que es donde se llevará a cabo el análisis, que corresponde a la siguiente sección.

Análisis del volcado de memoria

Una vez tenemos el entorno de análisis preparado es cuestión de empezar a trabajar con el volcado de memoria. En esta sección, el objetivo es utilizar las diferentes herramientas que nos ofrece Volatility para adquirir la información que nos aporte valor para, por ejemplo, un caso jurídico. En este caso en concreto, el objetivo es realizar un análisis completo, para comprobar la configuración y el contenido del volcado de memoria, con el fin de encontrar posibles

²² System.map: <https://en.wikipedia.org/wiki/System.map>

vulnerabilidades, fallos en la configuración y, en general, cualquier aspecto mejorable que pueda comprometer la integridad y la seguridad del sistema, en caso de que haya.

Debido a la gran variedad de herramientas que ofrece Volatility, es necesario establecer un orden respecto a que aspecto se está auditando en todo momento. Para ello, seguiré el propio esquema que ofrece *Volatility Foundation* en el repositorio del framework, para seguir un orden con relación a que sección del volcado se está analizando. El orden que se seguirá es el siguiente [70]:

- 1) Selección del perfil de análisis.
- 2) Listar procesos del volcado.
- 3) Análisis individual de cada proceso.
- 4) Distribución de los procesos en memoria.
- 5) Memoria y objetos del kernel.
- 6) Información de la red.
- 7) Información del sistema.
- 8) Detección de rootkits.

Esta distribución tiene una explicación racional detrás. Normalmente en un entorno profesional, no se conoce previamente el sistema operativo del volcado a tratar. Suponemos el caso de que recibimos una petición por parte de una empresa, donde resulta que en una de sus máquinas se ha instalado un malware a través de un PDF corrupto descargado por uno de sus empleados. Dicha empresa a causa de la situación generó un volcado de memoria y nos pide que intentemos encontrar la causa de la infección. Ante esta situación de ejemplo, nosotros como no disponemos de conocimiento relacionado con el tipo de sistemas que corren en la empresa, el primer paso que se ha de hacer sería conocer qué sistema operativo dispone el volcado de memoria, para elegir el perfil correcto de análisis.

Por dicho motivo, el primer paso será seleccionar el perfil del análisis, esencial para los procesos posteriores. Seguidamente, el conseguir información relacionada con los procesos activos en el momento de la toma del volcado, suele ser un paso primordial en cualquier análisis forense sobre volcados de memoria. Esto es debido a que gran parte de los pasos posteriores necesitan información de los procesos, normalmente a través del PID. El siguiente paso será la adquisición de la información de cada proceso individualmente, analizando las diferentes cadenas de texto que estos contienen. Los pasos posteriores ya es cuestión del caso a tratar, es decir, si estamos frente a un caso de filtración de datos, el adquirir información de la red del sistema tendrá una mayor prioridad, sin embargo, si estamos frente a un caso de eliminación de archivos o disposición de archivos sensibles, adquirir la información del sistema tendrá prioridad sobre el resto.

En este caso, al tratarse de un proyecto académico de investigación, se realizará todas las fases del análisis, así que el orden no tiene tanta importancia en este caso, ya que realizaremos un análisis completo de todo el volcado. Por último, tal y como se comentó durante el “Estado del arte”, el análisis forense en volcados de memoria es uno de los métodos más efectivos, si no el que más, para la detección de rootkits en el sistema. Debido a esta importancia, Volatility ofrece herramientas dedicadas específicamente a buscar rootkits en la memoria, con lo que finalizaremos el análisis con dicha fase.

Seleccionando el perfil de análisis.

Tal y como se ha comentado anteriormente, Volatility dispone de una gran variedad de perfiles. Por defecto, desde la herramienta se puede acceder a varios perfiles Windows, desde Windows XP, hasta Windows 10, pasando por 2003, 2008 y Windows 7 y 8, todas en sus distintas versiones y para diferentes arquitecturas de procesadores (x32, x64 y x86). Además, el propio framework permite la instalación de perfiles preconstruidos de Linux y MAC OS, así como los creados propiamente por los usuarios, tal y como se ha explicado en el apartado “Generando el perfil/profile de Volatility del entorno Raspbian 4.19 e instalación en Volatility.”

Una vez disponemos un volcado de memoria, Volatility nos ofrece un plugin denominado *imageinfo* el cual nos devuelve información primordial relacionado con el volcado y el sistema al que pertenece (direcciones de memorias importantes, fecha de creación, perfiles, número de procesadores, etc.). En el apartado “Ejemplos de uso de Volatility” se explica cómo usar correctamente dicho plugin, así como otros ejemplos de uso. Entre la información que nos devuelve [Ilustración 14. Volatility imageinfo plugin] nos interesa especialmente la siguiente:

- *Suggested Profile*: devuelve la lista de los perfiles recomendados según la estructura.
- *KPCR (Kernel Processor Control Region*²³): devuelve la posición en memoria de la estructura de datos utilizada por el kernel para almacenar los datos específicos del procesador.
- *KDBG (kernel debugger block*²⁴). Simplemente busca firmas de encabezado KDBG vinculadas a los perfiles.

En especial el valor de *Suggested Profile* es de suma importancia, ya que nos devuelve el/los perfiles con los que podremos sacar la información del volcado de memoria. Por ejemplo, en la imagen [Ilustración 14. Volatility imageinfo plugin] nos devuelve que un perfil válido para dicha imagen es el *VistaSP1x86*. Volatility adicionalmente ofrece dos funciones adicionales para la selección del perfil correcto. Como se ha comentado, el plugin *imageinfo* nos devuelve normalmente más de un perfil recomendado, visto en el objeto “*Suggested Profile*”. Si queremos conocer a ciencia cierta qué perfil nos conviene en todo momento, podemos hacer uso de las siguientes funciones:

- *Kdgbscan*. A diferencia de *imageinfo*, que simplemente proporciona sugerencias de perfil, *kdgbscan* está diseñado para identificar el perfil correcto y la dirección KDBG correcta (si es que hay múltiples) mediante las denominadas firmas *KDBGHeader*.
- *Kpcrscan*. Conviene utilizar este comando para buscar posibles estructuras de KPCR.

Para el caso que nos ocupa, como ya he importado previamente por mi cuenta el perfil al framework, ya que tengo acceso a la máquina a auditar y sé de primera mano qué sistema operativo tiene la imagen tomada y qué perfil utilizar, visto en el apartado “Creación del perfil de Volatility e instalación.”, este paso no es necesario ya que el perfil que se utilizará se sabe con anterioridad, pero considero importante documentar la importancia de elegir un perfil correcto para según qué imagen se va a tratar (ya que la cantidad y la calidad de la información extraída está totalmente ligado al perfil escogido).

²³ KPCR, https://en.wikipedia.org/wiki/Processor_Control_Region

²⁴ KDBG, https://es.wikipedia.org/wiki/Depurador_del_kernel

Visualización de procesos en ejecución

La parte primordial en cualquier auditoría o análisis forense sobre volcados de memoria RAM, independientemente del software de análisis que se utilice, el volcado objetivo o la finalidad que se pretende conseguir es la visualización de todos los procesos en ejecución en el momento de la toma de la imagen. La mayoría de los plugins y funcionalidades de Volatility son dependientes de los procesos del volcado y de ahí la importancia de conocer todos los procesos que estaban en ejecución en el momento de la toma de la imagen, con su respectivo PID²⁵ (identificación de proceso), PPID (identificador del proceso padre que hace la llamada) y UID (identificador del usuario que lanza dicho proceso). Volatility nos ofrece varias herramientas para visualizar dicha información y de diferentes maneras, las más destacables son las siguientes [36]:

- *linux_pslist*: devuelve todos los procesos en ejecución junto con su PID, PPID, UID, etc.
- *linux_pstree*: similar al anterior, pero muestra la información en formato jerárquico (relación padre-hijo).
- *linux_psscan*: escanea la memoria en busca de procesos activos, ocultos y finalizados.
- *linux_psxview*. Este plugin es útil para descubrir procesos maliciosos, identificando procesos que utilizan listados alternativos. La lista puede ser referenciada con diferentes fuentes de información para identificar discrepancias (muy útil y eficiente para encontrar rootkits). Esto le permite al analista revisar los listados y determinar si hay una razón legítima por la cual el proceso no se encuentra ahí (ocultos).
- *linux_psaux*. Recopila procesos junto con la línea de comando completa utilizada para invocar a este y la hora de ejecución.

En el anexo “Visualización de procesos en ejecución” se muestra la ejecución de los plugins antes mencionados para la visualización del listado completo de los procesos en el volcado de memoria, junto con la salida completa de la ejecución de las herramientas. Si preferimos ver los diferentes procesos mediante los diferentes hilos de ejecución, Volatility nos ofrece el plugin denominado *linux_threads* [Visualización de los hilos de ejecución] el cual nos permite conocer los diferentes hilos de ejecución de los procesos de la memoria tomada.

Suponemos el caso nombrado anteriormente sobre la situación de una empresa que sufre una infección a través de un PDF malicioso. A través de las herramientas anteriores, podemos ver los procesos en el sistema que trabajan con ficheros PDF, por ejemplo, Adobe Acrobat Reader, navegadores, Sumatra, etc., para, a través de ahí, buscar la relación de procesos que intervienen en la lectura del PDF (procesos padres e hijos). Por ejemplo, se observa que existe un proceso denominado *adobereader.exe*, lanzado por un proceso denominado *firefox.exe*. De primera mano, ya nos da motivos de sospechar de ambos procesos, por lo que convendría anotar su PID para proceder a la extracción completa de ambos procesos. También, si al listar los procesos observamos un proceso cuyo nombre resulta extraño o no conocido (podemos buscar su nombre por internet) o simplemente resulta sospechoso a simple vista, conviene anotar sus respectivos PID ya que puede tratarse de malware, especialmente rootkits.

Si nos dirigimos al caso que tenemos entre manos, al ejecutar la herramienta *linux_pslist* sobre el volcado vemos que nos devuelve una lista con bastantes procesos que estaban activos en el momento de la toma [Visualización de procesos en ejecución]. Si hacemos hincapié en la

²⁵ PID, https://es.wikipedia.org/wiki/Identificador_de_proceso

información obtenida podemos identificar varios procesos que nos puede llamar la atención o que podemos considerar importantes. Algunos son:

1. *Systemd* (PID=1, PPID=0, UID=0). *systemd* es un conjunto de demonios/*daemons*²⁶ de administración del sistema, bibliotecas y herramientas diseñadas como una plataforma de administración y configuración central para interactuar con el núcleo del sistema operativo GNU/Linux [70]. *systemd* se puede utilizar como un proceso de inicio de Linux (el proceso *init* llamado por el kernel de Linux para inicializar el espacio de usuario durante el proceso de arranque de Linux y gestionar posteriormente todos los demás procesos). Por esa razón, dicho servicio tendrá el ID=1, no tendrá un proceso padre que inicie la llamada y su UID=0 ya que es un proceso de sistema que siempre es lanzado por el *superusuario/root*²⁷.
2. *kthreadd*(PID=2, PPID=0, UID=0). Es el demonio/*daemon* de los hilos del núcleo, es decir, el padre de todos los hilos del núcleo (todos los *kthreads* se bifurcan en este hilo) [70].
 - 2.1. *Ksoftirqd* (PID=9, PPID=2, UID=0), es un hilo para cada CPU o núcleo. Controla las interrupciones "suaves" (interrupciones servidas o retornadas por interrupciones fuertes). Como se puede observar, su proceso padre es el PID=2 es decir, *kthreadd*.
 - 2.2. *Kworker* (PID=5, PPID=2, UID=0), es un proceso que engloba varios subprocesos de trabajo del kernel, los cuales realizan la mayor parte del procesamiento real del kernel, especialmente en los casos en los que hay interrupciones (por ejemplo, interrupciones ACPI de la BIOS), temporizadores o *timers*, dispositivos de entrada y salida, etc.. Como se puede observar, su proceso padre es el PID=2, es decir, *kthreadd*.
3. *bash* (PID=1194, PPID=1188, UID=1000). Proceso que hace referencia al GNU *Bash*, comúnmente conocido como el *shell* de Linux. *Bash* es un procesador de comandos que generalmente se ejecuta en una ventana de texto donde el usuario escribe comandos que causan acciones en el sistema. A diferencia de los procesos anteriores, dicho proceso es generado por el usuario cuyo UID=1000.
4. *sudo* (PID=1847, PPID=1194, UID=0). Dicho proceso se corresponde al proceso que hace referencia al comando de Linux *sudo*. Dicho comando es un programa para sistemas operativos tipo Unix que permite a los usuarios ejecutar programas con los privilegios de seguridad de otro usuario, por defecto el usuario *root*. Como se puede observar, dicho proceso tiene como PPID=1194, que precisamente corresponde al proceso *bash*.
5. *insmod* (PID=1848, PPID=1847, UID=0). Dicho proceso corresponde a la ejecución del comando *insmod*. El comando *insmod* inserta un módulo en el kernel de Linux. Es una forma de agregar una funcionalidad al núcleo del sistema operativo Linux [70]. Dicho comando fue utilizado para generar el volcado de memoria en el sistema auditado en conjunto con el comando *sudo* y mediante el *bash* [Generando el volcado de memoria de Raspbian con LiME.]. El proceso padre que hace la llamada es el proceso con PID=1847, es decir, *sudo*. Este proceso es importante, porque permite cargar módulos en el kernel, por lo que puede dar pie a vulnerabilidades o permitir a atacantes ganar permisos *root* en el sistema²⁸.

En vista de los resultados obtenidos, podemos prácticamente asegurar que los procesos *insmod*, *sudo* y *bash*, son subprocesos de un proceso padre común. Para comprobar las sospechas podemos hacer uso de la herramienta *linux_pstree* [Visualización de procesos en ejecución] para

²⁶ Daemon, [https://es.wikipedia.org/wiki/Daemon_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Daemon_(inform%C3%A1tica))

²⁷ Root, <https://es.wikipedia.org/wiki/Root>

²⁸ Insmod vulnerable (obsoleto), <https://packetstormsecurity.com/files/12021/insmod.linux.txt.html>

ver el árbol de los procesos. Si observamos la ilustración “Ilustración 32. Relación de procesos padre-hijo del proceso systemd.”, podemos observar la siguiente información:

Name	Pid	Uid
systemd	1	
.lightdm	553	
..Xorg	580	
..lightdm	875	
...lxsession	965	1000
....ssh-agent	997	1000
....openbox	1020	1000
....lxpolkit	1022	1000
....lxpanel	1024	1000
.....lxterminal	1188	1000
.....bash	1194	1000
.....sudo	1847	
.....insmod	1848	

Observamos la correlación entre los procesos. Los procesos, *insmod*, *sudo* y *bash* pertenecen al proceso principal *systemd* cuyo PID=1. Se utilizaron en conjunto para generar el volcado de memoria que estamos tratando [Ilustración 27. Generando el volcado de memoria con LiME.]. Esta información es bastante importante y la base de cualquier análisis forense en volcados de memoria. Suponemos que el proceso *insmod* es un proceso malicioso (aunque realmente pertenezca a LiME). Gracias a las utilidades *linux_pslist* y *linux_pstree* somos capaces de saber que procesos intervienen en la ejecución del comando *insmod*. Como sabemos los PID de todos los procesos que intervienen, seremos capaces de extraer todos ellos para su análisis.

Una vez identificamos los procesos que sospechamos que pueden contener algún aspecto malicioso, Volatility nos ofrece alguna herramienta para confirmar las sospechas. Por ejemplo, el plugin llamado *linux_malfind* nos permite encontrar códigos/DLLs ocultos o inyectados en la memoria, en función de características, como la etiqueta VAD²⁹ y los permisos de página. En el apartado “Utilización del plugin malfind sobre el volcado de memoria” se puede observar un ejemplo de uso de cómo funciona dicho plugin sobre el volcado de memoria. Sin embargo, la forma más común de analizar los procesos es de manera manual. Para ello habrá que realizar un volcado de los procesos que se quieran analizar.

Analizando los procesos

Una vez identificados los procesos que sospechamos que puedan contener malware, el siguiente paso natural es realizar un volcado de dichos procesos. Para ello Volatility nos ofrece un plugin llamado *linux_procdump* el cual nos permite realizar un vaciado de los procesos, tal y como se muestra en el anexo “Realizando el vaciado de los procesos con el plugin procdump”.

Este paso es bastante importante, ya que en este apartado seremos capaces de confirmar, o no, las sospechas que tenemos sobre los procesos que creemos que son maliciosos. Cuando generamos un volcado de un proceso, el resultado será normalmente un fichero binario con bastantes cadenas de texto que tendremos que analizar. Por ejemplo, si comprobamos la

²⁹ VAD, <http://bsodtutorials.blogspot.com/2013/11/virtual-address-descriptors-vads-vad.html>

cantidad de líneas que tiene el volcado del proceso *sudo*, tal y como podemos ver en la “Ilustración 36. Cantidad de líneas del volcado del proceso sudo.”, nos devuelve un total de 608 líneas. Como es lógico, el comprobar línea por línea las cadenas de texto de cada proceso sería un proceso bastante complejo y lento. Por lo que la utilización de herramientas externas a Volatility son necesarias para el filtrado de la información, por ejemplo, la utilidad *grep*³⁰, el cual nos permite buscar mediante expresión regular cualquier segmento de texto que queramos.

Por lo tanto, una vez realizado el volcado el siguiente paso suele ser el análisis de las cadenas de texto dentro de dichos volcados con el fin de encontrar palabras clave que nos den indicios de malware o actividad sospechosa, como pueden ser conexiones a web externas, peticiones FTP o HTTP, JavaScript, palabras claves (hacking, malware, bancos, credenciales y contraseñas...), etc. Como los volcados de memoria son ficheros binarios, debemos de hacer uso de aplicaciones externas a Volatility que nos permitan leer las cadenas de texto. Una de las aplicaciones/herramientas más utilizadas en Linux es la denominada *strings*³¹. Por ejemplo, en el caso ejemplo del PDF infectado, buscaríamos ficheros PDFs, o enlaces HTTP a algunas webs (es posible utilizar herramientas como *foremost*³², el cual nos permite recuperar ficheros tanto de memorias físicas, discos y USB como en procesos). Posteriormente subiremos los ficheros PDFs recuperados a alguna base de datos de virus, como, por ejemplo, Virustotal³³, para buscar si hay algún fichero infectado.

Normalmente, se suele hacer volcados de más de un proceso, ya que se debe de sacar todos los procesos relacionados con el proceso malicioso en cuestión, es decir, los procesos padres que llaman al proceso sospechoso, por ejemplo, suponemos el caso de que existe un proceso *AdobeAcrobatReader.exe* y es llamado por el proceso padre *Firefox.exe*, por lo tanto, se debería hacer *dump* de ambos procesos. O, por ejemplo, como comentamos anteriormente, si suponemos que el proceso *insmod* es un proceso malicioso, deberíamos hacer un volcado de todos los subprocesos que intervienen en la ejecución del comando *insmod*, que corresponde con el proceso padre *systemd* cuyo PID=1 (y sus correspondientes subprocesos) [Visualización de procesos en ejecución]. Si son pocos procesos el análisis de estos individualmente no debería de ser un problema. Sin embargo, si estamos frente a un caso en donde existen varios procesos maliciosos o simplemente queremos analizar todos los procesos de un volcado de memoria, el análisis individual de cada uno de ellos puede ser una carga de trabajo demasiado grande. Imaginaros buscar todas las conexiones HTTP de todas las cadenas de texto de todos los procesos activos del sistema. Dicho trabajo conlleva mucho tiempo y un esfuerzo excepcional.

Frente a esta situación, lo que se recomienda es la creación de *scripts* que automaticen el proceso de análisis de las cadenas de texto. Por ejemplo, en la “Ilustración 39. Resultado de la ejecución del script.sh.”, se muestra el resultado de la ejecución de un script que busca en todos los procesos descargados del sistema todas las conexiones HTTP y se muestran por pantalla. En vista de los resultados, no se encuentra ninguna URL que se considere maliciosa. Al igual que se buscan ficheros PDF en todos los procesos utilizando la herramienta *foremost*, no encontrándose ninguno.

³⁰ Grep, <https://es.wikipedia.org/wiki/Grep>

³¹ Strings, <https://linux.die.net/man/1/strings>

³² Foremost, <https://www.solvetic.com/tutoriales/article/7900-como-usar-foremost-linux-y-recuperar-archivos-borrados/>

³³ Virustotal, <https://www.virustotal.com/gui/home>

Esta sección puede abarcar en sí misma un proyecto completo debido a la gran cantidad de patrones de ataques a analizar. Suele ser la parte del análisis del volcado de memoria que lleva más tiempo y de mayor carga de trabajo debido precisamente a lo complejo y las peculiaridades de cada caso en particular, por ejemplo, el caso de la empresa que sufre una infección a través de un PDF malicioso. Precisamente en este caso, debemos buscar PDFs en los procesos, código JavaScript tanto en las cadenas de texto, como en los ficheros PDF (usando, por ejemplo, la herramienta PDFID³⁴), URLs maliciosas, etc. Como estamos en un trabajo general de análisis de un volcado estándar, es decir, no infectado previamente de manera intencional, se mostrará el proceso general a seguir y la idea principal en la que se basa esta fase de análisis, sin entrar en demasiado detalle de implementación ya que no estamos en un caso específico de investigación.

En el anexo denominado “Analizando los procesos” se muestra la creación e implementación del script para analizar las cadenas de texto de los diferentes procesos, así como la utilización de la herramienta *foremost* para la búsqueda de ficheros en estos.

Memoria de los procesos

Esta sección está bastante relacionada a las dos anteriores. A diferencia de las anteriores, donde priorizamos los procesos en sí y su relación con los demás procesos del sistema (PID, PPID, UID), en esta sección nos adentraremos en la distribución de los procesos en la memoria, es decir, las posiciones de memoria que cada proceso ocupa en la RAM (*memory mapping*). Volatility dispone de un plugin denominado *memmap* que permite saber exactamente qué páginas residen en la memoria relacionado a un proceso específico (o todos si no se especifica ninguno). Para cada proceso se muestra la dirección virtual de la página, el desplazamiento físico correspondiente a esta y el tamaño total de la página. Y si además queremos más información, tenemos disponible el plugin *proc_maps*. Dicho plugin nos permite conocer detalles de la memoria del proceso, incluidos permisos, pilas/*stacks* y bibliotecas compartidas.

La idea en esta fase es muy similar a lo visto en la sección anterior [Analizando los procesos]. El primer paso sería observar las diferentes secciones de memoria que corresponden a un determinado proceso que estemos desconfiando y que puede ser un proceso malicioso. La idea es observar qué recursos utiliza cada proceso (permisos, bibliotecas, ficheros en uso, etc.) en busca de configuraciones que puedan resultar extrañas a primera vista. Una vez identificamos un segmento de memoria que contiene una predisposición sospechosa, podemos ser capaces de generar un volcado de dicho segmento de memoria para el posterior análisis, de manera manual o mediante la utilización de herramientas de terceros, por ejemplo, *memtester*³⁵. Por ejemplo, podemos imaginar que un proceso a nivel de usuario tiene acceso de escritura sobre un directorio/fichero protegido del sistema que sólo los administradores o el usuario root tienen acceso. Este comportamiento a simple vista puede resultar extraño por lo que puede estar bajo una actividad sospechosa, por lo que conviene ser analizado en busca de malware.

Si observamos la salida al ejecutar el comando *linux_proc_maps*, con relación al proceso *insmod*, observamos que una de sus secciones de memoria apunta al directorio */home* con permisos de lectura. Probablemente debido al momento cuando era necesario leer el módulo de LiME (.ko)

³⁴ PDFID, <https://tools.kali.org/forensics/pdfid>

³⁵ Memtester, <https://linux.die.net/man/8/memtester>

para la generación del volcado de memoria [Ilustración 27. Generando el volcado de memoria con LiME.]:

Offset	Pid	Name	Start	End	Flags	Pgoff	Major	Minor	Inode	File Path
0x0000000e970de00	1848	insmod	0x0000000b7f9000	0x0000000b7f91000	r--	0x29000	8	1	2623200	/
0x0000000e970de00	1848	insmod	0x0000000b7f91000	0x0000000b7f92000	rw-	0x2a000	8	1	2623200	/
0x0000000e970de00	1848	insmod	0x0000000b7fa4000	0x0000000b7fa8000	r--	0x0	8	4	5505150	/home
0x0000000e970de00	1848	insmod	0x0000000b7fa8000	0x0000000b7faa000	rw-	0x0	0	0	0	

A parte de esto, la disponibilidad de los procesos no invita a desconfiar de ellos. Aparentemente podemos asegurar en vista de los resultados que no hay ningún proceso malicioso. Pero si imaginamos que en lugar del directorio */home*, *insmod* tiene acceso a */etc/shadow*, por poner un ejemplo, que sabemos que es un archivo protegido con las credenciales de los usuarios en el sistema, invita a desconfiar del proceso en sí, de ahí de la importancia de ver qué ficheros o directorios utiliza cada proceso para identificar vulnerabilidades y malware.

Volatility nos ofrece un plugin que considero bastante útil. El plugin al que me refiero es el denominado *linux_bash*, el cual nos permite ver el historial completo de las instrucciones del *bash* de Linux. Esto nos permite saber qué instrucciones ha introducido por consola el usuario y que nos puede ser de gran utilidad en cualquier caso jurídico en el que, por ejemplo, el acusado haya intentado borrar pruebas o haya existido alguna manipulación en los ficheros del sistema. Si ejecutamos este plugin sobre el volcado de memoria, observamos todos los comandos que he utilizado para generar el volcado en el sistema que estamos auditando, así como los comandos para la instalación de las librerías necesarias para ello [Ilustración 46. Visualización del historial de instrucciones ejecutadas a través del bash.].

En el anexo “Mapeo de memoria de los distintos procesos” se muestran los resultados de las diferentes instrucciones antes mencionadas, así como la salida completa de las ejecuciones de las diferentes herramientas sobre el volcado de memoria a tratar.

Objetos y memoria del kernel

En esta sección, el objetivo es similar a los anteriores. Primeramente, sería conveniente visualizar la lista de módulos del kernel cargados en el momento de la toma del volcado de la memoria. El objetivo de esto es identificar los posibles módulos del kernel que puedan contener problemas de seguridad o los relacionados con la actividad de un malware en nuestro sistema. Una vez identificamos los módulos kernel que sospechamos que dispongan de problemas de seguridad lo conveniente sería su volcado e inspección individual, para ello, Volatility nos ofrece la herramienta denominada *linux_moddump* que nos permite generar un volcado completo del módulo kernel que le indiquemos.

Cuando realizamos un volcado completo de un determinado módulo, se nos generará un fichero en formato *.lkm*. LKM (*Loadable Kernel Module*), el cual es un archivo/objeto que contiene código para extender el núcleo o kernel en ejecución de un sistema operativo. Los LKM generalmente se usan para agregar soporte para hardware (como controladores de dispositivo) y/o sistemas de archivos, o para agregar llamadas al sistema (*syscalls*) [41]. Uno de estos sistemas de archivos a los que los LKM dan soporte es precisamente al sistema *tmpfs*. El sistema de archivos *tmpfs* es el nombre que recibe un sistema de almacenamiento en muchos sistemas operativos de tipo Unix, como memoria volátil (normalmente montado sobre */tmp*). Es similar

a los discos RAM, que aparecen como discos virtuales, y pueden contener sistemas de archivos [42]. Como los datos están principalmente en memoria volátil, las velocidades para realizar operaciones en *tmpfs* son generalmente mucho mayores en comparación con un sistema de archivos en otros dispositivos de almacenamiento como discos rígidos.

Por usar memoria volátil, los datos en *tmpfs* no persisten después de reiniciar el sistema. Esta propiedad es aprovechada por diferentes malware, como, por ejemplo, los rootkits, debido a que pueden utilizar dichos directorios para almacenar información que en el momento del apagado será eliminada (convendría recuperar este sistema de ficheros para su análisis). Por lo tanto, en esta sección comprobaremos que módulos del kernel estaban cargados en búsqueda de algún módulo que resulte sospechoso, utilizando para ello el plugin que ofrece Volatility denominado *linux_lsmod*. En vista de los resultados [Ilustración 47. Plugin *linux_lsmod*.] podemos observar que la mayoría de los módulos kernel dan soporte a hardware y a estructuras de ficheros. Algunos interesantes son los siguientes:

- Relacionados con hardware: *psmouse*, *ohci_pci*, *cdrom*, *usbhid*, *ac97_bus*, etc. La mayoría de dichos módulos dan soporte a dispositivos de entrada y salida, conectores, buses, arquitectura de procesadores, etc.
- Relacionados con sistemas de ficheros: *mbcache*, *ext4*, *ex3*, *nfs*³⁶, *fscrypto*, etc. Soportan diferentes tipos de sistemas de ficheros, permisos sobre estos, seguridad, etc.
- Módulo *VBoxGuest*. Se encuentra cargado el módulo que permite administrar y supervisar los “invitados” o máquinas virtuales perteneciente a VirtualBox (VirtualBox Guest Additions³⁷).
- Módulo *IP_tables*. Está cargado el módulo referente a las políticas de seguridad (*iptables* para configurar el cortafuegos del kernel de Linux)³⁸. Por lo que podemos contar con que el sistema dispone de cortafuegos en funcionamiento (aunque no conocemos las políticas ni las reglas que rigen la protección). Este aspecto es importante saberlo sobre todo para la siguiente sección: “Visualización de la red”.

En general, a simple vista no se encuentra ningún módulo que incite a la sospecha (haciendo hincapié en los parámetros que requieren al lanzarlo). Por lo general no hay ningún módulo que requiera atributos o parámetros fuera de lugar (suelen ser mayormente atributos numéricos de cantidad de píxeles, referentes a controladores de vídeos, o aceleración en caso del controlador del ratón). Sin embargo, entre todos los módulos aparece un módulo adicional que llama la atención debido a que ya lo conocemos de secciones anteriores (y no se trata de un módulo que aparezca de manera natural en un sistema limpio). Se trata del módulo de LiME [Generando el volcado de memoria de Raspbian con LiME.]. Dicho módulo dispone de la siguiente estructura:

```
lime 24576
    compress=0
    timeout=1000
    digest=(null)
    localhostonly=0
    format=lime
    dio=0
    path=../../raspbian.mem
```

³⁶ Nfs, https://es.wikipedia.org/wiki/Network_File_System

³⁷ Guest Additions, <https://www.virtualbox.org/manual/ch04.html>

³⁸ Iptables, [https://wiki.archlinux.org/index.php/Iptables_\(Espa%C3%B1ol\)](https://wiki.archlinux.org/index.php/Iptables_(Espa%C3%B1ol))

Hablemos detenidamente del significado general. El primer valor que nos aparece, posterior al nombre del módulo en sí, corresponde al tamaño en bytes (24576). Seguido disponemos todos los parámetros con lo que se ha ejecutado el módulo en el sistema. Se establece un *timeout* de 1000 milisegundos, no se establece compresión al indicarle el valor 0, no se especifica cifrado, formato de tipo *lime* y por último aparece una ruta (*../..../raspbian.mem*), que en este caso corresponde a la salida del volcado de memoria. Como fui yo mismo quién ejecutó el módulo (con la utilidad *insmod*) en el momento de generar el volcado de memoria [Ilustración 27. Generando el volcado de memoria con LiME.], sabemos que no se trata de un módulo malicioso, pero a primera vista podríamos considerar que no es un proceso similar a los otros, sobre todo debido al parámetro “*Path*”, ya que ningún otro módulo activo del kernel en la memoria tiene un parámetro que haga referencia a un directorio y menos a un archivo en concreto y esto puede causar sospechas (por ejemplo, un módulo de un rootkit que utiliza un fichero situado en el sistema de ficheros *tmpfs*, bajo el directorio */tmp* tal y como vimos anteriormente).

Si al ver los diferentes módulos del sistema, observamos uno que resulta sospechoso, el siguiente paso natural es realizar un volcado completo de dicho módulo para su análisis, con la herramienta *moddump*. Por ejemplo, se podría realizar un volcado del núcleo de la herramienta LiME para demostrar el proceso tal y como se puede observar en el anexo “Memoria y objetos del kernel Linux”. Posteriormente habría que analizar dicho fichero utilizando herramientas automáticas de terceros o de manera manual viendo las diferentes cadenas de textos en busca de indicios de actividad maliciosa.

Con relación al sistema de ficheros *tmpfs*, Volatility ofrece la herramienta denominada *linux_tmpfs*. Dicha herramienta nos permite, por un lado, listar todos los directorios temporales disponibles del sistema de ficheros y, por otro lado, realizar un vaciado de todos los ficheros que dichos directorios contengan, en caso de que haya. Por ejemplo, tal y como se puede ver en la ilustración [Ilustración 48. Sistema de ficheros tmpfs.], la herramienta nos devuelve los siguientes directorios (identificados con un valor del 1 al 6):

- 1 -> /sys/fs/cgroup
- 2 -> /dev
- 3 -> /run
- 4 -> /run/lock
- 5 -> /dev/shm
- 6 -> /run/user/1000

Dichos directorios corresponden a directorios temporales pertenecientes al sistema de ficheros *tmpfs*, por lo tanto, estamos frente a directorios volátiles. Si intentamos recuperar los ficheros pertenecientes a dichos directorios, observamos que ninguno de dichos directorios dispone de ficheros descargables, por lo que la aplicación no nos devuelve ninguno.

En definitiva, este apartado cobra de una vital importancia en la búsqueda de rootkits, sobre todo, ya que podemos observar que módulos están activos en el sistema y si dichos módulos están utilizando recursos protegidos o tienen un comportamiento sospechoso. En el anexo “Memoria y objetos del kernel Linux” se puede acceder a todas las pruebas realizadas sobre el volcado de memoria, así como la salida completa de todas las herramientas utilizadas.

Antes de concluir esta sección, me gustaría mencionar en este apartado relacionado con el kernel de Linux, de la importancia de las bases de datos de vulnerabilidades, tales como:

- *National Vulnerabilities Database*: <https://nvd.nist.gov/>
- *Common Vulnerabilities and Exposures*: <https://cve.mitre.org/>
- *CVE Details*: <https://www.cvedetails.com/>
- *INCIBE-CERT* (Centro de Respuesta a Incidentes de Seguridad de INCIBE): <https://www.incibe-cert.es/alerta-temprana/vulnerabilidades>

En las cuales podemos ver en tiempo real, de las últimas vulnerabilidades que afectan a los diferentes sistemas operativos o versiones del kernel. Normalmente dichas vulnerabilidades están identificadas por su CVE. Las *vulnerabilidades y exposiciones comunes* (en inglés, *Common Vulnerabilities and Exposures*, siglas CVE), es una lista de información registrada sobre vulnerabilidades de seguridad conocidas, en la que cada referencia tiene un número de identificación CVE-ID, descripción de la vulnerabilidad, que versiones del software están afectadas, posible solución (si existe) o como configurar para mitigar la vulnerabilidad y referencias a publicaciones o entradas de foros o blog donde se ha hecho pública la vulnerabilidad o se demuestra su explotación [54].

Esto es de vital importancia para saber en todo momento las últimas vulnerabilidades conocidas que afectan a nuestro sistema y que son susceptibles de sufrir ataques si no se reacciona a tiempo aplicando los parches con la solución o modificando la máquina debidamente para mitigar los posibles efectos que puedan ocasionar. Por ejemplo, como sabemos la versión kernel (4.19) del volcado de memoria, podemos hacer una búsqueda rápida de todas las vulnerabilidades que afectan al sistema:

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2019-17351	400		DoS	2019-10-07	2019-10-11	4.9	None	Local	Low	Not required	None	None	Complete
An issue was discovered in drivers/xen/balloon.c in the Linux kernel before 5.2.3, as used in Xen through 4.12.x, allowing guest OS users to cause a denial of service because of unrestricted resource consumption during the mapping of guest memory, aka CID-6ef36ab967c7.														
2	CVE-2019-16995	772		DoS	2019-09-30	2019-10-04	7.8	None	Remote	Low	Not required	None	None	Complete
In the Linux kernel before 5.0.3, a memory leak exists in hsr_dev_finalize() in net/hsr/hsr_device.c If hsr_add_port fails to add a port, which may cause denial of service, aka CID-6caabe7f197d.														
3	CVE-2019-16994	772		DoS	2019-09-30	2019-10-04	7.8	None	Remote	Low	Not required	None	None	Complete
In the Linux kernel before 5.0, a memory leak exists in sit_init_net() in net/ipv6/sit.c when register_netdev() fails to register sitn->fb_tunnel_dev, which may cause denial of service, aka CID-07f12b26e21a.														
4	CVE-2019-16714	200		+Info	2019-09-23	2019-09-24	5.0	None	Remote	Low	Not required	Partial	None	None
In the Linux kernel before 5.2.14, rds6_inc_info_copy in net/rds/recvc.c allows attackers to obtain sensitive information from kernel stack memory because tos and flags fields are not initialized.														
5	CVE-2019-16413	835		DoS	2019-09-18	2019-10-04	5.0	None	Remote	Low	Not required	None	None	Partial
An issue was discovered in the Linux kernel before 5.0.4. The 9p filesystem did not protect i_size_write() properly, which causes an i_size_read() infinite loop and denial of service on SMP systems.														

Ilustración 6. Vulnerabilidades que afectan al kernel 4.19 de Linux.

En la imagen anterior³⁹ se muestran unos ejemplos de las muchas vulnerabilidades que afectan a la versión 4.19 del kernel de Linux. Cada vulnerabilidad está identificada con un ID, posibles *exploits*⁴⁰ que afecten a la vulnerabilidad, el tipo de vulnerabilidad (por ejemplo, *DDoS*⁴¹), fecha de publicación, etc. Si accedemos a ellas, nos aparecerán las posibles soluciones (si las hay) que mitiguen el impacto de cada vulnerabilidad.

³⁹ https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/version_id-261041/Linux-Linux-Kernel-4.19.html

⁴⁰ Exploits, <https://es.wikipedia.org/wiki/Exploit>

⁴¹ DDoS, https://es.wikipedia.org/wiki/Ataque_de_denegaci%C3%B3n_de_servicio

Visualización de la red

En esta sección se tratará de toda la información relacionada con la red del sistema al que pertenece la memoria. Desde la enumeración de las diferentes interfaces de red, hasta el listado de la tabla ARP, pasando por las conexiones abiertas, puertos en escucha y sockets disponibles. Este apartado tiene bastante importancia ya que podemos observar que procesos están a la escucha y pueden ser objetivos de ataques de denegación de servicio, por ejemplo (ignorando el uso de cortafuegos), o puertas traseras/backdoors⁴².

El primer paso sería comprobar que tarjetas de red (o interfaces) disponemos en el sistema. La tarjeta de red, también conocida como placa de red, adaptador de red o sus términos en inglés *Network Interface Card* o *Network interface controller* (NIC), es un componente de hardware que conecta un ordenador a una red informática y que posibilita compartir recursos (como archivos, discos duros enteros, impresoras e internet) entre dos o más computadoras, es decir, en una red de computadoras. Es el principal medio por el cual podemos conectarnos a internet y desde internet a nosotros (esto implica que es la principal fuente de infección de malware, es decir, sin interfaz de red no podríamos acceder a internet, ergo el sistema sería más seguro, pero perderíamos utilidad) [43]. Cuando hablamos de conexión a internet y viceversa, es decir, de internet a nosotros, existe una capa principal de seguridad que permite o no dichas conexiones en base a una política de reglas. Dicha capa de seguridad se trata de los denominados cortafuegos.

En informática, un cortafuegos (del término original en inglés *firewall*) es la parte de un sistema informático que está diseñada para bloquear el acceso no autorizado, permitiendo al mismo tiempo comunicaciones autorizadas. Se trata de un dispositivo o conjunto de dispositivos configurados para permitir, limitar, cifrar o descifrar el tráfico entre los diferentes ámbitos sobre la base de un conjunto de normas y otros criterios. Los cortafuegos pueden ser implementados en hardware (por ejemplo, serie Firepower de CISCO⁴³) o software (por ejemplo, *iptables*), o en una combinación de ambos.

Debido a que, en este TFM, nos dedicamos a analizar el volcado de memoria a nivel de software y procesos, sin hacer hincapié en la configuración del cortafuegos del sistema, supondremos el peor de los casos. Sabemos, gracias a la sección anterior [Objetos y memoria del kernel] que el módulo de los *iptables* está cargado en el sistema, sin embargo, no podemos acceder a la configuración y a las políticas de este, por lo que suponemos que se nos permite toda conexión al exterior, así como toda la conexión entrante.

Normalmente en sistemas Linux, existe un comando denominado *ifconfig*⁴⁴, que nos permite saber que tarjetas de red están actualmente operativas en el sistema, así como configurar dichas interfaces. A su vez Volatility nos ofrece una herramienta denominada *linux_ifconfig* que intenta simular precisamente dicho comportamiento. En vista de los resultados [Ilustración 50. Visualización de las interfaces de red con el plugin *linux_ifconfig*.], observamos que nos devuelve dos interfaces:

⁴² Backdoor, https://es.wikipedia.org/wiki/Puerta_trasera

⁴³ CISCO, https://www.cisco.com/c/es_es/products/security/firewalls/index.html#~productos

⁴⁴ Ifconfig, <https://linux.die.net/man/8/ifconfig>

- 1º interfaz. *lo* (*Loopback*). El dispositivo de red *loopback* es una interfaz de red virtual. Las direcciones del rango '127.0.0.0/8' son direcciones de *loopback*, de las cuales se utiliza, de forma mayoritaria, la '127.0.0.1' (como en este caso) por ser la primera de dicho rango. La dirección de *loopback* es una dirección especial que los hosts utilizan para dirigir el tráfico hacia ellos mismos [70]. Observamos también que la dirección MAC⁴⁵ está formada por valores 0: 00:00:00:00:00:00.
- 2º Interfaz. *eth0*. Interfaz ethernet 0. Dicha interfaz es la que nos comunica con internet. Tiene como dirección IP: 10.0.2.15 y, como dirección MAC: 08:00:27:df:de:ed.

Un aspecto importante a tener en cuenta para mejorar la configuración de la red es que, ambas tarjetas de red, en especial *eth0*, está con el modo promiscuo (*Promiscuous Mode*) en falso. El modo promiscuo es aquel en el que un ordenador conectado a una red captura todo el tráfico que circula por ella (este modo está muy relacionado con los *sniffers*⁴⁶ que se basan en este modo para realizar su tarea). Por lo tanto, conviene activar el modo promiscuo en ambas tarjetas (basta con modificar el fichero */etc/rc.local* e introducir el valor *ifconfig eth0 promisc*) para, en caso de fallos en la red o cuando recibamos conexiones externas en caso de que estemos frente a un sistema que ofrezca servicios al exterior, es de vital importancia capturar todo el tráfico para su posterior análisis (ante ataques DoS, por ejemplo).

Una vez conocemos las interfaces de red activas en el sistema. El siguiente paso podría ser la visualización de la tabla ARP. Para poder enviar paquetes de datos en redes TCP/IP, un servidor necesita, sobre todo, tres datos de dirección sobre el host al que se dirige: la máscara de subred, la dirección IP y la dirección MAC. Los dispositivos reciben la máscara de red y la dirección IP de manera automática y flexible cuando se establece la conexión con una red. Con este objetivo, los dispositivos de comunicación mediadores como *routers* o concentradores (*hubs*) recurren al protocolo DHCP⁴⁷. El fabricante del dispositivo correspondiente otorga la dirección de hardware, que queda vinculada a una dirección IP con ayuda del llamado *Address Resolution Protocol* (ARP) [70][46]. El *Address Resolution Protocol* (protocolo de resolución de direcciones) fue especificado en 1982 en el estándar RFC 826 para llevar a cabo la resolución de direcciones IPv4 en direcciones MAC. ARP es imprescindible para la transmisión de datos en redes Ethernet por dos razones: por un lado, las tramas de datos (también tramas *Ethernet*) de los paquetes IP solo pueden enviarse con ayuda de una dirección de hardware a los hosts de destino, pero el protocolo de Internet no puede obtener estas direcciones físicas por sí mismo. Por el otro, y debido a su limitada longitud, el protocolo IPv4 carece de la posibilidad de almacenar las direcciones de los dispositivos. Con un mecanismo de caché propio, el protocolo ARP también es, aquí, la solución más adecuada [46].

Desde Volatility es posible acceder a dicha tabla ARP de manera sencilla mediante la utilización del plugin *linux_arp* [Ilustración 49. Visualización de la tabla ARP.]. Tal y como se puede observar, se muestra la tabla relacional de las direcciones IP con las direcciones MAC sobre la interfaz *eth0*. El objetivo de comprobar el estado de esta tabla es impedir la utilización de técnicas maliciosas para comprometer el estado de la máquina. Una de las técnicas más utilizadas es la denominada suplantación de ARP.

⁴⁵ MAC, https://es.wikipedia.org/wiki/Direcci%C3%B3n_MAC

⁴⁶ Sniffers, https://es.wikipedia.org/wiki/Analizador_de_paquetes

⁴⁷ DHCP, https://es.wikipedia.org/wiki/Protocolo_de_configuraci%C3%B3n_din%C3%A1mica_de_host

La suplantación de ARP (en inglés *ARP spoofing* o *ARP poisoning*) es enviar mensajes ARP falsos a la *Ethernet*. Normalmente la finalidad es asociar la dirección MAC del atacante con la dirección IP de otro nodo (el nodo atacado), como por ejemplo la puerta de enlace predeterminada. Cualquier tráfico dirigido a la dirección IP de ese nodo, será erróneamente enviado al atacante, en lugar de a su destino real (*man-in-the-middle attack*, MitM⁴⁸). El atacante, puede entonces elegir, entre reenviar el tráfico a la puerta de enlace predeterminada real (ataque pasivo o escucha), o modificar los datos antes de reenviarlos (ataque activo). El atacante puede incluso lanzar un ataque de tipo DoS (denegación de servicio) contra una víctima, asociando una dirección MAC inexistente con la dirección IP de la puerta de enlace predeterminada de la víctima. El ataque de suplantación de ARP puede ser ejecutado desde una máquina controlada (el atacante ha conseguido previamente hacerse con el control de esta: intrusión), o bien la máquina del atacante está conectada directamente a la red local Ethernet [47].

Como estamos frente a un volcado de memoria de un sistema recién creado, dicha tabla ARP actualmente no es muy densa y no ha sido intervenida maliciosamente por lo que no se encuentran indicios de actividad maliciosa. Existen herramientas que pueden simular dichos ataques, como es la herramienta *Ettercap*⁴⁹, disponible en las distros de Kali Linux.

Por último, Volatility nos ofrece dos herramientas que se suelen ejecutar a la vez, ya que en gran parte van ligadas y que pueden sonar conocidas debido a que intentan simular comandos existentes de Linux. Dichas herramientas son *linux_netstat* y *linux_netscan*. Por una parte, *linux_netstat* imita el comando *netstat* de los sistemas Linux. Aprovecha la funcionalidad *linux_lsof* para enumerar los puertos abiertos en cada proceso, es decir, comprueba si existen sockets abiertos (muestra un listado de las conexiones activas de un sistema, tanto entrantes como salientes). Por otra parte, *netscan*, comprueba las estructuras de conexión de la red (comprueba puertos en escucha o cerrados).

La idea de utilizar estas herramientas es ver qué procesos están en escucha en todo momento y si deben de estar realmente en escucha. Es decir, si existe en el sistema un proceso que está en escucha y dicho proceso no se usa en ningún momento, por ejemplo, un proceso relacionado con impresoras a través de bluetooth, conviene cerrar dichos procesos debido a que pueden ser focos de vulnerabilidades. Por otra parte, *netstat* nos permite ver todas las conexiones actuales abiertas, ya sean externas o internas, por lo que puede ser útil para detectar ataques de denegación de servicio, conexiones a web maliciosas, detectar intrusos a través de *backdoors*, malware, etc.

En el caso que nos ocupa, afortunadamente al ejecutar el comando *linux_netstat* observamos que en el momento de la toma de la imagen no había ninguna conexión activa en el sistema. Esto es debido a que estamos frente a un volcado de ejemplo creado desde cero, específicamente para el desarrollo del TFM, no se trata de un volcado realizado en un sistema real en donde la máquina normalmente está en todo el momento en contacto con el exterior a través de internet. Por otra parte, al ejecutar el comando *linux_netscan* vemos que se muestra un proceso en escucha (así como una serie de puertos cerrados) [Ilustración 51. netscan y netstat sobre el volcado]:

⁴⁸ MitM, https://es.wikipedia.org/wiki/Ataque_de_intermediario

⁴⁹ Ettercap, <https://www.redeszone.net/2016/11/12/ataque-arp-poisoning-con-kali-linux/>

- TCP, 127.0.0.1. Puerto 25. Vemos que el puerto abierto corresponde al puerto 25 es decir, el usado por el protocolo SMTP para el envío de correos electrónicos.

Esta información es recomendable tenerla en cuenta debido a la posibilidad de que este puerto sea vulnerable por fallos de seguridad. Adicionalmente con esta información, es recomendable buscar en bases de datos de vulnerabilidades, por ejemplo [Adminsub](#), en la cual nos muestran diferentes vulnerabilidades y malware que puede afectar a dicho puerto/servicio. Todos los resultados obtenidos durante esta fase, así como la ejecución de las diferentes herramientas sobre el volcado de memoria están disponibles en el anexo “Networking”.

Información del sistema

En esta sección el objetivo es recuperar información general del sistema, tales como la CPU, sistema de ficheros montado, memoria de dispositivos de entrada y salida, etc.

El primer paso es descubrir qué procesadores y cuantos dispone el sistema, la arquitectura de este y el fabricante. El objetivo es descubrir qué vulnerabilidades dispone el sistema por el mero hecho de disponer según qué procesador, por ejemplo, la vulnerabilidad Meltdown⁵⁰ en los procesadores Intel x86 o Spectre⁵¹ y ZombieLoad⁵² en los procesadores de novena generación de Intel que permite acceder a información privilegiada.

Para recuperar esta información desde Volatility, se hará uso de la herramienta *linux_cpuinto* [Ilustración 52. Visualización de la información de la CPU.]. En vista de los resultados obtenemos la siguiente información: *GenuineIntel Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz*. Con lo cual sabemos que el sistema dispone de una sola CPU, en este caso se trata de un Intel i7-6700K a 4 GHz. Como estamos auditando un volcado de memoria proveniente de un sistema virtualizado, la CPU que nos aparece se corresponde con la CPU del sistema anfitrión.

Dicha gama de procesadores eran objetivo de las vulnerabilidades antes mencionadas (*Meltdown* y *Spectre*), pero afortunadamente hoy en día ya han sido parcheadas debidamente. Para saber si nuestro procesador es vulnerable a las vulnerabilidades anteriores, se puede hacer uso de la herramienta MDSTools de [MDSAttack](#) (sólo para procesadores Intel).

Esta herramienta identifica la CPU del sistema y según la información que dispone nos indica si somos vulnerables a los ataques (cada ataque está debidamente explicado en la web de [MDSAttack](#)):

- Direct/Indirect Branch Speculation.
- Speculative Store Bypass.
- Meltdown.
- L1 Terminal Fault.
- Micro-architectural Data Sampling.

⁵⁰ Meltdown, [https://es.wikipedia.org/wiki/Meltdown_\(vulnerabilidad\)](https://es.wikipedia.org/wiki/Meltdown_(vulnerabilidad))

⁵¹ Spectre, [https://es.wikipedia.org/wiki/Spectre_\(vulnerabilidad\)](https://es.wikipedia.org/wiki/Spectre_(vulnerabilidad))

⁵² ZombieLoad, <https://www.xataka.com/componentes/zombieload-vulnerabilidad-que-afecta-a-procesadores-intel-2011-que-permite-acceder-a-datos-privados-pc>

System	
Operating system:	Windows 10 Pro
Processor:	Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
Microarchitecture:	Skylake
Microcode:	0x00000000c6
Memory:	16.00 GiB
Direct Branch Speculation	
Status:	Vulnerable
_user pointer sanitization:	Disabled
Indirect Branch Speculation	
Status:	Vulnerable
Retpoline:	Disabled
IBPB:	Always
IBRS:	Enabled
STIBP:	Enabled
SMEP:	Disabled
Speculative Store Bypass	
Speculative Store Bypass	Vulnerable
Speculative Store Bypass Disable:	OS Support
Meltdown	
Status:	Vulnerable
KPTI Present:	Yes
KPTI Enabled:	Yes
PCID Accelerated:	Yes
PCID Invalidation:	Yes
L1 Terminal Fault	
Status:	Vulnerable
L1TF Present:	Yes
PTE Inversion:	Yes
SMT:	Vulnerable
L1d Flush Present:	No
L1d Flush:	Available
Micro-architectural Data Sampling	
Line Fill Buffers (MFBDS):	Vulnerable
Store Buffers (MSBDS):	Vulnerable
Load Ports (MLPDS):	Vulnerable
Uncached Memory (MDSUM)	Vulnerable
SMT:	Vulnerable
MD_CLEAR:	Not Available

Ilustración 7. MDS Tool para la búsqueda de vulnerabilidades.

Existe información técnica de dichas vulnerabilidades (CVE-2018-12126, CVE-2018-12127, CVE-2018-12130 y CVE-2019-11091) en [la web de Intel](#) y de los ataques asociados (*ZombieLoad*, *Fallout*⁵³, *RIDL*⁵⁴ y *Store-to-Leak Forwarding*⁵⁵) en [CPU.fail](#) [48]. Hoy en día existen parches que nos protegen frente a estas vulnerabilidades (aunque suponga la disminución del rendimiento en los procesadores, en algunos casos hasta un 20% según Intel [48]).

Una vez conocemos la información relativa a la CPU el siguiente paso será recuperar los mensajes de depuración del kernel. Esto nos permite comprobar si a la hora de cargar los diferentes módulos del kernel se ha producido errores, por ejemplo, de permisos. Además, se mostrará todos los módulos del kernel activos, los avisos de cada uno y, en general, toda la información de los módulos del kernel que nos puedan dar pistas para, por ejemplo, detectar

⁵³ Fallout, <https://mdsattacks.com/files/fallout.pdf>

⁵⁴ RIDL, <https://www.bleepingcomputer.com/news/security/new-ridl-and-fallout-attacks-impact-all-modern-intel-cpus/>

⁵⁵ Store-to-Leak Forwarding , <https://arxiv.org/pdf/1905.05725.pdf>

rootkits. Para ello Volatility ofrece una herramienta denominada *linux_dmesg*. Entre la información que podemos encontrar se encuentra la siguiente:

- EXT4-fs (sda4): mounted filesystem with ordered data mode.
- lime: module verification failed: signature and/or required key missing - tainting kernel
- Console: switching to colour frame buffer device 100x37
- input: VirtualBox mouse integration as
/devices/pci0000:00/0000:00:04.0/input/input7

Como se puede observar de los mensajes anteriores, la mayoría de los mensajes son avisos de todas las actividades que los módulos del kernel van generando. Por ejemplo, en el primero se muestra que se ha montado un sistema de ficheros de tipo EXT4 (en /sda4). En el segundo, el comando LiME espera una firma en la orden del comando y ha generado un aviso de que no se ha podido verificar la identidad debido a que no se ha pasado por parámetro ninguna clave. Las dos últimas corresponden a verificación de hardware en el sistema. Estos *logs* pueden ser muy útiles para buscar rootkits o módulos sospechosos en el sistema (visto los resultados, no se han encontrado nada destacable que puedan generar sospechas con relación a algún módulo o proceso malicioso), así como vulnerabilidades a nivel de módulos del kernel.

El siguiente paso será comprobar la memoria reservada para los dispositivos de entrada y salida del sistema, tales como buses PCI, ROM, etc. Normalmente esta memoria y los dispositivos de entrada y salida suelen estar gestionada por lo que se denomina unidad de gestión de memoria de entrada y salida (IOMMU). Una unidad de gestión de memoria de entrada y salida (*input-output memory management unit*, acrónimo IOMMU) es una unidad de gestión de memoria (MMU) que conecta un bus de E/S con capacidad de acceso directo a memoria (compatible con DMA⁵⁶, *Direct Memory Access* o acceso directo a memoria, permite a cierto tipo de componentes de un ordenador acceder a la memoria del sistema para leer o escribir independientemente de la CPU) a la memoria principal. Al igual que una MMU⁵⁷ tradicional, que traduce direcciones virtuales visibles de CPU a direcciones físicas, IOMMU correlaciona direcciones virtuales visibles de dispositivo (también llamadas direcciones de dispositivo o direcciones de E/S en este contexto) a direcciones físicas. Algunas unidades también brindan protección de memoria contra dispositivos defectuosos o maliciosos [49].

Los ataques basados en el acceso directo a la memoria permiten que un atacante ponga en peligro el sistema específico en pocos segundos al conectar un dispositivo malicioso de conexión en caliente, como una tarjeta de red externa, mouse, teclado, impresora, almacenamiento y tarjeta gráfica. Dichos ataques son posibles ya que el puerto Thunderbolt⁵⁸ permite que los periféricos conectados eviten las políticas de seguridad del sistema operativo para acceder a la memoria del sistema de lectura/escritura directa que contiene información confidencial, incluidas las contraseñas, los inicios de sesión bancarios, los archivos privados y la actividad del navegador. Esto significa que, con el simple hecho de conectar un dispositivo infectado se puede manipular el contenido de la memoria y ejecutar código arbitrario con privilegios mucho más

⁵⁶ DMA, https://es.wikipedia.org/wiki/Acceso_directo_a_memoria

⁵⁷ MMU, https://es.wikipedia.org/wiki/Unidad_de_gesti%C3%B3n_de_memoria

⁵⁸ Thunderbolt, [https://es.wikipedia.org/wiki/Thunderbolt_\(bus\)](https://es.wikipedia.org/wiki/Thunderbolt_(bus))

altos que los periféricos de bus serie universal, permitiendo a los atacantes eludir la pantalla de bloqueo o controlar los sistemas de forma remota.

Para bloquear los ataques basados en DMA, la mayoría de los sistemas operativos y dispositivos aprovechan precisamente la técnica de protección de la unidad de administración de memoria de entrada/salida (IOMMU), nombrada anteriormente, para controlar qué dispositivo periférico puede acceder a la memoria y qué región de la memoria. Pero IOMMU no es un sistema completamente infalible. Existen algunas herramientas o vulnerabilidades como *ThunderClap*⁵⁹ o *Row Hammer*⁶⁰, que permiten eludir la protección de IOMMU.

Debido a las vulnerabilidades antes mencionadas, cobra vital importancia comprobar que espacios de memoria están destinados para los dispositivos I/O. Para ello Volatility nos ofrece un plugin denominado *linux_iomem* que nos devuelve el mapeo de memoria [Ilustración 54. Direcciones reservadas para dispositivos de entrada y salida.]. Entre los espacios a destacar se encuentra los siguiente:

- *PCI Bus. Peripheral Component Interconnect o PCI* (en español: Interconexión de Componentes Periféricos), es un bus estándar de ordenadores para conectar dispositivos periféricos directamente a la placa base [51].
- *Memoria ROM* (adapter ROM, video ROM, etc.). La memoria de solo lectura, conocida también como ROM (acrónimo en inglés de *read-only memory*), es un medio de almacenamiento utilizado en ordenadores y dispositivos electrónicos, que permite solo la lectura de la información y no su escritura [50].
- *ACPI Tables*. ACPI es la sigla del inglés de "*Advanced Configuration and Power Interface*" (Interfaz Avanzada de Configuración y Energía). Es un estándar que controla el funcionamiento del BIOS y proporciona mecanismos avanzados para la gestión y ahorro de la energía [52].

No se encuentra en vista a los resultados ningún espacio de memoria alterado por un módulo malicioso que de indicios de existencia de malware en el sistema.

Por último, para terminar con esta sección es hora de hablar sobre los sistemas de ficheros que están montados en el sistema. En sistemas Linux, si se quiere montar un sistema de ficheros en un determinado directorio o conocer el estado de estos se puede utilizar la orden *mount*. Volatility en un acto de simular esta herramienta, dispone del plugin *linux_mount* [Ilustración 55. Plugin *linux_mount*.].

En vista de los resultados vemos que existe una lista de varios sistemas de ficheros, con su respectivo directorio donde se encuentra montado, el tipo de sistema de fichero (EXT4, TMPFS, etc.) y los permisos que dispone cada uno. Entre todos los sistemas de ficheros montados, identifico a todos los sistemas de tipo *tmpfs*, vistos en la sección "Objetos y memoria del kernel" y, además, me llaman la atención especialmente los siguientes:

- */dev/sda4* */home* *ext4* *rw,relatime*
- */dev/sda1* */var/tmp* *ext4* *rw,relatime*
- */dev/sda2* */boot* *ext4* *rw,relatime*

⁵⁹ Thunderclap, <http://thunderclap.io/>

⁶⁰ Row Hammer, https://en.wikipedia.org/wiki/Row_hammer

Se corresponden a tres particiones del disco, uno dedicado al directorio de usuario */home*, otro al directorio */tmp* y otro para los parámetros del kernel en */boot*. Todos comparten el mismo tipo y los mismos permisos. Sin embargo, a primera vista no se identifica ningún uso malicioso en los sistemas de ficheros (ni en los permisos de cada uno).

La finalidad de esto es observar qué sistemas de ficheros están montados en el sistema para observar si se está haciendo un uso correcto de los mismos, ya que puede ser que esté bajo los efectos de un rootkit o, por ejemplo, afectado por la vulnerabilidad CVE-2015-1328 que permiten el escalado de privilegios utilizando *mount*. En el anexo “Información del sistema” se adjunta la ejecución de todas las herramientas vistas anteriormente, así como la salida completa de todas las pruebas realizadas.

Detección de rootkits

El último aspecto del análisis del volcado de memoria es con relación a los rootkits. Durante la elaboración del “Estado del arte” se indicó la problemática que suponía los rootkits y la importancia que tenía el análisis forense en la lucha contra dicho malware. Es tal la importancia que tiene que Volatility en sí ofrece una serie de herramientas que van dirigidas íntegramente al descubrimiento de este tipo de malware en los volcados de memoria. Cada herramienta se basa en un principio de búsqueda. Estos son:

- *linux_check_ainfo*. Este complemento recorre las estructuras de *file_operations* y *sequence_operations* de los protocolos UDP y TCP (por ejemplo, *tcp6_seq_ainfo* y *tcp4_seq_ainfo*), con el fin de detectar cualquier manipulación de dichas estructuras [Ilustración 56. Plugin *linux_check_ainfo* para la detección de rootkits.]. Un ejemplo de rootkit que afecta a las estructuras es el denominado *KBeast*⁶¹.
- *linux_check_tty*. Este plugin detecta uno de los métodos de *keyloggers*⁶² a nivel de núcleo descritos en "[Bridging the Semantic Gap to Mitigate Kernel-level Keyloggers](#)". Funciona comprobando el puntero de la función *receive_buf* para cada controlador *tty*⁶³ activo en el sistema [Ilustración 57. *linux_check_tty* plugin.].
- *linux_check_creds*. Este complemento detecta rootkits que tienen privilegios elevados (root) utilizando técnicas DKOM (*Direct Kernel Object Manipulation*) [Ilustración 58. Búsqueda de rootkits utilizando técnicas DKOM.] [37].
 - En los núcleos/kernel Linux 2.6 y anteriores, el ID del usuario y el ID del grupo de un proceso se mantenían como números enteros en la memoria. Para que un rootkit eleve los privilegios de un proceso, simplemente se establece estos dos valores a cero (root). Esta simplicidad también hizo que fuera muy difícil usar solo la información en la propia estructura del proceso para detectar qué procesos habían sido elevados y cuáles simplemente fueron generados por el usuario root/administrador.

⁶¹ KBeast, <https://github.com/ruckuus/kernel-abuse/tree/master/kbeast>

⁶² Keylogger, <https://es.wikipedia.org/wiki/Keylogger>

⁶³ TTY, [https://en.wikipedia.org/wiki/Tty_\(unix\)](https://en.wikipedia.org/wiki/Tty_(unix))

- Esto cambió en versiones posteriores a la 2.6 cuando el núcleo adoptó una estructura de tipo *CRED (security credentials)*⁶⁴ para administrar toda la información relacionada con los privilegios de un proceso. Esta estructura es bastante complicada y obliga a los rootkits a adaptar sus métodos de elevación de permisos. Aunque el núcleo proporciona las funciones *prepare_creds* y *commit_creds* para asignar y almacenar nuevas credenciales, varios rootkits optan por no utilizar estas funciones. En cambio, simplemente encuentran otro proceso que tiene los privilegios de root, generalmente PID 1 (*systemd*), y establecen el indicador de credibilidad del proceso de destino al del PID 1. Esto le da al proceso del atacante un control total y el rootkit no tiene que intentar la tarea no trivial de asignar su propia estructura de credenciales.
- El préstamo de estructuras de credenciales conduce a una inconsistencia que Volatility puede aprovechar para encontrar procesos con permisos root. En el funcionamiento normal del núcleo, cada proceso tiene una estructura de credenciales única y estas nunca se comparten ni se toman prestadas. El complemento *linux_check_creds* utiliza esta premisa mediante la creación de una asignación de procesos y sus estructuras *CRED* y luego informa de cualquier proceso que los comparta, tal y como se puede ver en el anexo “Detección de rootkits”.
- *linux_check_fop*. Esta herramienta enumera el sistema de archivos */proc*, localiza todos los procesos abiertos y verifica que cada miembro de cada estructura *file_operations* sea válido (válido significa que el puntero de la función está en el kernel o en un módulo de kernel cargable conocido, y no en uno oculto) [Ilustración 59. Utilización del plugin *linux_check_fop*.].
- *linux_check_idt*. Este complemento enumera las direcciones y los símbolos de la tabla del descriptor de interrupciones (IDT⁶⁵). La tabla de descriptors de interrupción es una estructura de datos utilizada por la arquitectura x86 para implementar una tabla de vectores de interrupción. El procesador utiliza el IDT para determinar la respuesta correcta a las interrupciones y excepciones [Ilustración 60. Uso del plugin *linux_check_idt*.].
- *linux_check_syscall*. Este complemento imprime las tablas de llamadas al sistema (*syscall*) [Ilustración 61. Ejemplo del plugin *linux_check_syscall*.].
 - Las llamadas al sistema comúnmente usan una instrucción especial de la CPU que causa que el procesador transfiera el control a un código privilegiado (generalmente es el núcleo), previamente especificado. Esto permite al código privilegiado especificar donde se conecta, así como el estado del procesador.
 - Cuando una llamada al sistema es invocada, la ejecución del programa que invoca es interrumpida y sus datos son guardados, normalmente en su PCB (Bloque de Control de Proceso del inglés *Process Control Block*), para poder continuar ejecutándose luego. El procesador entonces comienza a ejecutar las instrucciones de código de bajo nivel de privilegio, para realizar la tarea requerida. Cuando esta finaliza, se retorna al proceso original, y continúa su ejecución. El retorno al proceso demandante no obligatoriamente es inmediato,

⁶⁴ CRED,

https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.ichd100/scred.htm

⁶⁵ IDT, https://en.wikipedia.org/wiki/Interrupt_descriptor_table

depende del tiempo de ejecución de la llamada al sistema y del algoritmo de planificación de CPU.

- El rootkit *KBeast* también es visible frente este tipo de análisis⁶⁶.
- *linux_check_modules*. Este complemento encuentra rootkits que se separan de la lista de módulos, pero no de *sysfs*⁶⁷. No se ha encontrado un rootkit que realmente se elimine de *sysfs*, por lo que, en un sistema real, suelen estar ocultos de *lsmod* y */proc/modules*, pero aún se pueden encontrar en */sys/modules* [Ilustración 62. Utilizando el plugin *linux_check_modules*.].

En vista de los resultados obtenidos de la ejecución de las diferentes herramientas antes descritas sobre el volcado de memoria, se han obtenido los resultados esperados. Como estamos analizando el volcado de memoria de un sistema creado recientemente, es decir, no ha estado en contacto con Internet de manera prolongada (no es una memoria tomada de un sistema real), se espera que el sistema no esté afectado por malware, en específico con rootkits. Y gracias a la implementación de las diferentes herramientas podemos finalmente confirmar que el sistema se podría considerar limpio de rootkits (no existen procesos ni llamadas del sistema “enganchadas”/*hooked*, ni punteros a procesos ocultos ni discordancia en cabeceras y en general en la configuración). En el anexo “Detección de rootkits” se muestra las ejecuciones de todas las herramientas descritas en esta sección, así como la salida completa de ellas.

Breve resumen de los resultados obtenidos

Una vez se ha completado el análisis y visto los resultados, conviene realizar un breve resumen/informe de los resultados obtenidos:

- Respecto a los diferentes procesos activos durante la ejecución del sistema, a priori no se observa procesos maliciosos. La mayoría de los procesos encontrados son subprocesos de los procesos de sistema *systemd* y *kthreadd*. Además, se observan los procesos relativos a *bash*, *sudo* y *LiME*, herramientas que intervienen en el proceso de adquisición de la memoria. No se observan ni software de correo electrónico, navegadores web, aplicaciones ofimáticas, etc.
- Con relación al análisis individual de los procesos, se ha buscado cadenas de caracteres que puedan identificar procesos maliciosos, como, por ejemplo, conexiones HTTP, palabras claves, credenciales, etc. sin encontrar ningún URL maliciosa, contraseñas en texto plano, ni en general, algún aspecto que se pueda considerar extraño y que incita a desconfiar de los procesos. Adicionalmente, mediante herramientas externas a Volatility (herramientas de Unix, bases de datos de malware, etc.) se ha indagado en la búsqueda de ficheros (PDF, DOC, JPEG, etc.) dentro de los procesos que puedan contener código JavaScript que puedan ser fuente de malware. Finalmente, no se ha encontrado indicios de peligrosidad.
- La distribución/mapeo de memoria es de las secciones a las que menos hincapié se ha hecho debido a la complejidad de análisis que se requiere para interpretar cada proceso.

⁶⁶ KBeast rootkit analysis, <https://volatility-labs.blogspot.com/2012/09/movp-15-kbeast-rootkit-detecting-hidden.html>

⁶⁷ SYSFS, <https://es.wikipedia.org/wiki/Sysfs>

Sin embargo, se ha comprobado además del mapeo de memoria, el historial de instrucciones de *bash* para comprobar las diferentes instrucciones que ha realizado el usuario hasta el momento de la toma del volcado de memoria. Al resultar ser un sistema nuevo, el historial del intérprete de comandos tan sólo disponía de la información relacionada con el proceso de la adquisición del volcado de memoria.

- Una sección importante es la relacionada con los módulos del kernel, ya que nos permite ver si existen módulos del kernel maliciosos, por ejemplo, rootkits. Se ha conseguido ver los módulos cargados en el momento de la toma de la imagen sin encontrar ningún módulo malicioso. Además, se ha identificado el sistema de ficheros temporal con el fin de encontrar actividad maliciosa (*tmpfs*), quedando estos limpios después del análisis.
- Se ha hecho especial interés en la información relativa a la red del sistema. Se ha recapitulado información referente a las interfaces del sistema, sockets abiertos, conexiones establecidas en el sistema, etc. Entre la información obtenida, se puede resaltar la obtención de tan sólo una interfaz (en modo no promiscuo), un puerto abierto (relativo al 25), se ha visualizado la tabla ARP (debido a la reciente actividad en la máquina, esta no es muy extensa, además no se ha identificado ataques, como el denominado ARP Spoofing) y, por último, no existe ninguna conexión establecida en el momento de la toma de la imagen.
- En la sección referida a la adquisición de la información del sistema, se ha recopilado información referente a sistemas de ficheros montados, información referida a la CPU (con vulnerabilidades relacionadas), mensajes/logs de depuración del kernel y memoria dedicada a dispositivos de entrada y salida con sus posibles vulnerabilidades. Respecto a la información obtenida, identificamos posibles vulnerabilidades a nivel de CPU (como *Meltdown* y *Spectre*), los logs de depuración del kernel no arrojan errores graves, la memoria de los dispositivos de entrada y salida no parece estar afectada por vulnerabilidades como *Thunderclap* y, respecto a los sistemas de ficheros montados, no se reconocen peligros (con relación a qué sistemas están montados en el momento de la toma y los permisos que cada uno dispone).
- Por último, se ha recapitulado información en base a identificar rootkits en el sistema, siguiendo una variedad de métodos y focos de actuación. Afortunadamente no se ha encontrado indicios de rootkits en las pruebas realizadas y los resultados obtenidos son los esperados en todo momento.

En general con la información obtenida, podemos afirmar que estamos frente a un volcado bastante limpio (a día de la realización de este TFM). Existe alguna configuración mejorable que se ha indicado en su respectivo momento, pero no se ha identificado un error crítico que obligue a la detención del sistema (en un entorno real) debido a que no se podría asegurar la fiabilidad ni la seguridad del sistema.

Estos resultados se podrían haber previsto desde un primer momento, ya que nos encontramos frente a un volcado de memoria de un sistema “nuevo”, es decir, creado e instalado para el propósito de este TFM. No estamos frente a un sistema que esté en contacto con usuarios, internet, etc. (entorno real de uso) por lo que es normal que la configuración y la información de los procesos no estén en contacto con malware y las vulnerabilidades estén *parcheadas*, ya que se trata de un sistema operativo instalado en su última versión.

Conclusiones

En este último apartado me gustaría hacer una vista atrás a todo lo acontecido con el desarrollo del TFM, las principales lecciones aprendidas, la consecución, o no, de los objetivos propuestos, ámbitos de mejora y, por último, un breve pensamiento personal a modo de conclusión.

A lo largo del desarrollo del proyecto he aprendido bastante sobre el análisis forense, cómo funciona la memoria como parte de un sistema, un sinfín de vulnerabilidades y métodos de ataques que, en su conjunto, concluye en una lección primordial (la cual se podía intuir desde un primer momento), y no es otra que siempre estaremos desprotegidos, en mayor o menor medida dependiendo del empeño que le dediquemos.

Todos los avances que hay en los sistemas de información día tras día incurren también en un aumento de las posibilidades de descubrir nuevas vulnerabilidades y enfoques de ataques, que, sin una actuación a tiempo, pueden acabar en consecuencias graves. Y no sólo me refiero al ámbito de la “memoria” de los sistemas de información, si no en cualquier hardware, software, protocolos de red, sistemas de información y, sobre todo, en el factor humano (la concienciación y la información cobra mayor importancia frente a la ingeniería social).

Haciendo más hincapié en el ámbito de trabajo, saco una conclusión bastante clara. Gran parte del nivel de seguridad, de lo vulnerable o de lo fiable que los sistemas que están bajo nuestro cuidado es responsabilidad nuestra, y se ha podido comprobar a lo largo de este TFM. Los resultados obtenidos a lo largo del análisis del volcado de memoria nos devolvían resultados esperanzadores y, en general, buenos, con lo que podríamos deducir que nuestro sistema era un sistema limpio y seguro (en parte debido a que el sistema era nuevo y actualizado a fecha de realización del TFM). Sin embargo, no estoy totalmente seguro si los resultados que obtendría al realizar las mismas pruebas sobre un sistema real, por ejemplo, un ordenador personal de una empresa o un ordenador de un lugar público (biblioteca, institutos, etc.) serían igual de buenos, de hecho, estoy totalmente seguro de que encontraría problemas en cada aspecto auditado.

Es un hecho que el grado de disposición frente al exterior (internet, sobre todo) afecta directamente al grado de seguridad que tenemos en los ordenadores. Como también afecta la concienciación en la importancia de mantener las normas de seguridad y aptitudes frente a navegar por internet y, en general, a trabajar con cualquier sistema de información (posibles ataques, qué está permitido y qué no cuando navegamos por internet, seguridad en aplicaciones de correo electrónico y ofimáticas, etc.).

Respecto a los logros obtenidos, podemos concluir que se ha conseguido, en mayor parte, cumplir todos los objetivos propuestos. El objetivo primordial era realizar un análisis de un volcado de memoria con el fin de conseguir información útil de este para identificar vulnerabilidades o posibles errores, tanto a nivel de diseño como de implementación.

A lo largo del desarrollo del análisis hemos conseguido identificar los posibles focos de infección gracias a la implementación de las diferentes herramientas que Volatility nos ofrece, ya que cada herramienta nos devuelve información que nosotros, como ingenieros informáticos y utilizando fuentes de información, tratamos de descubrir indicios en diferentes secciones del volcado de memoria (procesos, direcciones de memoria, CPU, sistemas de ficheros, etc.), que concluyan en aspectos maliciosos del sistema. En caso de descubrir vulnerabilidades o errores de

implementación, se ha indicado posibles mejoras o soluciones, en caso de haberlas, en forma de parches o mejoras de configuración (sin especificar la implementación de ellas, que quedaría fuera del alcance de este TFM). Además, se ha conseguido cumplir los diferentes objetivos secundarios, por ejemplo, la adquisición de nuevos conocimientos referido a la seguridad informática, análisis forense, nuevos modos de ataques, herramientas de generación y análisis de volcados, nuevas vulnerabilidades que desconocía y, en general, he formado una base de conocimiento relacionada a la gestión de la memoria RAM, tratamiento y generación de volcados e identificación de vulnerabilidades que anteriormente no disponía. Respecto a los objetivos académicos, se ha conseguido realizar todas las entregas en su correspondiente plazo de entrega sin ningún inconveniente.

La declaración del estado de emergencia a nivel mundial y local debido a la pandemia COVID-19 no ha afectado directamente a la planificación temporal debido al carácter telemático del máster, por lo que la planificación en general ha sido adecuada y se ha seguido tal y como estaba planeado desde un inicio (no se ha visto afectada), estableciendo primeramente las bases de conocimiento (necesario para el entendimiento de las fases posteriores), luego identificando las diferentes herramientas para generar los volcados de memoria, seguidamente analizando las diferentes herramientas de análisis de imágenes con sus pros y sus contras para finalmente aplicar todos los conocimientos adquiridos a un caso práctico.

El único problema encontrado durante el desarrollo del TFM, que, precisamente el desconocimiento que tenía antes de iniciar el proyecto me impedía reconocerlo, es la incompatibilidad de la arquitectura del kernel de Raspberry con Volatility (debido a como los dispositivos Raspberry Pi establecen la estructura del kernel) [Preparando el entorno de trabajo.]. Esta problemática la identifiqué en el momento en el que iba a comenzar a realizar el análisis de memoria, cuando descubrí que necesitaba crear un perfil específico para adquirir la información del volcado de memoria. Esto afectó brevemente a la planificación del TFM. Inicialmente, se esperaba que el análisis fuese realizado sobre un volcado estándar de un sistema ARMv7, en este caso una Raspberry Pi (sistema real), cuyo sistema operativo sería Raspbian en su versión ARM. Este volcado de hecho lo recibí por parte del tutor para realizar el análisis.

Al ver que necesitaba realizar un perfil del kernel específico, intenté simular una Raspberry Pi en Windows utilizando QEMU, pero debido a que no era posible crear un perfil específico (Raspbian en su versión ARM utiliza ficheros en *C*, *overlays* y *blobs* para generar la estructura del kernel, en lugar del fichero *System.map*) me vi obligado a modificar la planificación. Al observar que Raspbian ofrecía una versión de escritorio (sin soporte ARM) decidí seguir dicha vertiente. El aspecto positivo es que la estructura del volcado de memoria es equivalente a la versión ARM, excepto en la definición del hardware, que, a los efectos de la finalidad de este TFM, no es relevante, ya que está centrado más en el análisis del volcado, con relación a la identificación de vulnerabilidades y aspectos de mejora.

Durante la fase de análisis se han descubierto diferentes vulnerabilidades a las cuales se ha indicado la solución, normalmente en forma de parches o actualizaciones, pero que la aplicación de estas no entra dentro de los objetivos del proyecto. Hubiese estado interesante indagar más en la corrección de vulnerabilidades y en la instalación de parches en los kernel Linux. Otro aspecto que durante el desarrollo del TFM he encontrado interesante como mejora adicional a este proyecto es infectar el sistema auditado con un malware conocido, preferiblemente un rootkit (por ejemplo, [KBeast](#)), para posteriormente, utilizando las diferentes herramientas vistas

en la sección “Análisis del volcado de memoria”, analizarlo tanto a nivel de procesos como a nivel de núcleo del kernel, red (backdoor, puertos abiertos), etc. Y, una vez analizado, establecer las diferentes directrices para eliminarlo del sistema y compararlo con el sistema limpio para observar las diferencias.

Y ya, para concluir, un breve pensamiento personal general sobre lo que me ha supuesto la redacción del TFM. Como se sabe, estamos en una época bastante compleja a nivel global debido a la situación de emergencia a causa de la pandemia COVID-19, en donde debemos de estar en estado de confinamiento. Nunca pensé que, gracias a estudiar telemáticamente en la UOC y en particular, al desarrollo de este proyecto en los días de confinamiento, me han supuesto una vía de escape y de dedicación que me han ayudado a pasar de una manera más llevadera la situación en la que nos encontramos. Verdaderamente he aprendido mucho a lo largo de estos meses, hasta tal punto que el interés, específicamente en el análisis forense, ha sobrepasado los aspectos meramente de análisis de memoria, llevándome a investigar y sentir curiosidad por otros aspectos del análisis forense y respuestas ante incidentes, llegándome a preguntar si debería de buscar certificarme en esos ámbitos de manera profesional.

También me gustaría dar las gracias al Sr. Víctor Méndez Muñoz, tutor del proyecto, que me ha tutorizado durante los meses de desarrollo del TFM, que amablemente me ha resuelto de manera ágil las dudas que me han ido surgiendo, se ha ofrecido a ayudarme siempre y cuando lo necesitase y, en general, agradecer la predisposición que ha tenido durante el curso. Por último, he acabado bastante contento con el resultado del proyecto y de los conocimientos adquiridos, con la única espina clavada de no tratar un volcado de memoria de un entorno real. Sin duda fue un acierto elegir esta vertiente de estudio para realizar el proyecto.

Glosario

A

ACPI

ACPI es la sigla del inglés de "Advanced Configuration and Power Interface" (Interfaz Avanzada de Configuración y Energía). Es un estándar resultado de la actualización de APM a nivel de hardware, que controla el funcionamiento del BIOS y proporciona mecanismos avanzados para la gestión y ahorro de la energía., 44

Análisis forense

El análisis forense digital se corresponde con un conjunto de técnicas destinadas a extraer información valiosa de discos, sin alterar el estado de los mismos. Esto permite buscar datos que son conocidos previamente, tratando de encontrar un patrón o comportamiento determinado, o descubrir información que se encontraba oculta., 10

ARM

ARM es una arquitectura de 32 bits desarrollada en 1983 por la empresa Acorn Computers Ltd para usarse en computadoras personales que maneja un sistema de instrucciones realmente simple lo que le permite ejecutar tareas con un mínimo consumo de energía., 36

B

BEC

Belkafost Evidence Center. Solución forense todo en uno, que ayuda en la extracción y análisis de los dispositivos móviles, computadoras, RAM, nubes y sistemas remotos en una sola herramienta., 34

D

DMA

Direct Memory Access o acceso directo a memoria, permite a cierto tipo de componentes de un ordenador acceder a la memoria del sistema para leer o escribir independientemente de la CPU., 57

F

FTK

Forensic Toolkit (Accessdata). Herramienta forense que recopila datos de cualquier dispositivo o sistema digital que produzca, transmita o almacene datos, 33

G

GUI

La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface), es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz., 37

I

IOMMU

Unidad de gestión de memoria de entrada y salida., 57

IoT

La internet de las cosas (IoT, por sus siglas en inglés) es un sistema de dispositivos de computación interrelacionados, máquinas mecánicas y digitales, objetos, animales o personas que tienen identificadores únicos y la capacidad de transferir datos a través de una red, sin requerir de interacciones humano a humano o humano a computadora., 11

ISO

Una imagen ISO es un archivo informático donde se almacena una copia o imagen exacta de un sistema de archivos., 81

K

Kernel

El núcleo de un sistema operativo, también llamado kernel, es el que realiza toda la comunicación segura entre software y hardware del ordenador. El kernel es lo más importante del sistema operativo Unix y de todos sus derivados, como Linux y las demás distribuciones que dependen de él., 18

L

LiME

Linux Memory Extractor. LiME es una herramienta desarrollada por 504ensics Labs, de código abierto, que permite la adquisición de la memoria volátil de sistemas Linux y dispositivos basados en Linux., 30

M

MP

Memoria principal. Es la memoria de la computadora donde se almacenan temporalmente tanto los datos como los programas que la unidad central de procesamiento., 22

P

PAE

En informática, extensión de dirección física (en inglés, Physical Address Extension o PAE) se refiere a una característica de los procesadores x86 que permite a los sistemas de 32-bit utilizar hasta 64 gigabytes (64 GB) de memoria física, suponiendo que el sistema operativo proporcione el adecuado soporte., 38

PID

En computación, PID es una abreviatura de process ID, o sea, ID del proceso o bien identificador de procesos. El identificador de procesos es un número entero usado por el kernel de algunos sistemas operativos (como el de Unix o el de Windows NT) para identificar un proceso de forma unívoca., 41

R

RAM

La memoria de acceso aleatorio (Random Access Memory, RAM) se utiliza como memoria de trabajo de computadoras y otros dispositivos para el sistema operativo, los programas y la mayor parte del software., 29

RISC

El término microprocesador RISC significa "Reduced Instruction Set Computer". Esto significa que los microprocesadores RISC utilizan un conjunto sencillo de instrucciones para leer y procesar los datos., 37

Rootkits

Rootkit es un conjunto de herramientas usadas frecuentemente por los intrusos informáticos o crackers que consiguen acceder ilícitamente a un sistema informático. Estas herramientas sirven para esconder los procesos y archivos que permiten al intruso mantener el acceso al sistema, a menudo con fines maliciosos., 16

S

SoC

Un sistema en chip o SoC (del inglés system on a chip o system on chip), describe la tendencia cada vez más frecuente de usar tecnologías de fabricación que integran todos o gran parte de los módulos que componen un computador o cualquier otro sistema informático o electrónico en un único circuito integrado o chip., 37

T

TCP

TCP (Transmission Control Protocol) Protocolo de Control de Transmisión.

Este protocolo se encarga de crear "conexiones" entre sí para que se cree un flujo de datos. Este proceso garantiza que los datos sean entregados en destino sin errores y en el mismo orden en el que salieron. También se utiliza para distinguir diferentes aplicaciones en un mismo dispositivo., 24

TMPFS

El sistema de archivos tmpfs es el nombre que recibe un sistema de almacenamiento en muchos sistemas operativos de tipo Unix, como memoria volátil., 49

TSK

The Sleuth Kit. Herramienta forense para el análisis de imágenes de discos., 34

U

UDP

El protocolo de datagramas de usuario (en inglés User Datagram Protocol o UDP) es un protocolo del nivel de transporte basado en el intercambio de datagramas (Encapsulado de capa 4 o de Transporte del Modelo OSI). Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera., 24

URL

Se conoce en informática como URL (siglas del inglés Uniform Resource Locator, es decir, Localizador Uniforme de Recursos) a la secuencia estándar de caracteres que identifica y permite localizar y recuperar una información determinada en la Internet, 17

Bibliografía.

- [1] “5 fases fundamentales del análisis forense digital”, We Live Security por Lucas Paus (15 Abr 2015). Fecha de consulta: 21-02-2020. Fuente: <https://www.welivesecurity.com/la-es/2015/04/15/5-fases-analisis-forense-digital/>
- [2] “Análisis de volcado de memoria en investigaciones forenses computacionales”, Universidad Nacional Autónoma de México (UNAM) por David Eduardo Bernal Michelena. Fecha de consulta: 21-02-2020. Fuente: <https://revista.seguridad.unam.mx/numero-17/an%C3%A1lisis-de-volcado-de-memoria-en-investigaciones-forenses-computacionales>
- [3] “Rootkit”, Wikipedia (17 dic 2019). Fecha de consulta: 23-02-2020. Fuente: https://es.wikipedia.org/wiki/Rootkit#Tipos_de_rootkits
- [4] “OSForensics, realiza un análisis forense a tu ordenador”, Proyecto TIC por Juan Carlos Valle Berbes en Ribadesella, Llanes – Asturias. Fecha de consulta: 23-02-2020. Fuente: <https://www.proyecto-tic.es/osforensics/>
- [5] “In memoria veritas”, Incibe-Cert, por Miguel Herrero (12/05/2016). Fecha de consulta: 25-02-2020. Fuente: <https://www.incibe-cert.es/blog/in-memoria-veritas>
- [6] “Investigación forense de dispositivos móviles Android”, Francisco Lázaro Domínguez. Fecha de consulta: 29-02-2020. Fuente: <https://books.google.es/books?id=LI6fDwAAQBAJ&pg>
- [7] “¿Qué es el análisis forense informático?”, Praxmatic. Fecha de consulta: 04-03-2020. Fuente: <https://www.praxmatic.com/seguridad-ti/que-es-el-analisis-forense-informatico/>
- [8] “Cómputo forense”, Wikipedia. Fecha de consulta: 04-03-2020. Fuente: https://es.wikipedia.org/wiki/C%C3%B3mputo_forense
- [9] “Introducción: la informática forense, una disciplina técnico-legal”. Computación forense : descubriendo los rastros informáticos por José, Cano Martínez, Jeimy (2015). Fecha de consulta: 04-03-2020. Fuente: <https://www.worldcat.org/title/computacion-forense-descubriendo-los-rastros-informaticos/oclc/945459757>
- [10] “Memoria principal”, Wikipedia. Fecha de consulta: 04-03-2020. Fuente: https://es.wikipedia.org/wiki/Memoria_principal
- [11] “Memoria secundaria”, Wikipedia. Fecha de consulta: 04-03-2020. Fuente: https://es.wikipedia.org/wiki/Memoria_secundaria
- [12] “Dispositivo de almacenamiento de datos”, Wikipedia. Fecha de consulta: 04-03-2020. Fuente: https://es.wikipedia.org/wiki/Dispositivo_de_almacenamiento_de_datos
- [13] “Soporte de almacenamiento de datos”, Wikipedia. Fecha de consulta: 04-03-2020. Fuente: https://es.wikipedia.org/wiki/Soporte_de_almacenamiento_de_datos
- [14] “El núcleo Linux” por Josep Jorba Esteve (UOC). Fecha de consulta: 04-03-2020. Fuente: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/61265/1/Administraci%C3%B3n%20avanzada%20del%20sistema%20operativo%20GNU_Linux_M%C3%B3dulo1_El%20n%C3%BAcleo%20Linux.pdf
- [15] “Volcado de memoria”, Wikipedia. Fecha de consulta: 05-03-2020. Fuente: https://es.wikipedia.org/wiki/Volcado_de_memoria
- [16] “About The Volatility Foundation”, Volatility Foundation. Fecha de consulta: 05-03-2020. Fuente: <https://www.volatilityfoundation.org/about>
- [17] “Análisis forense con volatility”, Byte Mind (27 - feb - 2020). Fecha de consulta: 05-03-2020. Fuente: <https://byte-mind.net/analisis-forense-con-volatility/>
- [18] “Crear la Imagen Forense desde una Unidad utilizando FTK Imager”, Reydes (2014). Fecha de consulta: 10-03-2020. Fuente:

- <http://www.reydes.com/d/?q=Crear la Imagen Forense desde una Unidad utilizando FTK Imager>
- [19] "Volcado de memoria #RAM en #Linux – #LiME", fwhibbit.es por Marcos (2016). Fecha de consulta: 11-03-2020. Fuente: <https://fwhibbit.es/volcado-de-memoria-ram-en-linux-lime>
- [20] "Top 8 Tools To Search Memory Under Linux / Unix [Forensics Analysis]" NixCraft. Fecha de consulta: 11-03-2020. Fuente: <https://www.cyberciti.biz/programming/linux-memory-forensics-analysis-tools/>
- [21] "Redline FireEye Manual", FireEye. Fecha de consulta: 12-03-2020. Fuente: <https://www.fireeye.com/content/dam/fireeye-www/services/freeware/uq-redline.pdf>
- [22] "Accessdata FTK Forensic Toolkit", OnData. Fecha de consulta: 12-03-2020. Fuente: <https://www.ondata.es/recuperar/ftk-forensic-toolkit.htm>
- [23] "Belkafost Evidence Center", Belkafost. Fecha de consulta: 12-03-2020. Fuente: <https://belkasoft.com/es/new>
- [24] "Qué es un proceso informático y qué función tiene", ProfesionalReview por José Antonio Castillo (2019). Fecha de consulta: 12-03-2020. Fuente: <https://www.profesionalreview.com/2019/09/23/proceso-informatico/>
- [25] "El boom de los ataques de malware sin fichero: ¿cómo combatirlo?", Panda security. Fecha de consulta: 20-03-2020. Fuente: <https://www.pandasecurity.com/spain/mediacenter/seguridad/boom-ataques-fileless-malware/>
- [26] "IoT - Internet Of Things" en Deloitte por María Gracia. Fecha de consulta: 22-03-2020. Fuente: <https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html>
- [27] "Computación distribuida", Wikipedia. Fecha de consulta: 22-03-2020. Fuente: https://es.wikipedia.org/wiki/Computaci%C3%B3n_distribuida
- [28] "Acquisition and analysis of volatile memory from android devices" por Joe Sylve y Andrew Caseb. Fecha de consulta: 22-03-2020. Fuente: https://www.researchgate.net/publication/257687708_Acquisition_and_analysis_of_volatile_memory_from_android_devices
- [29] "The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux and MAC memory" por Michael Hale Ligh, Andrew Case y Jamie Levy. Fecha de consulta: 22-03-2020. Fuente: <https://www.wiley.com/en-us/The+Art+of+Memory+Forensics%3A+Detecting+Malware+and+Threats+in+Windows%2C+Linux%2C+and+Mac+Memory-p-9781118825099>
- [30] "A Review and Analysis of Ransomware Using Memory Forensics and Its Tools" por D. Paul Joseph y Jasmine Norman. Fecha de consulta: 22-03-2020. Fuente: https://link.springer.com/chapter/10.1007/978-981-13-9282-5_48
- [31] "Comparative Analysis of Volatile Memory Forensics: Live Response vs. Memory Imaging" por Amer Aljaedi, Dale Lindskog y Pavol Zavarisky. Fecha de consulta: 22-03-2020. Fuente: <https://ieeexplore.ieee.org/abstract/document/6113291>
- [32] "Simplifying RAM Forensics: A GUI and Extensions for the Volatility Framework" por Steffen Logen, Hans Höfken y Marko Schuba. Fecha de consulta: 22-03-2020. Fuente: <https://ieeexplore.ieee.org/abstract/document/6329239>
- [33] "QEMU", Wikipedia. Fecha de consulta: 12-04-2020. Fuente: <https://es.wikipedia.org/wiki/QEMU>
- [34] "Raspbian "stretch" for Raspberry Pi 3 on QEMU" GitHub por Wim Vanderbauwhede (10 Feb 2019). Fecha de consulta: 13-04-2020. Fuente: <https://github.com/wimvanderbauwhede/limited-systems/wiki/Raspbian-%22stretch%22-for-Raspberry-Pi-3-on-QEMU>

- [35] "Creating a new profile" Volatility Foundation. Fecha de consulta: 13-04-2020. Fuente: <https://github.com/volatilityfoundation/volatility/wiki/Linux>
- [36] "Memory Forensics Investigation using Volatility (Part 1)" por Raj Chandel en Hacking Articles. Fecha de consulta: : 14-04-2020. Fuente: <https://www.hackingarticles.in/memory-forensics-investigation-using-volatility-part-1/>
- [37] "Linux Command Reference", Volatility Foundation, GitHub. Fecha de consulta: 17-04-2020. Fuente: <https://github.com/volatilityfoundation/volatility/wiki/Linux-Command-Reference>
- [38] "systemd". Wikipedia. Fecha de consulta: 17-04-2020. Fuente: <https://es.wikipedia.org/wiki/Systemd>
- [39] "kthreadd". Fecha de consulta: 18-04-2020. Fuente: <http://ferranserafini.blogspot.com/2011/11/linux-los-procesos-del-kernel.html>
- [40] "Linux insmod command", Computer Hope. Fecha de consulta: 18-04-2020. Fuente: <https://www.computerhope.com/unix/insmod.htm>
- [41] "Loadable Kernel Module", Wikipedia. Fecha de consulta: 19-04-2020. Fuente: https://en.wikipedia.org/wiki/Loadable_kernel_module
- [42] "tmpfs", Wikipedia. Fecha de consulta: 19-04-2020. Fuente: https://en.wikipedia.org/wiki/Loadable_kernel_module
- [43] "Tarjeta de red", Wikipedia. Fecha de consulta: 19-04-2020. Fuente: https://es.wikipedia.org/wiki/Tarjeta_de_red
- [44] "Cortafuegos", Wikipedia. . Fecha de consulta: 19-04-2020. Fuente: [https://es.wikipedia.org/wiki/Cortafuegos_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Cortafuegos_(inform%C3%A1tica))
- [45] "Loopback", Wikipedia. Fecha de consulta: 19-04-2020. Fuente: <https://es.wikipedia.org/wiki/Loopback>
- [46] "ARP", IONOS (04.09.19). Fecha de consulta: 19-04-2020. Fuente: <https://www.ionos.es/digitalguide/servidores/know-how/arp-resolucion-de-direcciones-en-la-red/>
- [47] "ARP Spoofing", Wikipedia.). Fecha de consulta: 19-04-2020. Fuente: https://es.wikipedia.org/wiki/Suplantaci%C3%B3n_de_ARP
- [48] "Vulnerabilidades Meltdown y Spectre" por José Montes (16 mayo, 2019). Fecha de consulta: 19-04-2020. Fuente: <https://www.muycomputer.com/2019/05/16/nuevas-vulnerabilidades-en-procesadores-intel/>
- [49] "IOMMU", Wikipedia. Fecha de consulta: 19-04-2020. Fuente: https://es.wikipedia.org/wiki/Unidad_de_gesti%C3%B3n_de_memoria_de_entrada/salida
- [50] "ROM", Wikipedia. Fecha de consulta: 19-04-2020. Fuente: https://es.wikipedia.org/wiki/Memoria_de_solo_lectura
- [51] "PCI BUS", Wikipedia. Fecha de consulta: 19-04-2020. Fuente: https://es.wikipedia.org/wiki/Peripheral_Component_Interconnect
- [52] "ACPI", Wikipedia. Fecha de consulta: 19-04-2020. Fuente: https://es.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface
- [53] "Las 3 tecnologías clave para el Internet de las cosas", Xataka por Pablo Espeso. Fecha de consulta: 21-04-2020. Fuente: <https://www.xataka.com/internet-of-things/las-3-tecnologias-clave-para-el-internet-de-las-cosas>
- [54] "CVE", Wikipedia. Fecha de consulta: 21-04-2020. Fuente: https://es.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures

Anexos

Sistema operativo utilizado

Con la herramienta *hostnamectl* de Linux, podemos acceder a la información del sistema operativo utilizado:

```
root@osboxes:~# hostnamectl
  Static hostname: osboxes
            Icon name: computer-vm
            Chassis: vm
    Machine ID: 2f021f4d22e34f8a8538398f99615f7d
       Boot ID: 608ab18dee2e4a7ab77306c6969583b8
  Virtualization: oracle
 Operating System: Kali GNU/Linux Rolling
            Kernel: Linux 5.2.0-kali2-amd64
   Architecture: x86_64
root@osboxes:~# date
Thu 05 Mar 2020 03:01:27 PM EST
```

Ilustración 8. Funcionamiento de hostnamectl.

Instalación de FTK Imager y ejemplo de uso.

Para descargar el software tan solo debemos de acceder a la [página de descarga de AccessData](#) y rellenar el formulario de descarga que nos presentan. Posteriormente recibiremos un correo con el enlace de descarga de la aplicación listo para instalar.

Una vez instalado ya podemos acceder a la interfaz para proceder a realizar el volcado de memoria. Para realizar el volcado de memoria tan sólo debemos de apretar el botón “*Capture memory*” [Ilustración 9. FTK Imager, realizando el volcado.]. Seguidamente tan sólo debemos de indicar la ruta donde deseamos guardar el volcado y el nombre que tendrá este. El proceso puede durar unos minutos ya que se trata de un volcado completo (de varios GB de memoria).

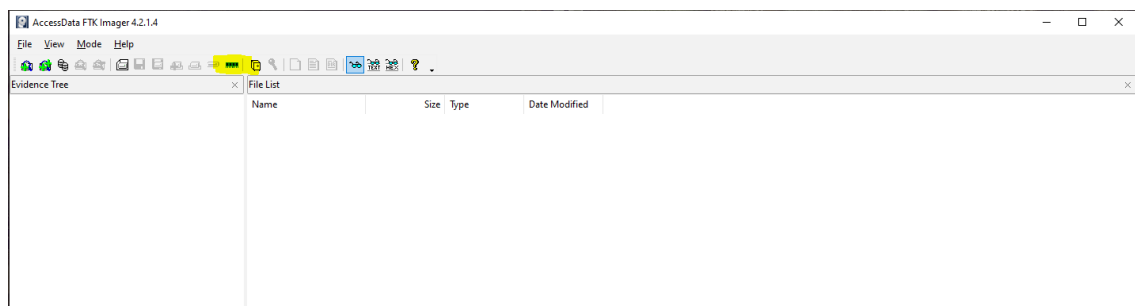
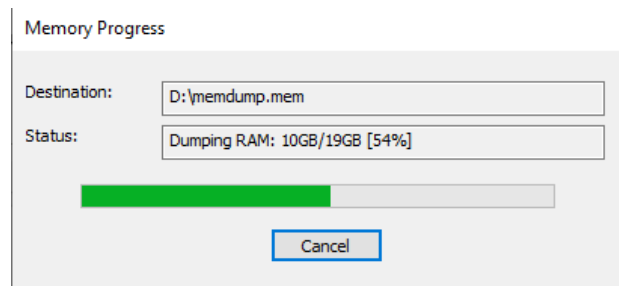


Ilustración 9. FTK Imager, realizando el volcado.



Una vez finalizado, dispondremos del fichero “*memdump.mem*” listo para analizar.

Instalación de LiME

1. Clonar el directorio oficial de la aplicación:
 - a. `git clone https://github.com/504ensicsLabs/LiME.git`
2. Posteriormente será necesario instalar los paquetes que necesita LiME para funcionar. LiME precisa de tres paquetes, (*make*, *build-essential* y *linux-headers*). Los instalamos mediante:
 - a. `apt-get install make build-essential linux-headers` (como usuario *root*)

```
root@osboxes:~/Desktop# git clone https://github.com/504ensicsLabs/LiME.git
Cloning into 'LiME'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 298 (delta 0), reused 3 (delta 0), pack-reused 292
Receiving objects: 100% (298/298), 1.60 MiB | 2.84 MiB/s, done.
Resolving deltas: 100% (151/151), done.
root@osboxes:~/Desktop# apt-get install make build-essential linux-headers
```

Ilustración 10. Clonando el repositorio de LiME.

3. Con la instalación de *linux-headers* será necesario conocer qué versión del kernel estamos usando. Para conocer este valor se podrá utilizar: `uname -r`
4. Por último, tan sólo debemos de compilar el código fuente y generar los módulos. Para ello nos dirigimos al directorio con el código fuente y ejecutamos lo siguiente:
 - a. `cd LiME/src`
 - b. `make`
5. Este comando nos generará un archivo con el siguiente formato: ‘*lime-versióndelkernel.ko*’. En mi caso particular (depende del kernel donde se compile) se generó el fichero *lime-5.2.0-kali2-amd64.ko*.
6. Una vez concluido los pasos anteriores, ya podríamos generar el volcado de memoria de manera local. Para ello, hacemos la llamada a la aplicación mediante la siguiente sentencia:
 - a. `insmod ./lime-5.2.0-kali2-amd64.ko "path=/root/Desktop format=lime"`
 - b. Es importante disponer de permisos *root* y modificar el archivo *lime-5.2.0-kali2-amd64.ko* por el generado al compilar ya que puede ser diferente si se ejecuta en otro kernel.

Instalación de Volatilitux.

1. Clonar el directorio oficial de la aplicación:
 - a. `git clone https://github.com/bashok001/volatilitux.git`
2. No es necesario ninguna instalación, tan solo accedemos al directorio y podremos ejecutar el fichero Python de la siguiente manera (utilizaremos el comando `memdump` para generar el volcado):
 - a. `volatilitux.py -f <dumpfile> [-c <configfile>] [-o] [-d] <command> [options]`

Instalación de Volatility

1. Para la instalación de Volatility, necesitaremos en primer lugar disponer de Python en la versión 2.7 o superior instalado en nuestro sistema. Para saber la versión instalada, usaremos: `python3 --version`:

```
root@osboxes:~# python3 --version
Python 3.7.4
```

Ilustración 11. Versión de python instalada.

2. En caso de no disponer de ella, se procederá mediante (necesarios permisos root): `apt-get install python3.1`
3. Descarga del [repositorio oficial en GitHub](https://github.com/volatilityfoundation/volatility) con el siguiente comando:
 - a. `git clone https://github.com/volatilityfoundation/volatility.git`

```
root@osboxes:~/Desktop# git clone https://github.com/volatilityfoundation/volatility
Cloning into 'volatility'...
remote: Enumerating objects: 68, done.
remote: Counting objects: 100% (68/68), done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 27289 (delta 29), reused 51 (delta 25), pack-reused 27221
Receiving objects: 100% (27289/27289), 20.91 MiB | 12.19 MiB/s, done.
Resolving deltas: 100% (19642/19642), done.
```

Ilustración 12. Clonando el repositorio de Volatility.

4. Completado este paso tendríamos dos opciones, instalar el software en nuestro sistema o ejecutar directamente el fichero `.py` disponible en dicho código descargado:
 - a. En caso de desear instalar el software es tan sencillo como ejecutar el fichero `setup.py`: `python setup.py install` (con permisos de administrador). Este comando se encargará de instalar las dependencias necesarias, así como los propios paquetes de Volatility, creando a su vez también un fichero ejecutable en la ruta `/usr/bin/volatility`.

```
root@osboxes:~/Desktop# cd volatility/
root@osboxes:~/Desktop/volatility# python setup.py install
running install
running bdist_egg
running egg_info
creating volatility.egg-info
writing volatility.egg-info/PKG-INFO
writing top-level names to volatility.egg-info/top_level.txt
```

Ilustración 13. Instalación de Volatility

- b. Si la opción no es instalarla, podemos ejecutar el software directamente con el siguiente comando:
 - i. `python vol.py [options]`

Ejemplos de uso de Volatility

Para comprobar que la herramienta está correctamente instalada, procederemos a utilizar alguna de las funciones que Volatility aporta [17].

Para probar alguna funcionalidad básica de Volatility es necesario disponer algún volcado de memoria. Para generar un volcado de memoria se puede hacer uso de las herramientas del apartado denominado “Herramientas para generar volcados de memoria.” [29] o descargar por internet algún volcado de memoria de ejemplo. En mi caso utilizaré para las pruebas un [volcado de memoria](#) de un Windows Vista. Para observar qué opciones y funcionalidades nos aporta Volatility haremos uso de: `volatility --help`.

La primera prueba consistirá en comprobar la información básica de la imagen/volcado, tales como el sistema operativo, fecha de generación de la memoria, etc. Para ello, Volatility nos ofrece un plugin (entre muchos otros) llamado *imageinfo*, que precisamente nos permite ver dicha información. Para ello ejecutaremos el siguiente comando:

- `volatility -f memdump.mem imageinfo`

Con la opción “-f” establecemos la imagen a tratar e *imageinfo* es el plugin que ejecutaremos, este específicamente nos devolverá información básica del volcado.

```
root@osboxes:~/Downloads# volatility -f memdump.mem imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO : volatility.debug : Determining profile based on KDBG search...
      Suggested Profile(s) : VistaSP1x86, Win2008SP1x86, Win2008SP2x86, VistaSP2x86
      AS Layer1 : IA32PagedMemoryPae (Kernel AS)
      AS Layer2 : FileAddressSpace (/root/Downloads/memdump.mem)
      PAE type : PAE
      DTB : 0x122000L
      KDBG : 0x81931c90L
      Number of Processors : 1
      Image Type (Service Pack) : 1
      KPCR for CPU 0 : 0x81932800L
      KUSER_SHARED_DATA : 0xffdf0000L
      Image date and time : 2014-01-08 17:54:20 UTC+0000
      Image local date and time : 2014-01-08 09:54:20 -0800
```

Ilustración 14. Volatility imageinfo plugin

Dicho plugin nos devuelve información esencial que puede ser interesante, como la fecha en la que se tomó la imagen, el sistema operativo de la máquina de la cual se obtuvo la imagen, número de procesadores, etc.

Ahora que disponemos del tipo de sistema (*Suggested Profile(s) : VistaSP1x86, Win2008SP1x86, Win2008SP2x86, VistaSP2x86*) el siguiente paso natural es conocer los procesos que se estaba ejecutando en el momento del volcado.

Establecemos en este caso el tipo de sistema con el parámetro *profile* e indicamos que el plugin a ejecutar es *pslist* (“Process list”), el cual nos devolverá la lista de los procesos. Nos generará

TÉCNICAS Y HERRAMIENTAS PARA EL ANÁLISIS DE DEBILIDADES EN VOLCADOS DE MEMORIA RAM DE SISTEMAS BASADOS EN LINUX

una salida similar a la siguiente (salida completa disponible en el anexo “PSLIST Plugin Volatility”). Para ello ejecutamos la siguiente sentencia:

- `volatility -f memdump.mem --profile=VistaSP1x86 pslist`

```
root@osboxes:~/Downloads# volatility -f memdump.mem --profile=VistaSP1x86 pslist
Volatility Foundation Volatility Framework 2.6.1
Offset(V)  Name                PID  PPID  Thds  Hnds  Sess  Wow64  Start                                Exit
-----
0x82db0910 System                4    0    100   541   -----  0  2014-01-08 02:17:35 UTC+0000
0x8454c118 smss.exe          404   4    4    28   -----  0  2014-01-08 02:17:35 UTC+0000
0x84561968 csrss.exe          472  460   11   466   0  2014-01-08 02:17:35 UTC+0000
0x84450770 csrss.exe          516  508   10   305   1  2014-01-08 02:17:36 UTC+0000
0x84453770 wininit.exe       524  460    3    98   0  2014-01-08 02:17:36 UTC+0000
0x84465770 winlogon.exe     552  508    3   116   1  2014-01-08 02:17:36 UTC+0000
0x83632170 services.exe       604  524    6   250   0  2014-01-08 02:17:36 UTC+0000
0x844bf770 lsass.exe          616  524   13   610   0  2014-01-08 02:17:36 UTC+0000
0x844c2680 lsm.exe              624  524   10   208   0  2014-01-08 02:17:36 UTC+0000
```

Ilustración 15. Volatility pslist plugin

En la lista que nos devuelve podemos ver una gran cantidad de información útil respecto a los procesos en memoria. Por ejemplo, podemos observar en la siguiente imagen [Ilustración 16] que el software utilizado para generar el volcado fue FTK Imager (visto en la sección “Herramientas para generar volcados de memoria” [29]):

```
0x84c73b60 svchost.exe      3224  604    9   228   0  2014-01-08 02:19:53 UTC+0000
0x84ce3020 iashost.exe      3336  788    2    97   0  2014-01-08 02:19:53 UTC+0000
0x84bd8020 wuauclt.exe     3680 1000    2   139   1  2014-01-08 02:20:55 UTC+0000
0x84c64a50 notepad.exe     3920 2496    1    51   1  2014-01-08 03:19:07 UTC+0000
0x84cfd958 FTK Imager.exe  1800 2496    5   251   1  2014-01-08 03:19:32 UTC+0000
0x848ab618 iexplore.exe   1888 2496   14   641   1  2014-01-08 03:20:24 UTC+0000
0x848a1340 notepad.exe     2708 2496    1    45   1  2014-01-08 17:33:08 UTC+0000
```

Ilustración 16. FTK Imager para la generación del volcado

Existe una gran cantidad de comandos y opciones que el framework Volatility nos ofrece y con los cuales podemos acceder a una gran cantidad de información.

PSLIST Plugin Volatility

```
root@osboxes:~/Downloads# volatility -f memdump.mem --profile=VistaSP1x86 pslist
Volatility Foundation Volatility Framework 2.6.1
Offset(V)  Name                PID  PPID  Thds  Hnds  Sess  Wow64  Start                                Exit
-----
0x82db0910 System                4    0    100   541   -----  0  2014-01-08 02:17:35 UTC+0000
0x8454c118 smss.exe          404   4    4    28   -----  0  2014-01-08 02:17:35 UTC+0000
0x84561968 csrss.exe          472  460   11   466   0  2014-01-08 02:17:35 UTC+0000
0x84450770 csrss.exe          516  508   10   305   1  2014-01-08 02:17:36 UTC+0000
0x84453770 wininit.exe       524  460    3    98   0  2014-01-08 02:17:36 UTC+0000
0x84465770 winlogon.exe     552  508    3   116   1  2014-01-08 02:17:36 UTC+0000
0x83632170 services.exe       604  524    6   250   0  2014-01-08 02:17:36 UTC+0000
0x844bf770 lsass.exe          616  524   13   610   0  2014-01-08 02:17:36 UTC+0000
0x844c2680 lsm.exe              624  524   10   208   0  2014-01-08 02:17:36 UTC+0000
0x84866d50 svchost.exe      788  604    6   298   0  2014-01-08 02:17:42 UTC+0000
0x845f37a8 svchost.exe      848  604    8   280   0  2014-01-08 02:17:42 UTC+0000
0x848fa118 svchost.exe      884  604   15   274   0  2014-01-08 02:17:42 UTC+0000
0x84914d90 svchost.exe      976  604    6   152   0  2014-01-08 02:17:42 UTC+0000
0x8491bd90 svchost.exe     1000  604   45  2072   0  2014-01-08 02:17:42 UTC+0000
0x8492a6d0 SLsvc.exe       1056  604    4    96   0  2014-01-08 02:17:42 UTC+0000
0x84937d90 svchost.exe     1088  604   17   567   0  2014-01-08 02:17:42 UTC+0000
0x84941d90 svchost.exe     1160  604   20   265   0  2014-01-08 02:17:42 UTC+0000
0x84945c30 svchost.exe     1188  604   22   596   0  2014-01-08 02:17:42 UTC+0000
0x8496e9f0 svchost.exe     1308  604   17   265   0  2014-01-08 02:17:43 UTC+0000
0x849c18a8 spoolsv.exe     1424  604   17   291   0  2014-01-08 02:17:49 UTC+0000
0x849d7618 armsvc.exe     1460  604    2    56   0  2014-01-08 02:17:49 UTC+0000
0x849dcd90 dns.exe        1480  604   10   164   0  2014-01-08 02:17:49 UTC+0000
0x849e1cc0 ftpbasicsvr.exe 1508  604    2    52   0  2014-01-08 02:17:49 UTC+0000
0x849f5888 svchost.exe     1576  604    5   124   0  2014-01-08 02:17:49 UTC+0000
0x849faad8 svchost.exe     1604  604    3    73   0  2014-01-08 02:17:49 UTC+0000
0x849f7380 snmp.exe       1620  604    4   147   0  2014-01-08 02:17:49 UTC+0000
0x84a0b020 vmtoolsd.exe     1640  604    7   273   0  2014-01-08 02:17:49 UTC+0000
0x84a21d90 svchost.exe     1696  604    4    44   0  2014-01-08 02:17:49 UTC+0000
0x83060608 TPAutoConnSvc.e    832  604    9   136   0  2014-01-08 02:17:52 UTC+0000
0x84ba8d90 dlhosh.exe     1924  604   13   240   0  2014-01-08 02:17:53 UTC+0000
0x84bb7ca8 msdtc.exe      1572  604   11   167   0  2014-01-08 02:17:53 UTC+0000
0x84ba1c30 taskeng.exe    2096 1000    5   137   0  2014-01-08 02:17:53 UTC+0000
0x84c13020 taskeng.exe    2352 1000   10   250   1  2014-01-08 02:18:17 UTC+0000
0x84c148e8 userinit.exe    2368  552    0   -----  1  2014-01-08 02:18:17 UTC+0000
0x84c148e8 userinit.exe    2368  552    0   -----  1  2014-01-08 02:18:17 UTC+0000
```

Ilustración 17. PSLIST Plugin de Volatility

Redline de FireEye, instalación y ejemplo de uso.

Pasos de instalación

Para descargar el software hay que acceder a la [página oficial de la compañía FireEye](#) y rellenar el formulario que se nos presenta para obtener el enlace de descarga.

Una vez tengamos a nuestra disposición el fichero .zip, tan sólo debemos de descomprimirlo y ejecutar el fichero *Redline.msi*. Navegamos por el cuadro de diálogo dejando todos los valores por defecto. También se nos pedirá aceptar las condiciones de uso.

Ejemplo de uso

Para el ejemplo de uso necesitamos nuevamente de una imagen/volcado de memoria. Para la prueba usaré nuevamente la imagen del Windows Vista analizada en el apartado “Ejemplos de uso de Volatility”. También es importante resaltar que la aplicación Redline nos permite crear imágenes de memoria. Para acceder a esta funcionalidad tan sólo debemos de clicar sobre “Create Standard Collector” [Ilustración 18. Principales funcionalidades de Redline.].

Abrimos la aplicación y seleccionamos la opción “Analyze Data from a Saved Memory File” [Ilustración 18. Principales funcionalidades de Redline.] .

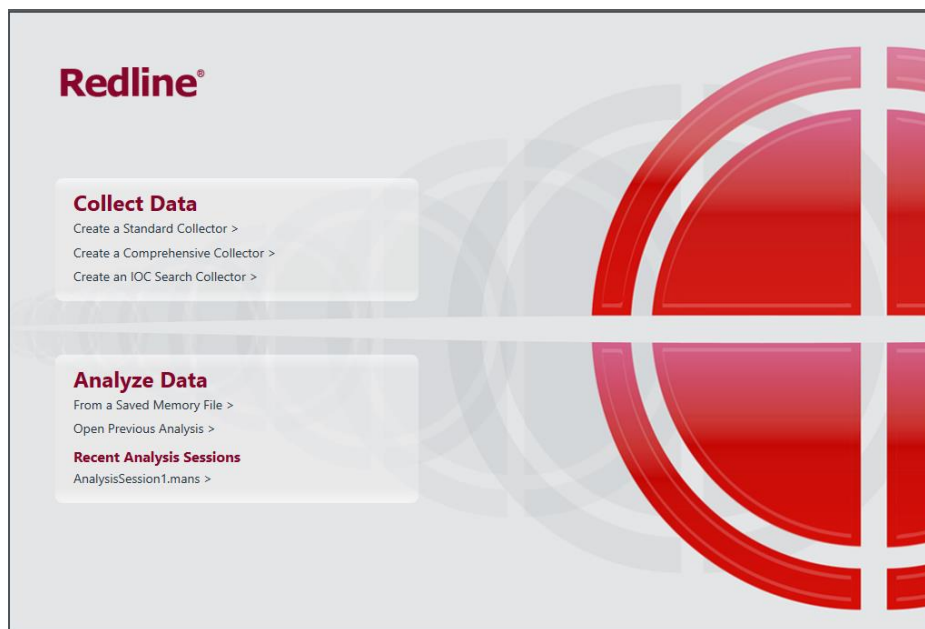


Ilustración 18. Principales funcionalidades de Redline.

Una vez accedido al panel siguiente, tan sólo debemos de indicar la ruta al fichero donde se encuentra el volcado de memoria y clicamos en “siguiente”. En la siguiente pestaña tan sólo debemos de indicar un nombre a la sesión que se generará a la hora de realizar el análisis y la

TÉCNICAS Y HERRAMIENTAS PARA EL ANÁLISIS DE DEBILIDADES EN VOLCADOS DE MEMORIA RAM DE SISTEMAS BASADOS EN LINUX

ruta donde se guardará el resultado de dicho análisis. Una vez indicado estos parámetros ya estaría todo listo para ejecutar el estudio del volcado. Dicho análisis tardará unos minutos en generarnos los resultados.

Una vez termine el proceso nos aparecerá por pantalla una serie de opciones para filtrar los resultados según el tipo de investigación que estamos desarrollando:

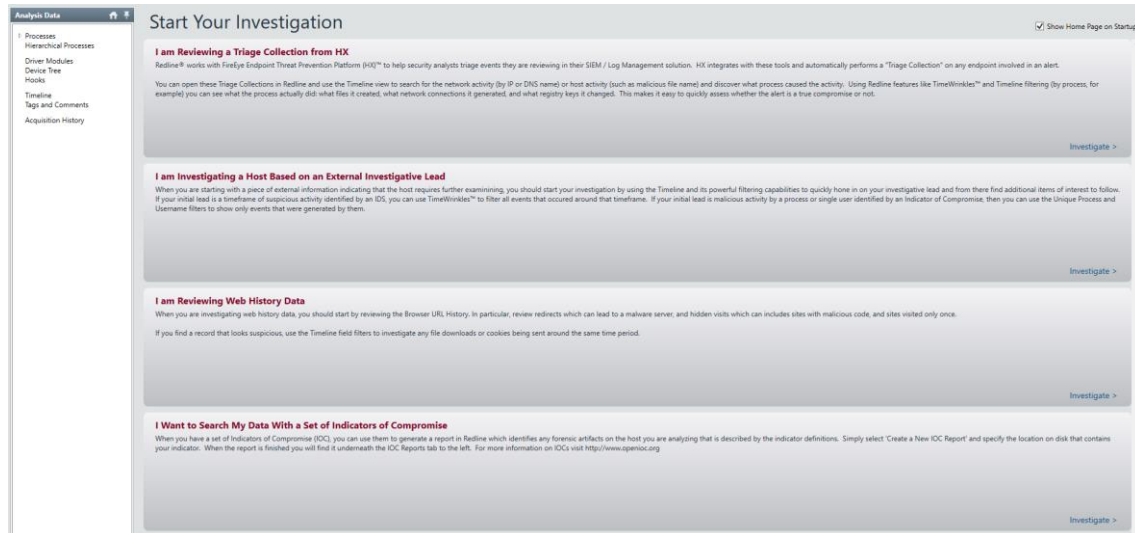


Ilustración 19. Perfiles de investigación de Redline

Depende de la opción que elijamos (en caso de elegir una) la información se nos presentará filtrada y ordenada de una manera u otra [Ilustración 22], por ejemplo, mostrando puertos, servicios, registros, etc. También se nos ordenará los resultados de los procesos mediante fechas de creación, peligrosidad en caso de encontrarse algún proceso malicioso, tamaño de memoria, etc. En el caso de querer acceder a la información desglosada sin ningún tipo de filtro, tan solo se debe de acceder al panel de la izquierda y seleccionar los atributos que se quieran observar [Ilustración 21].

Tal y como se hizo en Volatility es posible acceder a la información de todos los procesos tan sólo clicando en la pestaña “Processes”:

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

File Edit View Window Help

Ilustración 20. Procesos en Redline

En dicha ventana se podrá observar una infinidad de atributos e información importante como direcciones IP, puertos, rutas, PID, etc. Seleccionando un proceso nos aparecerá toda la información de este, así como unas pestañas adicionales que nos ofrecerá más información sobre el proceso separada adecuadamente para que el usuario de manera sencilla pueda filtrar la información que requiera [Ilustración 23]. En el anexo estará disponible la salida completa del resultado del análisis [Resultados del análisis de prueba utilizando Redline de FireEye.].

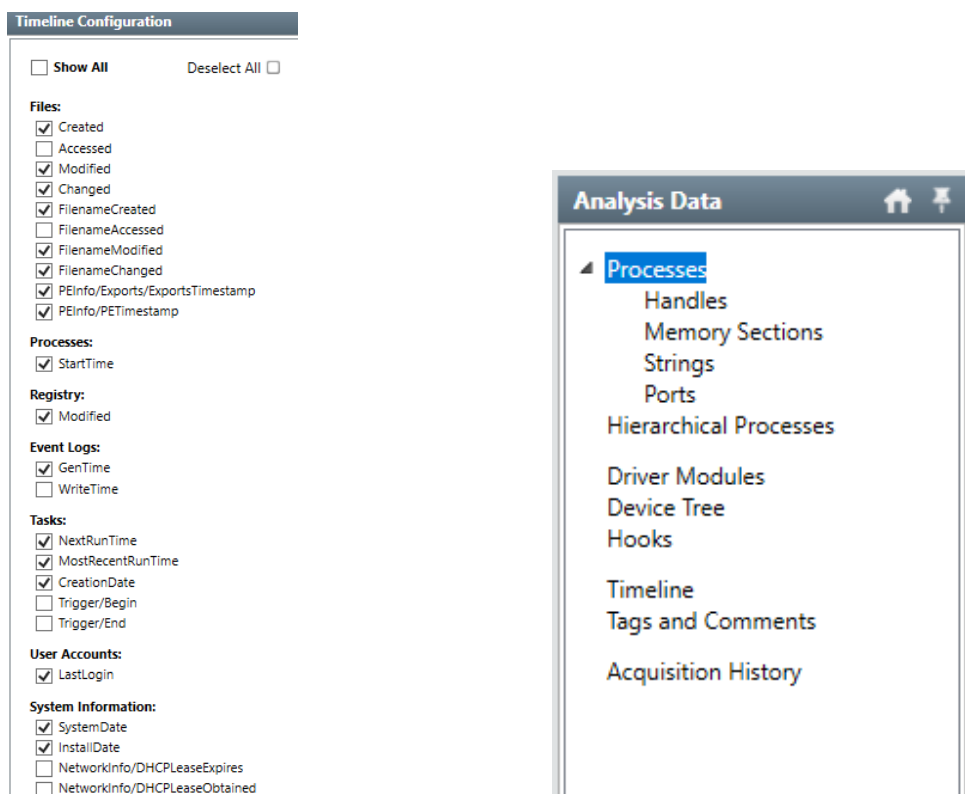


Ilustración 21. Panel de resultados del análisis

Ilustración 22. Filtros de Redline

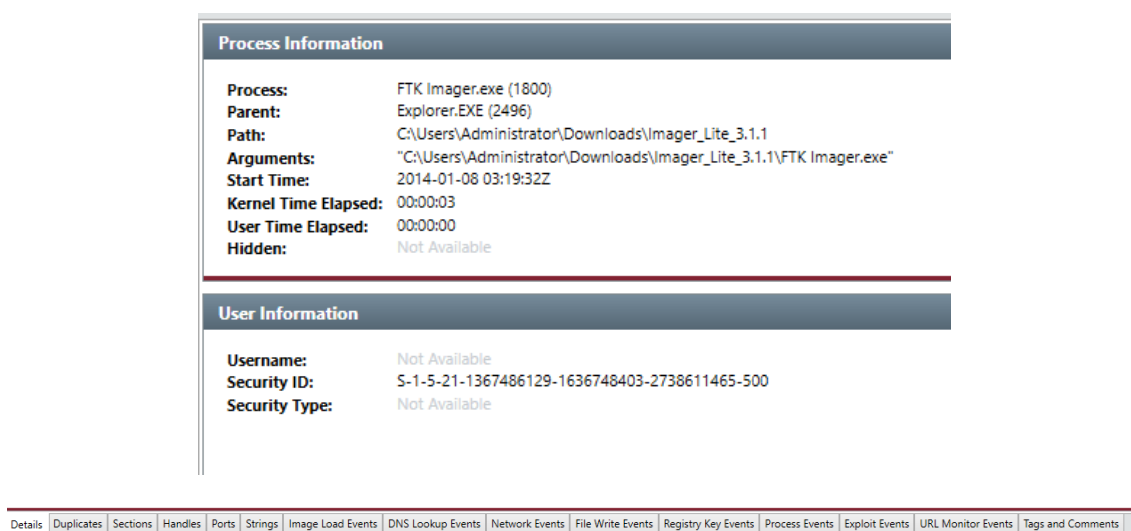


Ilustración 23. Detalles y pestañas asociadas a un proceso

Resultados del análisis de prueba utilizando Redline de FireEye.

En el siguiente enlace estará disponible la salida completa del resultado del análisis del volcado de memoria de prueba utilizando la herramienta Redline (disponible para la comunidad UOC):

- https://drive.google.com/file/d/1knYGUxNzUI_JCLqZof1IXPT2YZL04rxm/view?usp=sharing

Simulando arquitectura ARM con el sistema operativo Raspbian y QEMU

Para simular un sistema Raspbian con arquitectura ARM en Windows utilizando para ello el simulador de procesadores y virtualización QEMU tan sólo hay que seguir los siguientes pasos [34]:

1. Primeramente, descargaremos QEMU desde la [página web de la organización](#). Una vez ahí descargamos la versión de QEMU en base al sistema operativo y arquitectura de procesador. Para este caso en concreto descargaré la versión de Windows x64. Una vez descargado el ejecutable tan sólo se deberá de seguir el proceso de instalación a través del entorno de instalación y dejando los valores por defecto (importante la ubicación donde se instalará la aplicación, ya que desde ahí se ejecutará la simulación, aunque se podría crear una variable de entorno para acceder a los ejecutables en todo el sistema).
2. Una vez descargado el emulador necesitamos descargar la imagen del sistema operativo que queramos simular. En este caso, descargaremos la última versión del sistema operativo que esté disponible, se trata de [Raspbian Buster](#) versión de kernel 4.19 (con interfaz gráfica de escritorio).
3. Para poder simular correctamente el sistema se necesita un kernel de Linux separado y su correspondiente árbol con la estructura del núcleo. Para acceder a ello se irá al [repositorio de GitHub de la aplicación](#) y se descargará la versión del kernel que se adecue con la imagen previamente descargada (como descargamos la versión Buster, descargaremos el kernel Buster), así como el correspondiente fichero *versatile-pb.dtb* con el contenido de la estructura del núcleo kernel tal y como explican en el propio repositorio.

Una vez tengamos descargados los diferentes archivos necesarios nombrados anteriormente e instalado QEMU, debemos de disponer de los siguientes archivos:

- *vexpress-v2p-ca15_a7.dtb*
- *2019-11-13-raspbian-buster.img*
- *kernel-qemu-4.4.1-vexpress*

Si tenemos todo listo, podemos ejecutar QEMU, para ello accedemos a la carpeta donde se realizó la instalación de QEMU, abrimos un CMD y ejecutamos la siguiente sentencia: `qemu-system-arm -m 1024 -M vexpress-a15 -cpu cortex-a15 -kernel kernel-qemu-4.4.1-vexpress -no-reboot -dtb vexpress-v2p-ca15_a7.dtb -sd 2019-11-13-raspbian-buster.img -append "console=ttyAMA0 root=/dev/mmcblk0p2 rw rootfstype=ext4" -net nic,model=lan9118 -net user,hostfwd=tcp::2222-:22`

La explicación de los comandos es la siguiente:

- Utilizamos el modelo de placa *vexpress-a15* (-M) y el modelo de CPU *cortex-a15* (-cpu). Este modelo de placa admite como máximo 1024 MB de RAM (-m).
- Las opciones *-kernel* y *-dtb* toman las rutas relativas a los archivos kernel y *dtb* (descargados anteriormente).
- La opción *-serial stdio* redirige los mensajes de salida de arranque y la consola a su terminal. Se puede controlar el nivel de registro utilizando *loglevel* = en la cadena *append*, que contiene todas las opciones para pasar al núcleo en el momento del arranque. En el Raspberry Pi físico, estas opciones se pueden proporcionar en el archivo *cmdline.txt*.
- La opción clave es *root = /dev/mmcblk0p2* que le dice al núcleo la ubicación de la partición raíz que se va a montar.
- La tarjeta de memoria se modela usando *-sd*.
- Con *-net* habilitamos SSH a la máquina.

Ahora puede iniciar sesión con nombre de usuario “*pi*” y contraseña “*raspberry*”.

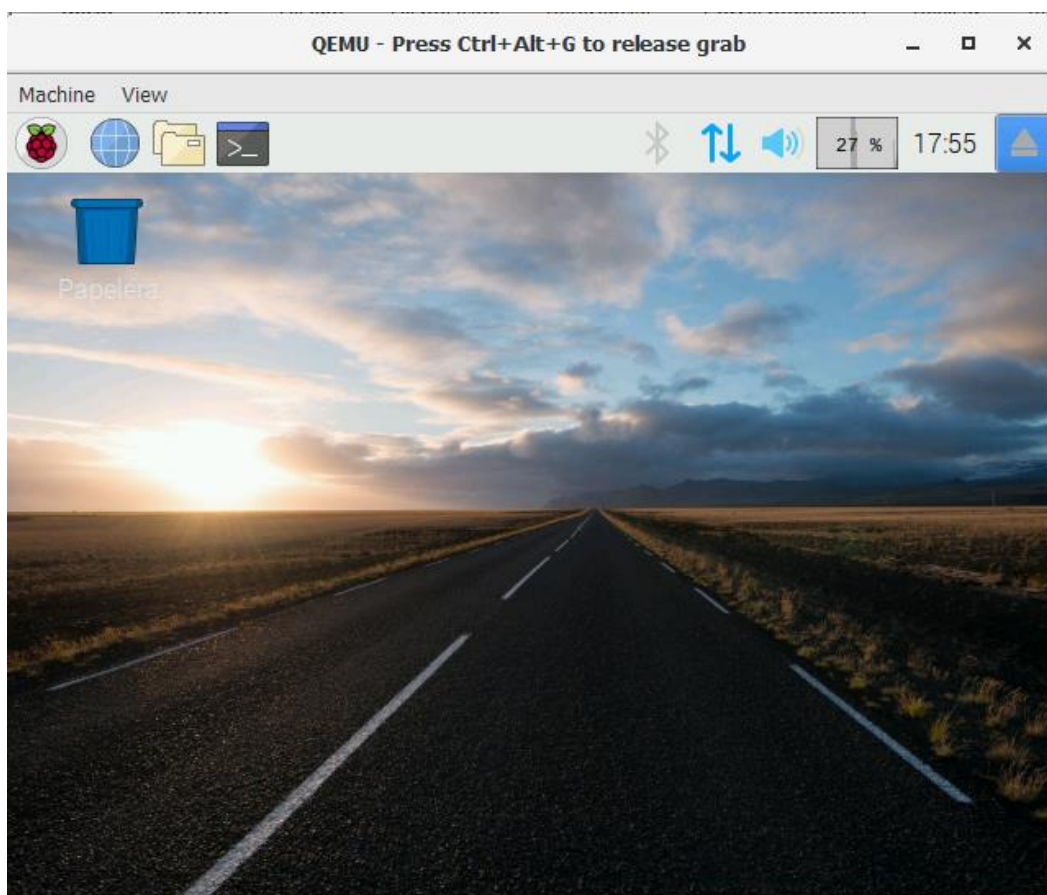


Ilustración 24. Raspbian emulado en QEMU.

Simulando Raspbian Desktop OS sobre un entorno Windows 10.

Este paso es meramente indicativo. Si no se dispone de un sistema con Raspbian Desktop, es posible hacer uso de un sistema de virtualización, como en mi caso VirtualBox, para la simulación. El paso es muy sencillo, basta con descargarse la imagen ISO desde el [repositorio oficial de Raspberry](#) para posteriormente crear la máquina virtual.

En VirtualBox, para crear una máquina virtual nueva tan sólo haremos clic sobre la pestaña “Nueva” y seguimos todos los pasos que nos piden (dejaremos todo por defecto, tan sólo nombraremos a la máquina virtual). Una vez creado tan sólo debemos de hacer una modificación adicional antes de instalar el sistema operativo. Esta modificación consiste en habilitar PAE en la máquina virtual, para ello accedemos a la configuración de la máquina > *Sistema > Procesador > Habilitar PAE*:

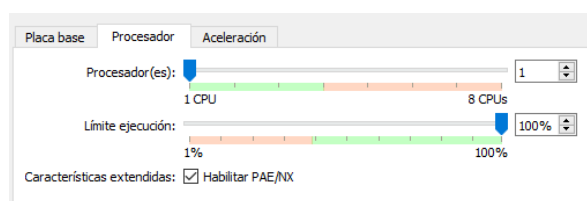


Ilustración 25. Habilitar PAE en VirtualBox.

Ya estamos listo para instalar el sistema operativo en la máquina virtual. Tan sólo iniciamos la máquina virtual y seleccionamos la ISO cuando nos pidan la imagen para iniciar la máquina. Seguiremos todo el paso de instalación tal y como se nos presentan (indicando la cuenta de usuario para iniciar sesión). Una vez finalizado el proceso de instalación, podemos autenticarnos en él y ya estaríamos listo para realizar el volcado de memoria:

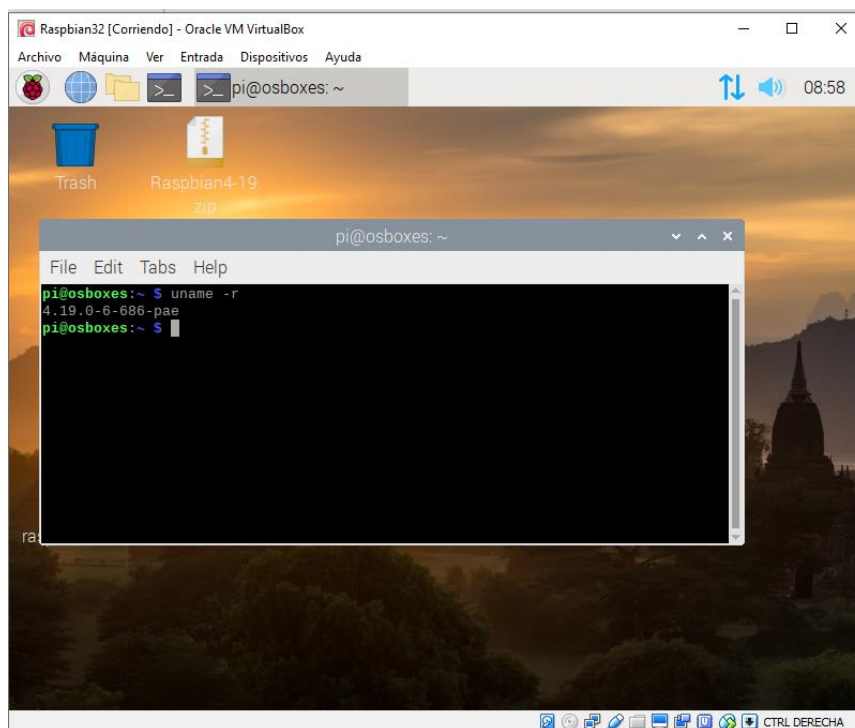


Ilustración 26.. Raspbian Desktop instalado en VirtualBox.

Generando el volcado de memoria de Raspbian con LiME.

Una vez instalado LiME en el sistema siguiendo los pasos del anexo “Instalación de LiME”, es decir, clonado, instalación de las dependencias y compilación de los módulos de la herramienta podemos generar fácilmente el volcado de memoria de la siguiente manera:

- `sudo insmod ./lime-4.19.0-6-686-pae.ko "path=../../raspbian.mem format=lime"`

Indicando para ello el ejecutable generado durante la compilación de los módulos de LiME (*make*), en este caso: *lime-4.19.0-6-686-pae.ko* y la ruta donde guardaremos el volcado de memoria. El volcado de memoria se llamará *raspbian.mem* y tendrá el formato *lime*. Esto es debido a que Volatility acepta el formato *lime* como tipo de fichero:

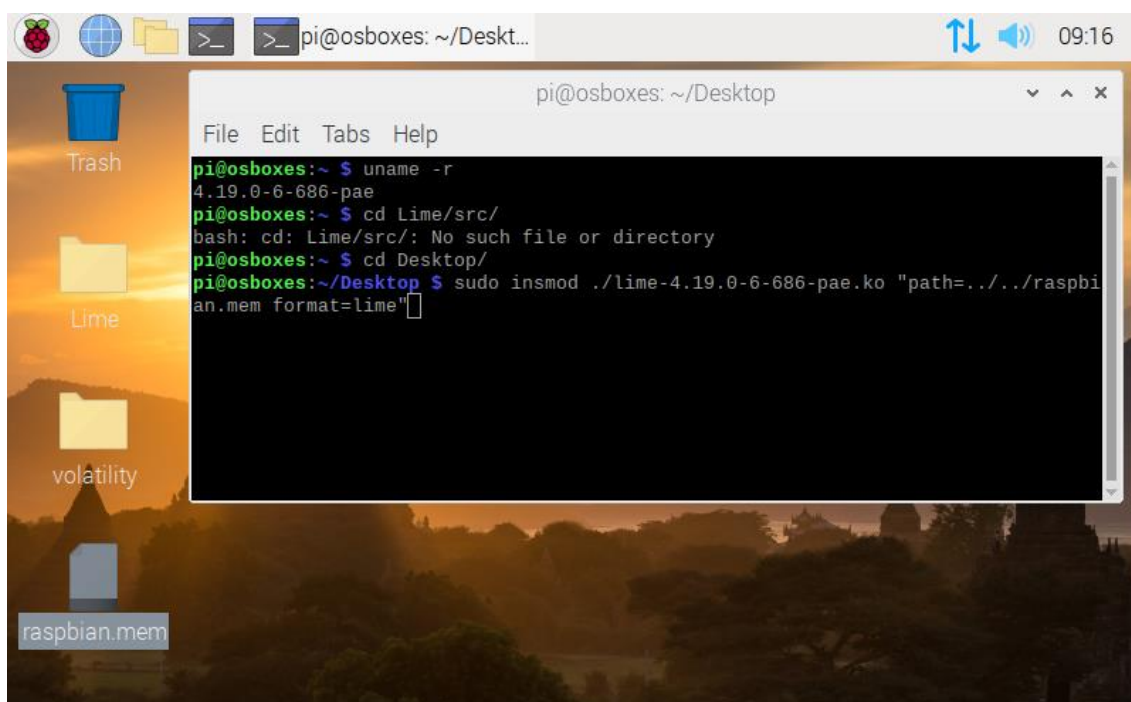


Ilustración 27. Generando el volcado de memoria con LiME.

Generando el perfil/profile de Volatility del entorno Raspbian 4.19 e instalación en Volatility.

Para poder generar el perfil de Volatility de Raspbian 4.19, se necesita disponer de Volatility instalado en el sistema. En el anexo “Instalación de Volatility” se describe todos los pasos para llevar a cabo dicho paso. Una vez instalado en el sistema debemos de instalar las dependencias necesarias (en caso de no tenerlas ya) [35]:

- `apt-get install dwarfdump`
- `apt-get install make`
- `apt-get install build-essential`
- `apt-get install linux-headers-generic`

- 1) Una vez instaladas las dependencias el siguiente paso es generar el fichero *module.dwarf*. Para ello nos dirigimos a la carpeta donde hemos clonado Volatility:
 - a. `cd volatility/tools/linux`

Y dentro de `/tools/Linux`, debemos de cerciorarnos que existe un fichero *Makefile*. Posteriormente compilamos los módulos mediante “*make*”. Al terminar de compilarse los módulos, deberá de aparecer en el mismo directorio el fichero *module.dwarf* ya creado:

```
pi@osboxes:~/Desktop/volatility/tools/linux $
pi@osboxes:~/Desktop/volatility/tools/linux $ ls
kcore Makefile Makefile.enterprise module.c
pi@osboxes:~/Desktop/volatility/tools/linux $ make
make -C //lib/modules/4.19.0-6-686-pae/build CONFIG_DEBUG_INFO=y M="/home/pi/Desktop/volatility/tools/linux" modules
make[1]: Entering directory '/usr/src/linux-headers-4.19.0-6-686-pae'
CC [M] /home/pi/Desktop/volatility/tools/linux/module.o
Building modules into the image tree
make[1]: Leaving directory '/usr/src/linux-headers-4.19.0-6-686-pae'
pi@osboxes:~/Desktop/volatility/tools/linux $ ls
kcore Makefile Makefile.enterprise module.c module.dwarf
pi@osboxes:~/Desktop/volatility/tools/linux $
```

Ilustración 28. Generando el fichero *module.dwarf*.

- 2) El siguiente paso es comprobar que el fichero *System.map* esté en nuestro sistema. Normalmente dicho fichero se encuentra bajo el directorio `/boot`:

```
pi@osboxes:~/Desktop/volatility/tools/linux $ ls /boot/
config-4.19.0-6-686-pae      lost+found
grub                        System.map-4.19.0-6-686-pae
initrd.img-4.19.0-6-686-pae vmlinuz-4.19.0-6-686-pae
```

Ilustración 29. Adquisición del fichero *System.map*.

Como se puede comprobar en la imagen anterior [Ilustración 29], el fichero *System.map* aparece junto con la versión del kernel.

- 3) Por último, debemos de generar un fichero *ZIP* que contenga los dos ficheros antes mencionados. Esto es debido a que para incluirlo en Volatility, se necesita tenerlos comprimidos en uno:
 - a. `sudo zip volatility/plugins/linux/Raspbian4-19.zip tools/linux/module.dwarf /boot/System.map-4.19.0-6-686-pae`
 - i. El fichero conviene nombrarlo con la distro y la versión de esta para poderla asociar de manera correcta en Volatility.

Ya el perfil ha sido creado correctamente. Ahora tan sólo queda importarlo en Volatility. Para ello lo enviamos a la máquina Kali que es donde realizaremos el análisis (nunca en la máquina auditada) e incluimos el fichero *.zip* recién creado bajo el directorio donde tenemos instalado Volatility: `volatility/volatility/plugins/overlays/linux`.

Para comprobar que la instalación se ha llevado a cabo correctamente, ejecutamos: `python vol.py -info`. Y deberá de aparecernos bajo la pestaña “*Profiles*”, el perfil creado (con el nombre que hemos puesto al zip):

```
root@osboxes:~/Desktop/volatility# python vol.py --info
Volatility Foundation Volatility Framework 2.6.1
```

```
Profiles
-----
LinuxDebian40r91x86 - A Profile for Linux Debian40r91 x86
LinuxDebian40r9x64 - A Profile for Linux Debian40r9 x64
LinuxDebian50101x86 - A Profile for Linux Debian50101 x86
LinuxDebian5010x64 - A Profile for Linux Debian5010 x64
LinuxDebian6081x86 - A Profile for Linux Debian6081 x86
LinuxDebian608x64 - A Profile for Linux Debian608 x64
LinuxDebian731x86 - A Profile for Linux Debian731 x86
LinuxDebian73x64 - A Profile for Linux Debian73 x64
LinuxDebian741x86 - A Profile for Linux Debian741 x86
LinuxDebian74x64 - A Profile for Linux Debian74 x64
LinuxDebian82x64 - A Profile for Linux Debian82 x64
LinuxDebian83x64 - A Profile for Linux Debian83 x64
LinuxDebian84x64 - A Profile for Linux Debian84 x64
LinuxDebian86x64 - A Profile for Linux Debian86 x64
LinuxDebian8x64 - A Profile for Linux Debian8 x64
LinuxDebian94x64 - A Profile for Linux Debian94 x64
LinuxRaspbian4-19x86 - A Profile for Linux Raspbian4-19 x86
```

Ilustración 30. Perfil instalado correctamente.

Visualización de procesos en ejecución

Para la visualización de los procesos en ejecución se puede utilizar el plugin *linux_pslist* de la siguiente manera:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_pslist`

Con “-f” seleccionamos la memoria a analizar y con *profile* indicamos el perfil de análisis a utilizar en el volcado, que en este caso es el perfil creado de Raspbian (*LinuxRaspbian4-19x86*). Por último, indicamos el plugin, que para este caso se usará *linux_pslist* para listar los procesos:

```
root@osboxes:~/Desktop/volatility# cat pslist.txt
```

Offset	Name	Pid	PPid	Uid	Gid	DTB	Start Time
0xf6110bc0	systemd	1	0	0	0	0x3541e000	0
0xf6114680	kthreadd	2	0	0	0	-----	0
0xf6115e00	rcu_gp	3	2	0	0	-----	0
0xf6112f00	rcu_par_gp	4	2	0	0	-----	0
0xf6111780	kworker/0:0	5	2	0	0	-----	0
0xf61169c0	kworker/0:0H	6	2	0	0	-----	0
0xf6113ac0	mm_percpu_wq	8	2	0	0	-----	0
0xf6115240	ksoftirqd/0	9	2	0	0	-----	0
0xf6110000	rcu_sched	10	2	0	0	-----	0
0xf613bac0	rcu_bh	11	2	0	0	-----	0
0xf613d240	migration/0	12	2	0	0	-----	0
0xf6138bc0	cpuhp/0	14	2	0	0	-----	0
0xf613c680	kdevtmpfs	15	2	0	0	-----	0
0xf613de00	netns	16	2	0	0	-----	0
0xf613af00	kauditd	17	2	0	0	-----	0
0xf6139780	khungtaskd	18	2	0	0	-----	0
0xf613e9c0	oom_reaper	19	2	0	0	-----	0
0xf613a340	writeback	20	2	0	0	-----	0
0xf616bac0	kcompactd0	21	2	0	0	-----	0
0xf616d240	ksmd	22	2	0	0	-----	0
0xf6168000	khugepaged	23	2	0	0	-----	0
0xf6168bc0	crypto	24	2	0	0	-----	0
0xf616c680	kintegrityd	25	2	0	0	-----	0
0xf616de00	kblockd	26	2	0	0	-----	0
0xf616af00	edac-poller	27	2	0	0	-----	0
0xf6169780	devfreq_wq	28	2	0	0	-----	0
0xf616e9c0	watchdogd	29	2	0	0	-----	0
0xf62e0bc0	kswapd0	32	2	0	0	-----	0

Ilustración 31. Listado de procesos del volcado.

El comando *linux_pstree* nos ofrece la información anterior, pero en formato árbol jerárquico con relación proceso padre-hijo:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_pstree`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_pstree
Volatility Foundation Volatility Framework 2.6.1
Name                               Pid      Uid
systemd                            1
systemd-journal                    251
systemd-udevd                      263
haveged                           377
alsactl                           379
avahi-daemon                       380      109
avahi-daemon                       391      109
thd                                382      65534
udisksd                           383
rsyslogd                          384
systemd-logind                     390
dbus-daemon                       393      105
wpa_supplicant                    395
cron                              406
polkitd                           454
dhcpcd                            463
ntpd                              538      104
lightdm                           553
Xorg                              580
lightdm                           875
lxsession                         965      1000
ssh-agent                         997      1000
openbox                          1020     1000
lxpolkit                         1022     1000
lxpanel                          1024     1000
lxterminal                       1188     1000
bash                             1194     1000
sudo                             1847
insmod                           1848
```

Ilustración 32. Relación de procesos padre-hijo del proceso *systemd*.

La salida completa de los comandos *pslist*, *pstree*, *psscan* y *psxview* se puede acceder a través del siguiente enlace:

<https://www.dropbox.com/sh/a6osk4vt1gdysjf/AADTnKZs9hGvP2vDdk9xXp3za?dl=0>

Visualización de los hilos de ejecución

Mediante la ejecución de la siguiente línea de comando visualizaremos los diferentes hilos de ejecución con sus respectivos procesos:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_threads`

Similar a las ejecuciones anteriores, tan sólo se modificará el plugin a ejecutar, que en este caso será *linux_threads*.

TÉCNICAS Y HERRAMIENTAS PARA EL ANÁLISIS DE DEBILIDADES EN VOLCADOS DE MEMORIA RAM DE SISTEMAS BASADOS EN LINUX

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_thread
Volatility Foundation Volatility Framework 2.6.1

Process Name: systemd
Process ID: 1
Thread PID    Thread Name
-----
1             systemd

Process Name: kthreadd
Process ID: 2
Thread PID    Thread Name
-----
2             kthreadd

Process Name: rcu_gp
Process ID: 3
Thread PID    Thread Name
-----
3             rcu_gp

Process Name: rcu_par_gp
Process ID: 4
Thread PID    Thread Name
-----
4             rcu_par_gp

Process Name: kworker/0:0
Process ID: 5
Thread PID    Thread Name
-----
```

Ilustración 33. Listado de hilos de ejecución del volcado.

La salida completa del comando se puede acceder a través del siguiente enlace:

https://www.dropbox.com/s/0zb9kwir8tnoaom/linux_threads.txt?dl=0

Utilización del plugin malfind sobre el volcado de memoria

Volatility nos ofrece el plugin denominado *malfind*. Dicha herramienta nos permite encontrar códigos/DLL ocultos o inyectados en la memoria, en función de características como la etiqueta VAD y los permisos de página. Con la opción *-p* podemos introducir el PID del proceso que queremos analizar. Por omisión se tratará todos los procesos activos:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_malfind`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 -p 580 linux_malfind
Volatility Foundation Volatility Framework 2.6.1
Process: Xorg Pid: 580 Address: 0xb7146000 File: /
Protection: VM_READ|VM_WRITE|VM_EXEC
Flags: VM_READ|VM_WRITE|VM_EXEC|VM_MAYREAD|VM_MAYWRITE|VM_MAYEXEC|VM_ACCOUNT

0x000000b7146000 cc 7e 01 00 40 39 0c 01 80 66 f9 b7 80 ce a8 b7 .~..@9...f.....
0x000000b7146010 e0 b9 9f b7 f0 37 92 b7 66 20 13 b7 40 b7 a8 b7 .....7..f...@...
0x000000b7146020 86 20 13 b7 96 20 13 b7 20 15 a9 b7 10 9f 99 b7 .....
0x000000b7146030 c6 20 13 b7 34 60 14 b7 8b 05 00 00 ff ff ff ff ....4`.....

0xb7146000 cc INT 3
0xb7146001 7e01 JLE 0xb7146004
0xb7146003 004039 ADD [EAX+0x39], AL
0xb7146006 0c01 OR AL, 0x1
0xb7146008 8066f9b7 AND BYTE [ESI-0x7], 0xb7
0xb714600c 80cea8 OR DH, 0xa8
0xb714600f b7e0 MOV BH, 0xe0
0xb7146011 b99fb7f037 MOV ECX, 0x37f0b79f
0xb7146016 92 XCHG EDX, EAX
0xb7146017 b766 MOV BH, 0x66
0xb7146019 2013 AND [EBX], DL
0xb714601b b740 MOV BH, 0x40
0xb714601d b7a8 MOV BH, 0xa8
0xb714601f b786 MOV BH, 0x86
0xb7146021 2013 AND [EBX], DL
0xb7146023 b796 MOV BH, 0x96
0xb7146025 2013 AND [EBX], DL
0xb7146027 b720 MOV BH, 0x20
0xb7146029 15a9b7109f ADC EAX, 0x9f10b7a9
0xb714602e 99 CDQ
0xb714602f 17 00 MOV AL, 0x00
```

Ilustración 34. Utilización del plugin malfind.

La salida completa del completa se puede acceder a través del siguiente enlace:

https://www.dropbox.com/s/5ir0x8xcbko36w8/linux_malfind.txt?dl=0

Realizando el vaciado de los procesos con el plugin procdump

Con el plugin *linux_procdump* Volatility nos permite hacer un vaciado del proceso indicado. Con el parámetro *-p* indicaremos el PID del proceso que queramos obtener. Por omisión, realizará un vaciado de todos los procesos activos del volcado. Finalmente, con la opción *-D* indicaremos el directorio donde almacenaremos el volcado de los procesos:

- *python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 -D ../TFM/ -p 580 linux_procdump*
 - Con la opción *-p* indicamos el PID del proceso
- *python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 -D ../TFM/ linux_procdump*
 - Si no indicamos un PID de algún proceso, se hará un vaciado completo de todos los procesos.

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 -D ../TFM/dumps/ linux_procdump
Volatility Foundation Volatility Framework 2.6.1
Offset  Name  Pid  Address  Output File
-----
0xf6110bc0  systemd  1  0x004dd000  ../TFM/dumps/systemd.1.0x4dd000
0xf4fd5240  systemd-journal  251  0x0043f000  ../TFM/dumps/systemd-journal.251.0x43f000
0xf400a340  systemd-udev  263  0x00461000  ../TFM/dumps/systemd-udev.263.0x461000
0xf3c569c0  haveged  377  0x00439000  ../TFM/dumps/haveged.377.0x439000
0xf3c52f00  alsactl  379  0x00434000  ../TFM/dumps/alsactl.379.0x434000
0xf3c54600  avahi-daemon  380  0x0042d000  ../TFM/dumps/avahi-daemon.380.0x42d000
0xf3c50000  thd  382  0x004d8000  ../TFM/dumps/thd.382.0x4d8000
0xf3c51700  udiskd  383  0x004e8000  ../TFM/dumps/udiskd.383.0x4e8000
0xf408de00  rsyslogd  384  0x00481000  ../TFM/dumps/rsyslogd.384.0x481000
0xf4fd2340  systemd-logind  390  0x004be000  ../TFM/dumps/systemd-logind.390.0x4be000
0xf44869c0  avahi-daemon  391  0x0042d000  ../TFM/dumps/avahi-daemon.391.0x42d000
0xf4482340  dbus-daemon  393  0x00478000  ../TFM/dumps/dbus-daemon.393.0x478000
0xf38e69c0  wpa_supplicant  395  0x004d3000  ../TFM/dumps/wpa_supplicant.395.0x4d3000
0xf38e0000  cron  406  0x00430000  ../TFM/dumps/cron.406.0x430000
0xf38e4680  polkitd  454  0x00431000  ../TFM/dumps/polkitd.454.0x431000
0xf3af4680  dhcpcd  463  0x00481000  ../TFM/dumps/dhcpcd.463.0x481000
0xf3418bc0  ntpd  538  0x004eb000  ../TFM/dumps/ntpd.538.0x4eb000
0xf3ae0000  lightdm  553  0x00498000  ../TFM/dumps/lightdm.553.0x498000
0xf3ae0bc0 agetty  555  0x004f4000  ../TFM/dumps/agetty.555.0x4f4000
0xf34b4680  Xorg  580  0x004a0000  ../TFM/dumps/Xorg.580.0x4a0000
0xf3408bc0  exim4  828  0x00485000  ../TFM/dumps/exim4.828.0x485000
0xf34b5e00  lightdm  875  0x004fa000  ../TFM/dumps/lightdm.875.0x4fa000
0xf3ae2340  systemd  954  0x00400000  ../TFM/dumps/systemd.954.0x400000
0xf3ae2f00  (sd-pam)  955  0x004dd000  ../TFM/dumps/(sd-pam).955.0x4dd000
0xf3ae4680  lxsession  965  0x0048f000  ../TFM/dumps/lxsession.965.0x48f000
0xf3578bc0  dbus-daemon  973  0x0049a000  ../TFM/dumps/dbus-daemon.973.0x49a000
0xf37e1700  ssh-agent  997  0x004e9000  ../TFM/dumps/ssh-agent.997.0x4e9000
0xf3579780  gvfsd  1005  0x0045e000  ../TFM/dumps/gvfsd.1005.0x45e000
0xf357e9c0  gvfsd-fuse  1010  0x0044f000  ../TFM/dumps/gvfsd-fuse.1010.0x44f000
0xf37e69c0  openbox  1020  0x00457000  ../TFM/dumps/openbox.1020.0x457000
0xf37e3ac0  lxpolkit  1022  0x00490000  ../TFM/dumps/lxpolkit.1022.0x490000
0xf37e0bc0  lxpanel  1024  0x00475000  ../TFM/dumps/lxpanel.1024.0x475000
0xf3405e00  pcmanfm  1027  0x004e8000  ../TFM/dumps/pcmanfm.1027.0x4e8000
0xf3400000  ssh-agent  1037  0x0044a000  ../TFM/dumps/ssh-agent.1037.0x44a000
0xe949af00  menu-cached  1072  0x00475000  ../TFM/dumps/menu-cached.1072.0x475000
0xf3635240  gvfs-udisks2-vo  1076  0x00435000  ../TFM/dumps/gvfs-udisks2-vo.1076.0x435000
0xf3632340  gvfs-goa-volume  1080  0x0046c000  ../TFM/dumps/gvfs-goa-volume.1080.0x46c000
0xf357c680  gvfs-mtp-volume  1084  0x004cf000  ../TFM/dumps/gvfs-mtp-volume.1084.0x4cf000
0xe9570bc0  gvfs-gphoto2-vo  1088  0x004a3000  ../TFM/dumps/gvfs-gphoto2-vo.1088.0x4a3000
0xe9570000  gvfs-afc-volume  1092  0x0046d000  ../TFM/dumps/gvfs-afc-volume.1092.0x46d000
0xe95d5240  gvfsd-trash  1115  0x004b3000  ../TFM/dumps/gvfsd-trash.1115.0x4b3000
0xf36c4680  lxterminal  1188  0x0045a000  ../TFM/dumps/lxterminal.1188.0x45a000
0xf36c69c0  bash  1194  0x0042e000  ../TFM/dumps/bash.1194.0x42e000
0xe970a340  sudo  1847  0x0040d000  ../TFM/dumps/sudo.1847.0x40d000
0xe970de00  insmod  1848  0x0046f000  ../TFM/dumps/insmod.1848.0x46f000
```

Ilustración 35. Vaciado de todos los procesos activos.

La salida completa de la ejecución de la herramienta está disponible a través del siguiente enlace: <https://www.dropbox.com/s/4c8srrxgmeipqiy/dump%20process.zip?dl=0>

Analizando los procesos

Una vez se ha realizado el volcado de los procesos, el siguiente paso es realizar el análisis de las cadenas de textos de estos. Para ello, haré uso de la herramienta *strings* de Linux, el cual nos permite ver las cadenas de textos de ficheros binarios. Para ello hacemos lo siguiente, vamos al directorio donde almacenamos los procesos descargados y ejecutamos la siguiente sentencia:

- *strings sudo.1847.0x40d000 | tail -30*
 - Con esto conseguimos ver las últimas 30 cadenas de texto del volcado del proceso indicado, en este caso, *sudo*, cuyo PID es 1847. Imprimimos tan sólo las últimas 30 como demostración ya que el volcado contiene muchas cadenas de texto (*wc -l sudo.1847.0x40d000*).

```
root@osboxes:~/Desktop/TFM/dumps/dump process# wc -l sudo.1847.0x40d000
608 sudo.1847.0x40d000
```

Ilustración 36. Cantidad de líneas del volcado del proceso *sudo*.

```
root@osboxes:~/Desktop/TFM/dumps/dump process# strings sudo.1847.0x40d000 | tail -30
failed to set new role %s
failed to set new type %s
%s is not a valid context
failed to get old_context
chr_file
unable to set new tty context
-sesh-noexec
-sesh
--execfd=%d
newrole: old-context=%s new-context=%s
unable to restore context for %s
you must specify a role for type %s
unable to get default type for role %s
unable to determine enforcing mode.
unable to open %s, not relabeling tty
%s is not a character device, not relabeling tty
unable to get current tty context, not relabeling tty
unknown security class "chr_file", not relabeling tty
unable to get new tty context, not relabeling tty
unable to set tty context to %s
internal error: sesh path not set
unable to set exec context to %s
unable to set key creation context to %s
selinux_execve
audit_role_change
relabel_tty
selinux_setup
get_exec_context
selinux_restore_tty
;*2$"
```

Ilustración 37. Uso de la herramienta *strings* para el análisis de ficheros binarios.

Esta ejecución por sí sola carece de utilidad en la práctica, ya que el mero hecho de buscar las cadenas de texto por sí solas no nos aporta ninguna utilidad. Para ello, debemos de combinar dicha *strings* con buscadores o filtros, por ejemplo, *grep* (*sort* para ordenar, *uniq* para evitar duplicados, etc.). De esta manera seremos capaces de buscar o filtrar cadenas de texto que contengan información que puede ser útil y nos aporte valor para la auditoría o el caso que estemos tratando, por ejemplo, para búsquedas de ficheros PDF, conexiones a webs externas, credenciales, código JavaScript, etc. Veamos un ejemplo:

- `strings dbus-daemon.393.0x478000 | grep http`
 - En este caso estamos analizando el volcado del proceso `dbus-daemon` con PID 393 en busca de URLs que puedan ser conexiones maliciosas, por ejemplo.

```
root@osboxes: ~/Desktop/TFM/dumps/dump process# strings dbus-daemon.393.0x478000 | grep http
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
```

Ilustración 38. Strings y grep para el filtrado de procesos.

De esta manera se podrá ver si hay algún proceso que descarga algún fichero malicioso de algún servidor web. En el caso de que trabajemos con muchos procesos a la vez, la utilización del comando anterior de manera individual para cada proceso puede ser un trabajo costoso y complejo. Por ende, se recomienda la creación de scripts que permitan automatizar el proceso e identificar los procesos maliciosos de manera eficaz. Para demostrar las ventajas que los scripts nos aportan veamos el siguiente ejemplo. Crearemos un script que permita ver conexiones HTTP en todos los procesos del directorio, para ello crearemos un script denominado `script.sh` y le daremos permisos de ejecución de la siguiente manera: `chmod +x script.sh`.

Una vez creado y con permisos de ejecución introduciremos la siguiente línea de código:

- `for file in $(ls); do echo $file; strings $file | grep http; done`

Básicamente, con la línea de código anterior, recorreremos todos los ficheros del directorio y para cada fichero, instanciado por `$file`, aplicaremos la herramienta `strings` y mediante `grep` buscaremos la palabra `http` en cada cadena de texto de cada proceso del directorio.

Si ahora procedemos a ejecutar el script, mediante `./script.sh`, generaremos el siguiente resultado:

```
root@osboxes: ~/Desktop/TFM/dumps/dump process# ./script.sh
agetty.555.0x4f4000
alsactl.379.0x434000
avahi-daemon.380.0x42d000
avahi-daemon.391.0x42d000
bash.1194.0x42e000
cron.406.0x430000
dbus-daemon.393.0x478000
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
dbus-daemon.973.0x49a000
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
dhcpcd.463.0x481000
exim4.828.0x485000
gvfs-afc-volume.1092.0x46d000
gvfsd.1005.0x45e000
gvfsd-fuse.1010.0x44f000
gvfsd-trash.1115.0x4b3000
gvfs-goa-volume.1080.0x46c000
gvfs-gphoto2-vo.1088.0x4a3000
gvfs-mtp-volume.1084.0x4cf000
gvfs-udisks2-vo.1076.0x435000
haveged.377.0x439000
insmod.1848.0x46f000
lightdm.553.0x498000
lightdm.875.0x4fa000
lxpanel.1024.0x475000
lxpolkit.1022.0x490000
lxsession.965.0x48f000
lxterminal.1188.0x45a000
http://lxde.org/
```

Ilustración 39. Resultado de la ejecución del script.sh.

Observamos que nos aparece para cada fichero, el nombre del fichero y debajo, el enlace si hay alguna ocurrencia. Para este caso observamos que tanto los procesos cuyo PID son 393, 973 como 1188 han tenido ocurrencias, en los dos primeros casos parece que se referencia a un binario denominado *introspect.dtd* perteneciente a la URL *freedesktop.org*. No sólo con *strings* y *grep* podemos analizar los procesos. Existen herramientas como *foremost* que nos permiten buscar ficheros PDFs dentro de los volcados de los procesos obtenidos. Para poder utilizar dicha herramienta tan sólo hay que instalarla mediante:

- `apt-get install foremost`

Normalmente, en distribuciones Kali Linux, esta herramienta está preinstalada en el sistema. *Foremost* no busca en los procesos ficheros con terminación *.pdf* (ya que se pueden camuflar), si no se busca por las cabeceras de los archivos para determinar qué tipo de archivo se trata (los ficheros PDF tienen como cabecera la siguiente ristra de valores: 25 50 44 46). Veamos un ejemplo:

- `foremost -i systemd* -t pdf -o pdfs2`
 - Dicho comando busca los procesos que empiezan por *systemd* (*-i, input*), y busca si dichos procesos contienen ficheros de tipo *pdf* (*-t*). El resultado del análisis, así como los ficheros, en caso de que existan, se almacenarán en el directorio indicado con el parámetro *-o, output*.

```
root@osboxes: ~/Desktop/TFM/dumps/dump process# foremost -i systemd* -t pdf -o pdfs2
Processing: systemd.1.0x4dd000
||
Processing: systemd.954.0x400000
||
Processing: systemd-journal.251.0x43f000
|*|
Processing: systemd-logind.390.0x4be000
|*|
Processing: systemd-udev.263.0x461000
||
```

Ilustración 40. Usando la herramienta *foremost*.

Precisamente en esta ejecución, ninguno de los procesos auditados contenía ficheros con la cabecera que corresponde a los ficheros *pdf*. Independientemente de si se encuentra o no coincidencias, la herramienta genera un fichero en la ruta de salida denominado *audit.txt* con el resumen del análisis:

```
root@osboxes: ~/Desktop/TFM/dumps/dump process# cat pdfs2/audit.txt
Foremost version 1.5.7 by Jesse Kornblum, Kris Kendall, and Nick Mikus
Audit File

Foremost started at Thu Apr 16 14:00:00 2020
Invocation: foremost -i systemd.1.0x4dd000 systemd.954.0x400000 systemd-journal.251.0x43f000 systemd-logind.390.0x4be000 systemd-udev.263.0x461000 -t pdf -o pdfs2
Output directory: /root/Desktop/TFM/dumps/dump process/pdfs2
Configuration file: /etc/foremost.conf
-----
File: systemd.1.0x4dd000
Start: Thu Apr 16 14:00:00 2020
Length: Unknown

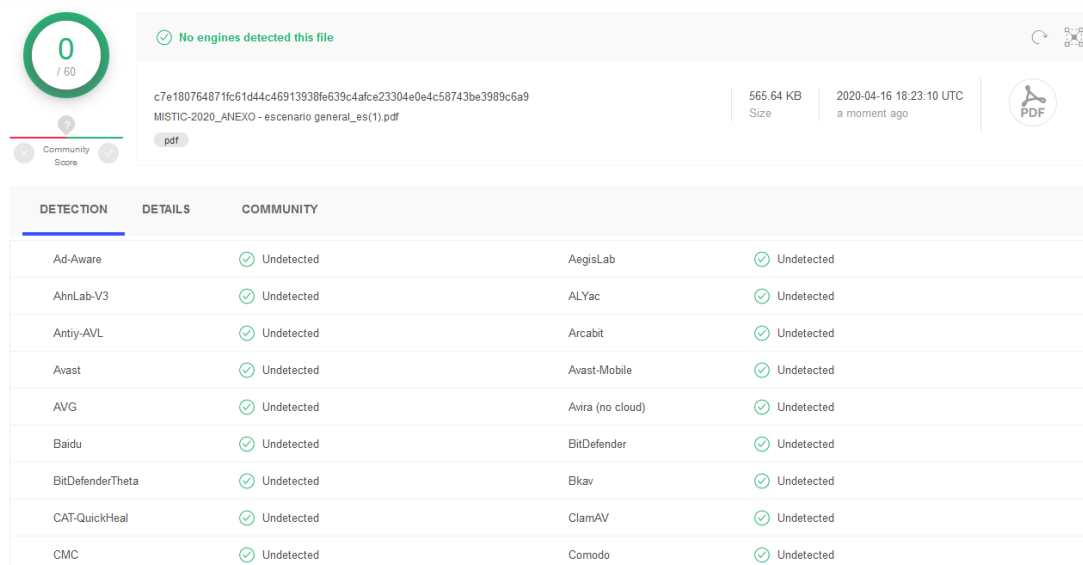
Num      Name (bs=512)      Size      File Offset      Comment
-----
Finish: Thu Apr 16 14:00:00 2020
-----
File: systemd.954.0x400000
Start: Thu Apr 16 14:00:00 2020
Length: Unknown

Num      Name (bs=512)      Size      File Offset      Comment
-----
Finish: Thu Apr 16 14:00:00 2020
-----
File: systemd-journal.251.0x43f000
Start: Thu Apr 16 14:00:00 2020
Length: 148 KB (151552 bytes)

Num      Name (bs=512)      Size      File Offset      Comment
-----
Finish: Thu Apr 16 14:00:00 2020
```

Ilustración 41. Contenido fichero *audit.txt*.

Si se hubiera encontrado algún fichero PDF, ejecutables, *dlls* etc. que puede resultar sospechoso, el siguiente paso natural es hacer uso de alguna base de datos para ver si dichos ficheros contienen malware según los diferentes antivirus del mercado. Una de las bases de datos más conocida es la de Virustotal (<https://www.virustotal.com/gui/home>) donde tan sólo subiendo cualquier tipo de fichero nos devuelve si alguno de los antivirus del mercado devuelve positivo en malware. Por ejemplo:



DETECTION	DETAILS	COMMUNITY
Ad-Aware	Undetected	AegisLab
AhnLab-V3	Undetected	ALYac
Antiy-AVL	Undetected	Arcabit
Avast	Undetected	Avast-Mobile
AVG	Undetected	Avira (no cloud)
Baidu	Undetected	BitDefender
BitDefenderTheta	Undetected	Bkav
CAT-QuickHeal	Undetected	ClamAV
CMC	Undetected	Comodo

Ilustración 42. Ejemplo de detección de malware utilizando Virustotal.

Para este ejemplo, tan sólo se ha subido un fichero de un ejercicio de una asignatura para comprobar si existe algún malware. Tal y como se puede comprobar, el resultado ha salido totalmente limpio.

Tanto el script creado, como la salida de este y el resumen generado con la herramienta *foremost* están disponibles a través del siguiente enlace:

<https://www.dropbox.com/sh/j8rliykihw2d6it/AACGe8OpFgl9ADZIL5bEWtPQa?dl=0>

Maapeo de memoria de los distintos procesos

El primer comando que vamos a tratar es el denominado *memmap*. Dicho plugin permite saber exactamente qué páginas residen en la memoria dado un proceso específico (o todos si no se especifica ninguno). Muestra la dirección virtual de la página, el desplazamiento físico correspondiente y el tamaño total de la página. Para comprobar la distribución de los procesos en memoria ejecutamos la siguiente instrucción:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 -p 1847 linux_memmap`
 - Con la ejecución de la sentencia anterior estamos comprobando la distribución en memoria del proceso con PID=1847, que en este caso se trata del proceso *sudo*.

TÉCNICAS Y HERRAMIENTAS PARA EL ANÁLISIS DE DEBILIDADES EN VOLCADOS DE MEMORIA RAM DE SISTEMAS BASADOS EN LINUX

Task	Pid	Virtual	Physical	Size
sudo	1847	0x0040d000	0x3dcfc000	0x1000
sudo	1847	0x0040e000	0x3def3000	0x1000
sudo	1847	0x0040f000	0x3e25d000	0x1000
sudo	1847	0x00410000	0x3dee9000	0x1000
sudo	1847	0x00411000	0x3defa000	0x1000
sudo	1847	0x00412000	0x3b485000	0x1000
sudo	1847	0x00413000	0x3e246000	0x1000
sudo	1847	0x00414000	0x3def1000	0x1000
sudo	1847	0x00415000	0x3b2b4000	0x1000
sudo	1847	0x00416000	0x3db56000	0x1000
sudo	1847	0x00417000	0x3b3bb000	0x1000
sudo	1847	0x00418000	0x3a77e000	0x1000
sudo	1847	0x00419000	0x3b27c000	0x1000
sudo	1847	0x0041a000	0x3dcfa000	0x1000
sudo	1847	0x0041b000	0x3b370000	0x1000
sudo	1847	0x0041c000	0x3b2b7000	0x1000
sudo	1847	0x0041d000	0x3dee2000	0x1000
sudo	1847	0x0041e000	0x3b3db000	0x1000
sudo	1847	0x0041f000	0x3b3aa000	0x1000
sudo	1847	0x00420000	0x3def8000	0x1000
sudo	1847	0x00421000	0x3def2000	0x1000
sudo	1847	0x00422000	0x38fed000	0x1000
sudo	1847	0x00423000	0x3df5d000	0x1000
sudo	1847	0x00424000	0x38fda000	0x1000
sudo	1847	0x00425000	0x22794000	0x1000

Ilustración 43. Ejecución del plugin memmap.

El siguiente comando que podemos resaltar es el denominado *proc_maps*. Dicho plugin nos permite conocer detalles de la memoria del proceso, incluidos permisos, pilas/*stacks* y bibliotecas compartidas. Veamos un ejemplo:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_proc_maps`
 - Con el parámetro *-s (segment)* podemos indicar una dirección de un segmento de memoria para extraer y, con la opción *-D*, estableceremos el directorio de salida.

Offset	Pid	Name	Start	End	Flags	Pgoff	Major	Minor	Inode	File Path
0x00000000f6110bc0	1	systemd	0x0000000004dd0000	0x00000000004f3000	r--	0x0	8	1	2625912	/
0x00000000f6110bc0	1	systemd	0x00000000004f3000	0x00000000005ac000	r-x	0x16000	8	1	2625912	/
0x00000000f6110bc0	1	systemd	0x00000000005ac000	0x000000000061e000	r--	0xc0000	8	1	2625912	/
0x00000000f6110bc0	1	systemd	0x000000000061e000	0x000000000063e000	r--	0x140000	8	1	2625912	/
0x00000000f6110bc0	1	systemd	0x000000000063e000	0x000000000063f000	rw-	0x160000	8	1	2625912	/
0x00000000f6110bc0	1	systemd	0x000000000063f000	0x0000000000670000	rw-	0x0	0	0	0	[heap]
0x00000000f6110bc0	1	systemd	0x0000000000670000	0x00000000006021000	rw-	0x0	0	0	0	
0x00000000f6110bc0	1	systemd	0x00000000006021000	0x00000000006100000	---	0x0	0	0	0	
0x00000000f6110bc0	1	systemd	0x00000000006100000	0x00000000006160000	---	0x0	0	0	0	
0x00000000f6110bc0	1	systemd	0x00000000006160000	0x00000000006960000	rw-	0x0	0	0	0	
0x00000000f6110bc0	1	systemd	0x00000000006960000	0x00000000006961000	---	0x0	0	0	0	
0x00000000f6110bc0	1	systemd	0x00000000006961000	0x00000000007165000	rw-	0x0	0	0	0	
0x00000000f6110bc0	1	systemd	0x00000000007165000	0x0000000000716f000	r--	0x0	8	1	2623203	/
0x00000000f6110bc0	1	systemd	0x0000000000716f000	0x00000000007231000	r-x	0xa000	8	1	2623203	/
0x00000000f6110bc0	1	systemd	0x00000000007231000	0x00000000007269000	r--	0xcc000	8	1	2623203	/
0x00000000f6110bc0	1	systemd	0x00000000007269000	0x0000000000726a000	r--	0x103000	8	1	2623203	/
0x00000000f6110bc0	1	systemd	0x0000000000726a000	0x0000000000726b000	rw-	0x104000	8	1	2623203	/
0x00000000f6110bc0	1	systemd	0x0000000000726b000	0x0000000000726e000	r--	0x0	8	1	2623299	/
0x00000000f6110bc0	1	systemd	0x0000000000726e000	0x00000000007285000	r-x	0x3000	8	1	2623299	/
0x00000000f6110bc0	1	systemd	0x00000000007285000	0x00000000007290000	r--	0x1a000	8	1	2623299	/
0x00000000f6110bc0	1	systemd	0x00000000007290000	0x00000000007291000	r--	0x24000	8	1	2623299	/
0x00000000f6110bc0	1	systemd	0x00000000007291000	0x00000000007292000	rw-	0x25000	8	1	2623299	/
0x00000000f6110bc0	1	systemd	0x00000000007292000	0x00000000007295000	r--	0x0	8	1	2623186	/
0x00000000f6110bc0	1	systemd	0x00000000007295000	0x000000000072a8000	r-x	0x3000	8	1	2623186	/
0x00000000f6110bc0	1	systemd	0x000000000072a8000	0x000000000072b5000	r--	0x16000	8	1	2623186	/
0x00000000f6110bc0	1	systemd	0x000000000072b5000	0x000000000072b6000	r--	0x22000	8	1	2623186	/
0x00000000f6110bc0	1	systemd	0x000000000072b6000	0x000000000072b7000	rw-	0x23000	8	1	2623186	/

Ilustración 44. Plugin linux_proc_maps.

Dicho comando se suele ejecutar en conjunto con el denominado *linux_dump_map* para extraer segmentos/rangos de memoria, para su posterior análisis.

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 --dump-dir=../dump_map/ linux_dump_map`
 - Indicamos con la opción *-D* el directorio donde guardamos el volcado del segmento indicado (por omisión se hará un volcado completo).

Task	VM Start	VM End	Length	Path
1	0x004dd000	0x004f3000	0x16000	../dump_map/task.1.0x4dd000.vma
1	0x004f3000	0x005ac000	0xb9000	../dump_map/task.1.0x4f3000.vma
1	0x005ac000	0x0061e000	0x72000	../dump_map/task.1.0x5ac000.vma
1	0x0061e000	0x0063e000	0x20000	../dump_map/task.1.0x61e000.vma
1	0x0063e000	0x0063f000	0x1000	../dump_map/task.1.0x63e000.vma
1	0x0096a000	0x00a67000	0xfd000	../dump_map/task.1.0x96a000.vma
1	0xb6000000	0xb6021000	0x21000	../dump_map/task.1.0xb6000000.vma
1	0xb6021000	0xb6100000	0xdf000	../dump_map/task.1.0xb6021000.vma
1	0xb615f000	0xb6160000	0x1000	../dump_map/task.1.0xb615f000.vma
1	0xb6160000	0xb6960000	0x800000	../dump_map/task.1.0xb6160000.vma
1	0xb6960000	0xb6961000	0x1000	../dump_map/task.1.0xb6960000.vma
1	0xb6961000	0xb7165000	0x804000	../dump_map/task.1.0xb6961000.vma
1	0xb7165000	0xb716f000	0xa000	../dump_map/task.1.0xb7165000.vma
1	0xb716f000	0xb7231000	0xc2000	../dump_map/task.1.0xb716f000.vma
1	0xb7231000	0xb7269000	0x38000	../dump_map/task.1.0xb7231000.vma
1	0xb7269000	0xb726a000	0x1000	../dump_map/task.1.0xb7269000.vma
1	0xb726a000	0xb726b000	0x1000	../dump_map/task.1.0xb726a000.vma
1	0xb726b000	0xb726e000	0x3000	../dump_map/task.1.0xb726b000.vma
1	0xb726e000	0xb7285000	0x17000	../dump_map/task.1.0xb726e000.vma
1	0xb7285000	0xb7290000	0xb000	../dump_map/task.1.0xb7285000.vma
1	0xb7290000	0xb7291000	0x1000	../dump_map/task.1.0xb7290000.vma
1	0xb7291000	0xb7292000	0x1000	../dump_map/task.1.0xb7291000.vma

Ilustración 45. Utilización del plugin linux_dump_map.

Por último, es importante destacar el comando *linux_bash*. Dicho comando nos permite recuperar el historial del *bash* de la memoria, incluso si se intenta prohibir que el sistema guarde dicha información contra la lucha con el análisis forense y la seguridad informática por parte de los cibercriminales (por ejemplo, si *HISTSIZE*⁶⁸ se establece en 0 o *HISTFILE* apunta a */dev/null*).

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_bash`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_bash
Volatility Foundation Volatility Framework 2.6.1
Pid      Name      Command Time      Command
-----
1194 bash      2020-04-12 10:35:50 UTC+0000 ls /boot/
1194 bash      2020-04-12 10:35:50 UTC+0000 ps
1194 bash      2020-04-12 10:35:50 UTC+0000 su root
1194 bash      2020-04-12 10:35:50 UTC+0000 sudo passwd root
1194 bash      2020-04-12 10:35:50 UTC+0000 su root
1194 bash      2020-04-12 10:35:53 UTC+0000 cd Desktop/
1194 bash      2020-04-12 10:36:08 UTC+0000 git clone
1194 bash      2020-04-12 10:36:23 UTC+0000 ping github.com
1194 bash      2020-04-12 10:37:52 UTC+0000 git clone https://github.com/504ensicsLabs/Lime.git
1194 bash      2020-04-12 10:37:56 UTC+0000 ls
1194 bash      2020-04-12 10:38:15 UTC+0000 ls -R -l Lime
1194 bash      2020-04-12 10:38:31 UTC+0000 cd Lime/
1194 bash      2020-04-12 10:38:33 UTC+0000 ls
1194 bash      2020-04-12 10:38:39 UTC+0000 cd ..
1194 bash      2020-04-12 10:38:59 UTC+0000 ls -R -l Lime
1194 bash      2020-04-12 10:39:37 UTC+0000 sudo apt-get install make
1194 bash      2020-04-12 10:39:57 UTC+0000 sudo apt-get install build-essential
1194 bash      2020-04-12 10:40:08 UTC+0000 sudo apt-get install linux-headers
1194 bash      2020-04-12 10:43:19 UTC+0000 uname -r
1194 bash      2020-04-12 10:43:56 UTC+0000 apt-get install linux-header-$(uname -r)
1194 bash      2020-04-12 10:44:06 UTC+0000 sudo apt-get install linux-header-$(uname -r)
1194 bash      2020-04-12 10:44:21 UTC+0000 sudo apt-get install linux-header-generic
1194 bash      2020-04-12 10:44:41 UTC+0000 sudo apt-get install linux-headers-$(uname -r)
1194 bash      2020-04-12 10:45:03 UTC+0000 cd Lime/src/
1194 bash      2020-04-12 10:45:05 UTC+0000 make
1194 bash      2020-04-12 10:49:04 UTC+0000 sudo insmod ../lime-4.19.0-6-686-pae.ko "path=../raspbian.mem format=lime"
```

Ilustración 46. Visualización del historial de instrucciones ejecutadas a través del bash.

Este plugin es bastante útil. Supongamos un ejemplo en el que una persona intenta borrar documentos importantes en una empresa que contienen información útil. La utilización de esta herramienta permite saber exactamente qué instrucciones ha ejecutado dicho usuario en un determinado tiempo, pudiendo servir como prueba en un juzgado.

Todos los resultados de las instrucciones anteriores así como los volcados generados están disponibles a través del siguiente enlace:

<https://www.dropbox.com/sh/y3dfaat4p555plw/AADm31Blo7ezR8vkhGEFz6Sca?dl=0>

⁶⁸ HISTSIZE, <https://www.unix.com/unix-for-dummies-questions-and-answers/191301-histsize-histfilesize.html>

Memoria y objetos del kernel Linux

El primer comando que vamos a ver en esta sección se trata del llamado *linux_lsmmod*. Este plugin imprime la lista de módulos de kernel cargados (en el símbolo del sistema *modules* y en la lista *modules.list*). Existen dos modos de utilización principales:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_lsmmod`
- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_lsmmod -P`
 - La diferencia es que, con la opción *-P*, se imprimirá los parámetros con los cuales se han lanzado los módulos en cuestión.

```
root@osboxes: ~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_lsmmod -P
Volatility Foundation Volatility Framework 2.6.1
f94f1040 lime 24576
    compress=0
    timeout=1000
    digest=(null)
    localhostonly=0
    format=lime
    dio=0
    path=../TFM/raspbian.mem
f7cf9c40 cfg80211 495616
    cfg80211_disable_40mhz_24ghz=Y
    ieee80211_regdom=00
    bss_entries_limit=1000
f7c872c0 rfkill 20480
    default_state=1
    master_switch_mode=2
f7c7c180 8021q 28672
f7c74040 garp 16384
    garp_join_time=200
f7c6f080 stp 16384
f7c6a080 mrp 20480
    mrp_periodic_time=1000
    mrp_join_time=200
f7c54080 llc 16384
f7c642c0 binfmt_misc 20480
f7c59100 intel_powerclamp 16384
    window_size=2
    duration=24
```

Ilustración 47. Plugin *linux_lsmmod*.

El siguiente comando va ligado con el visto anteriormente. Se trata del plugin denominado *moddump*. Este plugin vuelca los módulos del kernel de Linux en el disco para su posterior inspección. Los archivos se nombran de acuerdo con su nombre *lkm* (*Loadable Kernel Module*), su dirección inicial en la memoria del kernel y finaliza con una extensión *.lkm*⁶⁹. Veamos un ejemplo de uso:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 -D ../linux_moddump/ linux_moddump`
 - Tan sólo debemos de indicar el directorio donde guardaremos los volcados con la opción *-D*. Por omisión se hará un volcado completo de todos los módulos del kernel cargados, si se quiere extraer uno en específico se deberá de señalar con la opción *-r*.

Debido a la gran cantidad de módulos activos, se ha hecho un volcado de ejemplo del módulo de LiME, usado para general el volcado de memoria que estamos trabajando, disponible a través del siguiente enlace:

<https://www.dropbox.com/sh/q3mpjau8b36nvyq/AAAlfu32mTOq4MiQ8IKcsgVBa?dl=0>

⁶⁹ LKM, https://en.wikipedia.org/wiki/Loadable_kernel_module

Por último, en relación con los objetos del Kernel, utilizaremos el plugin *linux_tmpfs*. Este complemento enumera y recupera los sistemas de archivos *tmpfs* de la memoria. Esto es muy útil en investigaciones forenses, ya que estos sistemas de archivos nunca se escriben en el disco y los atacantes aprovechan este hecho para ocultar sus datos en lugares como */dev/shm*. Primeramente, deberemos enumerar los sistemas de ficheros, para posteriormente elegir que directorios recuperar:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 -L linux_tmpfs`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 -L linux_tmpfs
Volatility Foundation Volatility Framework 2.6.1
1 -> /sys/fs/cgroup
2 -> /dev
3 -> /run
4 -> /run/lock
5 -> /dev/shm
6 -> /run/user/1000
```

Ilustración 48. Sistema de ficheros tmpfs.

Como se puede ver en la imagen anterior, se muestran los diferentes directorios *tmpfs* que se han recuperado, identificado cada uno con un ID. Con la misma herramienta y teniendo en cuenta los diferentes identificadores, podemos recuperar dichos directorios, en caso de que tengan información. Para ello, ejecutamos lo siguiente:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_tmpfs -S 2 -D ../tmpfs`
 - Con la opción *-S* indicamos que directorio volcaremos y con la opción *-D* la ruta donde almacenaremos la información generada.

Todas las salidas de los comandos utilizados anteriormente, así como los ficheros *.lkm* generados están disponibles en el siguiente enlace:

<https://www.dropbox.com/sh/q3mpjau8b36nvyq/AAAlfu32mTOq4MiQ8IKcsgVBa?dl=0>

Networking

El primer comando que veremos en esta sección se trata de *linux_arp*. Tal y como se deduce de su nombre, nos permite visualizar la tabla ARP:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_arp`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_arp
Volatility Foundation Volatility Framework 2.6.1
[224.0.0.22] at 01:00:5e:00:00:16 on eth0
[224.0.0.251] at 01:00:5e:00:00:fb on eth0
[10.0.2.2] at 52:54:00:12:35:02 on eth0
[ff02::2] at 33:33:00:00:00:02 on eth0
[ff02::16] at 33:33:00:00:00:16 on eth0
[ff02::fb] at 33:33:00:00:00:fb on eth0
[ff02::1:ffbd:6a1e] at 33:33:ff:bd:6a:1e on eth0
```

Ilustración 49. Visualización de la tabla ARP.

El siguiente comando que vamos a ejecutar se trata de un comando bastante importante y que nos ofrece información valiosa. Se trata del comando *linux_ifconfig*. Tal y como podemos entender con su nombre, similar al comando de Linux *ifconfig*, este plugin imprime la

información de la interfaz activa, incluidas las direcciones IP, el nombre de la interfaz, la dirección MAC y si la NIC está en modo promiscuo o no (*sniffing*):

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_ifconfig`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_ifconfig
Volatility Foundation Volatility Framework 2.6.1
Interface      IP Address      MAC Address      Promiscuous Mode
-----
lo             127.0.0.1       00:00:00:00:00:00 False
eth0           10.0.2.15       08:00:27:df:de:ed False
lo             127.0.0.1       _ 00:00:00:00:00:00 False
```

Ilustración 50. Visualización de las interfaces de red con el plugin `linux_ifconfig`.

Anteriormente, en las versiones anteriores a las 3.6 de Volatility, existía un comando que nos devolvía los datos de la caché de la tabla de enrutamiento, con lo que nos permitía demostrar con qué sistemas se comunicó una máquina en el pasado. Dicho comando era el `linux_route_cache`. Sin embargo, este plugin permitía realizar ataques de denegación de servicio de manera sencilla con un mal uso de la herramienta, por esa razón se decidió por eliminar dicha herramienta del kit de Volatility, tal y como explica David S. Miller en el repositorio de [Volatility](#).

Existen dos herramientas que pueden resultar conocidas por sus nombres y que se suelen utilizar a la vez en prácticamente cualquier análisis. Dichos plugins son `linux_netstat` y `linux_netscan`. Por una parte, `linux_netstat` imita el comando `netstat` de los sistemas Linux. Aprovecha la funcionalidad `linux_lsof` para enumerar los puertos abiertos en cada proceso, es decir, comprueba si existen sockets abiertos. Por otra parte, `netscan`, comprueba las estructuras de conexión de la red. Comprobemos como funciona sobre el volcado:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_netstat`
- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_netscan`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_netstat
Volatility Foundation Volatility Framework 2.6.1
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_netscan
Volatility Foundation Volatility Framework 2.6.1
f3421440 TCP      127.0.0.1      : 25 0.0.0.0      : 0 LISTEN
f3423600 TCP      10.0.2.15     :41016 140.82.118.3   : 443 CLOSE
f342a300 TCP      ::1           : 25 ::           : 0 LISTEN
f3485c00 TCP      ::            : 0 ::            : 0 CLOSE
f4054380 TCP      ::            : 6 ::            : 0 CLOSE
f555f500 TCP      ::            : 6 ::            : 0 CLOSE
```

Ilustración 51. `netscan` y `netstat` sobre el volcado

Información del sistema

El primer comando que vamos a ver en esta sección se trata del plugin `linux_cpuinfo`, que tal y como su nombre indica, nos muestra la información de la CPU del volcado. La utilización es muy sencilla:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_cpuinfo`

TÉCNICAS Y HERRAMIENTAS PARA EL ANÁLISIS DE DEBILIDADES EN VOLCADOS DE MEMORIA RAM DE SISTEMAS BASADOS EN LINUX

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_cpuinfo
Volatility Foundation Volatility Framework 2.6.1
Processor      Vendor      Model
-----
0              GenuineIntel  Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
```

Ilustración 52. Visualización de la información de la CPU.

El siguiente plugin se trata de *linux_dmesg*. Este complemento descarga el búfer de depuración del núcleo:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_dmesg`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_dmesg
Volatility Foundation Volatility Framework 2.6.1
[0.0] Linux version 4.19.0-6-686-pae (debian-kernel@lists.debian.org) (gcc version 8.3.0 (Debian 8.3.0-6)) #1 SMP Debian 4.19.67-2+deb10u1 (2019-09-20)
[0.0] -----[ cut here ]-----
[0.0] XSAVE consistency problem, dumping leaves
[0.0] WARNING: CPU: 0 PID: 0 at arch/x86/kernel/fpu/xstate.c:614 fpu__init_system_xstate+0x424/0x7ea
[0.0] Modules linked in:
[0.0] CPU: 0 PID: 0 Comm: swapper Not tainted 4.19.0-6-686-pae #1 Debian 4.19.67-2+deb10u1
[0.0] EIP: fpu__init_system_xstate+0x424/0x7ea
[0.0] Code: fb 0a 0f 85 8f fd ff ff 3b 35 48 13 a5 d8 74 2b 80 3d 8b 02 93 d8 00 75 14 c6 05 8b 02 93 d8 01 68 b0 76 7c d8 e8 df 28 70 ff <0f> 0b 5f 83
[0.0] EAX: 00000029 EBX: 0000000a ECX: d8a5fde4 EDX: 00000000
[0.0] ESI: 00000340 EDI: 00000100 EBP: d88abedc ESP: d88abea4
[0.0] DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068 EFLAGS: 00210086
[0.0] CR0: 80050033 CR2: 00000000 CR3: 18a48000 CR4: 00040620
[0.0] Call Trace:
[0.0] fpu__init_system+0x1fc/0x244
[0.0] ? early_init_intel+0x16c/0x370
[0.0] early_cpu_init+0x245/0x267
[0.0] setup_arch+0x12f/0xc8f
[0.0] ? vprintk_default+0x17/0x20
[0.0] ? vprintk_func+0x3d/0xb7
[0.0] start_kernel+0x5d/0x45f
[0.0] ? early_idt_handler_common+0x44/0x44
[0.0] i386_start_kernel+0xac/0xb0
[0.0] startup_32_smp+0x164/0x168
[0.0] random: get_random_bytes called from print_oops_end_marker+0x2c/0x50 with crng_init=0
[0.0] ---[ end trace ab4fabf4af18fb24 ]---
[0.0] CPUID[0d, 00]: eax=00000007 ebx=00000440 ecx=00000440 edx=00000000
[0.0] CPUID[0d, 01]: eax=00000000 ebx=00000440 ecx=00000000 edx=00000000
[0.0] CPUID[0d, 02]: eax=00000100 ebx=00000740 ecx=00000000 edx=00000000
```

Ilustración 53. Plugin dmesg.

EL siguiente plugin es *linux_iomem*, que tal y como se puede deducir, muestra las direcciones físicas reservadas para dispositivos de E/S como PCI y memoria de tarjeta de video. Se ejecuta de la siguiente manera:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_iomem`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_iomem
Volatility Foundation Volatility Framework 2.6.1
Reserved                0x0                0xFFF
System RAM              0x1000             0x9FBFF
Reserved                0x9FC00           0x9FFFF
PCI Bus 0000:00         0xA0000         0xBFFFF
Video RAM area          0xA0000         0xBFFFF
Video ROM               0xC0000         0xC7FFF
Adapter ROM             0xE2000         0xE2FFF
Reserved                0xF0000         0xFFFFF
System RAM              0xF0000         0xFFFFF
System RAM              0x100000          0x3FFFFFFF
Kernel code             0x18000000       0x1869E507
Kernel data             0x1869E508       0x1894F37F
Kernel bss              0x18A44000       0x18AB4FFF
ACPI Tables             0x3FFF0000       0x3FFFFFFF
PCI Bus 0000:00         0x40000000       0xFFDFFFFF
0000:00:02.0            0xE0000000       0xE0FFFFFFF
0000:00:03.0            0xF0000000       0xF001FFFFF
e1000                   0xF0000000       0xF001FFFFF
0000:00:04.0            0xF0400000       0xF07FFFFF
vboxguest               0xF0400000       0xF07FFFFF
0000:00:04.0            0xF0800000       0xF0803FFF
0000:00:06.0            0xF0804000       0xF0804FFF
ohci_hcd                 0xF0804000       0xF0804FFF
0000:00:0b.0            0xF0805000       0xF0805FFF
ehci_hcd                 0xF0805000       0xF0805FFF
0000:00:0d.0            0xF0806000       0xF0807FFF
ahci                     0xF0806000       0xF0807FFF
Reserved                0xFEC00000       0xFEC00FFF
Local APIC              0xFEE00000       0xFEE00FFF
Reserved                0xFEE00000       0xFEE00FFF
Reserved                0xFFFC0000       0xFFFFFFF
```

Ilustración 54. Direcciones reservadas para dispositivos de entrada y salida.

El último plugin que convendría destacar es el denominado *linux_mount*, el cual imita la salida de */proc/mounts* en un sistema Linux en ejecución. Para cada punto de montaje, imprime los indicadores, la fuente montada (unidad, recurso compartido de red, etc.) y el directorio en el que está montado:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_mount`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_mount
Volatility Foundation Volatility Framework 2.6.1
tmpfs /sys/fs/cgroup ro,nosuid,nodev,noexec
/dev/sda4 /home rw,relatime
sysfs /sys rw,relatime,nosuid,nodev,noexec
cgroup2 /sys/fs/cgroup/unified rw,relatime,nosuid,nodev,noexec
tmpfs /dev ro,nosuid,noexec
gvfsd-fuse /run/user/1000/gvfs rw,relatime,nosuid,nodev
/dev/sda1 /var/tmp rw,relatime
devpts /dev/pts rw,relatime,nosuid,noexec
debugfs /sys/kernel/debug rw,relatime
fusectl /sys/fs/fuse/connections rw,relatime
tmpfs /run rw,relatime,nosuid,noexec
cgroup /sys/fs/cgroup/cpuset rw,relatime,nosuid,nodev,noexec
cgroup /sys/fs/cgroup/rdma rw,relatime,nosuid,nodev,noexec
proc /proc rw,relatime,nosuid,nodev,noexec
udev /dev rw,relatime,nosuid
cgroup /sys/fs/cgroup/blkio rw,relatime,nosuid,nodev,noexec
sunrpc /run/rpc_pipefs rw,relatime
mqueue /dev/mqueue rw,relatime
tmpfs /run/lock rw,relatime,nosuid,nodev,noexec
tmpfs /dev/shm rw,nosuid,nodev
binfmt_misc /proc/sys/fs/binfmt_misc rw,relatime
cgroup /sys/fs/cgroup/net_cls,net_prio rw,relatime,nosuid,nodev,noexec
systemd-1 /proc/sys/fs/binfmt_misc rw,relatime
cgroup /sys/fs/cgroup/freezer rw,relatime,nosuid,nodev,noexec
bpf /sys/fs/bpf rw,relatime,nosuid,nodev
tmpfs /run/user/1000 rw,relatime,nosuid,nodev
cgroup /sys/fs/cgroup/systemd rw,relatime,nosuid,nodev,noexec
pstore /sys/fs/pstore rw,relatime,nosuid,nodev,noexec
cgroup /sys/fs/cgroup/devices rw,relatime,nosuid,nodev,noexec
/dev/sda2 /boot rw,relatime
securityfs /sys/kernel/security rw,relatime,nosuid,nodev,noexec
cgroup /sys/fs/cgroup/perf_event rw,relatime,nosuid,nodev,noexec
hugetlbfs /dev/hugepages rw,relatime
cgroup /sys/fs/cgroup/cpu,cpuacct rw,relatime,nosuid,nodev,noexec
cgroup /sys/fs/cgroup/pids rw,relatime,nosuid,nodev,noexec
cgroup /sys/fs/cgroup/memory rw,relatime,nosuid,nodev,noexec
```

Ilustración 55. Plugin *linux_mount*.

Todos los resultados de los comandos antes vistos se pueden acceder a través del siguiente enlace: <https://www.dropbox.com/sh/7vo4uk783eb3plq/AAB4jIHMZLLTiRA1VDb16ZRHa?dl=0>

Detección de rootkits

El primero de los comandos para la detección de rootkits será el llamado *linux_check_ainfo*. Este complemento recorre las estructuras de *file_operations* y *sequence_operations* de todas las estructuras de protocolo UDP y TCP, incluidos *tcp6_seq_ainfo*, *tcp4_seq_ainfo*, *udplite6_seq_ainfo*, *udp6_seq_ainfo*, *udplite4_seq_ainfo* y *udp4_seq_ainfo*. Esto detecta cualquier manipulación de las estructuras antes vistas. Por ejemplo:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_ainfo`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_ainfo
Volatility Foundation Volatility Framework 2.6.1
Symbol Name Member Address
```

Ilustración 56. Plugin *linux_check_ainfo* para la detección de rootkits.

El siguiente plugin se trata del complemento denominado *linux_check_tty*. Este complemento detecta uno de los métodos de *keyloggers* a nivel de núcleo descritos en "[Bridging the Semantic Gap to Mitigate Kernel-level Keyloggers](#)". Funciona comprobando el puntero de la función *receive_buf* para cada controlador *tty* activo en el sistema. Si el puntero de la función no está enganchado/utilizado, se imprime su nombre de símbolo; de lo contrario, se imprime "HOOKED":

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_tty`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_tty
Volatility Foundation Volatility Framework 2.6.1
Name      Address      Symbol
-----
tty1      0xd846ee70  n_tty_receive_buf
tty6      0xd846ee70  n_tty_receive_buf
tty7      0xd846ee70  n_tty_receive_buf
```

Ilustración 57. *linux_check_tty* plugin.

El siguiente plugin se trata del llamado *linux_check_creds*. Este complemento detecta rootkits que tienen privilegios elevados (*root*) utilizando técnicas DKOM⁷⁰, explicadas detalladamente en el apartado “Detección de rootkits”. Para utilizar dicho plugin sobre el volcado de memoria tan sólo hay que ejecutar lo siguiente:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_creds`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_creds
Volatility Foundation Volatility Framework 2.6.1
PIDs
-----
```

Ilustración 58. Búsqueda de rootkits utilizando técnicas DKOM.

El siguiente plugin es el llamado *linux_check_fop*. Este complemento enumera el sistema de archivos */proc* y todos los archivos abiertos y verifica que cada miembro de cada estructura *file_operations* sea válido (válido significa que el puntero de la función está en el kernel o en un módulo de kernel cargado conocido, y no oculto). Para ejecutarlo tan sólo basta con la siguiente sentencia:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_fop`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_fop
Volatility Foundation Volatility Framework 2.6.1
Symbol Name      Member      Address
```

Ilustración 59. Utilización del plugin *linux_check_fop*.

El siguiente plugin se trata del denominado *linux_check_idt*. Este complemento enumera las direcciones y los símbolos de la tabla del descriptor de interrupciones (IDT). Si alguna de las entradas está “enganchada” o siendo utilizadas por rootkits, aparecerá "HOOKED" en la columna de la derecha en lugar del nombre del símbolo:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_idt`

⁷⁰ DKOM, https://en.wikipedia.org/wiki/Direct_kernel_object_manipulation

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_idt
Volatility Foundation Volatility Framework 2.6.1
-----
Index Address Symbol
-----
0x0 0xd869c34c divide_error
0x1 0xd869c8cc debug
0x2 0xd869c8dc nmi
0x3 0xd869ca74 int3
0x4 0xd869c2cc overflow
0x5 0xd869c2dc bounds
0x6 0xd869c2ec invalid_op
0x7 0xd869c2b8 device_not_available
0x9 0xd869c2fc coprocessor_segment_overnrun
0xa 0xd869c30c invalid_TSS
0xb 0xd869c31c segment_not_present
0xc 0xd869c32c stack_segment
0xd 0xd869cb50 general_protection
0xe 0xd869c7a0 page_fault
0xf 0xd869c36c spurious_interrupt_bug
0x10 0xd869c298 coprocessor_error
0x11 0xd869c33c alignment_check
0x12 0xd869c35c machine_check
0x13 0xd869c2a8 simd_coprocessor_error
0x80 0xd869aa40 entry_INT80_32
-----
```

Ilustración 60. Uso del plugin `linux_check_idt`.

El penúltimo plugin se trata del llamado `linux_check_syscall`. Este complemento imprime las tablas de llamadas del sistema (`syscall`) y comprueba las funciones “enganchadas”. Para sistemas de 64 bits, imprime tanto la tabla de 32 bits como la de 64 bits. Si una función está “enganchada” por rootkits, se verá “HOOKED” en la salida, de lo contrario se verá el nombre de la función de llamada del sistema (`syscall`):

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_syscall`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_syscall
Volatility Foundation Volatility Framework 2.6.1
Table Name Index System Call Handler Address Symbol
-----
32bit 0 0x00000d807bbd0 sys_restart_syscall
32bit 1 0x00000d8074a70 __se_sys_exit
32bit 2 0x00000d806f020 sys_fork
32bit 3 0x00000d8225910 sys_read
32bit 4 0x00000d8225a00 sys_write
32bit 5 0x00000d822670 sys_open
32bit 6 0x00000d8220b80 __se_sys_close
32bit 7 0x00000d8074d30 __se_sys_waitpid
32bit 8 0x00000d8226b0 __se_sys_creat
32bit 9 0x00000d82365d0 __se_sys_link
32bit 10 0x00000d8236160 __se_sys_unlink
32bit 11 0x00000d822c8d0 sys_execve
32bit 12 0x00000d8221e00 sys_chdir
32bit 13 0x00000d80e3cb0 sys_time
32bit 14 0x00000d8235b30 sys_mknod
32bit 15 0x00000d822090 __se_sys_chmod
32bit 16 0x00000d80ff030 __se_sys_lchown16
32bit 17 0x00000d808e450 sys_pkey_free
32bit 18 0x00000d8229900 __se_sys_stat
-----
```

Ilustración 61. Ejemplo del plugin `linux_check_syscall`.

Por último, toca hablar del plugin llamado `linux_check_modules`. Este complemento encuentra rootkits que se separan de la lista de módulos, pero no de `sysfs`. No se ha encontrado un rootkit que realmente se elimine de `sysfs`, por lo que, en un sistema real, suelen estar ocultos de `lsmod` y `/proc/modules`, pero aún se pueden encontrar en `/sys/modules`:

- `python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_modules`

```
root@osboxes:~/Desktop/volatility# python vol.py -f ../TFM/raspbian.mem --profile=LinuxRaspbian4-19x86 linux_check_modules
Volatility Foundation Volatility Framework 2.6.1
Module Address Core Address Init Address Module Name
-----
```

Ilustración 62. Utilizando el plugin `linux_check_modules`.

Todos los resultados al completo están disponibles a través del siguiente enlace:

https://www.dropbox.com/sh/yd406g3fcwjj9yr/AAAF2zvI_pZQI_IHRvu9wotoa?dl=0