Tutorial pour le PEanalysis de Kharneth

Outil nécessaire:

Hex WorkShop ou n'importe quel éditeur héxa
ShowPE ou PEview
Papier, crayon, cerveau

Niveau: débutant ayant des bases sur le PE

Let's go !

Bon, bien que je vais rappeler quelques détails, mais je vais supposer que vous avez lu quelques tutoriaux sur le sujet. Commençons donc par ouvrir le fichier dans notre éditeur hexa. La chose qu'on remarque avant tout est le manque du **IMAGE_DOS_HEADER.e_magic** c'est a dire le fameux MZ. La première modification sera donc la suivante:

1. offset 0000 <- 4D5A (= 'MZ')

Etant donné que ce membre a été enlevé, il y a fort à parier que le jump vers le PE header a été enlevé aussi. On se rend donc en 3C, place du **IMAGE_DOS_HEADER.e_lfanew** et on trouve bien un champ vide. On repère donc ou commence le PE header et on complète ce champs. Ce sera la deuxième modification:

2. offset 003C <- C000

Bien ces deux champs complèté, on peux donc utiliser au choix le programme d'une petite demoiselle, ou PEview. Je vais donc supporter la demoiselle qui débute dans le milieu (i.e: elooo et son ShowPE) :p. Donc concentrons nous d'abors sur le PE header, nous verrons après pour l'optional header. Le log me donne donc ceci:

```
*****************************************************************
* ShowPE 1.2 *
* coded in asm 32 bits *
* by elooo *
*****************************************************************

-----> Log for : PEanalysis.exe
_____________________________________________________________________

_____________________________________________________________________

IMAGE_FILE_HEADER
_____________________________________________________________________
0XC4 WORD Machine : 0X014C
-----> Build for Intel 80386 processor
0XC6 WORD NumberOfSections : 0X0000
0XC8 DWORD TimeDateStamp : 0X425647A9
0XCC DWORD PointerToSymbolTable : 0X00000000
0XD0 DWORD NumberOfSymbols : 0X00000000
0XD4 WORD SizeOfOptionalHeader : 0X0000
```

```
0XD6 WORD Characteristics : 0X0000
-----> Non-32-bit-word machine
-----> Big endian
-----> Contains base relocations
-----> COFF line numbers haven't been removed
-----> COFF symbol table entries for local symbols haven't been removed
-----> Debugging information not removed from image file
-----> Removable media : not needed to copy and run from swap file.
-----> Network : not needed to copy and run from swap file.
-----> Runs on MP or UP machine
-----> Normally trim working set
-----> Object/library file
-----> Not a system file
-----> File is not a DLL
```

On vois tout de suite que le NumberOfSections (le nom est assez explicite) est à 0 alors que si on compte les sections, il y en a 4. Il faut donc corriger ca!!

3. offset 00C6 <- 0400

Le PointerToSymbolTable et NumberOfSymbols ne sont pas important, on ne va donc pas s'en occuper. Par contre, la SizeOfOptionalHeader est important puisqu'il donne l'information sur la taille du Optional Header. Pour déterminer cette taille, il suffit de prendre la taille entre le début de l'Optional Header (il commence juste après le Characteristics du PE header, soit en 0xD8), et l'offset juste avant le section .text. Soit de D8 à 1B8, ce qui nous donne comme SizeOfOptionalHeader: 1B8 - B8 = E0.

4. offset 00D4 <- E000

Viens ensuite le Characteristics. Pour cette valeur, je n'ai pas d'explication précise. On m'a d'ailleurs conseillé de regarder les valeurs dans d'autres .exe. La 5ème modification est donc celle ci:

5. offset 00D6 <- 0F01

C'est donc fini pour le PE header. Avant de nous attaquer à l'Optional Header, je voudrais d'abors faire un tour du côté de L'IMAGE_SECTION_HEADER, (i.e: vers les sections). On réutilise donc le gentil petit programme de la gentille petite elooo:

```
SECTION
0X1B8 Name : .text
0X1C0 DWORD VirtualSize : 0X00000026
0X1C4 DWORD VirtualAddress : 0X00000000
0X1C8 DWORD SizeOfRawData : 0X00000000
0X1CC DWORD PointerToRawData : 0X00000000
0X1D0 DWORD PointerToRelocations :
0X00000000
0X1D4 DWORD PointerToLinenumbers :
0X00000000
0X1D8 WORD NumberOfRelocations : 0X0000
0X1DA WORD NumberOfLinenumbers : 0X0000
0X1DC DWORD Characteristics : 0X60000020
-----> Contains executable code
-----> Executable as code
-----> Readable
```

```
SECTION
0X1E0 Name : .rdata
0X1E8 DWORD VirtualSize : 0X00000092
0X1EC DWORD VirtualAddress : 0X00002000
0X1F0 DWORD SizeOfRawData : 0X00000200
0X1F4 DWORD PointerToRawData :
0X00000600
0X1F8 DWORD PointerToRelocations :
0X00000000
0X1FC DWORD PointerToLinenumbers :
0X00000000
0X200 WORD NumberOfRelocations : 0X0000
0X202 WORD NumberOfLinenumbers : 0X0000
0X204 DWORD Characteristics : 0X40000040
-----> Contains initialized data
-----> Readable
```

SECTION 0X208 Name : .data 0X210 DWORD VirtualSize : 0X00000040 0X214 DWORD VirtualAddress : 0X00003000 0X218 DWORD SizeOfRawData : 0X00000200 0X21C DWORD PointerToRawData : 0X00000800 0X220 DWORD PointerToRelocations : 0X00000000 0X224 DWORD PointerToLinenumbers : 0X00000000 0X228 WORD NumberOfRelocations : 0X0000 0X22A WORD NumberOfLinenumbers : 0X0000 0X22C DWORD Characteristics : 0XC0000040 -----> Contains initialized data -----> Readable -----> Writeable	SECTION 0X230 Name : .rsrc 0X238 DWORD VirtualSize : 0X000000B8 0X23C DWORD VirtualAddress : 0X00004000 0X240 DWORD SizeOfRawData : 0X00000200 0X244 DWORD PointerToRawData : 0X00000A00 0X248 DWORD PointerToRelocations : 0X00000000 0X24C DWORD PointerToLinenumbers : 0X00000000 0X250 WORD NumberOfRelocations : 0X0000 0X252 WORD NumberOfLinenumbers : 0X0000 0X254 DWORD Characteristics : 0X40000040 -----> Contains initialized data -----> Readable

Ici la chose que l'on observe est l'absence de renseinement pour la section .text. Les autres sont quand à elles correctement remplies. Pour commencer à remplir les informations de cette section, on peut regarder vite fais les champs de l'Optional Header, pour voir ou se situe le BaseOfCode. C'est en effet le point de départ de la section .text quand le programme est en mémoire. On voit que la valeur de BaseOfCode est 0x1000, et au passage, on voit que SizeOfCode vaut quand a lui 0x200. SizeOfCode correspond à la Raw Size. Pouc connaitre le Raw Offset de notre section, c'est a dire le début de la section quand justement notre programme est en dur (donc pas en mémoire, et je vous rapelles que l'éditeur hexa ne charge pas en mémoire), on regarde la section suivante. Le Raw Offset de la section .rdata est 0x600 et on sait que notre section .text a une Raw Size de 200. La section .rdata commence là ou la section .text fini ! Donc Raw Offset(.text) = Raw Offset(.rdata) - Raw Size(.text), ce qui nous donne 0x600 - 0x200 = 0x400. On a donc tout les renseignements pour notre sections .text:

6. offset 1C4 <- 0x1000

7. offset 1C8 <- 0x0002

8. offset 1CC <- 0x0004

Bien, nous avons fais la moitié du travail. Passons maintenant à l'Optional Header. Comme d'hab on sort ShowPE, et on obtient ce log:

```
IMAGE_OPTIONAL_HEADER
_____________________________________________________________

0XD8 WORD Magic : 0X010B
0XDA BYTE MajorLinkerVersion : 0X05
0XDB BYTE MinorLinkerVersion : 0X0C
-----> Linker version: 5.12
0XDC DWORD SizeOfCode : 0X00000200
-----> Size of the executable code : 512 bytes
0XE0 DWORD SizeOfInitializedData : 0X00000600
-----> Size of the data segment : 1536 bytes
0XE4 DWORD SizeOfUninitializedData : 0X00000000
0XE8 DWORD AddressOfEntryPoint : 0X00001000
-----> Execution starts at 0x00001000 (RVA) or at 0x00001000 (VA)
0XEC DWORD BaseOfCode : 0X00001000
```

```
-----> RVA to the executable code : 0x00001000
0XF0 DWORD BaseOfData : 0X00002000
-----> RVA to the initialized data : 0x00002000
0XF4 DWORD ImageBase : 0X00000000
-----> Preferred load base is 0x00000000
0XF8 DWORD SectionAlignment : 0X00001000
0XFC DWORD FileAlignment : 0X00000200
-----> Sections aligned to 4096 bytes in RAM, 512 bytes in file
0X100 WORD MajorOperatingSystemVersion : 0X0004
0X102 WORD MinorOperatingSystemVersion : 0X0000
0X104 WORD MajorImageVersion : 0X0004
0X106 WORD MinorImageVersion : 0X0000
0X108 WORD MajorSubsystemVersion : 0X0004
0X10A WORD MinorSubsystemVersion : 0X0000
-----> Versions: NT 4.0, Win32 4.0, App 4.0
0X10C DWORD Win32VersionValue : 0X00000000
0X110 DWORD SizeOfImage : 0X00000000
-----> Size of image in RAM: 0 bytes
0X114 DWORD SizeOfHeaders : 0X00000000
-----> Size of headers / offset to the 1st section in file: 0x0
0X118 DWORD CheckSum : 0X000108B1
-----> Checksum: 0x000108b1
0X11C WORD Subsystem : 0X0000
-----> Subsystem unknown !
0X11E WORD DllCharacteristics : 0X0000
0X120 DWORD SizeOfStackReserve : 0X00100000
-----> Size of reserved stack : 1048576 bytes
0X124 DWORD SizeOfStackCommit : 0X00001000
-----> Size of initially committed stack : 4096 bytes
0X128 DWORD SizeOfHeapReserve : 0X00100000
-----> Size of the reserved heap : 1048576 bytes
0X12C DWORD SizeOfHeapCommit : 0X00001000
-----> Size of the committed heap : 4096 bytes
0X130 DWORD LoaderFlags : 0X00000000
0X134 DWORD NumberOfRvaAndSizes : 0X00000000
-----> Number of valid entries in the IMAGE_DATA_DIRECTORY : 0
0X138 IMAGE_DATA_DIRECTORY
```

J'ai mis en rouge les endroits que nous allons complèter. Tout d'abors à l'offset 0xF4 se trouve l'Image Base. Celle-ci est défini lors de la compilation mais la plupart du temps, elle vaut 0x00400000.

9. offset 00F4 <- 00004000

Vient ensuite le SizeOfImage. Le SizeOfImage st déterminé en ajoutant le Virtual Offset a la Virtual Size de la dernière section. Ici notre dernière section est .rsrc, son Virtual Offset vaut 0x4000 et sa Virtual Size 0x00B8. On a donc que SizeOfImage = 0x4000 + 0x00B8 = 0x40B8.

10. offset 0110 <- B840

Le SizeOfHeader peut-être trouvé de deux manières: soit vous le trouvez en faisant Taille du fichier - Somme de la taille des sections, ou alors plus simplement, il est égal au Raw Offset de la première section. Notre permière section est .text et son Raw Offset est 0x400. Notre SizeOfHeader vaut donc 0x400 !

11. offset 0114 <- 0004

Passons au Subsystem. Cette valeur indique le type de système va tourner le programme. Je vous remet la liste des valeurs possibles (prise comme un voleur dans le M5):

Valeur	Nom	Sur quoi ca tourne
0	IMAGE_SUBSYSTEM_UNKNOWN	Subsystem inconnu.
1	IMAGE_SUBSYSTEM_NATIVE	L'image n'a pas besoin de subsystem (cas des drivers).
2	IMAGE_SUBSYSTEM_WINDOWS_GUI	L'image tourne sous un subsystem Windows GUI.
3	IMAGE_SUBSYSTEM_WINDOWS_CUI	L'image tourne sous un subsystem de type Windows (i.e. en mode console aka DOS).
5	IMAGE_SUBSYSTEM_OS2_CUI	L'image tourne sous un subsystem de type OS/2.
7	IMAGE_SUBSYSTEM_POSIX_CUI	L'image tourne sous un subsystem de type Posix.
8	IMAGE_SUBSYSTEM_NATIVE_WINDOWS	L'image est un driver "native" (d'origine?) Win9x.
9	IMAGE_SUBSYSTEM_WINDOWS_CE_GUI	L'image tourne sous un subsystem Windows CE.

Dans notre cas, on prend la valeur de IMAGE_SUBSYSTEM_WINDOWS_GUI, a savoir 2:

12. offset 011C <- 0200

Bien, pour l'Optional Header, il nous reste plus que le NumberOFRvaAndSize (=NumberOfDirectory). C'est le nombre d'entrée valide dans le DataDirectory. Si on prend PEview, et que l'on va dans IMAGE_OPTIONAL_HEADER, on pourra compte 16 entrée (en décimal), soit 0x10 (en hexadécimal). Modifions donc cette valeur:

13. offset 0134 <- 1000

On a donc fini de corriger l'Optional Header. Que peut-il nous rester a corriger ? L'IAT et l' Import Table. Un petit coup de PEView et on voit que le Virtual Offset de l' IAT est 0x2000 (soit le même que la section .rdata), et celui de l'Import Table est 0x2010. Faisons une petite pause. Ce que nous avons n'est pas le Raw Offset (donc l'adresse a laquelle on pourait allez voir sous notre éditeur hexa), mais bien le Virtual Offset. Cependant, on remarque que nos deux tableaux sont dans la section .rdata. Le Raw Offset de notre section est donc le même pour l'IAT, et quand on se baladera dans nos tableaux, les valeurs comme 0x20XX seront à transformer (dans nos tête, pour pouvoir nous y retrouver dans notre éditeur hexa) en 0x6XX. Notre IAT a l'air d'être correctement rempli. Passons donc a l'Import Table, qui, dans notre éditeur hexa, commen en 0x610 et a une taille de 3C. En regardant un peu en dessous, on voit que 2 dll sont importées, on doit donc avoir 2 IMAGE_IMPORT_DESCRIPTOR plus 1 autre vide. Chaque IMAGE_IMPORT_DESCRIPTOR s'étale sur 5 dword, donc 20d bytes, comme on en a 3, on retrouve bien la taille de 0x3C. On remarque que le deuxième tableau est rempli, mais le premier est vide. Il faut donc savoir pour quelle dll le second IMAGE_IMPORT_DESCRIPTOR a été remplis. Le DLLname se situe au 4è Dword. Il vaut 0x2086, ce qui nous donne comme Raw Offset 0x686 et c'est l'adresse de user32.dll dans notre fichier. Le deuxième IMAGE_IMPORT_DESCRIPTOR est donc utilisé pour

déclarer user32.dll. On note au passage que celui-ci est correctement remplis. Passons donc a la déclaration de Kernel32.dll. On peut d'hors et déjà remplir le champ DLLname, qui est un RVA vers le nom de la dll. Dans notre éditeur, elle se situe en 0x66A, mais ceci est le Raw Offset. Le virtual offset est 0x206A, et c'est ce que l'on va mettre :

14. offset 061C <- 6A200000

Bien, il ne nous reste plus que le FirstThunk et l'OrgininalFirstThunk. L'OriginalFirstThunk peut-être mis de coté ici, car il pointe vers une adresse qui elle même pointe vers la fonction. De plus comme le FirstThunk pointe sur l'IAT qui contient le RVA de notre fonction, on peut se permettre de ne corriger que ce dernier membre. Il nous suffit de regarder l'adresse de ExitProcess (0x65C, ce qui donne comme Virtual Offset 0x205C), et de regarder ou se situe cette adresse dans l'IAT. Elle se trouve en 0x600 (donc virtual offset = 0x2000). On peut donc complèté maintenant notre 5è Dword de notre 1er IMAGE_IMPORT_DESCRIPTOR:

15. offset 0620 <- 00200000

Sauvez donc toutes les modifications, lancez votre .exe et...

Greetz

Kharneth, qui m'apporte énormément chaques jours, sans doute plus qu'il ne le pense,

Kef pour ses explications,

Tout ceux avec qui j'ai l'honneur de discuté de temps en temps (Kaine, Deicide, Neitsa, mimas, elooo, Kyshaam, meow, yarblub, TTO ect, ect...), ainsi que les teams FFF et FRET.