

# Extraindo Características de Arquivos Binários Executáveis

Marcus Botacin<sup>1</sup>

<sup>1</sup>Universidade Federal do Paraná (UFPR)  
mfbotacin@inf.ufpr.br

2021

# Por que Investigar Binários Executáveis?

- Para detectar malware (classificação malware vs. goodware)
- Para realizar análises forenses (o que tem dentro?).
- Para se identificar versões (Windows 8 vs. Windows 10).
- Para se criar uma filogenia (qual software veio antes de qual?).
  - E se você não tem o código-fonte?

## Certeza que isso funciona?

```
marcus@tux:/tmp$ hexdump -C binario.exe | head -20
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 20 01 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  01 6d 3a 0e 45 0c 54 5d  45 0c 54 5d 45 0c 54 5d  |.m:.E.T]E.T]E.T]|
00000090  2a 68 57 5c 4a 0c 54 5d  2a 68 51 5c 8a 0c 54 5d  |*hW\J.T]*hQ\..T]|
000000a0  04 6b 51 5c 4d 0c 54 5d  56 6a 57 5c 5f 0c 54 5d  |.kQ\M.T]VjW\..T]|
000000b0  56 6a 50 5c 60 0c 54 5d  56 6a 51 5c 17 0c 54 5d  |VjP\`.T]VjQ\..T]|
000000c0  2a 68 50 5c 5e 0c 54 5d  2a 68 52 5c 46 0c 54 5d  |*hP\^T]*hR\F.T]|
000000d0  2a 68 53 5c 44 0c 54 5d  2a 68 55 5c 5c 0c 54 5d  |*hS\D.T]*hU\\T]|
000000e0  45 0c 55 5d 2a 0e 54 5d  04 6b 5d 5c dc 0d 54 5d  |E.U]*.T].k]\..T]|
000000f0  04 6b ab 5d 44 0c 54 5d  04 6b 56 5c 44 0c 54 5d  |.k.]D.T].kV\D.T]|
00000100  52 69 63 68 45 0c 54 5d  00 00 00 00 00 00 00 00  |RichE.T].....|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
00000120  50 45 00 00 4c 01 05 00  63 41 9e 5c 00 00 00 00  |PE..L...CA.\....|
00000130  00 00 00 00 e0 00 02 01  0b 01 0e 0e 00 3e 21 00  |.....>!.|
```

# Como Funcionam Binários Executáveis?

## Definição

- Binários executáveis precisam seguir uma estrutura pré-definida para que o sistema operacional saiba como carregá-los.

## Informações Importantes para o SO

- Quantos *bytes* devem ser alocados para carregar um dado binário?
- Qual a primeira instrução a ser executada?

# Binários Executáveis Dependem do SO

## Iguais, mas diferentes!

- Em geral, binários executáveis respondem as mesmas perguntas ao SO, mas seus campos tem tamanhos diferentes e estão localizados em diferentes *structs*.

## Formatos para cada SO

- **Linux:** Executable and Linkable Format (ELF).
- **Windows:** Portable Executable (PE).<sup>a</sup>

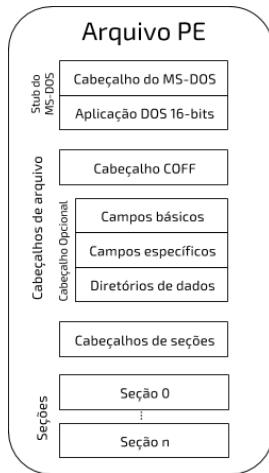
---

<sup>a</sup>**Busque Conhecimento:** A Wikipedia sabe tudo sobre estes formatos!

# O formato PE

## Estrutura Básica

- **Cabeçalhos:** Respostas básicas e apontadores para dados.
- **Seções:** Os dados em si.



## Como o OS interpreta um arquivo binário executável

```
marcus@tux:/tmp$ file binario.exe  
binario.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

## Como você deveria interpretar

```
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;  
9     uint16_t e_ss;  
10    uint16_t e_sp;  
11    uint16_t e_csum;
```

```
1     uint16_t e_ip;  
2     uint16_t e_cs;  
3     uint16_t e_lfarlc;  
4     uint16_t e_ovno;  
5     uint16_t e_res[4];  
6     uint16_t e_oemid;  
7     uint16_t e_oeminfo;  
8     uint16_t e_res2[10];  
9     uint32_t e_lfanew;  
10 } IMAGE_DOS_HEADER;
```



## O que tem dentro das seções?

### Uma generalização

- **.text**: Instruções.
- **.data**: Dados inicializados (constantes?).
- **.bss**: Dados não-inicializados (variáveis?).
- **outras**: Binários podem ter diferentes números e nomes de seções.

## O Que Podemos descobrir sabendo disso?

- Se identifiquei o *magic* MZ num buffer: Injeção de código.
- Se identifiquei um *timestamp* anterior a 1970: Valor forjado.
- Se o *checksum* não bate: Binário adulterado.
- Seções com permissões RWX: Código auto-modificável.

## Como Manipular Binários Executáveis?

- **ELF:** `pyelftools` (<https://github.com/eliben/pyelftools>)
- **PE:** `pefile` (<https://github.com/erocarrera/pefile>)
- **Multi:** `lief` (<https://lief.quarkslab.com/>)

## Exemplos com pefile (1/2)

```
import sys
import pefile
pe = pefile.PE(sys.argv[1])

print("[*] e_magic value: %s" % hex(pe.DOS_HEADER.e_magic))
print("[*] Signature value: %s" % hex(pe.NT_HEADERS.Signature))
```

```
marcus@tux:/tmp$ python 1.py binario.exe
[*] e_magic value: 0x5a4d
[*] Signature value: 0x4550
```

## Exemplos com pefile (2/2)

```
import sys
import pefile
pe = pefile.PE(sys.argv[1])

for section in pe.sections:
    print(section.Name.decode('utf-8'))
    print("\tVirtual Address: " + hex(section.VirtualAddress))
    print("\tVirtual Size: " + hex(section.Misc_VirtualSize))
    print("\tRaw Size: " + hex(section.SizeOfRawData))
```

## Exemplos com pefile (2/2)

```
marcus@tux:/tmp$ python 2.py binario.exe
.text
    Virtual Address: 0x1000
    Virtual Size: 0x213ccb
    Raw Size: 0x213e00
.rdata
    Virtual Address: 0x215000
    Virtual Size: 0x80fe8
    Raw Size: 0x81000
.data
    Virtual Address: 0x296000
    Virtual Size: 0x8620
    Raw Size: 0x3400
.rsrc
    Virtual Address: 0x29f000
    Virtual Size: 0xb00f8
    Raw Size: 0xb0200
.reloc
    Virtual Address: 0x350000
    Virtual Size: 0x24104
    Raw Size: 0x24200
```

## Tarefa 1/2

- Enumerar as seções executáveis de um binário PE.

```
marcus@tux:/tmp$ python enum.py binario.exe  
Seção: .text é executável:      r-x  
Seção: .data [não] é executável:rw-
```

## Tarefa 2/2

- Comparar as seções de dois binários PE.

```
marcus@tux:/tmp$ python compare.py binario.exe binario2.exe
Binário binario.exe e binario2.exe contém a seção .text executável:      r-x
Apenas binário binario.exe contém a seção .data [não] executável:rw-
```