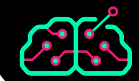
A Arte da Engenharia Reversa

Arquivos Executáveis



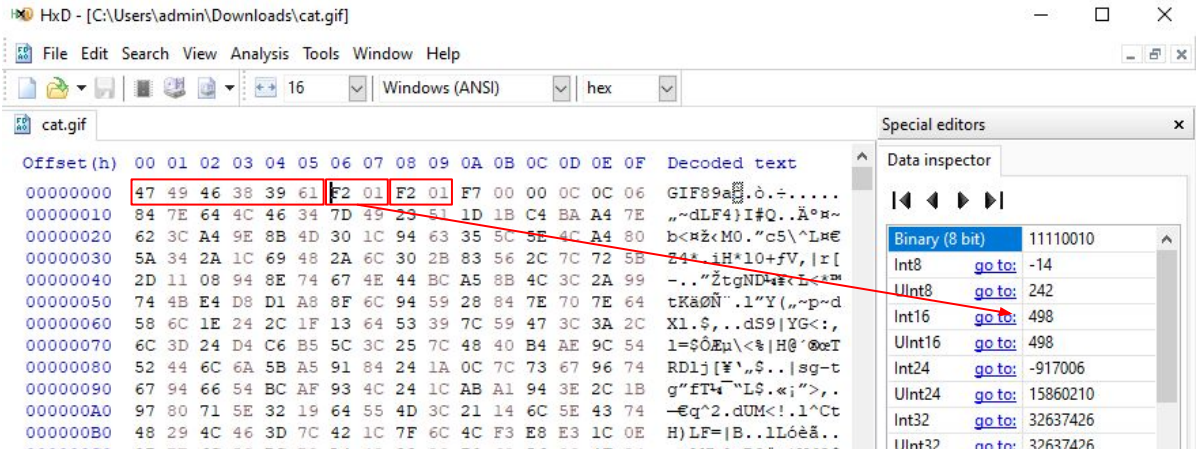
Arquivos Executáveis

- Muitos arquivos possuem um formato definido (por exemplo: PNG, MP3, ZIP, etc).
- O formato PE é o formato para arquivos executáveis (.exe), bibliotecas (.dll) e drivers (.sys) do Windows.
- Já no Linux, o formato para executáveis, bibliotecas (*shared objects / .so*) e drivers/módulos de *kernel* (*kernel objects / .ko*) é o ELF.

O que é um formato?

Table of example GIF image values

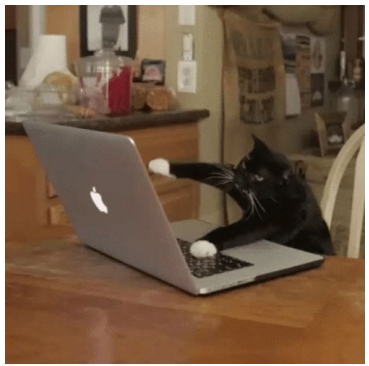
Byte # (hex)	Hexadecimal	Text or value	Meaning
0	47 49 46 38 39 61	GIF89a	Header
			Logical Screen Descriptor
6	03 00	3	Logical screen width
8	05 00	5	Logical screen height
A	F7		GCT follows for 256 colors with resolution 3 × 8 bits/primary, the lowest 3 bits represent the bit depth minus 1, the highest true bit means that the GCT is present
B	00	0	Background color: index #0; #000000 black
C	00	0	Default pixel aspect ratio, 0:0

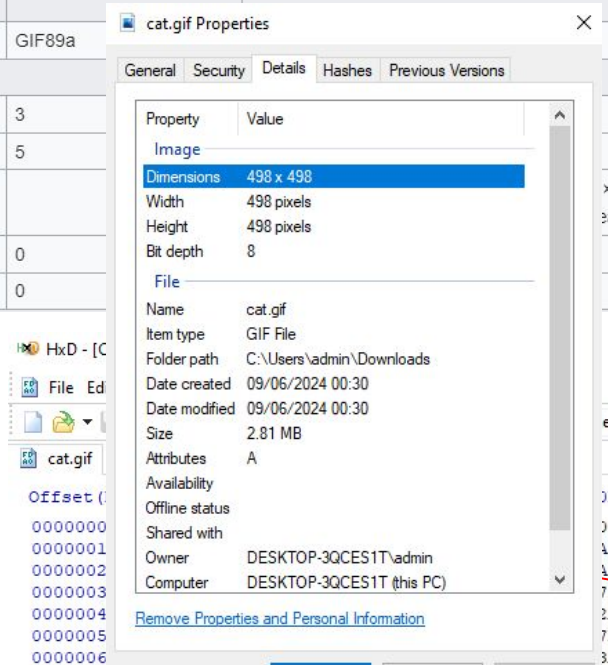


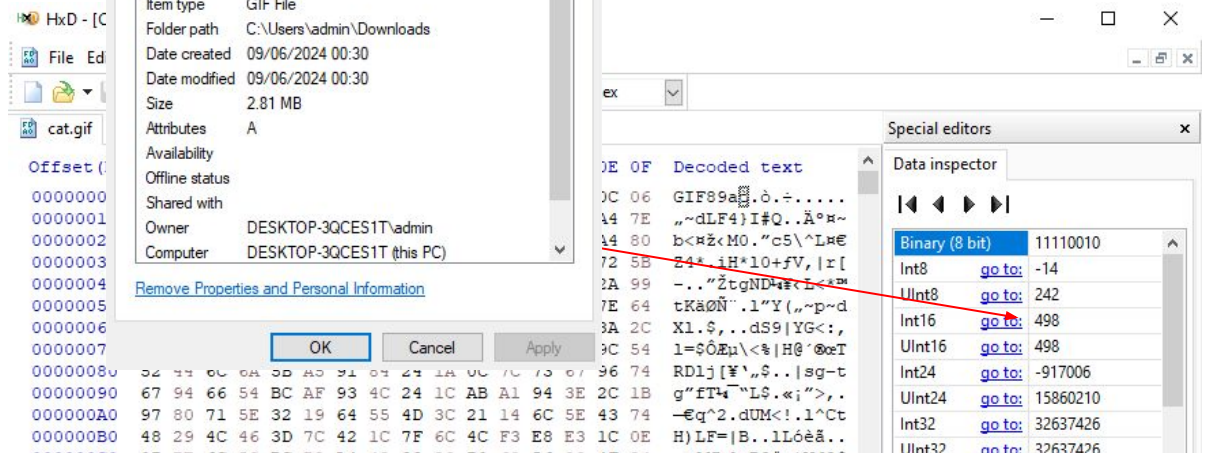
O que é um formato?

Table of example GIF image values

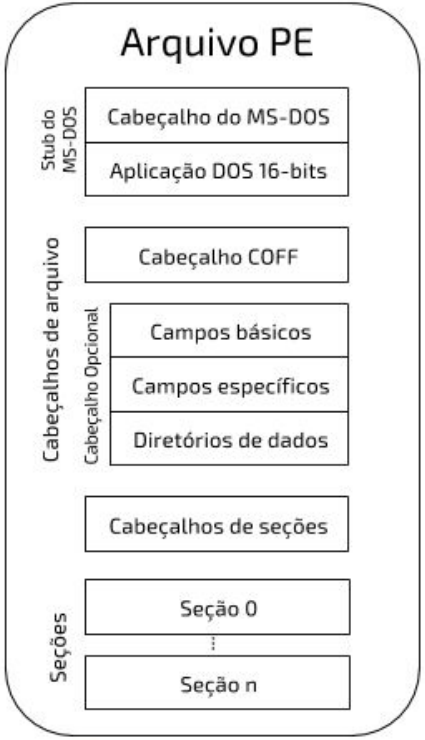
Byte # (hex)	Hexadecimal	Text or value	Meaning
0	47 49 46 38 39 61	GIF89a	
6	03 00	3	
8	05 00	5	
A	F7		x 8 bits/primary, the lowest 3 bits represent eans that the GCT is present
B	00	0	
C	00	0	







O Formato PE



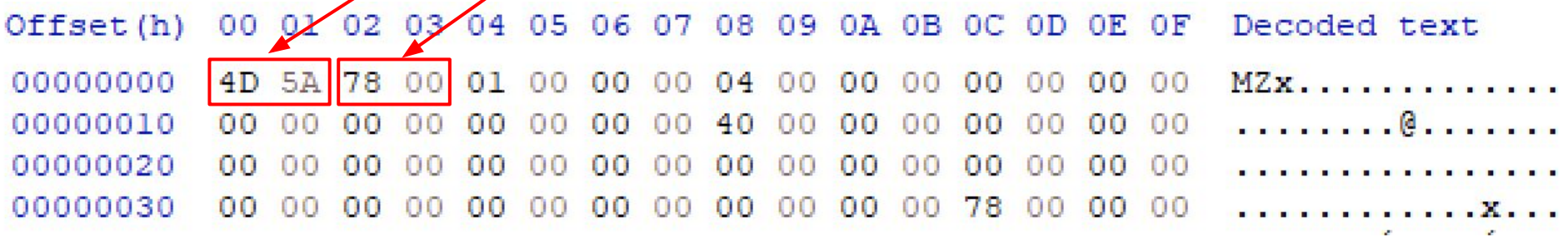
Estrutura de um arquivo PE

Tamanhos dos dados

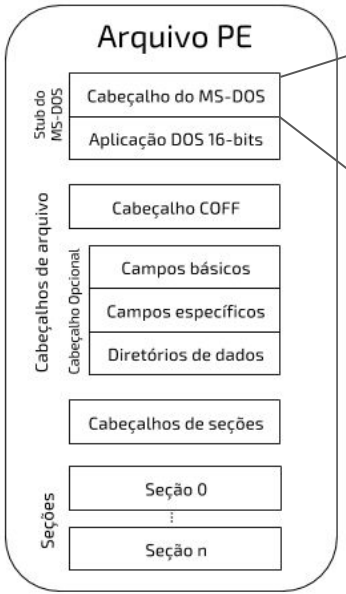
Microsoft	**C (MSVSC)**	**<stdint.h>**	**Tamanho em bytes**
BYTE	unsigned char	uint8_t	1
WORD	unsigned short	uint16_t	2
DWORD	unsigned long	uint32_t	4
QWORD	__int64	uint64_t	8

Cabeçalhos e Campos

```
1 typedef struct {
2     uint16_t e_magic;
3     uint16_t e_cblp;
4     uint16_t e_cp;
5     uint16_t e_crlc;
6     uint16_t e_cparhdr;
7     uint16_t e_minalloc;
8     uint16_t e_maxalloc;
```



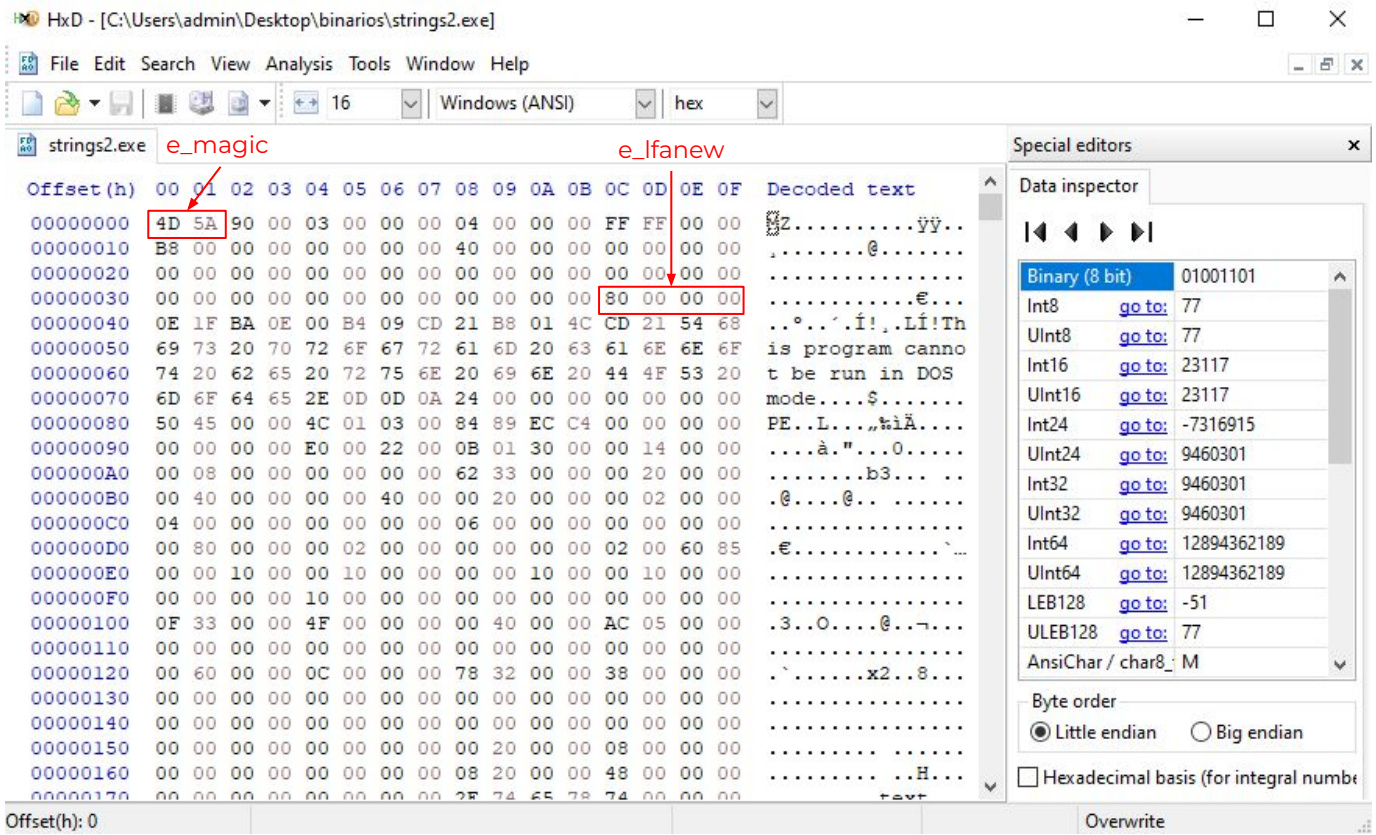
Cabeçalho do DOS



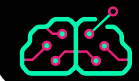
Estrutura de um arquivo PE

```
typedef struct {
    uint16_t e_magic;
    uint16_t e_cblp;
    uint16_t e_cp;
    uint16_t e_crlc;
    uint16_t e_cparhdr;
    uint16_t e_minalloc;
    uint16_t e_maxalloc;
    uint16_t e_ss;
    uint16_t e_sp;
    uint16_t e_csum;
    uint16_t e_ip;
    uint16_t e_cs;
    uint16_t e_lfarlc;
    uint16_t e_ovno;
    uint16_t e_res[4];
    uint16_t e_oemid;
    uint16_t e_oeminfo;
    uint16_t e_res2[10];
    uint32_t e_lfanew;
} IMAGE_DOS_HEADER;
```

Campos - visualização no HxD



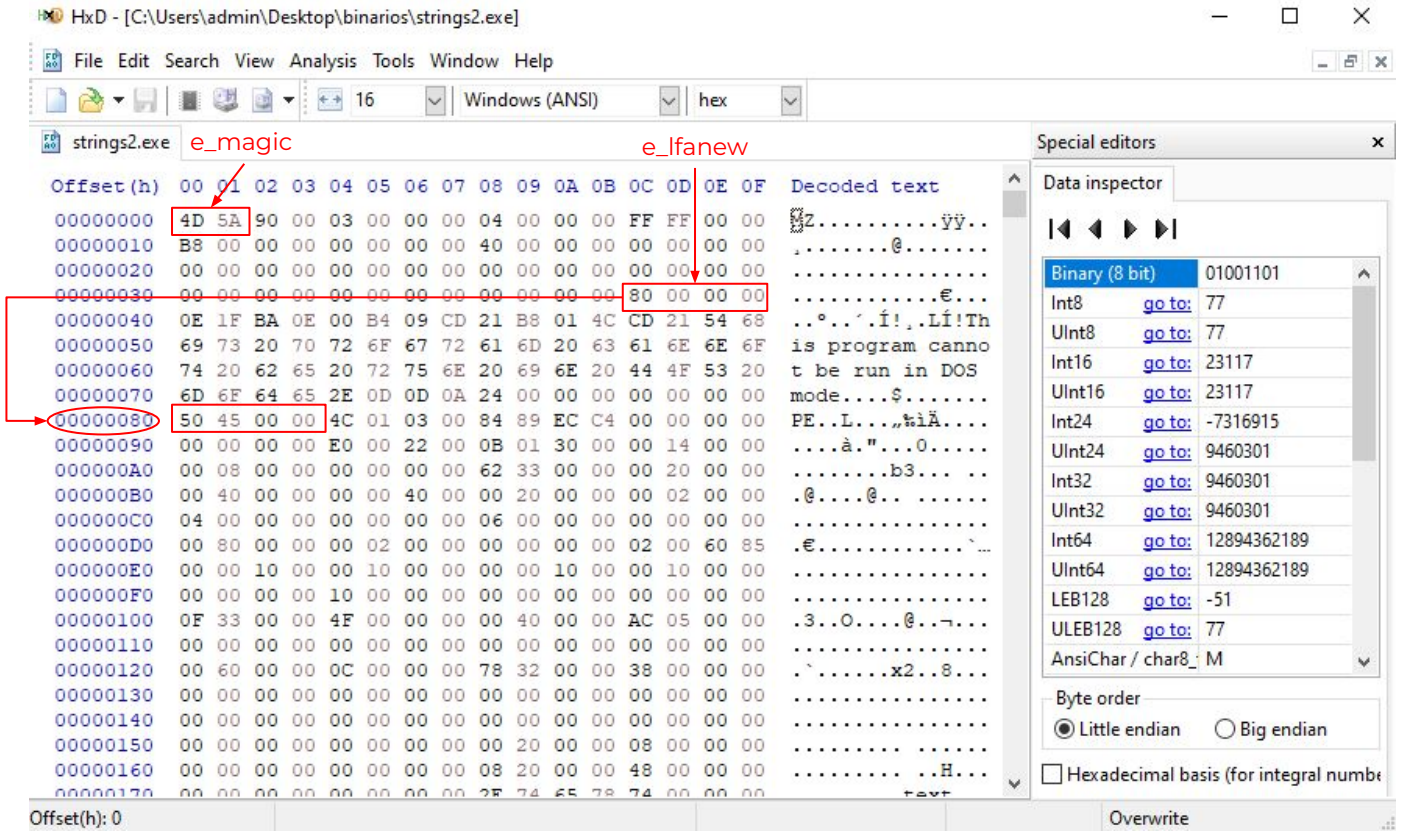

```c
typedef struct {
    uint16_t e_magic;
    uint16_t e_cblp;
    uint16_t e_cp;
    uint16_t e_crlc;
    uint16_t e_cparhdr;
    uint16_t e_minalloc;
    uint16_t e_maxalloc;
    uint16_t e_ss;
    uint16_t e_sp;
    uint16_t e_csum;
    uint16_t e_ip;
    uint16_t e_cs;
    uint16_t e_lfarlc;
    uint16_t e_ovno;
    uint16_t e_res[4];
    uint16_t e_oemid;
    uint16_t e_oeminfo;
    uint16_t e_res2[10];
    uint32_t e_lfanew;
} IMAGE_DOS_HEADER;
```



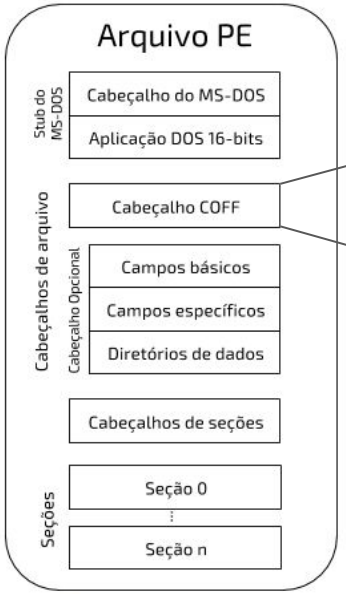
Campos importantes do cabeçalho do DOS

- uint16_t **e_magic**
  - ‘M’ (0x4d) e ‘Z’ (0x5a), iniciais do Mark Zbikowski, designer do formato de executável do MS-DOS, dentre outras coisas na Microsoft.
- uint32_t **e_lfanew**
  - Offset (no arquivo) da assinatura PE.
    - Assinatura PE: 50 45 00 00 ("PE\0\0")

Campos - visualização no HxD



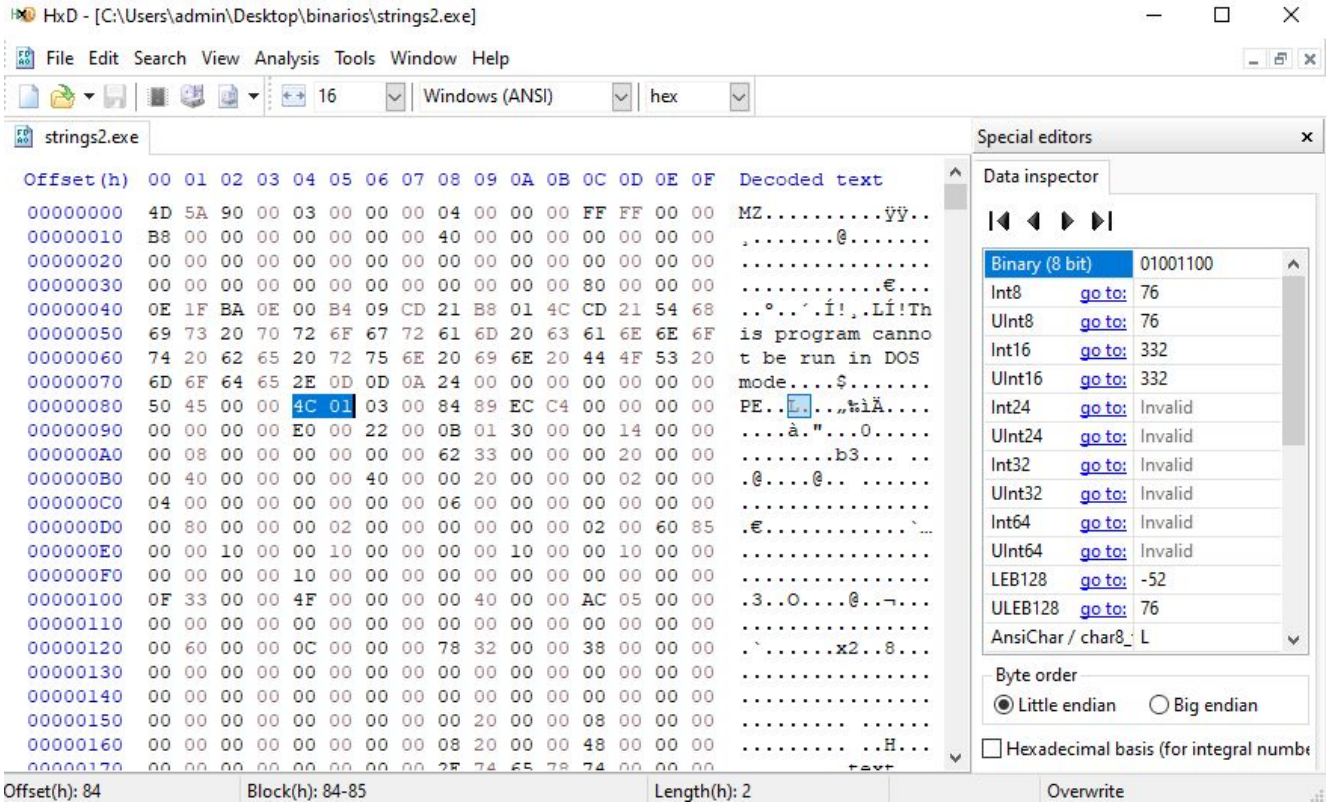
Cabeçalho COFF / (PE) File Header



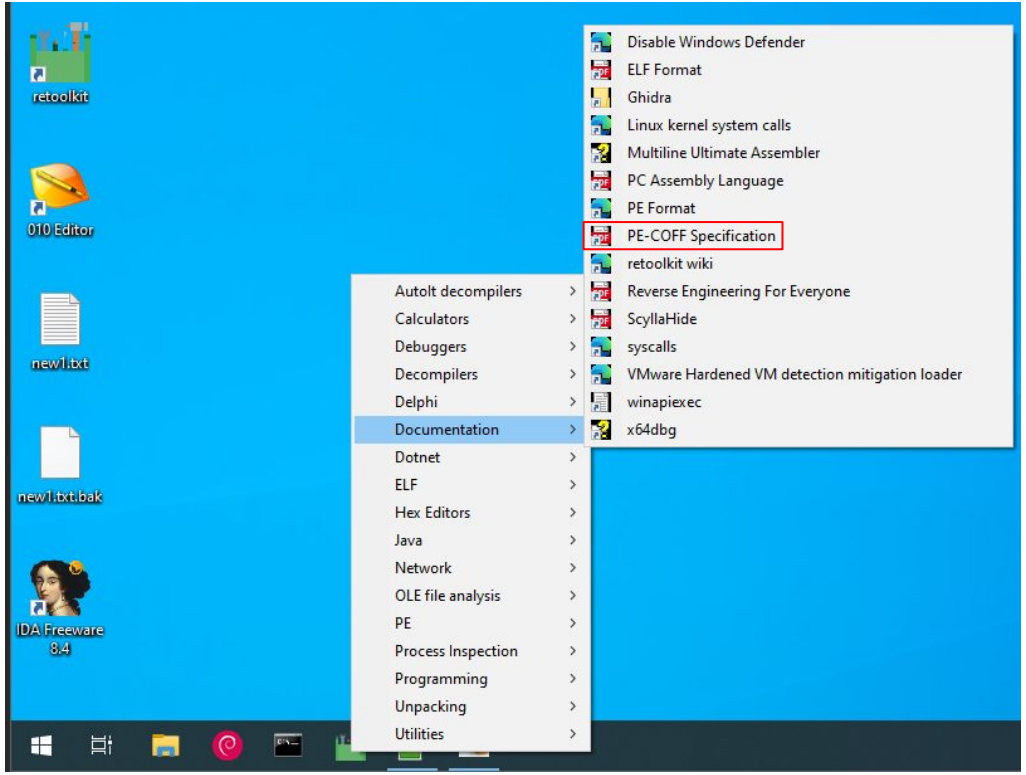
Estrutura de um arquivo PE

```
1 typedef struct {
2     uint16_t Machine;
3     uint16_t NumberOfSections;
4     uint32_t TimeDateStamp;
5     uint32_t PointerToSymbolTable;
6     uint32_t NumberOfSymbols;
7     uint16_t SizeOfOptionalHeader;
8     uint16_t Characteristics;
9 } IMAGE_FILE_HEADER, IMAGE_COFF_HEADER;
```

Campos - visualização no HxD



Visualização da documentação do PE-COFF



Visualização da documentação do PE-COFF

Windows

Microsoft Portable Executable and Common Object File Format Specification

Revision 11 – June 20, 2017

Abstract

This specification describes the structure of executable (image) files and object files under the Windows® family of operating systems. These files are referred to as Portable Executable (PE) and Common Object File Format (COFF) files, respectively.

Note

This document is provided to aid in the development of tools and applications for Windows but is not guaranteed to be a complete specification in all respects. Microsoft reserves the right to alter this document without notice.

This revision of the Microsoft Portable Executable and Common Object File Format Specification replaces all previous revisions of this specification.

For the latest information, see:
http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx

Documentação do PE-COFF - Machine types

3.3.1. Machine Types

The Machine field has one of the following values that specifies its CPU type. An image file can be run only on the specified machine or on a system that emulates the specified machine.

Constant	Value	Description
IMAGE_FILE_MACHINE_UNKNOWN	0x0	The contents of this field are assumed to be applicable to any machine type
IMAGE_FILE_MACHINE_AM33	0x1d3	Matsushita AM33
IMAGE_FILE_MACHINE_AMD64	0x8664	x64
IMAGE_FILE_MACHINE_ARM	0x1c0	ARM little endian
IMAGE_FILE_MACHINE_ARM64	0xaa64	ARM64 little endian
IMAGE_FILE_MACHINE_ARMNT	0x1c4	ARM Thumb-2 little endian
IMAGE_FILE_MACHINE_EBC	0xebc	EFI byte code
IMAGE_FILE_MACHINE_I386	0x14c	Intel 386 or later processors and compatible processors
IMAGE_FILE_MACHINE_IA64	0x200	Intel Itanium processor family
IMAGE_FILE_MACHINE_M32R	0x9041	Mitsubishi M32R little endian
IMAGE_FILE_MACHINE_MIPS16	0x266	MIPS16
IMAGE_FILE_MACHINE_MIPSFPU	0x366	MIPS with FPU
IMAGE_FILE_MACHINE_MIPSFPU16	0x466	MIPS16 with FPU
IMAGE_FILE_MACHINE_POWERPC	0x1f0	Power PC little endian
IMAGE_FILE_MACHINE_POWERPCFP	0x1f1	Power PC with floating point support
IMAGE_FILE_MACHINE_R4000	0x166	MIPS little endian
IMAGE_FILE_MACHINE_RISCV32	0x5032	RISC-V 32-bit address space

Campos interessantes do cabeçalho COFF

- uint16_t **Machine**
  - Define a arquitetura do binário. Valores comuns incluem 0x14c (32-bits) e 0x8664 (64-bits).
- uint16_t **NumberOfSections**
- uint32_t **TimeDateStamp**
  - Segundos desde 1 de Janeiro de 1970 às 00h (Epoch time), quando o binário foi compilado.

Campos importantes do cabeçalho COFF

- uint16_t **Characteristics**
  - Máscara de bits com características do binário. Valores comuns:

Bit	Nome	Comentários
2	IMAGE_FILE_EXECUTABLE_IMAGE	Obrigatório para arquivos executáveis
9	IMAGE_FILE_32BIT_MACHINE	Arquivo de 32-bits
14	IMAGE_FILE_DLL	O arquivo é uma DLL

Lab 06

Identificando os cabeçalhos do PE

Lab 06 - Identificando os cabeçalhos do PE

1. Abra o programa *strings1.exe* no HxD (ou 010 Editor*) e responda:

a) Qual o offset no arquivo da assinatura PE?
*DICA: No offset 0x3c há o campo **uint32_t e_lfanew**, que contém o offset da assinatura PE.*

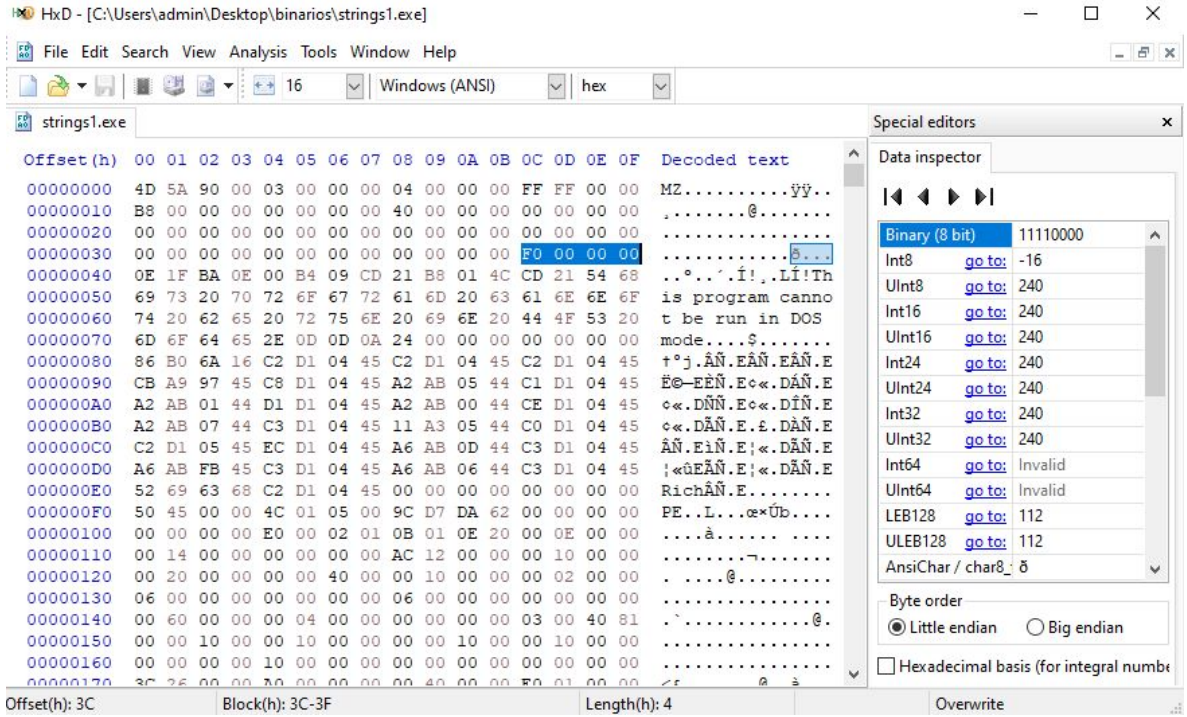
b) Este é um executável de 32 ou 64-bits?
c) Quantas seções este executável possui?
d) Quando ele foi compilado?

* O 010 Editor não está instalado porque a versão de avaliação funciona por 30 dias. Para instalá-lo, basta abrir um Prompt de Comando ou PowerShell e comandar:

```
scoop install 010editor
```

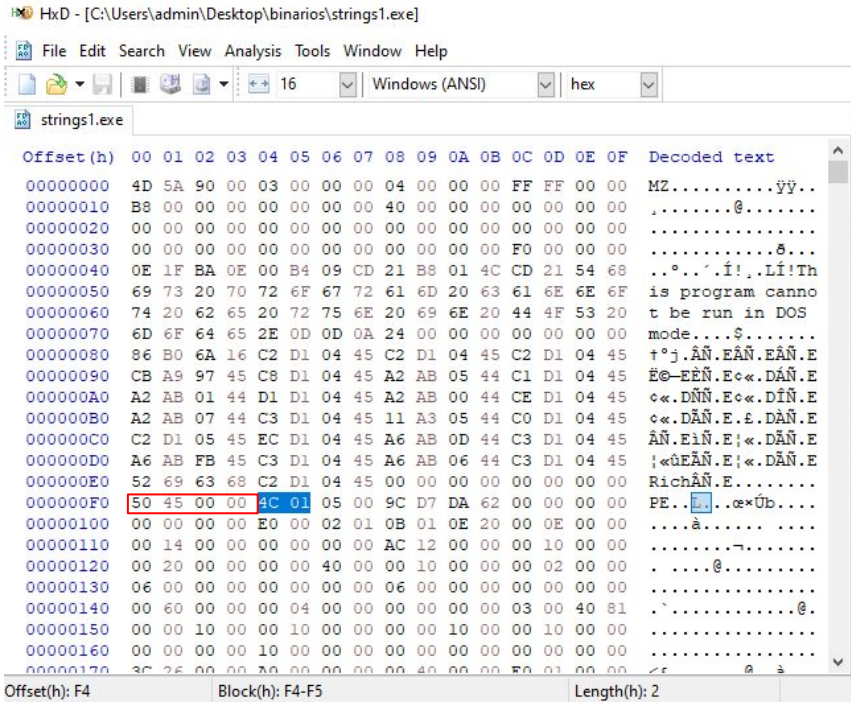
Lab 06 - Respostas

1. a) Qual o offset no arquivo da assinatura PE?



Lab 06 - Respostas

1. b) Este é um executável de 32 ou 64-bits?



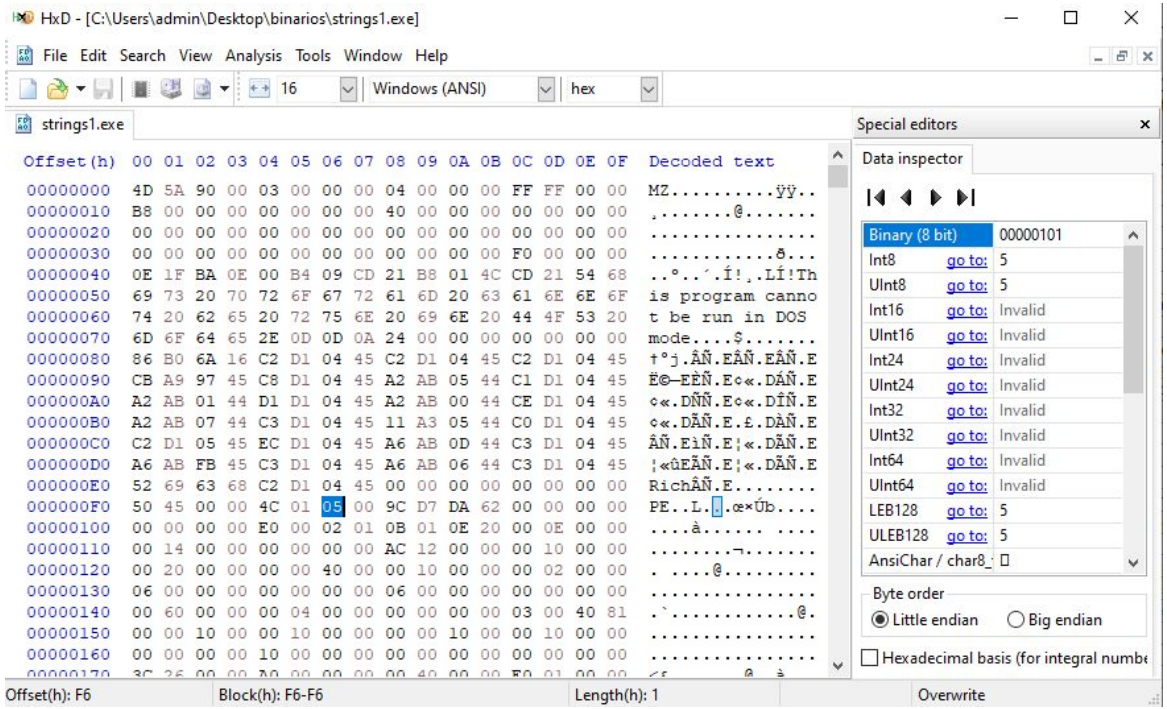
3.3.1. Machine Types

The Machine field has one of the following values that specifies its CPU type. An image file can be run only on the specified machine or on a system that emulates the specified machine.

Constant	Value	Description
IMAGE_FILE_MACHINE_UNKNOWN	0x0	The contents of this field are assumed to be applicable to any machine type
IMAGE_FILE_MACHINE_AM33	0x1d3	Matsushita AM33
IMAGE_FILE_MACHINE_AMD64	0x8664	x64
IMAGE_FILE_MACHINE_ARM	0x1c0	ARM little endian
IMAGE_FILE_MACHINE_ARM64	0xaa64	ARM64 little endian
IMAGE_FILE_MACHINE_ARMNT	0x1c4	ARM Thumb-2 little endian
IMAGE_FILE_MACHINE_EBC	0xebc	EFI byte code
IMAGE_FILE_MACHINE_I386	0x14c	Intel 386 or later processors and compatible processors
IMAGE_FILE_MACHINE_IA64	0x200	Intel Itanium processor family
IMAGE_FILE_MACHINE_M32R	0x9041	Mitsubishi M32R little endian
IMAGE_FILE_MACHINE_MIPS16	0x266	MIPS16
IMAGE_FILE_MACHINE_MIPSFPU	0x366	MIPS with FPU
IMAGE_FILE_MACHINE_MIPSFPU16	0x466	MIPS16 with FPU
IMAGE_FILE_MACHINE_POWERPC	0x1f0	Power PC little endian
IMAGE_FILE_MACHINE_POWERPCFP	0x1f1	Power PC with floating point support
IMAGE_FILE_MACHINE_R4000	0x166	MIPS little endian
IMAGE_FILE_MACHINE_RISCV32	0x5032	RISC-V 32-bit address space

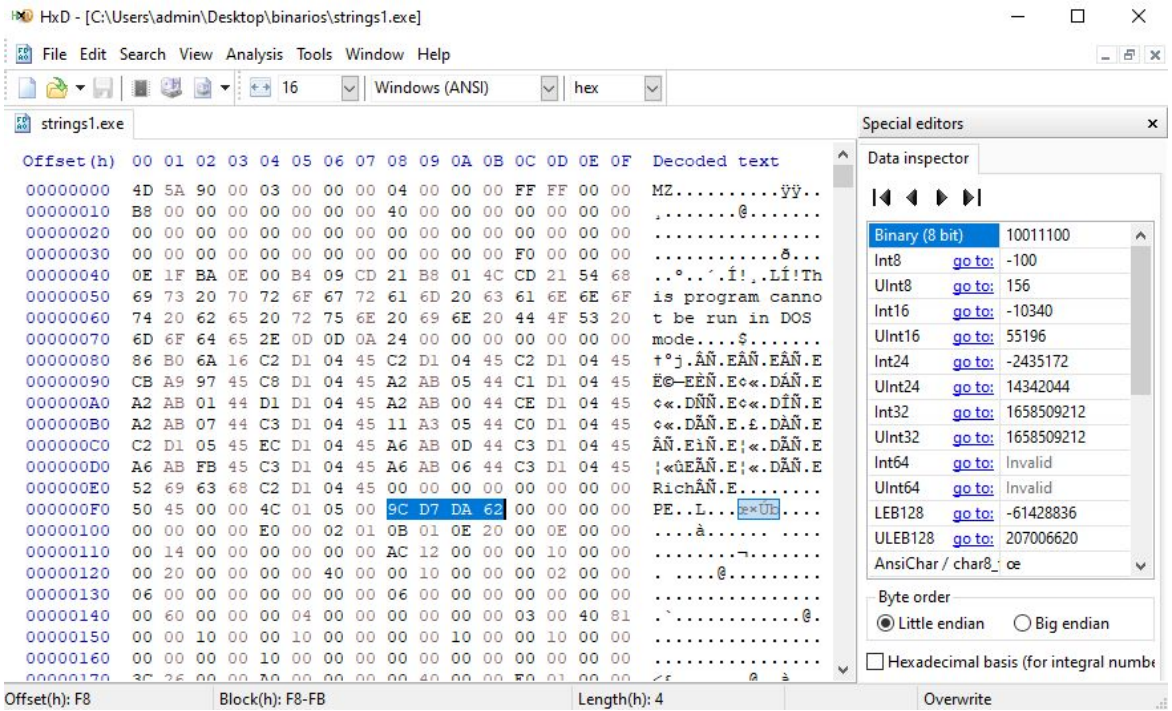
Lab 06 - Respostas

1. c) Quantas seções este executável possui?



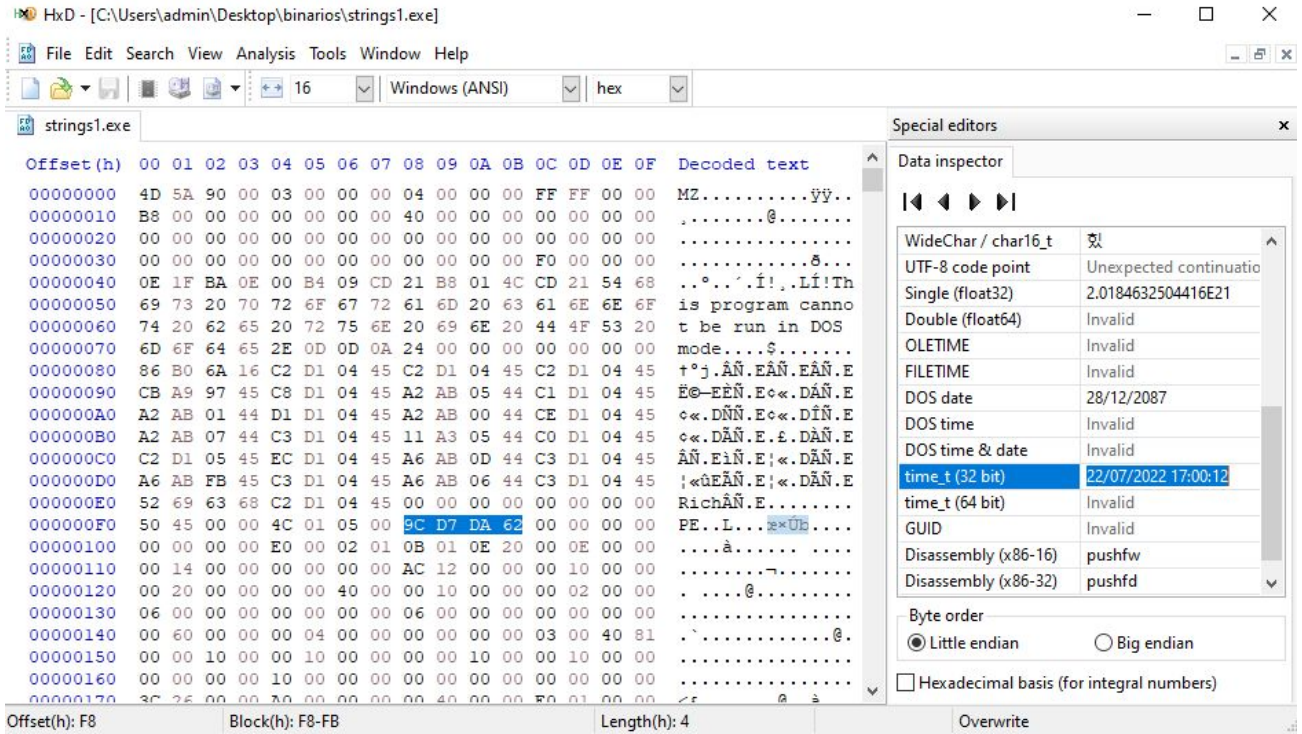
Lab 06 - Respostas

1. d) Quando ele foi compilado?

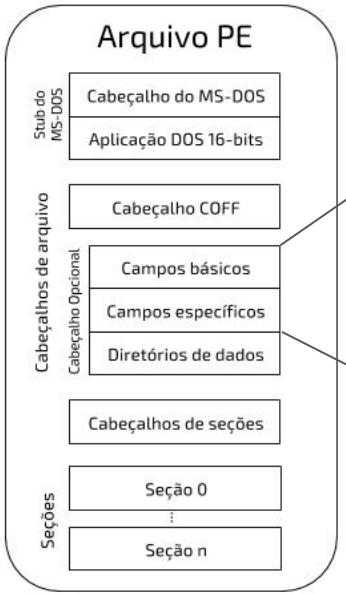


Lab 06 - Respostas

1. d) Quando ele foi compilado?



Cabeçalho Opcional

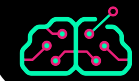


Estrutura de um arquivo PE

```
typedef struct {
    uint16_t Magic;
    uint8_t MajorLinkerVersion;
    uint8_t MinorLinkerVersion;
    uint32_t SizeOfCode;
    uint32_t SizeOfInitializedData;
    uint32_t SizeOfUninitializedData;
    uint32_t AddressOfEntryPoint;
    uint32_t BaseOfCode;
    uint32_t BaseOfData;
    uint32_t ImageBase;
    uint32_t SectionAlignment;
    uint32_t FileAlignment;
    uint16_t MajorOperatingSystemVersion;
    uint16_t MinorOperatingSystemVersion;
    uint16_t MajorImageVersion;
    uint16_t MinorImageVersion;
    uint16_t MajorSubsystemVersion;
    uint16_t MinorSubsystemVersion;
    uint32_t Reserved1;
    uint32_t SizeOfImage;
    uint32_t SizeOfHeaders;
    uint32_t CheckSum;
    uint16_t Subsystem;
    uint16_t DllCharacteristics;
    uint32_t SizeOfStackReserve;
    uint32_t SizeOfStackCommit;
    uint32_t SizeOfHeapReserve;
    uint32_t SizeOfHeapCommit;
    uint32_t LoaderFlags;
    uint32_t NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[MAX_DIRECTORIES];
} IMAGE_OPTIONAL_HEADER_32;
```

Campos importantes do cabeçalho Opcional

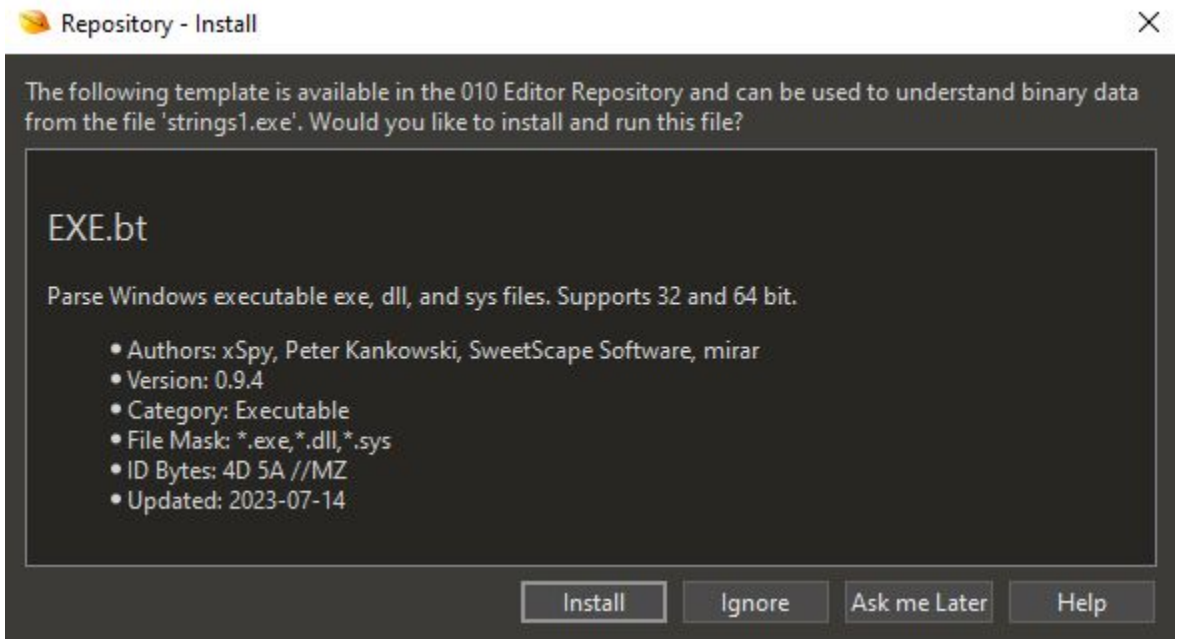
- uint16_t **Magic**
  - Define a arquitetura do binário. Valores comuns incluem 0x10b (32-bits) e 0x20b (64-bits).
- uint32_t **AddressOfEntrypoint**
  - Endereço do *entrypoint* (EP), onde a execução começa.* É opcional para DLLs.



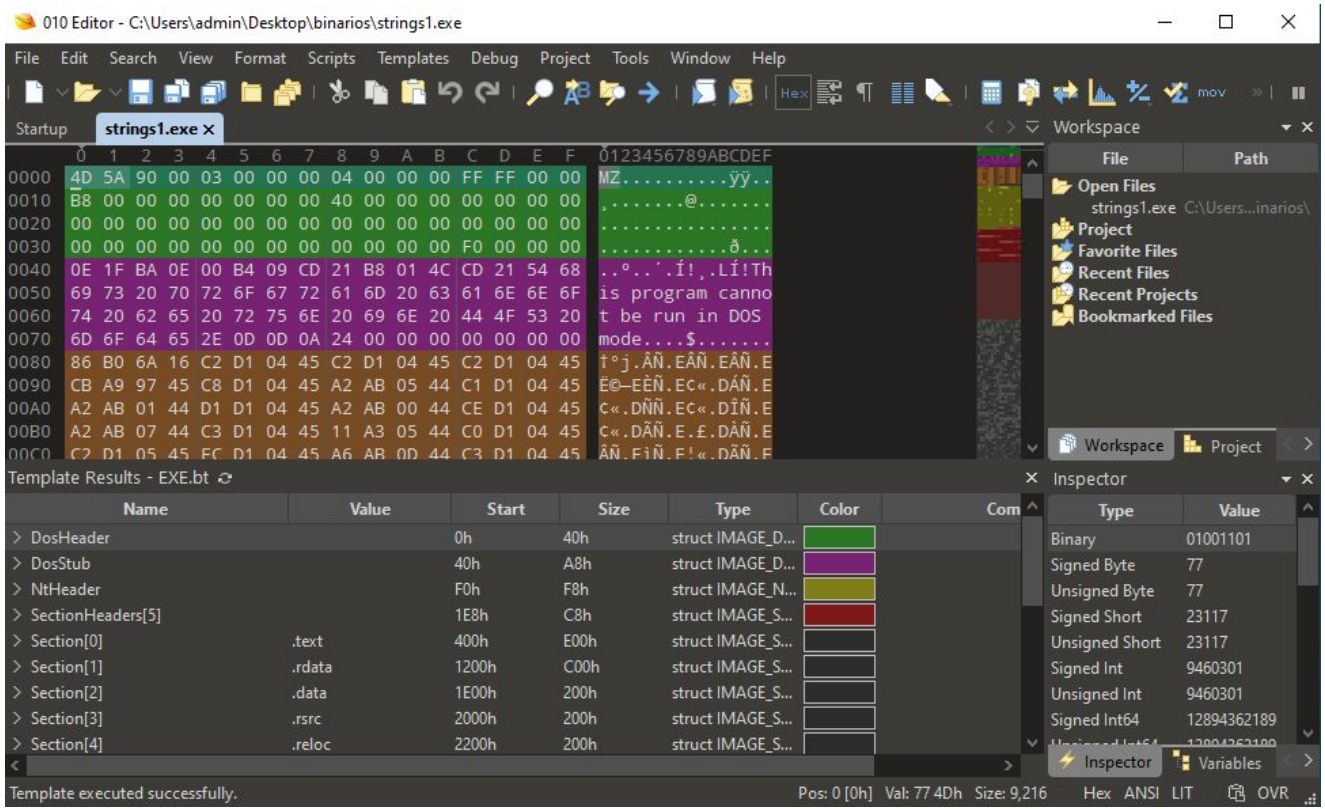
Campos importantes do cabeçalho Opcional

- uint32_t / uint64_t **ImageBase**
  - Endereço virtual do primeiro byte da imagem (cópia do arquivo em memória). Para executáveis em geral, o padrão é 0x400000, com exceção do Windows CE, que usa 0x10000. Já para DLLs, o padrão é 0x10000000.

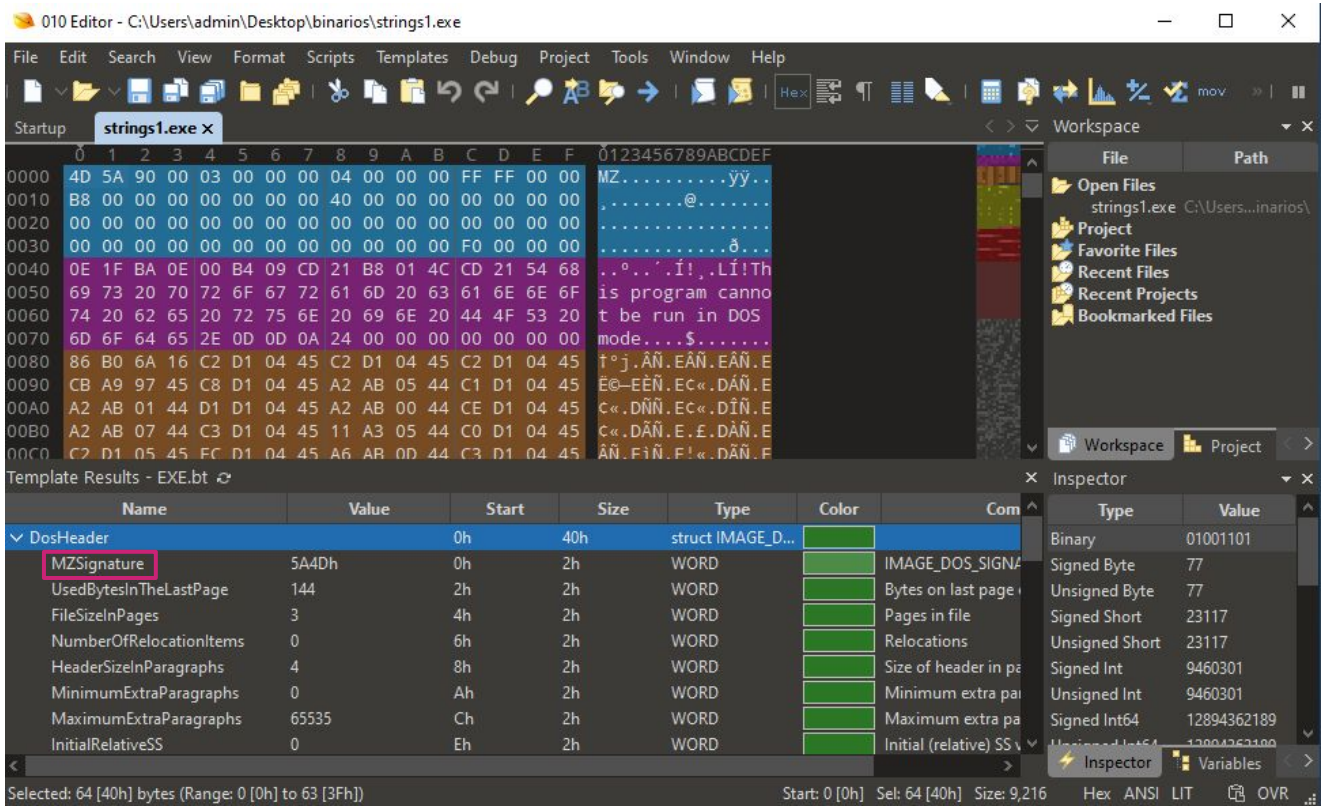
Prática com 010 Editor



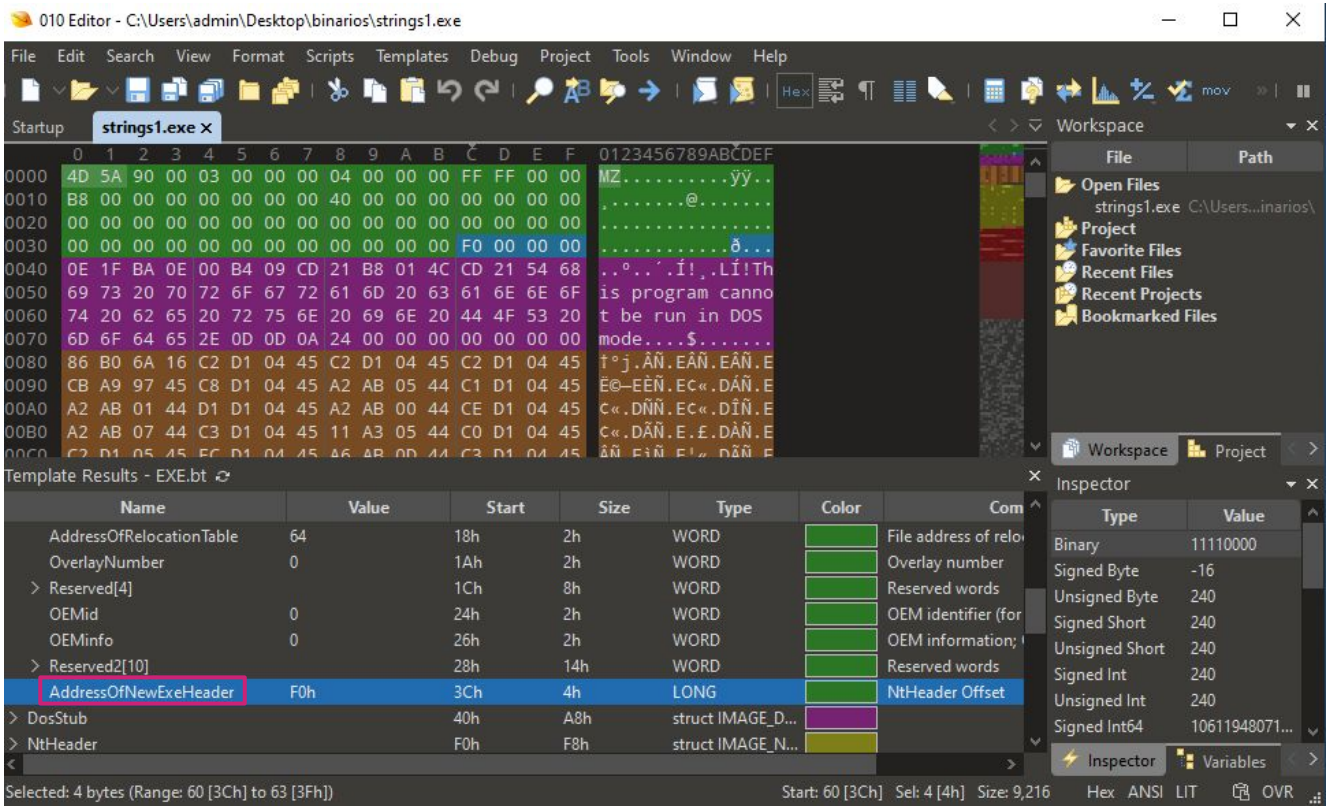
Visualizando PE com 010 Editor



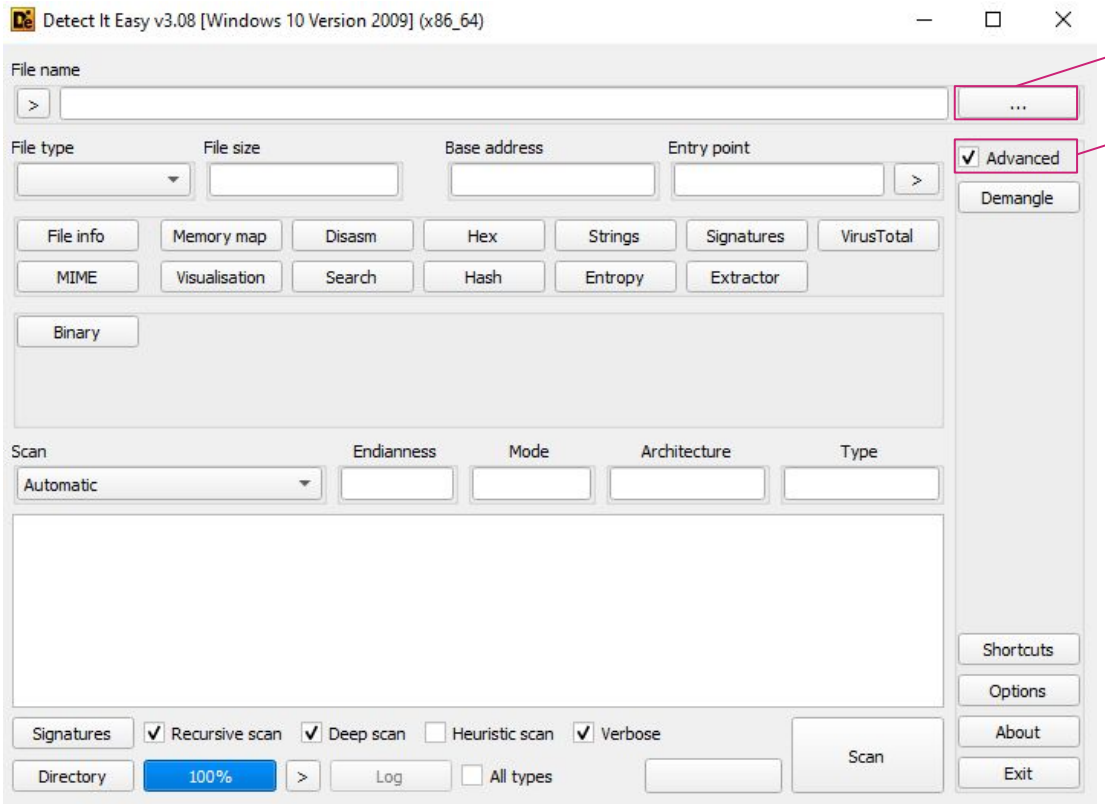
Visualizando PE com 010 Editor



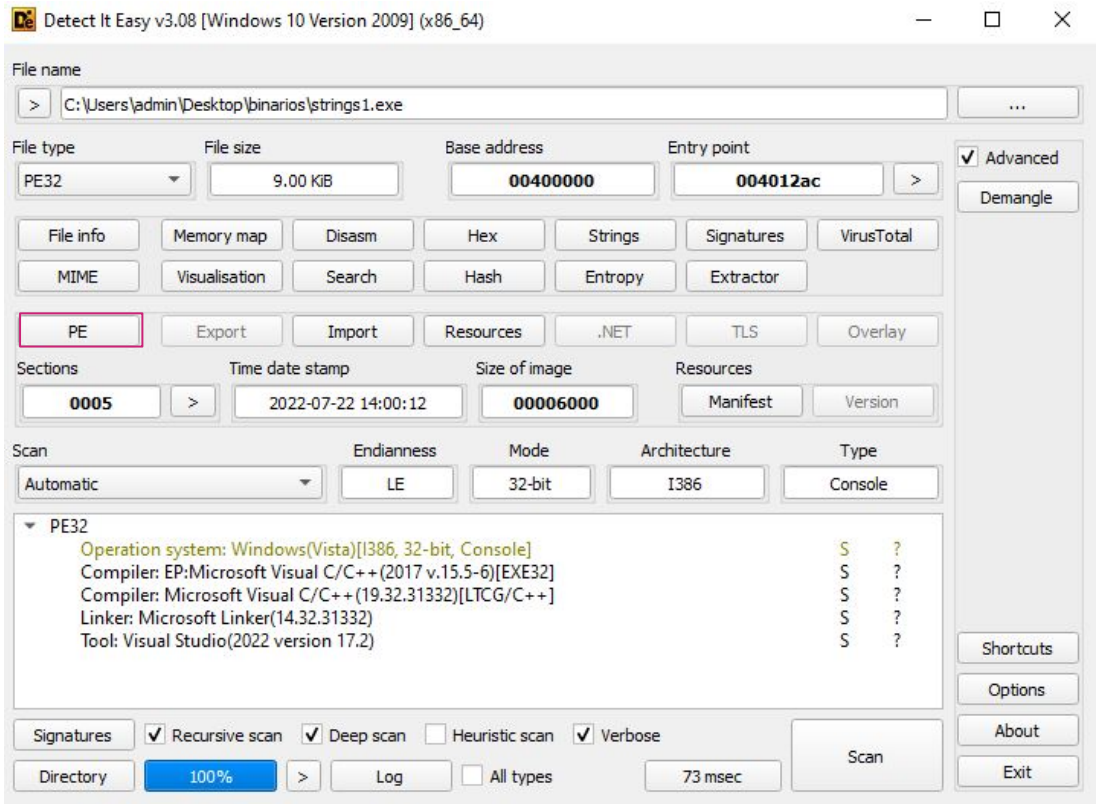
Visualizando PE com 010 Editor



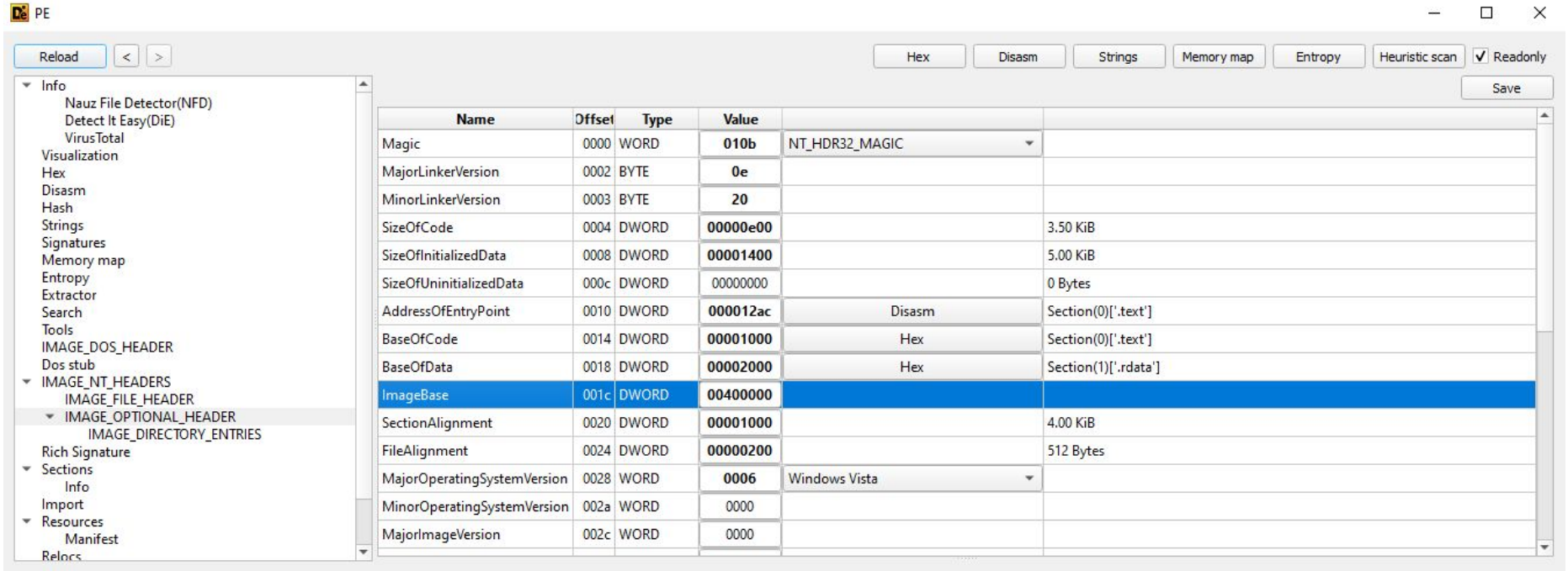
Visualizando PE com Detect It Easy (DIE)



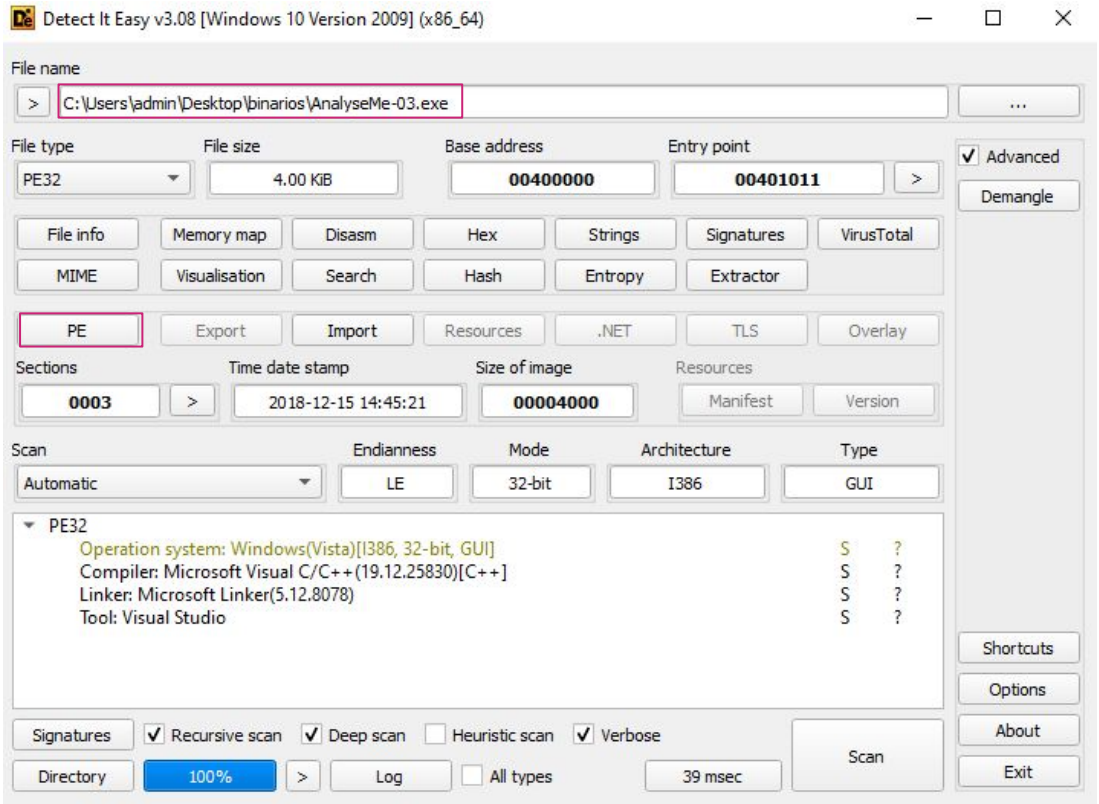
Visualizando PE com Detect It Easy (DIE)



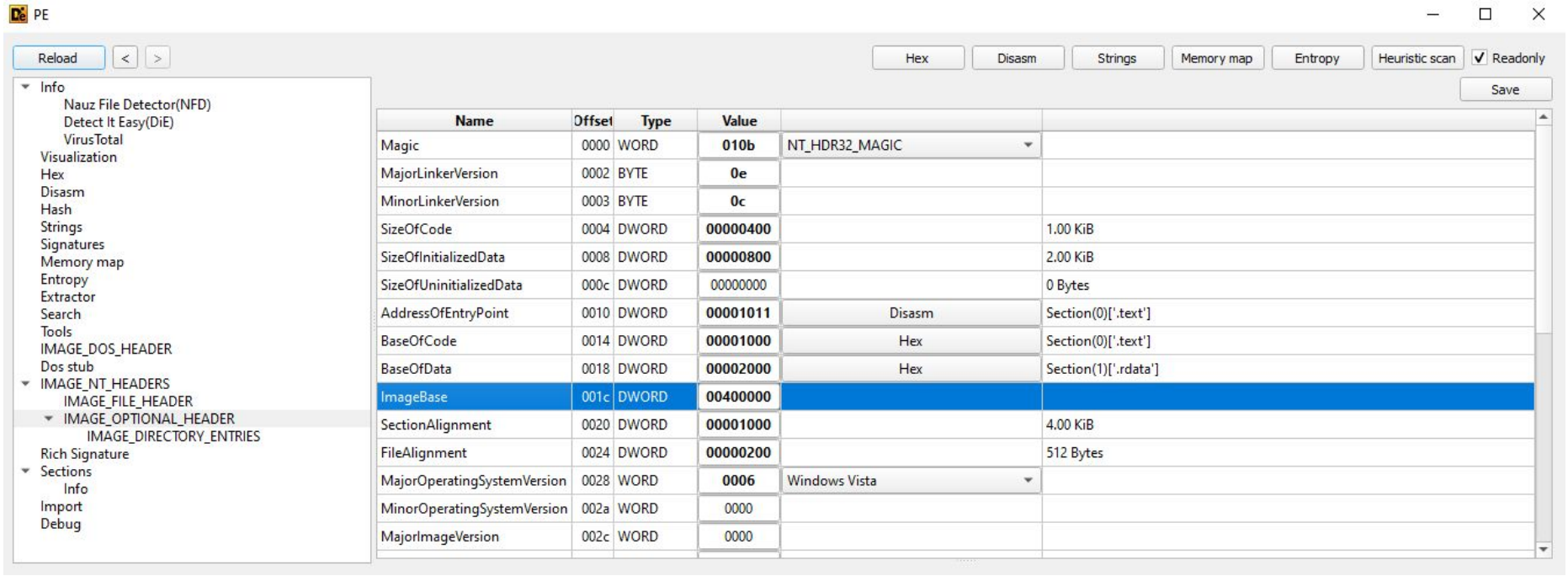
Visualizando PE com Detect It Easy (DIE)

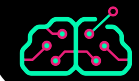


Visualizando PE com Detect It Easy (DIE)



Visualizando PE com Detect It Easy (DIE)

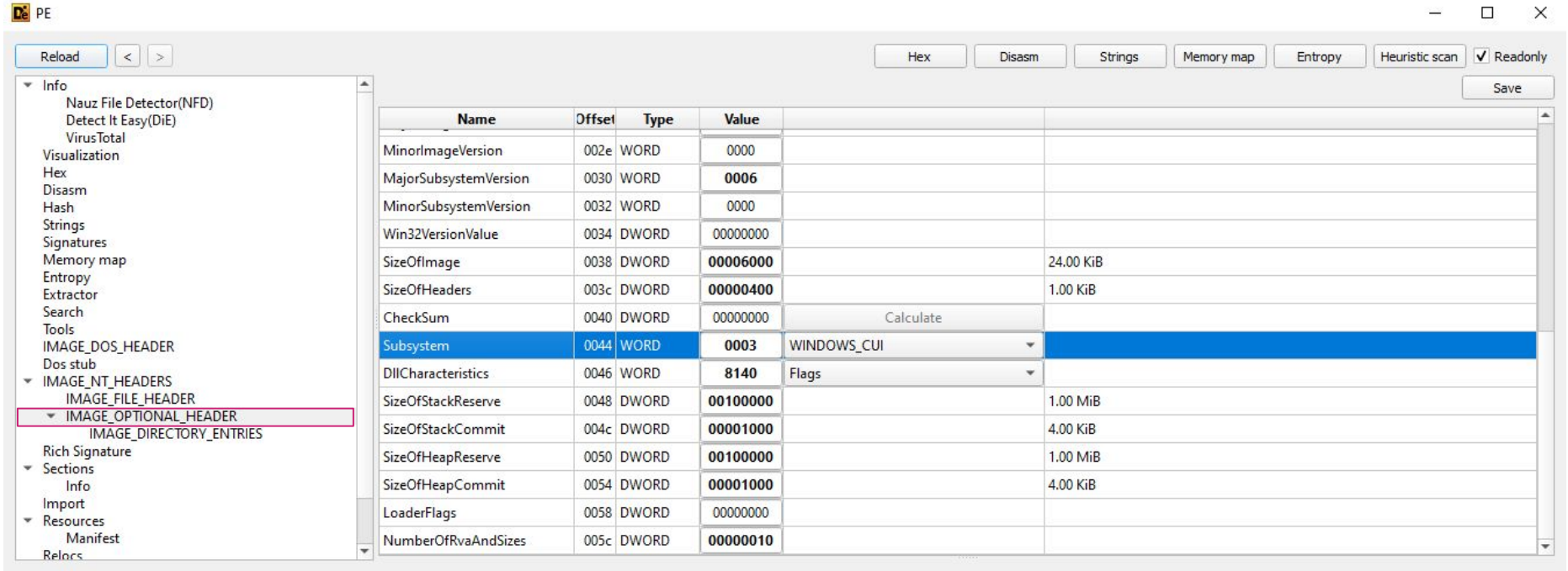




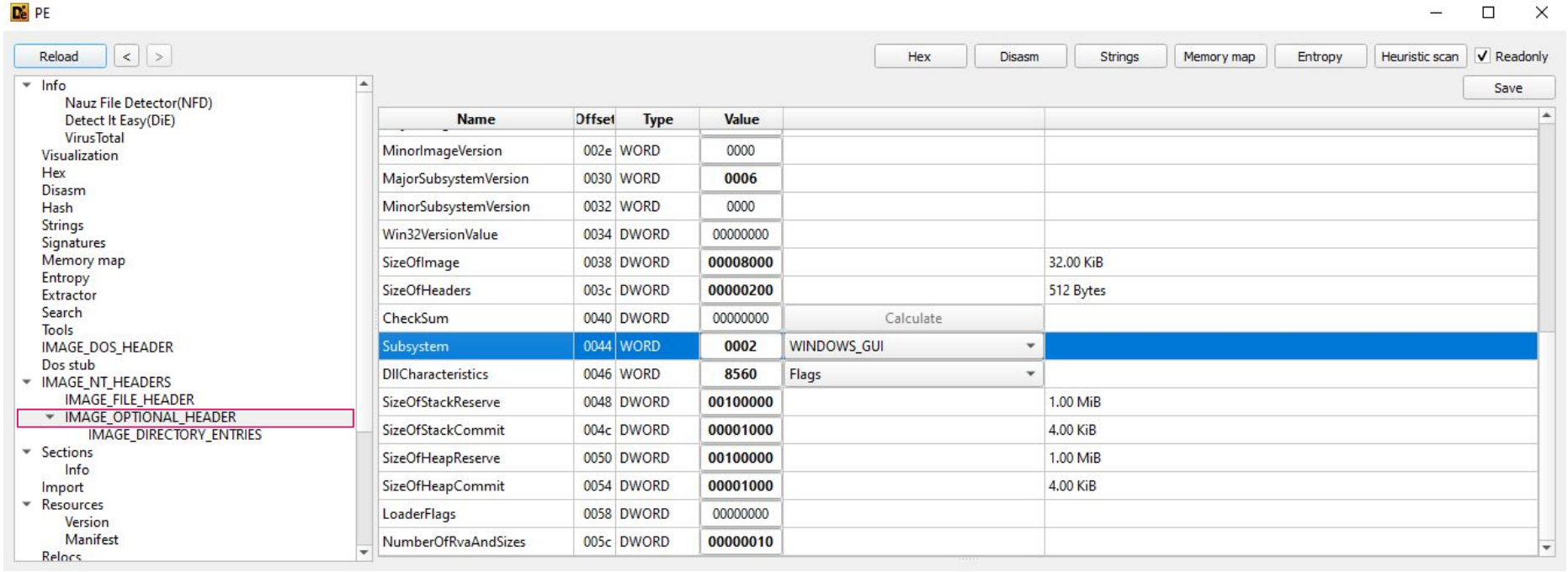
Campos importantes do cabeçalho Opcional

- uint16_t **Subsystem**
  - Subsistema para o qual o binário foi compilado. Os valores mais comuns são 2 (GUI), 3 (CLI) e 1 (Native).
- uint16_t **DllCharacteristics**
  - Máscara de bits com características do binário. Bits importantes incluem o 6 (ASLR) e 8 (DEP/NX).

Strings1.exe no DIE



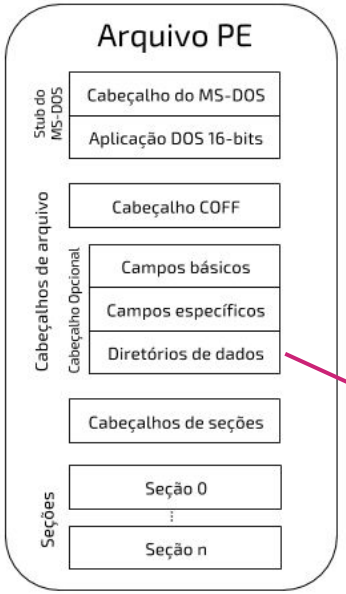
Strings2.exe no DIE



DllCharacteristics

Constante	**Valor**
IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE	0x40
IMAGE_DLLCHARACTERISTICS_NX_COMPAT	0x100

Cabeçalho Opcional
Diretório de Dados



Estrutura de um arquivo PE

```
typedef struct {
    uint16_t Magic;
    uint8_t MajorLinkerVersion;
    uint8_t MinorLinkerVersion;
    uint32_t SizeOfCode;
    uint32_t SizeOfInitializedData;
    uint32_t SizeOfUninitializedData;
    uint32_t AddressOfEntryPoint;
    uint32_t BaseOfCode;
    uint32_t BaseOfData;
    uint32_t ImageBase;
    uint32_t SectionAlignment;
    uint32_t FileAlignment;
    uint16_t MajorOperatingSystemVersion;
    uint16_t MinorOperatingSystemVersion;
    uint16_t MajorImageVersion;
    uint16_t MinorImageVersion;
    uint16_t MajorSubsystemVersion;
    uint16_t MinorSubsystemVersion;
    uint32_t Reserved1;
    uint32_t SizeOfImage;
    uint32_t SizeOfHeaders;
    uint32_t CheckSum;
    uint16_t Subsystem;
    uint16_t DllCharacteristics;
    uint32_t SizeOfStackReserve;
    uint32_t SizeOfStackCommit;
    uint32_t SizeOfHeapReserve;
    uint32_t SizeOfHeapCommit;
    uint32_t LoaderFlags;
    uint32_t NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[MAX_DIRECTORIES];
} IMAGE_OPTIONAL_HEADER_32;
```

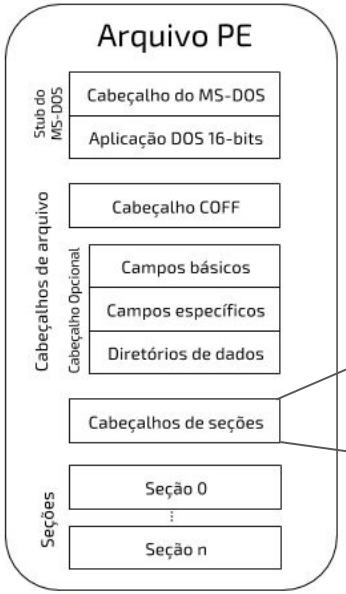
Diretórios de dados

- 16 diretórios ao todo, mas nem todos precisam estar preenchidos.
- Export Table
  - Aponta para a tabela de *exports*, funções exportadas pelo binário.
- Import Table
  - Aponta para a tabela de *imports*, funções importadas (de DLLs) pelo binário.

Diretórios de dados

- Resource Table
  - Aponta para uma estrutura de árvore que armazena os *resources* (ícones, janelas, etc) de um binário.

Cabeçalhos de Seções

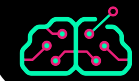


Estrutura de um arquivo PE

```
typedef struct {
    uint8_t Name[SECTION_NAME_SIZE];
    uint32_t VirtualSize;
    uint32_t VirtualAddress;
    uint32_t SizeOfRawData;
    uint32_t PointerToRawData;
    uint32_t PointerToRelocations;
    uint32_t PointerToLinenumbers; // descontinuado
    uint16_t NumberOfRelocations;
    uint16_t NumberOfLinenumbers;  // descontinuado
    uint32_t Characteristics;
} IMAGE_SECTION_HEADER;
```

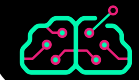
Seções

- São áreas do binário PE que são mapeadas em memória com permissões específicas
- Contém código ou dados
- Uma seção de código, por exemplo, é mapeada em páginas de memória marcadas com permissão de execução



Campos importantes dos cabeçalhos das Seções

- uint8_t **Name[8]**
  - Nome da seção (ex.: .text, .data, etc). Não termina com null.
- uint32_t **VirtualSize**
  - Tamanho em bytes da seção mapeada em memória.
- uint32_t **VirtualAddress**
  - Endereço relativo à base da imagem (ImageBase do cabeçalho opcional) de onde a seção começa em memória.



Campos importantes dos cabeçalhos das Seções

- uint32_t **SizeOfRawData**
  - Tamanho em bytes da seção no arquivo.
- uint32_t **PointerToRawData**
  - Offset no arquivo do início da seção.
- uint32_t **Characteristics**
  - Máscara de bits com características da seção.

uint32_t **Characteristics**

Bit	Nome da flag	Descrição
5	IMAGE_SCN_CNT_CODE	A seção contém código executável
6	IMAGE_SCN_CNT_INITIALIZED_DATA	A seção contém dados inicializados
7	IMAGE_SCN_CNT_UNINITIALIZED_ DATA	A seção contém dados não inicializados
29	IMAGE_SCN_MEM_EXECUTE	Terá permissão de execução
30	IMAGE_SCN_MEM_READ	Terá permissão de leitura
31	IMAGE_SCN_MEM_WRITE	Terá permissão de escrita

Principais seções no PE

- .text ou CODE
  - Contém o código do programa. Leitura e execução.
- .data ou DATA
  - Dados inicializados. Leitura e escrita.
- .rdata
  - Dados inicializados. Somente leitura.

Seções no PE - Exemplos

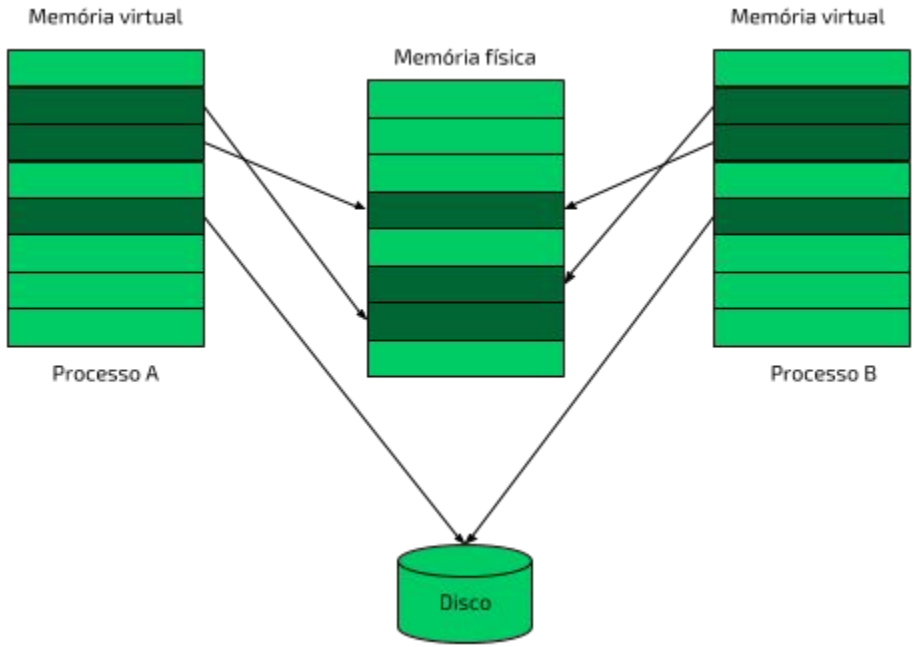
- Caso 1

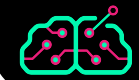
```
char nome[] = "menteb";
// É possível fazer?
nome[0] = 'n';
```

- Caso 2

```
const char *nome = "mente";
// É possível fazer?
nome[0] = 'n';
```

Endereçamento





VA x RVA x File Offset

- **Virtual Address (VA)**, é o endereço **virtual** em memória de determinada instrução ou dado
- **Relative Virtual Address (RVA)**, é um valor que, se somado ao VA base, resulta no VA de determinada instrução ou dado
- **File offset** é a posição de determinada instrução ou dado a partir do início do arquivo

Exemplo:
VA: 0x4023A4

RVA para offset em arquivo

$$S_{RVA} = S \mid S_{VirtualAddress} \leqslant RVA \leqslant S_{VirtualAddress} + S_{SizeOfRawData}$$

$$Offset = RVA - S_{RVA_{VirtualAddress}} + S_{RVA_{PointerToRawData}}$$

Lab 07

Seções do PE

Lab 07 - Seções do PE

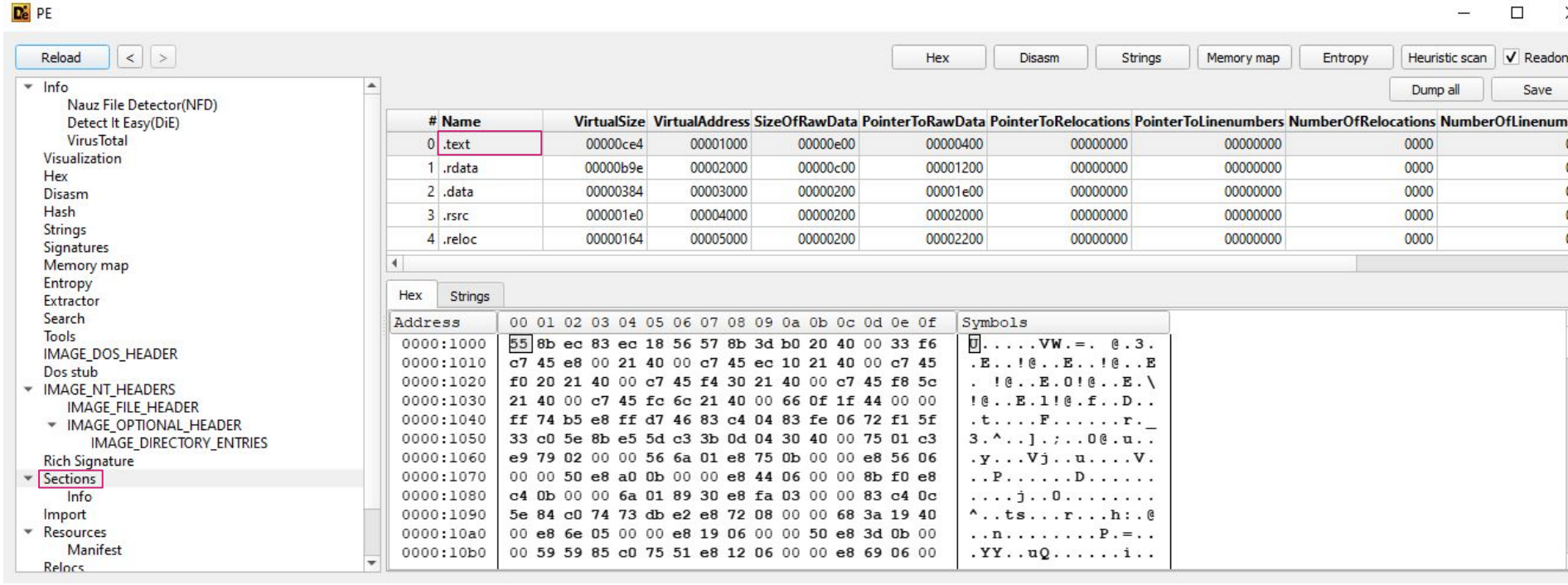
1. Abra o programa *strings1.exe* no DIE e responda:

a) Quais as características da seção **.text** em memória? Elas fazem sentido?
b) Em que seção a string **Papo Binario** está? Quais as características dela em memória? Elas fazem sentido?

- retoolkit → PE → Detect It Easy (DIE)
  - Habilitar a caixa "Advanced" pra poder clicar no botão "PE".
    - Sections
    - Strings

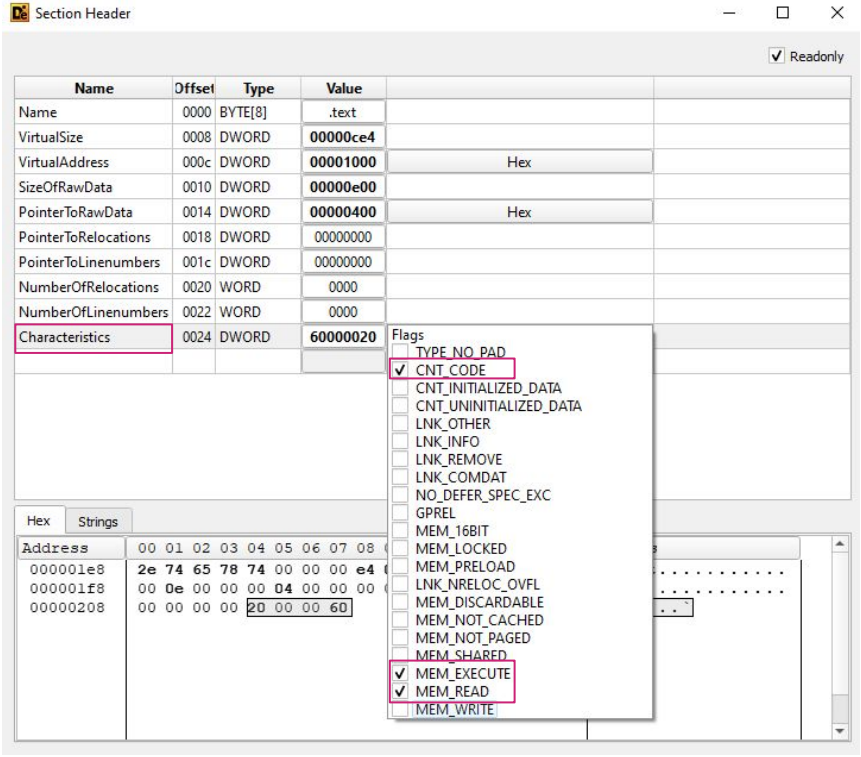
Lab 07 - Seções do PE - Respostas

a) Quais as características da seção **.text** em memória? Elas fazem sentido?



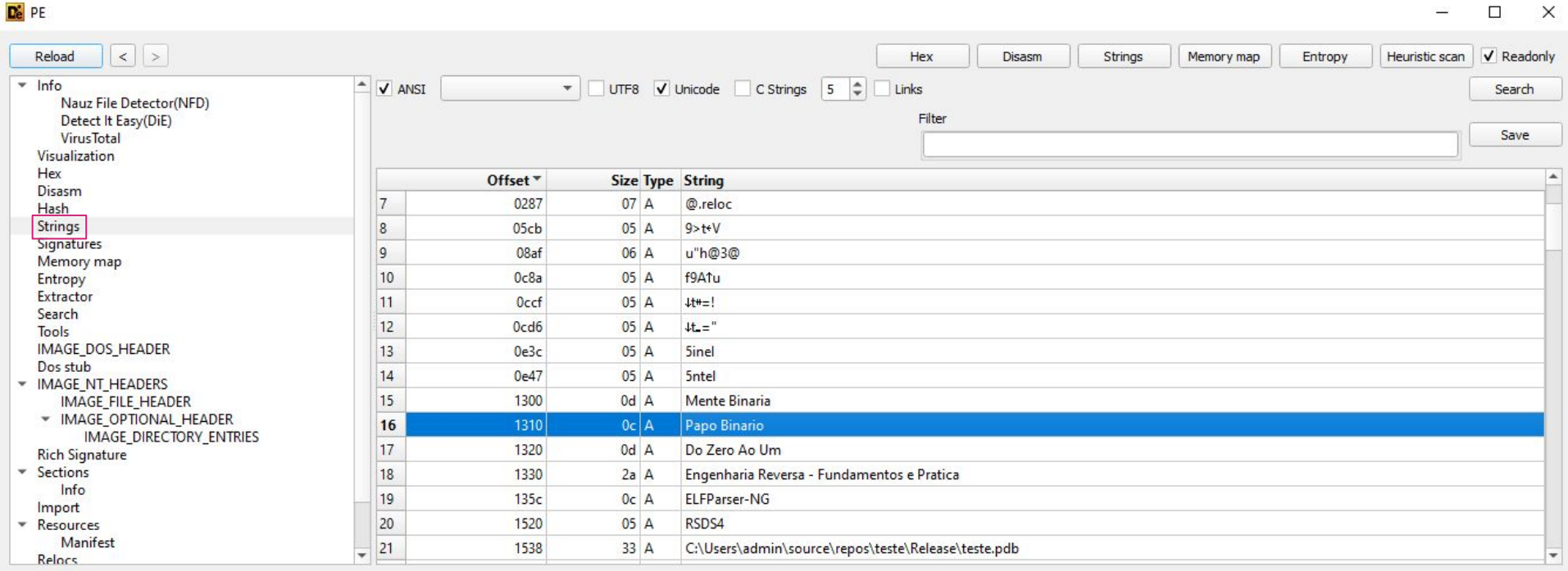
Lab 07 - Seções do PE - Respostas

a) Quais as características da seção **.text** em memória? Elas fazem sentido?



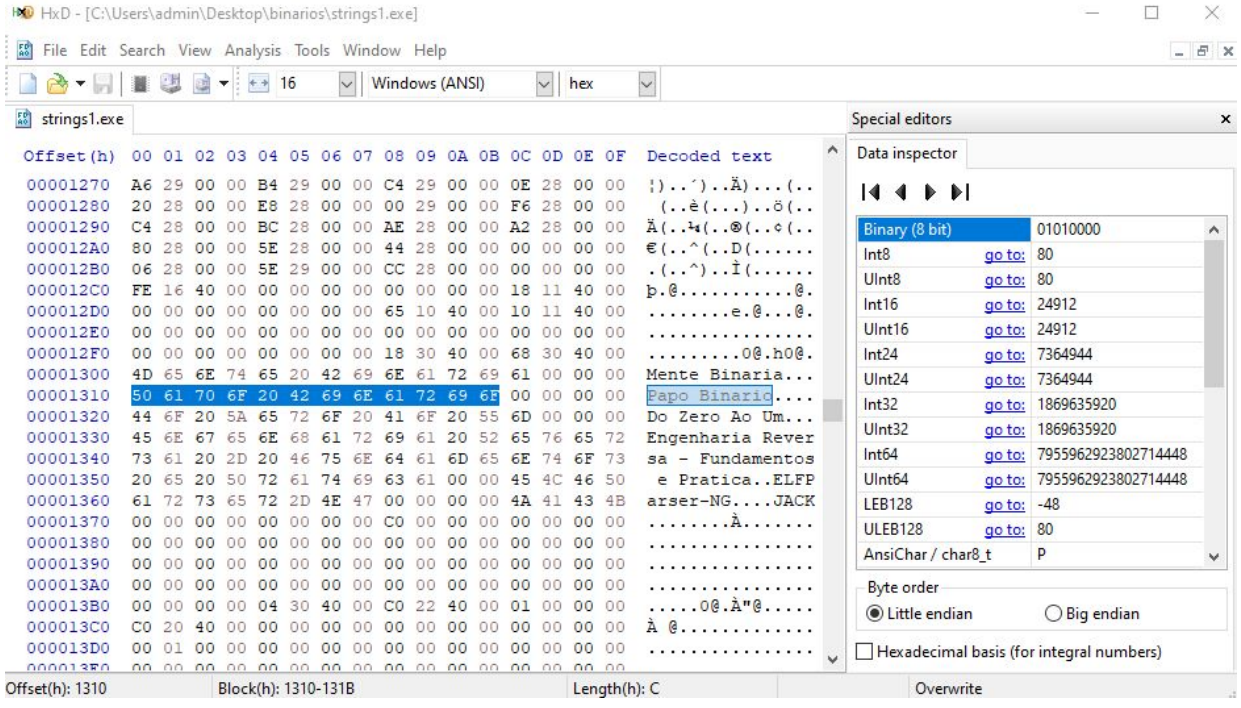
Lab 07 - Seções do PE

b) Em que seção a string **Papo Binario** está? Quais as características dela em memória? Elas fazem sentido?



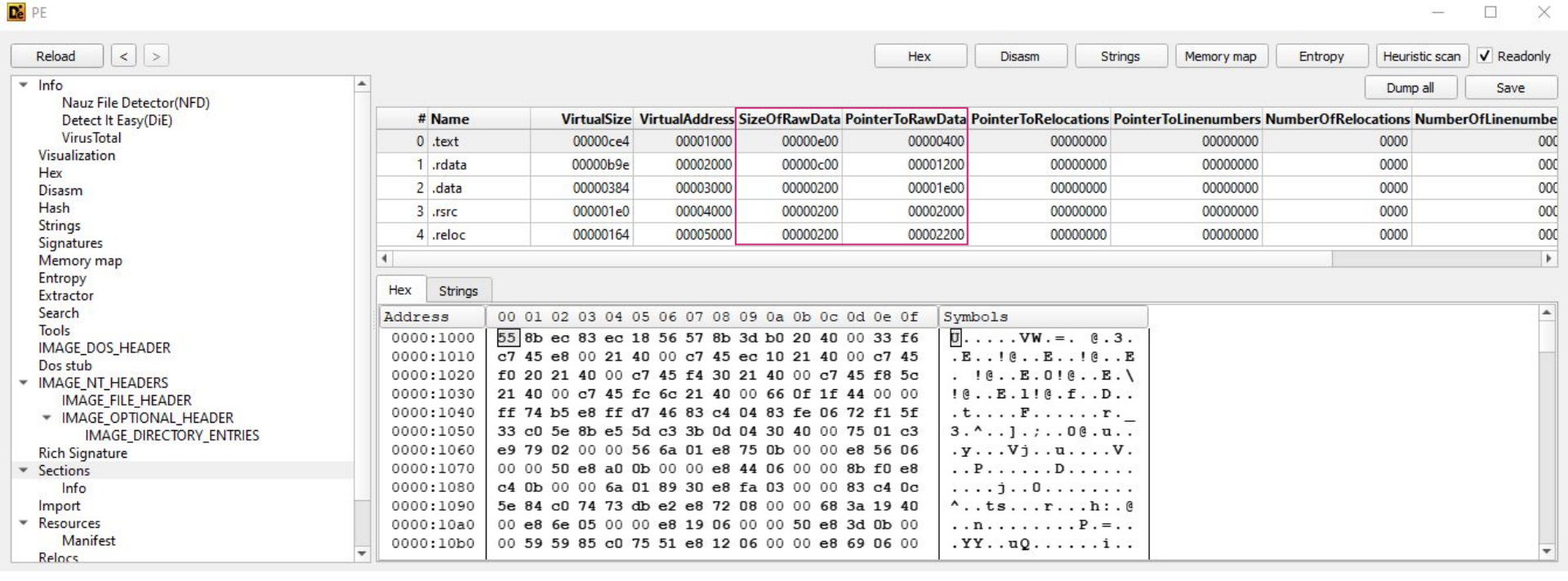
Lab 07 - Seções do PE

b) Em que seção a string **Papo Binario** está? Quais as características dela em memória? Elas fazem sentido?



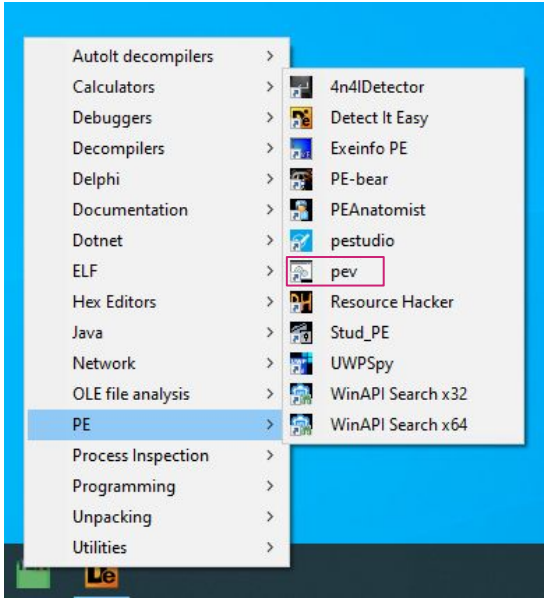
Lab 07 - Seções do PE

b) Em que seção a string **Papo Binario** está? Quais as características dela em memória? Elas fazem sentido?



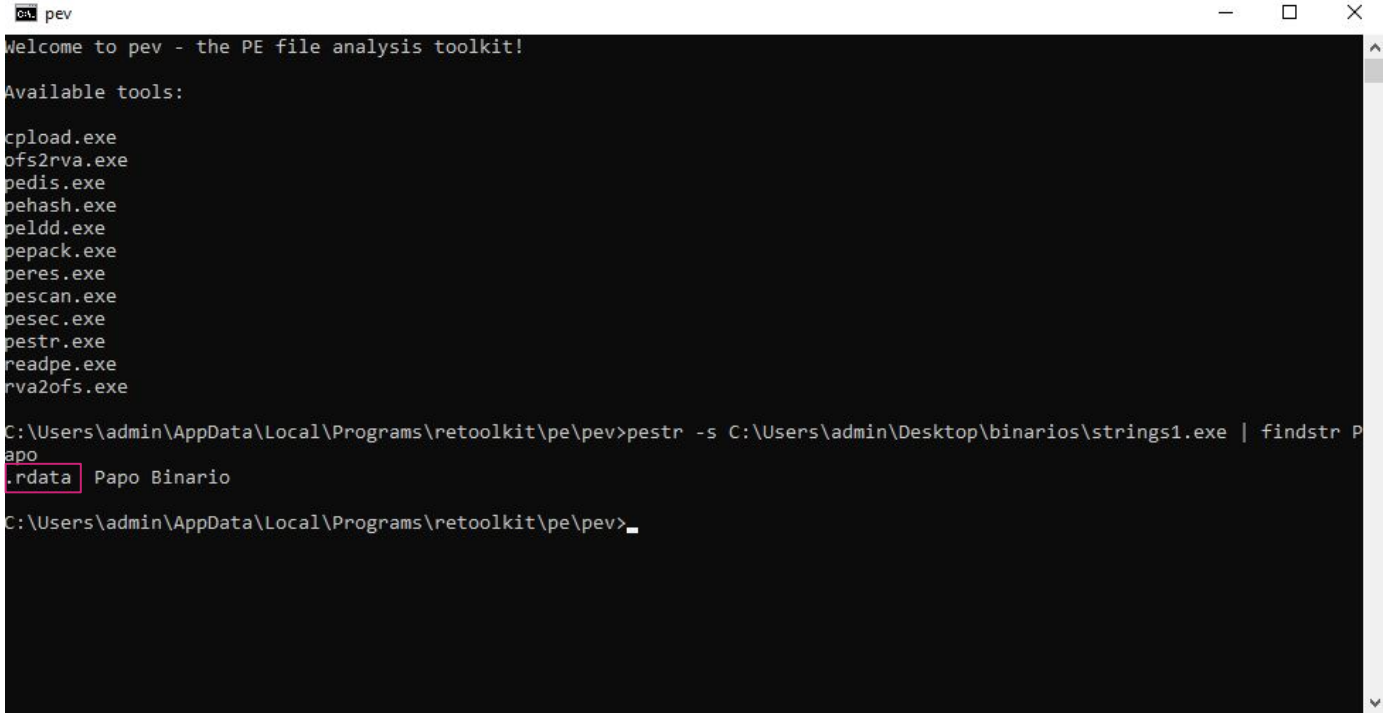
Lab 07 - Seções do PE

b) Em que seção a string **Papo Binario** está? Quais as características dela em memória? Elas fazem sentido?



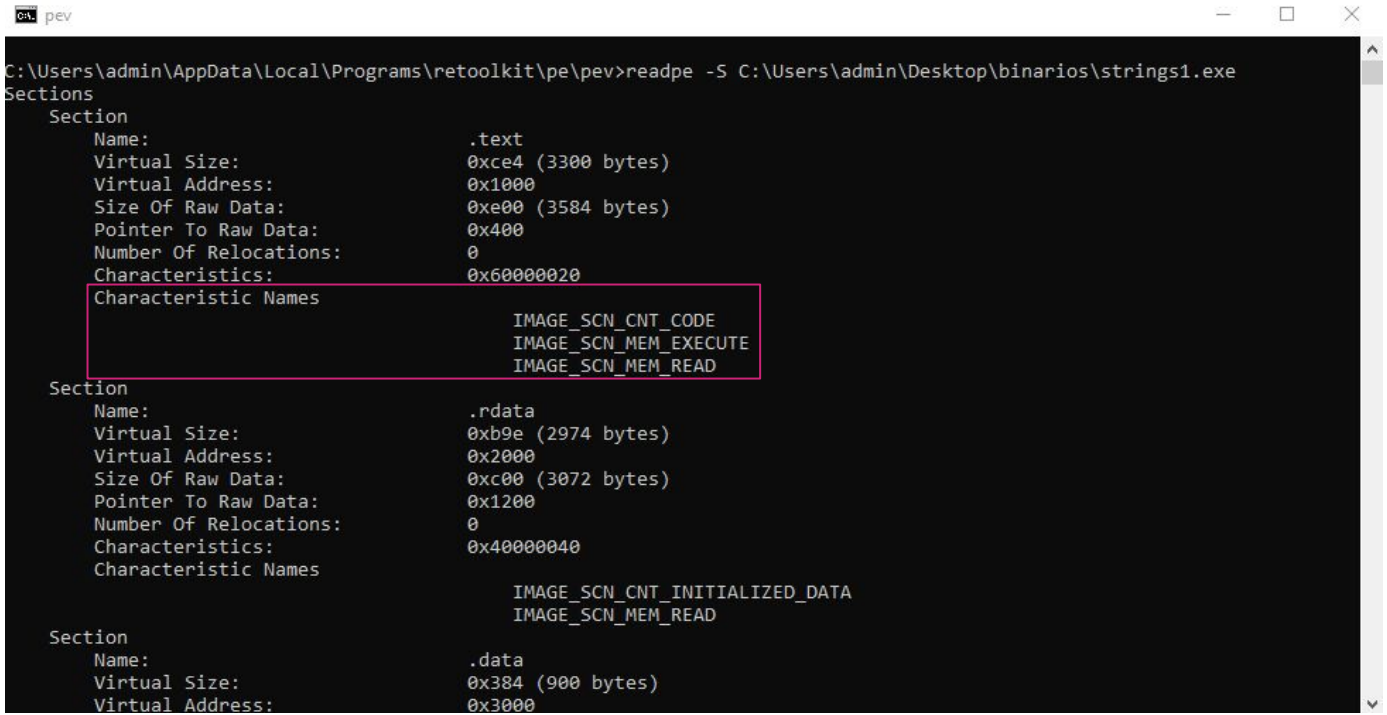
Lab 07 - Seções do PE

b) Em que seção a string **Papo Binario** está? Quais as características dela em memória? Elas fazem sentido?



Lab 07 - Seções do PE

Também daria para verificar as características da seção **.text** com readpe

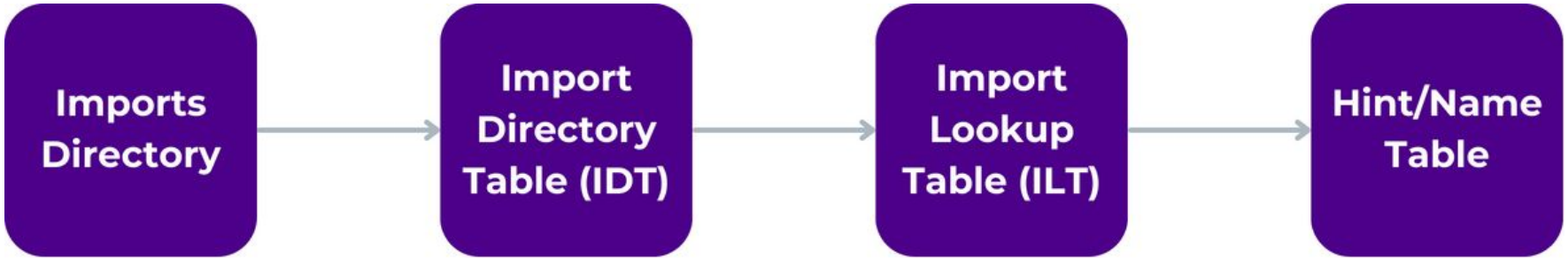


Imports

Imports

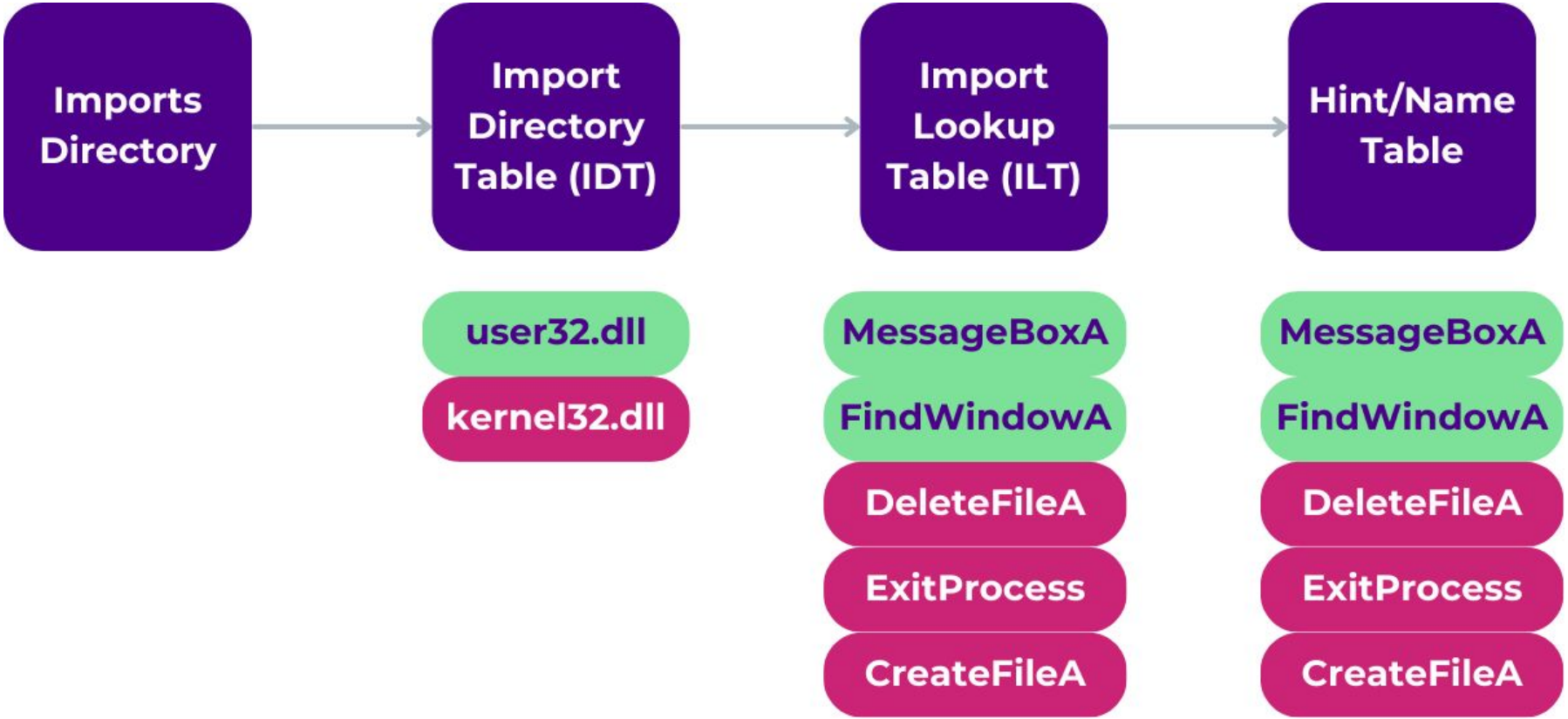
- São as definições de funções importadas de outros módulos. Por exemplo:
  - kernel32.ExitProcess
  - user32.MessageBox
  - advapi32.RegOpenKeyA

Imports



- Para cada elemento na IDT, o *loader*:
  - Carrega a DLL correspondente.
  - Lê as entradas na ILT (uma por função importada).
  - Lê o nome da função na Hint/Name Table

Imports - Exemplo



Import Directory

- Os 16 elementos do diretório de dados possuem a seguinte estrutura:

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

- O elemento de índice 1 dele é o IMAGE_DIRECTORY_ENTRY_IMPORT, que é o **Import Directory**. O VA dele contém o endereço da **Import Directory Table (IDT)**.

Import Directory Table

- Esta tabela é um array de elementos do seguinte tipo:

```
typedef struct {
    union {
        uint32_t Characteristics;
        uint32_t OriginalFirstThunk; // Endereço da ILT
    } u1;
    uint32_t TimeDateStamp;
    uint32_t ForwarderChain;
    uint32_t Name; // Endereço do nome da DLL
    uint32_t FirstThunk; // Endereço da IAT
} IMAGE_IMPORT_DESCRIPTOR;
```

RVAs (mesmo cálculo)

Import Directory Table (Atualização da MS)

Offset	Size	Field	Description
0	4	Import Lookup Table RVA (Characteristics)	The RVA of the import lookup table. This table contains a name or ordinal for each import. (The name "Characteristics" is used in Winnt.h, but no longer describes this field.)
4	4	Time/Date Stamp	The stamp that is set to zero until the image is bound. After the image is bound, this field is set to the time/data stamp of the DLL.
8	4	Forwarder Chain	The index of the first forwarder reference.
12	4	Name RVA	The address of an ASCII string that contains the name of the DLL. This address is relative to the image base.
16	4	Import Address Table RVA (Thunk Table)	The RVA of the import address table. The contents of this table are identical to the contents of the import lookup table until the image is bound.

Import Lookup Table

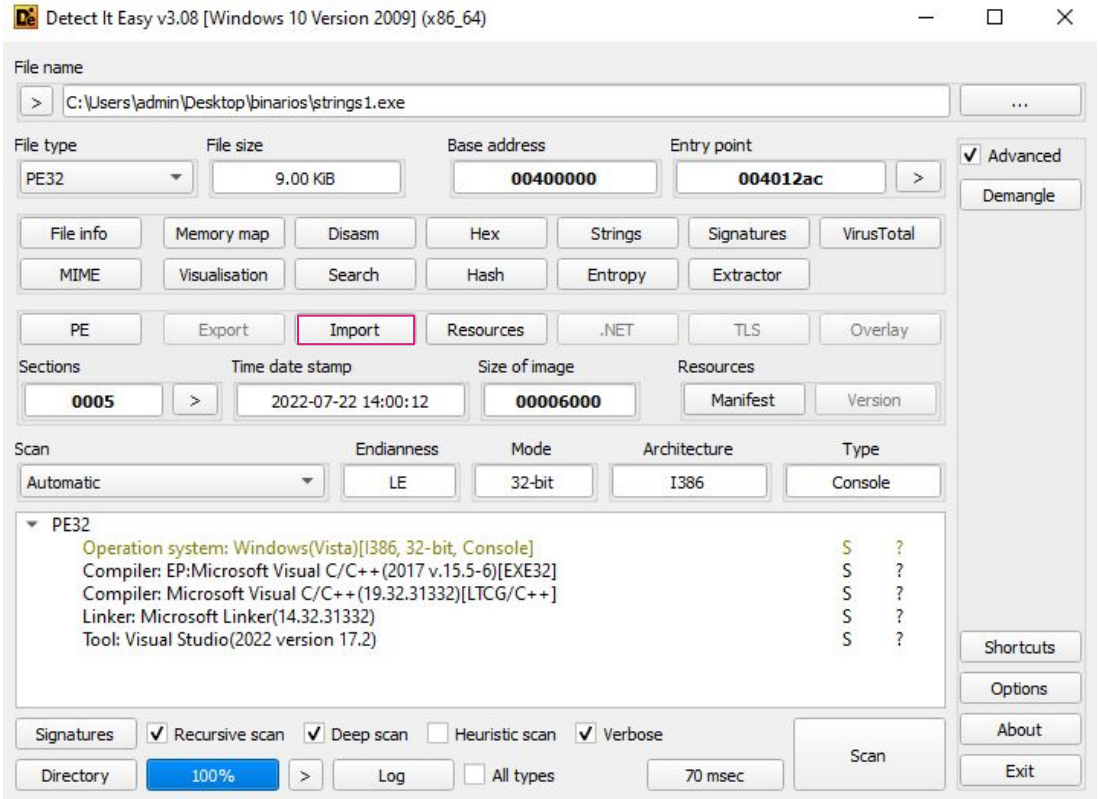
- A ILT é um array de números de 32-bits (ou 64-bits para PE32+). O último bit diz se a importação é ordinal (1) ou nome (0). Os outros bits armazenam o endereço da estrutura do nome, chamada de Hint/Name Table:

```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;
    CHAR    Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

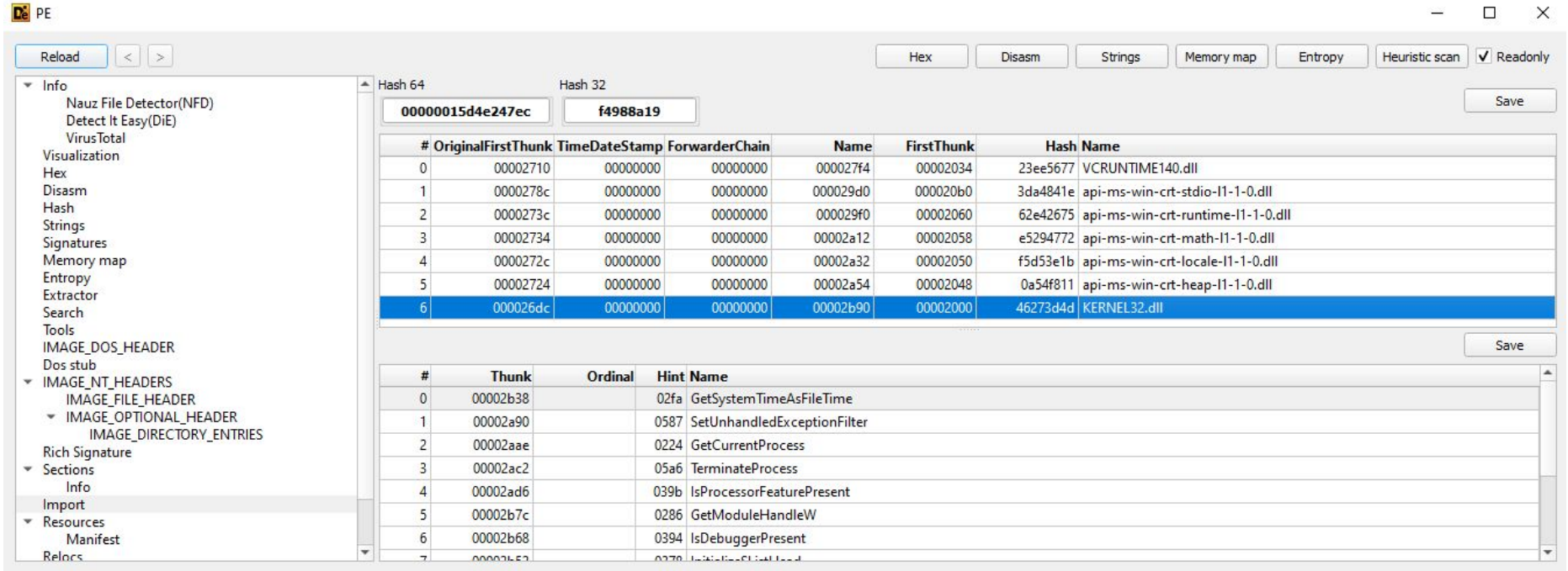
Import Address Table

- Finalmente, a IAT é igual a ILT, até que o binário seja carregado pelo *loader*.
- Após o carregamento, a IAT contém os endereços, em 32-bits, das funções importadas.

Vendo imports no DIE



Vendo imports no DIE



Lab 08

Observando a IAT

Lab 08 - Observando a IAT

1. No DIE, abra o arquivo **AnalyzeMe-04.exe**, marque a caixa **Advanced** e clique em **PE**.
2. Vá até **IMAGE_NT_HEADERS → IMAGE_OPTIONAL_HEADER → IMAGE_DIRECTORY_ENTRIES** e navegue até o diretório **12 - IAT** e memorize o valor em **Address** dele, que na verdade é um RVA.
3. Agora clique em **Sections** no menu da esquerda e observe a qual seção este RVA pertence.
4. Calcule o VA a partir do RVA e volte para a **IAT** em **IMAGE_DIRECTORY_ENTRIES**. Você vai precisar deste VA no passo seguinte.
5. Feche o DIE. Abra o binário no x32dbg. Clique em qualquer byte na aba **Dump 1** para selecioná-lo e aperte **Ctrl+G**. Digite o VA calculado no passo 4 e pressione **ENTER**.

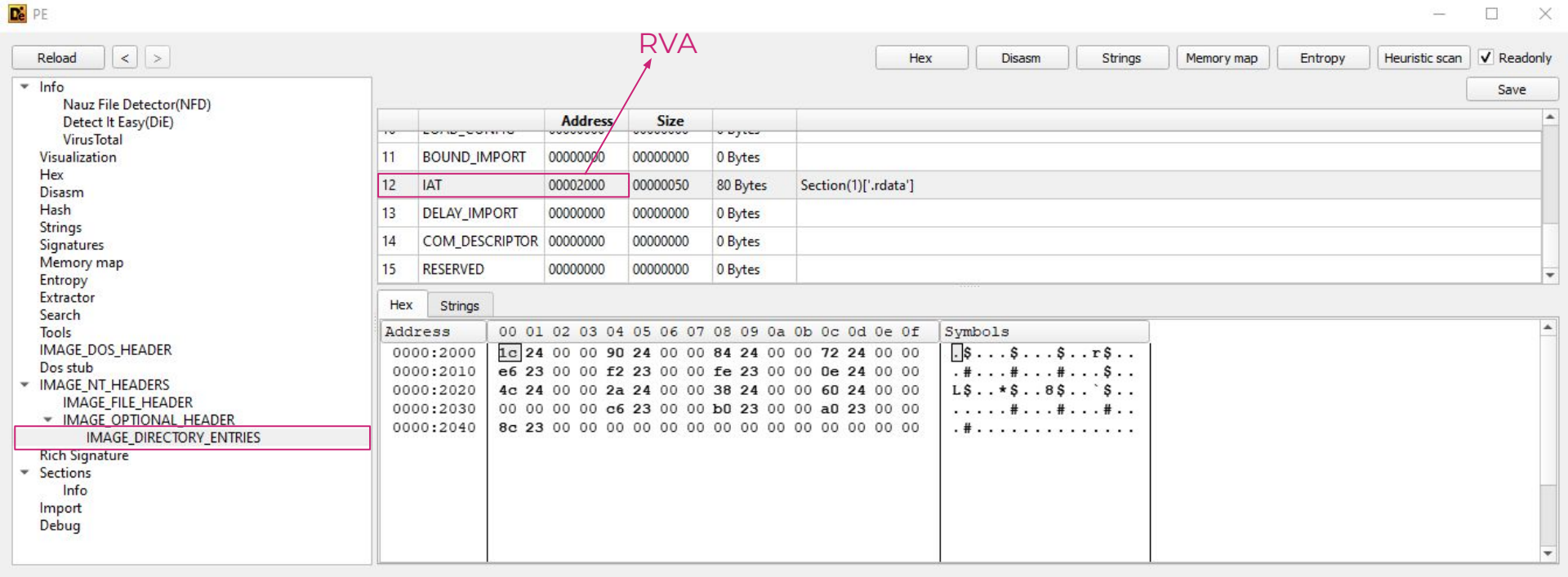
Lab 08 - Observando a IAT

Agora observando no dump, responda às seguintes perguntas:

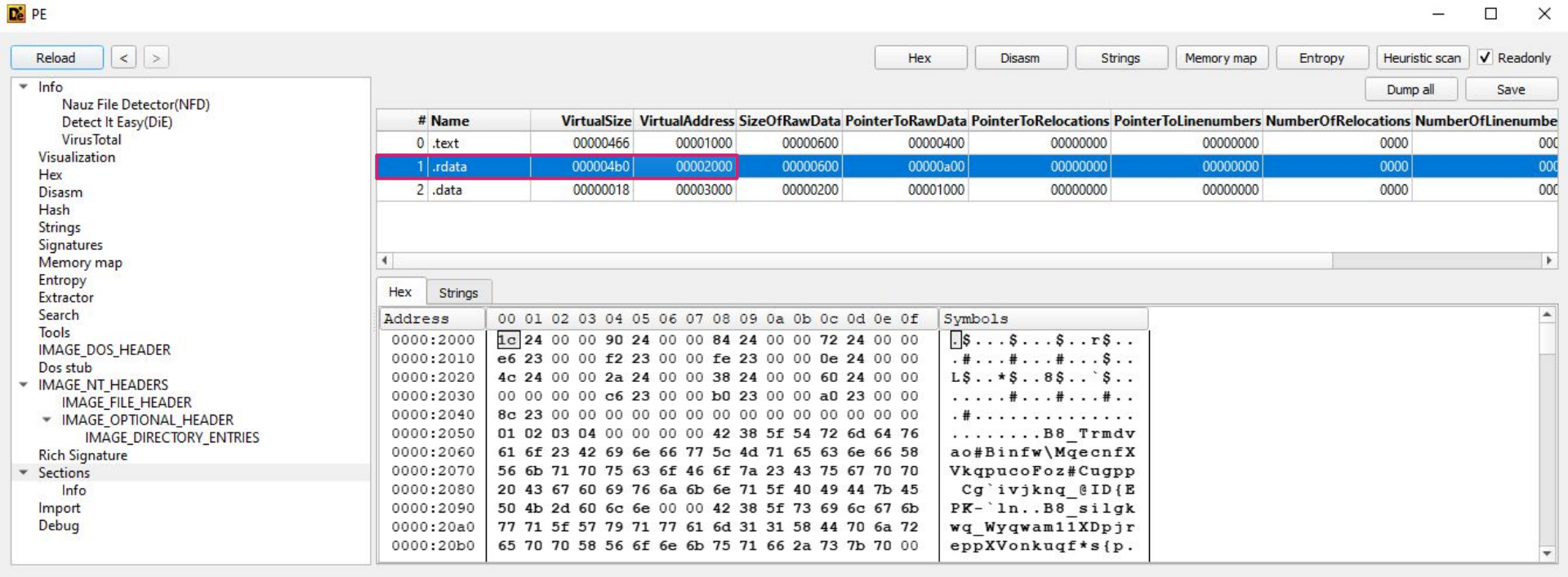
1. Por que os valores mudaram na imagem?
2. Quantas funções este binário importa?
3. EXTRA: Clique com o botão direito no dump e escolha a opção **Address** para visualizar endereços*. Há alguma diferença entre as funções na IAT e na ILT (veja **Imports** no DIE)? Pesquise o motivo.

* Para voltar a visualizar os bytes em hexa com o dump em ASCII, clique novamente com o botão direito em qualquer byte no dump e escolha a opção **Hex → ASCII**.

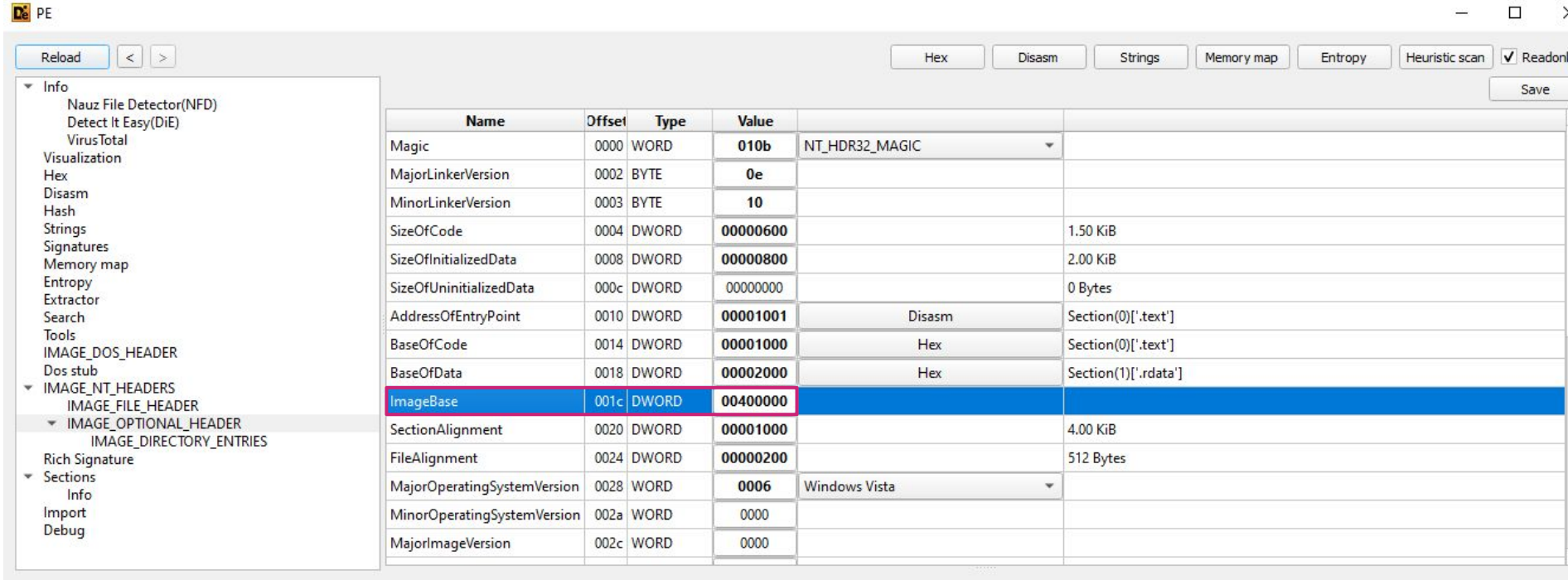
Lab 08 - Observando a IAT - Respostas



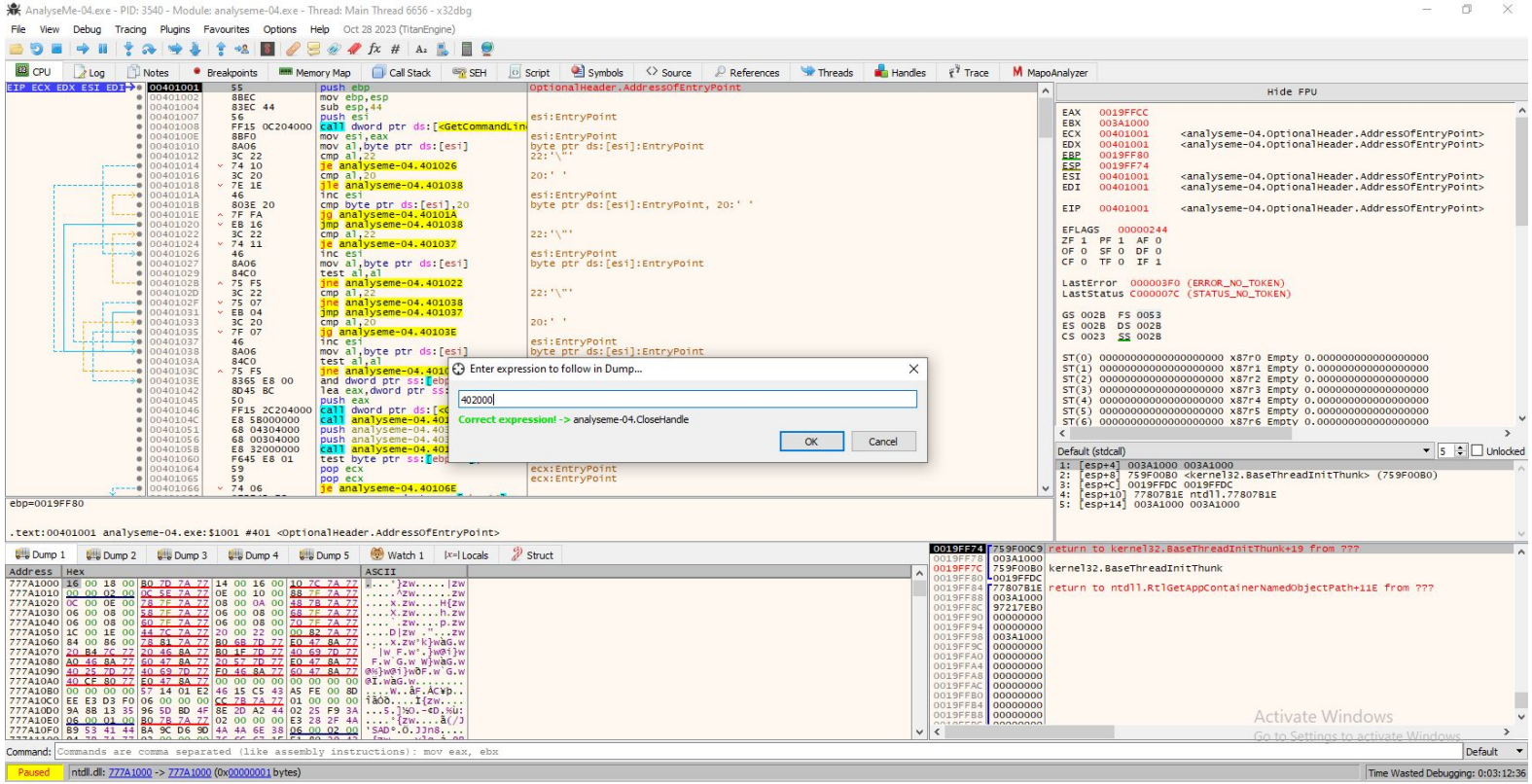
Lab 08 - Observando a IAT - Respostas



Lab 08 - Observando a IAT - Respostas

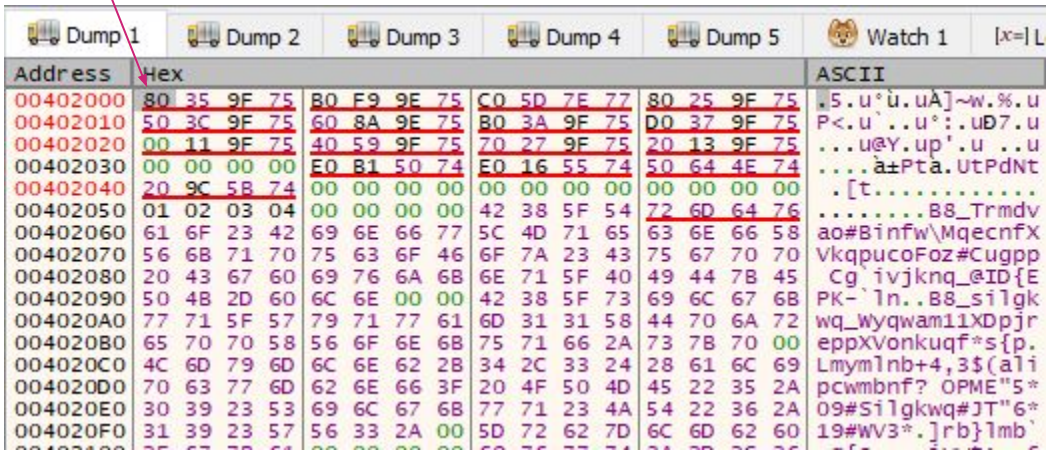


Lab 08 - Observando a IAT - Respostas



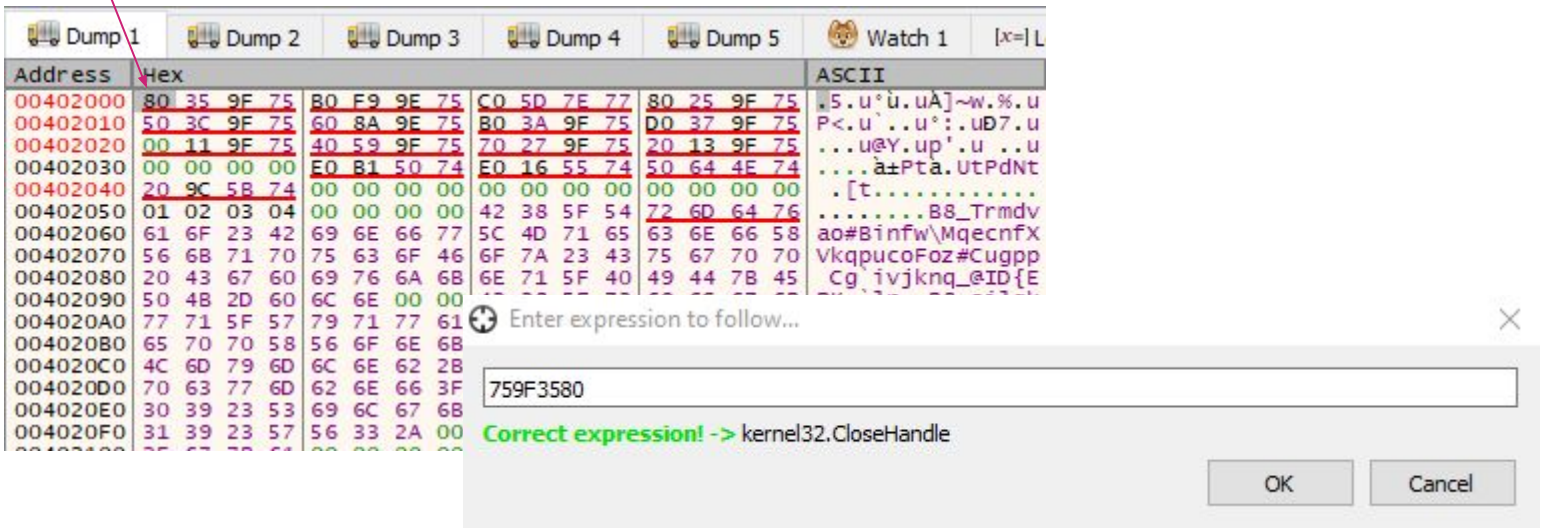
Lab 08 - Observando a IAT - Respostas

Endereços de funções

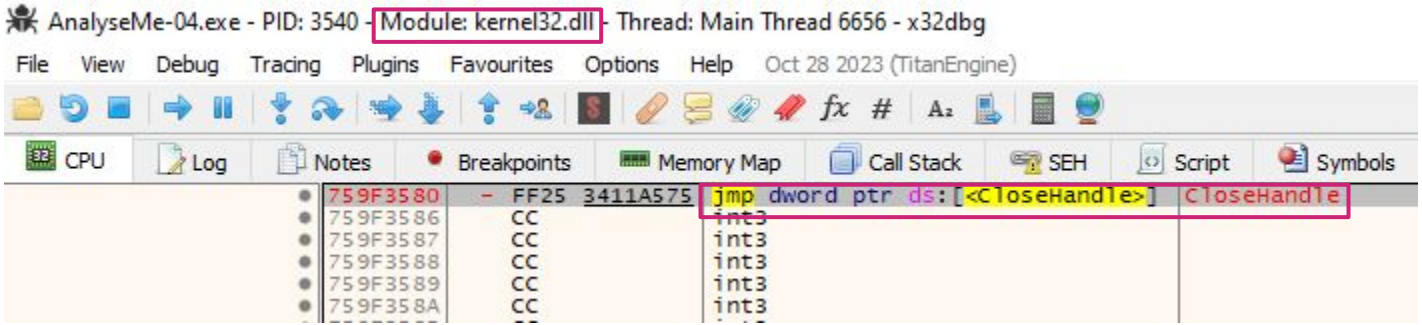


Lab 08 - Observando a IAT - Respostas

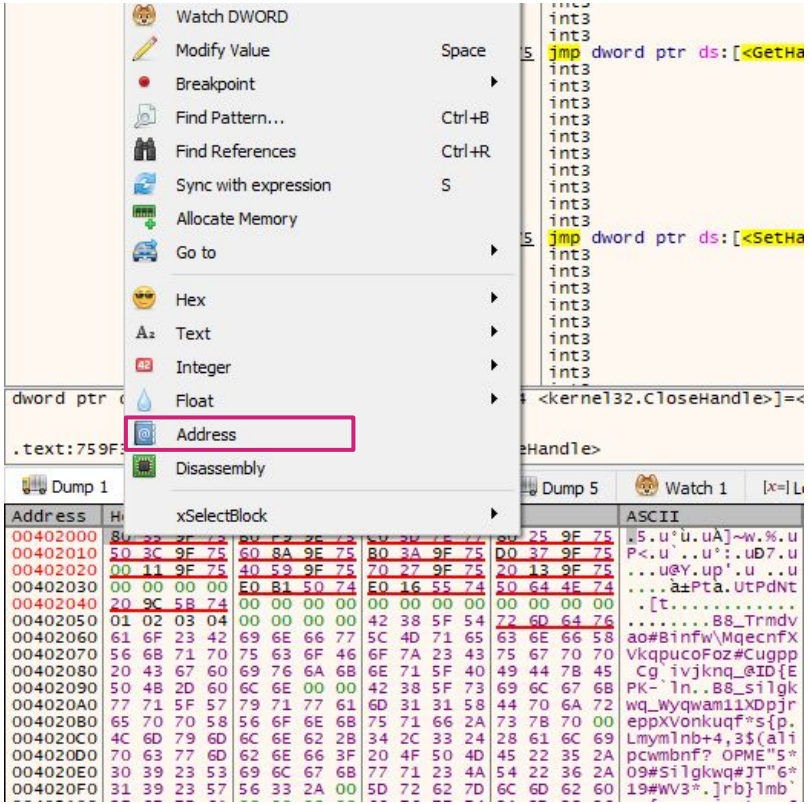
Endereços de funções



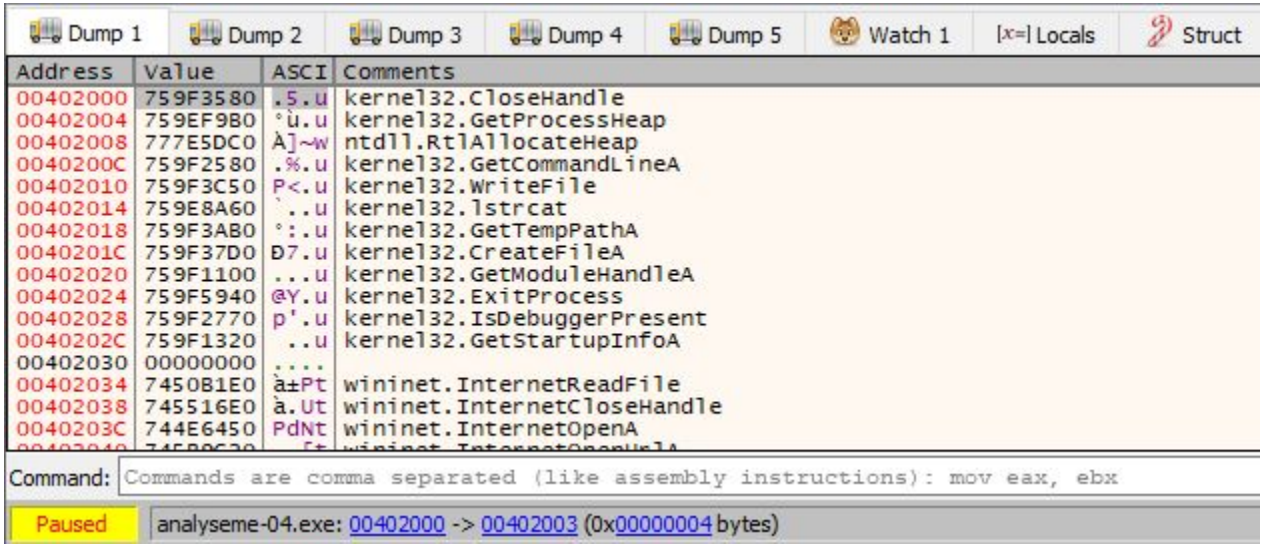
Lab 08 - Observando a IAT - Respostas



Lab 08 - Observando a IAT - Respostas

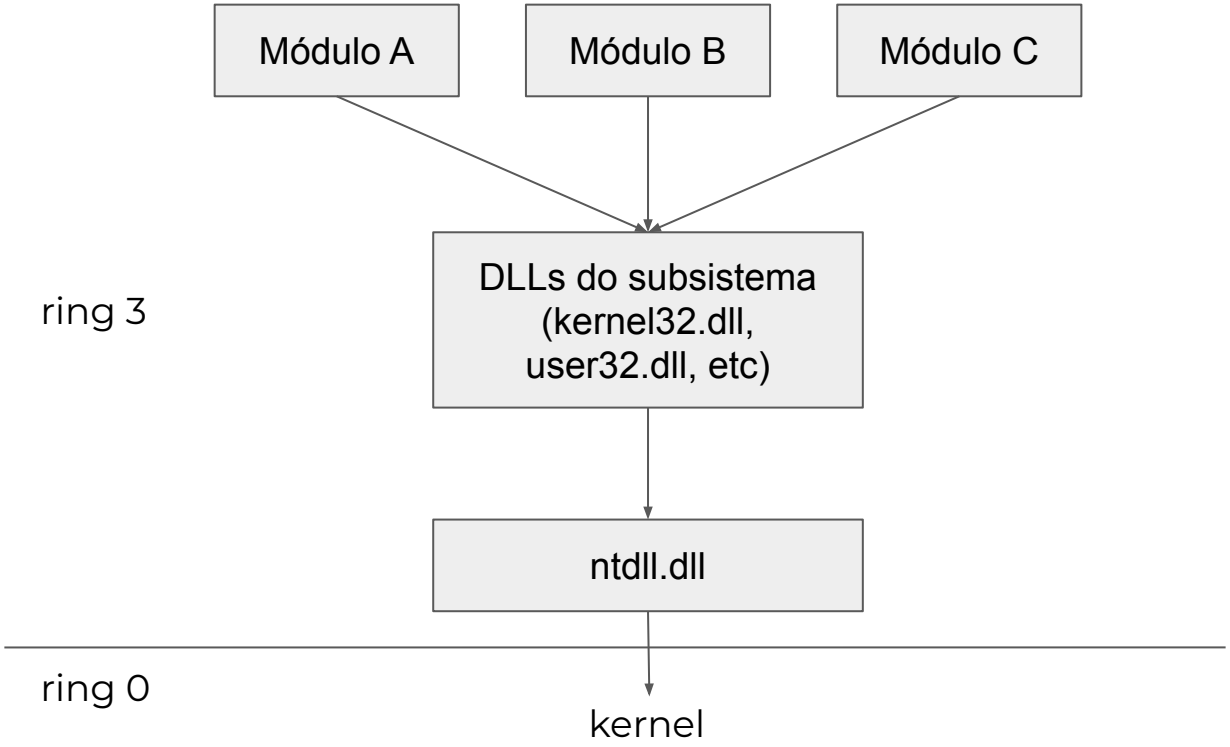


Lab 08 - Observando a IAT - Respostas



Execução de Programas

Execução de Programas



Windows API

MessageBox (USER32.DLL)

```
int MessageBox(
  [in, optional] HWND    hWnd,
  [in, optional] LPCTSTR lpText,
  [in, optional] LPCTSTR lpCaption,
  [in]           UINT    uType
);
```

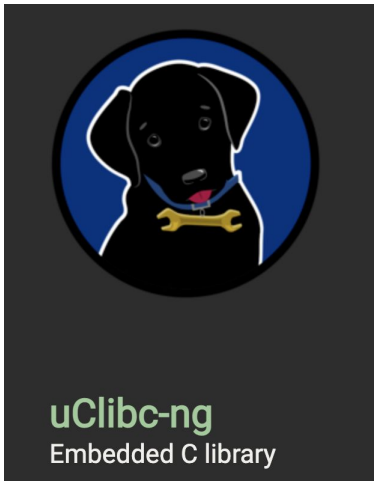
```c
#include <windows.h>

int main(void) {
    //MessageBox(0, ...)
    //MessageBox(nullptr, ...)
    MessageBox(NULL, "Cash", "Johnny", MB_OK);
}
```

Linux C Library (libc)

Funções da libc

- printf, malloc, strstr, strlen, etc.

The GNU C Library (glibc)



Linux libc wrappers

write

```c
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

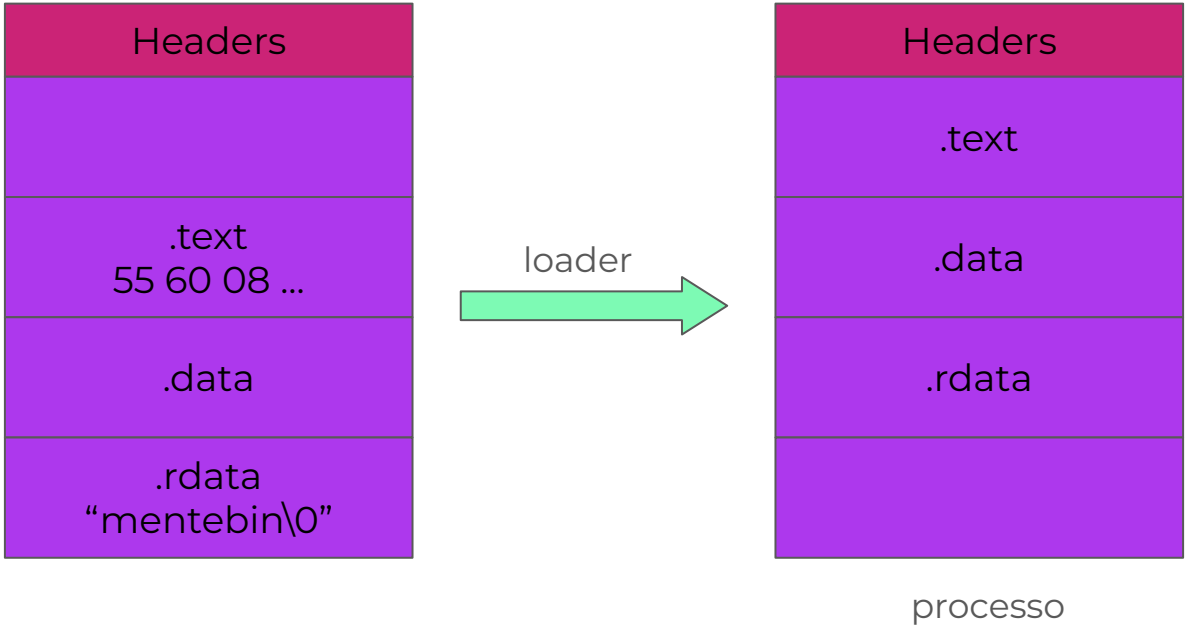
```c
#include <unistd.h>

int main(void) {
        char *msg = "oie\n";
        write(1, msg, 4);
}
```

Introdução à Assembly x64

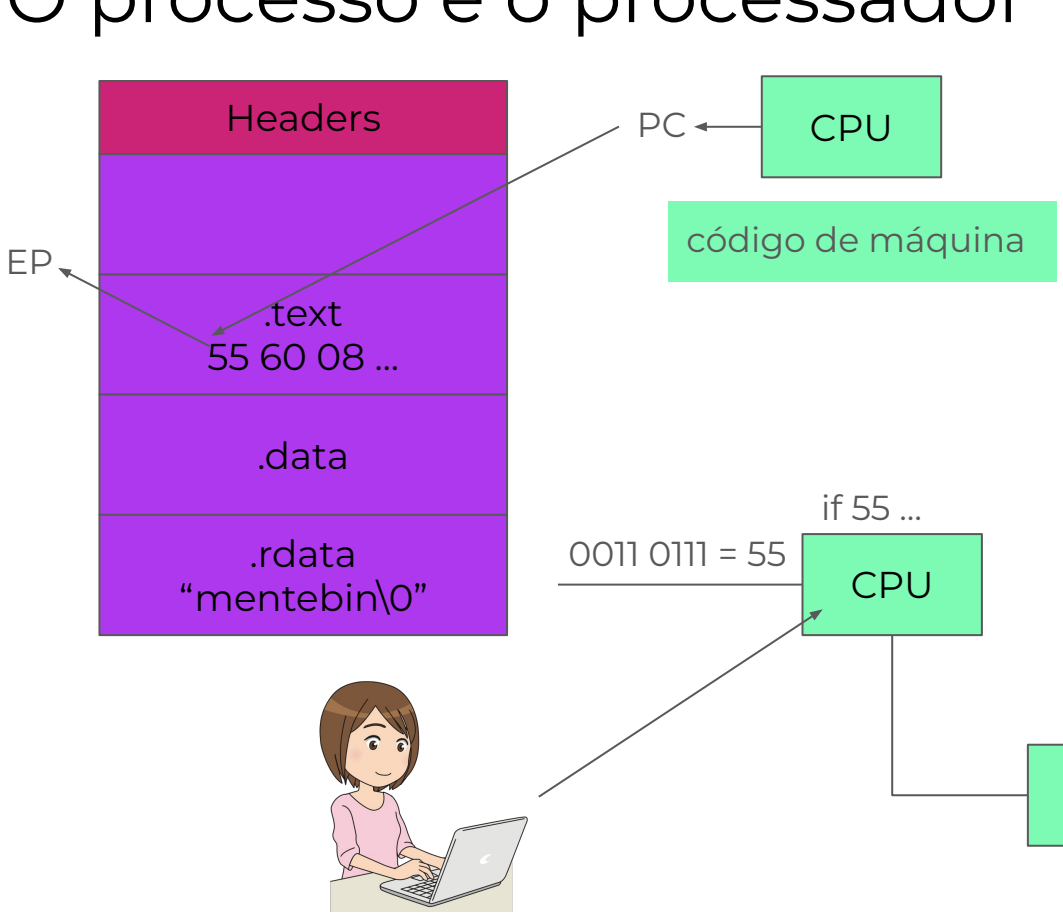
Arquiteturas

- O que é, exatamente, Assembly?

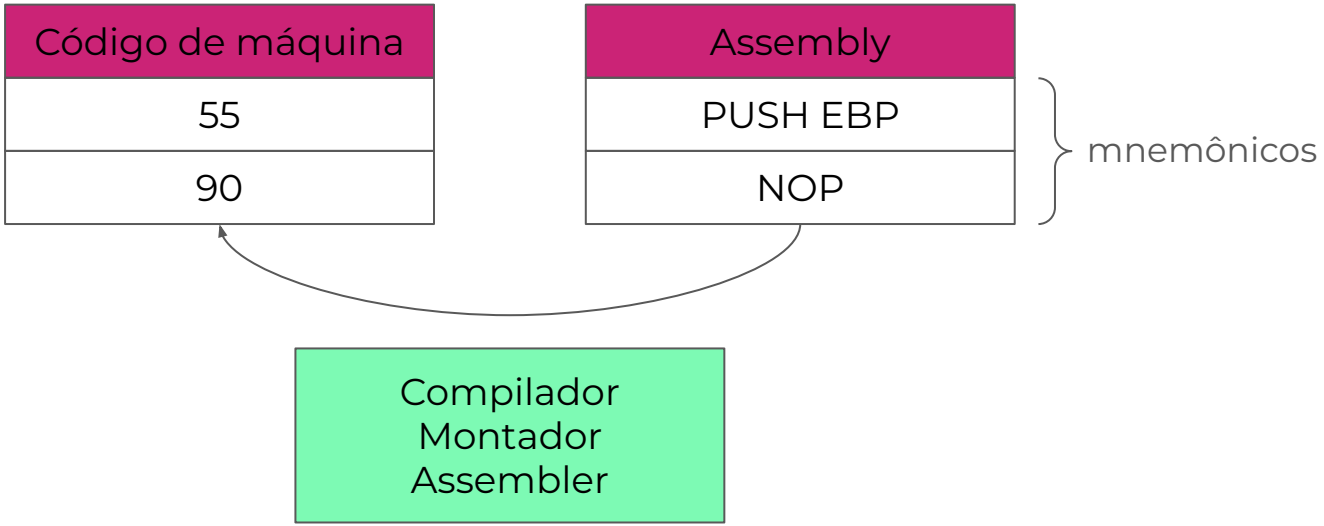
Criação do processo



O processo e o processador



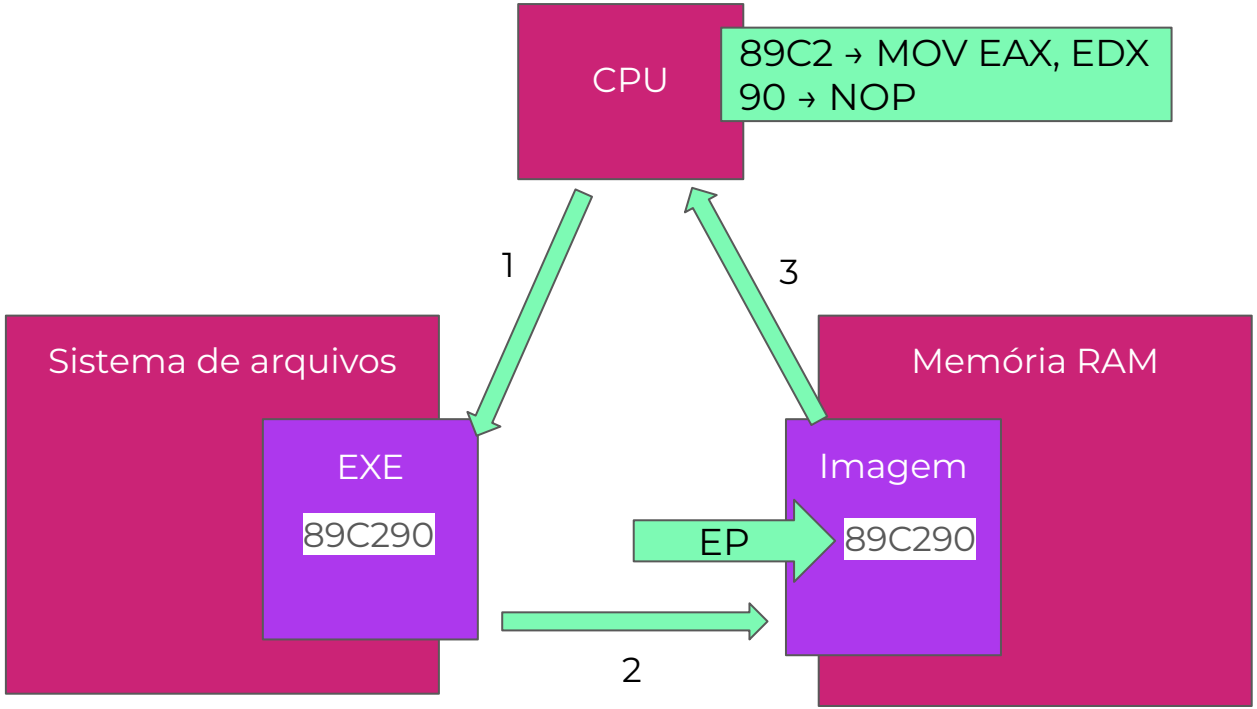
O processo e o processador



Registradores

- Control Registers (CR)
- **Status/Control Register EFLAGS**
- Segment Registers
- **General Purpose Registers (GPR)**
- **Instruction Pointer (IP)**
- Model Specific Registers (MSR)
- Descriptor Table Registers (DTR)
- SSE
- x87 FPU e MMX
- Debug Registers

Processo de carregamento (simplificado)



Registradores de Propósito Geral (GPRs)

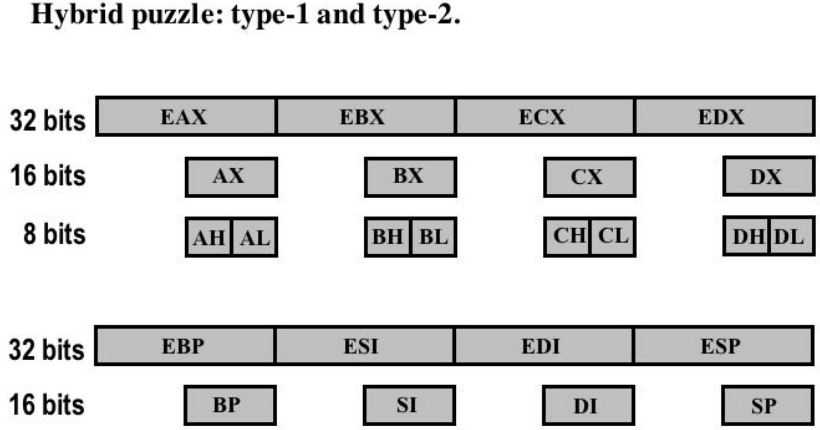
- Oito GPRs no modo IA-32.
- Todos de 32-bits.

Registrador	Significado	Uso sugerido
EAX	Accumulator	Usado em operações aritiméticas
EBX	Base	Ponteiro para dados
ECX	Counter	Contador em repetições
EDX	Data	Usado em operações de E/S
ESI	Source Index	Ponteiro para uma *string* de origem
EDI	Destination Index	Ponteiro para uma *string* de destino
ESP	Stack Pointer	Ponteiro para o topo da pilha
EBP	Base Pointer	Ponteiro para a base do *stack frame*

Registradores de Propósito Geral (GPRs)

- Os primeiros quatro podem ser acessados como registradores de 8 e 16-bits.
- Os registradores de 16-bits consistem em dois de 8-bits.
  - Ex.: EAX (32-bits), AX (16-bits), AL (metade baixa, de 8-bits, de AX) e AH (metade alta de AX).

X86-32 register board:

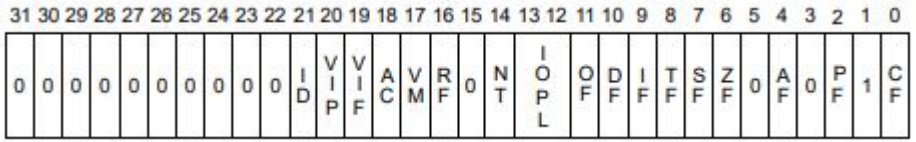


Instruction Pointer (IP)

- Contém o endereço da próxima instrução a ser executada.
- É incrementado de acordo com o tamanho da instrução.

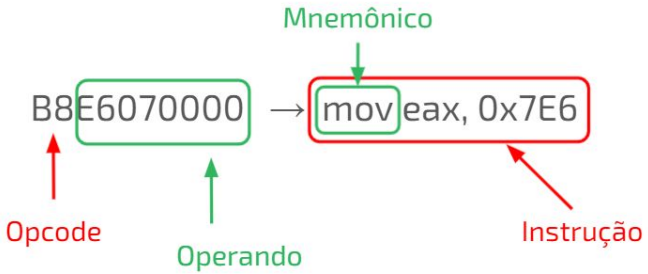
EFLAGS

- Dez flags de sistema
- Uma flag de controle
- Seis flags de estado



Instruções

- O que é uma instrução?
  - Sequência de 1 a 15 bytes de tamanho, podendo compreender *operation code* (***opcode***), **operandos** e **prefixo**.
- O processador entende determinadas sequências de bytes como instruções válidas. Exemplo:



MOV

- Copia dados de um lugar para outro. O destino pode ser um registrador ou um endereço de memória. Já a origem pode ser um literal (*immediate*), um registrador ou um endereço de memória.
  - MOV EAX, 1234
  - MOV EAX, EBX
  - MOV EAX, DWORD PTR DS:[402000]
  - MOV DWORD PTR DS:[402000], 1234
  - MOV BX, WORD PTR DS:[402000]
  - MOV AL, BYTE PTR SS:[19FF7C]
  - MOV ECX, [ESP]