Tue 10:30
Thur 10:30

Robot #301

역공학

Reverse Engineering

Basic Static Analysis

Determining File Type, Fingerprinting Files, Extracting Strings,
Inspecting PE File Format, Comparing & Classifying Executable Files

박 정 흠

Park, Jungheum

jungheumpark@korea.ac.kr

Korea University
School of Cybersecurity

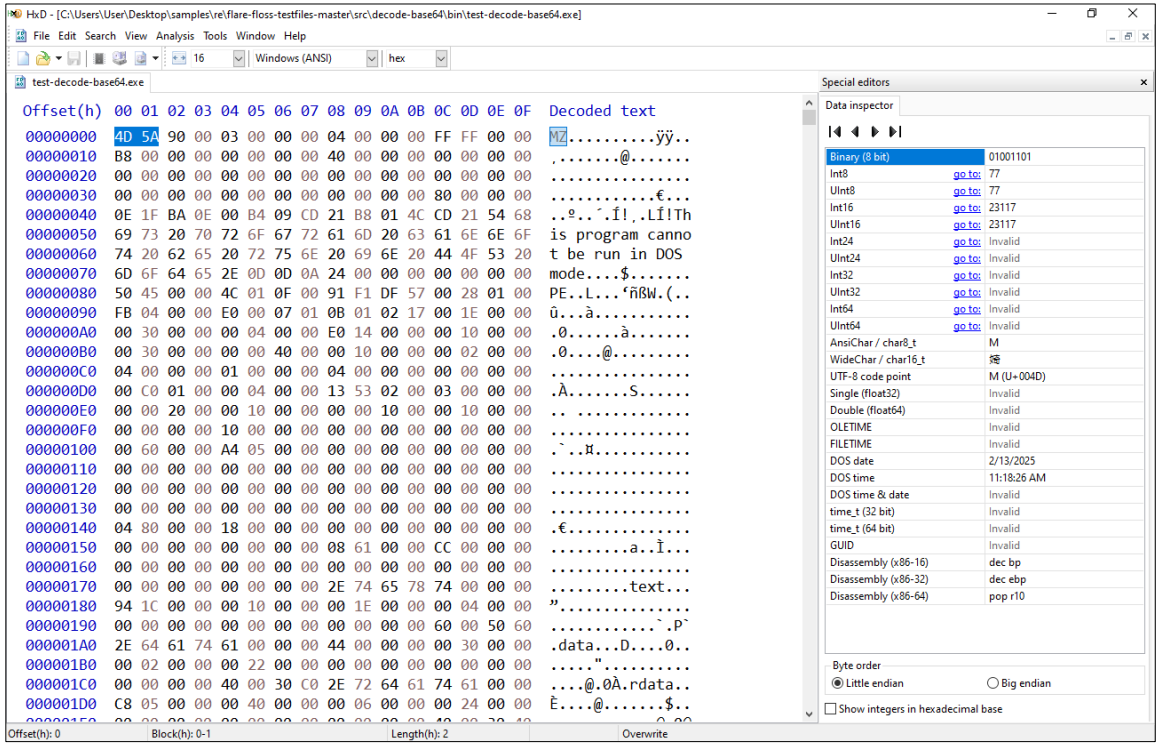
Table of Contents

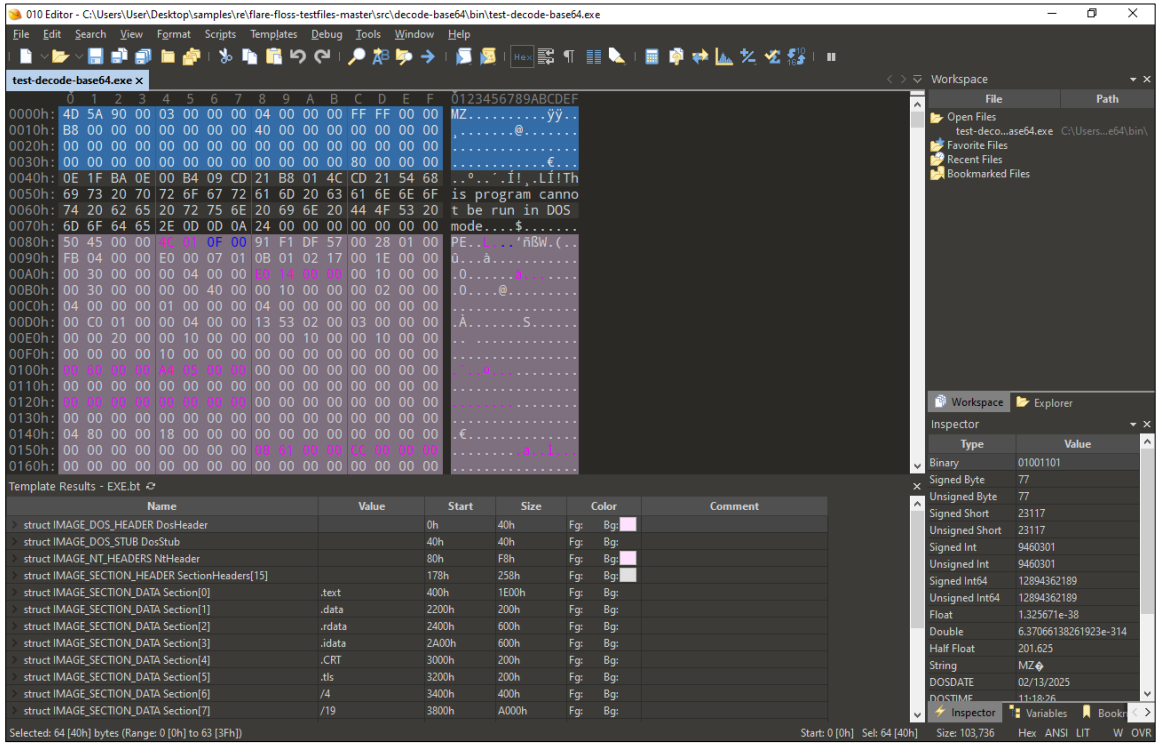
Determining File Type

Identifying File Type

Using Manual Method

- **Look for the File Signature through Hex Editors**
  - HxD, 010 Editor, fileinsight, etc.





Identifying File Type

Using Automated Tools

- **Use File Identification Tools**
  - *file* utility

```
λ file flare-floss-testfiles-master.zip
flare-floss-testfiles-master.zip: Zip archive data, at least v1.0 to extract

λ file test-decode-base64
test-decode-base64: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.24,
BuildID[sha1]=04c1040c926688e0b65067b26ed5359f3830ec0e, not stripped

λ file test-decode-base64.exe
test-decode-base64.exe: PE32 executable (console) Intel 80386, for MS Windows

λ file test-decode-base6464.exe
test-decode-base6464.exe: PE32+ executable (console) x86-64, for MS Windows
```

https://github.com/mandiant/flare-floss-testfiles

Fingerprinting Files

Fingerprinting Files

Generating Cryptographic Hash

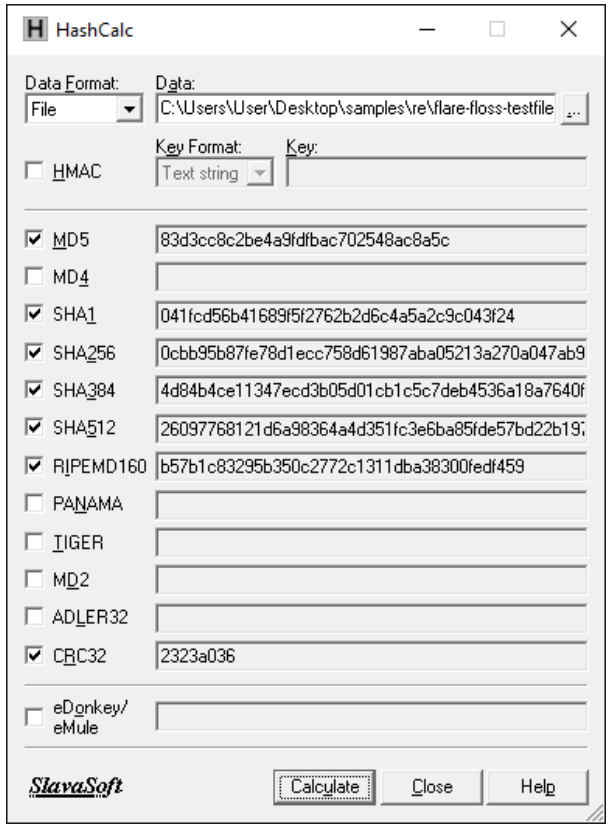
- **Hash Tools**
  - md5sum, sha1sum, sha256sum, sha512sum, etc.

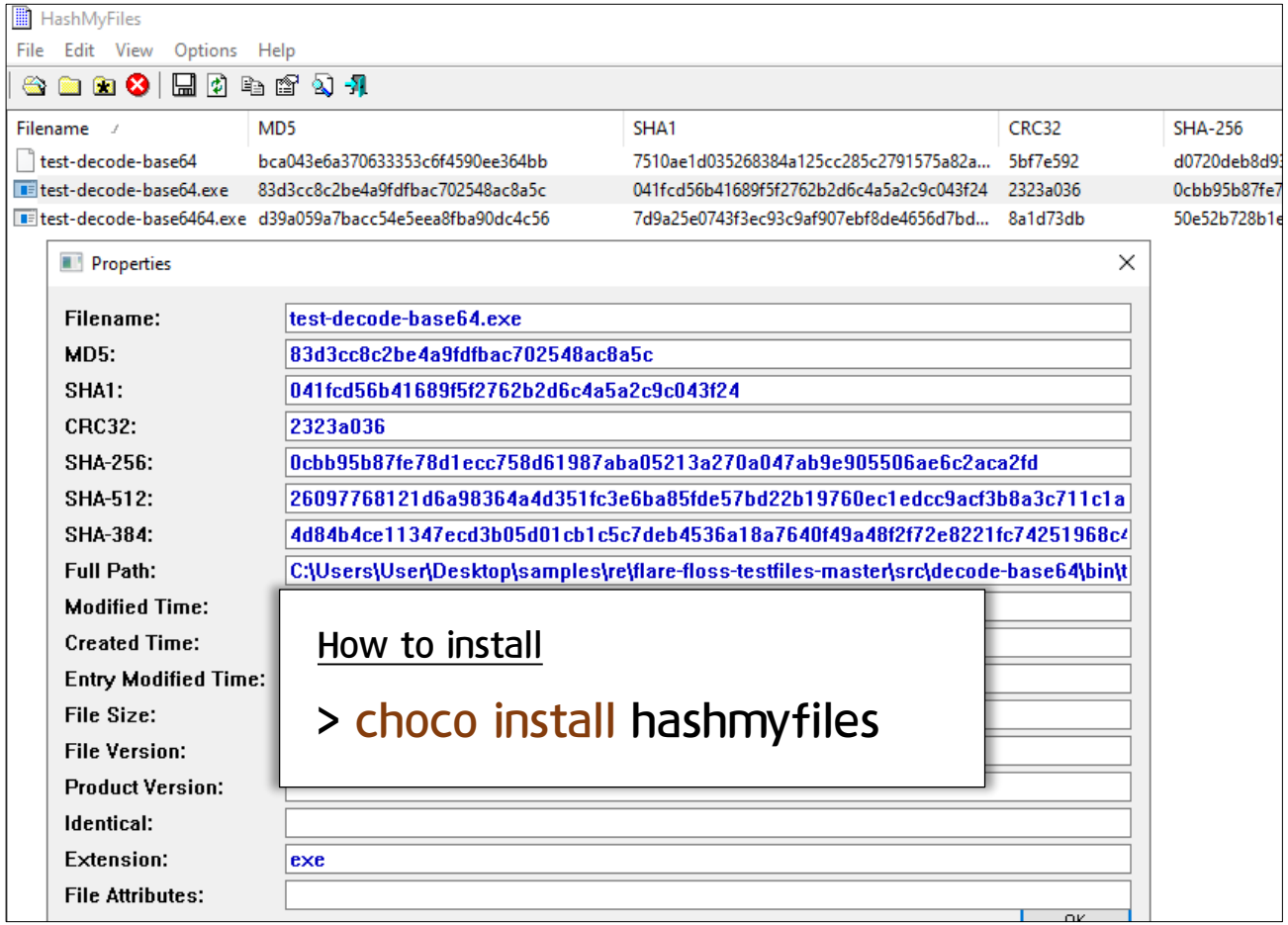
```
λ md5sum test-decode-base64.exe
83d3cc8c2be4a9fdfbac702548ac8a5c *test-decode-base64.exe

λ sha1sum test-decode-base64.exe
041fcd56b41689f5f2762b2d6c4a5a2c9c043f24 *test-decode-base64.exe

λ sha256sum test-decode-base64.exe
0cbb95b87fe78d1ecc758d61987aba05213a270a047ab9e905506ae6c2aca2fd *test-decode-base64.exe

λ sha512sum test-decode-base64.exe
26097768121d6a98364a4d351fc3e6ba85fde57bd22b19760ec1edcc9acf3b8a3c711c1a957fc42b4dc10e601d1b4081d811f
3f05da584acf974e33ffac23235 *test-decode-base64.exe
```

Fingerprinting Files

Generating Cryptographic Hash

- **Hash Tools**
  - HashCalc, HashMyFiles, etc.



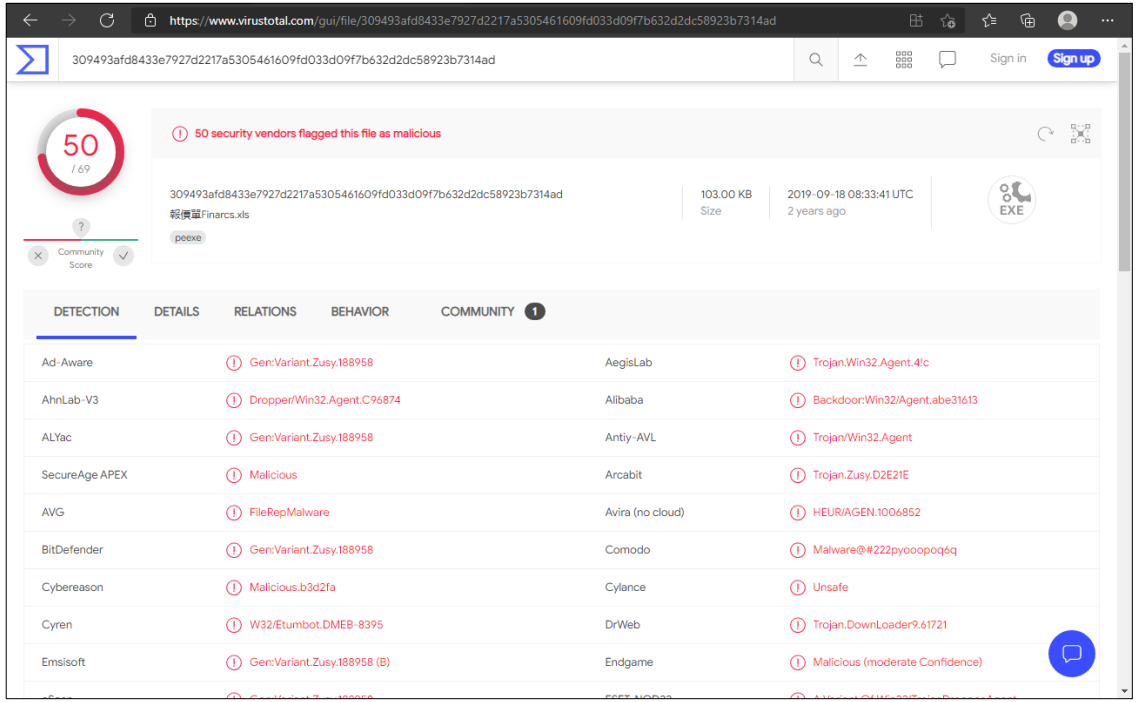


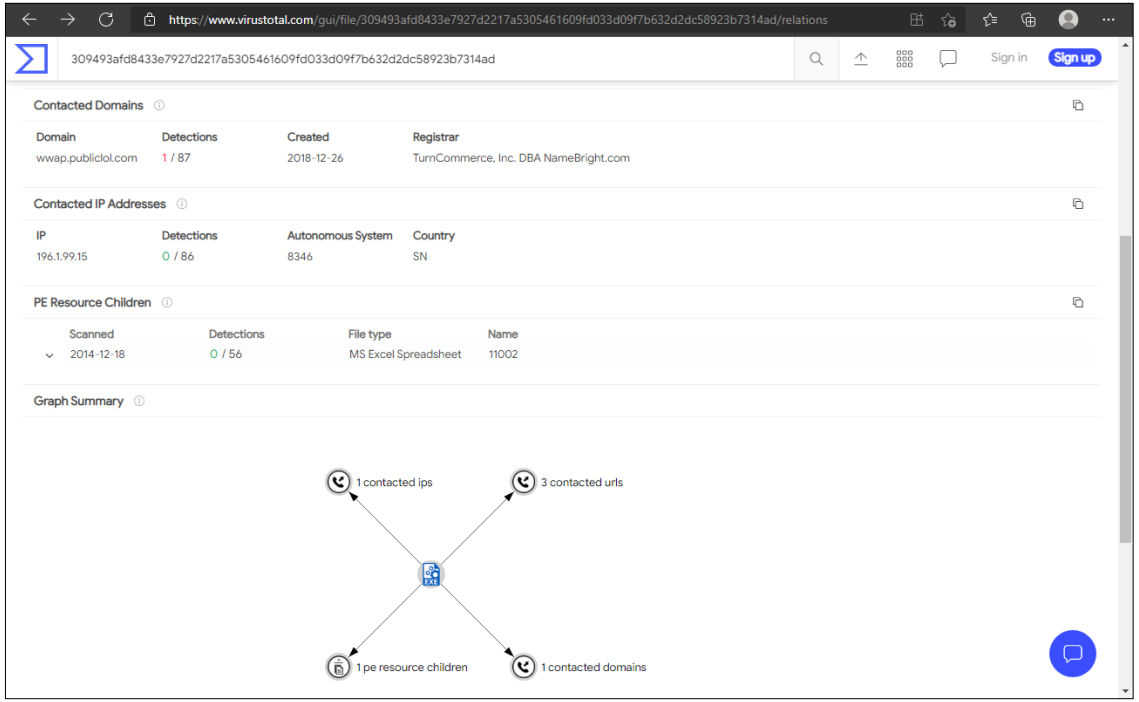
Multiple Anti-Virus Scanning

Multiple Anti-Virus Scanning

Scanning the Suspect Binary with VirusTotal

- **VirusTotal (www.virustotal.com)**
  - Web-based malware scanning service

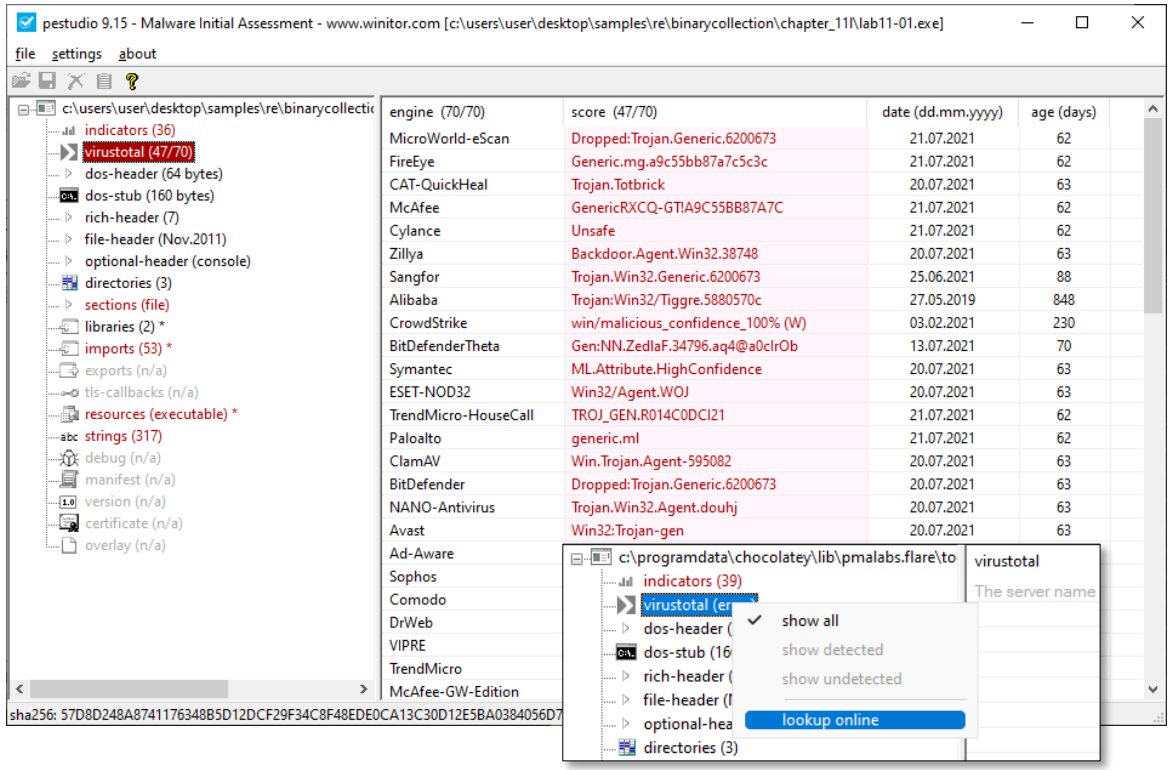


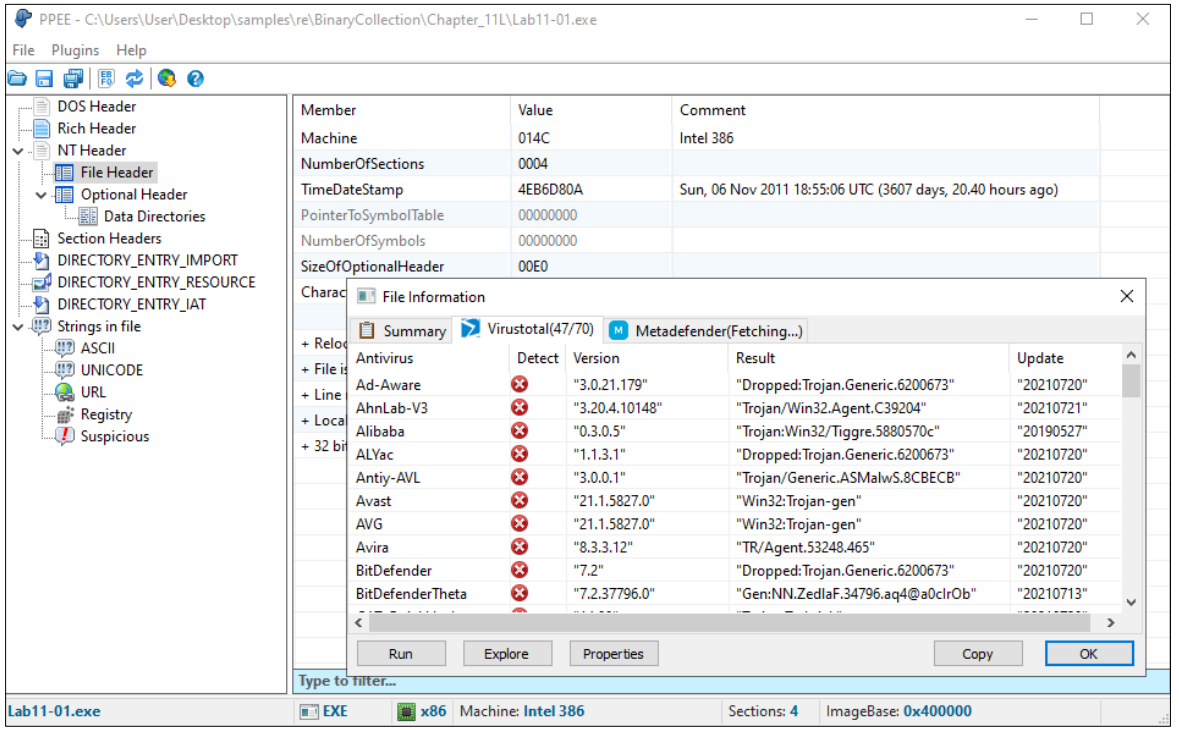


Multiple Anti-Virus Scanning

Querying Hash Values Using VirusTotal API

- **VirusTotal (www.virustotal.com)**
  - Web-based malware scanning service





Extracting Strings

Extracting Strings

String Extraction Using Tools

- **Sample Codes and Binaries**

```
_wsetlocale(LC_ALL, L"Korean");

CHAR  cbuffer1[] = "Hello, Reverse Engineering.";
CHAR* cbuffer2 = (CHAR*)malloc(48 * sizeof(CHAR));

if (cbuffer2 != nullptr) {
   ZeroMemory(cbuffer2, 48 * sizeof(CHAR));
   ::strcpy_s(cbuffer2, _countof(cbuffer1), cbuffer1);
}

WCHAR  wbuffer1[48] = L"리버싱이 적성에 잘 맞나요?";
WCHAR* wbuffer2 = (WCHAR*)malloc(48 * sizeof(WCHAR));

if (wbuffer2 != nullptr) {
   ZeroMemory(wbuffer2, 48 * sizeof(WCHAR));
   ::wcscpy_s(wbuffer2, 48, wbuffer1);
}

printf("CHAR  buffer1: %s (stack)\n", cbuffer1);
printf("CHAR  buffer2: %s (heap) \n", cbuffer2);
wprintf(L"WCHAR buffer1: %s (stack)\n", wbuffer1);
wprintf(L"WCHAR buffer2: %s (heap) \n", wbuffer2);

free(cbuffer2); free(wbuffer2);
```

Extracting Strings

String Extraction Using Tools

- **Sample Codes and Binaries**

```
λ ls -al

-rwxrwxrwx   1 user     group       63488        (0x03)-Basic-Static-Analysis-1-(x64-debug).exe
-rwxrwxrwx   1 user     group       11776        (0x03)-Basic-Static-Analysis-1-(x64-release).exe
-rwxrwxrwx   1 user     group       40960        (0x03)-Basic-Static-Analysis-1-(x86-debug).exe
-rwxrwxrwx   1 user     group       10240        (0x03)-Basic-Static-Analysis-1-(x86-release).exe
```

```
λ "(0x03)-Basic-Static-Analysis-1-(x86-release).exe"

CHAR  buffer1: Hello, Reverse Engineering. (stack)
CHAR  buffer2: Hello, Reverse Engineering. (heap)
WCHAR buffer1: 리버싱이 적성에 잘 맞나요? (stack)
WCHAR buffer2: 리버싱이 적성에 잘 맞나요? (heap)
```

Extracting Strings

String Extraction Using Tools

- ***string*** **utility**
  - -a : extracting ASCII strings

```
λ "(0x03)-Basic-Static-Analysis-1-(x86-release).exe"

CHAR  buffer1: Hello, Reverse Engineering. (stack)
CHAR  buffer2: Hello, Reverse Engineering. (heap)
WCHAR buffer1: 리버싱이 적성에 잘 맞나요? (stack)
WCHAR buffer2: 리버싱이 적성에 잘 맞나요? (heap)
```

```
λ strings -a "(0x03)-Basic-Static-Analysis-1-(x86-release).exe"

!This program cannot be run in DOS mode.
Richc
.text
`.rdata
@.data
.rsrc
……
=p3@
h0@
Hello, Reverse Engineering.
CHAR  buffer1: %s (stack)
CHAR  buffer2: %s (heap)
@#@
……
```

Extracting Strings

String Extraction Using Tools

- ***string*** **utility**
  - -u : extracting UNICODE strings

```
λ "(0x03)-Basic-Static-Analysis-1-(x86-release).exe"

CHAR  buffer1: Hello, Reverse Engineering. (stack)
CHAR  buffer2: Hello, Reverse Engineering. (heap)
WCHAR buffer1: 리버싱이 적성에 잘 맞나요? (stack)
WCHAR buffer2: 리버싱이 적성에 잘 맞나요? (heap)
```

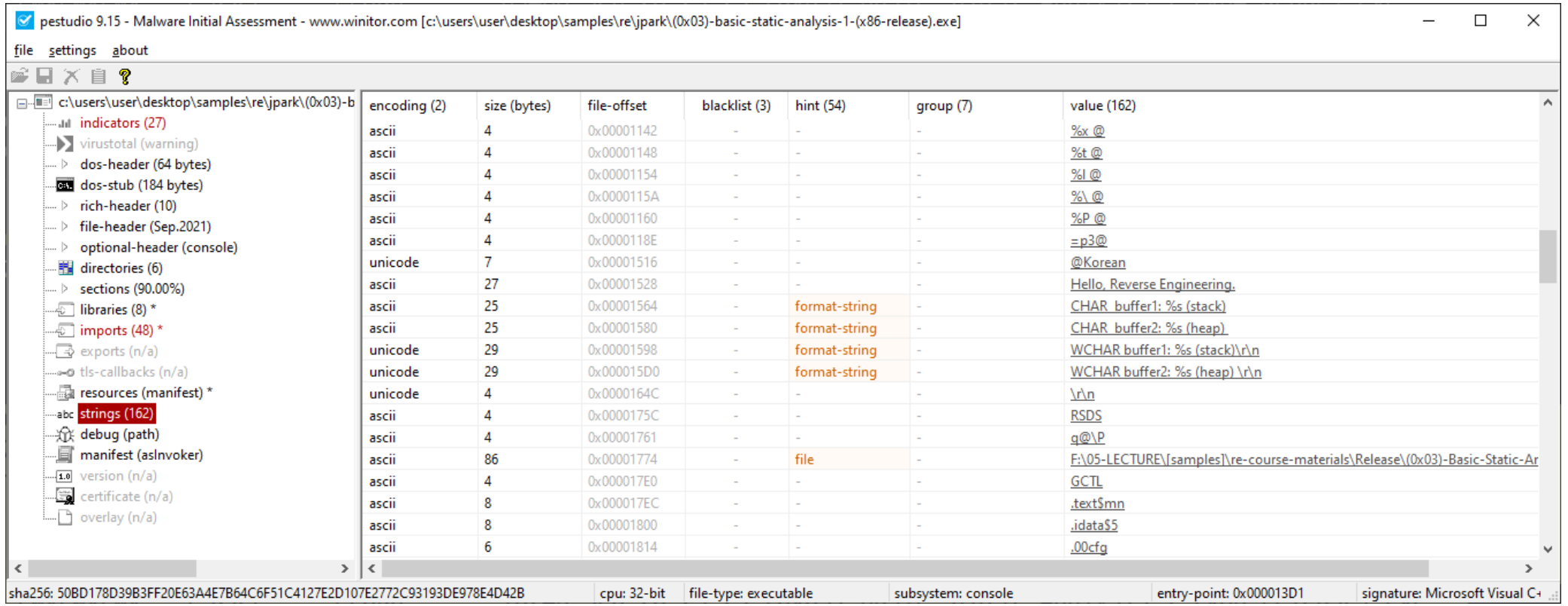
```
λ strings -u -o "(0x03)-Basic-Static-Analysis-1-(x86-release).exe"

1027:@
2183:@
2299:U
2642:<@
2726:
2757:U
3113:@
3717:
5398:@Korean
5442:.
5464:
5532:WCHAR buffer1: %s (stack)
5588:WCHAR buffer2: %s (heap)
```

Extracting Strings

String Extraction Using Tools

- **PE related Tools**
  - pestudio, PPEE, etc.

```
λ "(0x03)-Basic-Static-Analysis-1-(x86-release).exe"

CHAR  buffer1: Hello, Reverse Engineering. (stack)
CHAR  buffer2: Hello, Reverse Engineering. (heap)
WCHAR buffer1: 리버싱이 적성에 잘 맞나요? (stack)
WCHAR buffer2: 리버싱이 적성에 잘 맞나요? (heap)
```



Extracting Strings

Decoding Obfuscated Strings Using FLOSS

- **FLOSS ?**
  - FireEye Labs Obfuscated String Solver (https://github.com/mandiant/flare-floss)
    - Uses advanced static analysis techniques to automatically deobfuscate strings from binaries
    - Official test files (https://github.com/mandiant/flare-floss-testfiles)

```
λ floss -n 10 "(0x03)-Basic-Static-Analysis-1-(x86-release).exe"

FLOSS static ASCII strings
!This program cannot be run in DOS mode.
Hello, Reverse Engineering.
CHAR  buffer1: %s (stack)
CHAR  buffer2: %s (heap)
……
FLOSS static Unicode strings
WCHAR buffer1: %s (stack)
WCHAR buffer2: %s (heap)
……
FLOSS extracted 1 stackstrings
Hello, Reverse Engineering.
```

Extracting Strings

Decoding Obfuscated Strings Using FLOSS

- **Sample 1: decode-base64**

```c
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stdlib.h>

#include "base64.h"


char in[] = "aGVsbG8gd29ybGQ=";

int main(int argc, char **argv) {
    char out[sizeof(in) + 1];
    Base64decode(out, in);
    printf("%s\n", out);
    return 0;
}
```

```
λ floss test-decode-base64.exe

FLOSS static ASCII strings
!This program cannot be run in DOS mode.
.text
P`.data
……
__imp__EnterCriticalSection@4
__imp__fwrite

FLOSS static Unicode strings

FLOSS decoded 1 strings
hello world

FLOSS extracted 0 stackstrings
```

Extracting Strings

Decoding Obfuscated Strings Using FLOSS

- **Sample 2: decode-single-byte-xor**

```c
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    char in[] = "idmmn!vnsme";
    char out[12] = { 0 };
    unsigned char key = 0x1;

    if (decode(out, sizeof(out), in, sizeof(in), 0x1)) {
        perror("failed to decode.\n");
        return -1;
    }
    printf("%s\n", out);
    return 0;
}
```

```
λ floss test-decode-single-byte-xor.exe

FLOSS static ASCII strings
!This program cannot be run in DOS mode.
.text
P`.data
……
__imp__fwrite

FLOSS static Unicode strings

FLOSS decoded 2 strings
hello world
csec

FLOSS extracted 1 stackstrings
idmmn!vnsme
```

Extracting Strings

Decoding Obfuscated Strings Using FLOSS

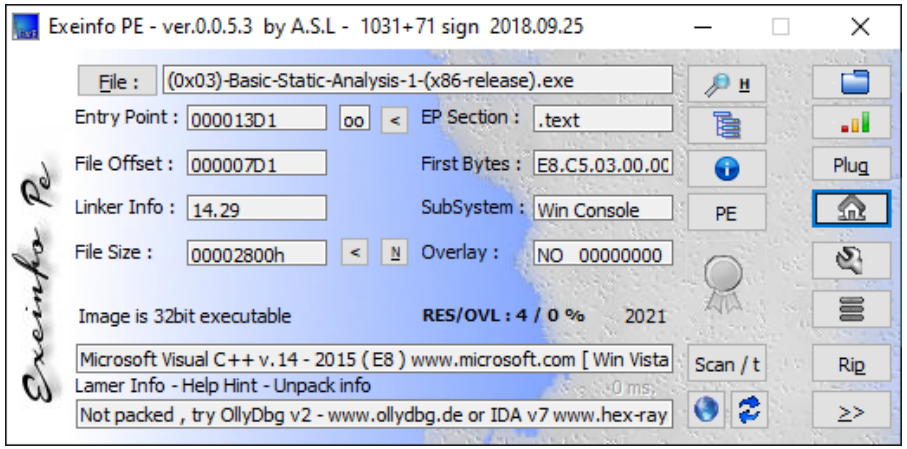
- **[REF] How FLOSS works ?**
  - Analyze program to identify data, code, functions, basic blocks, cross-references, etc.
    - Uses vivisect to disassemble and analyze the control flow of a program
  - Use heuristics to find potential decoding routines
  - Brute-force emulate all code paths among basic blocks and functions
  - Snapshot emulator state (registers, memory) at appropriate points
  - Emulate decoder functions using emulator state snapshots
  - Compare memory state from before and after function emulation
  - Extract human-readable strings from memory state difference

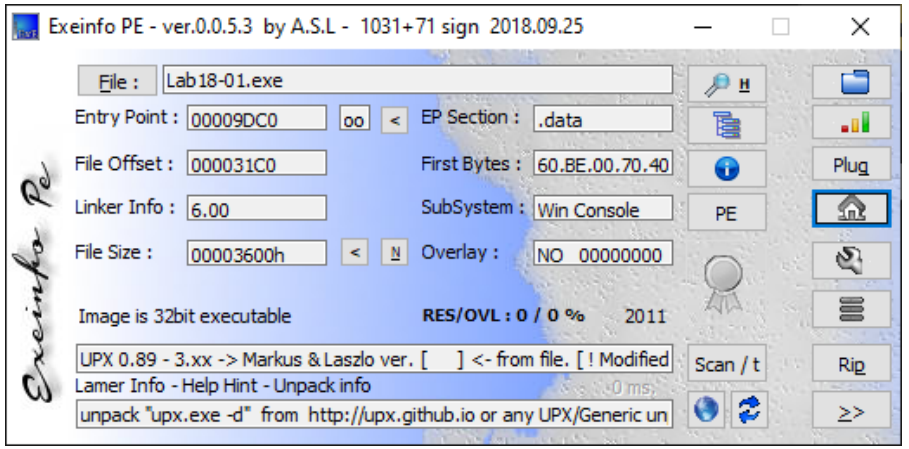
Checking PE File’s Properties

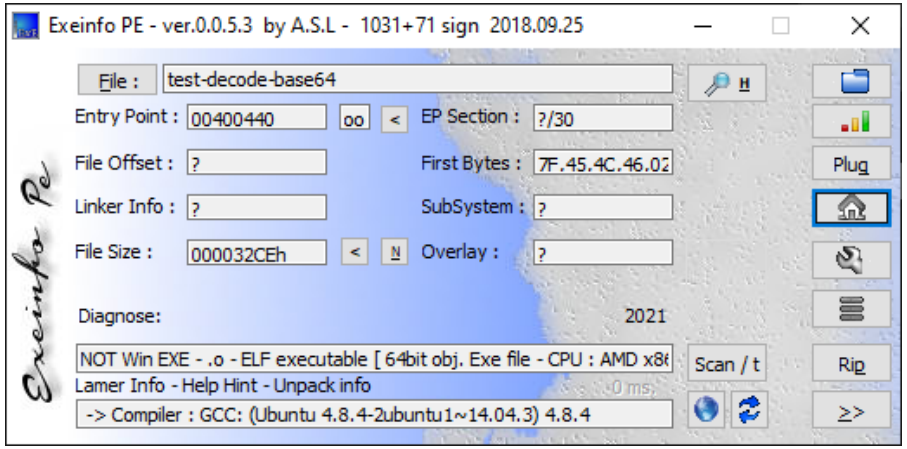
Checking PE File's Properties

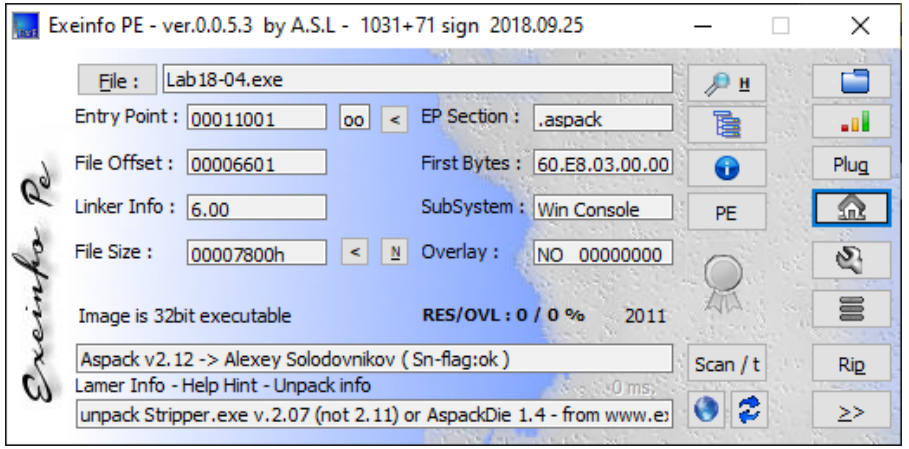
Detecting Compilers and Packers

- **Exeinfo PE (http://www.exeinfo.xn.pl)**









Understanding PE File Format

Understanding PE File Format

PE File Format

- **PE ?**
  - Portable Executable
  - An executable file format supported since Windows 3.1
  - Based on COFF(Common Object File Format) of Unix

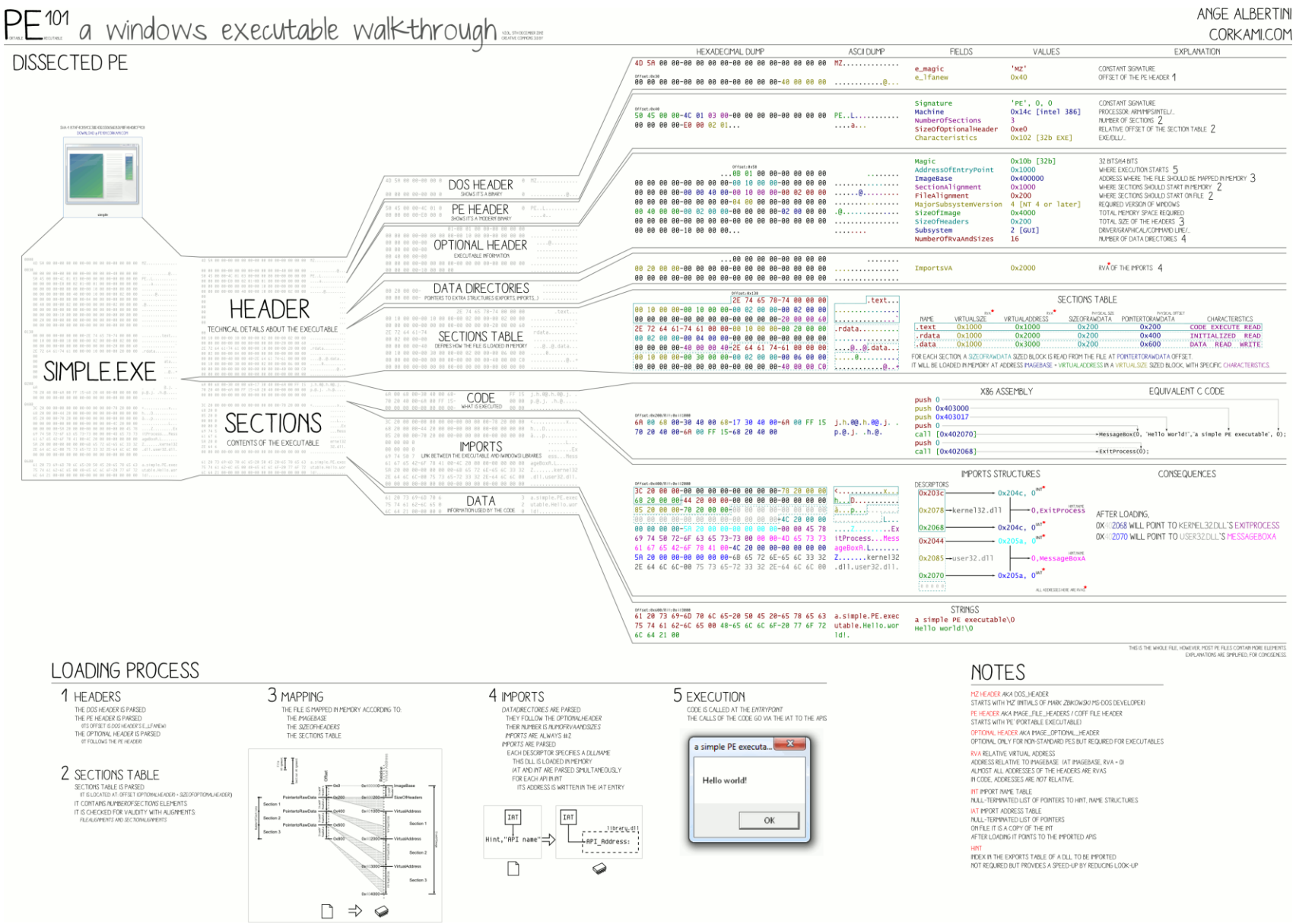
- **Extensions of Windows PE Files**
  - Executable File: EXE, SCR
  - Library: DLL, OCX
  - Driver: SYS
  - Object File: OBJ

Understanding PE File Format

PE File Format

- **Understanding Concepts of VA, RVA & File Offset**
  - **VA** (Virtual Address) : An absolute address in a virtual memory space of each process
  - **RVA** (Relative Virtual Address) : A relative address starting from a reference position (`ImageBase`)
    - Used in a PE file format for indicating a position
  - **VA = `ImageBase` + RVA**
  - **File Offset (RAW)** : A physical offset in a PE file
  - **RVA ↔ RAW**

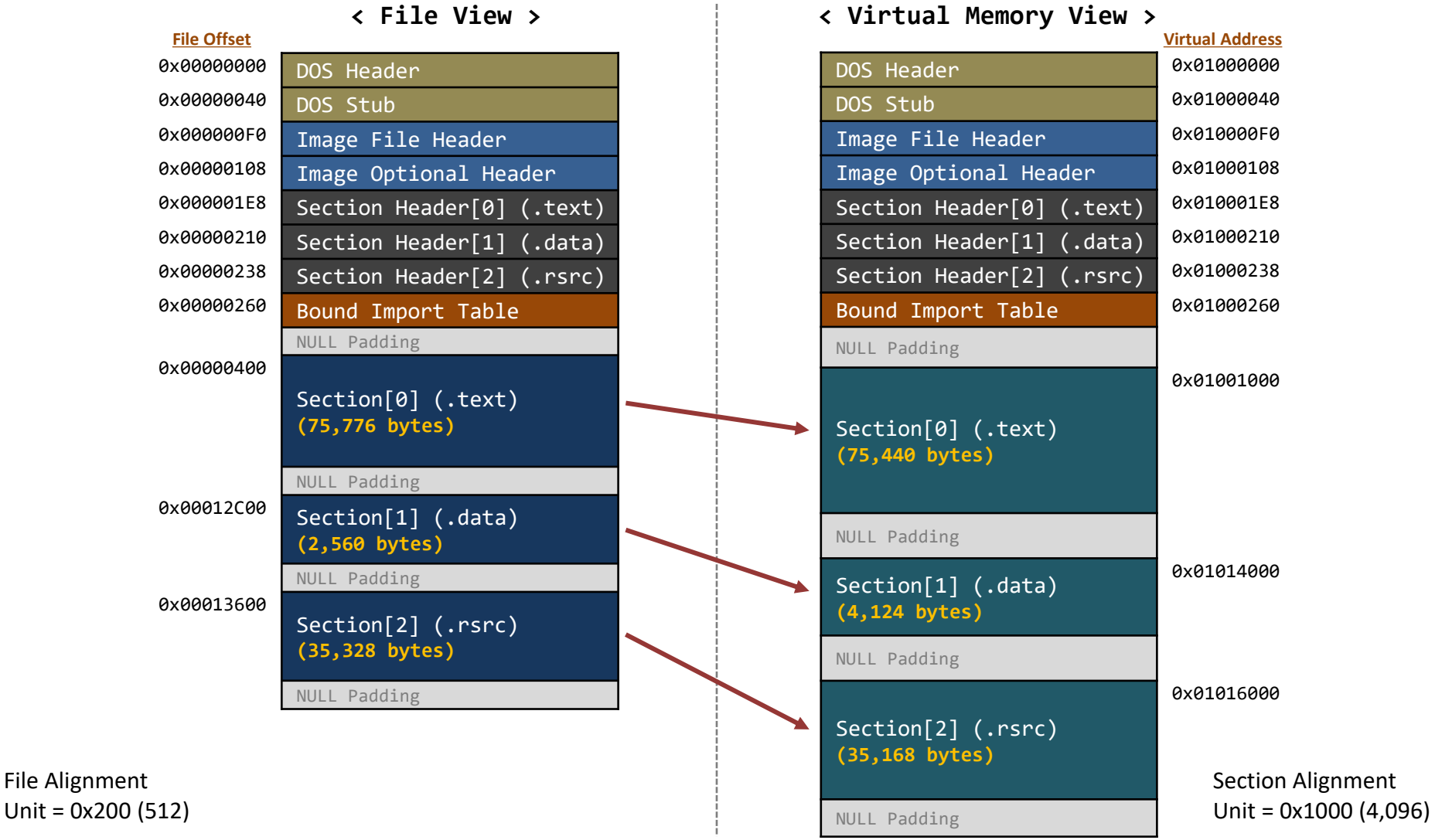
Understanding PE File Format



https://github.com/corkami/pics/blob/master/binary/pe101/README.md

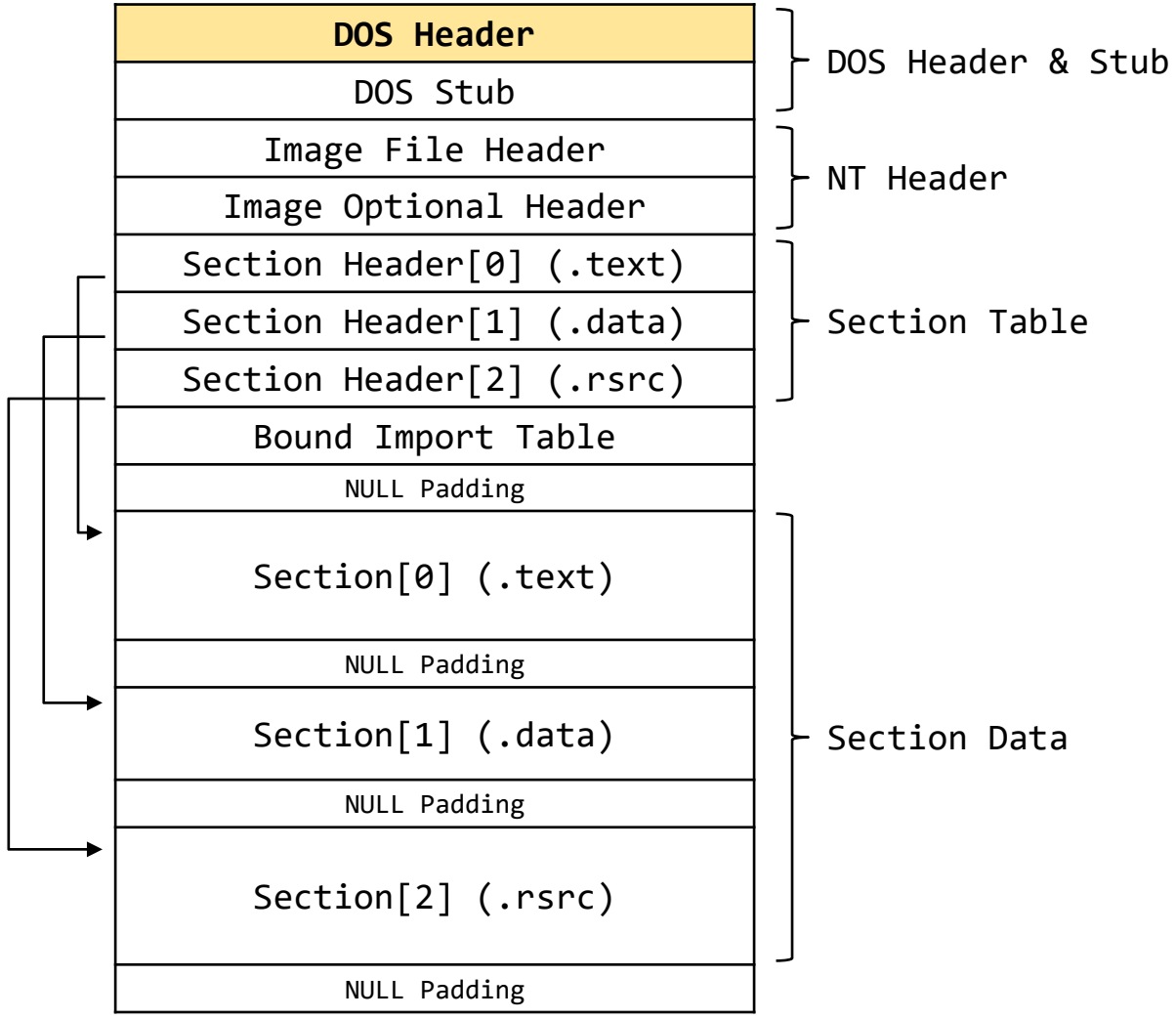
Understanding PE File Format

Overall Structure of PE File Format



Understanding PE File Format

DOS Header

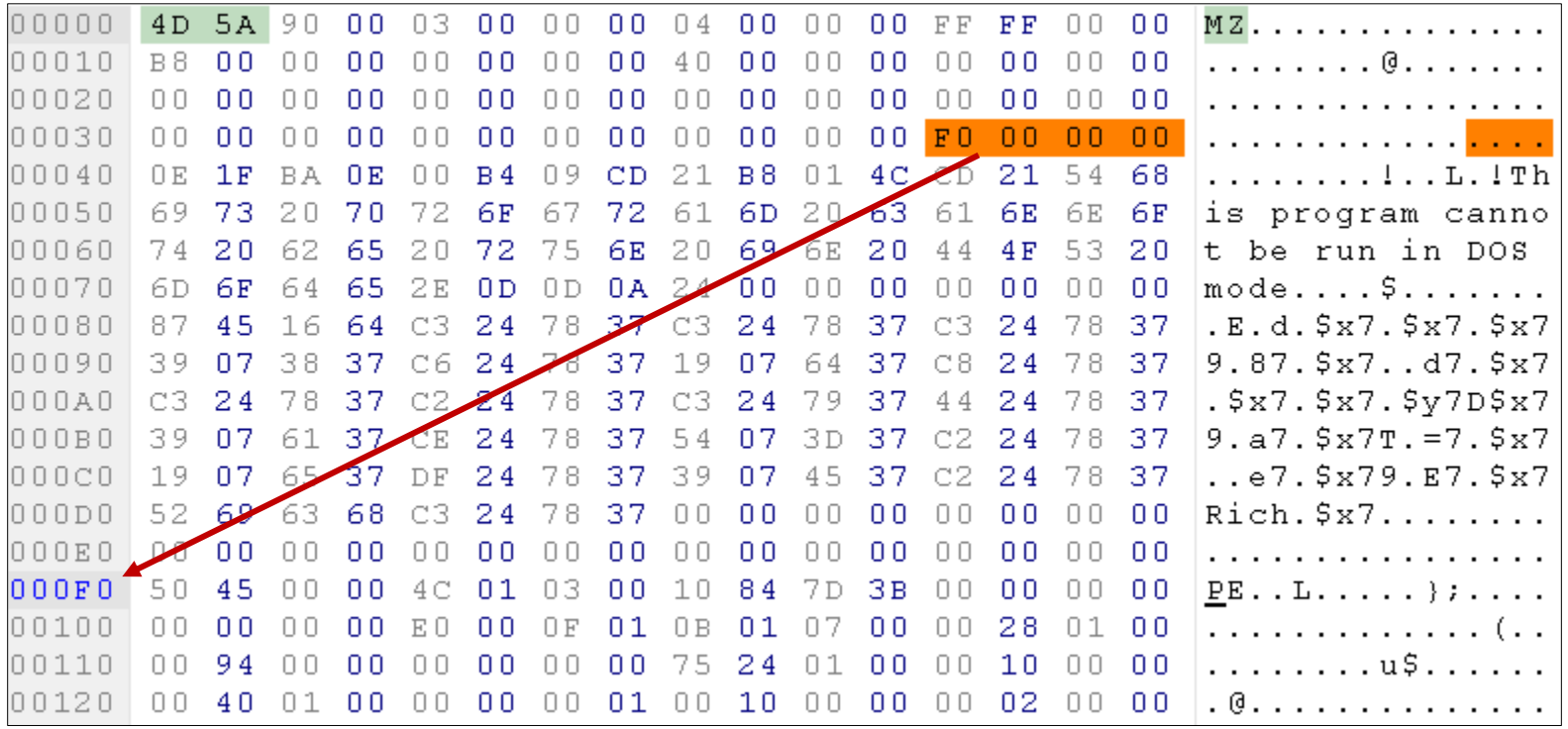


- **DOS Header** [0x40, 64 bytes]
  - Only used for backward compatibility
  - IMAGE_DOS_HEADER struct (defined in winnt.h)

```
typedef struct _IMAGE_DOS_HEADER {
   WORD  e_magic;     /* 00: 'MZ' Header signature */
   WORD  e_cblp;      /* 02: Bytes on last page of file */
   WORD  e_cp;        /* 04: Pages in file */
   WORD  e_crlc;      /* 06: Relocations */
   WORD  e_cparhdr;   /* 08: Size of header in paragraphs */
   WORD  e_minalloc;  /* 0a: Minimum extra paragraphs needed */
   WORD  e_maxalloc;  /* 0c: Maximum extra paragraphs needed */
   WORD  e_ss;        /* 0e: Initial (relative) SS value */
   WORD  e_sp;        /* 10: Initial SP value */
   WORD  e_csum;      /* 12: Checksum */
   WORD  e_ip;        /* 14: Initial IP value */
   WORD  e_cs;        /* 16: Initial (relative) CS value */
   WORD  e_lfarlc;    /* 18: File address of relocation table */
   WORD  e_ovno;      /* 1a: Overlay number */
   WORD  e_res[4];    /* 1c: Reserved words */
   WORD  e_oemid;     /* 24: OEM identifier (for e_oeminfo) */
   WORD  e_oeminfo;   /* 26: OEM information; e_oemid specific */
   WORD  e_res2[10];  /* 28: Reserved words */
   DWORD e_lfanew;    /* 3c: Offset to extended header */
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

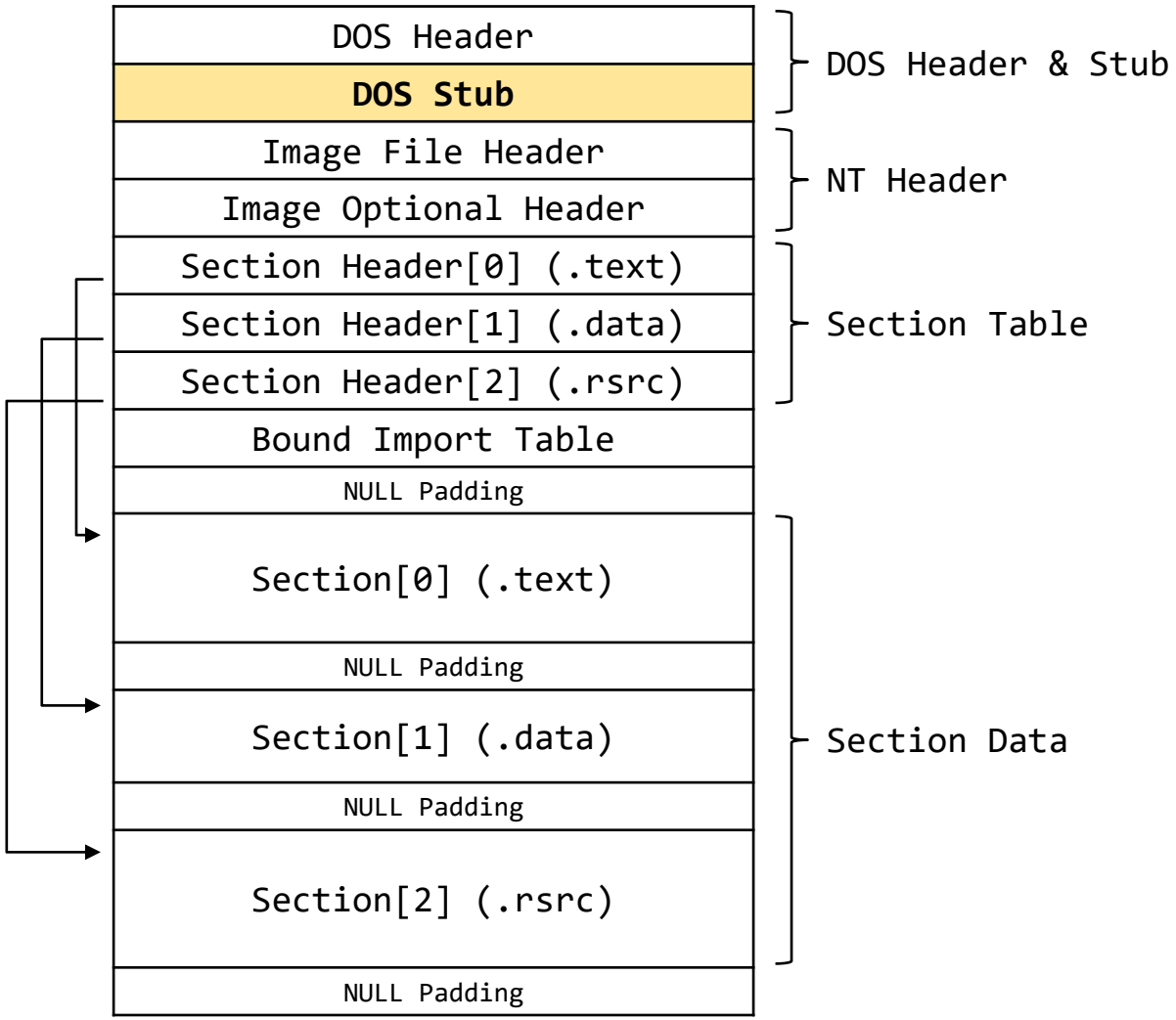
Understanding PE File Format

DOS Header



Understanding PE File Format

DOS Stub

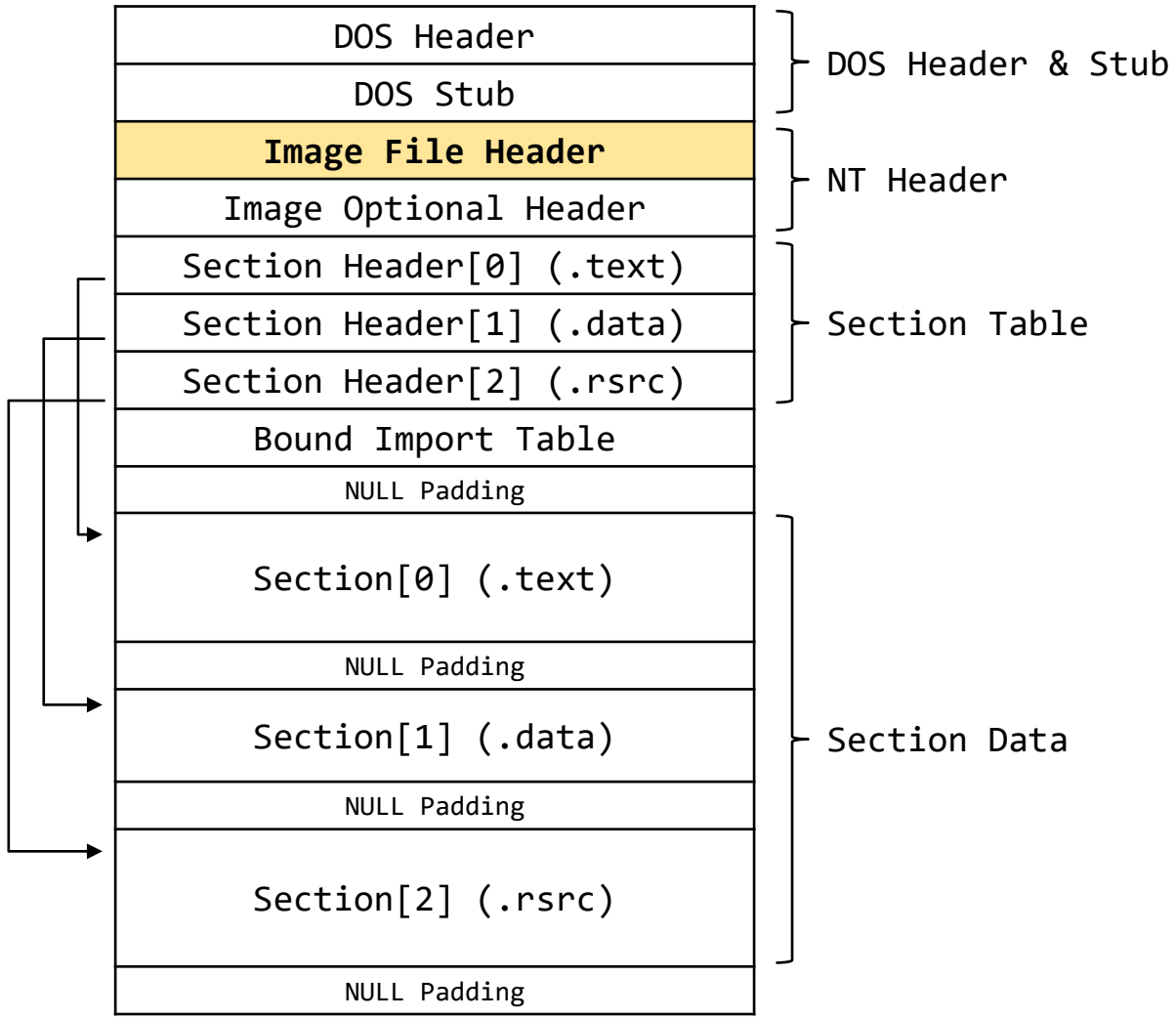


- **DOS Stub**
  - Executable codes and data for DOS
  - String data: 'This program cannot be run in DOS mode'

```
00000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..............
00010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
00020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00030  00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00  ................
00040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ........!..L.!Th
00050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00080  87 45 16 64 C3 24 78 37 C3 24 78 37 C3 24 78 37  .E.d.$x7.$x7.$x7
00090  39 07 38 37 C6 24 78 37 19 07 64 37 C8 24 78 37  9.87.$x7..d7.$x7
000A0  C3 24 78 37 C2 24 78 37 C3 24 79 37 44 24 78 37  .$x7.$x7.$y7D$x7
000B0  39 07 61 37 CE 24 78 37 54 07 3D 37 C2 24 78 37  9.a7.$x7T.=7.$x7
000C0  19 07 65 37 DF 24 78 37 39 07 45 37 C2 24 78 37  ..e7.$x79.E7.$x7
000D0  52 69 63 68 C3 24 78 37 00 00 00 00 00 00 00 00  Rich.$x7........
000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000F0  50 45 00 00 4C 01 03 00 10 84 7D 3B 00 00 00 00  PE..L.....};....
00100  00 00 00 00 E0 00 0F 01 0B 01 07 00 00 28 01 00  .............(..
00110  00 94 00 00 00 00 00 00 75 24 01 00 00 10 00 00  ........u$......
00120  00 40 01 00 00 00 00 01 00 10 00 00 00 02 00 00  .@..............
```

Understanding PE File Format

NT Header – Image File Header



- **NT Header (32 bits)**

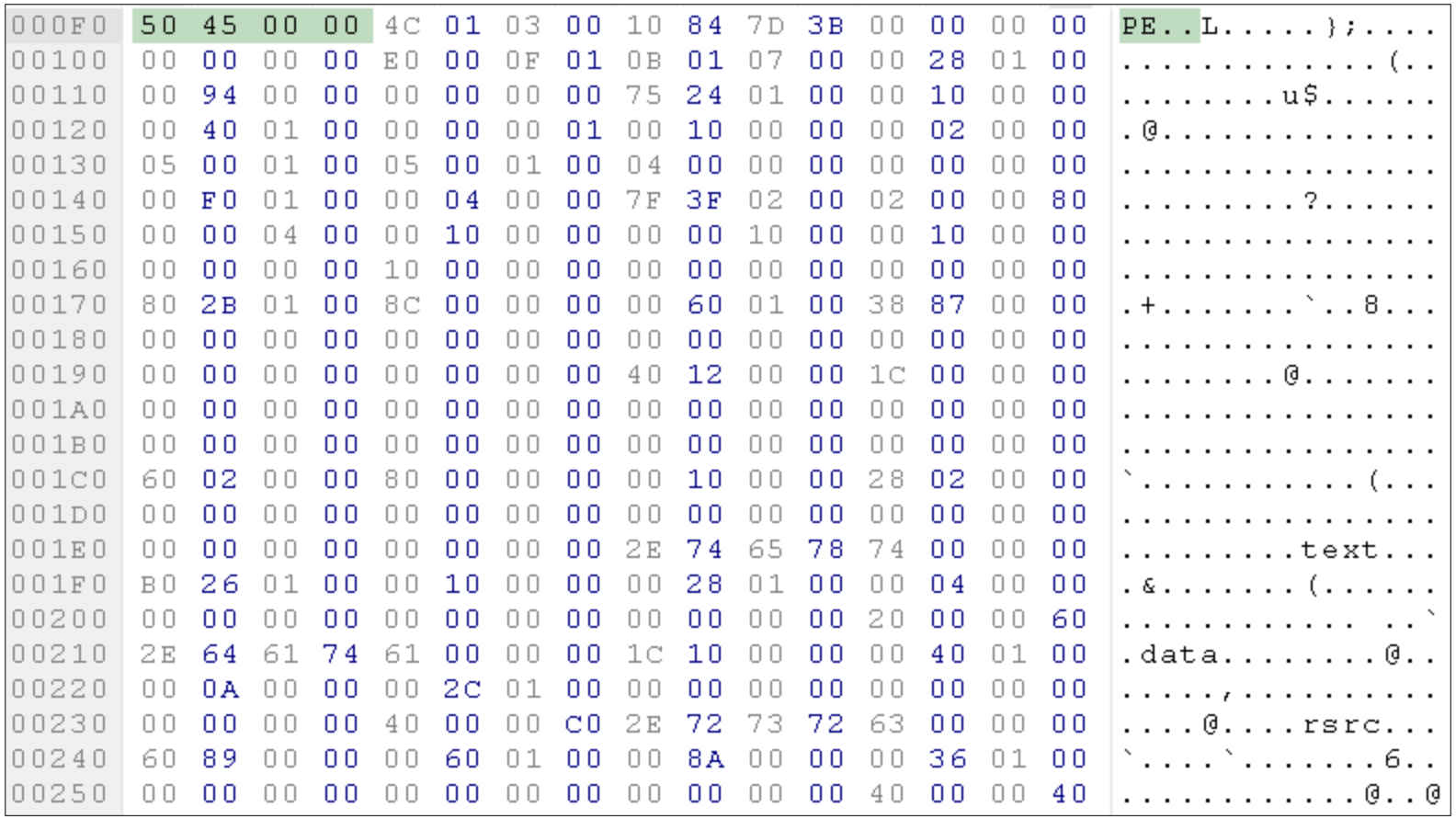
```
typedef struct _IMAGE_NT_HEADERS {

  DWORD Signature;
  IMAGE_FILE_HEADER FileHeader;
  IMAGE_OPTIONAL_HEADER OptionalHeader;

} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

- **Signature**: 4 bytes ------- 0x50450000 (‘PE’00)
- **Image File Header**: 0x14 (20) bytes
- **Image Optional Header**: 0xE0 (224) bytes
- **Total Size**: 0xF8 (248) bytes

Understanding PE File Format

NT Header – Image File Header

- **Signature:** 4 bytes -------- 0x50450000 (‘PE’00)



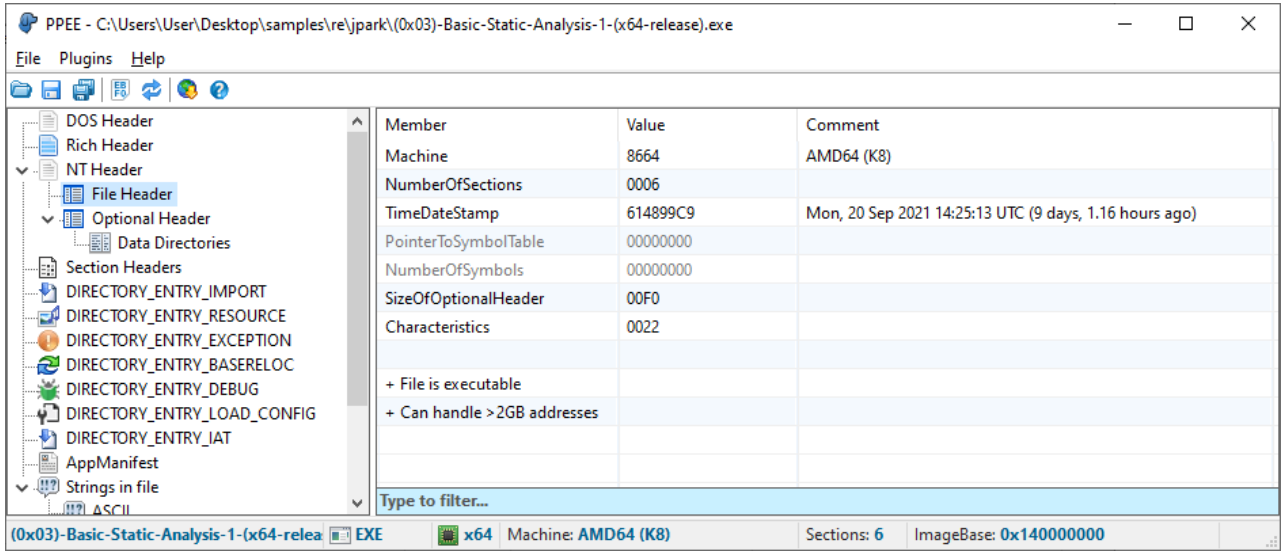
```
000F0  50 45 00 00 4C 01 03 00 10 84 7D 3B 00 00 00 00  PE..L.....};....
00100  00 00 00 00 E0 00 0F 01 0B 01 07 00 00 28 01 00  .............(..
00110  00 94 00 00 00 00 00 00 75 24 01 00 00 10 00 00  ........u$......
00120  00 40 01 00 00 00 00 01 00 10 00 00 00 02 00 00  .@..............
00130  05 00 01 00 05 00 01 00 04 00 00 00 00 00 00 00  ................
00140  00 F0 01 00 00 04 00 00 7F 3F 02 00 02 00 00 80  .........?......
00150  00 00 04 00 00 10 00 00 00 00 10 00 00 10 00 00  ................
00160  00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00  ................
00170  80 2B 01 00 8C 00 00 00 00 60 01 00 38 87 00 00  .+.......`..8...
00180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00190  00 00 00 00 00 00 00 00 40 12 00 00 1C 00 00 00  ........@.......
001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
001C0  60 02 00 00 80 00 00 00 00 10 00 00 28 02 00 00  `...........(...
001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
001E0  00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00  .........text...
001F0  B0 26 01 00 00 10 00 00 00 28 01 00 00 04 00 00  .&.......(......
00200  00 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00  ............ ..`
00210  2E 64 61 74 61 00 00 00 1C 10 00 00 00 40 01 00  .data........@..
00220  00 0A 00 00 00 2C 01 00 00 00 00 00 00 00 00 00  .....,..........
00230  00 00 00 00 40 00 00 C0 2E 72 73 72 63 00 00 00  ....@....rsrc...
00240  60 89 00 00 00 60 01 00 00 8A 00 00 00 36 01 00  `....`.......6..
00250  00 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00  .............@..@
```

Understanding PE File Format

NT Header – Image File Header

- **Image File Header**: 0x14 (20) bytes

```
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;
```



- **Machine**: CPU architecture
- **NumberOfSections**: Number of sections within this PE file (greater than 0)
- **TimeDateStamp**: Compiled datetime (UNIXTIME, UTC)
- **SizeOfOptionalHeader**: Size of Image Optional Header
- **Characteristics**: Characteristics of this PE file

Understanding PE File Format

NT Header – Image File Header

- **IMAGE_FILE_HEADER.Machine**

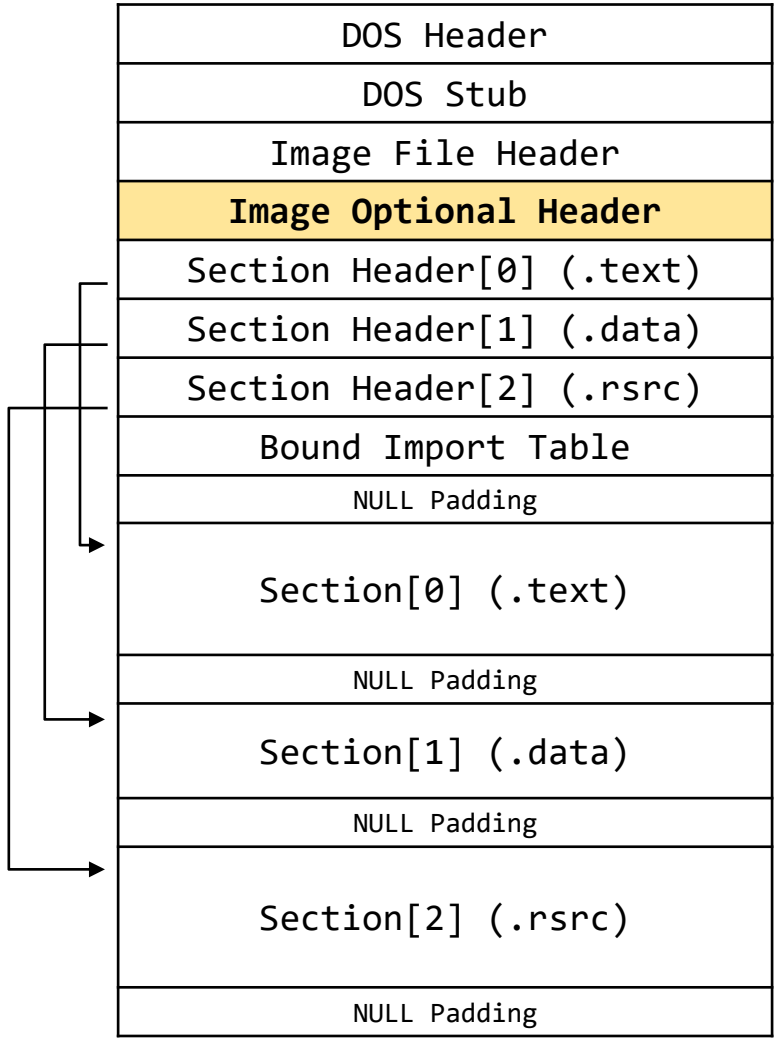
```
#define  IMAGE_FILE_MACHINE_UNKNOWN        0
#define  IMAGE_FILE_MACHINE_I386           0x014c // Intel 386.
#define  IMAGE_FILE_MACHINE_R3000          0x0162 // MIPS little-endian
#define  IMAGE_FILE_MACHINE_R4000          0x0166 // MIPS little-endian
#define  IMAGE_FILE_MACHINE_R10000         0x0168 // MIPS little-endian
#define  IMAGE_FILE_MACHINE_WCEMIPSV2      0x0169 // MIPS little-endian WCE v2
#define  IMAGE_FILE_MACHINE_ALPHA          0x0184 // Alpha_AXP
#define  IMAGE_FILE_MACHINE_SH3            0x01a2 // SH3 little-endian
#define  IMAGE_FILE_MACHINE_SH3DSP         0x01a3
#define  IMAGE_FILE_MACHINE_SH3E           0x01a4 // SH3E little-endian
#define  IMAGE_FILE_MACHINE_SH4            0x01a6 // SH4 little-endian
#define  IMAGE_FILE_MACHINE_SH5            0x01a8 // SH5
#define  IMAGE_FILE_MACHINE_ARM            0x01c0 // ARM Little-Endian
#define  IMAGE_FILE_MACHINE_THUMB          0x01c2
#define  IMAGE_FILE_MACHINE_AM33           0x01d3
#define  IMAGE_FILE_MACHINE_POWERPC        0x01F0 // IBM PowerPC Little-Endian
#define  IMAGE_FILE_MACHINE_POWERPCFP      0x01f1
#define  IMAGE_FILE_MACHINE_IA64           0x0200 // Intel 64
#define  IMAGE_FILE_MACHINE_MIPS16         0x0266 // MIPS
#define  IMAGE_FILE_MACHINE_ALPHA64        0x0284 // ALPHA64
#define  IMAGE_FILE_MACHINE_MIPSFPU        0x0366 // MIPS
#define  IMAGE_FILE_MACHINE_MIPSFPU16      0x0466 // MIPS
#define  IMAGE_FILE_MACHINE_AXP64          IMAGE_FILE_MACHINE_ALPHA64
#define  IMAGE_FILE_MACHINE_TRICORE        0x0520 // Infineon
#define  IMAGE_FILE_MACHINE_CEF            0x0CEF
#define  IMAGE_FILE_MACHINE_EBC            0x0EBC // EFI Byte Code
#define  IMAGE_FILE_MACHINE_AMD64          0x8664 // AMD64 (K8)
#define  IMAGE_FILE_MACHINE_M32R           0x9041 // M32R little-endian
#define  IMAGE_FILE_MACHINE_CEE            0xC0EE
```

- **IMAGE_FILE_HEADER.Characteristics**

```
#define IMAGE_FILE_RELOCS_STRIPPED         0x0001 // Relocation info stripped from file.
#define IMAGE_FILE_EXECUTABLE_IMAGE        0x0002 // File is executable.
#define IMAGE_FILE_LINE_NUMS_STRIPPED      0x0004 // Line numbers stripped from file.
#define IMAGE_FILE_LOCAL_SYMS_STRIPPED     0x0008 // Local symbols stripped from file.
#define IMAGE_FILE_AGGRESIVE_WS_TRIM       0x0010 // Agressively trim working set.
#define IMAGE_FILE_LARGE_ADDRESS_AWARE     0x0020 // App can handle >2GB addresses.
#define IMAGE_FILE_BYTES_REVERSED_LO       0x0080 // Bytes of machine word are reversed.
#define IMAGE_FILE_32BIT_MACHINE           0x0100 // 32 bit word machine.
#define IMAGE_FILE_DEBUG_STRIPPED          0x0200 // Debugging info stripped from file.
#define IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP 0x0400 // If image is on removable media,
                                                  copy and run from the swap file.
#define IMAGE_FILE_NET_RUN_FROM_SWAP       0x0800 // If image is on Net,
                                                  copy and run from the swap file.
#define IMAGE_FILE_SYSTEM                  0x1000 // System File.
#define IMAGE_FILE_DLL                     0x2000 // File is a DLL.
#define IMAGE_FILE_UP_SYSTEM_ONLY          0x4000 // File should only be run on a UP machine.
#define IMAGE_FILE_BYTES_REVERSED_HI       0x8000 // Bytes of machine word are reversed.
```

Understanding PE File Format

NT Header – Image Optional Header




```
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[16];

} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

Understanding PE File Format

NT Header – Image Optional Header

- **Magic**
  - `#define IMAGE_NT_OPTIONAL_HDR32_MAGIC  0x10b // 32bits`
  - `#define IMAGE_NT_OPTIONAL_HDR64_MAGIC  0x20b // 64bits`
- **SizeOfCode**
  - The size of the code (.text) section
- **AddressOfEntryPoint**
  - The RVA of the entry point function
- **ImageBase**
  - The address of the image when it is loaded in memory
  - `EXE's default ImageBase: 0x00400000`
  - `DLL's default ImageBase: 0x01000000`
  - '/BASE' linker option sets a base address for the program
- **SectionAlignment**
  - The alignment of sections loaded in memory, in bytes
  - The default value is the page size for the system (generally, 0x1000 = 4096 bytes)
- **FileAlignment**
  - The alignment of the raw data of sections in the image file, in bytes
  - The default is 0x200 (512) bytes
- **SizeOfImage**
  - The size of the image, in bytes, including all headers
  - Must be a multiple of SectionAlignment
- **SizeOfHeader**
  - The combined size of DOS Header, NT Header, and Section Table
  - Must be a multiple of FileAlignment

Understanding PE File Format

NT Header – Image Optional Header

- **Subsystem**

```
#define    IMAGE_SUBSYSTEM_UNKNOWN                  0
#define    IMAGE_SUBSYSTEM_NATIVE                   1    // System Driver
#define    IMAGE_SUBSYSTEM_WINDOWS_GUI              2    // Windows GUI subsystem
#define    IMAGE_SUBSYSTEM_WINDOWS_CUI              3    // Windows CUI subsystem
#define    IMAGE_SUBSYSTEM_OS2_CUI                  5
#define    IMAGE_SUBSYSTEM_POSIX_CUI                7
#define    IMAGE_SUBSYSTEM_NATIVE_WINDOWS           8    // native Win9x driver
#define    IMAGE_SUBSYSTEM_WINDOWS_CE_GUI           9    // Windows CE subsystem
#define    IMAGE_SUBSYSTEM_EFI_APPLICATION          10
#define    IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER  11
#define    IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER       12
#define    IMAGE_SUBSYSTEM_EFI_ROM                  13
#define    IMAGE_SUBSYSTEM_XBOX                     14
#define    IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION 16
```

- **NumberOfRvaAndSizes**
  - The count of entries in the `IMAGE_DATA_DIRECTORY` array
  - Always `16` (`0x00000010`)

Understanding PE File Format

NT Header – Image Optional Header

- **DataDirectory[0] ~ DataDirectory[15]**

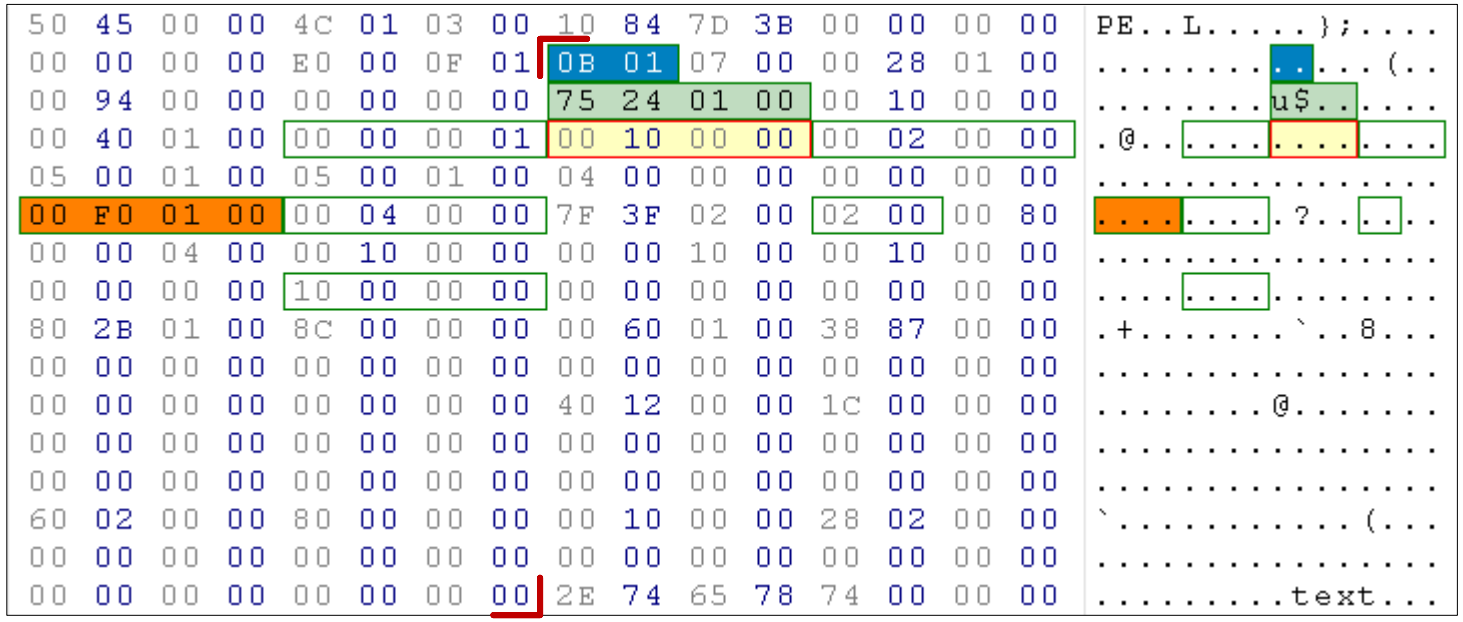
```
typedef struct _IMAGE_DATA_DIRECTORY {

        DWORD VirtualAddress;
        DWORD Size;

} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

```
#define    IMAGE_DIRECTORY_ENTRY_EXPORT              0
#define    IMAGE_DIRECTORY_ENTRY_IMPORT              1         // Import Descriptor
#define    IMAGE_DIRECTORY_ENTRY_RESOURCE            2
#define    IMAGE_DIRECTORY_ENTRY_EXCEPTION           3
#define    IMAGE_DIRECTORY_ENTRY_SECURITY            4
#define    IMAGE_DIRECTORY_ENTRY_BASERELOC           5
#define    IMAGE_DIRECTORY_ENTRY_DEBUG               6
#define    IMAGE_DIRECTORY_ENTRY_COPYRIGHT           7
#define    IMAGE_DIRECTORY_ENTRY_GLOBALPTR           8
#define    IMAGE_DIRECTORY_ENTRY_TLS                 9
#define    IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG         10
#define    IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT        11
#define    IMAGE_DIRECTORY_ENTRY_IAT                 12        // Import Address Table
#define    IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT        13
#define    IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR      14
#define    IMAGE_DIRECTORY_ENTRY_RESERVED            15
```

Understanding PE File Format

NT Header – Image Optional Header

Size = 0xE0 (224) bytes

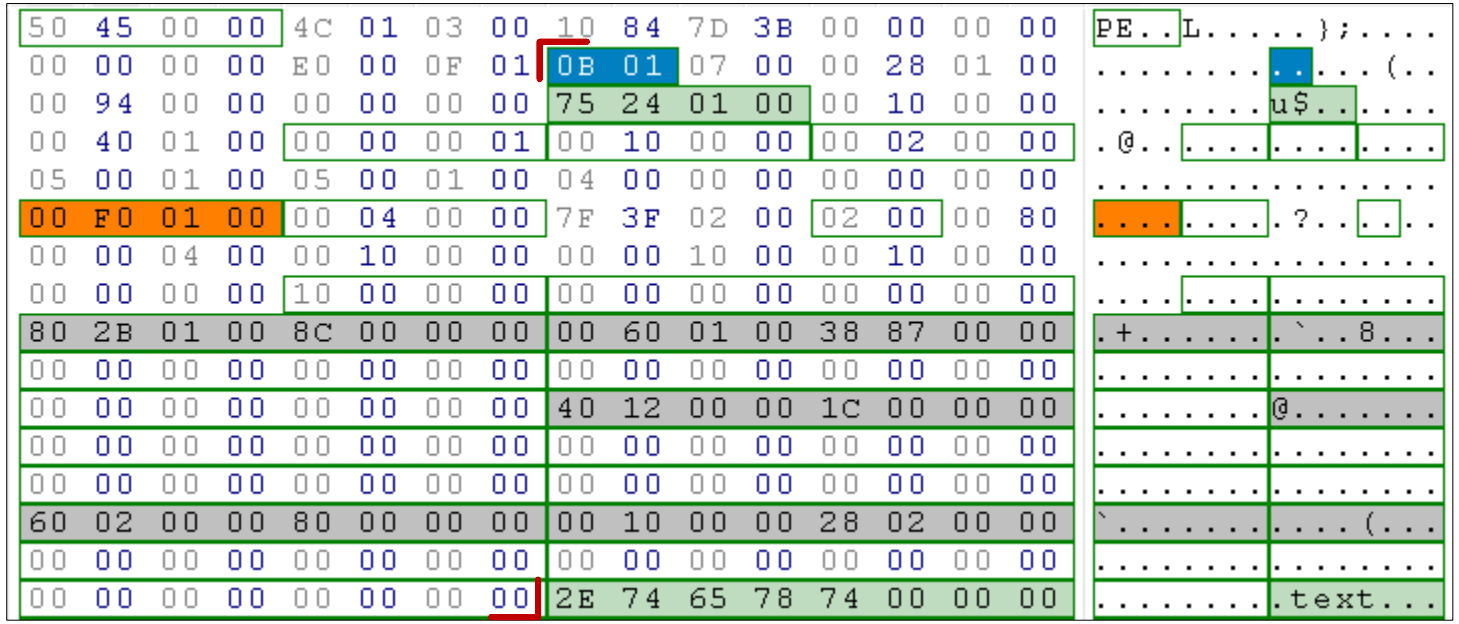


- **Magic (2):** 0x010b (32 bits)
- **AddressOfEntryPoint (4):** 0x00012475
- **ImageBase (4):** 0x01000000
- **SectionAlignment (4):** 0x00001000 (4096)
- **FileAlignment (4):** 0x00000200 (512)
- **SizeOfImage (4):** 0x0001F000 (126976)
- **SizeOfHeaders (4):** 0x00000400 (1024)
- **SubSystem (2):** 0x0002 (WINDOWS_GUI)
- **NumberOfRvaAndSizes (4):** 0x00000010 (16)

Understanding PE File Format

NT Header – Image Optional Header

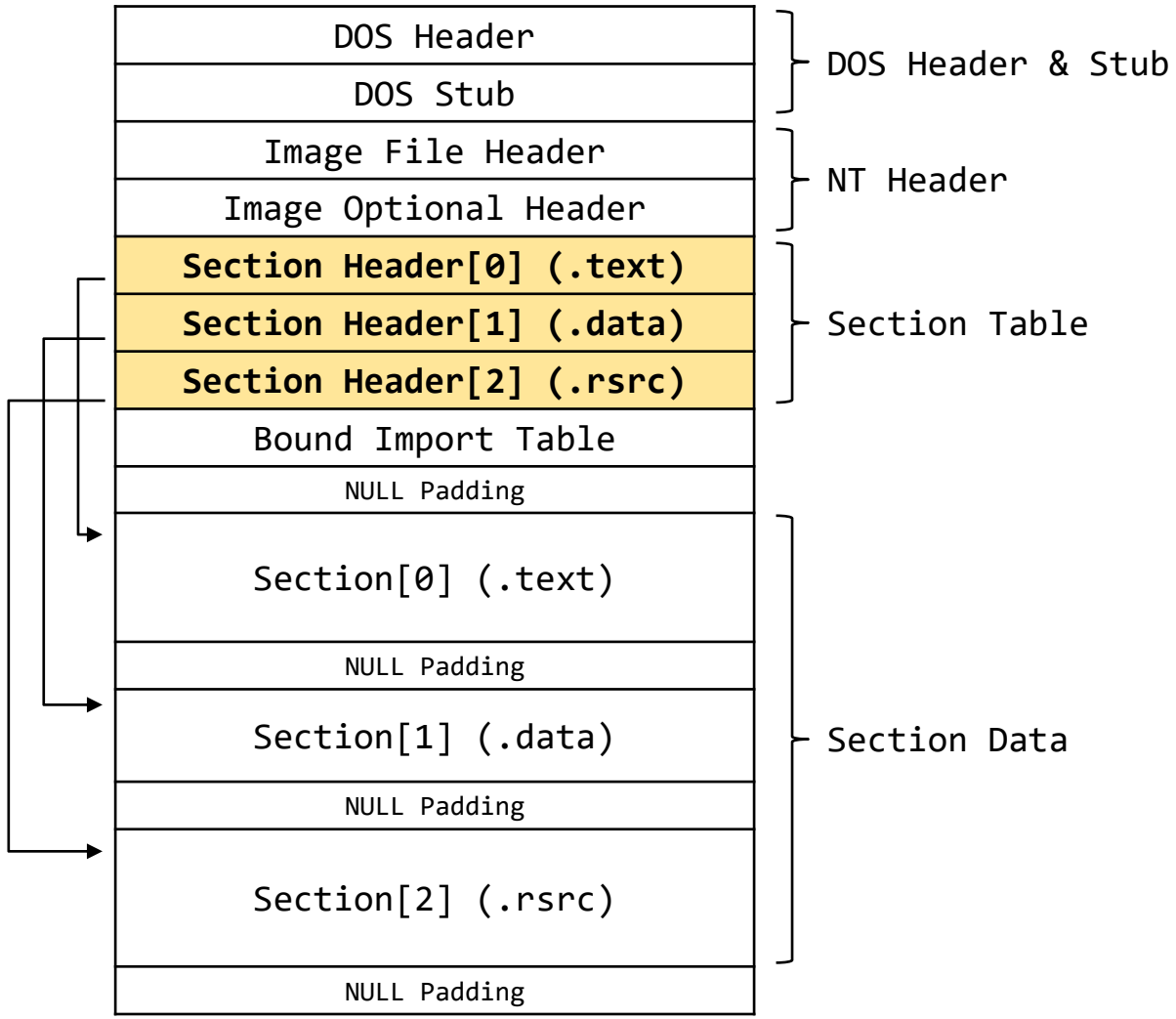
Size = 0xE0 (224) bytes



- **IMAGE_DIRECTORY_ENTRY_IMPORT (8)** — Image Import Descriptor → RVA: 0x00012B80, Size: 0x0000008C (140)
- **IMAGE_DIRECTORY_ENTRY_RESOURCE (8)** → RVA: 0x00016000, Size: 0x00008738 (34616)
- **IMAGE_DIRECTORY_ENTRY_DEBUG (8)** → RVA: 0x00001240, Size: 0x0000001C (28)
- **IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT (8)** → RVA: 0x00000260, Size: 0x00000080 (128)
- **IMAGE_DIRECTORY_ENTRY_IAT (8)** — Import Address Table → RVA: 0x00001000, Size: 0x00000228 (552)

Understanding PE File Format

Section Table (Section Headers)



- **Section Header**

```
#define IMAGE_SIZEOF_SHORT_NAME  8

typedef struct _IMAGE_SECTION_HEADER {

   BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
   DWORD VirtualSize;
   DWORD VirtualAddress;
   DWORD SizeOfRawData;
   DWORD PointerToRawData;
   DWORD PointerToRelocations;
   DWORD PointerToLinenumbers;
   WORD  NumberOfRelocations;
   WORD  NumberOfLinenumbers;
   DWORD Characteristics;

} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

- [REF] NumberOfSections field of the Image File Header

Understanding PE File Format

Section Table (Section Headers)

- **Name**
  - An 8-byte, null-padded UTF-8 string
- **VirtualSize**
  - The total size of the section when loaded into memory
- **VirtualAddress**
  - The address of the first byte of the section when loaded into memory (RVA)
- **SizeOfRawData**
  - The size of the initialized data in the image file
- **PointerToRawData**
  - A file pointer to the first page within the image file
- **Characteristics**
  - The characteristics of the section

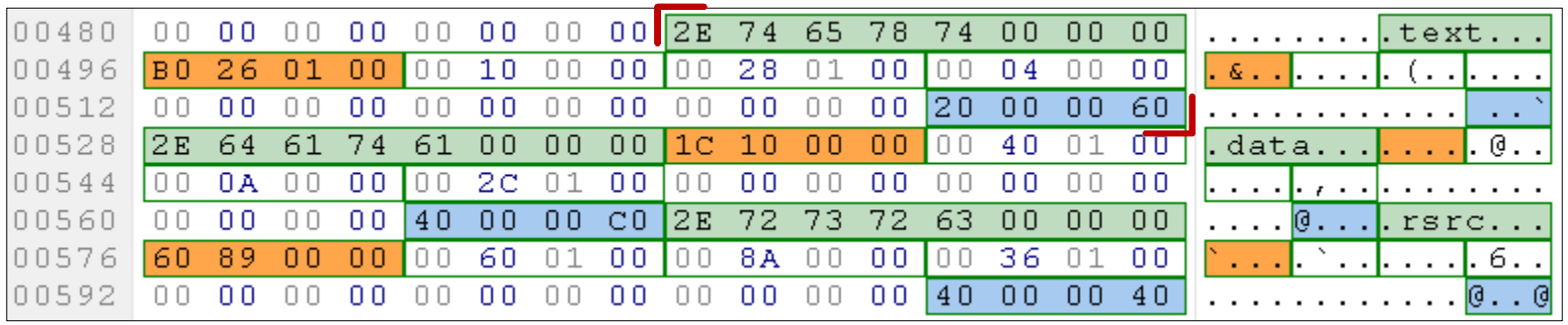
- **Characteristics**

```
#define     IMAGE_SCN_CNT_CODE                  0x00000020
#define     IMAGE_SCN_CNT_INITIALIZED_DATA      0x00000040
#define     IMAGE_SCN_CNT_UNINITIALIZED_DATA    0x00000080
#define     IMAGE_SCN_LNK_OTHER                 0x00000100
#define     IMAGE_SCN_LNK_INFO                  0x00000200
#define     IMAGE_SCN_LNK_REMOVE                0x00000800
#define     IMAGE_SCN_LNK_COMDAT                0x00001000
#define     IMAGE_SCN_MEM_FARDATA               0x00008000
#define     IMAGE_SCN_MEM_PURGEABLE             0x00020000
#define     IMAGE_SCN_MEM_16BIT                 0x00020000
#define     IMAGE_SCN_MEM_LOCKED                0x00040000
#define     IMAGE_SCN_MEM_PRELOAD               0x00080000

#define     IMAGE_SCN_ALIGN_XBYTES              (0x00100000 ~ 0x00E00000)
#define     IMAGE_SCN_ALIGN_MASK                0x00F00000

#define     IMAGE_SCN_LNK_NRELOC_OVFL           0x01000000
#define     IMAGE_SCN_MEM_DISCARDABLE           0x02000000
#define     IMAGE_SCN_MEM_NOT_CACHED            0x04000000
#define     IMAGE_SCN_MEM_NOT_PAGED             0x08000000
#define     IMAGE_SCN_MEM_SHARED                0x10000000
#define     IMAGE_SCN_MEM_EXECUTE               0x20000000
#define     IMAGE_SCN_MEM_READ                  0x40000000
#define     IMAGE_SCN_MEM_WRITE                 0x80000000
```

Understanding PE File Format

Section Header – .text

Size = 0x28 (40) bytes



00480	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00
00496	B0	26	01	00	00	10	00	00	00	28	01	00	00	04	00	00
00512	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60
00528	2E	64	61	74	61	00	00	00	1C	10	00	00	00	40	01	00
00544	00	0A	00	00	00	2C	01	00	00	00	00	00	00	00	00	00
00560	00	00	00	00	40	00	00	C0	2E	72	73	72	63	00	00	00
00576	60	89	00	00	00	60	01	00	00	8A	00	00	00	36	01	00
00592	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	40

00480	.	.	.	.	.	.	.	.	.	t	e	x	t	.	.	.
00496	.	&	.	.	.	.	.	.	.	(	.	.	.	.	.	.
00512	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	`
00528	.	d	a	t	a	.	.	.	.	.	.	.	.	@	.	.
00544	.	.	.	.	.	,	.	.	.	.	.	.	.	.	.	.
00560	.	.	.	.	@	.	.	.	.	r	s	r	c	.	.	.
00576	`	.	.	.	.	`	.	.	.	.	.	.	.	6	.	.
00592	.	.	.	.	.	.	.	.	.	.	.	.	.	@	.	@

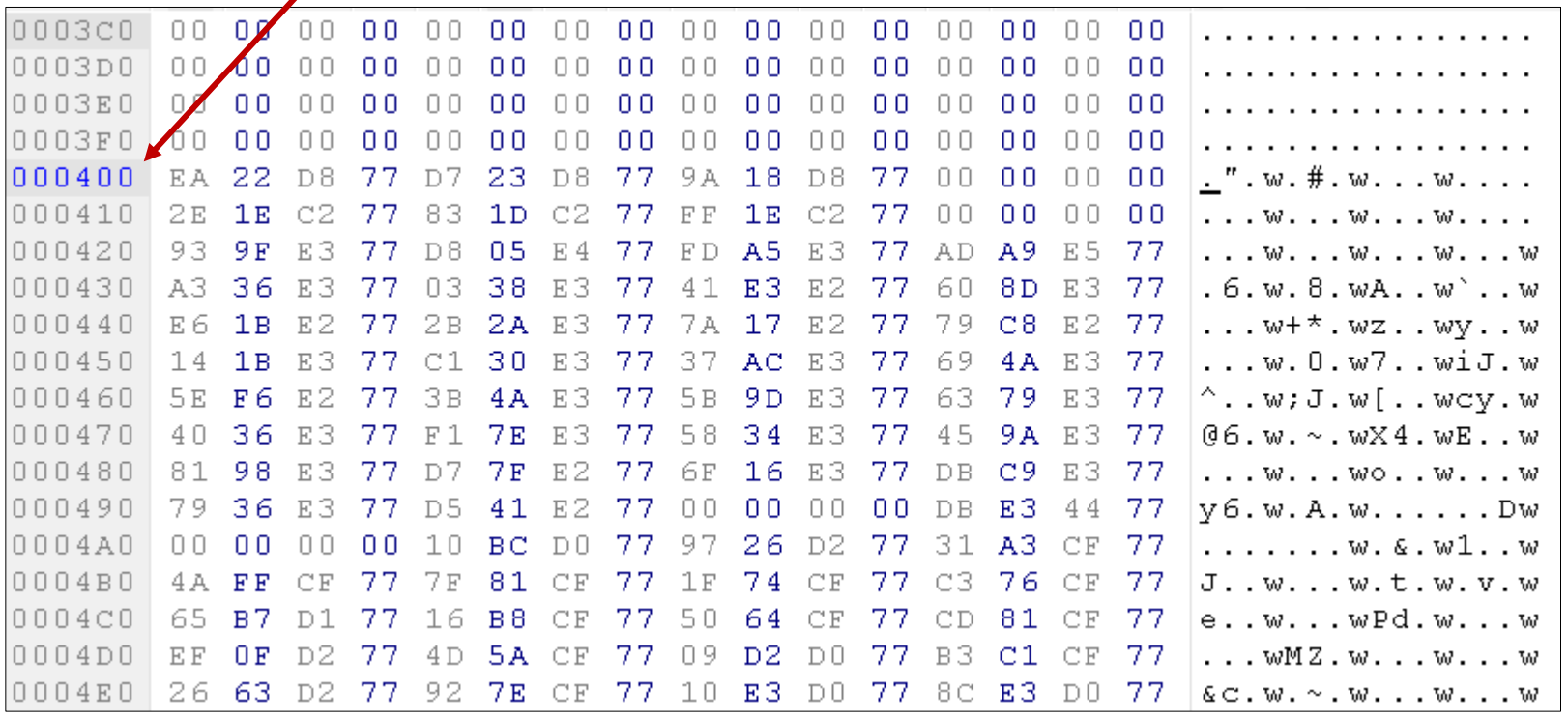
- **Name (8):** .text
- **VirtualSize (4):** 0x000126B0 (75440)
- **VirtualAddress (4):** 0x00001000 (4096)
- **SizeOfRawData (4):** 0x00012800 (75776)
- **PointerToRawData (4):** 0x00000400 (1024)

- **Characteristics (4):** 0x60000020
  - IMAGE_SCN_CNT_CODE 0x00000020
  - IMAGE_SCN_MEM_EXECUTE 0x20000000
  - IMAGE_SCN_MEM_READ 0x40000000

Understanding PE File Format

Section Header – .text

- **Section Data – .text**
  - **PointerToRawData**: 0x00000400 (1,024) offset

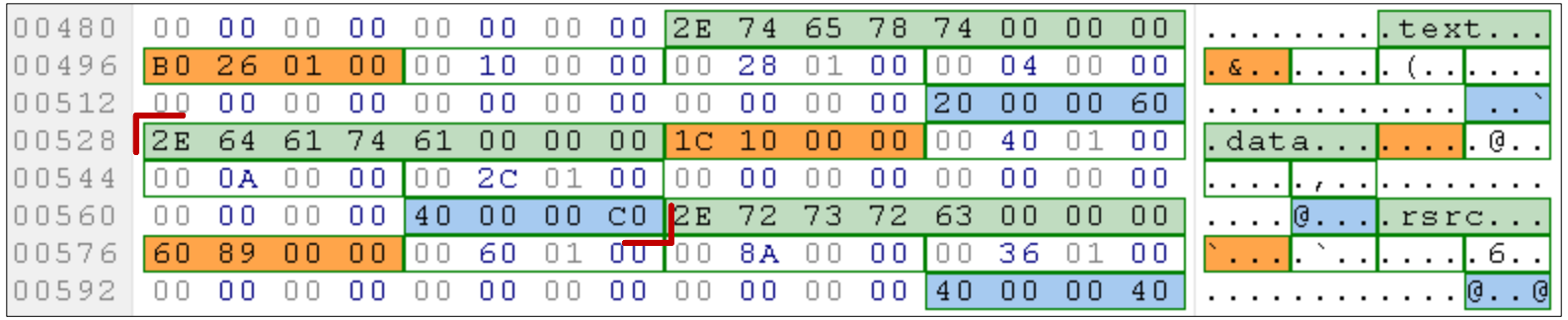


```
0003C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0003D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0003E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0003F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000400  EA 22 D8 77 D7 23 D8 77 9A 18 D8 77 00 00 00 00  ê".w.#.w...w....
000410  2E 1E C2 77 83 1D C2 77 FF 1E C2 77 00 00 00 00  ...w...w...w....
000420  93 9F E3 77 D8 05 E4 77 FD A5 E3 77 AD A9 E5 77  ...w...w...w...w
000430  A3 36 E3 77 03 38 E3 77 41 E3 E2 77 60 8D E3 77  .6.w.8.wA..w`..w
000440  E6 1B E2 77 2B 2A E3 77 7A 17 E2 77 79 C8 E2 77  ...w+*.wz..wy..w
000450  14 1B E3 77 C1 30 E3 77 37 AC E3 77 69 4A E3 77  ...w.0.w7..wiJ.w
000460  5E F6 E2 77 3B 4A E3 77 5B 9D E3 77 63 79 E3 77  ^..w;J.w[..wcy.w
000470  40 36 E3 77 F1 7E E3 77 58 34 E3 77 45 9A E3 77  @6.w.~.wX4.wE..w
000480  81 98 E3 77 D7 7F E2 77 6F 16 E3 77 DB C9 E3 77  ...w...wo..w...w
000490  79 36 E3 77 D5 41 E2 77 00 00 00 00 DB E3 44 77  y6.w.A.w......Dw
0004A0  00 00 00 00 10 BC D0 77 97 26 D2 77 31 A3 CF 77  .......w.&.w1..w
0004B0  4A FF CF 77 7F 81 CF 77 1F 74 CF 77 C3 76 CF 77  J..w...w.t.w.v.w
0004C0  65 B7 D1 77 16 B8 CF 77 50 64 CF 77 CD 81 CF 77  e..w...wPd.w...w
0004D0  EF 0F D2 77 4D 5A CF 77 09 D2 D0 77 B3 C1 CF 77  ...wMZ.w...w...w
0004E0  26 63 D2 77 92 7E CF 77 10 E3 D0 77 8C E3 D0 77  &c.w.~.w...w...w
```

Understanding PE File Format

Section Header – .data

Size = 0x28 (40) bytes



00480	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00
00496	B0	26	01	00	00	10	00	00	00	28	01	00	00	04	00	00
00512	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60
00528	2E	64	61	74	61	00	00	00	1C	10	00	00	00	40	01	00
00544	00	0A	00	00	00	2C	01	00	00	00	00	00	00	00	00	00
00560	00	00	00	00	40	00	00	C0	2E	72	73	72	63	00	00	00
00576	60	89	00	00	00	60	01	00	00	8A	00	00	00	36	01	00
00592	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	40

00480	.	.	.	.	.	.	.	.	.	t	e	x	t	.	.	.
00496	.	&	.	.	.	.	.	.	.	(	.	.	.	.	.	.
00512	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	`
00528	.	d	a	t	a	.	.	.	.	.	.	.	.	@	.	.
00544	.	.	.	.	.	,	.	.	.	.	.	.	.	.	.	.
00560	.	.	.	.	@	.	.	.	.	r	s	r	c	.	.	.
00576	`	.	.	.	.	`	.	.	.	.	.	.	.	6	.	.
00592	.	.	.	.	.	.	.	.	.	.	.	.	@	.	.	@

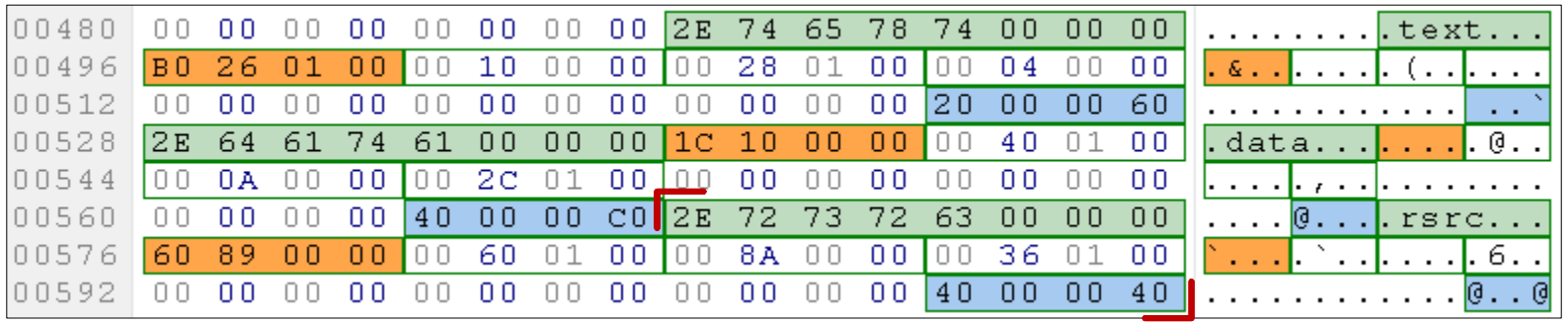
- **Name (8):** .data
- **VirtualSize (4):** 0x0000101C (4124)
- **VirtualAddress (4):** 0x00014000 (81920)
- **SizeOfRawData (4):** 0x00000A00 (2560)
- **PointerToRawData (4):** 0x00012C00 (76800)

- **Characteristics (4):** 0xC0000040
  - IMAGE_SCN_INITIALIZED_DATA 0x00000040
  - IMAGE_SCN_MEM_READ 0x40000000
  - IMAGE_SCN_MEM_WRITE 0x80000000

Understanding PE File Format

Section Header – .rsrc

Size = 0x28 (40) bytes



- **Name (8):** .rsrc
- **VirtualSize (4):** 0x00008960 (35168)
- **VirtualAddress (4):** 0x00016000 (90112)
- **SizeOfRawData (4):** 0x00008A00 (35328)
- **PointerToRawData (4):** 0x00013600 (79360)

- **Characteristics (4):** 0x40000040
  - IMAGE_SCN_INITIALIZED_DATA 0x00000040
  - IMAGE_SCN_MEM_READ 0x40000000

Understanding PE File Format

Section Header

- **Common Sections of a PE File for Windows**

Section	Description	Characteristics
.text	Executable code	IMAGE_SCN_CNT_CODE | IMAGE_SCN_MEM_EXECUTE | IIMAGE_SCN_MEM_READ
.data	Initialized data	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.rdata	Read-only initialized data	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.bss	Uninitialized data	IMAGE_SCN_CNT_UNINITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.idata	Import tables	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.edata	Export tables	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.pdata	Exception information	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.reloc	Image relocations	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_DISCARDABLE
.rsrc	Resource directory	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ

https://docs.microsoft.com/en-us/windows/win32/debug/pe-format

Understanding PE File Format

RVA to RAW

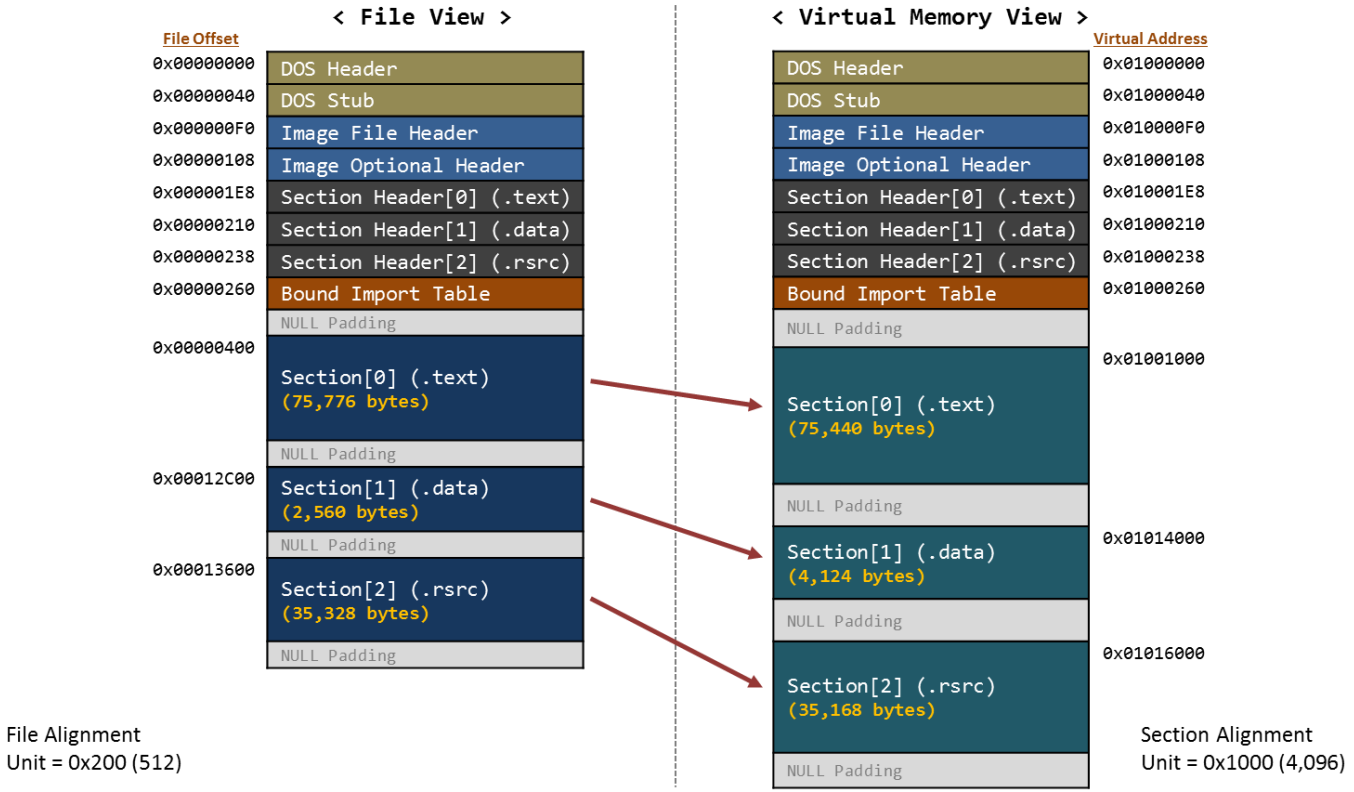
- **RVA to RAW ?**
  - Mapping a RVA to a PE file offset (RAW)

- **How to Convert between RVA and RAW**
  - Determining a section to which the target **RVA** belongs
  - Obtaining the `VirtualAddress` and `PointerToRawData` values of the section
  - **RAW** - `PointerToRawData` = **RVA** – `VirtualAddress`
  - **RAW** = **RVA** - `VirtualAddress` + `PointerToRawData`

Understanding PE File Format

RAW = RVA - VirtualAddress + PointerToRawData

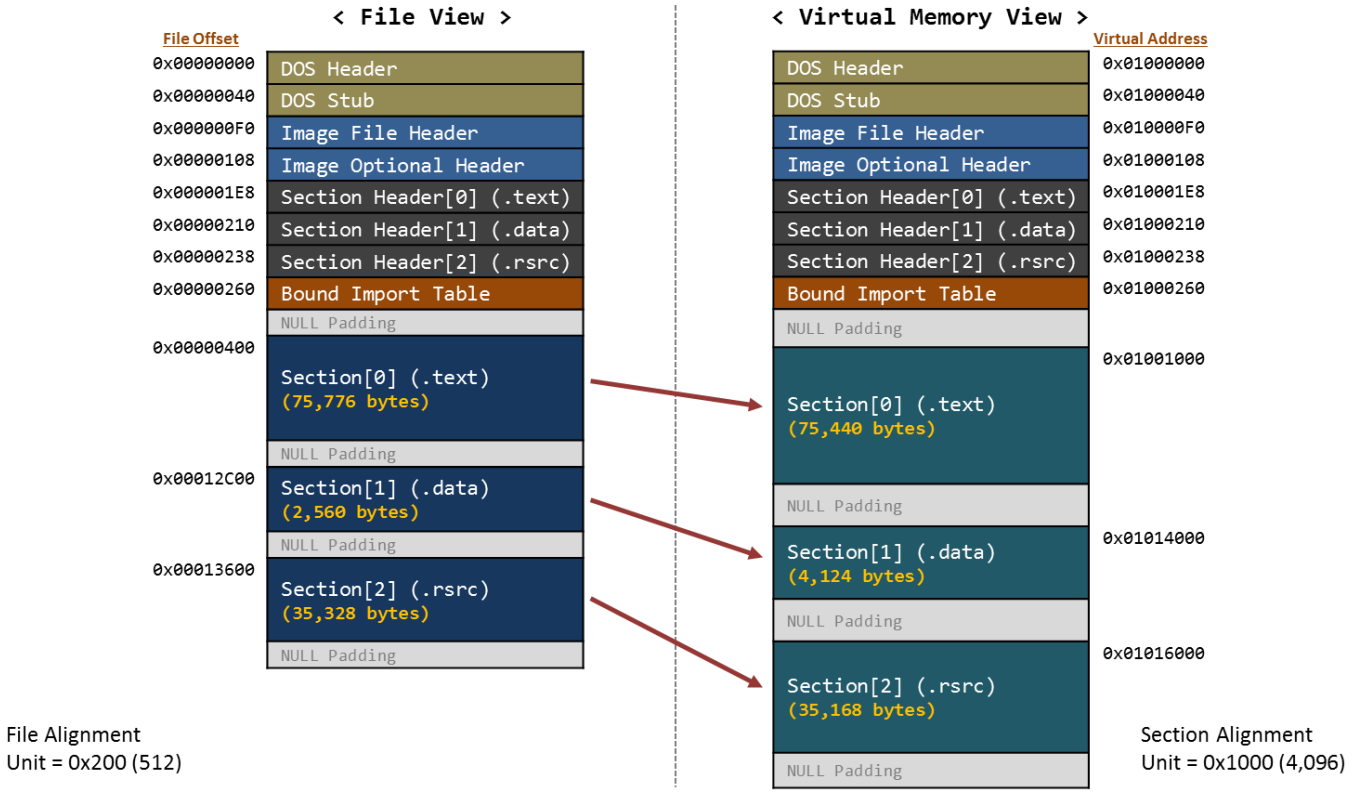
- **Q1) If the target RVA is 0x1100, what is the corresponding file offset?**
  - RVA 0x1100(VA 0x01001100) belongs to .text section (VirtualAddress: 0x1000, PointerToRawData: 0x400)
  - RAW = 0x1100 - 0x1000 + 0x400 = **0x500**



Understanding PE File Format

RAW = RVA - VirtualAddress + PointerToRawData

- **Q2) If the target RVA is 0x14500, what is the corresponding file offset?**
  - RVA 0x14500(VA 0x01014500) belongs to .data section (VirtualAddress: 0x14000, PointerToRawData: 0x12C00)
  - RAW = 0x14500 - 0x14000 + 0x12C00 = **0x13100**



Understanding PE File Format

Section Data – IMPORT

- **INT (Import Name Table) & IAT (Import Address Table)**

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {

    DWORD  OriginalFirstThunk;    // RVA to original unbound IAT (IMAGE_THUNK_DATA)
                                  // INT (Import Name Table) address (RVA)
    DWORD  TimeDateStamp;         // 0 if not bound, and -1 if bound
    DWORD  ForwarderChain;
    DWORD  Name;                  // library name's address (RVA)
    DWORD  FirstThunk;            // IAT (Import Address Table) address (RVA)

} IMAGE_IMPORT_DESCRIPTOR;
```

For each imported library

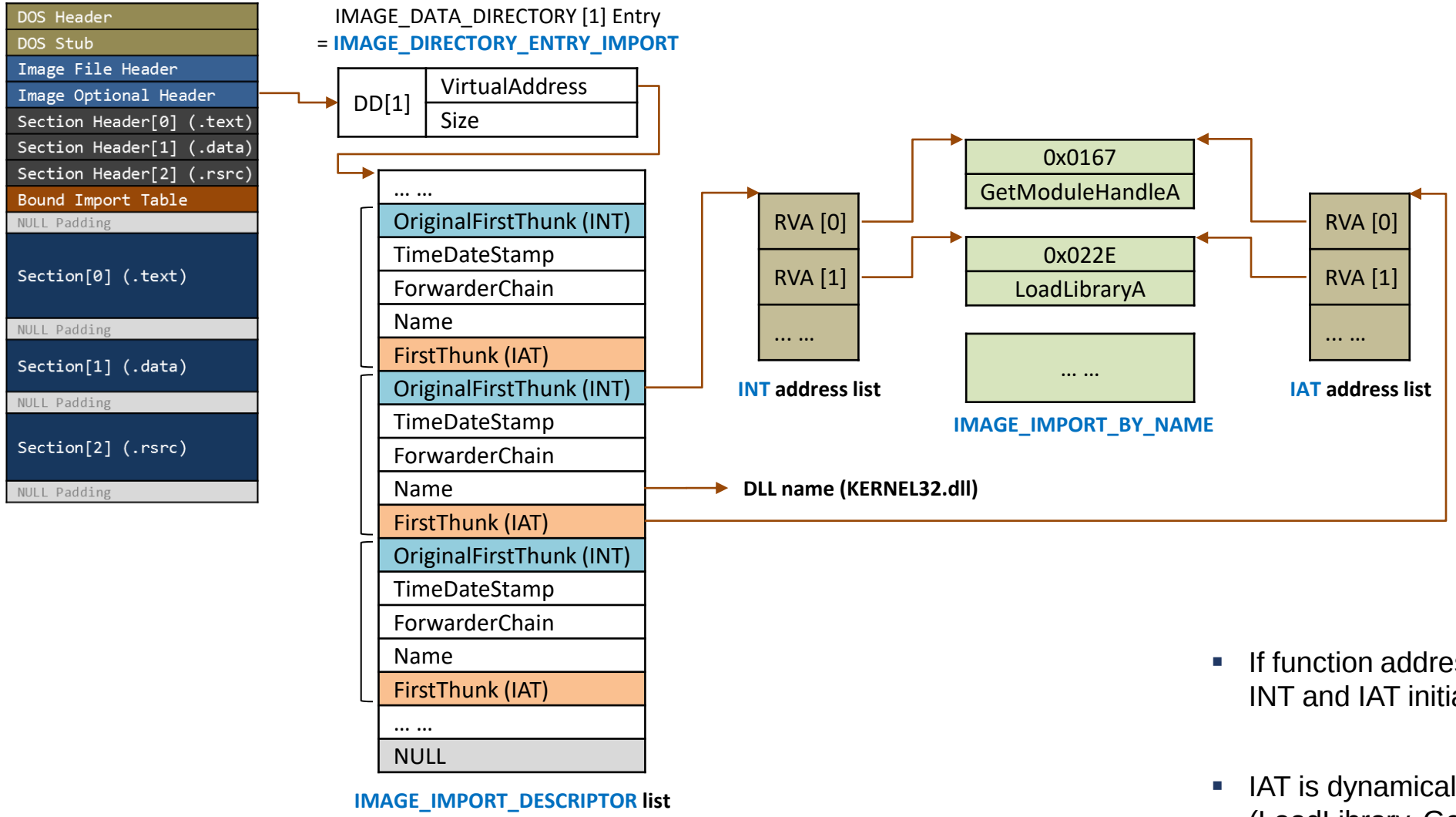
```
typedef struct _IMAGE_THUNK_DATA {

    union {
        DWORD ForwarderString;
        DWORD Function;
        DWORD Ordinal;
        DWORD AddressOfData;    // IMAGE_IMPORT_BY_NAME
    };

} IMAGE_THUNK_DATA;
```

For each imported function in a library

```
typedef struct _IMAGE_IMPORT_BY_NAME {

    WORD   Hint;        // ordinal
    BYTE   Name[1];     // function name

} IMAGE_IMPORT_BY_NAME;
```

Understanding PE File Format

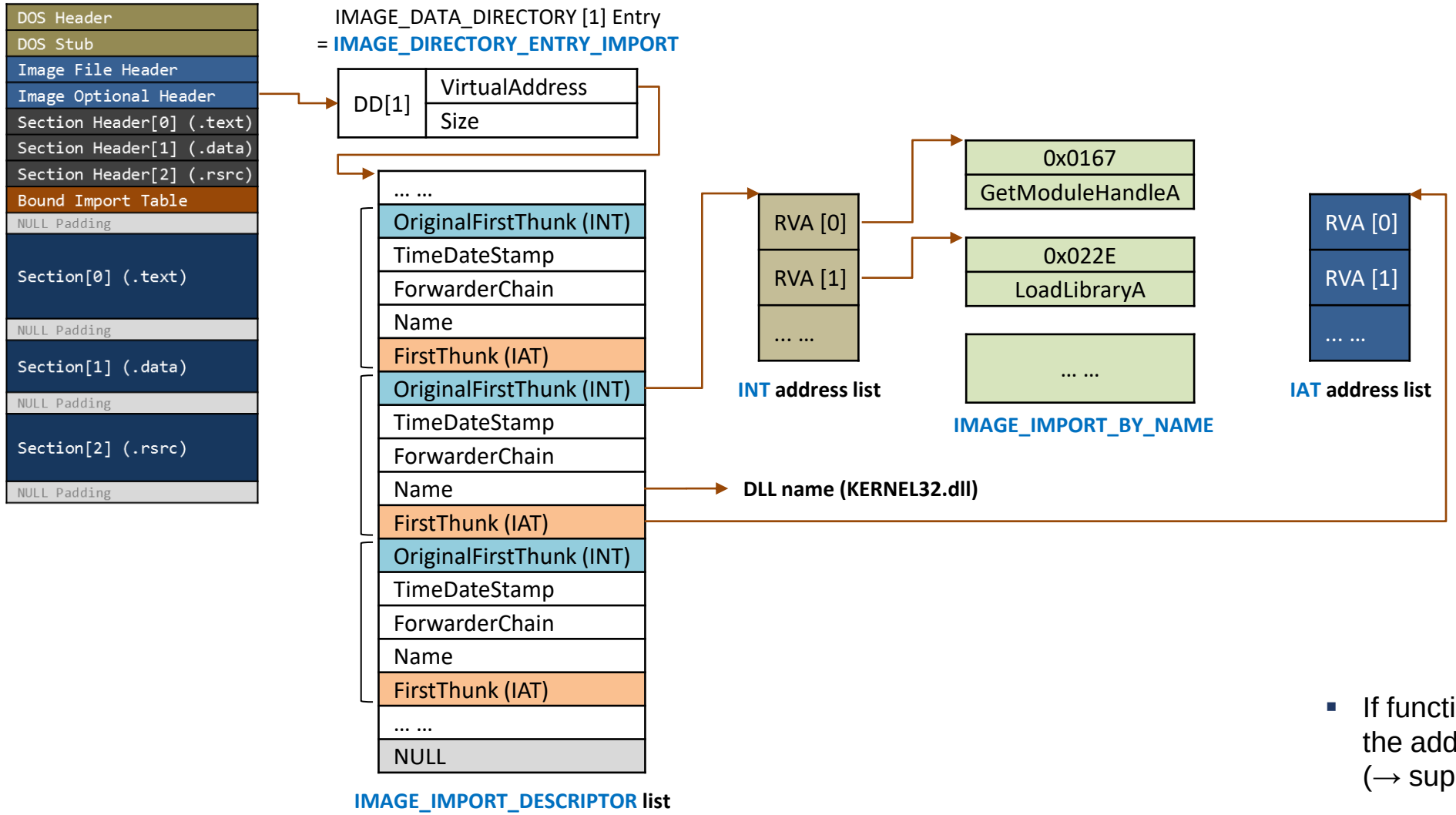
Section Data – IMPORT – INT & IAT



- If function addresses are not bound,
  INT and IAT initially have the same information
- IAT is dynamically set by the PE loader
  (LoadLibrary, GetProcAddress)

Understanding PE File Format

Section Data – IMPORT – INT & IAT



- If function addresses are bound, the addresses are stored in IAT (→ supporting faster loading)

Understanding PE File Format

Section Data – IMPORT – INT & IAT

0x11F80 (RAW) = 0x12B80 - 0x1000 + 0x400

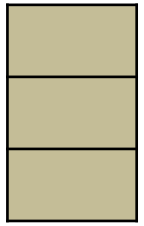
```
11F80  A8 2C 01 00 FF FF FF FF FF FF FF FF 42 2E 01 00  .,..........B...
11F90  9C 10 00 00 C8 2D 01 00 FF FF FF FF FF FF FF FF  ....-..........
11FA0  60 2F 01 00 BC 11 00 00 0C 2C 01 00 FF FF FF FF  `/.......,......
11FB0  FF FF FF FF FC 2F 01 00 00 10 00 00 2C 2C 01 00  ...../......,,..
11FC0  FF FF FF FF FF FF FF FF D4 31 01 00 20 10 00 00  .........1.. ...
11FD0  1C 2C 01 00 FF FF FF FF FF FF FF FF 0C 32 01 00  .,...........2..
11FE0  10 10 00 00 B0 2C 01 00 FF FF FF FF FF FF FF FF  ....,..........
11FF0  A4 36 01 00 A4 10 00 00 00 00 00 00 00 00 00 00  .6..............
12000  00 00 00 00 00 00 00 00 00 00 00 00 EC 2F 01 00  ............./..
```

IMAGE_DATA_DIRECTORY [1] Entry
= IMAGE_DIRECTORY_ENTRY_IMPORT

DD[1]	0x00012B80
	0x0000008C

		RVA	RAW
	… …		
	OriginalFirstThunk (INT)		
	TimeDateStamp		
	ForwarderChain		
	Name		
	FirstThunk (IAT)		
4th DLL	OriginalFirstThunk (INT)	0x00012C2C	0x0001202C
	TimeDateStamp	0xFFFFFFFF	0xFFFFFFFF
	ForwarderChain	0xFFFFFFFF	0xFFFFFFFF
	Name	0x000131D4	0x000125D4
	FirstThunk (IAT)	0x00001020	0x00000420
	OriginalFirstThunk (INT)		
	TimeDateStamp		
	ForwarderChain		
	Name		
	FirstThunk (IAT)		
	… …		
	NULL		

IMAGE_IMPORT_DESCRIPTOR list



INT address list

IMAGE_IMPORT_BY_NAME

IAT address list

Interpreting the Import Descriptor for the 4th imported DLL

Understanding PE File Format

Section Data – IMPORT – INT & IAT

IMAGE_DATA_DIRECTORY [1] Entry
= IMAGE_DIRECTORY_ENTRY_IMPORT

DD[1]	0x00012B80
	0x0000008C

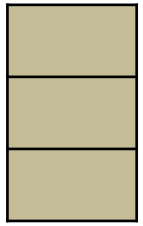
0x125D4 (RAW) = 0x131D4 - 0x1000 + 0x400

```
125D0  66 6F 41 00 4B 45 52 4E 45 4C 33 32 2E 64 6C 6C  foA.KERNEL32.dll
125E0  00 00 13 02 53 65 74 42 6B 43 6F 6C 6F 72 00 00  ....SetBkColor..
```

		RVA	RAW
	… …		
	OriginalFirstThunk (INT)		
	TimeDateStamp		
	ForwarderChain		
	Name		
	FirstThunk (IAT)		
4th DLL	OriginalFirstThunk (INT)	0x00012C2C	0x0001202C
	TimeDateStamp	0xFFFFFFFF	0xFFFFFFFF
	ForwarderChain	0xFFFFFFFF	0xFFFFFFFF
	Name	0x000131D4	0x000125D4
	FirstThunk (IAT)	0x00001020	0x00000420
	OriginalFirstThunk (INT)		
	TimeDateStamp		
	ForwarderChain		
	Name		
	FirstThunk (IAT)		
	… …		
	NULL		

IMAGE_IMPORT_DESCRIPTOR list

DLL name (KERNEL32.dll)



INT address list

IMAGE_IMPORT_BY_NAME

IAT address list

Obtaining DLL's name

Understanding PE File Format

Section Data – IMPORT – INT & IAT

0x1202C (RAW) = 0x12C2C - 0x1000 + 0x400

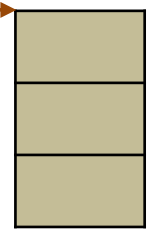
```
12020  F0 31 01 00 00 32 01 00 00 00 00 00 AE 31 01 00  .1...2.......1..
12030  9E 31 01 00 8C 31 01 00 7C 31 01 00 6E 31 01 00  .1...1..|1..n1..
12040  60 31 01 00 50 31 01 00 44 31 01 00 3C 31 01 00  `1..P1..D1..<1..
12050  26 31 01 00 C2 31 01 00 0A 31 01 00 FA 30 01 00  &1...1...1...0..
12060  EA 30 01 00 DA 30 01 00 CC 30 01 00 C0 30 01 00  .0...0...0...0..
12070  B4 30 01 00 9E 30 01 00 90 30 01 00 84 30 01 00  .0...0...0...0..
```

IMAGE_DATA_DIRECTORY [1] Entry
= **IMAGE_DIRECTORY_ENTRY_IMPORT**

DD[1]	0x00012B80
	0x0000008C

		RVA	RAW
	… …		
	OriginalFirstThunk (INT)		
	TimeDateStamp		
	ForwarderChain		
	Name		
	FirstThunk (IAT)		
4th DLL	OriginalFirstThunk (INT)	0x00012C2C	0x0001202C
	TimeDateStamp	0xFFFFFFFF	0xFFFFFFFF
	ForwarderChain	0xFFFFFFFF	0xFFFFFFFF
	Name	0x000131D4	0x000125D4
	FirstThunk (IAT)	0x00001020	0x00000420
	OriginalFirstThunk (INT)		
	TimeDateStamp		
	ForwarderChain		
	Name		
	FirstThunk (IAT)		
	… …		
	NULL		

IMAGE_IMPORT_DESCRIPTOR list



INT address list

IMAGE_IMPORT_BY_NAME

IAT address list

DLL name (KERNEL32.dll)

Accessing INT address list

Understanding PE File Format

Section Data – IMPORT – INT & IAT

0x1202C (RAW) = 0x12C2C - 0x1000 + 0x400

```
12020  F0 31 01 00 00 32 01 00 00 00 00 00 AE 31 01 00  .1...2.......1..
12030  9E 31 01 00 8C 31 01 00 7C 31 01 00 6E 31 01 00  .1...1..|1..n1..
12040  60 31 01 00 50 31 01 00 44 31 01 00 3C 31 01 00  `1..P1..D1..<1..
12050  26 31 01 00 C2 31 01 00 0A 31 01 00 FA 30 01 00  &1...1...1...0..
12060  EA 30 01 00 DA 30 01 00 CC 30 01 00 C0 30 01 00  .0...0...0...0..
12070  B4 30 01 00 9E 30 01 00 90 30 01 00 84 30 01 00  .0...0...0...0..
```

IMAGE_DATA_DIRECTORY [1] Entry
= **IMAGE_DIRECTORY_ENTRY_IMPORT**

DD[1]	0x00012B80
	0x0000008C

4th DLL

	RVA	RAW
… …		
OriginalFirstThunk (INT)	0x00012C2C	0x0001202C
TimeDateStamp	0xFFFFFFFF	0xFFFFFFFF
ForwarderChain	0xFFFFFFFF	0xFFFFFFFF
Name	0x000131D4	0x000125D4
FirstThunk (IAT)	0x00001020	0x00000420
… …		
NULL		

IMAGE_IMPORT_DESCRIPTOR list

0x131AE
0x1319E
… …

INT address list

IMAGE_IMPORT_BY_NAME

IAT address list

DLL name (KERNEL32.dll)

Obtaining INT addresses

Understanding PE File Format

Section Data – IMPORT – INT & IAT

0x125AE (RAW) = 0x131AE - 0x1000 + 0x400
0x1259E (RAW) = 0x1319E - 0x1000 + 0x400

```
12590  74 50 72 6F 63 41 64 64 72 65 73 73 00 00 2E 02  tProcAddress....
125A0  4C 6F 61 64 4C 69 62 72 61 72 79 41 00 00 67 01  LoadLibraryA..g.
125B0  47 65 74 4D 6F 64 75 6C 65 48 61 6E 64 6C 65 41  GetModuleHandleA
125C0  00 00 9C 01 47 65 74 53 74 61 72 74 75 70 49 6E  ..GetStartupIn
```

IMAGE_DATA_DIRECTORY [1] Entry
= **IMAGE_DIRECTORY_ENTRY_IMPORT**

DD[1]	0x00012B80
	0x0000008C

		RVA	RAW
	… …		
	OriginalFirstThunk (INT)		
	TimeDateStamp		
	ForwarderChain		
	Name		
	FirstThunk (IAT)		
4th DLL	OriginalFirstThunk (INT)	0x00012C2C	0x0001202C
	TimeDateStamp	0xFFFFFFFF	0xFFFFFFFF
	ForwarderChain	0xFFFFFFFF	0xFFFFFFFF
	Name	0x000131D4	0x000125D4
	FirstThunk (IAT)	0x00001020	0x00000420
	OriginalFirstThunk (INT)		
	TimeDateStamp		
	ForwarderChain		
	Name		
	FirstThunk (IAT)		
	… …		
	NULL		

IMAGE_IMPORT_DESCRIPTOR list

0x131AE
0x1319E
… …

INT address list

0x0167
GetModuleHandleA

0x022E
LoadLibraryA

… …

IMAGE_IMPORT_BY_NAME

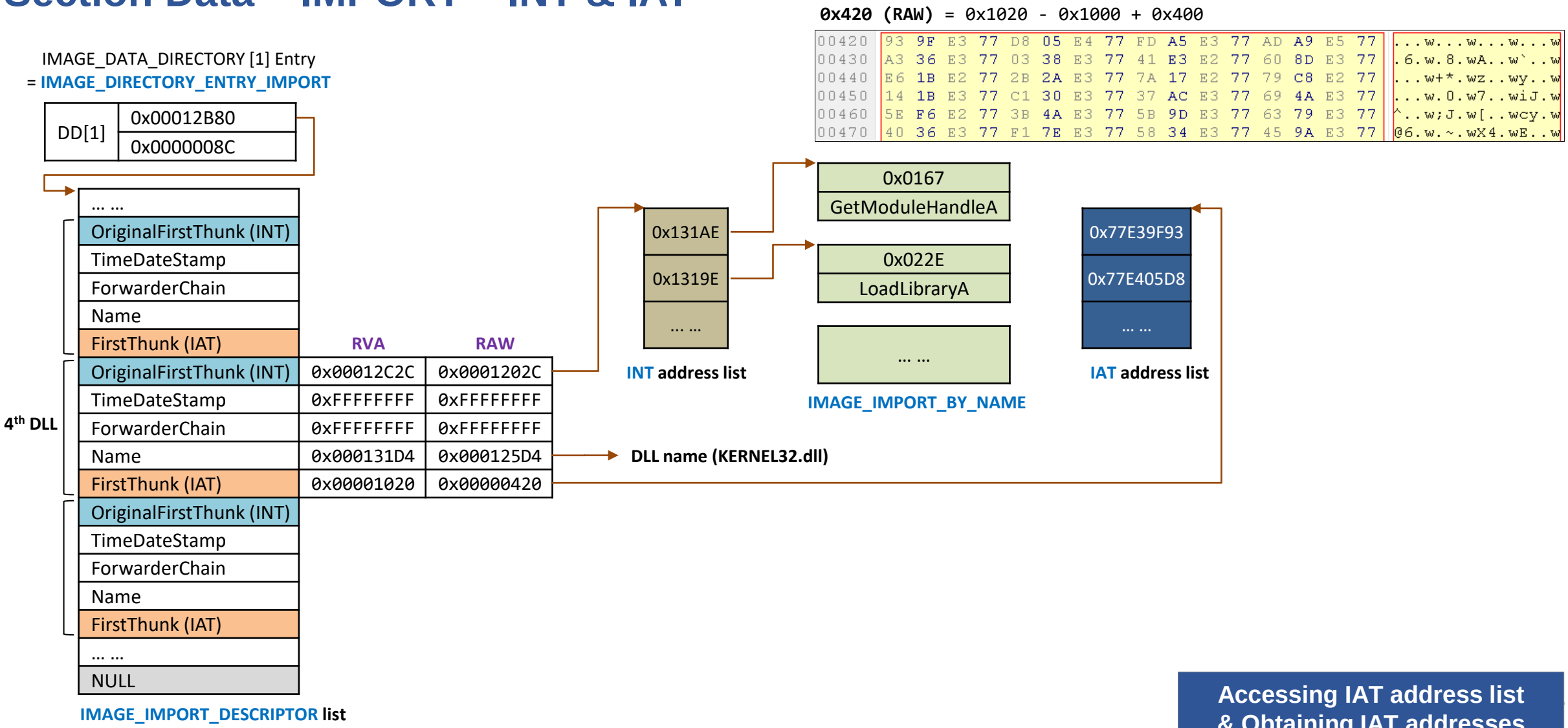
IAT address list

DLL name (KERNEL32.dll)

Obtaining function information

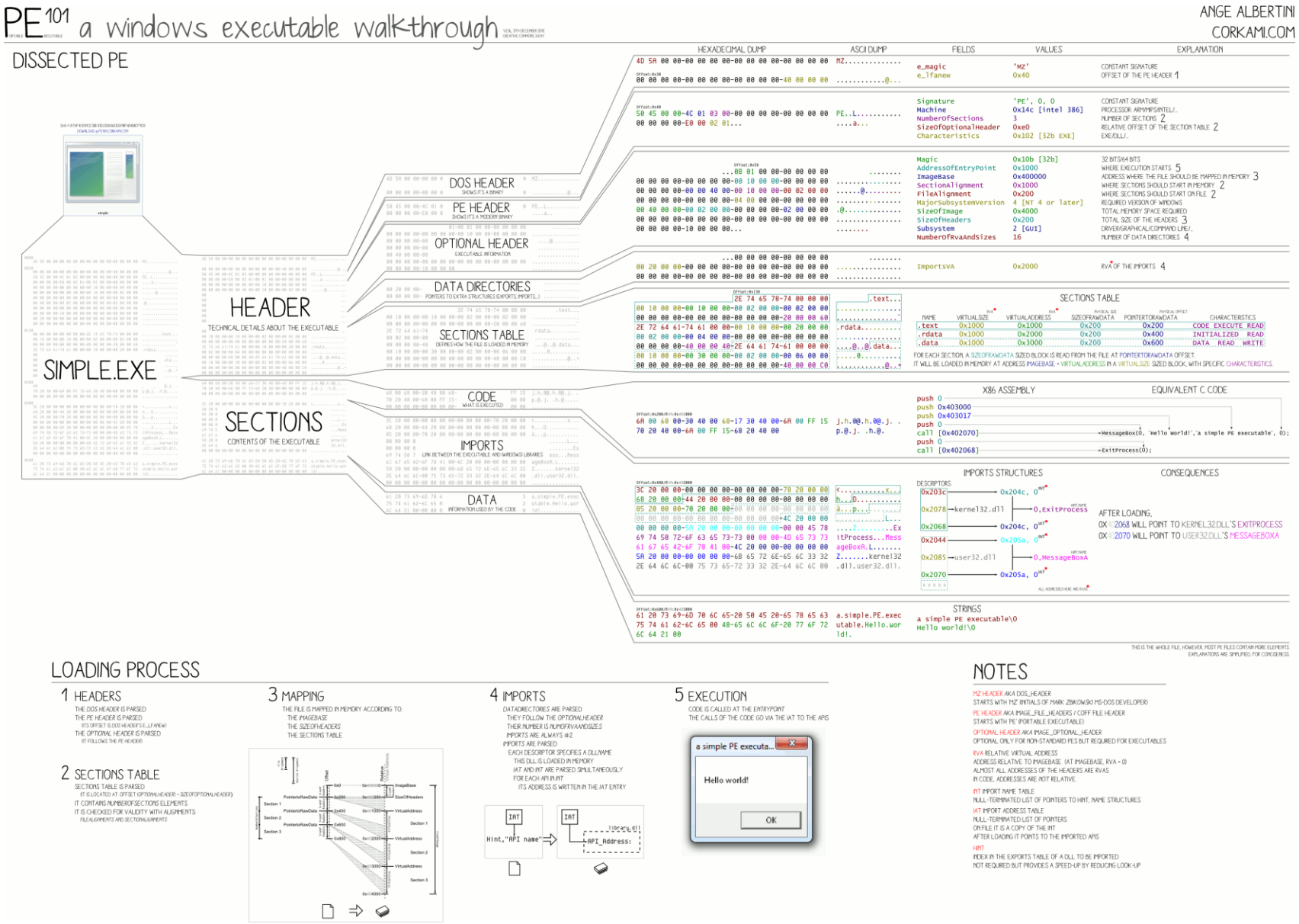
Understanding PE File Format

Section Data – IMPORT – INT & IAT



Accessing IAT address list & Obtaining IAT addresses

Understanding PE File Format



https://github.com/corkami/pics/blob/master/binary/pe101/README.md

Inspecting PE File Format

Inspecting PE File Format

Inspecting File Dependencies and Imports

- **DLL & API**
  - Windows exports most of its functions, called Application Programming Interfaces (**API**), required for these interactions in Dynamic Link Library (**DLL**) files
  - The functions that an executable imports from other files (mostly DLLs) are called **imported functions** (or **imports**)
  - e.g., On Windows systems, if a malware wants to create a file on a storage device, it can use an API `CreateFile()`, which is exported in `kernel32.dll`

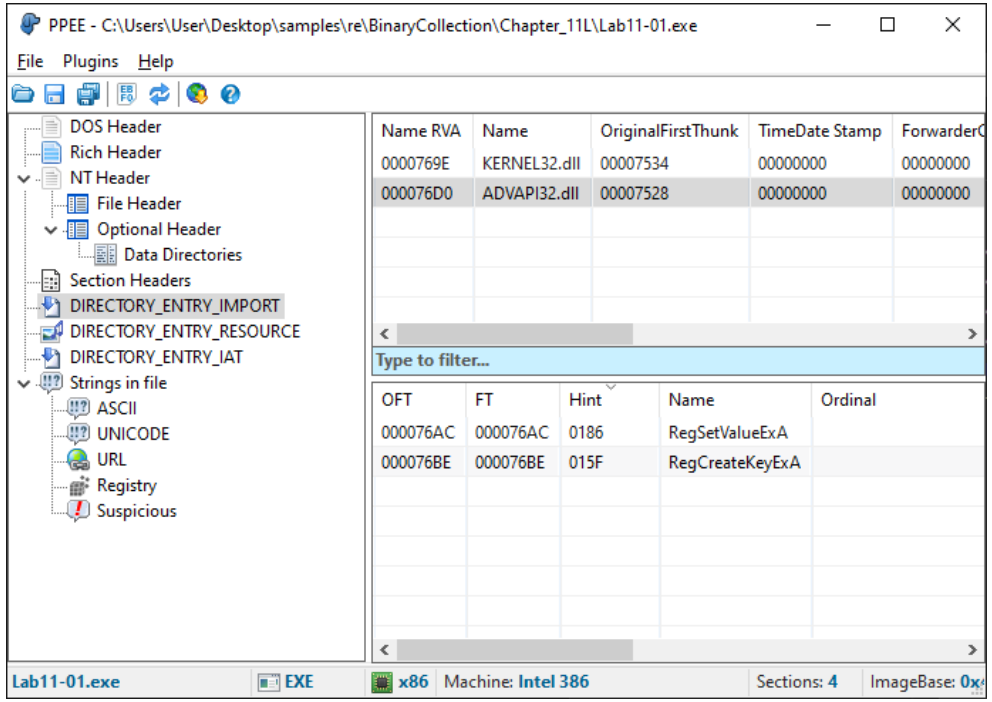
- **Import Table of PE File**
  - The file dependencies in executables are stored in the *import table* of the PE file structure

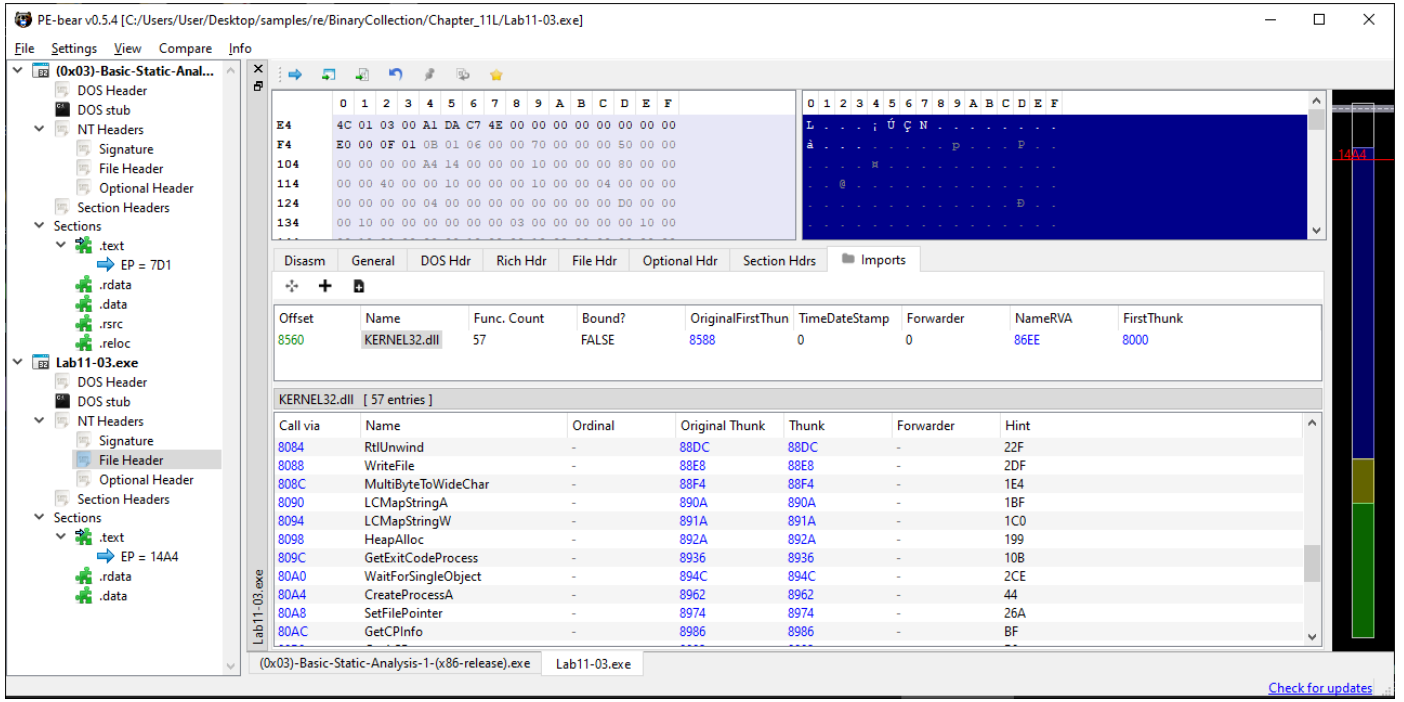
Inspecting PE File Format

Inspecting File Dependencies and Imports

- **PE-related Tools**
  - PPEE, pestudio, PE-bear, etc.

```
C:\Users\User
λ choco install pebear
Chocolatey v0.10.15
Installing the following packages:
pebear
By installing you accept licenses for the packages.
Progress: Downloading pebear 0.5.4... 100%
```

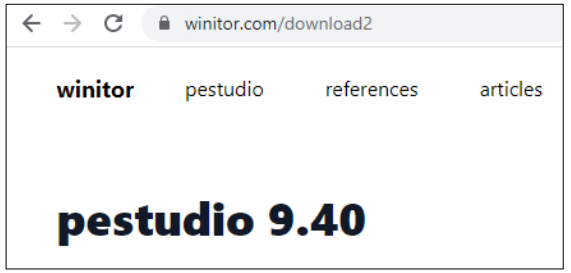


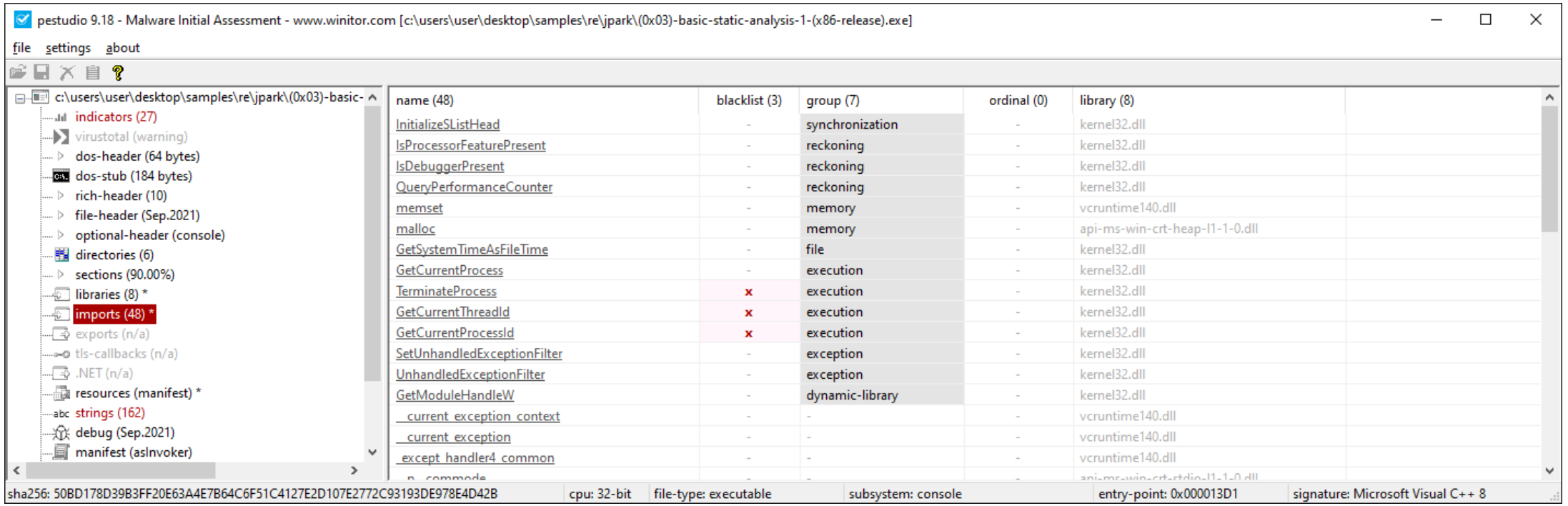


Inspecting PE File Format

Inspecting File Dependencies and Imports

- **PE-related Tools**
  - PPEE, pestudio, PE-bear, etc.
    - pestudio (winitor.com)

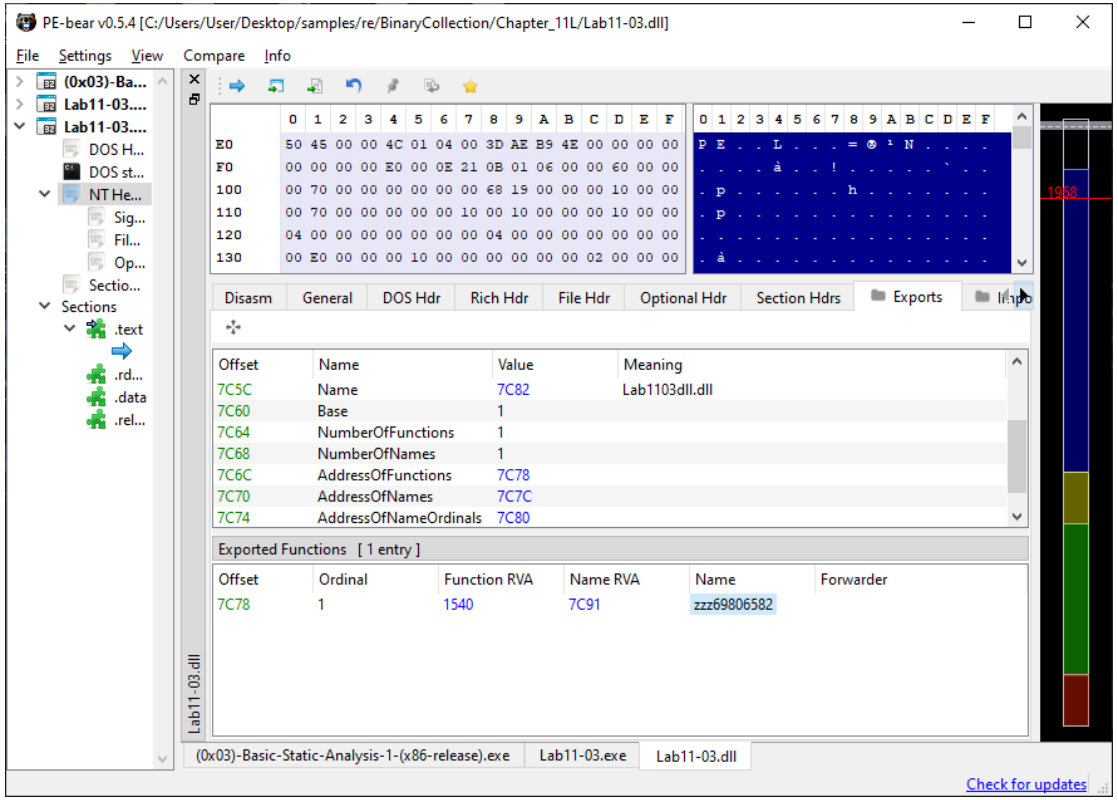




Inspecting PE File Format

Inspecting Exports

- **Exports (= Exported Functions)**
  - The executable and DLL can export functions, which **can be used by other programs**
  - Typically, a **DLL** *exports* functions (exports) that are imported by the executable
  - An attacker often creates a DLL that exports functions containing malicious functionality
  - **DLLs** can also *import* functions from other libraries (DLLs) to perform system operations



Inspecting PE File Format

Examining PE Section Table And Sections

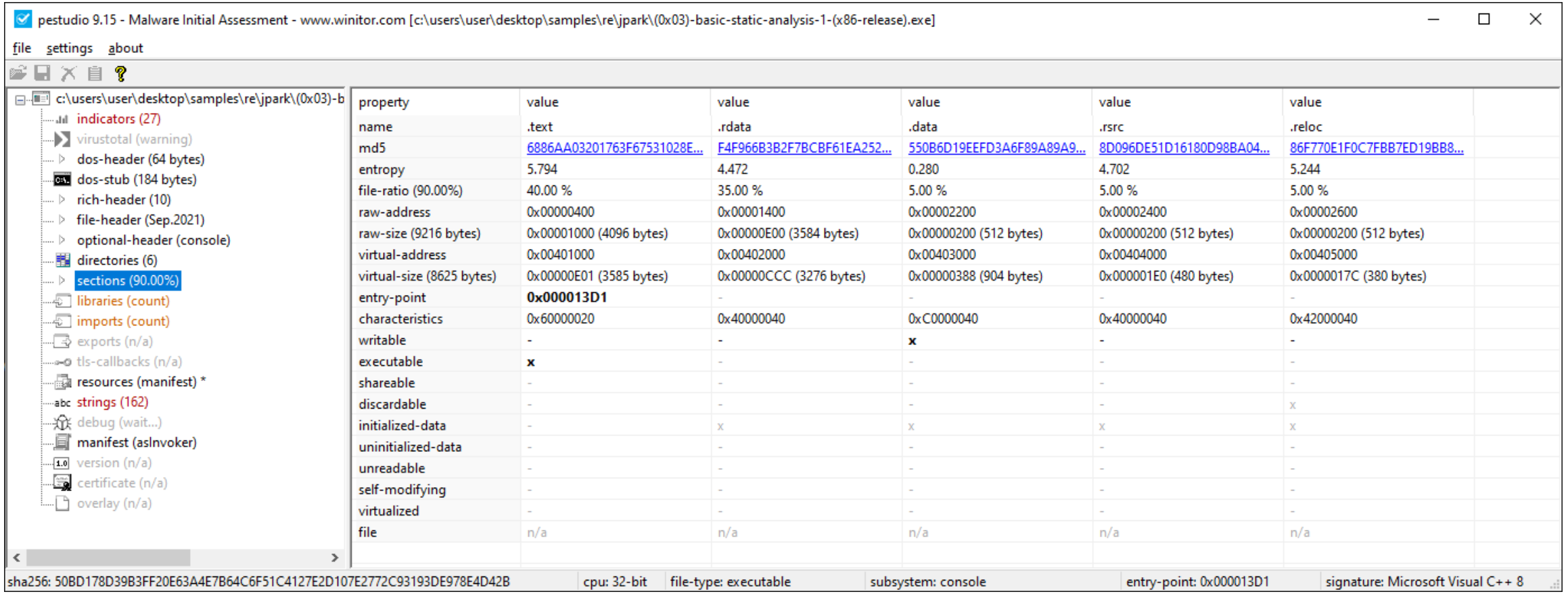
- **Common Sections of a PE File for Windows**

Section	Description	Characteristics
.text	Executable code	IMAGE_SCN_CNT_CODE | IMAGE_SCN_MEM_EXECUTE | IIMAGE_SCN_MEM_READ
.data	Initialized data	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.rdata	Read-only initialized data	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.bss	Uninitialized data	IMAGE_SCN_CNT_UNINITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.idata	Import tables	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.edata	Export tables	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.pdata	Exception information	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.reloc	Image relocations	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_DISCARDABLE
.rsrc	Resource directory	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ

Inspecting PE File Format

Examining PE Section Table And Sections

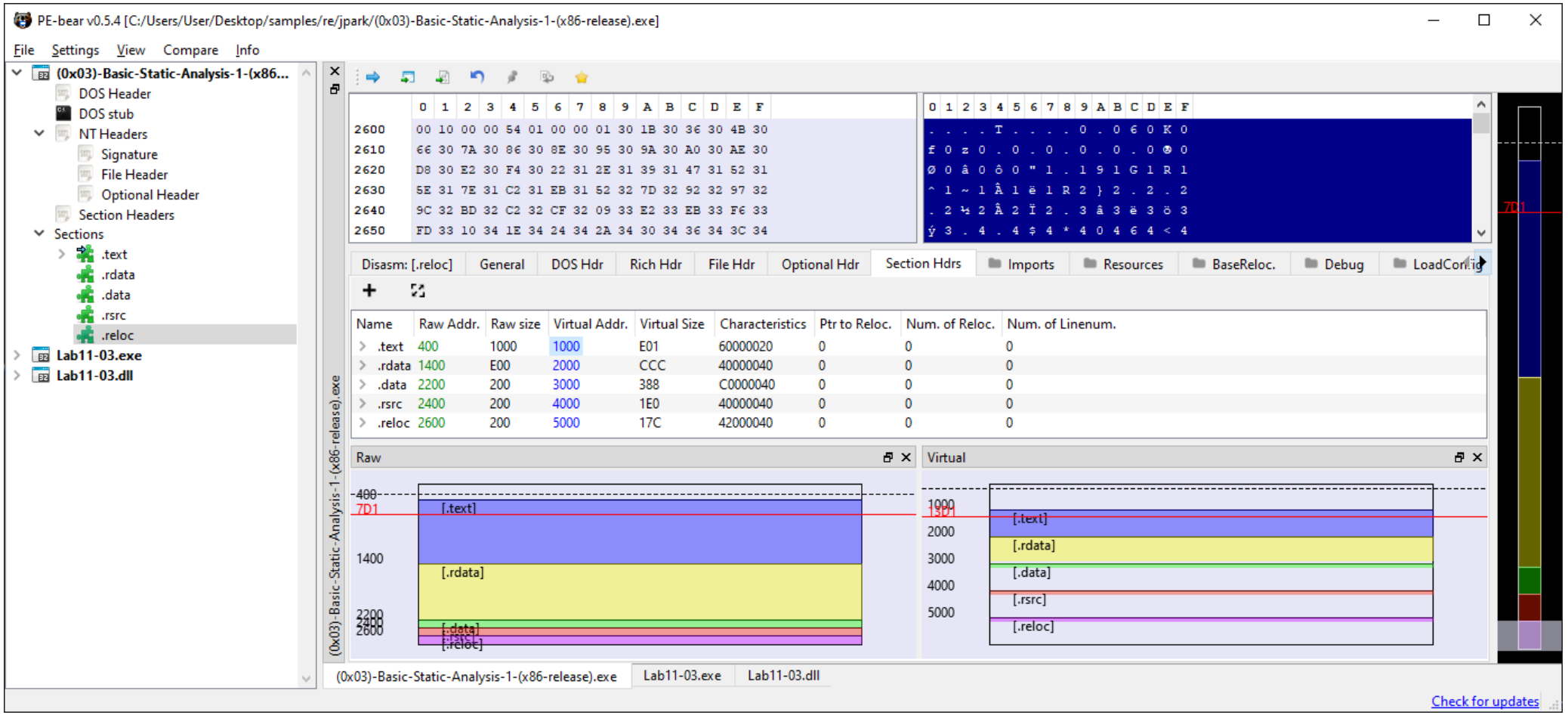
- **PE-related Tools**



Inspecting PE File Format

Examining PE Section Table And Sections

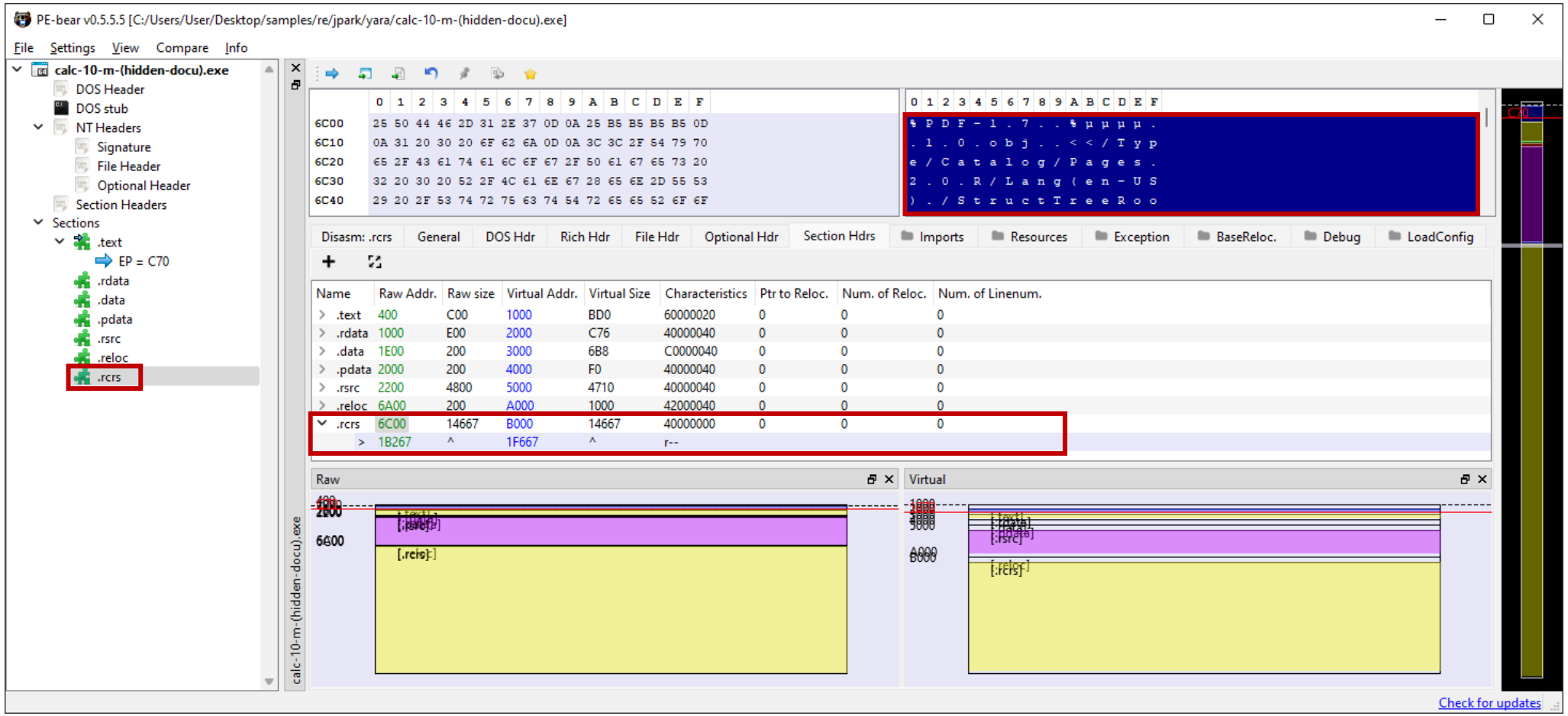
- **PE-related Tools**



Inspecting PE File Format

Examining PE Section Table And Sections

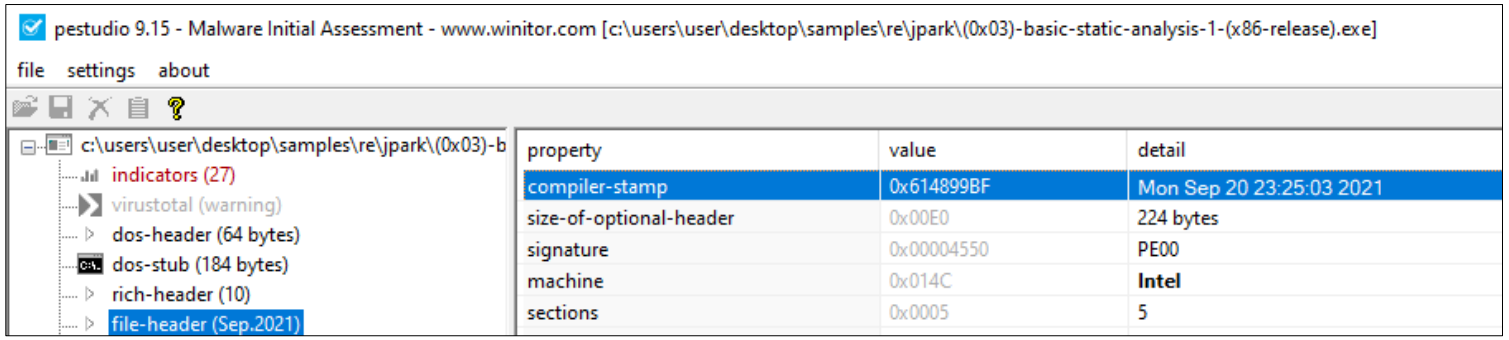
- **PE-related Tools**

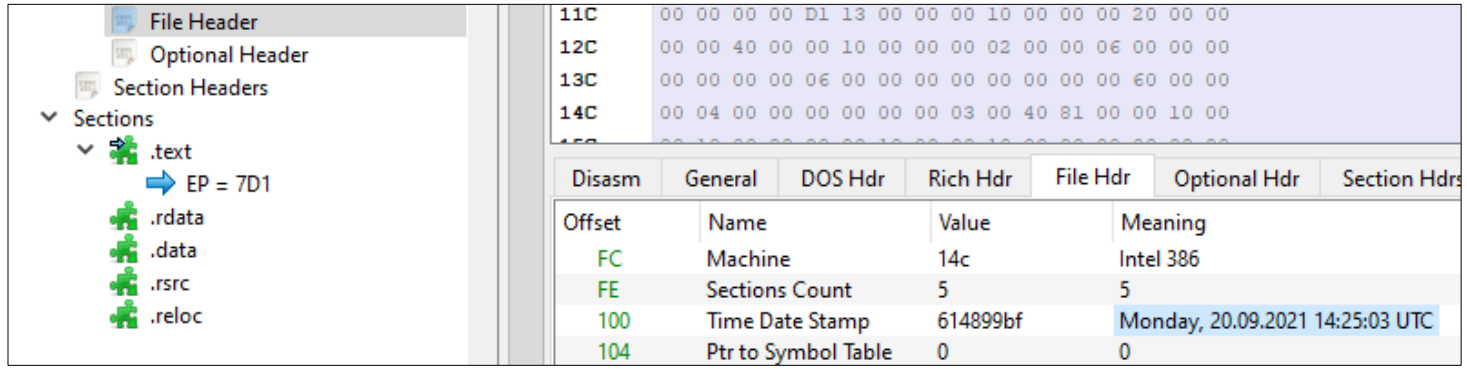


Inspecting PE File Format

Examining the Compilation Timestamp

- **Compilation Timestamp**
  - The PE header contains information that specifies when the binary was compiled
  - Examining this field can give an idea of when the malware was first created

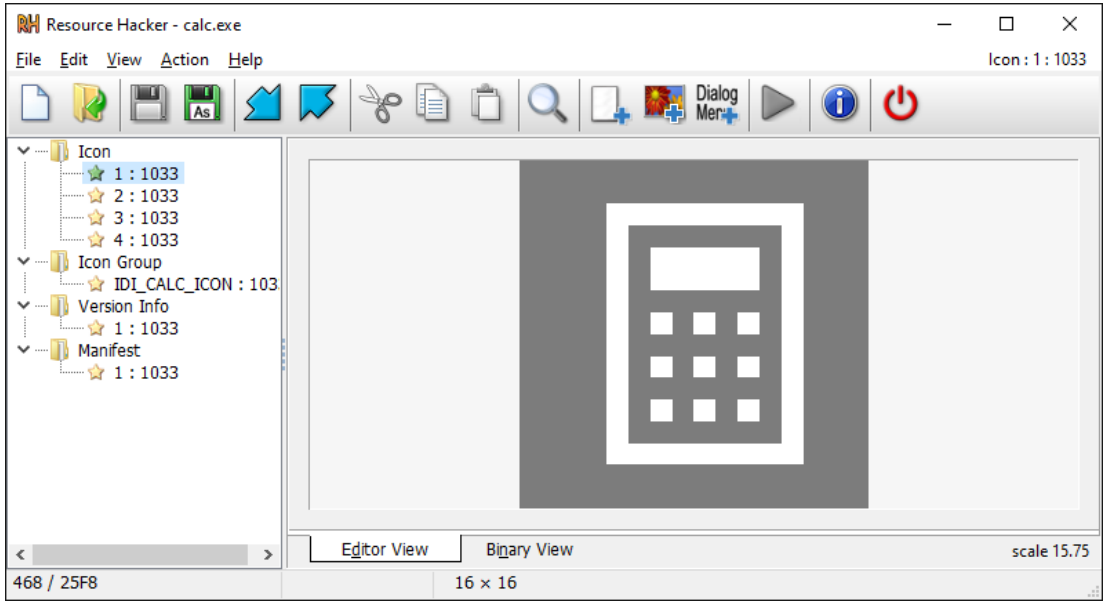


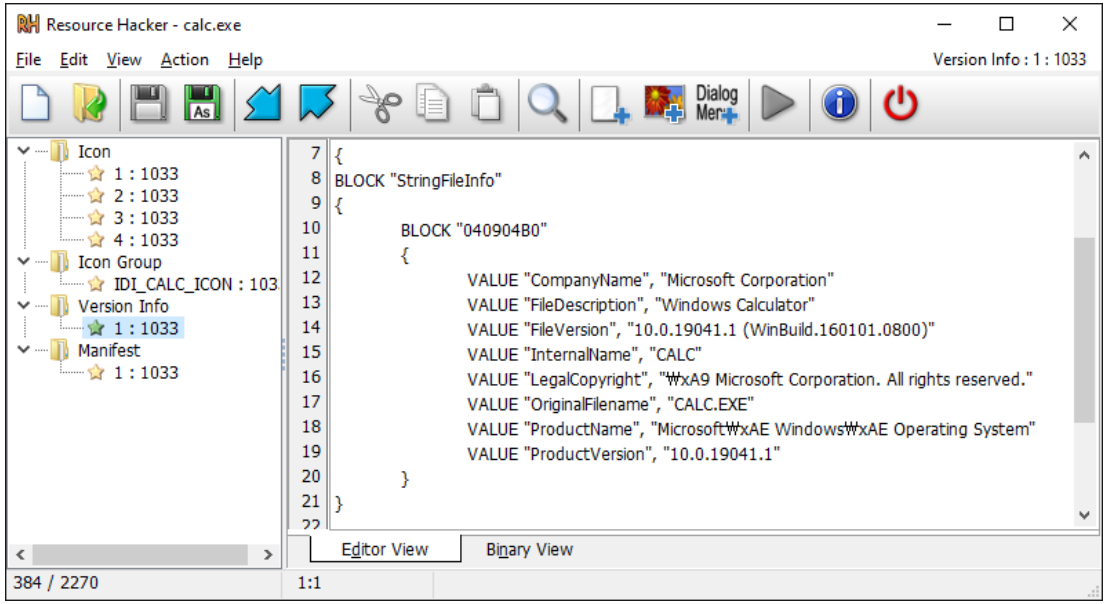


Inspecting PE File Format

Examining PE Resources

- **Resource (.rsrc) Section**
  - Icons, menu, dialog, strings, and version info. are stored in the resource section (.rsrc)
  - Often, attackers store information such as additional binary, decoy documents, and configuration data in the resource section





Comparing & Classifying Executable Files

Comparing & Classifying Executable Files

Classifying Executable Files Using Fuzzy Hashing

- **Fuzzy hashing**
  - A method to compare files for similarity
    - Useful in comparing a suspect binary with the samples in a repository
    - Useful in identifying the samples that belong to the same malware family or the same actor group
  - ssdeep
    - https://ssdeep-project.github.io/ssdeep/index.html
  - sdhash
    - https://github.com/sdhash/sdhash

Comparing & Classifying Executable Files

Classifying Executable Files Using Fuzzy Hashing

- **Fuzzy hashing**
  - ssdeep
    - https://github.com/ssdeep-project/ssdeep/releases

```
λ sha1sum calc-10*

ed13af4a0a754b8daee4929134d2ff15ebe053cd *calc-10.exe
509d9546ddcaee0bff00701516bafe617a105ceb *calc-10-m-(3bytes).exe
1bfd52f308cf680fd78269f20df1693ab7616d78 *calc-10-m-(hidden-resource).exe
```

```
λ ssdeep calc-10*

ssdeep,1.1--blocksize:hash:hash,filename
384:Otj8FKzuRxmeWCJxhd2WS/YWyiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiLiiiB:QXif4CbPQ7,"calc-10.exe"
384:ltj8FKzuRxmeWCJxhd2WS/YWyiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiLiiiB:DXif4CbPQ7,"calc-10-m-(3bytes).exe"
384:7tj8FKzuRxmeWCJxhYiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiLiiiiiriiiiI:BXif4CbJ+J,"calc-10-m-(hidden-resource).exe"
```

Comparing & Classifying Executable Files

Classifying Executable Files Using Fuzzy Hashing

- **Fuzzy hashing**
  - ssdeep
    - https://github.com/ssdeep-project/ssdeep/releases

```
λ ssdeep -pb calc-10*

calc-10-m-(3bytes).exe matches calc-10-m-(hidden-resource).exe (77)
calc-10-m-(3bytes).exe matches calc-10.exe (99)

calc-10-m-(hidden-resource).exe matches calc-10-m-(3bytes).exe (77)
calc-10-m-(hidden-resource).exe matches calc-10.exe (77)

calc-10.exe matches calc-10-m-(3bytes).exe (99)
calc-10.exe matches calc-10-m-(hidden-resource).exe (77)
```

Comparing & Classifying Executable Files

Classifying Executable Files Using Fuzzy Hashing

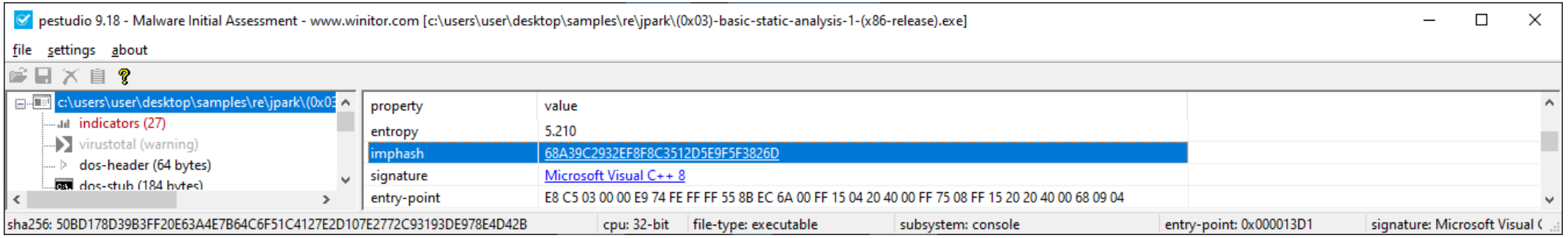
- **Fuzzy hashing**
  - ssdeep
    - https://github.com/ssdeep-project/ssdeep/releases

```
λ ssdeep calc-10-* > known_hashes.txt

----------------------------------------

λ ssdeep -m known_hashes.txt calc-10.exe

C:\Users\User\Desktop\samples\re\jpark\calc-10.exe matches
known_hashes.txt:C:\Users\User\Desktop\samples\re\jpark\calc-10-m-(3bytes).exe (99)

C:\Users\User\Desktop\samples\re\jpark\calc-10.exe matches
known_hashes.txt:C:\Users\User\Desktop\samples\re\jpark\calc-10-m-(hidden-resource).exe (77)
```

Comparing & Classifying Executable Files

Classifying Executable Files Using Import Hash

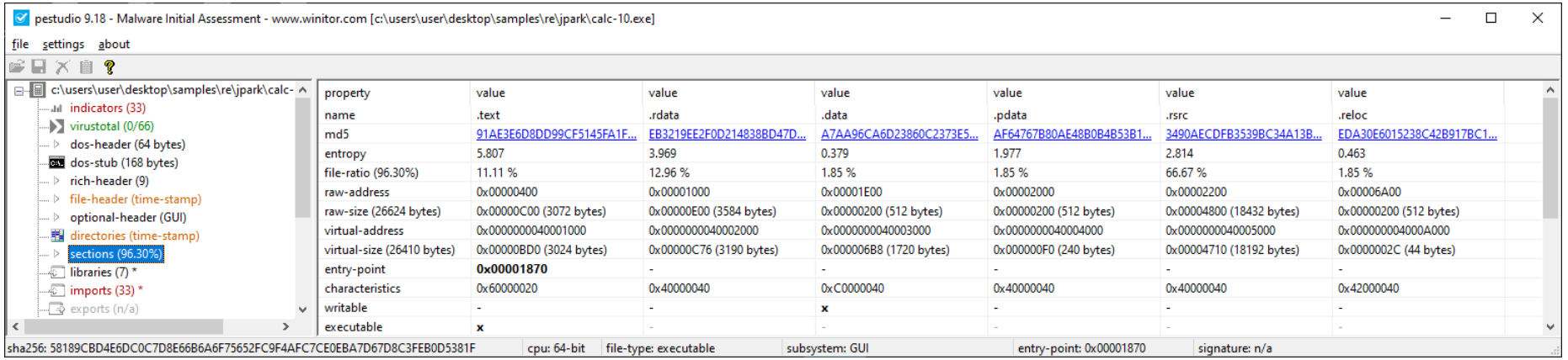
- **Import hash (or imphash)**
  - A technique in which hash values are calculated based on the library/imported function (API) names and their particular order within the executable
    - If the files were compiled from the same source and in the same manner, those files would tend to have the same **imphash** value
    - If you find samples that have the same **imphash** values, it means that they have the same import address table and are probably related

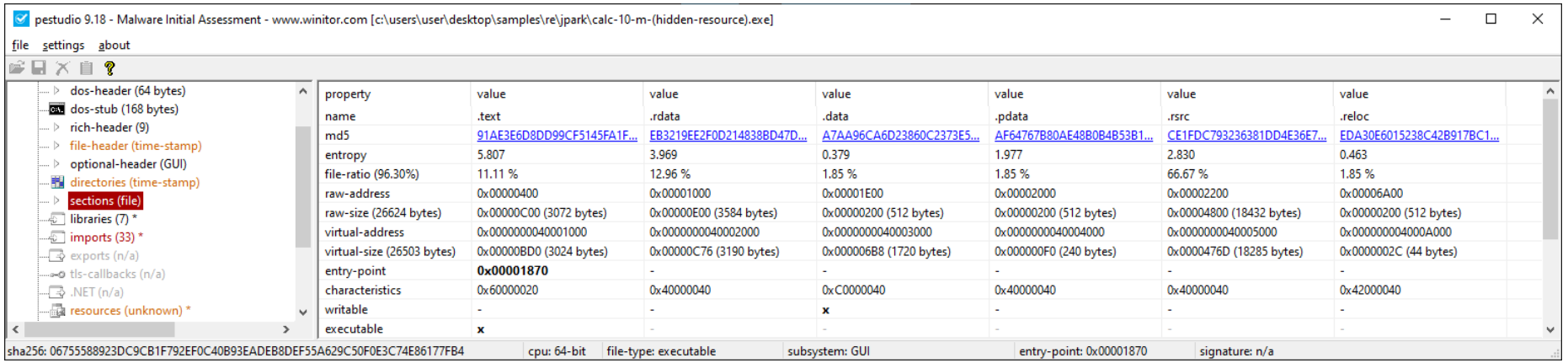


Comparing & Classifying Executable Files

Classifying Executable Files Using Section Hash

- **Section Hash**
  - Similar to import hashing, section hashing can also help in identifying related samples





Comparing & Classifying Executable Files

Classifying Executable Files Using YARA

- **Overview**
  - A malware can contain many strings or binary indicators
  - Recognizing the strings or binary data that are unique to a malware sample or a malware family can help in malware classification

- **YARA**
  - A powerful malware identification and classification tool
  - YARA rule
    - A set of strings and a Boolean expression, which determines its logic

Comparing & Classifying Executable Files

Classifying Executable Files Using YARA

- **YARA Rules**

```
rule suspicious_strings
{
     meta:
          description = "This is an example"

     strings:
          $a = {ac b9 84 bc f1 c2}
          $b = "Portscanner" ascii wide nocase
          $c = "Keylogger"   ascii wide nocase

     condition:
          ($a or $b or $c)
}
```

```
λ yara32 -r rule-basic.yara .

suspicious_strings .\(0x03)-Basic-Static-Analysis-1-(x64-release).exe
suspicious_strings .\(0x03)-Basic-Static-Analysis-1-(x86-release).exe
suspicious_strings .\rules.yara
```

Comparing & Classifying Executable Files

Classifying Executable Files Using YARA

- **YARA Rules**

```
rule suspicious_strings
{
     strings:
          $mz = {4D 5A}
          $a  = {ac b9 84 bc f1 c2}
          $b  = "Portscanner" ascii wide nocase
          $c  = "Keylogger"   ascii wide nocase

     condition:
          ($mz at 0) and ($a or $b or $c)
}
```

```
λ yara32 -r rule-basic.yara .

suspicious_strings .\(0x03)-Basic-Static-Analysis-1-(x64-release).exe
suspicious_strings .\(0x03)-Basic-Static-Analysis-1-(x86-release).exe
```

Comparing & Classifying Executable Files

Classifying Executable Files Using YARA

- **YARA Rules**

```
rule embedded_pdf
{
     meta:
          description = "Detects embedded PDF files (%PDF-1.)"

     strings:
          $mz = {4D 5A}
          $a  = {25 50 44 46 2D 31 2E ??}

     condition:
          ($mz at 0) and ($a in (512..filesize))
}
```

```
λ yara32 -r rule-pdf.yara .

embedded_pdf .\calc-10-m-(hidden-docu).exe
```

THANK YOU
for Listening!

jungheumpark@korea.ac.kr

KOREA UNIVERSITY Digital Forensic Research Center | dfrc.korea.ac.kr

Questions?