

Ofuscación de malware

Descripción de las herramientas necesarias para el análisis de malware

PEBear

Es una herramienta para analizar de forma estática ejecutables para Windows. Tiene una ventaja sobre los otros programas similares que he probado, y es que es mas visual que el resto. Además, permite editar directamente el archivo. Esto es importante porque, hay veces, que tenemos que arreglar la tabla PE para poder ejecutar el sample correctamente.

X64dbg

Como descrito en la practica anterior, es un debugger de ejecutables. Con este debugger podemos hacer análisis estático y análisis dinámico. Además, se puede editar el archivo, para añadir comentarios, para poder hacer que la practica de reversing sea mas sencilla. Estos cambios, no se guardan directamente en el binario, sino que se guardan en una base de datos, que al detectar el binario, añade los comentarios, puntos de ruptura y trabajo que hayas realizado con el ejecutable. Tiene una ventaja sobre IDA, por ejemplo, y es que cuenta con la herramienta Scylla de fabrica.

Scylla

Prueba 1

Análisis estático de la prueba de malware descargada

Abrimos el malware descargado con PEBear para ver que efectivamente, se trata de un programa para Windows. Este ejecutable, como la mayoría de malwares, esta compilado para una arquitectura de 32 bits. De esta forma, tiene mas posibilidades de correr en distintas maquinas.

Análisis de malware desde el punto de vista de ofuscación

Análisis desde el punto de vista del empaquetado de ejecutables

En esta sección se pretende desempaquetar el malware seleccionado para posteriormente hacer un volcado de memoria virtual a disco y averiguar las cabeceras. De esta forma, podemos analizar el malware como se ha creado desde el inicio, sin los inconvenientes introducidos por los packers.

De primeras, vemos que el ejecutable que estamos analizando no esta packed con ningún malware. Sabemos que el malware es autocontenido porque el perfil del mismo en VirusShare, vemos que no crea ningún hilo de ejecución. Esto quiere

decir que tenemos que estar atentos a las funciones: `Kernel32::VirtualAlloc()` y `Kernel32::VirtualProtect()`.

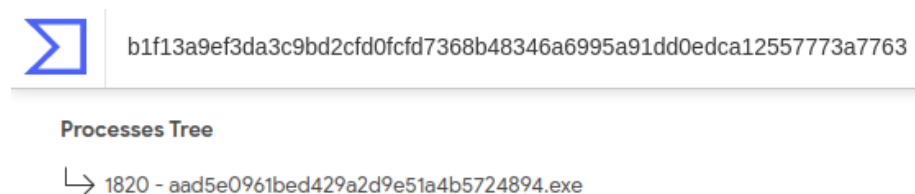


Figure 1: Process Tree del malware sample

Efectivamente, siguiendo la sección de memoria que se copula, vemos que se inserta un archivo PE entero. Volcando el contenido de memoria a disco, vemos que el malware esta packed. Además, podemos determinar que las cabeceras están sin capear, que es como las espera Windows, por tanto, podemos directamente ir a desempaquetar el malware. La herramienta que se ha usado para el empaquetado del mismo es UPX, una herramienta muy conocida en el mundo de los packers.

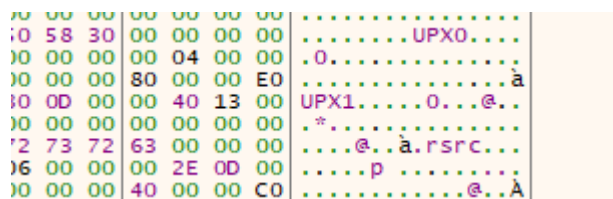


Figure 2: Malware Packed volcado en memoria

Desempaquetado automático

Para hacer el unpack, vamos a usar la misma herramienta que se usa para packear, ya que si se le pasa la opción `-d`, el packer te unpackea cualquier archivo que ha packeado el programa.

```
C:\Users\windows1\Desktop\Malware\Shade-Ransom>upx -d Shade_02250000.bin
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2020
UPX 3.96w Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020

File size      Ratio      Format      Name
-----
2079232 <- 868352 41.76% win32/pe Shade_02250000.bin

Unpacked 1 file.
```

Figure 3: UPX -d desempaquetado

En este momento, tenemos el malware unpacked y ya podemos pasar a analizarlo.

Desempaquetado manual

Se ha decidido usar X64dbg para hacer el desempaquetado manual. Se inicia el procedimiento, con los programas de análisis dinámico iniciados, para monitoreos el proceso.

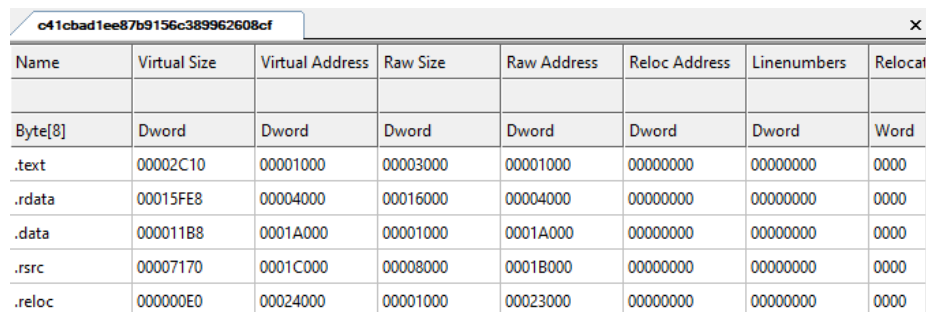
El procedimiento es el siguiente:

1. Se descomprime el virus a analizar
2. Se abre ProcessHacker, para poder tener un rastro del proceso
3. Se abre Noriben, para que nos de un resumen de los procesos que hemos mirado con process monitor
4. Se inicia x64dbg con privilegios de administrador
 1. En este momento, esta el virus cargado en memoria
 2. Tenemos que llegar al momento en el que se acaba de descomprimir el malware y se le cede el control
5. Seguimos ejecutando el malware hasta que los avisa de que estamos cediendo el control a una dirección desconocida
6. La siguiente instrucción es el OEP del malware.

Prueba 2

Análisis Estático

Con el análisis de las cabeceras, podemos ver que es un ejecutable, porque tiene la cabecera mágica. Además, observando el tamaño del RAW Size de la sección .text y el tamaño del virtual size, vemos que existe una diferencia importante.



Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocat
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word
.text	00002C10	00001000	00003000	00001000	00000000	00000000	0000
.rdata	00015FE8	00004000	00016000	00004000	00000000	00000000	0000
.data	000011B8	0001A000	00001000	0001A000	00000000	00000000	0000
.rsrc	00007170	0001C000	00008000	0001B000	00000000	00000000	0000
.reloc	000000E0	00024000	00001000	00023000	00000000	00000000	0000

Figure 4: Cabeceras

Análisis Dinámico

Al ver que es una muestra que esta packed, podemos ejecutar la muestra con nuestro debugger hasta que se llame al proceso `VirtualAlloc` del API `Kernel32.dll`.

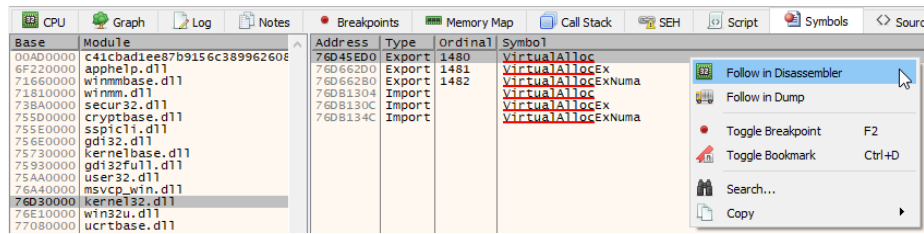


Figure 5: VirtualAlloc Breakpoint

Hacemos esto porque sabemos que el virus es autocontenido y que no crea un nuevo proceso desde si mismo. Esto quiere decir, que en algún momento, se debe reservar memoria para el virus.

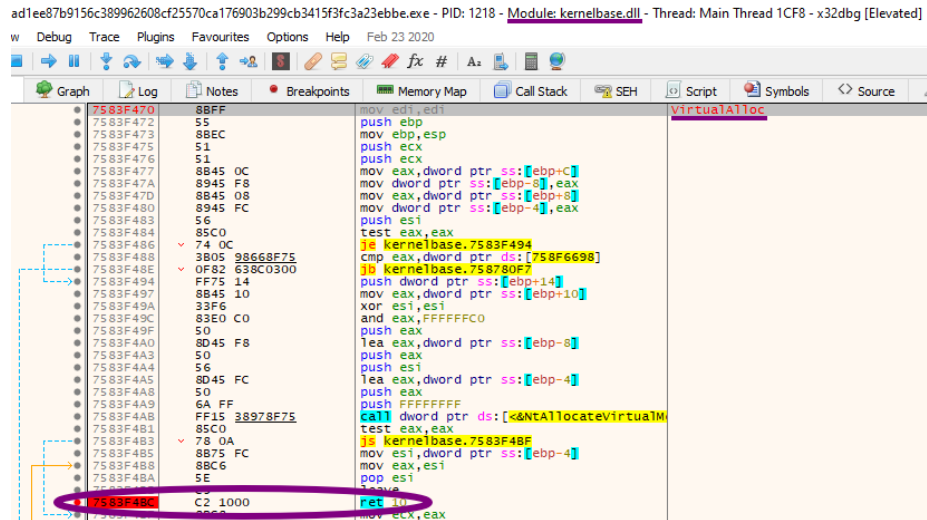


Figure 6: VirtualAlloc return de KernelBase

En este momento, tenemos los siguientes Breakpoints:

Type	Address	Module/Label/Exception	State	Disassembly	Hits	Summary
Software	00AD16F0	<c41cbad1ee87b9156c389962608cf25570ca176903b299cb3415f3fc3a23ebbe.exe	One-time	xchg ebp,eax	0	entry breakpoint
	7583F48C	kernelbase.dll	Enabled	ret 10	0	

Figure 7: Breakpoints

Según la documentación de Microsoft, podemos saber que este método devuelve el puntero a memoria en el registro EAX, entonces, basta con seguir el registro en el Dump para obtener el programa Unpacked. Tras dos iteraciones, se ha generado un archivo PE en una sección de memoria.

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x]= Loc
Address	Hex				ASCII	
00AB0000	4D 5A 90 00	03 00 00 00	04 00 00 00	FF FF 00 00	MZ.....	yy
00AB0010	B8 00 00 00	00 00 00 00	40 00 00 00	00 00 00 00@...	
00AB0020	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
00AB0030	00 00 00 00	00 00 00 00	00 00 00 00	D0 00 00 00D...	
00AB0040	0E 1F BA 0E	00 B4 09 CD	21 B8 01 4C	CD 21 54 68	...I...I!Th	
00AB0050	69 73 20 70	72 6F 67 72	61 6D 20 63	61 6E 6E 6F	is program canno	
00AB0060	74 20 62 65	20 72 75 6E	20 69 6E 20	44 4F 53 20	t be run in DOS	
00AB0070	6D 6F 64 65	2E 0D 0D 0A	24 00 00 00	00 00 00 00	mode...\$...	
00AB0080	78 9B 95 E3	3C FA FB 80	3C FA FB 80	3C FA FB 80	x...a...u...u...	
00AB0090	3C FA FA 80	20 FA FB 80	35 82 68 80	35 FA FB 80	<u...u...5...h...u...	
00AB00A0	3C FA FB 80	3D FA FB 80	31 A8 25 80	3D FA FB 80	<u...=u...1...%...=u...	
00AB00B0	52 69 63 68	3C FA FB 80	00 00 00 00	00 00 00 00	Rich<u...u...	
00AB00C0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
00AB00D0	50 45 00 00	4C 01 04 00	5A 88 5F 5A	00 00 00 00	PE...L...Z..._...	
00AB00E0	00 00 00 00	E0 00 02 01	08 01 0C 00	B0 29 00 00	...a...(...)	
00AB00F0	80 2A 00 00	00 00 00 00	02 39 00 00	00 10 00 00	*...9...	
00AB0100	00 40 00 00	00 00 40 00	00 10 00 00	10 00 00 00	@...@...	
00AB0110	05 00 00 00	00 00 00 00	05 00 00 00	00 00 00 00	
00AB0120	00 E0 00 00	70 02 00 00	00 00 00 00	03 00 40 81	...a...p...@...	

Figure 8: PE generado

En este momento, sabemos que se ha guardado el archivo unpacked en memoria. Para llegar a el, tenemos que observar las direcciones de memoria que tienen el bit de ejecutable habilitado. En este sample en concreto, tenemos dos. Siguiendo la primera en el dump, podemos ver que esta en esta dirección nuestro PE. Para extraer el sample unpacked, tenemos que volcar el contenido de la memoria en disco, como en la siguiente imagen:

The screenshot shows the 'Memory Map' window in Immunity Debugger. The table lists memory regions with columns for Address, Size, Info, Content, Type, Protection, and Initial. The region 00A00000-00A00000 is highlighted in blue and labeled 'Reserved (00A0A0000)'. A red circle highlights the 'Dump Memory to File' option in the context menu. A red arrow points to the 'Dump Memory to File' option.

Figure 9: PE encontrado en memoria y volcando

Una vez tenemos el programa ejecutable en disco, lo volvemos a analizar con un editor PE.

Al abrirlo y analizar la sección `.text` que contiene el código ejecutable, vemos que hace referencia a una sección vacía. Esto es debido a que las cabeceras PE están cambiadas para apuntar a las direcciones de las secciones en memoria, no en disco. Debemos arreglar esto antes de poder abrir el PE con un debugger. Para hacer esto, lo primero que tenemos que hacer es cambiar las direcciones referenciadas en `RawAddress` por las referenciadas en `VirtualAddress`, de forma que ambos campos tengan los mismos valores.

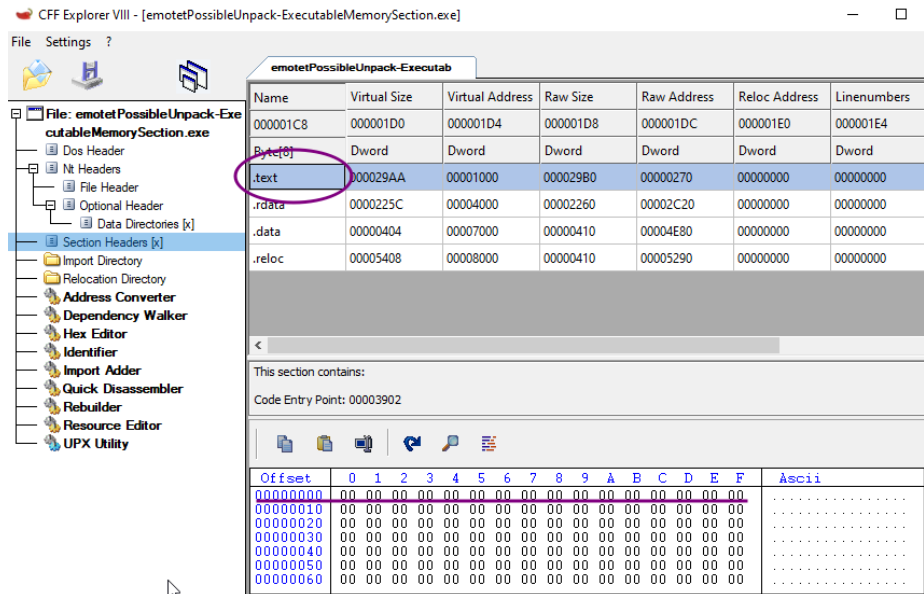
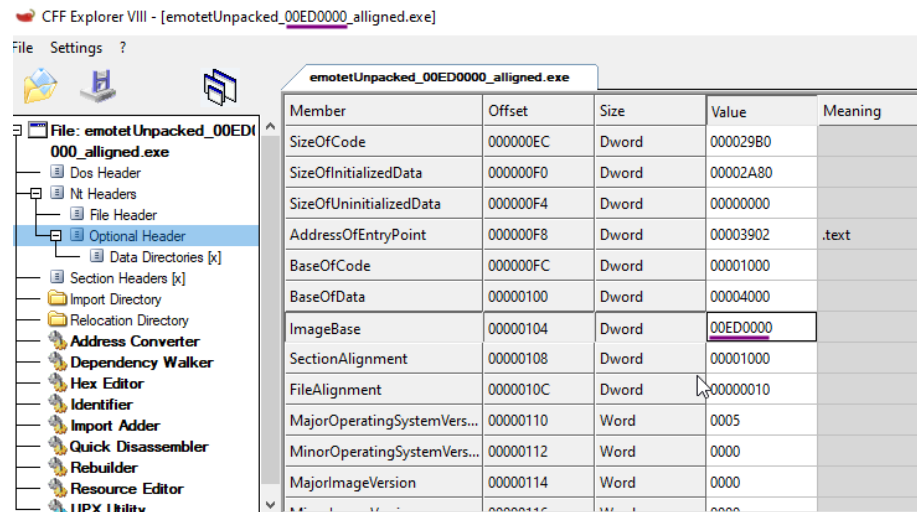


Figure 10: Cabeceras del binario mapeadas mal

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocat
000001C8	000001D0	000001D4	000001D8	000001DC	000001E0	000001E4	000001
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word
.text	000029AA	00001000	000029B0	00001000	00000000	00000000	0000
.rdata	0000225C	00004000	00002260	00004000	00000000	00000000	0000
.data	00000404	00007000	00000410	00007000	00000000	00000000	0000
.reloc	00005408	00008000	00000410	00008000	00000000	00000000	0000

Figure 11: Cambio de valores de la tabla de cabeceras

Lo segundo que tenemos que hacer es cambiar la dirección base de la tabla de cabeceras opcionales para que coincida con la dirección de entrada del programa.



Ofuscación de malware

Descripción de las herramientas necesarias para el análisis de malware

PEBear