

Formatul Fisierelor PE (Portable Executable)

Autor: Ionut Popescu (Nytro)

Website: <https://rstforums.com/forum/> (Romanian Security Team)

1. Introducere

Portable Executable este formatul de fisiere folosit de sistemul de operare Microsoft Windows pentru fisierele executabile si alte tipuri de fisiere care contin cod executabil, cod masina - care este executat direct de catre procesor.

Acest format este folosit in special de catre fisierele executabile cu extensia “.exe” si de catre bibliotecile de functii ce sunt incarcate de catre aceste executabile avand extensia “.dll” dar si de alte tipuri de fisiere cum ar fi controalele ActiveX (extensia “.ocx”) sau driverele/modulele sistemului de operare (extensia “.sys”).

Numele de “Portable” provine de la faptul ca formatul este acelasi pentru mai multe arhitecturi: x86, x86-64, IA32 si IA64, este “portabil”. Practic acest format este o versiune modificata a formatului COFF (Common Object File Format) folosit de sistemele UNIX in timp ce sistemele Linux folosesc formatul ELF (Executable and Linkable Format).

Acest format a aparut pentru prima oara odata cu sistemul de operare Windows NT 3.1 si a fost creat pentru a pastra compatibilitatea cu vechiul format, MS-DOS, sistemul anterior sistemului Windows.

2. Structura generala a fisierelor PE

Din motive de compatibilitate cu formatul folosit de MS-DOS (sistemul de operare al celor de la Microsoft de dinainte de Windows), fisierele PE contin practic un mic program pentru MS-DOS care afiseaza mesajul: *“This program cannot be run in DOS mode.”*, acesta fiind singurul lucru pe care il face, astfel, daca se incearca executarea unui fisier PE pe sistemul MS-DOS, se va afisa acel mesaj.

Formatul foloseste mai multe anteturi/headere pentru a defini datele folosite. Asadar fiecare fisier PE va contine atat un header MS-DOS necesar programului MS-DOS, cat si un header PE, necesar programului nostru.

Headerele sunt un grup de octeti, numar fix sau variabil, care definesc datele ce vor urma, niste octeti care ofera informatii despre structurile ce urmeaza, structuri pe care le definesc.

Pentru o detaliere mai simpla a acestui format voi afisa definitiile acestor headere ca structuri in C/C++. Astfel se vor putea observa mai usor dimensiunile fiecarui camp din structuri. Structurile pot fi regasite in SDK-uri in fisierul “WinNT.h”.

Ca structura generala un fisier PE este destul de simplu, este alcatuit din programul MS-DOS (denumit adesea "stub") care e alcatuit dintr-un header MS-DOS si programul MS-DOS, din headerele PE (vom vedea ca nu este unul singur) care include si tabelul de sectiuni, urmate de sectiunile propriu-zise, adica datele si instructiunile programului nostru.

Pe intelesul tuturor, primii octeti dintr-un fisier PE il reprezinta un mic program MS-DOS, urmat de niste headere care definesc datele ce urmeaza, urmate de datele propriu-zise ale fisierului PE: sectiuni de date si de cod.

Structura fisierelor PE

Header MS-DOS	Program MS-DOS
Program MS-DOS	
Header PE	Headere PE
Header optional PE	
Tabel sectiuni	
Sectiunea 1	Date si cod
Sectiunea 2	
...	
Sectiunea n	

Asadar prima parte a oricarui fisier executabil o reprezinta un mic program MS-DOS, adesea in jur de 240 de bytes. Deschis intr-un Hex Editor, acest program MS-DOS compus din header si cod arata cam asa (intreg programul MS-DOS, header si cod):

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
00000000	4d	5a	90	00	03	00	00	00	04	00	00	00	ff	ff	00	00	MZ
00000010	b8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	000.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	008...
00000030	00	00	00	00	00	00	00	00	00	00	00	00	f0	00	00	00!
00000040	0e	1f	ba	0e	00	b4	09	cd	21	b8	01	4c	cd	21	54	68	..°..'í!..Lí!Th
00000050	69	73	20	70	72	6f	67	72	61	6d	20	63	61	6e	6e	6f	is program cannc
00000060	74	20	62	65	20	72	75	6e	20	69	6e	20	44	4f	53	20	t be run in DOS
00000070	6d	6f	64	65	2e	0d	0d	0a	24	00	00	00	00	00	00	00	mode....\$.....
00000080	63	8a	9f	9f	27	eb	f1	cc	27	eb	f1	cc	27	eb	f1	cc	çŠŸŸ'ëñì'ëñì'ëñì
00000090	2e	93	62	cc	16	eb	f1	cc	27	eb	f0	cc	55	e8	f1	cc	."bì.ëñì'ësiUëñì
000000a0	2e	93	63	cc	26	eb	f1	cc	2e	93	64	cc	20	eb	f1	cc	."cì&ëñì."dì ëñì
000000b0	2e	93	72	cc	d1	eb	f1	cc	2e	93	75	cc	c4	eb	f1	cc	."rìÑëñì."uìÄëñì
000000c0	2e	93	65	cc	26	eb	f1	cc	2e	93	60	cc	26	eb	f1	cc	."eì&ëñì."ì&ëñì
000000d0	52	69	63	68	27	eb	f1	cc	00	00	00	00	00	00	00	00	Rich'ëñì.....
000000ef	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Un fisier PE deschis intr-un Hex Editor

Daca veti deschide un executabil/DLL cu Notepad sau orice alt editor, veti obtine un rezultat asemanator. Scopul acestui tutorial este de a intelege ce reprezinta acele date.

2.1. Header-ul MS-DOS

Headerul care contine informatii despre acest mic program este o structura (C/C++) numita **_IMAGE_DOS_HEADER**. Structura are dimensiunea de 64 de octeti si ca orice header contine informatii despre programul MS-DOS, informatii necesare pentru ca programul sa poata fi executat cu succes.

Pentru non-programatori, primii 64 de octeti din fisier ii reprezinta niste date care reprezinta headerul MS-DOS, niste informatii necesare pentru a putea executa acel program pe sistemul MS-DOS. Pentru a putea intelege aceste notiuni, e necesara o intelegere la nivel de baza a structurilor din C/C++.

Practic datele din fisier se mapeaza peste aceasta structura, astfel, primii 2 octeti dintr-un fisier PE vor fi reprezentati de campul **"e_magic"** din structura (daca veti deschide un fisier executabil sau o biblioteca de functii .dll intr-un editor de text, veti observa ca primele 2 caractere din fisier il reprezinta sirul de caractere "MZ", 2 caractere insumand 2 octeti, ceea ce in structura reprezinta campul **"e_magic"**), urmatorii 2 octeti reprezinta campul **"e_cblp"** si tot asa. Asadar, primii 64 de octeti ii reprezinta un header care contine informatiile definite mai jos.

In cazul in care nu sunteti familiari cu limbajele C/C++, nu va sperati, veti intelege despre ce e vorba.

```
typedef struct _IMAGE_DOS_HEADER { // DOS .EXE header
    WORD e_magic; // Magic number
    WORD e_cblp; // Bytes on last page of file
    WORD e_cp; // Pages in file
    WORD e_crlc; // Relocations
    WORD e_cparhdr; // Size of header in paragraphs
    WORD e_minalloc; // Minimum extra paragraphs needed
    WORD e_maxalloc; // Maximum extra paragraphs needed
    WORD e_ss; // Initial (relative) SS value
    WORD e_sp; // Initial SP value
    WORD e_csum; // Checksum
    WORD e_ip; // Initial IP value
    WORD e_cs; // Initial (relative) CS value
    WORD e_lfarlc; // File address of relocation table
    WORD e_ovno; // Overlay number
    WORD e_res[4]; // Reserved words
    WORD e_oemid; // OEM identifier (for e_oeminfo)
    WORD e_oeminfo; // OEM information; e_oemid specific
    WORD e_res2[10]; // Reserved words
    LONG e_lfanew; // File address of new exe header
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Structura care defineste header-ul MS-DOS

Unul dintre lucrurile de baza pe care trebuie sa le cunoasteti atunci cand programati pentru Windows il reprezinta tipurile de date¹. Astfel avem:

- *BYTE* – Un numar pe 8 biti (**1 byte**), definit ca “unsigned char”
- *CHAR* – Un numar pe 8 biti (**1 byte**), definit ca “char”
- *DWORD* – Un numar pe 32 de biti (**4 bytes**) fara semn, definit ca “unsigned long”
- *LONG* – Un numar pe 32 de biti (**4 bytes**) cu semn, definit ca “long”
- *ULONGLONG* – Un numar pe 64 de biti (**8 bytes**) fara semn, definit ca “unsigned long long”
- *WORD* – Un numar pe 16 biti (**2 bytes**) fara semn, definit ca “unsigned short”

Chiar daca sunteti cunoscatori ai limbajelor C/C++ va trebui sa va familiarizati cu tipurile de date specifice sistemului de operare Windows. Nu este nimic complicat, sunt practic doar “alte nume” pentru tipurile de date pe care le cunoasteti deja.

Practic avem nevoie sa stim aceste informatii deoarece fisierele PE se mapeaza perfect peste aceste structuri. Astfel, daca vom citii primii 64 de bytes dintr-un executabil, acestia vor fi exact acest header, iar daca vom citi 64 de bytes intr-o astfel de structura vom putea avea acces simplu si rapid la fiecare dintre campurile acestui header.

Astfel, primii 2 octeti (tipul WORD ocupa 2 bytes) ai fiecarui fisier PE vor reprezenta campul **e_magic** (un WORD are 2 octeti), urmatorii 2 octeti vor fi **e_cblp** si tot asa.

Sa citim o astfel de structura si sa vedem cum arata. Am scris un mic program care citeste un fisier PE, “kernel32.dll” si afiseaza informatiile din headerul sau PE.

Vom vedea ca majoritatea campurilor au valoarea 0 deoarece nu sunt folosite. Pe noi ne intereseaza doar doua dintre aceste campuri: e_magic si e_lfanew.

```
/*
   Afisarea structurii IMAGE_DOS_HEADER a unui fisier PE
   Autor: Ionut Popescu (Nytro) - Romanian Security Team
*/

#include <stdio.h>
#include <windows.h>

int main()
{
    FILE *FD = NULL;
    IMAGE_DOS_HEADER dos;
    size_t Len = 0;
```

¹ Windows Data Types (Windows) - <http://msdn.microsoft.com/en-us/library/windows/desktop/aa383751%28v-vs.85%29.aspx>

```

// Deschidem fisierul pentru citire

FD = fopen("C:\\Windows\\system32\\kernel32.dll", "rb");

if(!FD)
{
    puts("Nu am putut deschide fisierul, verificati locatia!");
    return 1;
}

// Citim fisierul
// Vom citi primii 64 de octeti din fisier
// Deoarece marimea structurii IMAGE_DOS_HEADER este de 64 de octeti
// Astfel, vom avea in structura "dos" (64 de octeti) primii 64 de
// octeti din fisier, care se vor suprapune perfect peste datele
// continute de structura IMAGE_DOS_HEADER

Len = fread(&dos, 1, sizeof(dos), FD);

if(Len != sizeof(dos))
{
    puts("Nu am putut citi fisierul!");
    return 1;
}

fclose(FD);

// Afisam informatiile
// Afisam practice fiecare camp din structura IMAGE_DOS_HEADER

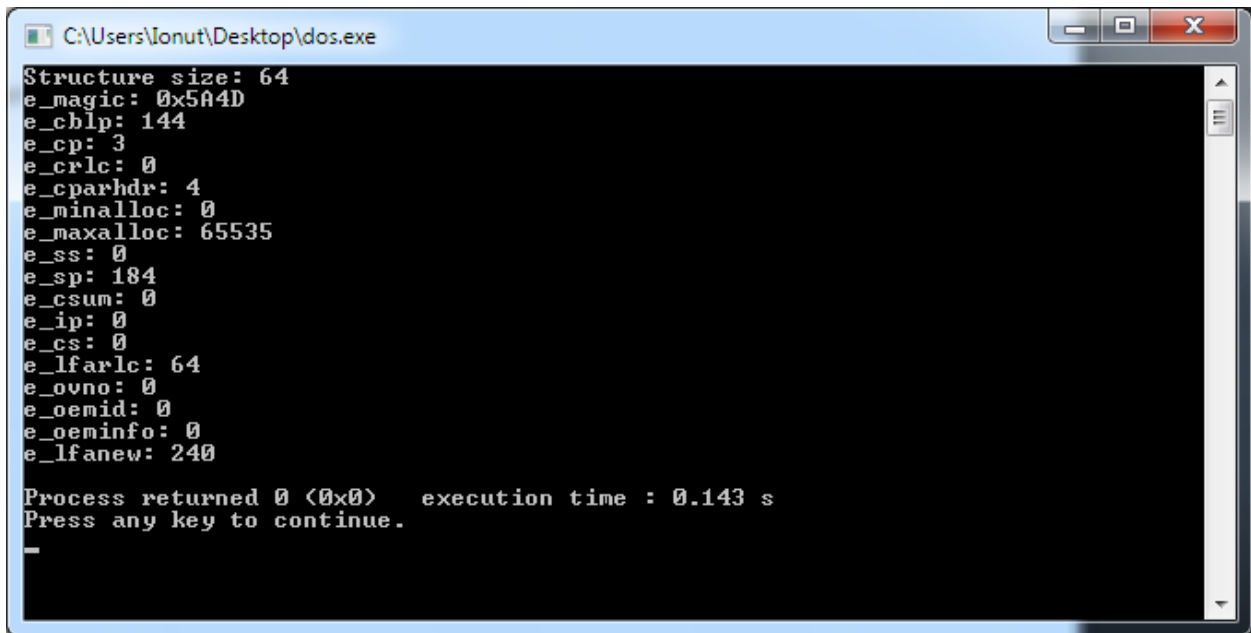
printf("Structure size: %d\n", sizeof(dos));
printf("e_magic: 0x%X\n", dos.e_magic);
printf("e_cblp: %d\n", dos.e_cblp);
printf("e_cp: %d\n", dos.e_cp);
printf("e_crlc: %d\n", dos.e_crlc);
printf("e_cparhdr: %d\n", dos.e_cparhdr);
printf("e_minalloc: %d\n", dos.e_minalloc);
printf("e_maxalloc: %d\n", dos.e_maxalloc);
printf("e_ss: %d\n", dos.e_ss);
printf("e_sp: %d\n", dos.e_sp);
printf("e_csum: %d\n", dos.e_csum);
printf("e_ip: %d\n", dos.e_ip);
printf("e_cs: %d\n", dos.e_cs);
printf("e_lfarlc: %d\n", dos.e_lfarlc);
printf("e_ovno: %d\n", dos.e_ovno);
printf("e_oemid: %d\n", dos.e_oemid);
printf("e_oeminfo: %d\n", dos.e_oeminfo);
printf("e_lfanew: %lu\n", dos.e_lfanew);

return 0;
}

```

Programul care afiseaza headerul DOS al unui fisier PE

O executie a acetui program ar produce rezultate asemanatoare:



```
C:\Users\Ionut\Desktop\dos.exe
Structure size: 64
e_magic: 0x5A4D
e_cblp: 144
e_cp: 3
e_crlc: 0
e_cparhdr: 4
e_minalloc: 0
e_maxalloc: 65535
e_ss: 0
e_sp: 184
e_csum: 0
e_ip: 0
e_cs: 0
e_lfarlc: 64
e_ovno: 0
e_oemid: 0
e_oeminfo: 0
e_lfanew: 240

Process returned 0 (0x0)   execution time : 0.143 s
Press any key to continue.
-
```

Campul **e_magic**, primii doi octeti ai fiecarui fisier PE, reprezinta o “semnatura”, un identificator care ne spune ca avem un fisier MS-DOS. Daca ati deschis vreodata un executabil in Notepad ati observat ca incepe cu literele “MZ”. Aceste litere provin de la numele inginerului Microsoft - Mark Zbykowski.²

Pentru programatori, aceste litere apar in fisier ca fiind octetii 0x4D (M) si 0x5A (Z). De observat ca in screenshot-ul de mai sus, numarul este **0x5A4D**, deoarece procesoarele Intel sunt little-endian si in memorie numerele sunt memorate in ordinea inversa a octetilor (de exemplu, numarul **3**, intr-o variabila **int**, care ocupa 4 bytes, va fi memorat in memorie ca “**03 00 00 00**”, mai exact primul octet de la adresa la care e memorat numarul in memorie va fi “cel mai putin semnificativ”). In WinNT.h:

```
#define IMAGE_DOS_SIGNATURE          0x5A4D    // MZ
```

Campul **e_lfanew** este cel care ne intereseaza in mod special, deoarece acesta ne indica unde se termina programul MS-DOS si unde incepe headerul PE. Astfel vom putea sari peste acest program si vom ajunge la ceea ce ne intereseaza.

Insa ce face acest “stub” (cum este numit acest mic program) MS-DOS? Cum afiseaza acel mesaj? Contine cod?

Dupa cum spuneam mai sus, headerul MS-DOS are 64 de bytes, asadar programul MS-DOS va incepe imediat dupa acest header, de la octetul 0x40 (64 in zecimal).

² MZ - <http://wiki.osdev.org/MZ>

In imaginea de mai jos se poate vedea portiunea care afiseaza acel mesaj si mesajul.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
00000000	4d	5a	90	00	03	00	00	00	04	00	00	00	ff	ff	00	00	MZ
00000010	b8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	,.....@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	f0	00	00	008...
00000040	0e	1f	ba	0e	00	b4	09	cd	21	b8	01	4c	cd	21	54	68	..°..'í!„Lí!Th
00000050	69	73	20	70	72	6f	67	72	61	6d	20	63	61	6e	6e	6f	is program cannc
00000060	74	20	62	65	20	72	75	6e	20	69	6e	20	44	4f	53	20	t be run in DOS
00000070	6d	6f	64	65	2e	0d	0d	0a	24	00	00	00	00	00	00	00	node....\$......

Codul de afisare si mesajul programului MS-DOS

Partea selectata va afisa acel mesaj. Primii octeti sunt:

```
0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd
```

Octetii (in hexazecimal) ce reprezinta codul (in limbajului procesorului) ce afiseaza mesajul

Daca vom dezasambla acest cod (cu un disassembler pe 16 biti, NASM de exemplu) vom obtine:

```
505 C:\Program Files\NASM>ndisasm.exe -b 16 ms-dos.dll
506 00000000 0E          push cs
507 00000001 1F          pop ds
508 00000002 BA0E00     mov dx,0xe
509 00000005 B409      mov ah,0x9|
510 00000007 CD21      int 0x21
```

Codul MS-DOS, dezasamblat, care afiseaza mesajul

Primele doua linii vor pune in registrul "ds" (data segment) valoarea registrului "cs" (code segment), ceea ce va indica faptul ca datele se afla in acelasi segment ca si codul. Vom vedea pe viitor ca exista sectiuni/segmente separate pentru date si pentru cod. Primele doua instructiuni (push cs, pop ds) reprezinta ceva de forma "ds = cs;", adica sectiunea de date este aceeasi ca si sectiunea de cod.

Urmatoarele doua linii vor seta registrul "dx" la valoarea 0xE si registrul "ah" la valoarea 0x9 urmand apoi instructiunea "int 0x21". Aceasta instructiune va apela interruptul 0x21, un fel de functie a sistemului MS-DOS care practic afiseaza un sir de caractere pe ecran.³ Functia este identificata prin registrul "ah" (valoarea 0x9), iar parametrul, sirul de caractere care urmeaza sa fie afisat, va fi un sir de caractere care se va termina cu caracterul "\$", e indicat de registrul "dx".

³ DOS Interrupts - <http://spike.scu.edu.au/~barry/interrupts.html#ah09>

Daca ne uitam in hex editor putem observa cu usurinta faptul ca sirul de caractere se afla la pozitia **0xE** (fata de inceputul codului executabil) si ca se termina cu caracterul "\$".

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
00000000	4d	5a	90	00	03	00	00	00	04	00	00	00	ff	ff	00	00	MZÿÿ..
00000010	b8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	,@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	f0	00	00	008...
0000004e	0e	1f	ba	0e	00	b4	09	cd	21	b8	01	4c	cd	21	54	68	..°..'í!..Lí!T:
00000050	69	73	20	70	72	6f	67	72	61	6d	20	63	61	6e	6e	6f	is program canno
00000060	74	20	62	65	20	72	75	6e	20	69	6e	20	44	4f	53	20	t be run in DOS
00000070	6d	6f	64	65	2e	0d	0d	0a	24	00	00	00	00	00	00	00	mode....\$.....

Sirul de caractere afisat de programul MS-DOS

2.2. Header-ul PE

Dupa cum spuneam la prezentarea header-ului MS-DOS, ultimul cand din aceasta structura, ultimii 4 bytes din cei 64, campul **e_lfanew**, reprezinta un numar pe 4 octeti care indica locatia "noului" header, indica exact pozitia in fisier de unde incepe header-ul PE.

Campul este un numar LONG, formatul fiind little-endian, astfel octetii sunt in ordine inversa. Desigur, citirea acestor 4 octeti intr-o variabila "long" se va face corect, iar variabila va contine valoarea corecta deoarece formatul din fisier este formatul in care e memorata acea valoare in memorie, pe procesoarele little-endian.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
00000000	4d	5a	90	00	03	00	00	00	04	00	00	00	ff	ff	00	00	MZÿÿ..
00000010	b8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	,@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000003c	00	00	00	00	00	00	00	00	00	00	00	00	f0	00	00	00\$...
00000040	0e	1f	ba	0e	00	b4	09	cd	21	b8	01	4c	cd	21	54	68	..°..'í!..Lí!Th
00000050	69	73	20	70	72	6f	67	72	61	6d	20	63	61	6e	6e	6f	is program canno
00000060	74	20	62	65	20	72	75	6e	20	69	6e	20	44	4f	53	20	t be run in DOS
00000070	6d	6f	64	65	2e	0d	0d	0a	24	00	00	00	00	00	00	00	mode....\$.....
00000080	63	8a	9f	9f	27	eb	f1	cc	27	eb	f1	cc	27	eb	f1	cc	cŠÿÿ'ëñì'ëñì'ëñì
00000090	2e	93	62	cc	16	eb	f1	cc	27	eb	f0	cc	55	e8	f1	cc	..`bì.ëñì'ëñìUëñì
000000a0	2e	93	63	cc	26	eb	f1	cc	2e	93	64	cc	20	eb	f1	cc	..`cìëñì. `dì ëñì
000000b0	2e	93	72	cc	d1	eb	f1	cc	2e	93	75	cc	c4	eb	f1	cc	..`rìñëñì. `uìäëñì
000000c0	2e	93	65	cc	26	eb	f1	cc	2e	93	60	cc	26	eb	f1	cc	..`eìëñì. `ìëñì
000000d0	52	69	63	68	27	eb	f1	cc	00	00	00	00	00	00	00	00	Rich'ëñì.....
000000e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000f0	50	45	00	00	4c	01	04	00	15	3b	b8	50	00	00	00	00	PE...;P....
00000100	00	00	00	00	e0	00	02	21	0b	01	09	00	00	50	0c	00ä...!.....P..

Locatia din headerul MS-DOS (rosu) ce indica headerul PE (verde)

Dupa cum se poate vedea in exemplul de mai sus, header-ul PE se afla la locatia **0xf0** (240 zecimal) in executabil.

Imediat la aceasta locatie putem gasi noul header PE, care poate fi gasit in "WinNT.h" cu numele **IMAGE_NT_HEADERS**.

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

Structura IMAGE_NT_HEADERS

Un lucru pe care ar trebui sa il avem in considerare il reprezinta faptul ca exista aceste structuri sunt diferite pe versiunile de 32 de biti si de 64 de biti:

```
#ifdef _WIN64
typedef IMAGE_NT_HEADERS64 IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS64 PIMAGE_NT_HEADERS;
#else
typedef IMAGE_NT_HEADERS32 IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS32 PIMAGE_NT_HEADERS;
#endif
```

Diferenta dintre structura pe 32 de biti si cea pe 64 de biti

Structura este aparent simpla, contine doar 3 campuri:

- **Signature**: un numar de 4 octeti care identifica aceasta structura ca fiind o structura PE, definitia sa poate fi gasita ca

```
#define IMAGE_NT_SIGNATURE 0x00004550 // PE00
```

Numaui pe 4 octeti (little endian) care indica semnatura unui header PE

Se poate identifica usor in executabil, caracterele text "**PE**" identifica inceputul acestui header in fisier. De acolo incepe structura IMAGE_NT_HEADERS.

Urmatoarele doua campuri din aceasta structura sunt de asemenea structuri:

- **FileHeader**, o structura de tipul **IMAGE_FILE_HEADER**, simpla, care contine informatii referitoare la proprietatile fisierului

- **OptionalHeader**, o structura complexa, de tipul **IMAGE_OPTIONAL_HEADER** care contine mai multe informatii. Desi se gaseste in orice fisier executabil sau in orice biblioteca de functii, aceasta structura poate fi optionala pentru fisierele obiect, cum ar fi rezultatul compilarii unui singur fisier

Campul "**FileHeader**", structura **IMAGE_FILE_HEADER** este definita astfel:

```
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;
```

Structura IMAGE_FILE_HEADER

Informatiile continute de aceasta structura (20 de octeti) sunt urmatoarele:

- **Machine**: Un numar care identifica arhitectura procesorului, de exemplu i386, IA64...

Exemple:

```
#define IMAGE_FILE_MACHINE_I386          0x014c  // Intel 386.
#define IMAGE_FILE_MACHINE_ARM          0x01c0  // ARM Little-Endian
#define IMAGE_FILE_MACHINE_POWERPC     0x01f0  // IBM PowerPC
#define IMAGE_FILE_MACHINE_IA64        0x0200  // Intel 64
```

Exemple de arhitecturi ale procesorului pentru un fisier PE

In cazul exemplului nostru, imediat dupa primii 4 octeti care reprezinta semnatura PE (50 45 00 00), urmeaza 2 octeti care reprezinta valoarea acestui camp: 4c 01, adica numarul **0x014c**, ceea ce inseamna ca executabilul nostru este destinat arhitecturii procesoarelor **Intel 386** sau mai noi.

- **NumberOfSections**: Reprezinta numarul de sectiuni din fisier. Vom observa ca fisierele contin date inpartite in sectiuni. In fisierul analizat de noi, imediat dupa cei 2 octeti care reprezinta campul "Machine", observam ca fisierul nostru contine **4** sectiuni.
- **TimeDateStamp**: Un numar pe 4 octeti (time_t din C) care reprezinta data la care fisierul a fost creat, masurat in numarul de secunde trecute de la 1 Ian. 1970. Putem observa urmatoarea valoare pentru acest camp: **15 3b b8 50** adica numarul **0x50b83b15**, in zecimal 1354251029, ceea ce inseamna ca fisierul nostru a fost creat la data de "**Fri, 30 Nov 2012 04:50:29 GMT**".

- **PointerToSymbolTable:** Acest camp ar trebui sa fie un pointer catre un tabel de simboluri COFF insa nu mai este folosit si ar trebui sa aiba valoarea 0.
- **NumberOfSymbols:** Numarul de simboluri din tabelul de mai sus care nu este folosit, asadar ar trebui sa aiba valoarea 0.
- **SizeOfOptionalHeader:** Contine marimea header-ului optional, cel care urmeaza imediat dupa aceasta structura
- **Characteristics:** Un set de flag-uri care indica informatii despre fisier: daca este executabil sau biblioteca de functii, sau altele.

Exemple:

```
#define IMAGE_FILE_EXECUTABLE_IMAGE          0x0002 // File is executable
(i.e. no unresolved external references).
#define IMAGE_FILE_LARGE_ADDRESS_AWARE      0x0020 // App can handle >2gb
addresses
#define IMAGE_FILE_32BIT_MACHINE            0x0100 // 32 bit word machine.
#define IMAGE_FILE_SYSTEM                   0x1000 // System File.
#define IMAGE_FILE_DLL                      0x2000 // File is a DLL.
```

Caracteristici posibile pentru un fisier PE

Campul **OptionalHeader** (224 de octeti), structura **IMAGE_FILE_HEADER**, este definita astfel:

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    //
    // Standard fields.
    //

    WORD    Magic;
    BYTE    MajorLinkerVersion;
    BYTE    MinorLinkerVersion;
    DWORD   SizeOfCode;
    DWORD   SizeOfInitializedData;
    DWORD   SizeOfUninitializedData;
    DWORD   AddressOfEntryPoint;
    DWORD   BaseOfCode;
    DWORD   BaseOfData;

    //
    // NT additional fields.
    //

    DWORD   ImageBase;
    DWORD   SectionAlignment;
    DWORD   FileAlignment;
    WORD    MajorOperatingSystemVersion;
```

```

WORD    MinorOperatingSystemVersion;
WORD    MajorImageVersion;
WORD    MinorImageVersion;
WORD    MajorSubsystemVersion;
WORD    MinorSubsystemVersion;
DWORD   Win32VersionValue;
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_HEADER32, *PIMAGE_OPTIONAL_HEADER32;

```

Structura IMAGE_OPTIONAL_HEADER

Intr-un fisier PE, un exemplu de astfel de structura, cei 224 de octeti din care este alcatuita:

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
00000080	63	8a	9f	9f	27	eb	f1	cc	27	eb	f1	cc	27	eb	f1	cc	cšŸÿ'ëñì'ëñì'ëñì
00000090	2e	93	62	cc	16	eb	f1	cc	27	eb	f0	cc	55	e8	f1	cc	."bì.ëñì'ësiUëñì
000000a0	2e	93	63	cc	26	eb	f1	cc	2e	93	64	cc	20	eb	f1	cc	."cìëñì."dì ëñì
000000b0	2e	93	72	cc	d1	eb	f1	cc	2e	93	75	cc	c4	eb	f1	cc	."rìÑëñì."uìÄëñì
000000c0	2e	93	65	cc	26	eb	f1	cc	2e	93	60	cc	26	eb	f1	cc	."eìëñì."`ìëñì
000000d0	52	69	63	68	27	eb	f1	cc	00	00	00	00	00	00	00	00	Rich'ëñì.....
000000e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000f0	50	45	00	00	4c	01	04	00	15	3b	b8	50	00	00	00	00	PE..L....;P....
00000100	00	00	00	00	e0	00	02	21	0b	01	09	00	00	50	0c	00	...à..!.....P..
00000110	00	e0	00	00	00	00	00	00	6f	cd	04	00	00	10	00	00	..à.....cí.....
00000120	00	00	0c	00	00	00	de	77	00	10	00	00	00	10	00	00Pw.....
00000130	06	00	01	00	06	00	01	00	06	00	01	00	00	00	00	00
00000140	00	40	0d	00	00	10	00	00	c7	ff	0d	00	03	00	40	01	..@.....Çÿ.....@.
00000150	00	00	04	00	00	10	00	00	00	00	10	00	00	10	00	00
00000160	00	00	00	00	10	00	00	00	c0	51	0b	00	b1	a9	00	00ÀQ..±@..
00000170	74	fb	0b	00	f4	01	00	00	00	70	0c	00	28	05	00	00	tú..ó...p..(..
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000190	00	80	0c	00	b0	b0	00	00	b4	59	0c	00	38	00	00	00	..e..""`Y..8...
000001a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001b0	00	00	00	00	00	00	00	00	90	28	08	00	40	00	00	00(..@...
000001c0	00	00	00	00	00	00	00	00	00	10	00	00	fc	0d	00	00ú...
000001d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001e0	00	00	00	00	00	00	00	00	2e	74	65	78	74	00	00	00text...
000001f0	15	4a	0c	00	00	10	00	00	00	50	0c	00	00	10	00	00	..J.....P.....
00000200	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60`

Partea selectata reprezinta structura IMAGE_OPTIONAL_HEADER

Structura **IMAGE_FILE_HEADER** este alcatuita din urmatoarele campuri:

- **Magic:** Un numar pe 2 octeti care identifica practic tipul de executabil, de cele mai multe ori PE32 (valoarea 0x10b) dar poate fi si PE32+ (valoarea 0x20b), executabile ce permit adresarea pe 64 de biti in timp ce limiteaza dimensiunea executabilului la 2 GB
- **MajorLinkerVersion, MinorLinkerVersion:** Dupa cum spune numele, aceste campuri de cate un octet identifica versiunea link-erului folosita. Cum aceasta structura, IMAGE_FILE_HEADER nu se regaseste in fisierele obiect, inseamna ca structura e creata la linkare, de catre linker

Vom observa ca datele dintr-un fisier PE sunt impartite in sectiuni. Astfel avem sectiuni pentru cod (cel mai adesea o sectiune cu numele “.text”), sectiuni pentru date initializate sau date neinitializate (cel mai adesea numele “.bss”) care contin variabilele globale dar si variabilele statice, sectiuni pentru date “read-only” ca “.rdata” ce contin siruri de caractere sau constante.

- **SizeOfCode:** Indica dimensiunea sectiunii de cod (“.text”) sau dimensiunea tuturor sectiunilor de cod in cazul in care sunt mai multe
- **SizeOfInitializedData:** Indica dimensiunea sectiunii de date initializate sau dimensiunea tuturor sectiunilor de date initializate in cazul in care sunt mai multe
- **SizeOfUninitializedData:** Indica dimensiunea sectiunii de date neinitializate sau dimensiunea tuturor sectiunilor de date neinitializate in cazul in care sunt mai multe
- **AddressOfEntryPoint:** Un camp foarte important, o valoare RVA (Relative Value Address) adica un pointer catre o locatie relativa la adresa la care este incarcat modulul (.exe, .dll) in memorie (vezi mai jos campul ImageBase). E adresa de unde se incepe executia codului pentru un executabil, iau pentru o biblioteca de functii este locatia functiei de initializare (DllMain) sau “0” daca nu exista o astfel de functie
- **BaseOfCode:** La fel, o adresa relativa la ImageBase, locatia la care incepe sectiunea de cod a fisierului, incarcat in memorie
- **BaseOfData:** Adresa relativa unde datele sunt incarcate in memorie
- **ImageBase:** Un camp important, adresa (in spatiul de adrese al procesului respectiv) unde fisierul ar “prefera” sa fie incarcat in memorie. Trebuie sa fie un multiplu de 64KB si adresa implicita este 0x00400000
- **SectionAlignment:** Un lucru important despre fisierele PE este faptul ca difera modul in care arata pe disc si modul in care sunt incarcate in memorie, iar alinierea sectiunilor este aceasta cauza. Dupa cum am spus anterior, un fisier PE este format din sectiuni. Aceste sectiuni, in

memorie, trebuie aliniate la dimensiunea specificata de acest camp, de cele mai multe ori 0x1000 (4096 de octeti – dimensiunea unei pagini de memorie). Astfel, chiar daca o sectiune ar trebui sa inceapa de la adresa 0x2001, acesta nu este un multiplu de 0x1000 si sectiunea va fi incarcata in memorie la adresa 0x3000.

- **FileAlignment:** De asemenea, ca si campul anterior, sectiunile trebuie sa fie aliniate si in fisierul de pe disc, la un multiplu de valoarea indicata de acest camp. Implicit are valoarea 512 bytes, dar poate fi orice putere a lui 2 intre 512 si 64K.
- **MajorOperatingSystemVersion, MinorOperatingSystemVersion:** Versiunea sistemului de operare destinat executie fisierului. De exemplu 5.1 e Windows XP, 6.0 e Windows Vista si 6.1 e Windows 7.
- **MajorImageVersion, MinorImageVersion:** Versiunea fisierului
- **MajorSubsystemVersion, MinorSubsystemVersion:** Versiunea subsystemului (Windows Console, Windows...) – Vezi campul “Subsystem”
- **Win32VersionValue:** Nu e folosit, ar trebui sa fie 0
- **SizeOfImage:** Dimensiunea imaginii incarcate in memorie, inclusiv headerele. Trebuie sa fie un multiplu de “SectionAlignment”
- **SizeOfHeaders:** Dimensiunea tuturor headerelor: header DOS, header PE, headerle sectiunilor, rotunjita la un multiplu al campului “FileAlignment”
- **Checksum:** O suma pe 4 octeti, verificata la incarcare pentru DLL-urile care sunt incarcate in timpul procesului de boot, DLL-urile incarcate intr-un proces critic si pentru toate driverele. Algoritmul este incorporat in ImagHelp.dll
- **Subsystem:** Subsystemul fisierului. De exemplu, poate fi:

```
#define IMAGE_SUBSYSTEM_NATIVE          1    // Image doesn't require a
subsystem.
#define IMAGE_SUBSYSTEM_WINDOWS_GUI     2    // Image runs in the Windows
GUI subsystem.
#define IMAGE_SUBSYSTEM_WINDOWS_CUI     3    // Image runs in the Windows
character subsystem.
```

Subsystemul sau tipul fisierului PE

Mai exact, are valoarea 1 daca este un driver, valoarea 2 daca este un executabil grafic (foloseste ferestre) sau 3 daca este o aplicatie in linie de comanda.

- **DllCharacteristics:** Diverse trasaturi cum ar fi: poate fi mutat la o adresa diferita, nu foloseste SEH (Structured Exception Handler) sau e necesara fortarea verificarii integritatii
- **SizeOfStackReserve:** Ce dimensiune sa fie rezervata, pastrata pentru stiva folosita de executabil. Nu este alocata, ci este doar specificata ca dimensiune maxima
- **SizeOfStackCommit:** Cat spatiu sa fie alocat pe stiva initial, si cat spatiu sa fie alocat cand mai e necesara o alocare, pana cand se atinge dimensiunea specificata de campul anterior
- **SizeOfHeapReserve:** Cat spatiu sa fie rezervat pentru heap
- **SizeOfHeapCommit:** Cat spatiu sa fie alocat initial si cat sa fie alocat atunci cand este necesar, pana se ating dimensiunea campului anterior
- **LoaderFlags:** Camp rezervat, nefolosit, ar trebui sa fie 0
- **NumberOfRvaAndSizes:** Numarul de intrari in tabelul urmator, DataDirectory, cel mai adesea 16 (0x10). Inainte de a parcurge acest tabel trebuie verificata aceasta valoare
- **DataDirectory:** Un vector [NumberOfRvaAndSizes] de structuri de tipul "IMAGE_DATA_DIRECTORY".

Campul **DataDirectory** contine de obicei **16** structuri de tipul **IMAGE_DATA_DIRECTORY**, structura definita astfel:

```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;

#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES    16
```

Structura IMAGE_DATA_DIRECTORY

Campul "VirtualAddress" reprezinta adresa RVA (relativa la ImageBase, adica adresa la care este incarcat in memorie fisierul nostru) unde se afla datele respective iar campul "Size" reprezinta dimensiunea datelor.

Datele continute de acest **vector**, de DataDirectory, sunt predefinite. Astfel, indecsii elementelor de tipul IMAGE_DATA_DIRECTORY din vector sunt urmatorii:

```

// Directory Entries

#define IMAGE_DIRECTORY_ENTRY_EXPORT 0 // Export Directory
#define IMAGE_DIRECTORY_ENTRY_IMPORT 1 // Import Directory
#define IMAGE_DIRECTORY_ENTRY_RESOURCE 2 // Resource Directory
#define IMAGE_DIRECTORY_ENTRY_EXCEPTION 3 // Exception Directory
#define IMAGE_DIRECTORY_ENTRY_SECURITY 4 // Security Directory
#define IMAGE_DIRECTORY_ENTRY_BASERELOC 5 // Base Relocation Table
#define IMAGE_DIRECTORY_ENTRY_DEBUG 6 // Debug Directory
// IMAGE_DIRECTORY_ENTRY_COPYRIGHT 7 // (X86 usage)
#define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE 7 // Architecture Specific
Data
#define IMAGE_DIRECTORY_ENTRY_GLOBALPTR 8 // RVA of GP
#define IMAGE_DIRECTORY_ENTRY_TLS 9 // TLS Directory
#define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG 10 // Load Configuration
Directory
#define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT 11 // Bound Import Directory
in headers
#define IMAGE_DIRECTORY_ENTRY_IAT 12 // Import Address Table
#define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT 13 // Delay Load Import
Descriptors
#define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14 // COM Runtime descriptor

```

Indicii elementelor din structura DataDirectory

Mai exact, primul element din vector (indicele 0) va contine tabelul de functii exportate, urmatorul va contine functiile importate, urmatorul resursele continute de fisier etc. Elementele din vector nu sunt obligatorii, daca un element nu este prezent, atunci va avea valoarea 0.

In cazul in care v-ati pierdut printre atatea structuri, aveti mai jos o imagine de ansamblu a acestor headere: structurile si cateva campuri luate ca exemplu.

Datele colorate in aceasta imagine sunt:

1. Portocaliu (cel de sus): primii 2 octeti din fisier, campul "e_magic" din structura IMAGE_DOS_HEADER
2. Negru: Headerul complet MS-DOS, intreaga structura IMAGE_DOS_HEADER
3. Galbel: Ultimii 4 octeti din headerul MS_DOS, campul "e_lfanew", practice locatia la care se afla headerul PE, dupa cum se poate vedea mai jos, tot cu galben, acolo incepe headerul PE
4. Rosu: Sirul de caractere afisat de programul MS-DOS
5. Portocaliu (cel de jos): primii 4 octeti din header-ul PE, campul Signature din structura IMAGE_NT_HEADERS, indica inceputul headerelor PE
6. Verde: Structura IMAGE_FILE_HEADER, imediat dupa campul prezentat mai sus
7. Roz: Doua campuri din structura IMAGE_FILE_HEADER, primele doua elemente ale structurii
8. Albastru: Structura IMAGE_OPTIONAL_HEADER (incompleta, nu a incapat tot)
9. Mov: Ca exemplu, elementele "Major/minor linker version", ceea ce indica versiunea 09.00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ	
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00		
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000030	00	00	00	00	00	00	00	00	00	00	00	00	F0	00	00	00		
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68!..L. Th															
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno															
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS															
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode...\$.....															
00000080	63	8A	9F	9F	27	EB	F1	CC	27	EB	F1	CC	27	EB	F1	CC	c.....'.....'															
00000090	2E	93	62	CC	16	EB	F1	CC	27	EB	F0	CC	55	E8	F1	CC	..b.....'.....U..															
000000A0	2E	93	63	CC	26	EB	F1	CC	2E	93	64	CC	20	EB	F1	CC	..c.&.....d. ...															
000000B0	2E	93	72	CC	D1	EB	F1	CC	2E	93	75	CC	C4	EB	F1	CC	..r.....u.															
000000C0	2E	93	65	CC	26	EB	F1	CC	2E	93	60	CC	26	EB	F1	CC	..e.&.....`&...`															
000000D0	52	69	63	68	27	EB	F1	CC	00	00	00	00	00	00	00	00	Rich'.....															
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000000F0	50	45	00	00	4C	01	04	00	3E	BD	6D	50	00	00	00	00	PE...!...>.mP....															
00000100	00	00	00	00	E0	00	02	21	0B	01	09	00	00	50	0C	00!.....P..															
00000110	00	E0	00	00	00	00	00	00	6F	CD	04	00	00	10	00	00o.....															
00000120	00	00	0C	00	00	00	DE	77	00	10	00	00	00	10	00	00w.....															
00000130	06	00	01	00	06	00	01	00	06	00	01	00	00	00	00	00															
00000140	00	40	0D	00	00	10	00	00	25	11	0E	00	03	00	40	01	..@.....\$.....@.															
00000150	00	00	04	00	00	10	00	00	00	00	10	00	00	10	00	00															
00000160	00	00	00	00	10	00	00	00	C0	51	0B	00	B1	A9	00	00Q.....															
00000170	74	FB	0B	00	F4	01	00	00	00	70	0C	00	28	05	00	00	t.....p..(....															
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
00000190	00	80	0C	00	B0	B0	00	00	B4	59	0C	00	38	00	00	00Y..8....															
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000001B0	00	00	00	00	00	00	00	00	90	28	08	00	40	00	00	00(..@....															
000001C0	00	00	00	00	00	00	00	00	10	00	00	FC	0D	00	00	00															

Imaginea de ansamblu a unui fisier PE

Putem verifica ce contin fiecare dintre aceste campuri cu un program simplu, care afiseaza datele din FileHeader, din OptionalHeader dar si vectorul DataDirectory.

```

/*
  Afisarea structurilor IMAGE_FILE_HEADER si IMAGE_OPTIONAL_HEADER
  Autor: Ionut Popescu (Nytro) - Romanian Security Team
*/

#include <stdio.h>
#include <windows.h>

int main()
{
    FILE *FD = NULL;
    IMAGE_DOS_HEADER dos;
    IMAGE_NT_HEADERS ntHeaders;
    size_t Len = 0;

    // Deschidem fisierul pentru citire

```

```

FD = fopen("C:\\Windows\\system32\\kernel32.dll", "rb");

if(!FD)
{
    puts("Nu am putut deschide fisierul!");
    return 1;
}

// Citim mai intai headerul MS-DOS
Len = fread(&dos, 1, sizeof(dos), FD);

if(Len != sizeof(dos))
{
    puts("Nu am putut citi fisierul!");
    return 1;
}

// Sarim la headerele PE

if(fseek(FD, dos.e_lfanew, SEEK_SET) != 0)
{
    printf("Nu am putut sari le headerul PE: %lu", dos.e_lfanew);
    return 1;
}

// Citim structura IMAGE_NT_HEADERS
// Structura ce contine IMAGE_FILE_HEADER si IMAGE_OPTIONAL_HEADER

Len = fread(&ntHeaders, 1, sizeof(ntHeaders), FD);

if(Len != sizeof(ntHeaders))
{
    puts("Nu am putut citi IMAGE_NT_HEADERS!");
    return 1;
}

fclose(FD);

// Afisam marimile structurilor

printf("Sizeof IMAGE_NT_HEADERS: %d\n", sizeof(IMAGE_NT_HEADERS));
printf("Sizeof IMAGE_FILE_HEADER: %d\n", sizeof(IMAGE_FILE_HEADER));
printf("Sizeof IMAGE_OPTIONAL_HEADER: %d\n",
sizeof(IMAGE_OPTIONAL_HEADER));
printf("Sizeof IMAGE_DATA_DIRECTORY: %d\n\n",
sizeof(IMAGE_DATA_DIRECTORY));

// Afisam campurile FileHeader si Signature

printf("Signature: 0x%.8lX\n", ntHeaders.Signature);
printf("FileHeader.Machine: 0x%.4X\n", ntHeaders.FileHeader.Machine);
printf("FileHeader.NumberOfSections: %d\n",
ntHeaders.FileHeader.NumberOfSections);
printf("FileHeader.TimeDateStamp: 0x%.8lX\n",
ntHeaders.FileHeader.TimeDateStamp);

```

```

    printf("FileHeader.PointerToSymbolTable: 0x%.8lX\n",
ntHeaders.FileHeader.PointerToSymbolTable);
    printf("FileHeader.NumberOfSymbols: %lu\n",
ntHeaders.FileHeader.NumberOfSymbols);
    printf("FileHeader.SizeOfOptionalHeader: %d\n",
ntHeaders.FileHeader.SizeOfOptionalHeader);
    printf("FileHeader.Characteristics: 0x%.4X\n\n",
ntHeaders.FileHeader.Characteristics);

    // Afisam OptionalHeader (IMAGE_OPTIONAL_HEADER)

    printf("OptionalHeader.Magic: 0x%.4X\n",
ntHeaders.OptionalHeader.Magic);
    printf("OptionalHeader.MajorLinkerVersion: %d\n",
ntHeaders.OptionalHeader.MajorLinkerVersion);
    printf("OptionalHeader.MinorLinkerVersion: %d\n",
ntHeaders.OptionalHeader.MinorLinkerVersion);
    printf("OptionalHeader.SizeOfCode: %lu\n",
ntHeaders.OptionalHeader.SizeOfCode);
    printf("OptionalHeader.SizeOfInitializedData: %lu\n",
ntHeaders.OptionalHeader.SizeOfInitializedData);
    printf("OptionalHeader.SizeOfUninitializedData: %lu\n",
ntHeaders.OptionalHeader.SizeOfUninitializedData);
    printf("OptionalHeader.AddressOfEntryPoint: 0x%.8lX\n",
ntHeaders.OptionalHeader.AddressOfEntryPoint);
    printf("OptionalHeader.BaseOfCode: 0x%.8lX\n",
ntHeaders.OptionalHeader.BaseOfCode);
    printf("OptionalHeader.BaseOfData: 0x%.8lX\n",
ntHeaders.OptionalHeader.BaseOfData);
    printf("OptionalHeader.ImageBase: 0x%.8lX\n",
ntHeaders.OptionalHeader.ImageBase);
    printf("OptionalHeader.SectionAlignment: %lu\n",
ntHeaders.OptionalHeader.SectionAlignment);
    printf("OptionalHeader.FileAlignment: %lu\n",
ntHeaders.OptionalHeader.FileAlignment);
    printf("OptionalHeader.MajorOperatingSystemVersion: %d\n",
ntHeaders.OptionalHeader.MajorOperatingSystemVersion);
    printf("OptionalHeader.MinorOperatingSystemVersion: %d\n",
ntHeaders.OptionalHeader.MinorOperatingSystemVersion);
    printf("OptionalHeader.MajorImageVersion %d\n",
ntHeaders.OptionalHeader.MajorImageVersion);
    printf("OptionalHeader.MinorImageVersion: %d\n",
ntHeaders.OptionalHeader.MinorImageVersion);
    printf("OptionalHeader.MajorSubsystemVersion: %d\n",
ntHeaders.OptionalHeader.MajorSubsystemVersion);
    printf("OptionalHeader.MinorSubsystemVersion: %d\n",
ntHeaders.OptionalHeader.MinorSubsystemVersion);
    printf("OptionalHeader.Win32VersionValue: %lu\n",
ntHeaders.OptionalHeader.Win32VersionValue);
    printf("OptionalHeader.SizeOfImage: %lu\n",
ntHeaders.OptionalHeader.SizeOfImage);
    printf("OptionalHeader.SizeOfHeaders: %lu\n",
ntHeaders.OptionalHeader.SizeOfHeaders);
    printf("OptionalHeader.CheckSum: 0x%.8lX\n",
ntHeaders.OptionalHeader.CheckSum);
    printf("OptionalHeader.Subsystem: 0x%.4X\n",
ntHeaders.OptionalHeader.Subsystem);

```

```

        printf("OptionalHeader.DllCharacteristics: 0x%.4X\n",
ntHeaders.OptionalHeader.DllCharacteristics);
        printf("OptionalHeader.SizeOfStackReserve: %lu\n",
ntHeaders.OptionalHeader.SizeOfStackReserve);
        printf("OptionalHeader.SizeOfStackCommit: %lu\n",
ntHeaders.OptionalHeader.SizeOfStackCommit);
        printf("OptionalHeader.SizeOfHeapReserve: %lu\n",
ntHeaders.OptionalHeader.SizeOfHeapReserve);
        printf("OptionalHeader.SizeOfHeapCommit: %lu\n",
ntHeaders.OptionalHeader.SizeOfHeapCommit);
        printf("OptionalHeader.LoaderFlags: 0x%.8lX\n",
ntHeaders.OptionalHeader.LoaderFlags);
        printf("OptionalHeader.NumberOfRvaAndSizes: %lu\n\n",
ntHeaders.OptionalHeader.NumberOfRvaAndSizes);

        // Afisam DataDirectory

        for(int i = 0; i < ntHeaders.OptionalHeader.NumberOfRvaAndSizes; i++)
            printf("DataDirectory[%d].VirtualAddress: 0x%.8lX [%lu bytes]\n",
i,
                ntHeaders.OptionalHeader.DataDirectory[i].VirtualAddress,
                ntHeaders.OptionalHeader.DataDirectory[i].Size);

        return 0;
}

```

Program care afiseaza intreaga structura IMAGE_NT_HEADERS (toate headerle PE)

3. Tabelul de sectiuni

Dupa aceste headere (IMAGE_NT_HEADERS), urmeaza tabelul de sectiuni al fisierului PE.

In structura care defineste headerle PE, structura **IMAGE_NT_HEADERS**, campul "NumberOfSections" din cadrul structurii "FileHeader" de tipul IMAGE_FILE_HEADER (devine mai complicat), contine numarul de sectiuni pe care le contine fisierul.

Acest tabel de sectiuni care urmeaza imediat dupa headerle PE contine un vector de structuri de tipul "**IMAGE_SECTION_HEADER**", structura ce defineste o sectiuni. Acest vector are exact "NumberOfSections" sectiuni.

Daca in fisier se vor adauga sau sterge sectiuni, acel "NumberOfSections" va trebui actualizat.

Structura "**IMAGE_SECTION_HEADER**" (40 de octeti) e definita astfel:

```

#define IMAGE_SIZEOF_SHORT_NAME          8

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;

#define IMAGE_SIZEOF_SECTION_HEADER    40

```

Structura IMAGE_SECTION_HEADER

In aceasta structura, gasim urmatoarele campuri:

- **Name:** Numele sectiunii, un sir de maxim 8 caractere. Atentie, sirul nu se termina obligatoriu cu NULL!
- **VirtualSize:** Dimensiunea sectiunii atunci cand este incarcata in memorie
- **VirtualAddress:** Adresa RVA (relativa la ImageBase) la care incepe sectiunea
- **SizeOfRawData:** Dimensiunea sectiunii in fisier, trebuie sa fie un multiplu al campului "fileAlignment" din OptionalHeader
- **PointerToRawData:** Adresa RVA la care incepe sectiunea in fisier
- **PointerToRelocations:** Ar trebui sa fie 0 pentru fisiere executabile, sau o alta valoare in cazul in care la incarcarea in memorie trebuie sa se faca "relocari". Relocarile pot fi necesare pentru DLL-uri. In cazul in care un DLL nu se poate incarca la adresa preferata, exista posibilitatea ca anumite adrese sa fie modificate pentru ca corespunde acestor modificari, acest lucru fiind realizat printr-un tabel de adrese care trebuie relocate in astfel de situatii
- **PointerToLinenumbers:** Informatii de debug care nu se mai folosesc, ar trebui sa aiba valoarea 0
- **NumberOfRelocations:** Daca sunt necesare relocari, aici se va putea gasi numarul de adrese care trebuiesc "relocate"
- **NumberOfLinenumbers:** Informatii de debug care nu se mai folosesc, ar trebui sa aiba valoarea 0

- **Characteristics:** Campul este o masca de biti care specifica mai multe detalii despre sectiune: daca aceasta contine cod, date initializate sau date neinitializate, permisiunile pentru sectiune cand aceasta este incarcata in memorie si multe altele

Cateva exemple de biti care pot fi setati pentru o sectiune:

```
#define IMAGE_SCN_CNT_CODE                0x00000020 // Section contains
code.
#define IMAGE_SCN_CNT_INITIALIZED_DATA    0x00000040 // Section contains
initialized data.
#define IMAGE_SCN_CNT_UNINITIALIZED_DATA  0x00000080 // Section contains
uninitialized data.
#define IMAGE_SCN_MEM_DISCARDABLE         0x02000000 // Section can be
discarded.
#define IMAGE_SCN_MEM_NOT_CACHED          0x04000000 // Section is not
cachable.
#define IMAGE_SCN_MEM_NOT_PAGED           0x08000000 // Section is not
pageable.
#define IMAGE_SCN_MEM_SHARED              0x10000000 // Section is
shareable.
#define IMAGE_SCN_MEM_EXECUTE             0x20000000 // Section is
executable.
#define IMAGE_SCN_MEM_READ                 0x40000000 // Section is
readable.
#define IMAGE_SCN_MEM_WRITE                0x80000000 // Section is
writeable.
```

Caracteristici posibile pentru sectiuni

Pentru a intelege mai usor acest tabel de sectiuni, tabel care incepe imediat dupa headerele PE (IMAGE_NT_HEADERS), am detaliat fiecare sectiune. Se poate observa, ca si in structura, ca primii 8 octeti ai fiecari sectiuni (in tabel), il reprezinta numele.

Culorile reprezinta:

1. Rosu: sectiunea “.text” este sectiunea de cod
2. Negru: sectiunea “.data” reprezinta sectiunea de date initializate
3. Verde: sectiunea “.rsrc” va contine resurse (adesea icon-ul fisierului si versiunea/copyright-ul fisierului, dar poate contine multe alte informatii)
4. Albastru: sectiunea “.reloc” va contine acele “relocari” in cazul in care fisierul nostrul (in exemplu de fata o biblioteca .dll) nu poate fi incarat la adresa specifica

Pe langa aceste sectiuni, pot sa apara si altele:

4. Sectiunea “.bss” – date neinitializate globale
5. Sectiunea “.rdata” – date “read-only” sau alte sectiuni

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000130	06	00	01	00	06	00	01	00	06	00	01	00	00	00	00	00															
00000140	00	40	0D	00	00	10	00	00	25	11	0E	00	03	00	40	01	.@.....															
00000150	00	00	04	00	00	10	00	00	00	00	10	00	00	10	00	00															
00000160	00	00	00	00	10	00	00	00	C0	51	0B	00	B1	A9	00	00															
00000170	74	FB	0B	00	F4	01	00	00	00	70	0C	00	28	05	00	00	t.....															
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
00000190	00	80	0C	00	B0	B0	00	00	B4	59	0C	00	38	00	00	00															
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000001B0	00	00	00	00	00	00	00	00	90	28	08	00	40	00	00	00															
000001C0	00	00	00	00	00	00	00	00	00	10	00	00	FC	0D	00	00															
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000001E0	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00															
000001F0	15	4A	0C	00	00	10	00	00	00	50	0C	00	00	10	00	00	.J.....															
00000200	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60															
00000210	2E	64	61	74	61	00	00	00	F0	0F	00	00	00	60	0C	00	.data.....															
00000220	00	10	00	00	00	60	0C	00	00	00	00	00	00	00	00	00															
00000230	00	00	00	00	40	00	00	C0	2E	72	73	72	63	00	00	00@...															
00000240	28	05	00	00	00	70	0C	00	00	10	00	00	00	70	0C	00	(...p.....															
00000250	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	40@..@															
00000260	2E	72	65	6C	6F	63	00	00	B0	B0	00	00	00	80	0C	00	.reloc.....															
00000270	00	C0	00	00	00	80	0C	00	00	00	00	00	00	00	00	00@..B															
00000280	00	00	00	00	40	00	00	42	00	00	00	00	00	00	00	00@..B															
00000290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000002A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000002B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000002C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000002D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000002E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															
000002F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00															

Tabelul de sectiuni al fisierului "kernel32.dll"

Pentru a citi tabelul de sectiuni trebuie doar sa sarim peste headerule NT si citim "NumberOfSections" * sizeof(IMAGE_SECTION_HEADER) octeti.

```

/*
  Afisarea tabelului de sectiuni al unui fisier PE
  Autor: Ionut Popescu (Nytro) - Romanian Security Team
*/
#include <stdio.h>
#include <windows.h>

int main()
{
    FILE *FD = NULL;
    IMAGE_DOS_HEADER dos;
    IMAGE_NT_HEADERS ntHeaders;
    IMAGE_SECTION_HEADER Section;
    size_t Len = 0;
    unsigned char *pRawSectionTable = NULL;

```

```

// Deschidem fisierul

FD = fopen("C:\\Windows\\system32\\kernel32.dll", "rb");

if(!FD)
{
    puts("Nu am putut deschide fisierul!");
    return 1;
}

// Citim headerul MS-DOS

Len = fread(&dos, 1, sizeof(dos), FD);

if(Len != sizeof(dos))
{
    puts("Nu am putut citi fisierul!");
    return 1;
}

// Sarim la headerele PE

if(fseek(FD, dos.e_lfanew, SEEK_SET) != 0)
{
    printf("Nu am putut sari la headerul PE: %lu", dos.e_lfanew);
    return 1;
}

// Citim IMAGE_NT_HEADER, adica toate headerele NT

Len = fread(&ntHeaders, 1, sizeof(ntHeaders), FD);

if(Len != sizeof(ntHeaders))
{
    puts("Nu am putut citi IMAGE_NT_HEADERS!");
    return 1;
}

// Am citit toate headerele NT, astfel suntem positionati in fisier
// imediat dupa, adica exact la inceputul tabelului de sectiuni
// Astfel alocam memorie pentru tabelul de sectiuni si citim tabelul

pRawSectionTable = (unsigned char
*)malloc(ntHeaders.FileHeader.NumberOfSections *
sizeof(IMAGE_SECTION_HEADER));

if(pRawSectionTable == NULL)
{
    puts("Nu am putut alocati memorie!");
    return 1;
}

// Citim tabelul (NumberOfSections * marimea_unei_structuri)

Len = fread(pRawSectionTable, 1, ntHeaders.FileHeader.NumberOfSections
* sizeof(IMAGE_SECTION_HEADER), FD);

```



```

    if(Len != ntHeaders.FileHeader.NumberOfSections *
sizeof(IMAGE_SECTION_HEADER))
    {
        puts("Nu am putut citi tabelul de sectiuni!");
        return 1;
    }

    // Afisam fiecare sectiune

    for(size_t i = 0; i < ntHeaders.FileHeader.NumberOfSections *
sizeof(IMAGE_SECTION_HEADER); i += sizeof(IMAGE_SECTION_HEADER))
    {
        Section = *(IMAGE_SECTION_HEADER *) (pRawSectionTable + i);

        printf("Section #d\n", i / sizeof(IMAGE_SECTION_HEADER));
        printf("Name: %s\n", Section.Name);
        printf("VirtualSize: %.8lX\n", Section.Misc.VirtualSize);
        printf("VirtualAddress: %.8lX\n", Section.VirtualAddress);
        printf("SizeOfRawData %lu\n", Section.SizeOfRawData);
        printf("PointerToRawData: %.8lX\n", Section.PointerToRawData);
        printf("PointerToRelocations: %.8lX\n",
Section.PointerToRelocations);
        printf("PointerToLinenumbers: %.8lX\n",
Section.PointerToLinenumbers);
        printf("NumberOfRelocations: %d\n", Section.NumberOfRelocations);
        printf("NumberOfLinenumbers: %d\n", Section.NumberOfLinenumbers);
        printf("Characteristics: %.8lX\n\n", Section.Characteristics);
    }

    fclose(FD);

    return 0;
}

```

Program care afiseaza tabelul de sectiuni al unui fisier PE

4. Concluzii

Articolul nu este nici pe departe complet, sunt prezentate doar notiunile de baza, intr-o maniera simpla si pe intelesul tuturor.

Pentru a putea intelege complet aceste notiuni sunt necesare notiuni de baza de C/C++, in special lucrul cu structuri, dar si o cunoastere de ansamblu a modului in care functioneaza un compilator.

Dupa cum se poate observa, nu este nici simplu dar nici foarte complicat sa intelegem cum functioneaza un fisier PE.

Ionut Popescu (Nytro) – 09 August 2013 (RST)