Universidad Zaragoza

Trabajo Fin de Máster

Extracción de indicadores de compromiso mediante análisis forense de memoria

Extraction of Indicators of Compromise using Memory Forensics

AUTOR

Miguel Santiago Moniente Pannocchia

DIRECTORES

Javier Carrillo Mondéjar

Ricardo Julio Rodríguez Fernández

ESCUELA DE INGENIERÍA Y ARQUITECTURA

DICIEMBRE 2024

RESUMEN

Hoy en día, los sistemas informáticos son fundamentales tanto para el funcionamiento de las empresas como para la vida cotidiana, por lo que día tras día emergen constantemente nuevos tipos de código dañino (*malware*), utilizados por los ciberdelincuentes para comprometer o impactar dichos sistemas para su propio beneficio. Para conseguirlo, el desarrollo de *malware* está en constante evolución, aplicando técnicas avanzadas de evasión, ofuscación y ocultación, de tal manera que los sistemas de detección modernos no sean capaces de identificarlo fácilmente. Por este motivo, cuando las medidas de detección fallan y los sistemas son infectados, es imprescindible realizar un análisis forense digital de estos sistemas como parte del proceso de gestión y respuesta ante incidentes. En este contexto, el análisis de *malware* y el análisis forense de memoria juegan un papel fundamental a la hora de estudiar el *software* malicioso para generar indicadores de compromiso (IoCs) que permitan su detección temprana en futuros incidentes de seguridad.

Este trabajo busca evaluar las herramientas de análisis forense de memoria publicadas recientemente en ámbitos académicos y utilizarlas para expandir el enfoque tradicional de análisis de *malware* mediante la inclusión de una fase adicional basada en análisis forense de memoria. Además, en este trabajo se busca automatizar en la medida de lo posible la extracción de IoCs, minimizando así el tiempo invertido por un analista de seguridad en realizar estas tareas manualmente. La extracción de estos indicadores facilita la integración de los resultados del análisis en herramientas de ciberinteligencia, mejorando así los sistemas de detección para una respuesta ante incidentes más rápida y efectiva. Para lograr este objetivo, en este trabajo se propone un sistema automatizado para análisis de *malware* en varias fases, junto con la extracción automática de indicadores de compromiso a partir del análisis realizado.

ABSTRACT

Nowadays, computer systems are essential both for business operations and daily life. As a result, new types of harmful code (malware) appear constantly, developed by cybercriminals to compromise or impact these systems for their own benefit. To achieve this, malware development is in constant evolution, applying advanced techniques like evasion, obfuscation, and covert techniques so that modern detection systems are not able to easily identify it. For this reason, when detection measures fail and systems are infected, it is essential to perform digital forensic analysis of these systems as part of the incident response process. In this context, *malware* analysis and memory forensic analysis play a fundamental role in studying malicious software to generate Indicators of Compromise (IoCs) that helps its early detection in future security incidents.

This work aims to evaluate recently published memory forensic tools from academic domains, and use them to expand the traditional approach to malware analysis by including an additional phase based on memory forensics. Moreover, this work aims to automate as much as possible the extraction of IoCs, minimizing the time spent by a security analyst performing these tasks manually. The extraction of these indicators allow the integration of the analysis results into cyberintelligence tools, thus improving detection systems for a faster and more effective incident response. To achieve this, this work proposes an automated multi-phase malware analysis system, along with the automatic extraction of indicators of compromise based on the conducted analysis.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

ASEP *Auto-Start Extensibility Point* (Punto de Extensibilidad de Auto-Inicio)

DFF *Digital Forensics Framework* (Marco de Trabajo de Forense Digital)

DLL *Dynamic-Linked Libraries* (Bibliotecas de Enlace Dinámico)

EC *Exclusion Criteria* (Criterio de Exclusión)

FOSS *Free and Open-Source Software* (*Software* Libre y de Código Abierto)

GPO *Group Policy Object* (Objeto de Directiva de Grupo)

GPU *Graphics Processor Unit* (Unidad de Procesamiento Gráfico)

HTTP *Hyper-Text Transfer Protocol* (Protocolo de Transferencia de Hipertexto)

IC *Inclusion Criteria* (Criterio de Inclusión)

IoC *Indicators of Compromise* (Indicadores de Compromiso)

IoT *Internet of Things* (Internet de las Cosas)

ML *Machine Learning* (Aprendizaje Automático)

MV Máquina Virtual

PEB *Process Environment Block* (Bloque de Entorno de Proceso)

PID *Process Identifier* (Identificador de Proceso)

PPID *Parent Process Identifier* (Identificador del Proceso Padre)

PTE *Page Table Entries* (Entradas de la Tabla de Páginas)

QMP *QEMU Machine Protocol* (Protocolo de Máquina QEMU)

RAM *Random Access Memory* (Memoria de Acceso Aleatorio)

RIS *Research Information Systems* (Sistemas de Información para la Investigación)

RQ *Research Question* (Pregunta de Investigación)

SLR *Systematic Literature Review* (Revisión Sistemática de la Literatura)

SO Sistema Operativo

UAC *User Account Control* (Control de Cuentas de Usuario)

VAD *Virtual Address Descriptors* (Descriptores de Direcciones Virtuales)

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

En los últimos años ha incrementado considerablemente el número de ciberataques utilizando código dañino [23] (especialmente de tipo *ransomware* [31]), por lo que las herramientas de detección son de extrema importancia para identificar y contrarrestar estas amenazas a los sistemas informáticos en el menor tiempo tiempo posible. Sin embargo, a pesar de la evolución de las medidas de detección, los ciberdelincuentes siguen desarrollando nuevas variantes de *malware*, aplicando técnicas avanzadas de evasión y ofuscación para no ser detectados [17].

Para hacer frente al ritmo de evolución del *malware*, es imprescindible el análisis de las nuevas muestras y variantes para comprender su funcionamiento y generar indicadores de compromiso (IoCs) que permitan su identificación temprana. Durante la gestión de incidentes, un paso fundamental es la realización de un análisis forense y adquisición de muestras para su posterior análisis [16]. Sin embargo, el análisis de *malware* tradicional [47] puede resultar insuficiente frente a *malware* más sofisticados, que se hacen pasar por proceso legítimos, se ocultan en memoria u ofuscan sus trazas en disco [1]. En este contexto, el análisis forense de memoria puede aportar un gran valor, permitiendo examinar la memoria volátil (RAM) del sistema en busca de artefactos e indicadores "invisibles" para otros tipos de análisis [28]. Por otro lado, el análisis forense de memoria suele requerir buena parte de trabajo manual, lo que ralentiza los procesos de respuesta ante incidentes.

De este modo, integrar el análisis forense de memoria con los métodos tradicionales de análisis de *malware* en un sistema de análisis automatizado permite analizar fácilmente muestras de *malware* más complejo y avanzado, así como generar IoCs con la mínima intervención de los analistas de ciberseguridad. Si bien este sistema no es infalible, a través de estos indicadores se incrementa la capacidad de detección de amenazas avanzadas, mejorando así la efectividad de respuesta ante incidentes.

1.2 OBJETIVOS

El principal objetivo de este trabajo es realizar una Revisión Sistemática de la Literatura [8] sobre las herramientas de análisis forense de memoria que se han presentado durante los últimos años en el ámbito académico, identificando las principales soluciones propuestas y sus potenciales limitaciones. A partir de las conclusiones de esta revisión sistemática, se propone implementar un sistema automático de análisis de *malware* que combina el análisis tradicional (fases de análisis estático y dinámico) [47] con una fase de análisis forense de memoria [28] utilizando las herramientas evaluadas. Este sistema tiene como objetivo automatizar, en la medida de lo posible, la extracción indicadores de compromiso y generar un reporte de los mismos con el fin de facilitar su investigación y mejorar las capacidades de los sistemas de detección de *malware*.

1.3 DESCRIPCIÓN DEL DOCUMENTO

En el Capítulo 2 se describen los conceptos previos que sientan las bases de este trabajo, el análisis de malware y el ciclo de vida de respuesta ante incidentes. En el Capítulo 3, se describe el proceso de Revisión Sistemática de la Literatura llevado a cabo, detallando la metodología y presentando los resultados obtenidos. Tras ello, en el Capítulo 4 se propone el sistema de análisis, para después ponerlo a prueba mediante un caso de estudio sobre una muestra de *malware* en el Capítulo 5. En este capítulo también se comentan las principales limitaciones encontradas. En el Capítulo 6 se discuten varios de los trabajos que proponen sistemas similares al desarrollado en este trabajo. Finalmente, en el Capítulo 7 se presentan las conclusiones y el posible trabajo futuro de este trabajo. Al final del documento, se encuentra el Apéndice A, que contiene el detalle de las horas dedicadas al proyecto.

2 CONCEPTOS PREVIOS

En este capítulo se describen algunos de los fundamentos necesarios para comprender el proceso de análisis de *malware* y el ciclo de vida de respuesta ante incidentes de seguridad.

2.1 ANÁLISIS DE MALWARE

El análisis de *malware* es un proceso mediante el cual se estudian las características y comportamientos de archivos sospechosos a través de diversas pruebas, con el fin de identificarlos y clasificarlos en las diferentes familias de *software* malicioso. Estas pruebas se clasifican generalmente en dos tipos: **análisis estático** y **análisis dinámico** [47]. Al utilizar ambos tipos de análisis, se obtiene una comprensión completa del *malware* que permite generar IoCs para su detección y prevención en futuras amenazas. Más concretamente:

Análisis estático: El análisis estático consiste en examinar la estructura y el contenido del archivo sin ejecutarlo, lo que permite extraer información sobre sus funcionalidades, componentes internos y posibles indicadores de contenido malicioso. Durante esta etapa, se utilizan técnicas como desensamblar el binario para obtener el código fuente original, extracción de cadenas de texto y búsqueda de patrones y firmas de *malware* conocido, entre otras.

Análisis dinámico: El análisis dinámico, a diferencia del estático, requiere la ejecución del archivo sospechoso para analizar su comportamiento en el sistema: instalación de programas, modificación de registros, conexiones a Internet, etc. Dado que el *malware* suele tener comportamientos dañinos en el sistema, es esencial realizar esta ejecución en un entorno virtual (o *sandbox*) para evitar comprometer equipos reales [24]. Además, esto permite registrar y monitorizar todas las acciones del *malware* y así perfilar su comportamiento.

2.2 RESPUESTA ANTE INCIDENTES

El ciclo de vida de la respuesta ante incidentes plantea una estrategia de acciones con el fin de mitigar los daños causados ante un incidente de seguridad [16]. Esta estrategia se divide en varias fases a lo largo del incidente, formando un ciclo retroalimentado para la mejora continua futuras ante amenazas de seguridad (véase la Figura 1). Estas fases son:

Preparación: Establecer las capacidades necesarias para manejar incidentes de seguridad de manera efectiva, mediante la preparación de políticas, procedimientos, formación y los métodos de detección apropiados.

Detección y análisis: Constantemente se evalúan los eventos de seguridad y se determina si constituyen un incidente. Si se determina el incidente, se analiza su

alcance para actuar en consecuencia.

Contención, erradicación y recuperación: Una vez identificado el incidente, se deben aplicar las medidas de contención para evitar su propagación. Tras esto, se erradica la causa raíz del incidente y se restauran los sistemas afectados a su estado normal.

Actividad post-incidente: Tras solucionar el incidente, se evalúa la efectividad de las fases anteriores y se determinan las lecciones aprendidas. Si es posible, se realiza un forense de los sistemas afectados para actualizar las medidas de detección.

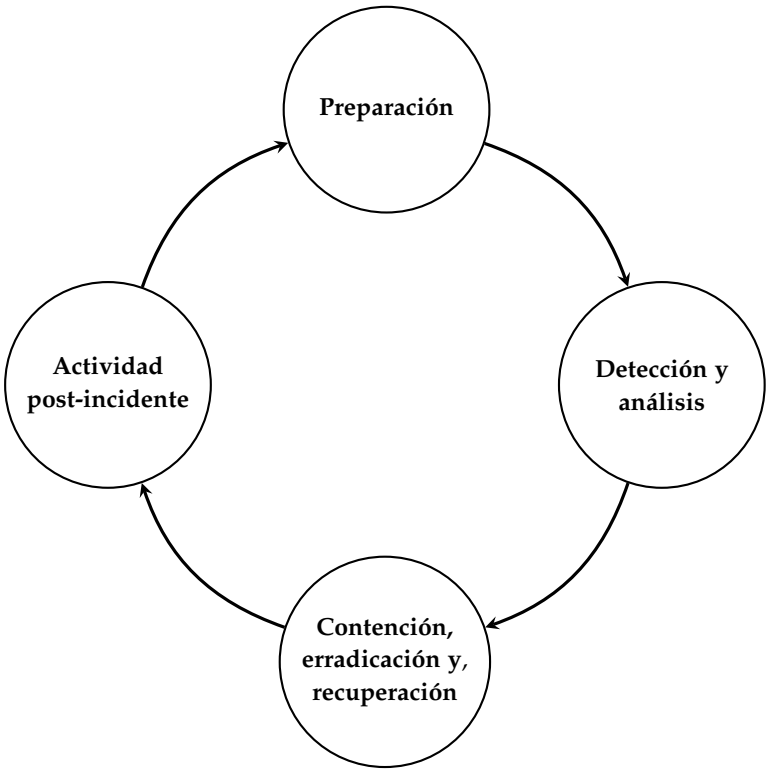

Figura 1: Ciclo de vida de respuesta ante incidentes.

3 REVISIÓN SISTEMÁTICA DE LA LITERATURA

En este capítulo se presenta la Revisión Sistemática de la Literatura realizada para este trabajo, detallando la metodología empleada para la selección y análisis de los estudios más relevantes. Posteriormente, se discuten los hallazgos y se presenta una tabla con las herramientas identificadas que sirven como base del sistema automático de análisis implementado.

3.1 METODOLOGÍA

Con el fin de identificar las herramientas más relevantes para el sistema que se desea implementar, se ha llevado a cabo una fase de investigación basada en una Revisión Sistemática de la Literatura (del inglés, *Systematic Literature Review* o SLR). Una SLR es un estudio metódico, completo, transparente y replicable, que permite la recopilación de resultados mediante investigaciones reproducibles y libres de sesgo a partir de consultas estructuradas sobre las principales bases de datos académicas.

En esta sección se detalla la metodología empleada, definiendo las preguntas de investigación, la estrategia de búsqueda en las bases de datos, los criterios de selección y finalmente la revisión de los artículos seleccionados.

3.1.1 *Preguntas de investigación*

El principal objetivo de esta SLR es realizar un estudio de todas las herramientas de análisis forense de memoria para identificación y detección de *malware* que se han propuesto a lo largo de los años, haciendo especial énfasis en aquellas que pueden ser utilizadas para construir el sistema automático de análisis objeto de este trabajo.

Para ello, se han formulado las siguientes preguntas de investigación (RQ) a las que se intenta dar respuesta mediante este estudio, y que son comentadas en detalle posteriormente.

- ***RQ1:*** *¿En qué fase del análisis de malware, si alguna, se involucra el análisis forense de memoria?*
- ***RQ2:*** *¿Cuáles son las principales herramientas o entornos utilizados para el análisis forense de memoria?*
- ***RQ3:*** *¿Cuáles son las principales limitaciones de las herramientas actuales de análisis forense de memoria?*
- ***RQ4:*** *¿Cómo se están integrando tecnologías emergentes en el análisis forense de memoria?*

3.1.2 *Estrategia de búsqueda*

Para realizar la búsqueda de artículos potencialmente relevantes, se han considerado las bases de datos académicas IEEE Xplore[1], ScienceDirect[2], ACM[3] y SpringerLink[4], utilizando sus funcionalidades de búsqueda avanzada para extraer los artículos más relevantes. Esta búsqueda avanzada permite filtrar artículos buscando que contenga diferentes combinaciones de palabras clave a partir de una cadena de texto previamente definida.

Para definir esta cadena, primero se eligen las palabras clave más interesantes para la búsqueda, y a partir de ahí se proponen distintas combinaciones mediante "**OR**" y "**AND**" para refinar la búsqueda final. Teniendo en cuenta que para este trabajo se buscan herramientas o entornos (*frameworks*) para el análisis forense de memoria y de malware, la cadena de búsqueda que se ha definido es:

("malware" **AND** "analysis" **AND** ("memory forensic" **OR** "memory forensics")) **AND** ("tool" **OR** "framework")

Tras realizar las consultas en las bases de datos, se exportan las referencias en formato RIS (*Research Information Systems*) de manera que se puedan importar fácilmente en herramientas para facilitar el procedimiento de SLR. Es importante que todas las referencias contengan, como mínimo, la información referente al título, resumen y palabras clave, ya que son necesarias para automatizar parte del proceso de filtrado de artículos.

Cabe destacar que SpringerLink no ofrece la posibilidad de exportar las referencias incluyendo los resúmenes y términos clave de cada artículo, por lo que ha sido necesario desarrollar un *script* para generar automáticamente un archivo en formato RIS con la información requerida de cada artículo. Este *script* se puede consultar en el repositorio de GitHub utilizado para este trabajo [36].

3.1.3 *Criterios de selección*

Tras obtener las referencias de todos los artículos iniciales, se ha utilizado la herramienta StArt [18, 56] con el fin de optimizar el proceso de investigación y selección de los artículos. StArt está diseñada específicamente para facilitar el proceso de revisiones sistemáticas, guiando a la persona investigadora durante las diferentes fases de la SLR e incluyendo funcionalidades como calcular automáticamente la puntuación de un artículo en base al conteo de palabras clave o la posibilidad de filtrar mediante criterios de inclusión y exclusión.

Como primera fase de filtrado, se ha configurado StArt para calcular la puntuación de cada artículo en función de la frecuencia de aparición de los términos clave definidos previamente en el título, resumen o palabras clave del artículo, y excluir aquellos que no superen un umbral mínimo establecido. El sistema de puntuación definido es el siguiente:

1 https://ieeexplore.ieee.org
2 https://www.sciencedirect.com
3 https://dl.acm.org
4 https://link.springer.com

- **5 puntos** por cada aparición de un término en el título del artículo.
- **3 puntos** por cada aparición de un término en el resumen del artículo.
- **2 puntos** por cada aparición de un término en las palabras clave del artículo.

En base a este sistema de puntuación, se considera que todos los artículos que no alcancen un umbral de al menos 20 puntos no son lo suficientemente relevantes para el estudio. Todos los demás artículos serán revisados y catalogados como incluidos o excluidos según los siguientes criterios de inclusión (IC) y exclusión (EC):

- **IC1**: El artículo se focaliza en análisis de *malware* o forense de memoria.
- **IC2**: El principal aporte del artículo es una herramienta de análisis forense de memoria.
- **EC1**: El artículo se focaliza en otros tipos de análisis en lugar de forense o malware.
- **EC2**: El artículo es corto, introductorio o un borrador no terminado.
- **EC3**: El artículo no está en inglés.

Estos criterios permiten identificar y priorizar los artículos más relevantes de cara a responder las preguntas de investigación planteadas.

3.1.4 *Revisión de artículos*

Durante la fase de búsqueda se identificaron **438** artículos entre las cuatro bases de datos seleccionadas para la investigación: **29** en IEEE Xplore, **167** en ScienceDirect, **87** en ACM y **155** en SpringerLink. De estos, **2** de ellos fueron descartados directamente por ser duplicados. Tras aplicar el umbral de 20 puntos, el número de artículos tras la fase de cribado es de **134** artículos. Posteriormente, durante la fase de elegibilidad, se seleccionaron únicamente **62** de estos artículos en función de los criterios de inclusión y exclusión definidos. Por último, tras revisar más en detalle estos artículos, se determinó que finalmente **32** de ellos proponen herramientas para el análisis forense de memoria enfocado a detección de malware. Además de estos artículos, se han incluido manualmente otras **3** herramientas halladas mediante búsquedas en Internet o mencionadas en las referencias de alguno de los artículos examinados.

El proceso descrito se representa de manera consolidada en la Figura 2, mediante un diagrama de flujo siguiendo la metodología PRISMA 2009 [35].

3.2 DISCUSIÓN

En esta sección se comentan los hallazgos obtenidos mediante la SLR y su relación con las preguntas de investigación planteadas anteriormente.

***RQ1**: ¿En qué fase del análisis de malware se involucra, si alguna, el análisis forense de memoria?*

En general, todos los artículos coinciden en que el análisis forense de memoria se realiza tras ejecutar el *malware* y obtener una imagen de la memoria RAM durante la ejecución del mismo para su posterior procesamiento. **Esto implica que la fase de análisis forense de memoria se realiza tras la fase de análisis dinámico.**

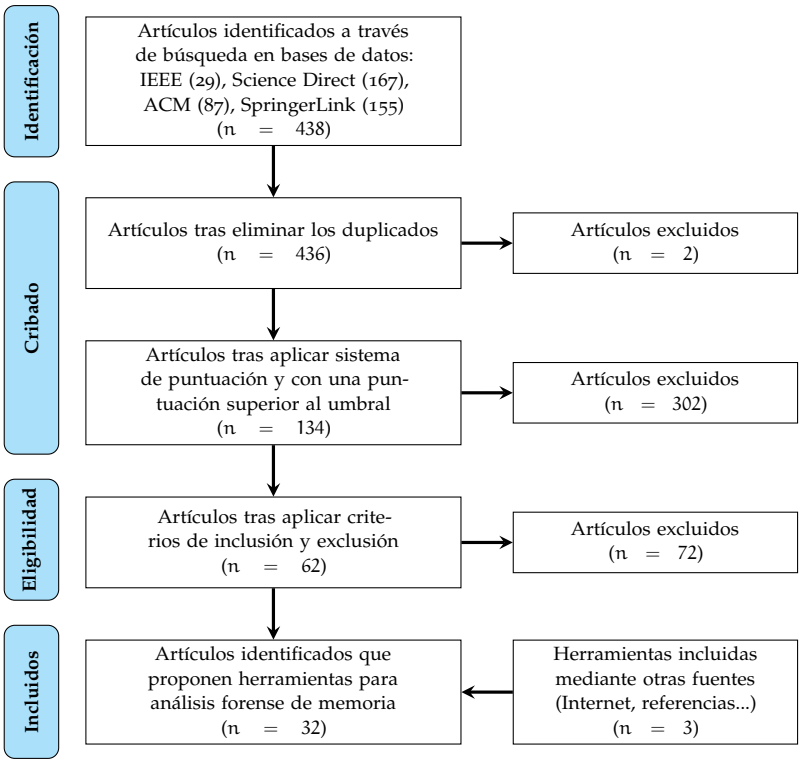

Figura 2: Diagrama PRISMA 2009 del estudio realizado.

***RQ2**: ¿Cuáles son las principales herramientas o entornos utilizados para el análisis forense de memoria?*

Como se puede ver en la Figura 3, la mitad de las propuestas en los artículos revisados consisten en *plugins* o extensiones para los principales *frameworks* estándar de análisis de memoria, como Volatility [53], Rekall [45] o Digital Forensics Framework (DFF) [3]. Entre estos, Volatility destaca considerablemente dada su extensibilidad y modularidad, que permite desarrollar *plugins* específicos e integrarlos fácilmente entre ellos. Aproximadamente, el 83% de los *plugins* son desarrollados para Volatility 2 o 3.

Por otro lado, herramientas como Rekall y DFF han caído en desuso debido a la falta de soporte por los desarrolladores, quedando Volatility como la única alternativa de código abierto viable a día de hoy. Existen otras alternativas como Memoryze (de Mandiant) [30] o Project Freta (de Microsoft) [46], pero su uso es infrecuente y, aunque son gratuitas, son alternativas propietarias, lo que limita su extensibilidad.

Dada la gran versatilidad de Volatility, la mitad de las herramientas independientes (*standalone*) lo utilizan como una parte fundamental en la extracción de artefactos de memoria mediante diversos *plugins*. El resto de las herramientas son anteriores a Volatility o proponen otras alternativas como el uso de modelos de aprendizaje automático (del inglés, *Machine Learning* o ML).

Es importante destacar que el 95% de las herramientas de código abierto estudiadas están desarrolladas en Python, lo que facilita su integración con multitud de librerías ya existentes y permite ejecutarse en diferentes plataformas, a diferencia de Memoryze y Project Freta, que solo están disponibles para entornos Windows.

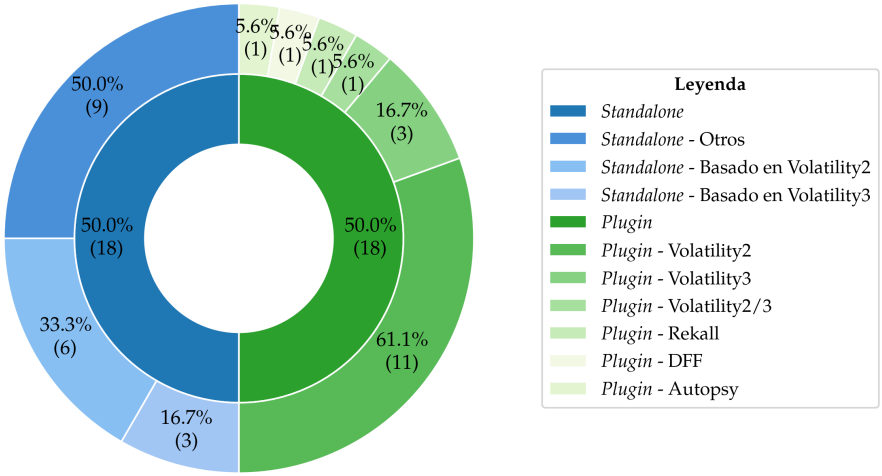

Figura 3: Proyección solar de herramientas en función de su tipo y *framework*.

***RQ3**: ¿Cuáles son las principales limitaciones de las herramientas actuales de análisis forense de memoria?*

Una de las principales limitaciones de los artículos estudiados es que la gran mayoría de las herramientas propuestas están diseñadas para analizar imágenes de memoria de sistemas Windows (véase la Figura 4). Aunque no afecta a este trabajo, esto puede ser un factor limitante a la hora de analizar muestras en otras plataformas. Estudios recientes presentan varias herramientas que permiten extraer artefactos de otras plataformas, como GPUs de NVIDIA [7], sistemas operativos móviles (en concreto, Android) [2], contenedores [22], e incluso sistemas de realidad virtual [14].

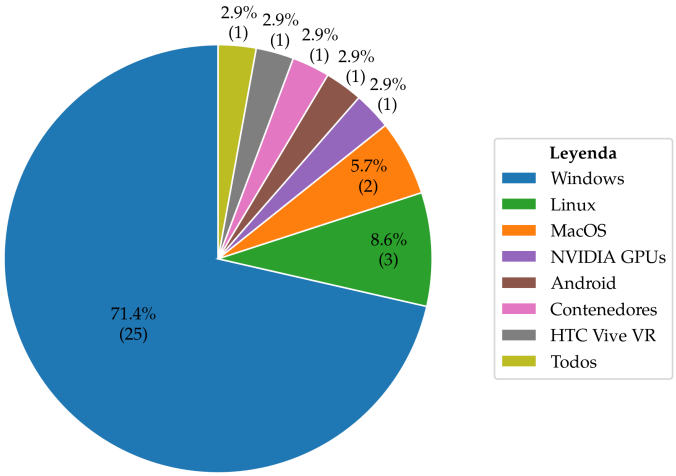

Figura 4: Distribución de herramientas por sistema objetivo.

Por otro lado, Volatility depende los símbolos de sistema y estructuras de datos en regiones concretas de memoria, por lo que no siempre se pueden analizar versiones del núcleo de sistema modificadas (como en dispositivos IoT). Varios estudios

presentan herramientas que permiten generar perfiles de memoria dinámicamente, permitiendo así utilizar Volatility con dichas imágenes [21, 41].

Por último, la mayor limitación surge de la existencia de dos versiones diferentes de Volatility: una de ellas basadas en Python 2 (ya obsoleto y fuera de soporte) y otra basada en Python 3, que es mantenida activamente. Si bien la versión anterior sigue siendo accesible, las imágenes de memoria capturadas en sistemas operativos modernos pueden no ser directamente compatibles. Debido a esto, el 61% de los *plugins* presentados en los artículos estudiados no se pueden utilizar, dado que se desarrollaron para la versión obsoleta de Volatility y no han sido migrados a la última versión (véase la Figura 3).

***RQ4**: ¿Cómo se están integrando tecnologías emergentes en el análisis forense de memoria?*

En los últimos años se han publicado varios artículos que proponen soluciones basadas en modelos de aprendizaje automático como alternativa a las herramientas tradicionales [19, 29, 32, 37, 43]. Estos modelos son entrenados con miles de muestras de *malware* previamente analizadas y categorizadas, generando un modelo capaz de identificar patrones y clasificar archivos como maliciosos o benignos sin la necesidad de una investigación manual. Sin embargo, dado que ninguno de estos artículos ha publicado el modelo o su implementación, no se ha podido evaluar su uso para el sistema a implementar.

Resultados adicionales

Por último, cabe destacar que algunos artículos proponen sistemas similares al que se presenta en este trabajo [29, 37, 38, 44]. Sin embargo, estas soluciones requieren de una investigación manual tras la ejecución, o no implementan todas las fases del análisis de *malware*. Se profundiza ligeramente sobre estos trabajos en el Capítulo 6.

3.3 HERRAMIENTAS IDENTIFICADAS

La Tabla 1 contiene la información más relevante de todas las herramientas revisadas. Esta tabla incluye, para cada herramienta: su nombre, *framework* para el que se ha desarrollado o *standalone* si es una herramienta independiente, sistema operativo objetivo del análisis, si es una herramienta FOSS (*Free and Open-Source Software*) y una breve descripción de la misma.

Tabla 1: Herramientas propuestas en los artículos estudiados.

HERRAMIENTA	FRAMEWORK	OBJETIVO	FOSS	DESCRIPCIÓN
CMAT [40]	*standalone*	Windows	✗	Extrae información de una captura de memoria (procesos, registro, usuarios...)
NVSYMMAP, checknvidia, nvoc_class_def_scan, reverse_structure_lookup [7]	Volatility2	NVIDIA GPUs	✓†	Identifica y extrae *NVIDIA Object Compiler Structures*.

†: *Es FOSS, pero no está disponible;* ‡: *Está integrada oficialmente en Volatility;* *: *Herramienta incluida externamente*

Continúa en la siguiente página

HERRAMIENTA	FRAMEWORK	OBJETIVO	FOSS	DESCRIPCIÓN
mac_interest_handlers, mac_timers, mac_devfs, mac_kernel_classes, mac_check_fop, mac_vfs_events, mac_kevents [12]	Volatility2	MacOS	✓‡	Plugins para detección de *rootkits* en MacOS.
Malfore [37]	*standalone*	Windows	✗	Framework para automatizar la extracción de artefactos de memoria utilizando Volatility.
OAGEN [2]	*standalone*	Android	✓	Identifica la procedencia y las relaciones entre objetos en memoria.
Automated-Malware-Detection-Using-Memory-Forensics [38]	*standalone*	Windows	✓	Realiza un volcado de memoria, extrae artefactos y los analiza con VirusTotal.
AutopsyVolatilityPlugin [34]	Autopsy	Windows	✓	Analiza una captura de RAM y extrae información útil para su posterior investigación.
kallsyms-extractor [41]	*standalone*	Linux	✓	Genera un perfil para escanear capturas de memoria con Volatility.
winesap [51]	Volatility2	Windows	✓	Analiza los ASEPs basados en el registro de Windows.
vadinfo [48]	Volatility2/3	Windows	✓‡	Localiza código inyectado en los nodos de estructuras VAD.
mac_observer, mac_swizzled, mac_launchd_ports [13]	Volatility2	MacOS	✓‡	Detección de *malware* en Objective-C en macOS.
ConPoint [22]	*standalone*	Containers	✓	Captura imágenes de contenedores para su análisis forense.
ubshunt, dhcphunt [49]	Volatility2	Windows	✓†	Detecta indicadores de compromiso de ataques tipo Rubber Ducky.
FACE, ramparser [9]	*standalone*	Linux	✗	Motor de correlación y extracción de artefactos de memoria en Linux.
FATkit [42]	*standalone*	Windows	✓†	Conjunto de herramientas completo para análisis forense.
Hooktracer [10]	Volatility2	Windows	✓†	Detecta sobre API *hooks* y genera firmas portables.
Hooktracer_messagehooks [11]	Volatility2	Windows	✓†	Utiliza Hooktracer para determinar si un *hook* está asociado a un *keylogger* malicioso.
vivedump [14]	Volatility2	HTC Vive	✓	Extrae artefactos en una captura de memoria de HTC Vive VR.
v8mapscan [55]	Volatility2	All	✓	Detecta *criptominers* en JavaScript.
KATANA [21]	*standalone*	Linux	✓	Alternativa a Volatility para análisis en entornos sin perfiles.
MACE [19]	*standalone*	Windows	✗	Obtiene una imagen de memoria y aplica modelo de ML para identificar *rootkits*.
MeMalDet [32]	*standalone*	Windows	✓†	Modelo de ML para categorizar *malware* en capturas de memoria.
Membrane [43]	*standalone*	Windows	✗	Detecta comportamientos de *malware* cargado en memoria utilizando ML.
dotnet_field_values, dotnet_memory_only [33]	Volatility2	Windows	✓†	Detecta *malware* en aplicaciones .NET.
modex, intermodex [20]	Volatility3	Windows	✓	Detecta ataques de *DLL hijacking*.
MRm-DLDet [29]	*standalone*	Windows	✓†	Detecta *malware* residente en memoria mediante modelo de ML.
sigcheck [52]	Volatility2	Windows	✓	Recupera archivos ejecutables de una captura de Windows.
(sin nombre) [44]	*standalone*	Windows	✗	Herramienta gráfica que captura una imagen de memoria y la analiza con Volatility.

†: Es FOSS, pero no está disponible; ‡: Está integrada oficialmente en Volatility; *: Herramienta incluida externamente

Continúa en la siguiente página

HERRAMIENTA	FRAMEWORK	OBJETIVO	FOSS	DESCRIPCIÓN
rkfinder [54]	DFF	Windows	✗	Detecta *rootkits* en sistemas Windows.
VolMemLyzer [27]	*standalone*	Windows	✓	Extrae multitud de características utilizando Volatility.
ptenum [5]	Volatility3 Rekall	Windows	✓	Identifica regiones de memoria con páginas ejecutables.
imgmalfind [4]	Volatility3	Windows	✓	Identifica modificaciones en DLLs y ejecutables hechas por *hooks* y parches.
DAMM [26]	*standalone*	Windows	✓*	Conjunto de comandos propios a partir de Volatility para investigación forense.
VolDiff [25]	*standalone*	Windows	✓*	Identifica *malware* comparando dos imágenes de memoria con Volatility.
check_spoof [15]	Volatility3	Windows	✓*	Identifica procesos maliciosos con padre (PPID) falseado.

†: *Es FOSS, pero no está disponible*; ‡: *Está integrada oficialmente en Volatility*; *: *Herramienta incluida externamente*

Para la selección de herramientas, se ha hecho especial énfasis en las herramientas publicadas y disponibles que permiten extraer artefactos de memoria en sistemas Windows, y que sean compatibles con *frameworks* mantenidos activamente, como Volatility3. Por otro lado, dado que el sistema no va a tener una interfaz gráfica, se han descartado todas aquellas que dependan de entornos gráficos. Por último, dado que el sistema de análisis se ejecuta desde una máquina GNU/Linux, se descartan todas las herramientas que solo se pueden ejecutar desde Windows.

La Tabla 2 presenta el listado final de herramientas para la fase de análisis de memoria. Aunque no se incluyen en la tabla, también se utilizan varios *plugins* que se incluyen por defecto en Volatility3. La lista completa se detallará en el Capítulo 4.

HERRAMIENTA	URL	DESCRIPCIÓN
vadinfo [48]	`https://github.com/volatilityfoundation/volatility3`	Localiza código inyectado en los nodos de estructuras VAD.
modex, intermodex [20]	`https://github.com/reverseame/modex`	Detecta ataques de *DLL hijacking*.
ptenum [5]	`https://github.com/f-block/volatility-plugins`	Identifica regiones de memoria con páginas ejecutables.
imgmalfind [4]	`https://github.com/f-block/volatility-plugins`	Identifica modificaciones en DLLs y ejecutables hechas por *hooks* y parches.
check_spoof [4]	`https://github.com/orchechik/check_spoof`	Identifica procesos con padre (PPID) falseado.

Tabla 2: Herramientas seleccionadas para el sistema de análisis.

4 SISTEMA DE ANÁLISIS

En este capítulo se presenta el sistema de análisis implementado, primero mediante una descripción de alto nivel y posteriormente detallando las herramientas utilizadas, la configuración de la plataforma y el funcionamiento del *script*.

4.1 DESCRIPCIÓN

El sistema que se propone en este trabajo consiste en un sistema automatizado que integra las fases tradicionales de análisis de *malware*, junto con una fase adicional de análisis forense de memoria. La Figura 5 representa un esquema general del funcionamiento de este sistema.

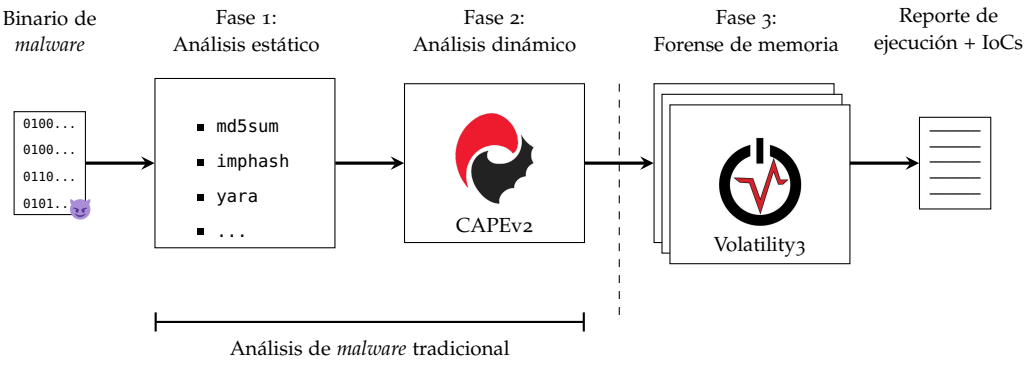

Figura 5: Diagrama del sistema de análisis

El flujo comienza con la introducción de un archivo potencialmente malicioso en el sistema, donde se somete a una primera fase de **análisis estático**. Durante esta fase, se aplican varias herramientas sobre el archivo sin ejecutarlo, lo que permite extraer diferentes IoCs como *hashes* del fichero, cadenas de texto, librerías utilizadas, etc. Estos análisis son generalmente rápidos, por lo que se pueden ejecutar de manera secuencial en unos pocos segundos.

Tras esta primera fase, el archivo es sometido a un **análisis dinámico**, en la que se estudia su comportamiento durante su ejecución en un entorno controlado y aislado, comúnmente denominado *sandbox*. Este entorno permite ejecutar muestras de *malware* y registrar todos los cambios que el *malware* realiza sobre el sistema infectado (instalación de *software*, conexiones de red, modificación de registros, etc.). Durante esta fase que se captura una imagen del estado de la memoria RAM para su posterior análisis forense.

Finalmente, se lleva a cabo una tercera fase dedicada al **análisis forense de memoria**. En esta etapa, se procesan las imágenes de memoria capturadas en la fase anterior utilizando las herramientas identificadas en la SLR, con el fin de extraer indicadores adicionales que no han sido detectados durante las fases anteriores. Dado que cada herramienta debe procesar toda la memoria RAM de la máquina, este proceso puede ser muy lento dependiendo del número de herramientas a

utilizar. Para ello, se ha optimizado el sistema de manera que se realizan varios análisis en paralelo, obteniendo los resultados en un menor tiempo.

El sistema concluye recopilando los resultados de las tres fases en un reporte exhaustivo con todos los indicadores (IoCs) extraídos, junto con una posible clasificación del *malware* analizado a partir de las características detectadas durante su análisis.

4.2 PLATAFORMA DE EXPERIMENTACIÓN

El sistema se ha implementado sobre dos máquinas virtuales (MV) en un servidor de Proxmox VE[1]. Proxmox es un hipervisor de tipo 1 de código abierto basado en QEMU-KVM, lo que permite gestionar la interfaz de virtualización a bajo nivel y ofrece un rendimiento similar al *hardware* real. Una de estas máquinas es la "máquina de ejecución" (**Cuckoo1**) en la que se ejecuta la muestra de *malware*, mientras que la otra es la "máquina de análisis" (**CAPEv2**), en la que se analizan los resultados. Las especificaciones tanto del servidor como los recursos asignados a cada MV se encuentran detallados en la Tabla 3.

TIPO	NOMBRE / SO	ESPECIFICACIONES
Host	**Proxmox** Proxmox VE 8.2.4	16 CPU (AMD Ryzen 5800x 5.8GHz) 64 GB RAM DDR4 3200MHz 2 TB NVMe 2.5Gbit NIC (asociada a vmbro) - 192.168.0.195
Guest	**CAPEv2** Ubuntu 24.04 LTS	12 vCPU (host passthrough) 16 GB RAM 128 GB (virtual SSD) virtio NIC 0 (asociada a vmbro) - 192.168.0.109 virtio NIC 1 (asociada a vmbr1) - 10.0.0.1
Guest	**Cuckoo1** Windows 10 Enterprise (evaluation)	8 vCPU (host passthrough) 8 GB RAM 128 GB (virtual SSD) e1000 NIC (asociada a vmbr1) - 10.0.0.2

Tabla 3: Especificaciones de la plataforma de experimentación

El servidor está configurado con dos interfaces de tipo *bridge*, `vmbr0` y `vmbr1`. La interfaz `vmbr0` está asociada a la red LAN, lo que le otorga direcciones IP accesibles desde fuera. Por otro lado, la interfaz `vmbr1` es una interfaz interna que se utiliza para interconectar ambas MVs, de manera que puedan comunicarse entre ellas, pero evitando que cualquier tráfico malicioso circule por la red LAN, siguiendo buenas prácticas en investigación con *malware* [6].

4.3 CONJUNTO DE HERRAMIENTAS

Para las fases de análisis estático y dinámico, se utilizan algunas de las herramientas estándar en análisis de *malware* para cada una de ellas, mientras que la última fase de análisis se apoya en las herramientas identificadas en la Sección 3.3.

1 Véase https://www.proxmox.com/en/proxmox-virtual-environment/overview

La Tabla 4 contiene un listado de todas las herramientas que se van a utilizar en este sistema y su descripción, clasificadas por las fases en las que se aplica.

FASE	HERRAMIENTA	DESCRIPCIÓN
Estática	virustotal	Análisis de *malware* y reputación (online).
	avclass	Clasificación de familias de *malware*.
	flare-capa	Detecta funcionalidades en archivos ejecutables.
	flare-floss	Extrae cadenas de texto ofuscadas en binarios.
	exiftool	Extrae metadatos de archivos.
	file	Identifica el tipo y formato de archivos.
	md5sum/sha265sum	Generan *hashes* criptográficos de archivos.
	yara	Detecta patrones definidos en reglas.
	rabin2	Extrae detalles de bajo nivel en binarios.
	imphash	Genera un *hash* de librerías importadas por el binario.
	die	Identifica características y empaquetadores.
	ssdeep	Detecta similitudes entre archivos mediante *hashes* de similud aproximada.
Dinámica	CAPEv2	*Sandbox* para análisis dinámico de *malware*.
Forense	volatility3	*Framework* de análisis forense de memoria.
	windows.pslist	Lista los procesos activos (*plugin*).
	windows.pstree	Estructura jerárquica los procesos activos (*plugin*).
	windows.dllist	Lista las librarías cargadas en procesos (*plugin*).
	windows.handles	Lista los identificadores abiertos en un proceso (*plugin*).
	windows.ldrmodule	Detalla módulos cargados por un proceso (*plugin*).
	windows.malfind	Identifica inyecciones de código en procesos (*plugin*).
	windows.hollowprocesses	Detecta procesos vaciados (*plugin*).
	windows.filescan	Lista los archivos encontrados en la captura (*plugin*).
	windows.netscan	Lista conexiones y puertos activos (*plugin*).
	windows.ptemalfind	Identifica inyecciones de código mediante PTE (*plugin*).
	windows.modex	Detecta ataques de DLL *hijacking* en módulos (*plugin*).
	windows.check_parent_spoofing	Detecta procesos con padre (PPID) falseado (*plugin*).
	windows.check_peb_spoofing	Detecta procesos con PEB falseado (*plugin*).
	windows.vadinfo	Localiza código inyectado en estructuras VAD (*plugin*).
	yarascan.YaraScan	Aplica patrones YARA sobre la memoria (*plugin*).

Tabla 4: Listado completo de herramientas para el sistema de análisis.

4.4 CONFIGURACIÓN DEL HIPERVISOR

Para permitir la captura de imágenes de memoria de manera programática durante la ejecución, es necesario el uso de UNIX *sockets* junto con la interfaz QEMU Machine Protocol (QMP) para interactuar directamente con QEMU-KVM y ejecutar el comando de bajo nivel `dump-guest-memory`. Este comando pausa momentáneamente el estado de la MV mientras se realiza la captura de memoria, y continúa con la ejecución de manera totalmente transparente al SO virtualizado. A diferencia de otros métodos de captura, este método asegura que la imagen de memoria es una **copia idéntica**, lo que aumenta considerablemente las probabilidades de éxito del posterior análisis [50].

Para controlar el momento de la captura es necesario el uso de un *hookscript* personalizado, que permite realizar tareas adicionales durante las cuatro fases de vida de la máquina virtual: *pre-start*, *post-start*, *pre-down* y *post-down*. De esta manera,

cuando la máquina arranca, se ejecuta el *hookscript* que lanza otro *script* en segundo plano para capturar periódicamente imágenes de la memoria, comprimirlas, y enviarlas al sistema de análisis mientras transcurre la ejecución. Tras finalizar la ejecución, el *hookscript* detiene el proceso de captura hasta que se vuelva a arrancar la MV. El código de estos *scripts* está publicado en el repositorio de este trabajo [36].

4.5 CONFIGURACIÓN DEL ENTORNO DE ANÁLISIS: UBUNTU 24.04

El entorno de análisis es la máquina virtual que se encarga de coordinar y ejecutar todos los pasos del sistema, analizar los resultados, y recopilar los indicadores para generar el reporte final. Para este entorno se ha utilizado la última versión de **Ubuntu 24.04** y se ha configurado la red de manera estática con IP **192.168.0.109** para la interfaz externa, y **10.0.0.1** para la interna.

Tras esto, se han instalado todas las herramientas necesarias para las fases de análisis estático y forense de memoria. De cara a facilitar su replicación, se ha preparado un *script* en el repositorio de GitHub [36] con instrucciones para el despliegue y configuración de las herramientas.

Por último, se ha instalado y configurado el *sandbox* **CAPEv2** [39], que realiza la función de análisis dinámico y coordina la ejecución del *malware* en la máquina de ejecución. Dada la complejidad de esta herramienta, su instalación no es trivial y se han encontrado grandes dificultades hasta hacerla funcionar correctamente.

4.6 CONFIGURACIÓN DEL ENTORNO DE EJECUCIÓN: WINDOWS 10

El entorno de ejecución es la máquina virtual que se encargará de ejecutar el *malware* para recopilar y transferir los resultados a la máquina de análisis. Para este entorno se ha utilizado la última versión de **Windows 10 Enterprise** en modo evaluación, y se ha configurado una única red de manera estática con IP **10.0.0.2**. Para asegurar que todo el tráfico de red pasa por la máquina de análisis, también se ha configurado la IP **10.0.0.1** como puerta de enlace y servidor DNS.

Para asegurar que el SO no interfiere en la ejecución del *malware*, es necesario aplicar una serie de modificaciones para deshabilitar los mecanismos de seguridad que vienen activados por defecto. En líneas generales, los cambios aplicados son los siguientes:

- Deshabilitar las actualizaciones automáticas mediante GPO.
- Deshabilitar la protección en tiempo real (antivirus) mediante GPO.
- Deshabilitar la solicitud de aprobación de escalada de privilegios y autorizar directamente privilegios de administrador (UAC) mediante GPO.
- Deshabilitar el Firewall de Windows en redes públicas y privadas.
- Instalar y configurar Autologon[2] para inicio de sesión automático sin introducir la contraseña.
- Eliminar servicios innecesarios utilizando un *script* de *debloating*[3].

2 https://learn.microsoft.com/en-us/sysinternals/downloads/autologon

3 Proceso de eliminación de *software* pre-instalado y servicios innecesarios. Véase https://github.com/W4RH4WK/Debloat-Windows-10

Adicionalmente, se han instalado otras aplicaciones comúnmente explotadas por *malware*, como navegadores web, lectores de PDF o extractores de archivos.

Para recopilar todos las modificaciones y eventos que ocurren durante la ejecución, es necesario instalar una versión de Python superior a 3.6 y configurar una tarea programada para lanzar un agente de Python durante el inicio de sesión. Este agente expone una interfaz web mediante la que la máquina de análisis interactúa con la máquina de ejecución, pudiendo así enviar los archivos maliciosos, ejecutarlos y recibir el resultado.

Por último, tras finalizar la configuración, es necesario apagar la máquina virtual y crear un *snapshot* de la misma. De esta manera, el estado original se restaurará tras cada análisis.

4.7 SCRIPT DE EJECUCIÓN Y ANÁLISIS

Una vez preparados los entornos de análisis y ejecución, se ha implementado el *script* que se encarga de coordinar la ejecución del sistema de análisis. Acorde con las herramientas investigadas durante la SLR, se ha decidido usar el lenguaje de programación Python para el desarrollo, debido su extensibilidad y compatibilidad con multitud de librerías y *frameworks* ya existentes.

Dada la extensión del código, únicamente se explicarán por encima secciones más relevantes para este trabajo. El código completo se puede consultar en el repositorio de GitHub de este trabajo [36].

Opciones y configuración

A la hora de programar el *script*, se ha intentado programar con la mayor modularidad posible, permitiendo al usuario seleccionar las herramientas que se desean en cada fase, así como elegir si ejecutar una fase individualmente o todas las fases del análisis.

La selección de herramientas se define mediante un fichero de configuración, que contiene el listado de las herramientas a utilizar en cada una de las fases. Cada herramienta lleva asociado un valor booleano indicando si se debe ejecutar o no. Por otro lado, para la fase 3 (análisis forense de memoria), se indican los nombres de los *plugins* de Volatility, junto con los argumentos necesarios para ejecutarlo, así como el valor booleano que permite habilitar o deshabilitar su ejecución. Dado que la fase de análisis dinámico se basa en una única herramienta, no es necesaria ninguna configuración adicional.

```
[Phase1]
virustotal = true
avclass = true
flare_capa = true
...

[Phase3]
windows.pslist.PsList = true
windows.malfind.Malfind,pid,args,--dump-dir malfind_dumps = true
```

```
windows.cmdline.CmdLine,pid = true
...
```

Para su ejecución, únicamente es obligatorio indicar la ubicación del archivo a analizar. No obstante, se ha añadido la posibilidad de controlar la ejecución mediante otros parámetros. Estos parámetros permiten, entre otras cosas, cambiar la ubicación de los resultados o elegir específicamente la fase del análisis que se desea ejecutar. Por defecto, se ejecutan todas las fases del análisis.

Flujo del programa

El programa inicia con la validación argumentos para seleccionar las fases que se van a ejecutar. Si los argumentos son correctos, se inicia el análisis ejecutando una por una las funciones asociadas a cada fase. Durante cada fase, se imprimen por pantalla las trazas asociadas a cada herramienta. Tras finalizar el análisis, se genera el reporte con los resultados y finaliza la ejecución del programa.

Fase 1: Análisis estático

Durante esta fase, se validan las herramientas habilitadas en el fichero de configuración y se lanza su ejecución en paralelo, almacenando los resultados en la carpeta destino seleccionada. Para ejecutar cada herramienta se hace uso de una función auxiliar, que recibe el comando necesario, lo ejecuta en segundo plano, y almacena los resultados en un fichero de texto.

Fase 2: Análisis dinámico

Dado que el análisis dinámico se realiza mediante el *sandbox* CAPE, durante esta fase se lanza el análisis y se mantiene a la espera hasta recopilar el reporte de resultados y filtrar los resultados más relevantes. Estos resultados se vuelcan a un fichero de texto para su posterior revisión.

Al mismo tiempo que se lanza el análisis, se configura un servidor HTTP a la espera de recibir la captura de memoria realizada por el hipervisor. Una vez recibida, se almacena localmente y se cierra el servidor.

Fase 3: Análisis forense de memoria

Esta fase es la más compleja de todas, ya que requiere de varios pasos previos para preparar el análisis, así como un filtrado exhaustivo para generar IoCs válidos.

El primer paso consiste en validar que se ha recibido la captura de memoria realizada por el hipervisor y descomprimirla para trabajar con ella. Tras esto, se extraen todos los procesos de la captura y se busca el proceso correspondiente al agente de Python que corre en la máquina de ejecución y es el encargado de ejecutar el *malware*. A partir de este proceso se obtienen también sus procesos hijos. De esta manera, es posible focalizar los esfuerzos de análisis únicamente en los procesos que han sido generados por el *malware*.

Una vez obtenidos los PIDs de los subprocesos a analizar, se ejecutan en paralelo los *plugins* de Volatility que se encuentren definidos en el fichero de configuración. Con el fin de optimizar la ejecución de los *plugins*, cada uno de ellos ejecuta en un hilo separado del procesador mediante una función auxiliar, lo que optimiza el

tiempo del análisis.

Por último, para obtener únicamente la información más relevante de cada uno de los *plugins*, se hace uso de una función auxiliar que busca patrones definidos mediante expresiones regulares. Estos patrones permiten identificar los posibles indicadores de compromiso. Por ejemplo, el uso del patrón `"base64|EncodedCommand|wscript|powershell|cmd\.exe"` permite encontrar comandos ejecutados mediante varias herramientas o codificados en base64.

Los resultados obtenidos de cada uno de los *plugins* se almacenan en ficheros de texto individuales dentro de la carpeta de destino seleccionada, tanto el resultado filtrado como el resultado original. Además, si el *plugin* admite extracción de binarios, estos se almacenan en un directorio a parte.

Reporte

Tras finalizar la ejecución de todas las fases, se hace uso de una función auxiliar que se encarga de recopilar algunos de los resultados individuales de las herramientas y genera un fichero de texto que contiene un resumen del análisis completo. Este resumen presenta los principales indicadores de compromiso extraídos automáticamente. El detalle de los resultados se encuentra en los archivos generados de cada fase.

5 CASO DE ESTUDIO: ZEUS

En este capítulo, se pone a prueba el sistema implementado con un caso de estudio analizando una muestra real de *malware* conocido. Posteriormente, se detallan las limitaciones que presenta este sistema.

5.1 RESULTADOS DE EJECUCIÓN

Como caso de uso, se ha analizado una muestra del *malware* **Zeus** (también conocido como Zbot), con MD5 `57ae9fb232797897223c72c89a5b243f`. Se trata de un troyano bancario diseñado para robar credenciales financieras mediante técnicas como el *keylogging* y la inyección de código en navegadores. Esta muestra se ha obtenido de la base de datos de MalwareBazaar [57] y se ha ubicado en un directorio de la máquina de análisis.

La Figura 6 representa la traza generada por el sistema de análisis, en el que se puede distinguir cada una de las fases y las herramientas utilizadas en cada una de ellas. El tiempo de ejecución del sistema completo para este caso ha sido de aproximadamente unos 4 minutos.

```
(python3-tools) capa@capev2:~/automated-ioc-extractor$ time ./ioc_extractor.py -f /opt/CAPEv2/samples/zeus.exe -o zeus --vt-api-key b649fc57791c1bdc2c6f5b99679e6d7abf692d44219a234e68
Starting static analysis (phase1) ...
==================================
Executing command: tools/capa -v /opt/CAPEv2/samples/zeus.exe
Executing command: floss --minimum-length 7 /opt/CAPEv2/samples/zeus.exe
Executing command: exiftool /opt/CAPEv2/samples/zeus.exe
Executing command: file /opt/CAPEv2/samples/zeus.exe
Executing command: md5sum /opt/CAPEv2/samples/zeus.exe
Executing command: sha256sum /opt/CAPEv2/samples/zeus.exe
Executing command: yara yara-rules-full.yar /opt/CAPEv2/samples/zeus.exe
Executing command: python -c "import pefile, sys; print(pefile.PE(sys.argv[1]).get_imphash())" /opt/CAPEv2/samples/zeus.exe
Executing command: rabin2 -g /opt/CAPEv2/samples/zeus.exe
Executing command: tools/Detect-It-Easy/docker/diec.sh -e -j /opt/CAPEv2/samples/zeus.exe
Executing command: curl https://www.virustotal.com/api/v3/files/abd0ad67b1f1332a44c6254324844dceaf3c35396509149fecc4084c58ac3fa0 --header 'X-Apikey: b649fc57791c1bdc2c6f5b99679e6d7ab
Executing command: ssdeep /opt/CAPEv2/samples/zeus.exe
Executing command: avclass -f zeus/static/virustotal_result.txt
Executing command: sed -i '/===== PROFILING INFORMATION =====/,$d' zeus/static/yara_result.txt
==================================
Static analysis completed. Results are available in zeus/static
==================================
Starting dynamic analysis (phase2) ...
==================================
Server starting on port 8888, dump path: /opt/CAPEv2/storage/analyses/9/memory/memdump.raw.zst
Waiting for /opt/CAPEv2/storage/analyses/9/reports/report.json to exist...
Waiting for /opt/CAPEv2/storage/analyses/9/reports/report.json to exist...
Waiting for /opt/CAPEv2/storage/analyses/9/reports/report.json to exist...
Waiting for /opt/CAPEv2/storage/analyses/9/reports/report.json to exist...
Waiting for /opt/CAPEv2/storage/analyses/9/reports/report.json to exist...
192.168.0.195 - - [26/Nov/2024 23:17:51] "POST / HTTP/1.1" 200 -
Stopping server after receiving the file...
Waiting for /opt/CAPEv2/storage/analyses/9/reports/report.json to exist...
Waiting for /opt/CAPEv2/storage/analyses/9/reports/report.json to exist...
Waiting for /opt/CAPEv2/storage/analyses/9/reports/report.json to exist...
Waiting for /opt/CAPEv2/storage/analyses/9/reports/report.json to exist...
Memory dump received.
==================================
Dynamic analysis completed. Results are available in zeus/dynamic
==================================
Starting memory forensics (phase3) ...
==================================
Memory dump file ready to process.
Executing command: ./tools/volatility3/vol.py -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -r json windows.pslist.PsList
Running analysis on these PIDs: [6768]
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.malfind --pid 6768 --dump
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.cmdline --pid 6768
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.dlllist --pid 6768
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.handles --pid 6768
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.ldrmodule --pid 6768
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.ptemalfind --pid 6768 --dump
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.check_parent_spoofing --pid 6768
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.vadinfo --pid 6768
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.hollowprocesses --pid 6768
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.dumpfiles --pid 6768
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.check_peb_spoofing --pid 6768
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps yarascan.yarascan --yara-file yara-rules-full.yar
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.modex --pid 6768 --detect --module kernel32.dll
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.netscan
Executing command: ./tools/volatility3/vol.py -q -f /opt/CAPEv2/storage/analyses/9/memory/memdump.raw -o zeus/memory/dumps windows.filescan
==================================
Memory forensics analysis completed. Results are available in zeus/memory
==================================
```

Figura 6: Traza de ejecución del sistema de análisis

Tras la finalización del análisis, los resultados se presentan en un reporte en formato de texto. Este reporte, presenta en primer lugar información básica sobre el binario para su identificación, extraída mediante el análisis estático (Figura 7), y

posteriormente detalla información relevante extraída del análisis dinámico, como el número de archivos modificados o claves de registro leídas (Figura 8). Finalmente, dado que el análisis forense genera gran cantidad de archivos, únicamente se presenta la estructura de directorios de los archivos generados para la posterior revisión de la persona investigadora. La Figura 9 presenta una vista simplificada del reporte de esta fase.

```
(python3-tools) cape@capev2:~/automated-ioc-extractor/zeus$ cat summary_report.txt

This is a summary report. For the full list of indicators, please check output folder: zeus

#### Phase 1: Static Analysis ####

+-----------------------------+-----------------------------------------------------------------------------------------+
| Attribute                   | Value                                                                                   |
+-----------------------------+-----------------------------------------------------------------------------------------+
| File Name                   | zeus.exe                                                                                |
| Directory                   | /opt/CAPEv2/samples                                                                     |
| File Size                   | 99 kB                                                                                   |
| File Modification Date/Time | 2024:11:26 22:15:22+00:00                                                               |
| File Access Date/Time       | 2024:11:26 22:17:34+00:00                                                               |
| File Inode Change Date/Time | 2024:11:26 22:15:45+00:00                                                               |
| File Permissions            | -rw-r--r--                                                                              |
| File Type                   | Win32 EXE                                                                               |
| File Type Extension         | exe                                                                                     |
| MIME Type                   | application/octet-stream                                                                |
| Machine Type                | Intel 386 or later, and compatibles                                                     |
| Time Stamp                  | 2007:12:09 13:46:53+00:00                                                               |
| Image File Characteristics  | No relocs, Executable, 32-bit                                                           |
| PE Type                     | PE32                                                                                    |
| Linker Version              | 9.0                                                                                     |
| Code Size                   | 72192                                                                                   |
| Initialized Data Size       | 30208                                                                                   |
| Uninitialized Data Size     | 0                                                                                       |
| Entry Point                 | 0x6101                                                                                  |
| OS Version                  | 4.0                                                                                     |
| Image Version               | 1.0                                                                                     |
| Subsystem Version           | 4.0                                                                                     |
| Subsystem                   | Windows GUI                                                                             |
| MD5 Hash                    | 57ae9fb232797897223c72c89a5b243f                                                        |
| SHA256 Hash                 | abd0ad67b1f1332a44c6254324844dceaf3c35396509149fecc4084c58ac3fa0                        |
| SSDEEP Hash                 | 1536:G4Baw30BsAJzbQrmYn4IFOsxUHOWxGdLaIKateRv3lwIT9SEp:pawmsAB2nMKUHzGJaHa4lw4J        |
| ImpHash                     |                                                                                         |
| PE Status                   | packed                                                                                  |
| Entropy                     | 6.63                                                                                    |
| AVClass                     | zbot                                                                                    |
| YARA_1                      | MALPEDIA_Win_Zeus_Auto /opt/CAPEv2/samples/zeus.exe                                     |
+-----------------------------+-----------------------------------------------------------------------------------------+
```

Figura 7: Indicadores obtenidos de la fase 1

```
#### Phase 2: Dynamic Analysis ####

Apis Resolved
==========
- Total Indicators: 0
- No indicators found.

Commands Executed
================
- Total Indicators: 2
- Examples:
  1. -c -s 0 -f 0 -t Empty -m Empty -a 0 -u Empty
  2. wlrmdr.exe -c -s 0 -f 0 -t Empty -m Empty -a 0 -u Empty

Domains
=======
- Total Indicators: 1
- Examples:
  1. www.msftconnecttest.com: 104.86.110.217

Files
=====
- Total Indicators: 12
- Examples:
  1. C:\Windows\Globalization\Sorting\sortdefault.nls
  2. C:\Windows\System32\windows.storage.dll
  3. C:\Users\admin\AppData\Local\Temp\Wldp.dll

Files Delete
==========
- Total Indicators: 1
- Examples:
  1. C:\Windows\System32\sdra64.exe

Files Read
=========
- Total Indicators: 1
- Examples:
  1. C:\Users\admin\AppData\Local\Temp\zeus.exe

Files Write
==========
- Total Indicators: 1
- Examples:
  1. C:\Windows\System32\sdra64.exe

Hosts
=====
- Total Indicators: 0
- No indicators found.

Keys
====
- Total Indicators: 58
- Examples:
  1. HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager
  2. HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Session Manager\ResourcePolicies
  3. HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Nls\CustomLocale
```

Figura 8: Indicadores obtenidos de la fase 2

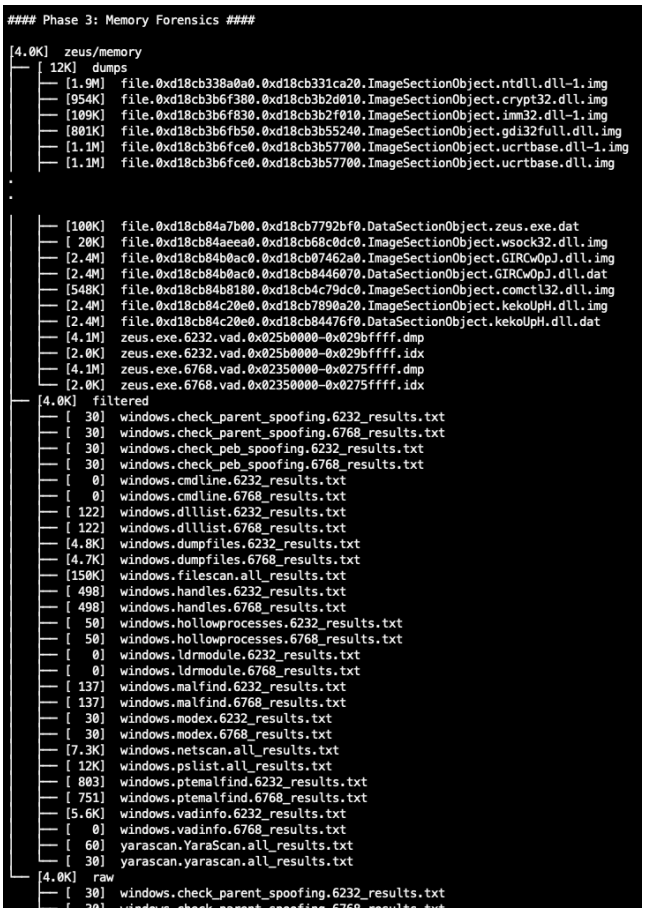

Figura 9: Indicadores obtenidos de la fase 3

5.2 LIMITACIONES

Una de las limitaciones encontradas es que, dado que las capturas de memoria se realizan tras un tiempo de espera fijo, puede darse el caso que no se haya ejecutado el *malware* todavía, o que este ya hubiera eliminado todas las trazas, lo que afectaría a los indicadores obtenidos. Un ejemplo de esto se puede ver en la Figura 8: el análisis dinámico ha detectado la creación de un binario **sdra64.exe** que posteriormente ha sido eliminado. Sin embargo, en la captura de memoria no se ha encontrado ningún indicador relacionado a este binario. Como posible solución, se puede hacer uso del agente instalado en la máquina de ejecución para coordinar el momento de la captura, y hacerlo justo tras la ejecución del *malware*.

Otra de las limitaciones de este sistema ocurre por el funcionamiento intrínseco de CAPE, dado que para poder analizar el comportamiento de los binarios, realiza comportamientos que podrían ser considerados maliciosos, como inyectar código en los procesos a ejecutar. Estos artefactos aparecen en las capturas realizadas y son detectados como indicadores, por lo que se considerarían falsos positivos. Una posible solución a este problema es intentar acotar los procesos de CAPE e introducirlos en una lista de procesos permitidos para evitar su detección. De igual manera, también se puede aplicar este filtrado para eliminar falsos positivos producidos por procesos legítimos del sistema.

Por otro lado, otra de las limitaciones de este sistema es la necesidad de preparar todo el entorno manualmente. Dada la complejidad del mismo, esto requiere gran cantidad de tiempo y pruebas para validar su funcionamiento correcto. Esto afecta considerablemente a la portabilidad y replicabilidad del sistema. Aunque se ha automatizado parte del proceso de instalación de herramientas, una posible mejora del sistema consiste en automatizar el despliegue del entorno completo utilizando herramientas de *Infrastructure as Code*, como Terraform, Ansible, etc., o el uso de contenedores.

Por último, las herramientas de forense de memoria no son infalibles a la hora de detectar *malware* avanzado, y generalmente requieren una investigación manual para confirmar si es malicioso o no. Aunque este sistema automático consigue obtener indicadores fácilmente, una posible mejora consiste en integrar con herramientas de aprendizaje automático para conseguir una mayor fiabilidad durante los análisis.

6 TRABAJO RELACIONADO

Durante el estudio realizado para este trabajo se han encontrado varios artículos en los que se presentan herramientas similares al sistema propuesto en este trabajo [29, 37, 38, 44]. Sin embargo, ninguno de ellos propone un enfoque de análisis completo que incluya las tres fases, sino que se centran en capturar la imagen de memoria y realizar un análisis forense sobre la misma.

En [44] se presenta una herramienta que recibe una imagen de memoria previamente adquirida, sobre la que se aplican varios *plugins* nativos de Volatility2 con el fin de hallar artefactos sospechosos (procesos, bibliotecas compartidas, etc.). Tras esto, se aplican reglas YARA para detectar si contienen firmas de *malware*. Por otro lado, los autores de [38] proponen otra herramienta muy similar, pero que utiliza Volatility3 para hallar los procesos sospechosos y extraer sus binarios. A diferencia del artículo anterior, estos binarios son analizados utilizando el servicio de VirusTotal, lo que ofrece un análisis más completo de la muestra.

En [29, 37] se propone un enfoque diferente, utilizando algoritmos de aprendizaje automático para determinar si se trata de *malware*. En [37], utilizan *Cuckoo* (versión obsoleta de CAPE) para ejecutar la muestra y extraer la memoria, para posteriormente analizar la imagen con Volatility2 y extraer artefactos. Tras extraer los artefactos, estos son clasificados mediante distintos algoritmos de aprendizaje automático. Por otro lado, el enfoque de [29] es, utilizando VMWare, capturar la memoria antes y después de la ejecución y generar una imagen RGB de los cambios entre ambas. Tras esto, la imagen se clasifica como malware o no mediante el uso de una red neuronal entrenada con miles de muestras.

7 CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se exponen las conclusiones del trabajo y posibles líneas de investigación a partir del mismo.

7.1 CONCLUSIONES

Tras un incidente de seguridad, el análisis de *malware* juega un papel fundamental a la hora de mejorar los sistemas de detección ante futuras amenazas. Para ello, en este trabajo se ha desarrollado un sistema automático para el análisis de *malware* estructurado en tres fases: análisis estático, análisis dinámico y análisis forense de memoria. Este enfoque combina técnicas clásicas de análisis de *malware* junto con el análisis forense de memoria para la extracción de indicadores de compromiso.

Para la implementación de la fase de análisis forense de memoria, previamente se ha llevado a cabo una revisión sistemática de la literatura, identificando las principales virtudes y limitaciones de las herramientas propuestas en la literatura, así como el estudio de sistemas similares para integrar las mejores prácticas en este sistema a desarrollar. Tras la implementación del sistema, se ha validado mediante el análisis de una muestra de *malware* real, demostrando así que cumple su propósito satisfactoriamente.

No obstante, este sistema aún tiene margen de mejora, particularmente en la identificación de falsos positivos y la portabilidad del entorno de análisis. A pesar de estas limitaciones, el sistema constituye un avance significativo en la automatización del análisis de *malware* y la extracción de indicadores de compromiso, sentando las bases para futuras investigaciones en este ámbito.

7.2 TRABAJO FUTURO

Una de las principales áreas de mejora es el estudio de los comportamientos realizados por CAPE y el SO, de manera que se puedan filtrar los falsos positivos y permitir la extracción de indicadores de mayor calidad.

Por otro lado, otra posible línea es la automatización completa del despliegue del entorno de análisis mediante herramientas de *Infrastructure as Code* o mediante el uso de contenedores con todas las herramientas necesarias. Estas estrategias mejorarían la portabilidad, replicabilidad y facilidad de uso del sistema.

Finalmente, dado que las herramientas de análisis forense de memoria no siempre detectan amenazas avanzadas, se puede explorar la integración del sistema con modelos de aprendizaje automático. Esto permitiría mejorar la detección y extracción de indicadores de compromiso y reducir la necesidad de investigación manual, aumentando la eficacia y precisión del análisis forense. Con estas mejoras, se optimizaría significativamente la utilidad del sistema.

BIBLIOGRAFÍA

[1] A. Afianian, S. Hashemi, A. Hamzeh y M. Conti, «An emerging threat: Fileless malware—a survey and research challenges», *Cybersecurity*, vol. 3, n.º 1, págs. 1-20, 2020. DOI: 10.1186/s42400-019-0043-x.

[2] A. Ali-Gombe, A. Tambaoan, A. Gurfolino y G. G. Richard III, «App-Agnostic Post-Execution Semantic Analysis of Android In-Memory Forensics Artifacts», en *Annual Computer Security Applications Conference*, ép. ACSAC '20, ACM, dic. de 2020, págs. 28-41. DOI: 10.1145/3427228.3427244.

[3] ArxSys, *Digital Forensics Framework*, Accedido: 2024-11-27, 2020. dirección: https://github.com/arxsys/dff.

[4] F. Block, «Windows memory forensics: Identification of (malicious) modifications in memory-mapped image files», *Forensic Science International: Digital Investigation*, vol. 45, pág. 301 561, jul. de 2023, ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2023.301561.

[5] F. Block y A. Dewald, «Windows Memory Forensics: Detecting (Un)Intentionally Hidden Injected Code by Examining Page Table Entries», *Digital Investigation*, vol. 29, S3-S12, jul. de 2019, ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.04.008.

[6] M. Botacin, F. Ceschin, R. Sun, D. Oliveira y A. Grégio, «Challenges and pitfalls in malware research», *Computers & Security*, vol. 106, pág. 102 287, 2021, ISSN: 0167-4048. DOI: 10.1016/j.cose.2021.102287.

[7] C. J. Bowen, A. Case, I. Baggili y G. G. Richard, «A step in a new direction: NVIDIA GPU kernel driver memory forensics», *Forensic Science International: Digital Investigation*, vol. 49, pág. 301 760, jul. de 2024, ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2024.301760.

[8] A. Carrera-Rivera, W. Ochoa, F. Larrinaga y G. Lasa, «How-to conduct a systematic literature review: A quick guide for computer science research», *MethodsX*, vol. 9, pág. 101 895, 2022. DOI: 10.1016/j.mex.2022.101895.

[9] A. Case, A. Cristina, L. Marziale, G. G. Richard y V. Roussev, «FACE: Automated digital evidence discovery and correlation», *Digital Investigation*, vol. 5, S65-S75, sep. de 2008, ISSN: 1742-2876. DOI: 10.1016/j.diin.2008.05.008.

[10] A. Case, M. M. Jalalzai, M. Firoz-Ul-Amin, R. D. Maggio, A. Ali-Gombe, M. Sun y G. G. Richard, «HookTracer: A System for Automated and Accessible API Hooks Analysis», *Digital Investigation*, vol. 29, S104-S112, jul. de 2019, ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.04.011.

[11] A. Case, R. D. Maggio, M. Firoz-Ul-Amin, M. M. Jalalzai, A. Ali-Gombe, M. Sun y G. G. Richard, «Hooktracer: Automatic Detection and Analysis of Keystroke Loggers Using Memory Forensics», *Computers and Security*, vol. 96, pág. 101 872, sep. de 2020, ISSN: 0167-4048. DOI: 10.1016/j.cose.2020.101872.

[12] A. Case y G. G. Richard, «Advancing Mac OS X rootkit detection», *Digital Investigation*, vol. 14, S25-S33, ago. de 2015, ISSN: 1742-2876. DOI: 10.1016/j.diin.2015.05.005.

[13] A. Case y G. G. Richard, «Detecting objective-C malware through memory forensics», *Digital Investigation*, vol. 18, S3-S10, ago. de 2016, ISSN: 1742-2876. DOI: 10.1016/j.diin.2016.04.017.

[14] P. Casey, R. Lindsay-Decusati, I. Baggili y F. Breitinger, «Inception: Virtual Space in Memory Space in Real Space – Memory Forensics of Immersive Virtual Reality with the HTC Vive», *Digital Investigation*, vol. 29, S13-S21, jul. de 2019, ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.04.007.

[15] O. Chechik e I. Weber, *check_parent_spoof: A Volatility Plugin for Detecting Parent Process Spoofing*, Accedido: 2024-11-28, 2020. dirección: https://github.com/orchechik/check_spoof.

[16] P. Cichonski, T. Millar, T. Grance y K. Scarfone, «Computer Security Incident Handling Guide», National Institute of Standards y Technology, Special Publication 800-61 Rev 2, 2012. DOI: 10.6028/NIST.SP.800-61r2.

[17] K. Cucci, *Evasive Malware: A Field Guide to Detecting, Analyzing, and Defeating Advanced Threats*. No Starch Press, 2024, ISBN: 978-1-7185-0326-7. dirección: https://www.penguinrandomhouse.com/books/735726/evasive-malware-by-kyle-cucci/.

[18] S. Fabbri, C. Silva, E. Hernandes, F. Octaviano, A. Di Thommazo y A. Belgamo, «Improvements in the StArt tool to better support the systematic review process», en *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, ép. EASE '16, ACM, jun. de 2016. DOI: 10.1145/2915970.2916013.

[19] Q. Feng, A. Prakash, H. Yin y Z. Lin, «MACE: high-coverage and robust memory analysis for commodity operating systems», en *Proceedings of the 30th Annual Computer Security Applications Conference*, ép. ACSAC '14, ACM, dic. de 2014. DOI: 10.1145/2664243.2664248.

[20] P. Fernández-Álvarez y R. J. Rodríguez, «Module extraction and DLL hijacking detection via single or multiple memory dumps», *Forensic Science International: Digital Investigation*, vol. 44, pág. 301 505, mar. de 2023, ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2023.301505.

[21] F. Franzen, T. Holl, M. Andreas, J. Kirsch y J. Grossklags, «Katana: Robust, Automated, Binary-Only Forensic Analysis of Linux Memory Snapshots», en *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*, ép. RAID 2022, ACM, oct. de 2022. DOI: 10.1145/3545948.3545980.

[22] T. Gharaibeh, S. Seiden, M. Abouelsaoud, E. Bou-Harb e I. Baggili, «Don't, Stop, Drop, Pause: Forensics of CONtainer CheckPOINTs (ConPoint)», en *Proceedings of the 19th International Conference on Availability, Reliability and Security*, ép. ARES 2024, ACM, jul. de 2024, págs. 1-11. DOI: 10.1145/3664476.3670895.

[23] Gobierno de España, *Real Decreto 311/2022, de 3 de mayo, por el que se regula el Esquema Nacional de Seguridad.* [Online; https://www.boe.es/buscar/act.php?id=BOE-A-2022-7191], Accedido: 2024-11-28, mayo de 2022.

[24] C. Greamo y A. Ghosh, «Sandboxing and Virtualization: Modern Tools for Combating Malware», *IEEE Security and Privacy*, vol. 9, n.º 2, págs. 79-82, 2011, ISSN: 1540-7993. DOI: 10.1109/MSP.2011.36.

[25] H. Hachicha, *VolDiff: Malware Memory Footprint Analysis*, Accedido: 2024-11-28, 2015. dirección: https://github.com/H2Cyber/VolDiff.

[26] 504ensics Labs, *DAMM: Differential Analysis of Malware in Memory*, Accedido: 2024-11-28, 2023. dirección: https://github.com/504ensicsLabs/DAMM.

[27] A. H. Lashkari, B. Li, T. L. Carrier y G. Kaur, «VolMemLyzer: Volatile Memory Analyzer for Malware Classification using Feature Engineering», en *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, IEEE, mayo de 2021, págs. 1-8. DOI: 10.1109/rdaaps48126.2021.9452028.

[28] M. H. Ligh, A. Case, J. Levy y A. Walter, *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. John Wiley & Sons, Inc., jul. de 2014, ISBN: 978-1-118-82509-9.

[29] J. Liu, Y. Feng, X. Liu, J. Zhao y Q. Liu, «MRm-DLDet: a memory-resident malware detection framework based on memory forensics and deep neural network», *Cybersecurity*, vol. 6, n.º 1, ago. de 2023, ISSN: 2523-3246. DOI: 10.1186/s42400-023-00157-w.

[30] MANDIANT, *MANDIANT Memoryze™ User Guide*, Accedido: 2024-11-27, 2012. dirección: https://fireeye.market/assets/apps/211368/documents/701164_en.pdf.

[31] Malwarebytes, *ThreatDown 2024 State of Malware*, Accedido: 2024-11-28, 2024. dirección: https://www.threatdown.com/wp-content/uploads/2024/08/ThreatDown-State-of-Malware-2024.pdf.

[32] P. Maniriho, A. N. Mahmood y M. J. M. Chowdhury, «MeMalDet: A memory analysis-based malware detection framework using deep autoencoders and stacked ensemble under temporal evaluations», *Computers and Security*, vol. 142, pág. 103 864, jul. de 2024, ISSN: 0167-4048. DOI: 10.1016/j.cose.2024.103864.

[33] M. Manna, A. Case, A. Ali-Gombe y G. G. Richard, «Memory analysis of .NET and .Net Core applications», *Forensic Science International: Digital Investigation*, vol. 42, pág. 301 404, jul. de 2022, ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2022.301404.

[34] C. Meyers, A. R. Ikuesan y H. S. Venter, «Automated RAM analysis mechanism for windows operating system for digital investigation», en *2017 IEEE Conference on Application, Information and Network Security (AINS)*, IEEE, nov. de 2017, págs. 85-90. DOI: 10.1109/ains.2017.8270430.

[35] D. Moher, A. Liberati, J. Tetzlaff, D. G. Altman y T. P. Group, «Preferred Reporting Items for Systematic Reviews and Meta-Analyses: The PRISMA Statement», *PLoS Medicine*, vol. 6, n.º 7, e1000097, 2009. DOI: 10.1371/journal.pmed.1000097.

[36] M. Moniente, *Automated IOC Extractor*, Accessed: 2024-11-28, 2024. dirección: https://github.com/iciouss/automated-ioc-extractor.

[37] M. Murthaja, B. Sahayanathan, A. Munasinghe, D. Uthayakumar, L. Rupasinghe y A. Senarathne, «An Automated Tool for Memory Forensics», en *2019 International Conference on Advancements in Computing (ICAC)*, IEEE, dic. de 2019, págs. 1-6. DOI: 10.1109/icac49085.2019.9103416.

[38] S. J. Nair y S. R. Syam, «Automated Malware Detection Using Memory Forensics», en *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, IEEE, jun. de 2024, págs. 1-5. DOI: 10.1109/icccnt61001.2024.10725096.

[39] K. O'Reilly y A. Brukhovetskyy, *CAPEv2: Malware Configuration And Payload Extraction*, ver. 2, Accedido: 2024-11-28, 2020. dirección: https://github.com/kevoreilly/CAPEv2.

[40] J. Okolica y G. Peterson, «A Compiled Memory Analysis Tool», en *Advances in Digital Forensics VI*. Springer Berlin Heidelberg, 2010, págs. 195-204, ISBN: 9783642155062. DOI: 10.1007/978-3-642-15506-2_14.

[41] F. Pagani y D. Balzarotti, «AutoProfile: Towards Automated Profile Generation for Memory Analysis», *ACM Transactions on Privacy and Security*, vol. 25, n.º 1, págs. 1-26, nov. de 2021, ISSN: 2471-2574. DOI: 10.1145/3485471.

[42] N. L. Petroni, A. Walters, T. Fraser y W. A. Arbaugh, «FATKit: A framework for the extraction and analysis of digital forensic data from volatile system memory», *Digital Investigation*, vol. 3, n.º 4, págs. 197-210, dic. de 2006, ISSN: 1742-2876. DOI: 10.1016/j.diin.2006.10.001.

[43] G. Pék, Z. Lázár, Z. Várnagy, M. Félegyházi y L. Buttyán, «Membrane: A Posteriori Detection of Malicious Code Loading by Memory Paging Analysis», en *Computer Security – ESORICS 2016*. Springer International Publishing, 2016, págs. 199-216, ISBN: 9783319457444. DOI: 10.1007/978-3-319-45744-4_10.

[44] V. Ravindra Sali y H. Khanuja, «RAM Forensics: The Analysis and Extraction of Malicious Processes from Memory Image Using GUI Based Memory Forensic Toolkit», en *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, IEEE, ago. de 2018, págs. 1-6. DOI: 10.1109/iccubea.2018.8697752.

[45] Rekall, *The Rekall memory forensic framework*, Accedido: 2024-11-28, 2014. dirección: https://github.com/google/rekall.

[46] M. Research, *Project Freta*, Accedido: 2024-11-27, 2020. dirección: https://www.microsoft.com/en-us/research/project/project-freta/.

[47] M. Sikorski y A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, feb. de 2012, ISBN: 9781593272906.

[48] A. Srivastava y J. H. Jones, «Detecting code injection by cross-validating stack and VAD information in windows physical memory», en *2017 IEEE Conference on Open Systems (ICOS)*, IEEE, nov. de 2017, págs. 83-89. DOI: 10.1109/icos.2017.8280279.

[49] T. Thomas, M. Piscitelli, B. A. Nahar e I. Baggili, «Duck Hunt: Memory forensics of USB attack platforms», *Forensic Science International: Digital Investigation*, vol. 37, pág. 301 190, jul. de 2021, ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2021.301190.

[50] C.-W. Tien, J.-W. Liao, S.-C. Chang y S.-Y. Kuo, «Memory forensics using virtual machine introspection for Malware analysis», en *2017 IEEE Conference on Dependable and Secure Computing*, 2017, págs. 518-519. DOI: 10.1109/DESEC.2017.8073871.

[51] D. Uroz y R. J. Rodríguez, «Characteristics and detectability of Windows autostart extensibility points in memory forensics», *Digital Investigation*, vol. 28, S95-S104, abr. de 2019, ISSN: 1742-2876. DOI: 10.1016/j.diin.2019.01.026.

[52] D. Uroz y R. J. Rodríguez, «On Challenges in Verifying Trusted Executable Files in Memory Forensics», *Forensic Science International: Digital Investigation*, vol. 32, pág. 300 917, abr. de 2020, ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2020.300917.

[53] Volatility Foundation, *Volatility 3*, Accedido: 2024-11-28, 2022. dirección: https://github.com/volatilityfoundation/volatility3/.

[54] S. Vomel y H. Lenz, «Visualizing Indicators of Rootkit Infections in Memory Forensics», en *2013 Seventh International Conference on IT Security Incident Management and IT Forensics*, IEEE, mar. de 2013, págs. 122-139. DOI: 10.1109/imf.2013.12.

[55] E. Wang, S. Zurowski, O. Duffy, T. Thomas e I. Baggili, «Juicing V8: A primary account for the memory forensics of the V8 JavaScript engine», *Forensic Science International: Digital Investigation*, vol. 42, pág. 301 400, jul. de 2022, ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2022.301400.

[56] A. Zamboni, A. Thommazo, E. C. M. Hernandes y S. Fabbri, «StArt uma ferramenta computacional de apoio à revisão sistemática», en *Proc.: Congresso Brasileiro de Software (CBSoft'10), Salvador, Brazil*, UFBA, 2010, págs. 91-96.

[57] abuse.ch, *MalwareBazaar: Malware Sample Exchange*, Accedido: 2024-11-28, 2020. dirección: https://bazaar.abuse.ch/.

A APÉNDICE

En este apéndice se exponen las horas dedicadas a cada tarea del proyecto, así como su distribución en el tiempo mediante un diagrama de Gantt.

A.1 TABLA DE HORAS DEDICADAS

TAREA	HORAS
1. Preparación de la revisión sistemática (preguntas, palabras clave y criterios)	15
2. Realización de la revisión sistemática (búsqueda, cribado y lectura de artículos)	110
3. Estudio, recopilación y preparación de las herramientas	70
4. Preparación del entorno de pruebas	35
5. Desarrollo e implementación del sistema	60
6. Pruebas y validación del sistema	20
7. Documentación y redacción de memoria	85
8. Reuniones con los directores	15
Total:	410

Tabla 5: Horas dedicadas al proyecto

A.2 DIAGRAMA DE GANTT DEL PROYECTO

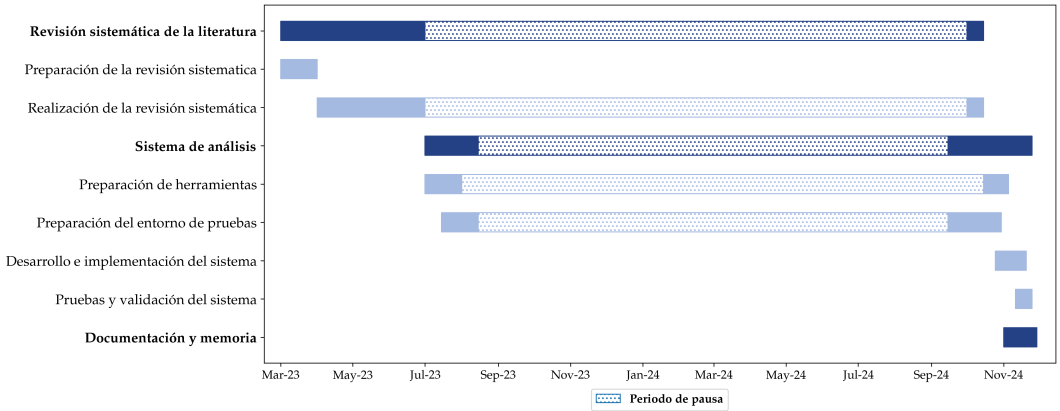

Figura 10: Diagrama de Gantt