

PE analysis from Kharneth

Sujet

En étudiant le Bugtraq#1, j'ai exécuté le virus par inadvertance et plusieurs de mes fichiers ont été corrompus! Je n'arrive pas à les réparer. S'il vous plait, aidez moi!!

Le fichier est un simple petit programme dont certaines structures ont été modifiées. Le but est de retrouver un exécutable valide (Une MessageBox est affichée). Il y a en tout 16 membres effacés (remplacés par des 00). Retrouvez lesquels ainsi que la valeur qu'ils doivent contenir. Un membre peut être du type BYTE, WORD, DWORD ou chaîne de caractères. Vous ne devez pas toucher aux autres valeurs. Analysez bien chaque structure du PE (PEHeader, SectionHeader, ImportTable...).

Vous pouvez utiliser les outils que vous voulez mais je ne pense pas qu'autre chose qu'un éditeur Hexa soit utile.

Les différents PE torturés pour cet exercice sont téléchargeables ici :

- Version classique - peanalysis.zip.
- Version leet - d4_1337_v3rsi0n.zip.

Etude du PE classique

Le DOS_HEADER

Définition de la structure

```
typedef struct _IMAGE_DOS_HEADER {  
    WORD e_magic;  
    WORD e_cblp;  
    WORD e_cp;  
    WORD e_crlc;  
    WORD e_cparhdr;  
    WORD e_minalloc;  
    WORD e_maxalloc;  
    WORD e_ss;  
    WORD e_sp;  
    WORD e_csum;  
    WORD e_ip;  
    WORD e_cs;  
    WORD e_lfarlc;  
    WORD e_ovno;  
    WORD e_res[4];  
    WORD e_oemid;  
    WORD e_oeminfo;  
    WORD e_res2[10];  
    LONG e_lfanew;  
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Cet entête est encore présent dans les exécutables PE pour assurer la compatibilité. Dans un environnement DOS (16bits) le système lit cet entête et exécute le DOS stub, un programme minimaliste qui indique que ce programme doit être lancé sous Win32 (la famille Windows en 32 bits)

Champ	FileOffset	remarque
e_magic	0	les exécutables DOS se reconnaissent grâce à 'MZ'
e_lfanew	03Ch	Il doit contenir l'adresse physique de l'entête spécifique au <u>PE</u> . Ce champ permet un DOS stub d'une longueur variable. Il nous faut corriger cette valeur avec la valeur de l'offset de l'entête des exécutables windows, ici c'est 0B0h.

l'entête Windows (IMAGE_NT_HEADERS)

Définition

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER OptionalHeader;
} IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;

typedef struct _IMAGE_FILE_HEADER {
    WORD Machine;
    WORD NumberOfSections;
    DWORD TimeDateStamp;
    DWORD PointerToSymbolTable;
    DWORD NumberOfSymbols;
    WORD SizeOfOptionalHeader;
    WORD Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;

#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES 16

typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD Magic;
    BYTE MajorLinkerVersion;
    BYTE MinorLinkerVersion;
    DWORD SizeOfCode;
    DWORD SizeOfInitializedData;
    DWORD SizeOfUninitializedData;
    DWORD AddressOfEntryPoint;
    DWORD BaseOfCode;
    DWORD BaseOfData;
    DWORD ImageBase;
    DWORD SectionAlignment;
    DWORD FileAlignment;
    WORD MajorOperatingSystemVersion;
    WORD MinorOperatingSystemVersion;
    WORD MajorImageVersion;
    WORD MinorImageVersion;
    WORD MajorSubsystemVersion;
    WORD MinorSubsystemVersion;
    DWORD Reserved1;
    DWORD SizeOfImage;
    DWORD SizeOfHeaders;
    DWORD CheckSum;
    WORD Subsystem;
    WORD DllCharacteristics;
    DWORD SizeOfStackReserve;
    DWORD SizeOfStackCommit;
    DWORD SizeOfHeapReserve;
    DWORD SizeOfHeapCommit;
    DWORD LoaderFlags;
    DWORD NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

Autant les champs du premier entête sont facilement manipulables, autant ceux de l'entête Windows le sont avec difficulté. Pour avoir une vision globale sur les offsets et les enregistrements des champs j'ai utilisé peview disponible à cette adresse <http://www.magma.ca/~wjr/> [<http://www.magma.ca/~wjr/>].

Excuse moi Elooo pour ne pas avoir utiliser ton programme ShowPE, cela ne m'interdit pas pour autant de ne pas de donner l'adresse où l'on peut télécharger ton bon outil : ShowPE [<http://elooo.fff.free.fr/coding.html>].

En navigant dans l'arborescence des différentes structures on regarde quels sont les champs qui ne sont pas rempli alors qu'il devraient l'être dans un PE normalement constitué.

IMAGE_FILE_HEADER

champ	FileOffset	Remarque
NumberOfSections	0C6h	On va le mettre à 4. C'est que nous donne visuellement l'exécutable dans un éditeur hexadécimale.
SizeOfOptionalHeader	0D4h	Je vais prendre la valeur par défaut de ce champ, c'est à dire 0E0h.
Characteristics	0D6h	Nous allons prendre la valeur de IMAGE_NT_OPTIONAL_HDR_MAGIC c-à-d 0×10B.

IMAGE_OPTIONAL_HEADER

Champ	FileOffset	Remarque
ImageBase	0F4h	On va mettre à 400000h car c'est une valeur classique.
SizeofImage	110h	Ce champs est la taille de l'image mappée en mémoire, ce qui nous fait 5000h (4 section de 1000h + l'entête du PE)
Sizeofheader	114h	on met 400h (⇒ début de la section code.)
Subsystem	11Ch	On met à 2 (IMAGE_SUBSYSTEM_WINDOWS_GUI, qui n'affiche pas de console).
NumberOfDirectories	134h	On met 10h (16) parce que c'est standard. :)

Les sections (IMAGE_SECTION_HEADER)

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
        DWORD PhysicalAddress;
        DWORD VirtualSize;
    } Misc;
    DWORD VirtualAddress;
    DWORD SizeOfRawData;
    DWORD PointerToRawData;
    DWORD PointerToRelocations;
    DWORD PointerToLinenumbers;
    WORD NumberOfRelocations;
    WORD NumberOfLinenumbers;
    DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

La section '.text', contenant le code exécutable et l'entrypoint, est fausse ; Nous allons corriger les valeurs manquantes.

Champ	FileOffset	Remarque
VirtualAddress	1C4h	que l'on met à 1000h
SizeOfRawData	1C8h	La taille de VirtualSize indique la taille du code mais puisque les sections sont alignées et que la taille du code est inférieure à cette alignement nous allons utiliser la valeur de SectionAligment, soit 200h
PointerToRawData	1CCh	L'entête du PE plus l'alignement fait 400h octets, on y met donc 400h

Import table et IAT

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    _ANONYMOUS_UNION union {
        DWORD Characteristics;
        DWORD OriginalFirstThunk;
    } DUMMYUNIONNAME;
    DWORD TimeDateStamp;
    DWORD ForwarderChain;
    DWORD Name;
```

```

        DWORD FirstThunk;
    } IMAGE_IMPORT_DESCRIPTOR, *PIMAGE_IMPORT_DESCRIPTOR;

```

Tout d'abord il faut comprendre le fonctionnement des imports et la résolution des adresses. Lorsque le loader charge le programme il va résoudre les fonctions importées et placer leur adresse à un endroit spécifique (IAT).

- Le champ **OriginalFirstThunk** contient l'adresse RVA d'une table dont les éléments sont des pointeur (en RVA, toujours) vers le noms des fonctions importés. cette table se termine par un 0. le remplissage de ce champs est optionel.
- Le champ **TimeDateStamp** et **ForwarderChain** sont peu importants ici.
- Les champs **Name** contient le pointeur RVA vers le nom de la DLL associé aux imports de la table pointée par **OriginalFirstThunk**.
- le Champ **FirstThunk** est un pointeur RVA vers la table similaire à celle pointée par **OriginalFirstThunk** sauf que les valeur de cette table seront remplacées par l'adresse VA des fonctions importées.

IMAGE_IMPORT_DESCRIPTOR est aussi un élément d'une table qui se termine par un élément *IMAGE_IMPORT_DESCRIPTOR* complètement à 0.

Le PE de Kharneth contient un élément *IMAGE_IMPORT_DESCRIPTOR* complètement à 0 ce qui fait qu'il ne peut pas y avoir de résolution des imports. L'adresse RVA précisant le début de cette table est contenue dans les **DataDirectory** de la structure *IMAGE_OPTIONAL_HEADER* dans le deuxième élément (ici 2010 RVA). Il faut donc trouver les différents tables utilisées pour résoudre les adresse des imports afin de remplir les bon champs.

```

00002000  5C 20 00 00 00 00 00 00 78 20 00 00 00 00 00 00  \ .....x .....
00002010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00402020  00 00 00 00 54 20 00 00 00 00 00 00 00 00 00 00  ....T .....
00002030  86 20 00 00 08 20 00 00 00 00 00 00 00 00 00 00  .....
00002040  00 00 00 00 00 00 00 00 00 00 00 00 00 5C 20 00 00  ..... \ ..
00002050  00 00 00 00 78 20 00 00 00 00 00 00 80 00 45 78  ....x .....€.Ex
00002060  69 74 50 72 6F 63 65 73 73 00 6B 65 72 6E 65 6C  itProcess.kernel
00002070  33 32 2E 64 6C 6C 00 00 9D 01 4D 65 73 73 61 67  32.dll....Messag
00002080  65 42 6F 78 41 00 75 73 65 72 33 32 2E 64 6C 6C  eBoxA.user32.dll

```

En repérant les nom des DLL et des fonctions à importer, on voit les deux tables qui sont similaires (celles de **FirstThunk** et **OriginalFirstThunk**) car nous avons deux fois les adresses 205Ch et 2078h, soit deux tables d'un élément. Mais une table verra ses valeurs réécrites par le système lors du chargement du programme, il nous faut trouvé laquelle. Cela ne sera pas bien dur car les appels vers ces adresses se trouvent dans le code.

Après correction des valeurs on a

Champ	FileOffset	Valeur
OriginalFirstThunk	610h	204Ch
TimeDateStamp	614h	0
ForwarderChain	618h	0

Name	61Ch	206Ah (⇒ kernel.dll)
FirstThunk	620h	2000h (⇒ IAT)

Etude du PE leet.

PE, da leet version from Kharneth.

La première surprise en ouvrant ce PE est qu'il est très petit 666 octets, puis, deuxième surprise, il y a un paquet d'octets mis à 0. Du travail en perspective mais c'est un PE pour leets.

J'ai été rapide sur certains points. La lecture de du texte suivant est donc réservée au personne ayant déjà acquis une bonne connaissance sur le format des PE, soit en lisant depuis le début ce document, soit en ayant farfouillé précédemment dedans.

Je ne voulais pas faire un cours sur le PE, j'ai juste voulu que ce soit un document pouvant apporter quelques réponses lors de l'étrippage de PE.

Let's get in!

Reconstruction de l'entête DOS.

On regarde tout d'abord à *e_lfanew* (3Ch) pour voir où se situe la structure *IMAGE_NT_HEADERS* dans le PE. Il est à zéro et il semble y avoir peu de place à la suite pour l'écrire car on ne doit pas modifier les octets existant et la section (.code) à l'offset 9Ch va nous gêner. On va se faire chevaucher *IMAGE_DOS_HEADER* et *IMAGE_NT_HEADERS*.

Pour ce faire il faudrait déterminer en calculant les offsets dans les différentes structures afin que les octets existants ou à venir ne perturbent ni l'une ni l'autre. J'avoue avoir choisi un peu au hasard de mettre *IMAGE_NT_HEADERS* à l'offset 0Ch, ce qui a été un choix judicieux puisque une valeur présente (10Ch) se trouve dans le champs *AddressOfEntryPoint*, et qu'à cet adresse se trouve une séquence qui semble être une suite d'instructions.

Reconstruction des entêtes NT.

Remplissons la structure *IMAGE_FILE_HEADER*

Champ	commentaire
Machine	on met 14Ch
NumberOfSections	il semble n'y avoir qu'une section.
PointerToSymbolTable	est un élément de <i>IMAGE_DOS_HEADER</i> qui est à 40h
NumberOfSymbols	on le laisse à 0 pour ne pas interférer avec la valeur ci-dessus.
SizeOfOptionalHeader	on fait la soustraction de l'offset de la section '.code' avec celui de l'adresse où débute <i>IMAGE_NT_HEADERS</i> ce qui donne 78h.
Characteristics	on garde traditionnellement 10Fh.

Remplissons la structure IMAGE_OPTIONAL_HEADER

Champ	commentaire
Magic	on met 10Bh car c'est le magic number pour un exécutable win32
MajorLinkerVersion	j'ai mis 6
MinorLinkerVersion	et 66h pour avoir le linker 6.66 comme version :)
SizeOfCode	On le remplira après
SizeOfInitializedData	idem
SizeOfUninitializedData	idem
AddressOfEntryPoint	ce champ à déjà été rempli (10Ch).
BaseOfCode	mis à 0 car le code semble se trouver dans l'entête du PE.
BaseOfData	il s'y trouve déjà l'octet de e_lfanew
ImageBase	on le remplira plus tard.
SectionAlignment	on met 1000h, un classique
FileAlignment	cela semble être 200h
MajorOperatingSystemVersion	on met 4
MinorOperatingSystemVersion	on garde 0
MajorImageVersion	on s'en fout un peu
MinorImageVersion	idem
MajorSubsystemVersion	il faut avoir 4 au minium (merci Kharneth :))
MinorSubsystemVersion	on garde à 0
SizeOfImage	plus tard
SizeOfHeaders	puisque'on a un FileAlignment à 200h on garde cette valeur.
Subsystem	on met 2 (voir le PE normal)
SizeOfStackReserve	on met 1000h pour les trois champs suivants. Ce n'est pas bien important ici.
NumberOfRvaAndSizes	étant donné la taille de IMAGE_OPTIONAL_HEADERS, il ne peut y avoir que 3 éléments au maximum dans le tableau suivant. On met 3.

La section '.code'

Maintenant que le PE est mieux formé, on peut naviguer dedans avec quelques outils et on pourra remplir quelques champs qui nous manquent.

Faisons le plan mémoire RVA du PE chargé avec l'aide des informations de la section

nom	offset	taille
Header	0	200
.code	1000	9A

- **SizeOfCode** doit avoir la taille de la section c-à-d 9Ah
- **SizeOfInitializedData** reste à 0 car la section ne définit pas de données initialisées
- **SizeOfUninitializedData** reste aussi à 0 pour la même raison.

Grâce à ce plan mémoire on peut définir la valeur de **SizeOfImage** : 109Ah.

Il faut maintenant reconstruire l'IAT. On va prendre PEiD afin de déterminer où se trouve les éléments de cette tables. En ouvrant le PE avec et en cliquant sur la flèche à coté de *First bytes* on obtient de désassemblage du code de l'exécutable. On voit qu'il y a deux call vers des destinations contenues dans des dwords qui sont, respectivement, 29A1079h et 29A106C.

On regardant nos offsets relatifs et l'offset absolu des deux adresses trouvées dans le listing, on peut déterminer **ImageBase** de façon à ce qu'elles soient dans l'espace mémoire de la section `'.code'` une fois mappée. L'**ImageBase** est donc 29A000h.

La reconstruction de la tables des IMPORT_DESCRIPTOR

J'ai choisi de mettre cette table dans la section et non pas dans le header car cela provoque une erreur lors du chargement. Je ne vais pas expliquer en détail les modification -car si vous avez tout compris il est inutile de répéter- je vais simplement mettre le listing généré par mon outil servant justement à lister les imports.

On remplit la table IMPORT dans *IMAGE_OPTIONAL_HEADER* avec comme offset RVA 1030h et taille 3C (soit la taille de trois structures *IMAGE_IMPORT_DESCRIPTOR*)

```
*** SECTIONS ***
Section: HEADER
  VirtualOffset: 0
  VirtualSize : 200
  RawOffset : 0
  RawSize : 200

Section: .code
  VirtualOffset: 1000
  VirtualSize : 9A
  RawOffset : 200
  RawSize : 9A

*** IMPORT TABLE ***

Address: 1030 (RVA)
Size : 3C
Section: .code

*** IMPORT_DESCRIPTOR TABLE ***

===== USER32.DLL =====

Name          RAW | RVA
OriginalFirstThunk: 0 | 0
TimeDateStamp : 0 | ---
ForwarderChain : 0 | 0
FirstThunk : 230 | 1030

Imported Functions
-----
| Hint | RVA | Name
-----
| 019D | 0126 | MessageBoxA

===== KERNEL32.DLL =====

Name          RAW | RVA
OriginalFirstThunk: 0 | 0
```

```
TimeStamp      :    0 | ----
ForwarderChain :    0 |    0
FirstThunk     :  244 | 1044
```

Imported Functions

```
-----
| Hint | RVA | Name
-----
| 0080 | 0118 | ExitProcess
-----
```

Les informations contenues dans ce listing sont suffisamment explicites pour remplir la table à la main.

EPILOGUE

Pfiou ce dernier exercice n'a pas été de tout repos intellectuellement. J'ai même un peu raccourci le tutoriel pour des raisons personnelles. Ce double TP de Karneth était très bien. Une fois que vous aurez compris toutes les subtilités du PE (je m'y mets progressivement) et compris ce tutoriel, vous deviendrez un vrai leet. :)

info/reverse/defi_pe.txt · Dernière modification: 2005/04/13 18:03