Detección de fileless malware utilizando herramientas de Endpoint Detection and Response

Lautaro Lecumberry
Director: Dr. Michael Denzel
Profesor representante: Prof. Dr. Nicolás Wolovick

Licenciatura en Ciencias de la Computación
Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba, 2023

Abstract

Damage caused by malware has been ramping up in the last five years [1]. One kind of malware is fileless malware, which increased 900 percent in 2020, and it is expected to be half of the attacks against enterprise environments in 2022 [8] [9]. To detect fileless malware, we matched code segments from the executables loaded into Random Access Memory to the original executable file stored on hard disk, using Endpoint Detection and Response tools to implement it. Furthermore, we tested the technique against real malware families, resulting in a detection rate of 77.78 percent, with a sensitivity rate of 92.11 percent. In summary, we present a technique to detect fileless malware, and the results of the testing phase sound promising.

Resumen

Los daños causados por el malware se han disparado en los últimos cinco años [1]. Un tipo de malware es el fileless malware, que aumentó un 900 por ciento en 2020, y se espera que sea la mitad de los ataques contra entornos empresariales en 2022 [8] [9]. Para detectar el fileless malware, comparamos los segmentos de código de los ejecutables cargados en la Random Access Memory con el archivo ejecutable original almacenado en el disco duro, utilizando para ello herramientas de Endpoint Detection and Response. Además, probamos la técnica con familias de malware reales, obteniendo una tasa de detección del 77,78 por ciento, con una tasa de sensibilidad del 92,11 por ciento. En resumen, presentamos una técnica para detectar fileless malware, y los resultados de la fase de pruebas parecen prometedores.

Índice general

Índice de figuras

Índice de cuadros

Lista de listings

Capítulo 1

Introducción

Los daños causados por los ciberataques han ido en aumento, incrementándose en los últimos 5 años. Sólo la ciberdelincuencia denunciada a Internet Crime Complaint Center (IC3) ha causado unos daños estimados en 7.000 millones de dólares estadounidenses durante 2021 [1]. Además, incluso la vida de las personas está en riesgo debido a algunos problemas causados por los ciberataques [2].

Otro objetivo que va en aumento son las infraestructuras críticas [3]. Infraestructuras como oleoductos y redes eléctricas son cada vez más susceptibles de ser atacadas con el objetivo de interrumpir la vida cotidiana de los objetivos y causarles daños económicos. [5].

Los atacantes buscan constantemente nuevas técnicas para mejorar los ataques, tanto para conseguir nuevos mecanismos de infección como para pasar desapercibidos después. Un ejemplo del uso de técnicas inéditas para lograr un ciberataque con éxito es el ataque Stuxnet [6].

Uno de los métodos posibles son los ataques sin archivos. Estos ataques son los que no se basan al 100 por ciento en archivos, lo que les da la ventaja de ser más difíciles de detectar, ya que no hay ningún archivo que permita descubrir el éxito del ataque. Teniendo en cuenta que los ataques sin archivos aumentaron un 900 por ciento durante 2020 y que se prevé que constituyan el 50 por ciento de todos los ataques contra entornos empresariales en 2022, es extremadamente importante detectar la amenaza [8]. [9].

Los ataques basados en memoria también hacen inútil el enfoque tradicional de los antivirus, ya que consiste en comparar el hash del archivo de malware con todos los hashes de malware ya conocido almacenados en una base de datos, pero no existe ningún archivo para calcular el hash [7]. Para solucionar el problema anterior, se pueden utilizar técnicas relacionadas con Random Access Memory (RAM) forense [10].

Otra herramienta disponible son los EDR. Son herramientas que pueden realizar una amplia gama de funciones relacionadas con la ciencia forense en diferentes dispositivos. Algunas de las funciones están relacionadas con la memoria forense.

En este proyecto se presentará una detección de malware sin archivos. Lo anterior se realizará en base a técnicas forenses de memoria utilizando EDRs con el fin de obtener la información necesaria.

1.1. Airbus

Este proyecto de investigación se ha realizado durante una pasantía en el Computer Security Incident Response Team (CSIRT) de Airbus Protect GmbH. Aunque el proyecto se llevó a cabo en este contexto, el objetivo principal era académico y todas las decisiones tomadas estaban encaminadas a obtener el mejor resultado posible desde el punto de vista de la investigación.

El equipo estaba formado por un grupo de incident responders y un grupo de penetration testers.

Capítulo 2

Fundamentos y revisión bibliografica

2.1. Ciencias forenses

Según el Diccionario de Cambridge, los forenses son los "métodos científicos para resolver crímenes, que implican examinar objetos o sustancias relacionados con un delito" [12].

2.1.1. Digital Forensics

Teniendo en cuenta la ciencia forense clásica, es posible trasladar el concepto al mundo digital y definir el ámbito de digital forensics (ó ciencia forense digital) como los métodos científicos que implican el examen de objetos digitales. Como era de esperar, es la idea que subyace a la definición de SysAdmin, Audit, Network and Security (SANS) de digital forensics como "la disciplina forense que se ocupa de la preservación, examen y análisis de las pruebas digitales"[13].

Desde la perspectiva temporal, el primer lugar donde se ha aplicado la ciencia forense digital ha sido en los dispositivos de almacenamiento. En este enfoque, primero se desconecta la máquina, segundo se crea una imagen del dispositivo de almacenamiento, es decir, se crea una copia bit a bit del mismo, y tercero se analiza la imagen. Teniendo en cuenta que el análisis forense implica métodos científicos, los pasos realizados durante la fase de análisis deben ser reproducibles por otra persona, lo que también es importante si los resultados del análisis forman parte de un proceso judicial [14].

Uno de los inconvenientes de este enfoque es que el contexto de la computadora se pierde en el momento en que se apaga, por lo que se pierde información, como las conexiones de red activas y los procesos en ejecución, lo que podría tener un gran impacto en el resultado de la investigación.

Otro inconveniente es la falta de escalabilidad. Puede tardar mucho tiempo en copiar un dispositivo de almacenamiento de, por ejemplo, 1 Terabyte (TB). Después de copiar el contenido, también es difícil analizar una imagen de ese tamaño. Teniendo en cuenta que hoy en día un dispositivo de almacenamiento de 1 TB es algo posible de tener en un ordenador personal y que, con toda seguridad, esas cifras seguirán creciendo, es probable que este problema se haga cada vez más grande [18].

Además, no es improbable la presencia de dispositivos Network Attached Storage (NAS) con varios TB de almacenamiento, lo que amplía masivamente el volumen de datos de análisis [19].

2.1.2. Live forensics

El concepto de live forensics lo resume Adelstein [20] en el título de uno de sus artículos: "diagnosticar su sistema sin matarlo primero". Sin apagar el ordenador, la idea de este enfoque es obtener constantemente información de los dispositivos de almacenamiento, en lugar de crear una imagen a partir de ella y luego analizarlo usando el enfoque estático.

No matar la computadora tiene muchas consecuencias. Una de ellas es que se mantiene el contexto completo. Además, es posible acceder a la información volátil almacenada en RAM pero también a los dispositivos de almacenamiento no volátil, por lo que se dispone de más información.

Además, el autor afirma que otra ventaja es la posibilidad de obtener constantemente información adicional, lo que conduce a un enfoque más escalable porque no es necesario obtener toda la información a la vez. Incluso puede que no sea necesario reunir toda la información al principio, ya que es posible obtenerla cuando sea necesario.

Sin embargo, un inconveniente de esta técnica es que se pierde la reproducibilidad de la fase de análisis porque la fuente de la información cambia constantemente [18] [21].

2.2. Portable executable file format

El formato de archivo PE es un tipo de archivos que especifica la estructura de los archivos ejecutables y archivos objeto en toda la familia de sistemas operativos Microsoft Windows, de donde viene la palabra "portable" en el nombre, ya que no es para una arquitectura específica sino para toda la familia [22].

Este formato de archivo también se conoce como archivos Common Object File Format (COFF) y las extensiones más comunes son *.exe* y *.dll*. Mach-O en macOS y Executable and Linkable Format (ELF) en Linux son los formatos similares.

Como se muestra en las figuras 2.1 y 2.2, hay diferentes tipos de cabeceras presentes en este formato de archivo. La primera es la cabecera Disk Operating System (DOS), que es necesaria para que el archivo ejecutable funcione bajo Microsoft DOS, ya que si esta cabecera está presente, entonces se ejecuta el stub DOS en lugar del ejecutable real, pero, sin esta cabecera, producirá un error. Entonces, el stub DOS anteriormente mencionado

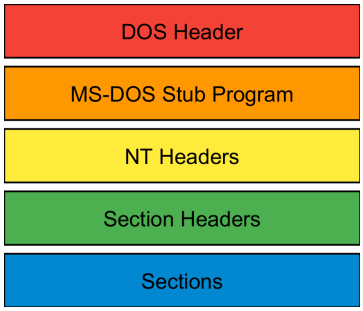

Figura 2.1: Estructura del Portable Executable (PE) file format.
Cada color significa una parte diferente del formato de archivo PE.

es una aplicación válida que es capaz de ejecutarse en Microsoft DOS pero sólo imprime "Este programa no puede ejecutarse en modo DOS" [23].

Después del stub DOS está el Rich Header, que es una estructura no documentada presente cuando el archivo ejecutable se compila utilizando Visual Studio y ha sido utilizado por los creadores de malware para fingir que otro grupo había desarrollado ciertas muestras de malware [24].

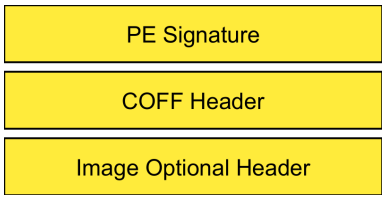

Figura 2.2: Subsecciones de NT header.
Todas las entradas son amarillas porque pertenecen a la parte de NT headers.

A continuación, están las cabeceras NT, que se dividen en tres partes. En primer lugar, la firma PE, que son los 4 bytes `PE\0\0` e identifica el archivo como un PE. La segunda es una cabecera estándar COFF, que tiene información relacionada con el tipo Central Processing Unit (CPU) donde se puede ejecutar el fichero, el número de secciones presentes en el fichero, información relacionada con los símbolos, el tamaño de la cabecera opcional, entre otras cosas. Por último, está la cabecera opcional que, de hecho, no es opcional: todos los archivos de imagen requieren una, pero no es necesaria en algunos otros archivos, como los archivos de objetos.

Los ocho primeros campos de la cabecera opcional son estándar para cada implementación de COFF y contiene el tamaño de la sección de código o la suma de ellas si hay más de una, tamaño de los datos inicializados y no inicializados, dirección del punto de entrada, que es la dirección de inicio de las imágenes del programa, entre otra información. También es un poco diferente si se trata de un formato PE32 o PE32+.

Después de los campos COFF, están los campos específicos de Windows, en los que se encuentra la dirección base de la imagen, por ejemplo, la dirección preferida para cargar la imagen en memoria, la alineación del archivo y las secciones cuando se cargan en memoria, el tamaño de la imagen tal y como se carga en memoria, el tamaño de todas las cabeceras combinadas, la suma de comprobación del archivo de imagen e información relacionada con las versiones del sistema operativo, la imagen y el subsistema, entre otros.

La última cabecera es la tabla de secciones, que tiene que estar inmediatamente después de la cabecera opcional porque no hay ningún puntero directo a esta sección en la cabecera del archivo. Tiene una entrada por cada sección del fichero y cada entrada tiene el nombre de la sección, la dirección virtual de la sección relativa a la base de la imagen cuando la sección se carga en memoria, información relacionada con la reubicación de la entrada, los permisos que debe tener la sección cuando se carga en memoria, el tamaño de la sección cuando se carga en memoria virtual y el tamaño de los datos brutos en disco, entre otros.

Figura 2.3: Secciones de Portable Executable (PE). Todas las entradas son azules porque pertenecen a la parte de las secciones.

Como se ve en la Fig. 2.3, hay un número indefinido de secciones en un fichero PE. Estas secciones pueden tener una combinación diferente de características en función de sus necesidades sin seguir ninguna regla especial. Sin embargo, hay algunas secciones especiales que son comunes entre los ficheros PE. Algunas de ellas son las siguientes:

- **.bss**: this section contains uninitialized data.
- **.data**: this section contains initialized data.
- **.edata**: this section contains the export tables.
- **.idata**: this section contains the import tables.
- **.pdata**: this section contains exception information.
- **.rdata**: this section contains read-only initialized data.
- **.reloc**: this section contains information related to image relocations.
- **.rsrc**: this section contains the resources used by the program, including images or embedded binaries.

- **.text**: this section contains the executable code, and it is the only special section with executable permissions.
- **.tls**: standing for thread local storage, provides storage for the program's executing threads.

2.3. Memoria

Los sistemas operativos modernos proporcionan una abstracción de la memoria a los procesos, que se denomina memoria virtual. La virtualización de la memoria es la técnica de utilizar memoria virtual y tiene como objetivo la eficiencia y la protección, ya que los procesos no interactúan con la memoria física. En su lugar, interactúan con la abstracción que les presenta el sistema operativo y el sistema operativo es el que tiene el control total sobre la memoria física [39].

El espacio de direcciones es el nombre de esta abstracción y cada proceso tiene un espacio de direcciones completo para sí mismo. Almacena todo el estado de la memoria, como el código del programa, y el tamaño del espacio de direcciones no está relacionado con la cantidad de RAM física del ordenador. Como cada proceso tiene el suyo propio, también sirve para fines de aislamiento, ya que no comparte directamente las direcciones de memoria.

2.3.1. Páginas de memoria

Una página de memoria es un fragmento contiguo de memoria virtual con un tamaño fijo. Es la división más pequeña de la memoria virtual y la división equivalente de la memoria física se denomina marco de página [39].

Como a veces la RAM es un recurso limitado, el sistema operativo dispone de mecanismos para reducir el uso de la memoria física. Estos mecanismos son el page swapping y el demand paging [14].

El intercambio de páginas es un mecanismo por el cual el sistema operativo almacena parte de la información que debería estar en memoria en otras fuentes, normalmente en disco.

La paginación bajo demanda es un mecanismo en el que la información no se carga en memoria hasta que dicha información se escribe o se lee.

2.3.2. Segmentación de memoria

En el contexto de la gestión de memoria, un segmento es una porción contigua de memoria virtual de una longitud no específica, por ejemplo el código del programa, la pila y el montón de un determinado proceso. La segmentación es la técnica de dividir la memoria en segmentos. Cada segmento consiste en una o más páginas de memoria.

Esto se utiliza con fines de seguridad, ya que se sabe qué segmentos se están utilizando y es posible dar diferentes permisos a los distintos segmentos. Los permisos posibles son lectura, escritura, ejecución y una combinación de ellos. Con ellos, el sistema operativo puede restringir comportamientos no deseados como ejecutar la entrada del usuario almacenada en la pila porque esa sección sólo tiene permisos de lectura y escritura, por lo que no es posible ejecutar esa sección [39].

Cuando más de un proceso mapea exactamente el mismo segmento, por ejemplo, un segmento de código de biblioteca, el sistema operativo puede mapear todos los segmentos en la memoria virtual de cada proceso al mismo segmento en la memoria física. Mediante esta técnica, los segmentos se comparten entre procesos por motivos de eficiencia. Sin embargo, esto no es posible con cualquier segmento, sólo con aquellos que no serían modificados por los diferentes procesos y no exponen información privada.

2.3.3. Carga en memoria de archivos de formato Portable Executable

Para ser accedidas y ejecutadas en tiempo de ejecución, todas las secciones PE mencionadas en la sección 2.2 deben estar presentes en memoria, con la adición de una sección de pila para cada hilo en ejecución del proceso y una sección de montón para todo el proceso. Debido a la segmentación, existe la posibilidad de tener diferentes permisos para cada una de las secciones del fichero PE.

Aunque cada sección puede tener cualquier permisos, las secciones especiales tiene algunas opciones predefinidas como se muestra en la Tabla 2.1 [22].

Nombre	Mem lectura	Mem escritura	Mem ejecutable
.bss	✓	✓	
.data	✓	✓	
.edata	✓		
.idata	✓	✓	
.pdata	✓		
.rdata	✓		
.reloc	✓		
.rsrc	✓		
.text	✓		✓
.tls	✓	✓	

Cuadro 2.1: Permisos de las secciones Portable Executable (PE) cuando se mapean en memoria.

2.3.4. Estructuras de datos del sistema operativo

Como el sistema operativo necesita información sobre el estado actual del ordenador, es necesario almacenar dicha información en algún tipo de estructura de datos, que se almacenan en memoria.

En la familia de sistemas operativos Windows una de ellas es el Virtual Address Descriptor (VAD), que es una estructura de datos en forma de árbol que describe los segmentos de memoria virtual reservados por cada proceso, e información relacionada con ellos como la protección de la memoria.

```
typedef struct _FILE_OBJECT {
  CSHORT                          Type;
  CSHORT                          Size;
  PDEVICE_OBJECT                  DeviceObject;
  ...
  ULONG                           Flags;
  UNICODE_STRING                  FileName;
  LARGE_INTEGER                   CurrentByteOffset;
  ...
} FILE_OBJECT, *PFILE_OBJECT;
```

Listing 2.1: `_FILE_OBJECT` struct.

Cada VAD entrada se crea cuando un proceso asigna memoria utilizando `VirtualAlloc`, no espera hasta que el proceso real referencie ese segmento de memoria. Si un archivo se asigna a este segmento, se puede acceder a una estructura de datos `_FILE_OBJECT`, desde la que es posible obtener el nombre del archivo asignado al segmento [34]. [35]. La estructura de datos `_FILE_OBJECT` representa un archivo abierto, dispositivo, directorio o volumen [41].

Otra estructura de datos importante del núcleo es la estructura `EPROCESS`, donde se almacena la información sobre los procesos en ejecución [42].

```
typedef struct _EPROCESS {
  KPROCESS Pcb;
  EX_PUSH_LOCK ProcessLock;
  LARGE_INTEGER CreateTime;
  LARGE_INTEGER ExitTime;
  EX_RUNDOWN_REF RundownProtect;
  PVOID UniqueProcessId;
  LIST_ENTRY ActiveProcessLinks;
  ULONG QuotaUsage[3];
  ULONG QuotaPeak[3];
  ULONG CommitCharge;
  ULONG PeakVirtualSize;
  ULONG VirtualSize;
  ...
  ULONG ActiveThreads;
  ULONG ImagePathHash;
  ULONG DefaultHardErrorProcessing;
  LONG LastThreadExitStatus;
  PPEB Peb;
  ...
  MM_AVL_TABLE VadRoot;
  ULONG Cookie;
  ALPC_PROCESS_CONTEXT AlpcContext;
} EPROCESS, *PEPROCESS;
```

Listing 2.2: `EPROCESS` struct.

2.3.5. Memory forensics

Se puede añadir el contenido del RAM del ordenador para ampliar el alcance de la ciencia forense digital tradicional. Añadiendo esa información, sería posible acceder a más información sobre el estado actual de la máquina.

Siguiendo los pasos de las técnicas forenses digitales tradicionales sobre dispositivos de almacenamiento, el primer paso es obtener una imagen de la memoria física en bruto y luego analizarla para recuperar información de alto nivel a partir de los bytes de la imagen. Existen frameworks que ayudan en la fase de análisis, ya que las imágenes pueden llegar a ser excesivamente grandes para hacerlo sin ayuda. Algunos de estos marcos son Volatility y rekall.

Aunque estos enfoques pueden parecer extremadamente fiables, existen algunos problemas relacionados con la fase de adquisición. Uno de ellos es el page smearing, que es la inconsistencia entre el estado de la memoria descrito por las tablas de páginas y el contenido real de la memoria. Esto ocurre debido a la diferencia de tiempo entre la adquisición de las diferentes secciones de la memoria [14].

Otro problema es que la imagen se realiza sobre el contenido de la memoria física, por

lo que todas las páginas intercambiadas y las páginas de demanda no se incluyen en la imagen, lo que puede afectar al análisis, ya que falta información.

2.3.6. Live memory forensics

Live memory forensics es el resultado de combinar live y memory forensics. La idea es acceder al contenido de la memoria mientras el ordenador sigue funcionando en lugar de crear una imagen de la memoria física [48].

Al no apagar el ordenador, el acceso a páginas intercambiadas y de demanda ya no es un problema porque las fuentes externas donde estas páginas pueden ser almacenadas también son accesibles [20].

También se evita el emborronamiento de páginas porque ya no es necesario crear una imagen, por lo que no hay problema de copiar las diferentes páginas en diferentes momentos.

Sin embargo, como ocurre en los análisis forenses en vivo, los resultados ya no son reproducibles.

2.4. Herramientas de Endpoint Detection and Response (EDR)

Una herramienta Endpoint Detection and Response (EDR) es un software que supervisa las actividades de los hosts finales en tiempo real con el objetivo de emitir una alerta si se detecta un comportamiento malicioso. Los datos recogidos de cada punto final pueden enviarse a una base de datos centralizada donde se correlaciona toda la información procedente de múltiples puntos finales.

Lo que hacen los antivirus tradicionales es comparar la firma digital de los archivos almacenados en el ordenador con las sumas de comprobación almacenadas en bases de datos de malware conocido, si esas firmas coinciden entonces el archivo en la computadora se considera un archivo peligroso.

Sin embargo, con el enfoque anterior se da por hecho que los antivirus que utilicen esta técnica no detectarán ningún malware nuevo, sólo detectarán los ya conocidos.

Como los EDRs levantan alertas basandose en el comportamiento y no basandose en firmas, es posible solucionar el problema anterior y detectar nuevo malware [37].

En la Tabla 2.2 se muestra una comparativa entre diferentes herramientas EDRs.

2.5. Malware

Malware, que significa software malicioso, es cualquier programa que se crea con el objetivo de dañar un ordenador o una red. Para conseguir tal daño, existen muchas técnicas conocidas que se utilizan para superar las dificultades de cada paso. Algunas de estas técnicas se presentarán en los siguientes capítulos [25] [27].

Nombre	Open source	Reme-diation	Dead RAM	Live RAM	Expand-able	Immediate deploy
Carbon Black		✓				
Crowdstrike		✓	✓	✓		✓
OSQuery	✓	✓	✓			
OSSEC	✓	✓			✓	✓
Sysmantec		✓				
TheHive	✓					
Velociraptor	✓	✓	✓	✓	✓	✓
Wazuh	✓					

Cuadro 2.2: Comparación de herramientas de Endpoint Detection and Response (EDR). Las columnas resaltadas en verde indican los requisitos de Section 5.1. Las celdas vacías significan que las herramientas no tienen esa característica.

2.5.1. Fileless malware

El malware sin archivos es un tipo de malware que no utiliza el ejecutable tradicional como recurso principal para realizar sus actividades. Teniendo esto en cuenta, es capaz de evadir los sistemas de detección basados en firmas.

Este tipo de malware utiliza procesos y herramientas legítimos y de confianza ya incluidos en los sistemas operativos para atacar el ordenador y ocultarse después [38].

2.5.2. Process injection

Process injection es una técnica utilizada para ejecutar código elegido por el atacante en el espacio de direcciones de memoria virtual de otro proceso que este corriendo. La idea detrás de esta técnica es crear un proceso legítimo y de confianza, para luego ejecutar el código elegido como si fuera el proceso de confianza y no uno malicioso. Las consecuencias de esto son el acceso a la memoria del proceso objetivo, el acceso a los recursos del proceso objetivo o la obtención de privilegios elevados, así como la evasión de la detección del usuario [28].

Cabe señalar que después de la inyección se ha hecho, la checksum del archivo no cambia porque el contenido en el disco hace sigue siendo el mismo así, solo cambia el contenido en memoria. Asi, la checksum también se mantendrá [15] [16].

2.5.3. Process hollowing

El Process hollowing es una subtécnica de process injection que consiste en crear un nuevo proceso en modo suspendido. A continuación, se asigna un nuevo segmento de memoria virtual, seguido de la escritura del código elegido en el segmento recién asignado.

Por último, es necesario ejecutar el proceso en el código recientemente inyectado [29].

También es posible desmapear el segmento de código existente, por lo que el código inyectado es el único código asignado en la memoria virtual del proceso [16] [17].

En los sistemas operativos Windows, la técnica descripta anteriormente se consigue llamando a las siguientes funciones de la API de Windows:

1. `CreateProcess` para crear un nuevo proceso en modo suspendido.
2. `VirtualAllocExe` para asignar un nuevo segmento de memoria virtual.
3. `WriteProcessMemory` por escribir el código para inyectar.
4. `SetThreadCreate` o `ResumeThread` para reanudar el proceso.
5. `NtUnmapViewOfSection` para desmapear el segmento de código original.

2.6. Herramientas

Utilizo herramientas adicionales para recopilar información sobre los procesos.

Una de ellas es Process Monitor. Es una herramienta del framework Windows Internals que proporciona información sobre los procesos en ejecución. Esta herramienta tiene la capacidad de obtener cierta información relacionada con las técnicas de process injection y process hollowing, como qué procesos se crean a partir del proceso de malware. Dispone de una GUI que muestra la actividad de los procesos y es posible filtrar la información deseada [30].

Otra herramienta es drstrace. Rastrea las llamadas a funciones del kernel de Windows por un determinado proceso [40].

2.7. Velociraptor Query Language (VQL)

VQL es un lenguaje de programación utilizado para consultar endpoints con el fin de recopilar información y el estado del endpoint. Después, los datos se procesan en el servidor [49] [50].

```
SELECT Pid
FROM pslist()
WHERE Name =~ "notepad"
```

Listing 2.3: Ejemplo de query en VQL.

Como ejemplo, una consulta escrita usando VQL puede verse en Listing 2.3. Esta consulta obtiene todos los procesos en ejecución, devueltos por el plugin `pslist()`. A continuación, filtra los procesos para mantener sólo aquellos cuyo nombre coincide con la

expresión regular `notepad`. Finalmente, obtiene el Process Identifier (PID) de todos los procesos filtrados.

VQL mantiene la misma estructura y palabras clave que Structured Query Language (SQL).

2.8. Trabajos similares

En esta sección, presento algunas otras técnicas de última generación que buscan inyecciones de procesos utilizando técnicas relacionadas con el análisis forense de memoria. Estas técnicas son:

- **Block et Al. Algoritmo [43]**: es un algoritmo que devuelve las entradas VADs con páginas ejecutables, que pueden contener código malicioso. Utiliza los valores de las entradas de la tabla de páginas para obtener la protección real de las páginas, y así saber cuáles son las ejecutables.
- **HollowFind**[44] [45]: detecta ataques de process holowing, comparando el contenido del Process Environment Block (PEB) con el VAD.
- **malfind [46]**: encuentra código oculto o inyectado en memoria, utilizando VAD información de entradas y permisos de página.
- **Técnica de Srivastava et Al. [47]**: utiliza el análisis del stack de ejecución para localizar el código inyectado.

Capítulo 3

Metodología

3.1. Muestras de malware

Para comprender mejor si la idea y la ejecución eran correctas, detono muestras de diferentes familias de malware para ver si es posible detectar actividades maliciosas en ellas. Me referiré al término detonar malware como la acción de ejecutar malware en un entorno controlado.

Elijo familias de malware que utilizan técnicas de process hollowing y process injection según la clasificación del marco Mitre Att&ck. Una vez seleccionadas las familias, busco muestras de las familias previamente seleccionadas en repositorios de malware donde están disponibles innumerables muestras de malware de diferentes familias. Los repositorios mencionados anteriormente son MalwareBazaar, TheZoo y Malware Family Explorer.

Presentaré cada uno de los repositorios de malware en detalle:

- Bazaar está alojado por la empresa de seguridad abuse.ch, pero cualquiera puede cargar muestras en él. El sitio web proporciona metadatos relacionados con la muestra, como la extensión del archivo, el hash SHA256, el hash SHA1 y, en algunos casos, asigna una firma de familia de malware a la muestra. El usuario también puede etiquetar las muestras según criterios como la familia de malware o el sistema operativo de destino, entre otros. He utilizado tanto etiquetas como firmas para encontrar muestras relevantes.

- TheZoo es un repositorio de malware activo alojado en GitHub, donde cualquiera puede crear una solicitud para subir nuevas muestras. Es posible encontrar directorios con el nombre de las muestras, con las muestras y metadatos sobre ellas dentro de cada directorio.

- Malware Family Explorer es una colección de malware creada por vx underground que además tiene una estructura de directorios con el nombre de cada familia y dentro de cada una hay muestras correspondientes a esa familia.

En el caso de las dos últimas fuentes, utilicé los nombres de los directorios para obtener las muestras correctas.

De los repositorios antes mencionados, obtuve 79 muestras de 41 familias diferentes. Todo el contenido descargado son archivos zip y las muestras reales estaban dentro del archivo zip.

Además de las muestras públicas mencionadas anteriormente, los penetration testers del equipo pusieron a mi disposición otras dos muestras de última generación. Estas muestras han sido desarrolladas por ellos mismos y utilizan process injection y process hollowing.

3.2. Architectura para las pruebas con malware

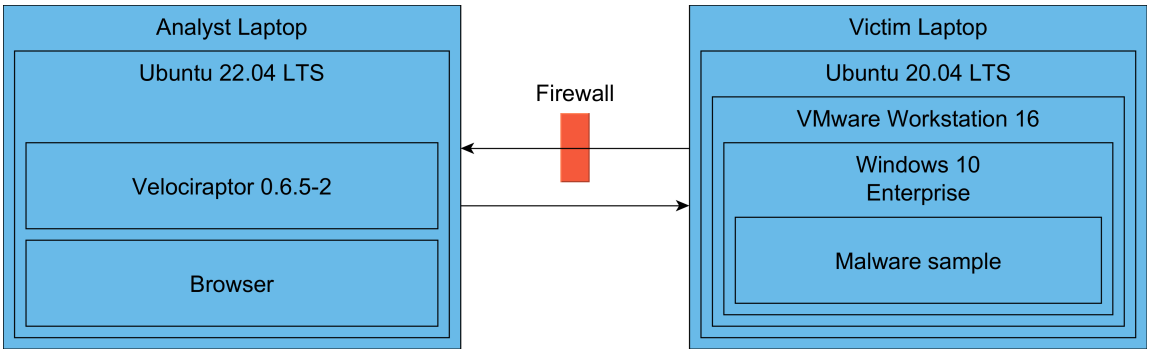

Figura 3.1: Architectura para las pruebas con malware.

Como se muestra en la Fig. 3.1, utilizo dos computadoras portátiles para detonar las muestras. Aunque podría ejecutar las muestras en uno de las computadoras, prefiero esta opción para impedir que el malware escape del entorno de pruebas.

La configuración consistió en una de las computadoras utilizada como máquina víctima, ejecutando Ubuntu 20.04 LTS kernel versión 5.15.0-46-generic como sistema operativo anfitrión con VMware Workstation 16 Pro versión 16.2.4 build-20089737 en él. Este portátil ejecuta una máquina virtual Windows como sistema operativo invitado, con sistema operativo Windows 10 Enterprise 2016 LTSB versión 1607 build 14393.0. Además, todas las opciones de Windows Defender están desactivadas y el directorio `C:\` está en la lista blanca.

Además, Velociraptor versión 0.6.5-2 se ejecuta en modo cliente para recopilar información de este equipo y enviarla al servidor Velociraptor que se ejecuta en la otra computadora.

Creo una snapshot de la máquina virtual mientras se ejecuta en estado limpio para poder restaurar a esta snapshot después de ejecutar cada muestra de malware. El objetivo de esto es recuperar el estado limpio de la máquina virtual para continuar ejecutando las siguientes muestras.

El segundo portátil se utiliza como máquina de análisis. Ejecuta Ubuntu 22.04 LTS y Velociraptor versión 0.6.5-2 en modo servidor para consultar la máquina víctima y extraer los resultados. Además, la conexión Wi-Fi está desactivada y el firefall ufw bloquea todas las conexiones entrantes excepto el puerto 8000, utilizado por Velociraptor para obtener los resultados. Además, el navegador Mozilla Firefox se está ejecutando con el fin de acceder a la interfaz gráfica de Velociraptor.

Para conseguir el aislamiento deseado, ninguno de los ordenadores estaba conectado a ninguna red. La única excepción fue la conexión de la computadora víctima a un hotspot temporal con el único propósito de descargar las muestras de malware. Sin embargo, este portátil sólo estuvo conectado a esa red mientras se descargaba el malware, con el hotspot apagado y la contraseña cambiada en cuanto terminó la fase de descarga.

3.3. Detonando el malware

Una vez tuve la arquitectura lista, procedí a detonar el malware. Esta arquitectura es la descrita en Section 3.2, compuesta por una máquina víctima y otra analista. Para detonar el malware, ejecuté los siguientes pasos en orden:

1. Victima: recuperar el snapshot.
2. Victima: mover el archivo zip de muestra del host a la máquina virtual huésped.
3. Victima: descomprimir el archivo.
4. Analista: recopilar datos con Velociraptor para conocer el estado del ordenador antes de que se detone el malware.
5. Victima: detonar el malware haciendo doble clic en el archivo ejecutable descomprimido en el paso anterior.
6. Analista: ejecutar las queries de Velociraptor con el fin de saber si se detecta el malware.

Para obtener más información sobre lo que ocurre durante la ejecución del malware, ejecuto Process Monitor durante toda la ejecución.

Además, detono las muestras una segunda vez también ejecutando drstrace junto con Process Monitor. Decido ejecutar las muestras una segunda vez y no sólo ejecutar drstrace junto a las otras herramientas durante la primera ejecución porque algunos malware no se comportan de la misma manera si están siendo monitorizados por un proceso externo de esta forma.

Capítulo 4

Resultados

4.1. Implementación propia

Con el fin de obtener la información necesaria para detectar si se está produciendo alguna inyección en alguno de los procesos en ejecución, he implementado dos artifacts propios en Velociraptor: `Mem2Disk` y `ExtraX`. Su código fuente se puede encontrar en GitHub.

El objetivo de `ExtraX` es obtener todas las secciones de memoria de los procesos con permisos ejecutables, excepto el segmento de código de cada archivo `.exe`, que está cubierto por la query `Mem2Disk`. Para ello, primero creo una tabla temporal con el PID y el nombre de todos los procesos actualmente en ejecución. A continuación, creo una segunda tabla temporal con todas las entradas del árbol VAD de un determinado proceso, que representan secciones de memoria, que tiene permisos de ejecutable y no tiene nombre de mapeo. Cabe señalar que el plugin `vad()` fallará en algunos procesos, como `System`, porque Velociraptor no tiene los privilegios necesarios para acceder a esos procesos.

Finalmente, he conseguido el resultado deseado de la consulta combinando las dos consultas anteriores. Es decir, obtengo las entradas VAD que cumplen los requisitos mencionados anteriormente para cada entrada de la primera tabla, es decir, cada proceso en ejecución. El código de este artifact está en Listing 4.1.

El objetivo de la consulta `Mem2Disk` es comprobar si el contenido de la sección `.text` del archivo PE almacenado en un dispositivo de almacenamiento no volátil y el contenido del segmento de código en memoria coinciden.

```
LET GetPids =
  SELECT Pid,
    Name
  FROM pslist()

LET Compare =
  SELECT Pid,
    Name,
    MappingName,
    Protection
  FROM vad(pid=Pid)
  WHERE Protection =~ "x"
    AND NOT MappingName

SELECT * FROM foreach(
  row=GetPids,
  query=Compare
)
```

Listing 4.1: Código Velociraptor Query Language (VQL) de la query ExtraX.

En primer lugar, utilizo la misma subquery utilizada en `ExtraX` para obtener los procesos que se ejecutan en el sistema. A continuación, obtengo la dirección y el path de las entradas del árbol que tienen un nombre de asignación existente, permisos de ejecución y lectura, y el path tiene una extensión `.exe`. Después, utilizo el path extraído de la entrada VAD para acceder al archivo PE y obtener el header de la sección `.text`, que es la primera de todas las secciones del archivo. El path, la dirección y el header `.text` se almacenan en la tabla `GetMetadata`.

Con la información de la tabla `GetMetadata`, obtengo el contenido del segmento de código desde memoria, accediendo al offset especificado en la entrada del árbol VAD, y obtengo el contenido de toda la sección `.text` leyendo de nuevo el archivo PE desde disco. Posteriormente, comparo ambos contenidos y, finalmente, repito el proceso para cada proceso que se ejecuta en el sistema.

El código del artifact `Mem2Disk` está en Listing 4.2.

```
LET GetPids =
  SELECT Pid,
    Username
  FROM pslist()

LET InfoFromVad =
  SELECT Address,
    format(
      format='''C:\%s''',
      args=strip(
        prefix='''\Device\HarddiskVolume2\''',
        string=MappingName)
      ) AS Path
  FROM vad(pid=Pid)
  WHERE MappingName
    AND Protection =~ "xr-"
    AND Path =~ "(exe)$"

LET GetMetadata =
  SELECT Path,
    Address,
    parse_pe(file=Path).Sections[0] AS TextSegmentData
  FROM InfoFromVad

LET GetContent =
  SELECT *,
    format(format="%#x", args=Address) AS AddressHex,
    format(format="%x",
      args=read_file(accessor="process",
        offset=Address,
        filename=format(format="/%d", args=Pid),
        length=TextSegmentData.Size)
      ) AS MemoryData,
    format(format="%x",
      args=read_file(accessor="file",
        offset=TextSegmentData.FileOffset,
        filename=Path,
        length=TextSegmentData.Size)
      ) AS DiskData
  FROM GetMetadata

LET Compare =
  SELECT Pid,
    Path,
    TextSegmentData.Size,
    MemoryData = DiskData AS comparison
  FROM GetContent
  WHERE NOT comparison

SELECT * FROM foreach(
  row=GetPids,
  query=Compare
)
```

Listing 4.2: Código Velociraptor Query Language (VQL) de la query Mem2Disk.

4.2. Resultados de las pruebas

En la siguiente sección, presento los resultados de las pruebas de las técnicas de detección con software malicioso y no malicioso. La información de las tablas es la siguiente:

- **Nombre**: nombre de la familia de malware o software ejecutado.
- **SHA256 hash**: los 8 primeros dígitos del hash SHA256 del archivo `.exe` ejecutado. Los hashes SHA256 completos se pueden encontrar en Table A.1.
- **extra alloc**: si se han asignado secciones ejecutables adicionales.
- **proc create**: si se ha creado algún proceso adicional.
- **ExtraX det**: si la detección ExtraX funcionó.
- **Mem2Disk det**: si la detección Mem2Disk funcionó.

4.2.1. Pruebas con software no malicioso

Los resultados de la ejecución del software no malicioso se presentan en la Tabla 4.1. Como se muestra en la tabla anterior, algunos de los programas no maliciosos son detectados por las técnicas `ExtraX` y `Mem2Disk`.

Nombre	SHA256 hash	extra alloc	proc create	ExtraX det	Mem2Disk det
Adobe Acrobat Reader	966cfa95...	✓	✓	✓	✗
Command Prompt	935c1861...	✗	✓	✗	✗
Discord	a3f9b57e...	✓	✓(own)	✓	✓
Google Chrome	af0bd408...	✓	✓	✓	✓
LibreOffice Writer	7e1ef3b9...	✓	✓	✓	✗
LibreOffice Calc	961ef417...	✓	✓	✓	✗
Microsoft Edge	eb8da1b8...	✓	✓	✓	✓
Microsoft Paint	061d41c4...	✗	✗	✗	✗
Microsoft Teams	8a94b091...	✓	✓(own)	✓	✓
Mozilla Firefox	ead605af...	✓	✓	✓	✓
Spotify	3b84962c...	✓	✓(own)	✓	✓
VLC	1a403269...	✗	✗	✗	✓
Windows Calculator	d7b378a4...	✗	✗	✗	✗
WordPad	0092cd4a...	✗	✗	✗	✗
Zoom	f61f4548...	✗	✓(own)	✗	✗

Cuadro 4.1: Pruebas con software no malicioso.
Un ✓en las columnas ExtraX y Mem2Disk det significa que las detecciones han detectado los programas, mientras que una ✗significa que no lo han hecho.
`(own)` significa que esos procesos han iniciado otra instancia de sí mismos.

4.2.2. Pruebas con malware

Los resultados de la detonación de las familias de malware mencionadas en la sección 3.1 se presentan en la tabla 4.2.

De las familias de malware detectadas, muchas de ellas crean múltiples procesos nuevos, no sólo los que se van a vaciar. Algunas familias como Ryuk, HyperBro, remcos, entre otras, ejecutan binarios desde carpetas temporales que no están allí antes de ejecutar el malware. Esta es una técnica conocida utilizada por diferentes familias de malware [52].

Las herramientas muestran que el malware vacía algunos de los binarios temporales después de ser ejecutados.

Sin embargo, algunos de los procesos adicionales creados no están siendo utilizados con el mismo propósito entre las diferentes familias. Por ejemplo, puedo detectar que las muestras AgentTesla y netwire están creando una instancia de `schtasks.exe`, que es un proceso para programar tareas en sistemas operativos Windows. Aunque la muestra netwire está haciendo process hollowing, la muestra AgentTesla `8118d7c7...` no lo está haciendo.

Además, las diferentes muestras de las mismas familias suelen tener el mismo comportamiento entre ellas, como es el caso de GuLoader, y pandora, entre otras familias.

Incluso cuando hay diferencias, a veces son menores, como es el caso de la familia lokibot. La diferencia entre las muestras de esta familia es que la muestra `97301bb...` hace process hollowing sobre `msinfo32.exe`, mientras que la muestra `afe2844c...` inyecta el código en el proceso `ReAgentC.exe`.

Además, WhisperGate y remcos hacen process injection en `WerFault.exe`. Este proceso es el que hace saltar el cartel de alerta en Windows.

La familia Hoplight elimina el binario original después de inyectar otros procesos.

Las dos muestras analizadas de la familia Pandora desmapean el segmento de texto y añaden segmentos ejecutables extra para ejecutar el código deseado.

Nombre	SHA256 hash	extra alloc	proc create	ExtraX det	Mem2Disk det
AgentTesla	eeedc8ec...	✓	✓	✓	✓
AgentTesla	8118d7c7...	✓	✓	✓	✗
AssemblyInjection	de4cd0c5...	✓	✓	✗	✗
Astaroth	972cae6f...	✓	✓	✗	✗
Astaroth	ce2928ab...	✓	✓	✗	✗
Azorult	08a6193d...	✓	✓	✓	✓
Azorult	5ebb3cc4...	✓	✗	✓	✗
BADNEWS	d07adf20...	✓	✓	✗	✓
bandook	4bf9325f...	✓	✓	✓	✓
bazar	300c0dab...	✓	✗	✓	✗
bazar	534d6039...	✓	✗	✓	✗
Donut	bada6d6d...	✓	✓	✓	✗
dtrack	bf8e3f03...	✓	✗	✗	✓
Dyre	1f8ba528...	✓	✓	✓	✓
Dyre	0350d2ab...	✓	✓	✓	✗
Empire	4a23326d...	✓	✓	✓	✗
formbook	d2f58b0f...	✗	✗	✗	✗
formbook	2c7540c6...	✓	✗	✓	✗
Gazer	c5db84fe...	✗	✗	✗	✗
Gazer	ca9e3ea2...	✗	✗	✗	✗
Gh0stRAT	5a9f06e3...	✓	✓	✓	✓
Gh0stRAT	10ed8da5...	✓	✓	✓	✓
GuLoader	3ae4d65b...	✓	✓	✓	✓
GuLoader	cdbe7a2f...	✓	✓	✓	✓
HopLight	032ccd6a...	✓	✓	✓	✗
HopLight	0237b186...	✓	✓	✓	✗
HTran	1b32e680...	✓	✗	✓	✗
HTran	15b529d0...	✓	✗	✓	✓
HyperBro	6e32c33c...	✗	✗	✗	✗
HyperBro	07f87f7b...	✓	✓	✓	✓
InjectionPoC	e75b681c...	✓	✓	✓	✗
InvisiMole	2debef67...	✗	✗	✗	✗
InvisiMole	d0062f47...	✗	✗	✗	✗
ISMAgent	33c187cf...	✓	✓	✗	✓
ISMAgent	74f61b6f...	✓	✓	✗	✓
Kimsuky	7d89a16f...	✓	✗	✓	✗
Kimsuky	b207265b...	✗	✗	✗	✗
lokibot	97301bb7...	✓	✓	✓	✓
lokibot	afe2844c...	✓	✓	✓	✓
netwire	ea32c3c3...	✓	✓	✓	✓
Pandora	1f172321...	✓	✓	✓	✓
Pandora	5b56c5d8...	✓	✗	✓	✓
PlatinumGroup	021bb772...	✓	✓	✗	✗
PlatinumGroup	46a9ac06...	✓	✓	✓	✗
poshc2	3c4c4cb0...	✓	✓	✓	✓
poshc2	8ba619e1...	✓	✓	✓	✗
qakbot	3be90506...	✓	✓	✓	✗

qakbot	6f00837f...	✓	✓	✓	✗
remcos	03e29815...	✓	✓	✓	✓
remcos	944ec3ee...	✓	✓	✓	✓
REvil	0c10cf1b...	✓	✓	✓	✓
REvil	00d015ed...	✗	✗	✗	✗
RokRAT	af61993f...	✓	✓	✓	✗
RokRAT	9b383ebc...	✓	✓	✓	✓
Ryuk	5e2c9d80...	✓	✓	✗	✓
shadowpad	aef610b6...	✗	✗	✗	✗
sliver	5adc6b62...	✗	✗	✗	✗
sliver	38895ca4...	✗	✗	✗	✗
SlothfulMedia	320cf030...	✓	✗	✗	✓
SlothfulMedia	83131292...	✓	✓	✗	✓
smokeloader	041a05dd...	✓	✓	✓	✓
synack	5b9dee21...	✗	✓	✗	✗
trickbot	b7cbc5e5...	✗	✗	✗	✗
trickbot	47ba62ce...	✗	✗	✗	✗
TsCookie	3cad2031...	✗	✗	✗	✗
TsCookie	80ffaea1...	✓	✓	✓	✗
Turla	44d6d67b...	✗	✗	✗	✗
Turla	95d1f440...	✗	✗	✗	✗
ursnif	104e6094...	✓	✓	✓	✓
ursnif	e3fb27a6...	✓	✓	✓	✓
WarzoneRAT	6c34c666...	✓	✓	✓	✓
WarzoneRAT	8590ebe9...	✓	✓	✓	✓
WhisperGate	b50fb203...	✗	✗	✗	✗
WhisperGate	dcbbae5a...	✓	✓	✓	✓

Cuadro 4.2: Resultados de la detonación de malware.

Un ✓en las columnas ExtraX y Mem2Disk det significa que las detecciones han detectado los programas, mientras que una ✗significa que no lo han hecho.

Capítulo 5

Discusión

5.1. Decisiones

5.1.1. Selección de la familia de sistemas operativos

Selecciono Microsoft Windows como familia de sistemas operativos principal porque tiene alrededor del 76 por ciento de la cuota de mercado de ordenadores de sobremesa y fue el objetivo de 41,4 millones de nuevas muestras de malware durante el primer semestre de 2022, de un total de 43,8 millones, lo que supone el 94,5 por ciento de las muestras [31]. [33].

Además, en la primera semana de noviembre de 2022 se subieron a virustotal más de 3 millones de muestras de archivos `.exe` y alrededor de 1,7 millones de archivos `.dll`. Esto significa que el segundo y el cuarto formato de archivo más cargados son específicos de Windows. Además, estos son el primer y segundo tipo de archivo más subido para formatos de archivo específicos del sistema operativo, teniendo en cuenta que el primero en el ranking general son los archivos HyperText Markup Language (HTML) con 3,1 millones de muestras y el tercero son los archivos JavaScript con algo más de 2 millones de muestras [32].

El hecho anterior garantiza un gran conjunto de muestras para probar cualquier técnica que decida implementar, asegurando que la falta de muestras no será un problema.

Sin embargo, la idea detrás de la detección se basa en tener un archivo donde se almacena el código de confianza y luego ese contenido se carga en memoria para ser ejecutado. Como estas características no son específicas de Windows, esta idea general es genérica y puede ser implementada para otros sistemas operativos.

5.1.2. Selección de malware

Como mi técnica de detección se centra en el comportamiento en memoria del proceso, el malware que considero interesante para probar es el que tiene algún tipo de actividad

en memoria.

Aunque es posible detonar cualquier otra muestra e intentar detectarla, no sería lo ideal, ya que no es probable que la detección funcione porque no está pensada para ello. Así pues, seleccioné familias de malware que utilizan la process injection y process hollowing, como se muestra en la tabla 5.1. Este parecía un conjunto representativo para evaluar las técnicas implementadas.

Name	Technique	Name	Technique
AgentTesla	Process Hollowing	lokibot	Process Hollowing
AssemblyInjection	Process Injection	netwire	Process Hollowing
Astaroth	Process Hollowing	Pandora	Process Injection
Azorult	Process Hollowing	PlatinumGroup	Process Injection
BADNEWS	Process Hollowing	poshc2	Process Injection
bandook	Process Hollowing	qakbot	Process Hollowing
bazar	Process Hollowing	remcos	Process Injection
Donut	Process Injection	REvil	Process Injection
dtrack	Process Hollowing	RokRAT	Process Injection
Dyre	Process Injection	Ryuk	Process Injection
Empire	Process Injection	shadowpad	Process Injection
formbook	Process Hollowing	sliver	Process Injection
Gazer	Process Injection	SlothfulMedia	Process Injection
Gh0stRAT	Process Injection	smokeloader	Process Hollowing
GuLoader	Process Injection	synack	Process Hollowing
HopLight	Process Injection	trickbot	Process Hollowing
HTran	Process Injection	TsCookie	Process Injection
HyperBro	Process Injection	Turla	Process Injection
InjectionPoC	Process Injection	ursnif	Process Hollowing
InvisiMole	Process Injection	WarzoneRAT	Process Injection
ISMAgent	Process Hollowing	WhisperGate	Process Injection

Cuadro 5.1: Diferentes técnicas utilizadas por cada familia de malware.

5.1.3. Selección de herramienta de Endpoint Detection and Response

Con el fin de permitir que un mayor número de personas reproduzcan nuestros resultados y ajustar fácilmente las características básicas de la herramienta si es necesario, decido utilizar un EDR de código abierto. Además, elijo un acrónimo con una interfaz de plugin

para la capacidad de expansión, teniendo en cuenta que sería una característica necesaria para llevar a cabo cualquier proyecto de investigación. Por último, para detectar malware sin archivos es necesario el acceso a RAM. Tanto el análisis muerto como el vivo son posibles, pero prefiero el vivo por las ventajas mencionadas en la sección 2.4.

En resumen, elijo Velociraptor como el EDR a utilizar porque, como se ve en la Tabla 2.2, la herramienta es de código abierto, tiene una interfaz de plugin, y permite el acceso a memoria en vivo.

5.1.4. Selección de técnica

La idea detrás de la creación de mis artefactos es cubrir dos lugares donde pueden ocurrir las inyecciones, ya que se necesitan segmentos de memoria con permisos ejecutables para que el atacante pueda realizar la inyección. Con la suposición anterior, hay dos métodos donde es más probable que se ejecute la inyección: asignando un nuevo segmento o utilizando los segmentos de memoria ejecutables ya disponibles.

La primera idea está cubierta por el artefacto `ExtraX` y la segunda está cubierta en su mayoría por el artefacto `Mem2Disk`. Sin embargo, el segmento de código de todas las Dynamic-Link Libraries (DLLs) mapeadas son un lugar donde una inyección puede ocurrir, pero decido no centrarme en las DLLs de memoria ya que es posible inyectar una nueva Dynamic-Link Library (DLL) entera con el código necesario sin necesidad de inyectar en una DLL existente.

En el artefacto `Mem2Disk` utilizo las secciones sin nombre de mapeo. Elijo estas secciones porque si la región está destinada a ser utilizada para un archivo mapeado, entonces la estructura `_FILE_OBJECT` está presente en el área de control que se puede encontrar desde la entrada VAD. Sin embargo, si la asignación es realizada por el usuario, no habrá un `_FILE_OBJECT` asociado porque no se supone que tenga un archivo mapeado relacionado con la sección, y, por lo tanto, esta sección no debe tener permisos ejecutables [34].

5.1.5. Detonando malware utilizando herramientas adicionales

Al detonar las primeras muestras, considero necesario disponer de información relacionada con la actividad del proceso.

En primer lugar, decido buscar procesos que se estén creando a partir del supuesto proceso de malware que detono. Si no hay ningún proceso nuevo creado a partir del proceso original, entonces no es probable que se produzca el process hollowing. Para obtener esta información utilizo Process Monitor.

Además, decido rastrear todas las llamadas a funciones del kernel de Windows. Con esto, es posible darse cuenta si el proceso ha asignado memoria ejecutable extra, que puede ser usada para inyectar código. Para rastrear las llamadas utilicé drstrace.

5.2. Comparación con otras técnicas.

En comparación con las técnicas mencionadas en la Sección 2.8, la técnica `ExtraX` es similar a `malfind` ya que utiliza la información del árbol VAD y también ambas utilizan la información de las páginas de memoria para detectar la inyección de código en la memoria de los procesos.

Sin embargo, no son iguales, ya que `malfind` se centra en detectar secciones ejecutables ocultas e inyectadas [46], mientras que `ExtraX` busca todas las secciones ejecutables.

Sin embargo, la técnica `Mem2Disk` no es directamente comparable con ninguna de las técnicas presentadas. Utiliza VAD contenido como Block et Al. [43] y `malfind`, pero la técnica en sí no es similar a ellas.

Aunque `Hollowfind` utiliza información de VAD para detectar las inyecciones de código, también utiliza información de PEB, por lo que es diferente de `Mem2Disk` y `ExtraX`.

Además, ni `Mem2Disk` ni `ExtraX` tienen un procedimiento similar al utilizado en Srivastava et Al. [47]. Los tres sólo comparten el objetivo de detectar código inyectado.

5.3. Análisis de los resultados

5.3.1. Tasa de detección, sensibilidad y precisión

Como se muestra en la Tabla 4.2, he probado las técnicas `Mem2Disk` y `ExtraX` contra diferentes familias de malware. De las 79 muestras descargadas, muchas no corrieron debido a problemas de compatibilidad. Como sólo se han probado 59 muestras, hay que considerarlo con cautela. Sin embargo, los resultados preliminares parecen válidos, y considero que el análisis es razonablemente sólido. Los resultados de la fase de pruebas se muestran en la tabla 5.2.

	No detectado	**Detectado**	**Total**
No malware	10 % (6)	17 % (10)	27 % (16)
Malware	14 % (8)	59 % (35)	73 % (43)
Total	24 % (14)	76 % (45)	100 % (59)

Cuadro 5.2: Resultados de software no malicioso y de todas las familias de malware.
El porcentaje de verdaderos negativos (VN) es del 10 por ciento, mientras que el de falsos positivos (FP) es del 17 por ciento. Asimismo, el porcentaje de falsos negativos (FN) es del 14 por ciento y el de verdaderos positivos (VP), del 59 por ciento.
Las cifras entre paréntesis son los valores absolutos.

Sin embargo, cabe señalar que cinco de las familias de malware no se ejecutan en absoluto. La mayoría de ellas muestran la alerta de compatibilidad con Windows "Esta aplicación no puede ejecutarse en su PC". Estas familias son Gazer, InvisiMole, shadowpad, sliver y Turla. Aunque es posible pensar que la alerta de compatibilidad con Windows es el

malware tratando de recrear una interfaz de usuario válida para hacer que el usuario caiga en hacer clic en ella, estoy seguro a través de mi análisis con Process Monitor y drstrace que las muestras de malware no se ejecutaron.

Como ya que no se pueden ejecutar, descarto las cinco familias anteriores de la discusión. La tabla 5.2 se presenta para completar.

	No detectado	**Detectado**	**Total**
No malware	11 % (6)	19 % (10)	30 % (16)
Malware	6 % (3)	64 % (35)	70 % (38)
Total	17 % (9)	83 % (45)	100 % (54)

Cuadro 5.3: Resultado del software no malicioso y las familias de malware que pueden ser ejecutadas.
El porcentaje de verdaderos negativos (VN) es del 11 por ciento, mientras que el de falsos positivos (FP) es del 19 por ciento. Asimismo, los falsos negativos (FN) son del 6 por ciento y los verdaderos positivos (VP) del 64 por ciento.
Las cifras entre paréntesis son los valores absolutos.

En la tabla 5.3, presento los resultados en formato de matriz de confusión con las cinco familias ya descartadas. Con estos resultados, calcularé la tasa de detección, la sensibilidad y la precisión de las técnicas. Estos índices se han calculado según Ceponis et Al. [53].

Como se muestra en la ecuación 5.1, 5.2, y 5.3, la tasa de detección es del 77,78 por ciento, mientras que la sensibilidad es del 92,11 por ciento, y la precisión es del 75,93 por ciento.

$$Detection\ rate = \frac{TP}{TP + FP} * 100 = \frac{35}{35 + 10} * 100 = 77{,}78 \tag{5.1}$$

5.1: Calculo de la tasa de detección.

$$Sensitivity = \frac{TP}{TP + FN} * 100 = \frac{35}{35 + 3} * 100 = 92{,}11 \tag{5.2}$$

5.2: Calculo de la tasa de sensibilidad.

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} * 100 = \frac{6 + 35}{6 + 35 + 6 + 10} * 100 = 75{,}93 \tag{5.3}$$

5.3: Calculo de la tasa de precisión.

La tasa de detección es el porcentaje de probabilidad de que la técnica acierte cuando

predice que el proceso es malicioso. Cuanto menor sea el índice de detección, más software no malicioso se clasificará erróneamente como malicioso. Se puede considerar como el número de veces que la herramienta EDR molesta al usuario con alertas innecesarias, cuando en realidad no es necesario.

La sensibilidad es el porcentaje de muestras que se clasifican correctamente. Cuanto menor sea el porcentaje de detección, más familias de malware reales le faltarán a la técnica para etiquetarlas como tales.

Por otra parte, la precisión es el porcentaje de resultados correctos, incluyendo tanto los verdaderos positivos como los verdaderos negativos. Este índice no excluye ninguno de los valores presentes en la matriz de confusión de la tabla 5.3.

Estas tasas se ven afectadas por los falsos positivos. Tanto la tasa de detección como la precisión, son inferiores a la sensibilidad, ya que en las dos primeras se involucran los falsos positivos mientras que en la tercera no.

Sin embargo, el alto porcentaje de falsos positivos es intrínseco al diseño de la técnica. Mis supervisores y yo optamos por ser estrictos y detectar cualquier cambio que se produzca en la memoria. Aunque este enfoque conduce a minimizar la cantidad de falsos negativos, también aumenta la cantidad de falsos positivos. Sin embargo, preferimos esto ya que consideramos que el impacto de un malware no detectado puede ser mayor que etiquetar erróneamente un proceso no malicioso como malware.

5.3.2. Resultados de las pruebas

Con un porcentaje de falsos negativos del 6 por ciento, creo que no es prioritario centrarse en reducirlo. En su lugar, este porcentaje debería al menos mantenerse mientras se abordan las demás limitaciones.

Atribuyo los resultados falsos negativos principalmente a los problemas de sincronización mencionados en la subsección 5.3.3. Una de las familias que merece la pena analizar como ejemplo de este problema es la familia Ryuk.

Después de ejecutar la muestra Ryuk, `Mem2Disk` y `ExtraX` no pueden detectar ningún rastro de la actividad mostrada por Process Explorer y drstrace. Sin embargo, las herramientas pueden registrar la actividad de la memoria y los nuevos procesos que se crean a partir de la muestra original. Por lo tanto, considero que el ataque está ocurriendo pero es demasiado rápido para ser detectado, pero los procesos ya están terminados, lo que significa que ya no es posible acceder a la memoria del proceso.

Para solucionar esto, pongo el estado de los procesos recién creados en modo suspendido, por lo que el proceso no se termina, y puedo acceder a su memoria. Cuando accedo a la misma, es posible detectar una modificación en el segmento de código de `icacls.exe`, uno de los procesos que está creando el proceso Ryuk.

Las familias que la técnica no puede detectar también presentan comportamientos similares según las herramientas. Sin embargo, no puedo suspender los procesos de las familias no detectadas, al menos en el momento oportuno.

Dado que los comportamientos observados de las familias no detectadas son similares a los de las familias con problemas de sincronización, presumo que la falta de detección está relacionada con el problema de sincronización. Considero que estas familias se detectarían si no se mata el proceso inyectado, y es posible acceder a la memoria de este proceso.

Además, considero que los verdaderos positivos del 64 por ciento son un buen punto de partida para la primera iteración de la técnica, sobre todo si es posible mantener este porcentaje con un conjunto de pruebas mayor.

Cabe señalar que la firma de binarios es una técnica para que los creadores validen el código dentro del ejecutable, por lo que los antivirus a veces no analizan los binarios firmados [56]. Sin embargo, la técnica `Mem2Disk` detecta que la familia BADNEWS utiliza un binario firmado para iniciar el ataque y luego sustituye su propio segmento de texto por otro código, que presumo malicioso.

El porcentaje de 19 falsos positivos parece elevado. Teniendo en cuenta que se trata de más de la mitad de los programas no maliciosos analizados, considero importante seguir comprendiendo y mejorando este problema.

Para los casos de falsos positivos de la detección `ExtraX`, presumo que una de las razones es que las secciones ejecutables adicionales presentes en la memoria están relacionadas con binarios cargados bajo demanda. Especialmente en el caso de los navegadores, a veces es necesario ejecutar código después de que el binario se haya cargado en memoria. Un ejemplo de ello, es ejecutar WebAssembly en JavaScript, como se explica en la documentación de Mozilla [55].

En cuanto al segmento de texto modificado detectado por `Mem2Disk`, creo que está relacionado con algún mecanismo que cambia un offset mientras el archivo ejecutable se carga en memoria.

Uno de los programas detectados es VLC porque hay diferencias entre el segmento de texto en RAM y la sección de código en el disco. Sin embargo, nunca son más largas que un byte contiguo y, como se muestra en la Tabla 5.4, siempre se sustituye un valor en disco por el mismo byte en memoria.

Una modificación que mis supervisores y yo consideramos posible es Address Space Layout Randomization (ASLR). ASLR es una técnica que aleatoriza las direcciones de memoria de los procesos para dificultar el éxito de los ataques [54]. En la práctica, no cambia toda la dirección, sino la página de memoria, por lo que sólo se modifica realmente una parte de la dirección. Teniendo esto en cuenta, es posible que la modificación sea sólo de un byte en comparación con las direcciones del binario almacenado en disco.

Una situación similar a VLC ocurre en el proceso de Firefox. Como muestra la Tabla 5.5, sólo se modifican bloques de un byte contiguo, y las sustituciones son las mismas cada vez que ocurre.

Además, en el proceso `firefox.exe` cada byte que cambia hay 109 bytes en los que el contenido en memoria es igual al contenido en disco, por lo que la diferencia total es inferior a un 1 por ciento. Fig 5.1 muestra la distribución de los bytes cambiados. También vale la pena señalar que la diferencia entre los bytes son los mismos entre todos los reemplazos en

Memory content	Disk content	Difference	Times occurred
0x8D	0x40	0x4D	81
0x8E	0x41	0x4D	4
0x8F	0x42	0x4D	224
0x90	0x43	0x4D	67
0x91	0x44	0x4D	38
0x92	0x45	0x4D	57
0x93	0x46	0x4D	49
0x94	0x47	0x4D	127
0x95	0x48	0x4D	85
0x96	0x49	0x4D	145
0x97	0x4A	0x4D	109
0x98	0x4B	0x4D	1462
0x99	0x4C	0x4D	4980
0x9A	0x4D	0x4D	598

Cuadro 5.4: Modificación del contenido de `vlc.exe`.

cada proceso.

Memory content	Disk content	Difference	Times occurred
0xBF	0x40	0x7F	27
0xC0	0x41	0x7F	3
0xC1	0x42	0x7F	19
0xC2	0x43	0x7F	9
0xC3	0x44	0x7F	4358
0xC4	0x45	0x7F	548

Cuadro 5.5: Modificación del contenido de `firefox.exe`.

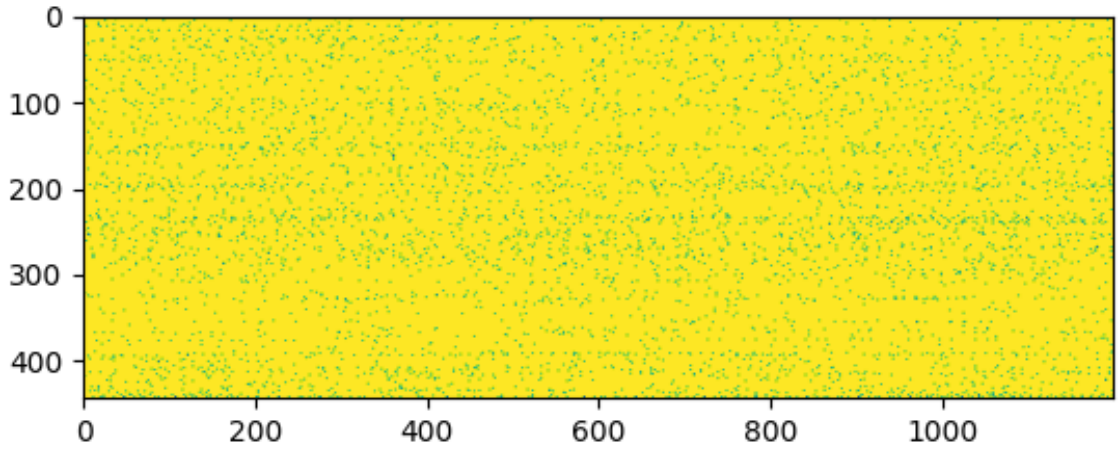

Figura 5.1: Bitmap del segmento de código de `firefox.exe`.
Cada bit del bitmap es un byte de RAM. Los puntos amarillos significan que el valor del byte en memoria es el mismo que el valor en disco, y los azules que el contenido es diferente.

Inyecté un código de 1024 bytes en el segmento de código de `firefox.exe` utilizando un depurador. El mapa de bits del segmento de código inyectado se muestra en Fig 5.2. Esta figura muestra cómo un bloque largo de bytes contiguos en memoria son diferentes y no sólo un byte cada 100 de bytes iguales, como parece el mapa de bits del software no malicioso en Fig 5.1.

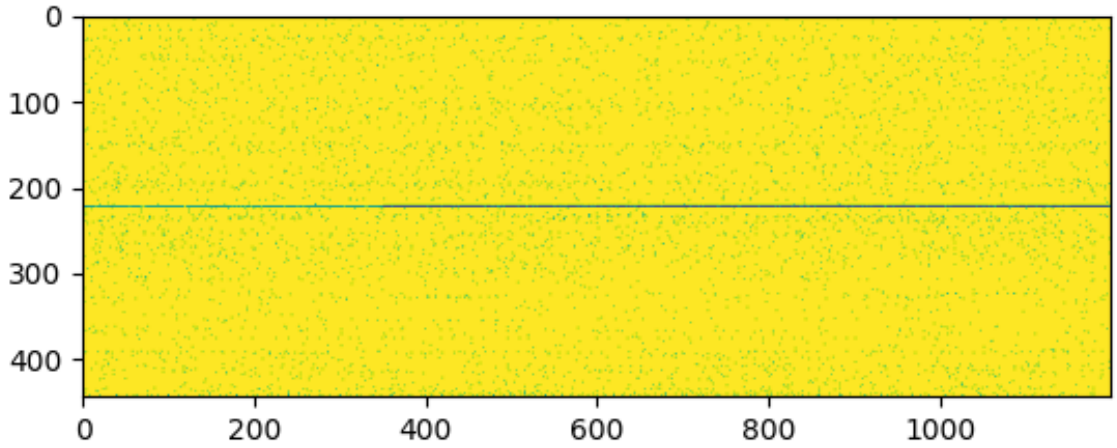

Figura 5.2: Bitmap del segmento de código luego de inyectar malware.
Cada bit del bitmap es un byte de RAM. Los puntos amarillos significan que el valor del byte en memoria es el mismo que el valor en disco, y los azules que el contenido es diferente.

5.3.3. Limitaciones

Uno de los mayores problemas que tengo para detectar las inyecciones de memoria es el tiempo. Ejecutar los artifacts Velociraptor lleva tiempo debido a la necesidad de acceder a muchas estructuras de datos almacenadas en memoria y, sobre todo, tiene que leer el contenido desde el disco.

Teniendo en cuenta que el RAM del ordenador cambia continuamente de estado, uno de los mayores retos de esta técnica es conseguir la memoria con el código inyectado aún presente. A veces este código se ejecuta en un corto período de tiempo y luego se mata el proceso, lo que borra la memoria virtual del proceso sin dejar rastro de la inyección. Si el código inyectado ya no está presente, no es posible detectar el malware con nuestro método.

De las 35 muestras detectadas, se encuentran problemas de sincronización en 8 de ellas, que son: ISMAgent, donut, GuLoader, Pandora, TsCookie, Ryuk y WarzoneRAT. Además, de las 3 familias de técnicas que no se detectan, Process Monitor y drstrace indican que tienen un comportamiento similar a las otras muestras pero la ejecución es más rápida.

Aunque esto es claramente una desventaja de la técnica, es una cuestión inherente al análisis forense como área: si el detective no puede llegar a tiempo, es probable que las pruebas no estén allí. Más concretamente, en el caso de la memoria forense, habrá lagunas en la detección debido a que el estado de la memoria cambia constantemente.

Como ya se ha mencionado, otra limitación es el tamaño del conjunto de familias de malware. Considero que 43 familias no es suficiente para una decisión final sobre la técnica. Creo que es necesario aumentar este número, y creo que la automatización de la fase de pruebas como una opción para aumentar este número.

Capítulo 6

Conclusión

En resumen, presento una técnica para detectar malware basado en memoria, centrándome en los ataques process hollowing y process injection. Para lograr estos objetivos, comparo el contenido entre RAM y disco con el fin de comprobar que todos los segmentos ejecutables en memoria no han sido manipulados. Además, busco segmentos ejecutables adicionales que se asignan después de que el programa se carga inicialmente en la memoria.

Los resultados fueron prometedores, con una tasa de detección del 77,78 por ciento y una tasa de sensibilidad del 92,11 por ciento al excluir las muestras de malware que no se ejecutan.

Como trabajo futuro, es necesario investigar más a fondo la tasa de falsos positivos para poder reducirla, y mitigar el problema de la temporización. Además, es necesario aumentar la cantidad de familias de malware analizadas para generar una tabla de resultados más sólida.

Bibliografía

[1] Published by Statista Research Department, & 3, A. (2022, August 3). Cyber crime: Reported damage to the IC3 2021. Statista. Retrieved November 30, 2022, from https://www.statista.com/statistics/267132/total-damage-caused-by-by-cyber-crime-in-the-us/ Accessed 23 Nov. 2022.

[2] Eddy, Melissa, and Nicole Perlroth. "Cyber Attack Suspected in German Woman's Death." The New York Times, The New York Times, 18 Sept. 2020, https://www.nytimes.com/2020/09/18/world/europe/cyber-attack-germany-ransomeware-death.html. Accessed 04 Nov. 2022.

[3] "Cost of a Data Breach 2022." IBM, https://www.ibm.com/reports/data-breach. Accessed 28 Nov. 2022.

[4] Baram, Gil. "Analysis — How the Cyberwar between Iran and Israel Has Intensified." The Washington Post, WP Company, 25 July 2022, https://www.washingtonpost.com/politics/2022/07/25/iran-israel-cyber-war/. Accessed 28 Nov. 2022.

[5] Baram, Gil. "Analysis — How the Cyberwar between Iran and Israel Has Intensified." The Washington Post, WP Company, 25 July 2022, https://www.washingtonpost.com/politics/2022/07/25/iran-israel-cyber-war/. Accessed 28 Nov. 2022.

[6] Zhioua, Sami. "The Middle East under Malware Attack Dissecting Cyber Weapons." 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops (2013): 11-16.

[7] Masdari, Mohammad and Hemn Khezri. "A survey and taxonomy of the fuzzy signature-based Intrusion Detection Systems." Appl. Soft Comput. 92 (2020): 106301.

[8] WatchGuard Technologies, Inc. "New Research: Fileless Malware Attacks Surge by 900 % and Cryptominers Make a Comeback, While Ransomware Attacks Decline." GlobeNewswire News Room, WatchGuard Technologies, Inc, 30 Mar. 2021,

https://www.globenewswire.com/news-release/2021/03/30/2201173/0/en/New-Research-Fileless-Malware-Attacks-Surge-by-900-and-Cryptominers-Make-a-Comeback-While-Ransomware-Attacks-Decline.html. Accessed 28 Nov. 2022.

[9] "Only in Memory: Fileless Malware – an Elusive TTP." CIS, https://www.cisecurity.org/insights/blog/only-in-memory-fileless-malware-an-elusive-ttp. Accessed 29 Nov. 2022.

[10] Kara, Ilker. "Fileless Malware Threats: Recent Advances, Analysis Approach Through Memory Forensics and Research Challenges." SSRN Electronic Journal (2022)

[11] Gartner Inc. "Named: Endpoint Threat Detection and Response." Anton Chuvakin, 18 June 2015, https://blogs.gartner.com/anton-chuvakin/2013/07/26/named-endpoint-threat-detection-response/.

[12] "Forensics." Cambridge Dictionary, https://dictionary.cambridge.org/ dictionary/english/forensics.

[13] "For308.1: Introduction to Digital Investigation." Digital Forensics Essentials Course — SANS FOR308, http://www.sans.org/cyber-security-courses/digital-forensics-essentials.

[14] Case, Andrew and Golden G. Richard. "Memory forensics: The path forward." Digit. Investig. 20 (2017): 23-33.

[15] Sun, Yixin et al. "Detecting Malware Injection with Program-DNS Behavior." 2020 IEEE European Symposium on Security and Privacy (EuroS&P) (2020): 552-568.

[16] Tien, Chin-Wei et al. "Memory forensics using virtual machine introspection for Malware analysis." 2017 IEEE Conference on Dependable and Secure Computing (2017): 518-519.

[17] Security, Microsoft. "Detecting Stealthier Cross-Process Injection Techniques with Windows Defender ATP: Process Hollowing and Atom Bombing." Microsoft Security Blog, 22 July 2019, https://www.microsoft.com/en-us/security/blog/2017/07/12/detecting-stealthier-cross-process-injection-techniques-with-windows-defender-atp-process-hollowing-and-atom-bombing/.

[18] Garfinkel, Simson L.. "Digital forensics research: The next 10 years." Digit. Investig. 7 (2010): S64-S73.

[19] Nance, Kara L. et al. "Digital Forensics: Defining a Research Agenda." 2009 42nd Hawaii International Conference on System Sciences (2009): 1-6.

[20] Frank Adelstein. 2006. Live forensics: diagnosing your system without killing it first. Commun. ACM 49, 2 (February 2006), 63–66. https://doi.org/10.1145/1113034.1113070

[21] C. P. Grobler, C. P. Louwrens and S. H. von Solms, ."A Multi-component View of Digital Forensics,"2010 International Conference on Availability, Reliability and Security, 2010, pp. 647-652, doi: 10.1109/ARES.2010.61.

[22] Karl-Bridge-Microsoft. "PE Format - win32 Apps." Win32 Apps — Microsoft Learn, https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.

[23] 0xRick. "A Dive into the PE File Format - Introduction." 0xRick's Blog, 22 Oct. 2021, https://0xrick.github.io/win-internals/pe1/.

[24] Authors GReAT, et al. "The Devil's in the Rich Header." Securelist English Global Securelistcom, 13 May 2021, https://securelist.com/the-devils-in-the-rich-header/84348/. Accessed 17 Sep. 2022.

[25] "The Difference between Malware and a Virus: CrowdStrike." Crowdstrike.com, 15 Aug. 2022, https://www.crowdstrike.com/cybersecurity-101/malware/malware-vs-virus/.

[26] "What Is Malware and How Cybercriminals Use It." McAfee, https://www.mcafee.com/en-us/antivirus/malware.html. Accessed 27 Nov. 2022.

[27] "Mitre ATT&CK®." MITRE ATT&CK®, https://attack.mitre.org/.

[28] "Process Injection." Process Injection, Technique T1055 - Enterprise — MITRE ATT&CK®, https://attack.mitre.org/techniques/T1055/.

[29] "Process Injection: Process Hollowing." Process Injection: Process Hollowing, Sub-Technique T1055.012 - Enterprise — MITRE ATT&CK®, https://attack.mitre.org/techniques/T1055/012/.

[30] Markruss. "Process Explorer - Sysinternals." Process Explorer - Sysinternals — Microsoft Learn, https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer.

[31] "Desktop Operating System Market Share Worldwide." StatCounter Global Stats, https://gs.statcounter.com/os-market-share/desktop/worldwide.

[32] Virustotal, https://www.virustotal.com/gui/stats. Accessed 07 Nov. 2022.

[33] "Linux Malware on a Rise Reaching All-Time High in H1 2022 - Atlas VPN." AtlasVPN, https://atlasvpn.com/blog/linux-malware-on-a-rise-reaching-all-time-high-in-h1-2022.

[34] Dolan-Gavitt, Brendan. "The VAD tree: A process-eye view of physical memory." Digit. Investig. 4 (2007): 62-64.

[35] Hash, Imp. "Windows Process Internals: A Few Concepts to Know before Jumping on Memory Forensics [Part 4] -..." Medium, Medium, 4 Sept. 2020, https://imphash.medium.com/windows-process-internals-a-few-concepts-to-know-before-jumping-on-memory-forensics-part-4-16c47b89e826.

[36] Hassan, Wajih Ul et al. "Tactical Provenance Analysis for Endpoint Detection and Response Systems." 2020 IEEE Symposium on Security and Privacy (SP) (2020): 1172-1189.

[37] Karantzas, G.; Patsakis, C. An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors. J. Cybersecur. Priv. 2021, 1, 387-421.

[38] Sudhakar and Sushil Kumar. "An emerging threat Fileless malware: a survey and research challenges." Cybersecurity 3 (2020): 1-12.

[39] Arpaci-Dusseau, Remzi H.. "Operating Systems: Three Easy Pieces." login Usenix Mag. 42 (2017).

[40] System Call Tracer for Windows, https://drmemory.org/page_drstrace. html. Accessed 14 Oct. 2022.

[41] Barrygolden. "_file_object (WDM.H) - Windows Drivers." _FILE_OBJECT (Wdm.h) - Windows Drivers — Microsoft Learn, https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ns-wdm-_file_object.

[42] Tedhudek. "Windows Kernel Opaque Structures - Windows Drivers." Windows Drivers — Microsoft Learn, https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/eprocess. Accessed 19 Jul. 2022.

[43] Block, Frank and Andreas Dewald. "Windows Memory Forensics: Detecting (Un)Intentionally Hidden Injected Code by Examining Page Table Entries." Digit. Investig. 29-Supplement (2019): S3-S12.

[44] monnappa22, "Monnappa22/Hollowfind." GitHub, https://github.com/monnappa22/HollowFind. Accessed 13 Oct. 2022.

[45] MtlOop, et al. "Detecting Deceptive Process Hollowing Techniques Using HollowFind Volatility Plugin." Cysinfo, 24 Sept. 2016, https://cysinfo.com/detecting-deceptive-hollowing-techniques/.

[46] Volatilityfoundation. "Home · Volatilityfoundation/Volatility Wiki." GitHub, https://github.com/volatilityfoundation/volatility/wiki/CommandReference-Mal#malfind. Accessed 05 Aug. 2022.

[47] Srivastava, Anurag and James H. Jones. "Detecting code injection by cross-validating stack and VAD information in windows physical memory." 2017 IEEE Conference on Open Systems (ICOS) (2017): 83-89.

[48] Thing, Vrizlynn L. L. et al. "Live memory forensics of mobile phones." Digit. Investig. 7 (2010): S74-S82.

[49] Cohen, Mike. "The Velociraptor Query Language Pt 1." Medium, Velociraptor IR, 14 June 2020, https://velociraptor.velocidex.com/the-velociraptor-query-language-pt-1-d721bff100bf.

[50] "Basic VQL." Velociraptor, https://docs.velociraptor.app/vql_reference/ basic/. Accessed 29 Jul. 2022.

[51] https://github.com/lautarolecumberry/DetectingFilelessMalware. Accessed 10 Dec. 2022.

[52] Yan, Tao. "New Wine in Old Bottle: New Azorult Variant Found in Findmyname Campaign Using Fallout Exploit Kit." Unit 42, 11 Dec. 2018, https://unit42.paloaltonetworks.com/unit42-new-wine-old-bottle-new-azorult-variant-found-findmyname-campaign-using-fallout-exploit-kit/.

[53] Ceponis, Dainius and Nikolaj Goranin. "Investigation of Dual-Flow Deep Learning Models LSTM-FCN and GRU-FCN Efficiency against Single-Flow CNN Models for the Host-Based Intrusion and Malware Detection Task on Univariate Times Series Data." Applied Sciences 10 (2020): 2373.

[54] Marco-Gisbert, Héctor and Ismael Ripoll Ripoll. "Address Space Layout Randomization Next Generation." Applied Sciences (2019).

[55] "Loading and running WebAssembly code". https://developer.mozilla.org/en-US/docs/WebAssembly/Loading_and_running. Accessed 12 Dec. 2022.

[56] Kim, Doowon et al. "Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI." Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (2017).

Apéndice A

Apéndice

A.1. Hash SHA-256 completo de las muestras de malware

Name	Complete SHA256 hash
AgentTesla	eeedc8ecc3623353fedfcedf3f5402ad5c4ea2d7cff37be3a730415df2e8a68c
AgentTesla	8118d7c7ac15618f5517b0fec626096796084a2b44bba85a65ee7d2e7b4f1fc9
AssemblyInjection	de4cd0c58f2b44050feea01c56fe1f63b85eb178a15b2f087d963bbef113c644
Astaroth	972cae6f1dbd11ea90e931b27e78d640f0621abc2a5431269d39b2287653cc6c
Astaroth	ce2928ab5086fe869d7b16ffb0e01519fd7a735253e40e40983ad03f33f80600
Azorult	08a6193d0afc12de32573390251740b4b1d7a1af0b19ef0cc3a12c078db76449
Azorult	5ebb3cc4e09a0fb9434d07543cd821538008462dc037c6d6323a32b8bd26dd6e
BADNEWS	d07adf2032c1274ef810aa9146e0407dbe76335b2121c6f98667f67a90f63ce3
bandook	4bf9325fe8d721e60c2a5beee8dbdf275ab9c5de309e162ecc81d1cdf7369cef
bazar	300c0dab0af5de260c5e0a30ff799fd26758b39bb933870674ce632be22841a5
bazar	534d60392e0202b24d3fdaf992f299ef1af1fb5efef0096dd835fe5c4e30b0fa
Donut	bada6d6d493416c0992a375de60fe574ced09bef5496ebfac07c19a8b2785494
dtrack	bf8e3f0430a2a53608432cca208ac7d932e84a557defcfcdcb468b68cfacd7f8
Dyre	1f8ba528cef45357d5c7376e510077aeb9c580af0378b3b80f7d4f94dc531b2f
Dyre	0350d2ab1cb791672d7b3927c57bcdfe71fa4d2e3609201dd8c2f288ac341f4c
Empire	4a23326def54ca250c558925ef891ad92ecd9a1a6870ea85760f8b97fe28613f
formbook	d2f58b08f8abfe5055f3c1f0b8d991dfe1deb62807a5336b134ce2fb36d87284
formbook	2c7540c6d066510b73a1a5c668dc74ec6d0d3f0716bb3adb6cd83afdd07f35ff
Gazer	c5db84fe0f762ebc2abe484d59d51fdf35a37f3a32e6f44d8197b1e8cda98e84
Gazer	ca9e3ea2e21483612ec2d9ff4a91693e97ab24175ac00ccb52da89e4b89230c9
Gh0stRAT	10ed8da5de4785261aeb9a2c113b8d82316b2b0e7946ebb4203b3bf4c3b355e0
Gh0stRAT	5a9f06e3346cb716c79bdfaf347944539fccaeb2e503b6b5d434e47c458c6618
GuLoader	3ae4d65b8e2c2ac4866500331532095749c24778b2ba55a7cf75b7615676d299
GuLoader	cdbe7a2fc091f0b3b4ed8e35d600df24deadc377d7323d92aebf44ed38f41413
HopLight	032ccd6ae0a6e49ac93b7bd10c7d249f853fff3f5771a1fe3797f733f09db5a0
HopLight	0237b186086fa4d13e8c854dcf2d0f8a19fcbe62a58a415e9a5a933f1154e7d8
HTran	15b529d02b4d9effdc660b7546b4aab6f266af87bfca48a5a97ae7b46a725e64
HTran	1b32e6800b3a80e74f135b75925f3c1e081662adfac53262ec9a8a830398ff64
HyperBro	6e32c33c82efaf05822a0d5c610adbc2c1e8fd4d99955b1050496ad29ec927de

HyperBro	07f87f7b3313acd772f77d35d11fc12d3eb7ca1a2cd7e5cef810f9fb657694a0
InjectionPoC	e75b681cd8f4a69663dfcf0cc5337bc00dd8595c45f884e7e91136d764e27886
InvisiMole	2debef67c4e8e9a28af920688b23e858876be623573f2cf23edcd50856388622
InvisiMole	d0062f473c4350dc934752dc4f876c962eeb1c43968647695460f4c8ed629a46
ISMAgent	33c187cfd9e3b68c3089c27ac64a519ccc951ccb3c74d75179c520f54f11f647
ISMAgent	74f61b6ff0eb58d76f4cacfb1504cb6b72684d0d0980d42cba364c6ef28223a8
Kimsuky	7d89a16fc0d3afa3cd78cc51e7ae6a81343cb14de6fdca9325142deca5133515
Kimsuky	b207265be3bdb32536b3118e0d94660241988e2f862a0ccaf968d25d86b87a04
lokibot	97301bb711f9921f5e24aa2e249a4e76cc3ffa359f73153d77712bcc11a20f15
lokibot	afe2844c27424c053cc0e578dae9d5d1bea6cab5ab0227edbaf983289961452e
netwire	ea32c3c39c8c3f83e95916262d37b5aa49c920a9356dcebfa639b8391413ae9a
Pandora	1f172321dfc7445019313cbed4d5f3718a6c0638f2f310918665754a9e117733
Pandora	5b56c5d86347e164c6e571c86dbf5b1535eae6b979fede6ed66b01e79ea33b7b
PlatinumGroup	021bb772775dc4c7df1569c3ee8ed957207df810837bdff71104ea6e905e4681
PlatinumGroup	46a9ac069c20c505e6bc5fcd6de9a0f1d3a8ed3073133913e57d54604a0e8e8e
poshc2	3c4c4cb0e9a48e8203ebe67da38dcfdc0d888213424ddd335a767f6a04e798ff
poshc2	8ba619e1fb38bc0232347892b8fa0f0a3350be8d7397179de74549c07d684bab
qakbot	3be905066595dc785c9b6b98bfb2d9e0478f32df31337a8aeec96d7ccd52769e
qakbot	6f00837f83703021bc4f718a4df8a7fbdadf5fff50728dc09c050efa5259db89
remcos	03e298154f6a21a7f3e98d06193f5f3c902325887428ab1d7c9390685b239d02
remcos	944ec3ee0ba63f9535753135f92b2147efab49a3116c9d428335fcff92ba24ab
REvil	0c10cf1b1640c9c845080f460ee69392bfaac981a4407b607e8e30d2ddf903e8
REvil	00d015edbfb34e16b5b4086d25174ae435ca86d8cd267e0ed9b32db7d1d8ae2f
RokRAT	af619936fa29b7d0cf0c8441674bbf062cea427f9aaad4ea3173b5942956720b
RokRAT	9b383ebc1c592d5556fec9d513223d4f99a5061591671db560faf742dd68493f
Ryuk	5e2c9d80fa4528fe9777738a9cba9ede08cdae353fd4cb2d9caf0c9801fd5711
shadowpad	aef610b66b9efd1fa916a38f8ffea8b988c20c5deebf4db83b6be63f7ada2cc0
sliver	5adc6b62d26ad39c99407b3dfe2869f89a14d174ada9a732f3e1ef0c851c036f
sliver	38895ca4da6111265ad5d5f995d306085ccfcff13fcb2175d4596307a42135b1
SlothfulMedia	320cf030b3d28fcddcf0a3ef541dea15599d516cb6edaad53ec9be6b708d15c2
SlothfulMedia	8313129283310394 8d70b354b95905e484c34c9992cecd00fe9ab5eb1bbc7987
smokeloader	041a05dd902a55029449bf412cedbe59a593f8d4e67d4ae37cf7a9289e2f22ca
synack	68b87153d663663bd8e2e6644eeff9f5291167a2dca1a807e4918976e739ba95
trickbot	b7cbc5e5dc182c8d99809cd64d36734abeb6bfac15e6efc2ebcc2c57254bf172
trickbot	9da8a5a0b5957db6112e927b607a8fd062b870f2132c4ae3442eb63235f789e1
TsCookie	3cad20318f36b020cf4d6b44320eb5a6dae0a78339a0fdc3a1fe5e280a8507f1
TsCookie	80ffaea12a5ffb502d6ce110e251024e7ac517025bf95daa49e6ea6ddd0c7d5b
Turla	44d6d67b5328a4d73f72d8a0f9d39fe4bb6539609f90f169483936a8b3b88316
Turla	95d1f4407f3c725be2100cb72bb30e4fba08960ea83b51b5ead8e210eea51ec4
ursnif	104e6094ef239aae7e4317433e868b67108b8157627dc222f996cb087795334f
ursnif	e3fb27a6761d3a9403ff5b3ddbc86e5231664980149c8fd85bcfb319cc1ebb8c
WarzoneRAT	6c34c6663c544d9d1d255733c2f8d0bb090730467d0cf19b10e0b6abcbbd6fb4
WarzoneRAT	8590ebe9a64020b717f0fda3bc22a35bc6f4f488e63d9a5f8deadd67aa89921e
WhisperGate	b50fb20396458aec55216cc9f5212162b3459bc769a38e050d4d8c22649888ae
WhisperGate	dcbbae5a1c61dbbbb7dcd6dc5dd1eb1169f5329958d38b58c3fd9384081c9b78

Cuadro A.1: Hash SHA-256 completo de las muestras de malware.

A.2. Hash SHA-256 completo de software no malicioso

Name	Complete SHA256 hash
Adobe Acrobat Reader	966cfa9539314f7ff8bd0d403217708ffa17402959e9ef831d11ef7edd502fba
Command Prompt	935c1861df1f4018d698e8b65abfa02d7e9037d8f68ca3c2065b6ca165d44ad2
Discord	a3f9b57ec492d1ca666943899da3c6d01d22dbbc3a5cedb2e575cba8912f4a16
Google Chrome	af0bd408548165770baf8e5b455d7ecaa6de127c8d696c063806d07df7c3a6d5
LibreOffice Writer	7e1ef3b95fe273bd4c16aab6cf485221a7099e8d881a58258156bc18554ebe69
LibreOffice Calc	961ef417d04570f25a72509b52c72c0bcc8b6bb31e304072d5749f0d380f3ac9
Microsoft Edge	eb8da1b8b138179cfa9c97ebbd64af67e22a6ab1673a900c7aaadeb72944439e
Microsoft Paint	061d41c4239a0dbe2d9578d4d10a5db2a8f21c1c0f9b63aae490f1cdd683340a
Microsoft Teams	8a94b091015de99b0449e2df666563a03e35462843e80b709ffa1fb0d2e96a75
Mozilla Firefox	ead605af5446603cbcfa890e7f44cc380bdb69794487ce3ed68103e486b1ed28
Spotify	3b84962c10248d8128d52a3531432e786b236d5f4fcc2b24df0127eb41741d67
VLC	1a403269242218a67e401c7e321bf466ef6b381bc7cb8a56ea77d504f7f81a44
Windows Calculator	d7b378a4bc4deae748462d216d14a20ccb1bac1d3ffbc67711db2cc1d8b182b7
WordPad	0092cd4a00fb6095663dbbd50b8846cdb03f954274a7bf6a84d1face3db4eaf4
Zoom	f61f45486c1a3ea5330a87522e6fa139c3ebf33addd2e3bd3dfd9b2881b06917

Cuadro A.2: Hash SHA-256 completo de software no malicioso.

A.3. Acrónimos

ASLR Address Space Layout Randomization

COFF Common Object File Format

CPU Central Processing Unit

CSIRT Computer Security Incident Response Team

DOS Disk Operating System

DLL Dynamic-Link Library

EDR Endpoint Detection and Response

ELF Executable and Linkable Format

HTML HyperText Markup Language

IC3 Internet Crime Complaint Center

NAS Network Attached Storage

PE Portable Executable

PEB Process Environment Block

PID Process Identifier

RAM Random Access Memory

SANS SysAdmin, Audit, Network and Security

SQL Structured Query Language

TB Terabyte

VAD Virtual Address Descriptor

VQL Velociraptor Query Language