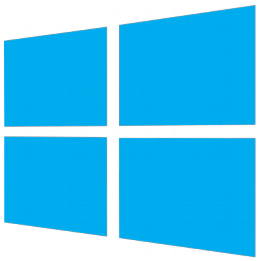
# Reverse Engineering
## Class 2

## Executable Binaries

martin.uy
# Open by default.

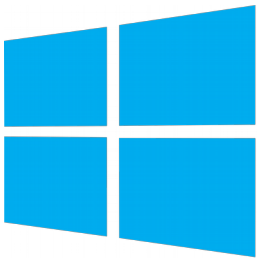
# PE



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010   B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030   00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00  ............ð...
00000040   0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050   69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060   74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070   6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080   F1 50 6A 0E B5 31 04 5D B5 31 04 5D B5 31 04 5D  ñPj.µ1.]µ1.]µ1.]
00000090   01 AD F5 5D BC 31 04 5D 01 AD F7 5D C2 31 04 5D  ..õ]¼1.]..÷]Â1.]
000000A0   01 AD F6 5D AD 31 04 5D 68 CE CF 5D B6 31 04 5D  ..ö].1.]hÎÏ]¶1.]
000000B0   B5 31 05 5D E1 31 04 5D 8E 6F 07 5C A4 31 04 5D  µ1.]á1.]Žo.\¤1.]
000000C0   8E 6F 01 5C A8 31 04 5D 8E 6F 00 5C A4 31 04 5D  Žo.\¨1.]Žo.\¤1.]
000000D0   22 6F 00 5C B4 31 04 5D 22 6F 06 5C B4 31 04 5D  "o.\´1.]"o.\´1.]
000000E0   52 69 63 68 B5 31 04 5D 00 00 00 00 00 00 00 00  Richµ1.]........
000000F0   50 45 00 00 4C 01 07 00 70 D5 DC 5A 00 00 00 00  PE..L...pÕÜZ....
00000100   00 00 00 00 E0 00 02 01 0B 01 0E 00 00 DA 04 00  ....à........Ú..
00000110   00 08 01 00 00 00 00 00 E6 15 00 00 00 10 00 00  ........æ.......
00000120   00 F0 04 00 00 00 40 00 00 10 00 00 00 02 00 00  .ð....@.........
00000130   06 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00  ................
00000140   00 30 06 00 00 04 00 00 00 00 00 00 03 00 40 81  .0............@.
00000150   00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00  ................
00000160   00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00  ................
00000170   BC D1 05 00 28 00 00 00 00 00 00 00 00 00 00 00  ¼Ñ..(...........
00000180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000190   00 00 06 00 3C 20 00 00 70 82 05 00 38 00 00 00  ....< ..p,..8...
000001A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```


**What information can be inferred from this PE at first glance?**

# PE



0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21

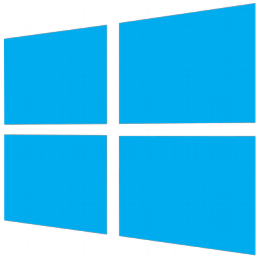
## Disassembly

```
0x0000: push cs
0x0001: pop ds
0x0002: mov dx, 0xe
0x0005: mov ah, 9
0x0007: int 0x21
0x0009: mov ax, 0x4c01
0x000c: int 0x21
```

**x86 (16 bits)**

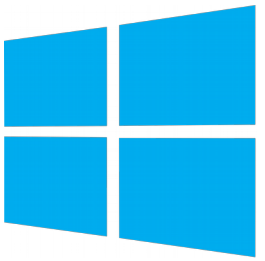

**What information can be inferred from this PE at first glance?**

# PE



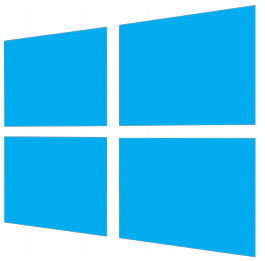
- Common Object File Format (COFF)
  - Executable binaries, objects, shared libraries
  - Binary format
  - Introduced in Unix System V
  - ELF predecessor
  - Extended to PE (also named PE/COFF)
- Portable Executable (PE) - Windows
  - Executable images (.exe, .dll, .sys)

# PE



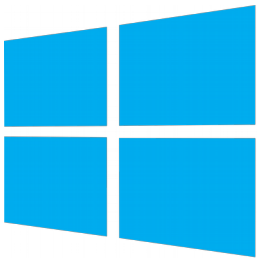
- Tools to parse or disassemble PE
  - dumpbin.exe (Visual Studio, Windows SDK)
    - Similar to objdump/readelf in Linux
    - I.e: `dumpbin.exe /ALL [file-path]`
  - CFF Explorer
  - IDA Pro
  - pefile.py
    - binary re-writing

# PE



- DOS Header (DOS Stub)
  - Present only in binary images (not in objects)
  - Magic number: MZ (0x4D, 0x5A)
  - Legacy. It’s a tiny executable binary for MS-DOS that prints “This program cannot be run in DOS mode” to *stdout*. When executing in Windows, it’s skipped.
  - In 0x3C → offset to PE Header

# PE



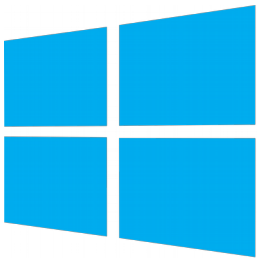
## PE

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

**WinNT.h**

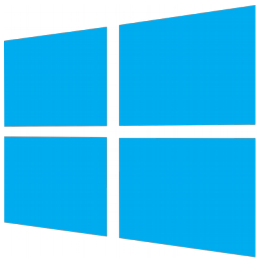
Signature magic number: PE (0x50, 0x45, 0x00, 0x00)

# PE



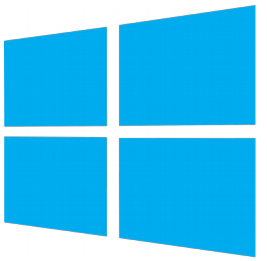
- COFF/PE File Header
  - Target architecture (I.e. Intel x86)
  - Number of sections
    - “Sections Table” size, available after File Header + Optional Headers
  - Executable creation date
  - Offset to Symbols Table

# PE



- COFF/PE File Header
  - Number of symbols (Symbols Table)
    - Useful to locate the Strings Table, after the Symbols Table
  - Optional Header size
  - Attributes
    - Is DLL? Is executable? Was debug information stripped? Is there relocation information?

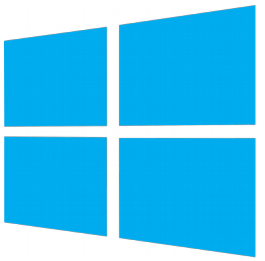
# PE



```
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;
```

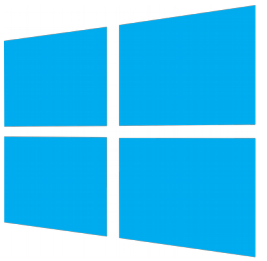
**WinNT.h**

# PE



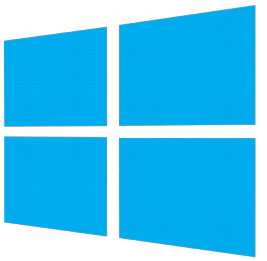
```
Dump of file main.exe

PE signature found

File Type: EXECUTABLE IMAGE

FILE HEADER VALUES
             14C machine (x86)
               5 number of sections
        5906B1BA time date stamp Mon May  1 00:55:38 2017
               0 file pointer to symbol table
               0 number of symbols
              E0 size of optional header
             102 characteristics
                   Executable
                   32 bit word machine
```

# PE



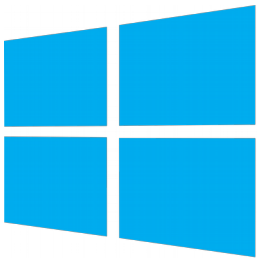
- Optional Header (binary images, not objects)
  - Entry point (offset)
  - Sizes
    - Code, initialized data, uninitialized data, executable image, heap, stack (reserved, committed  )
  - Base and alignment
    - Executable image, code, data

# PE



- Optional Header (binary images, not objects)
  - Subsystem (Win32, Linux, Posix, etc.)
    - APIs
  - Minimum and maximum version for the operating system and subsystem
  - DLL attributes (Code Integrity, NX, SEH, etc.)
  - Etc.

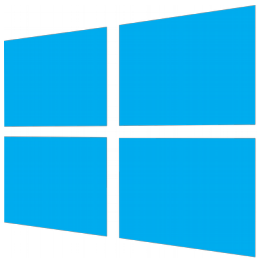
# PE



```
typedef struct _IMAGE_OPTIONAL_HEADER {
  WORD        Magic;
  BYTE        MajorLinkerVersion;
  BYTE        MinorLinkerVersion;
  DWORD       SizeOfCode;
  DWORD       SizeOfInitializedData;
  DWORD       SizeOfUninitializedData;
  DWORD       AddressOfEntryPoint;
  DWORD       BaseOfCode;
  DWORD       BaseOfData;
  DWORD       ImageBase;
  DWORD       SectionAlignment;
  DWORD       FileAlignment;
  WORD        MajorOperatingSystemVersion;
  WORD        MinorOperatingSystemVersion;
  WORD        MajorImageVersion;
  WORD        MinorImageVersion;
  WORD        MajorSubsystemVersion;
  WORD        MinorSubsystemVersion;
```

**WinNT.h**

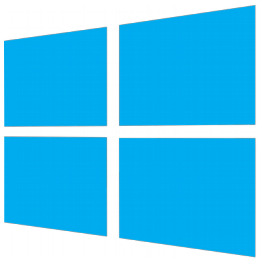
# PE



```
  DWORD           Win32VersionValue;
  DWORD           SizeOfImage;
  DWORD           SizeOfHeaders;
  DWORD           CheckSum;
  WORD            Subsystem;
  WORD            DllCharacteristics;
  DWORD           SizeOfStackReserve;
  DWORD           SizeOfStackCommit;
  DWORD           SizeOfHeapReserve;
  DWORD           SizeOfHeapCommit;
  DWORD           LoaderFlags;
  DWORD           NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY
DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

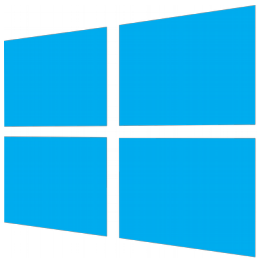
**WinNT.h**

# PE



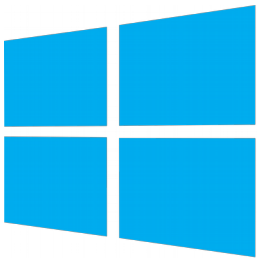
```
OPTIONAL HEADER VALUES
             10B magic # (PE32)
           14.00 linker version
            B000 size of code
            7A00 size of initialized data
               0 size of uninitialized data
            1234 entry point (00401234)
            1000 base of code
            C000 base of data
          400000 image base (00400000 to 00415FFF)
            1000 section alignment
             200 file alignment
            6.00 operating system version
            0.00 image version
            6.00 subsystem version
               0 Win32 version
           16000 size of image
             400 size of headers
               0 checksum
               3 subsystem (Windows CUI)
            8140 DLL characteristics
                   Dynamic base
                   NX compatible
                   Terminal Server Aware
```

# PE



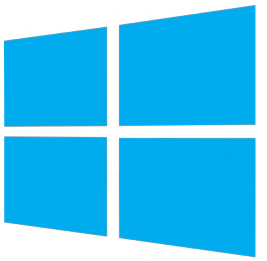
- Data Directories Header (optional)
  - Array with Relative Virtual Address (RVA) and size for different tables
    - Relative to what?
      - To the binary base in the virtual addresses space. Base assumed by the linker (available in Optional Header) or real base when loaded
      - Example: .text section location in the virtual address space
        - Assumed binary base address: 0x400000
        - .text RVA: 0x1000
        - .text address: 0x401000

# PE



- Data Directories Header (optional)
  - If binary is loaded in 0x600000, .text RVA remains (0x1000) but real address would be 0x601000.
    - RVA is different than file offset (memory vs filesystem)
  - VA (Virtual Address): absolute address in virtual memory
  - VA = base address + RVA

# PE

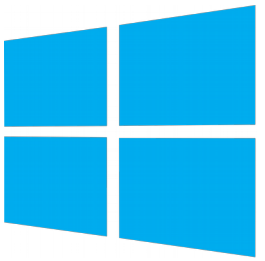


- Data Directories Header (optional)
  - Tables
    - Within “sections” (there is further information in the section, in addition to the table)
    - Available in run time (mapped to the process virtual memory)

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

**WinNT.h**

# PE

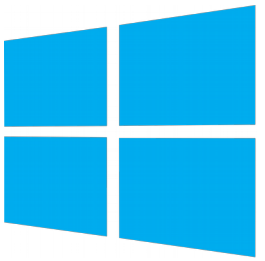


- Tables
  - Export Table
  - Import Table
  - Resource Table
  - Certificate Table
  - Import Address Table
  - Exception Table
  - Base Relocation Table
  - Thread Local Storage Table
  - Debugging Information
  - Etc.

# PE

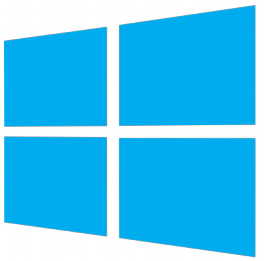
```
     0 [       0] RVA [size] of Export Directory
  1103C [      28] RVA [size] of Import Directory
     0 [       0] RVA [size] of Resource Directory
     0 [       0] RVA [size] of Exception Directory
     0 [       0] RVA [size] of Certificates Directory
  15000 [     DE0] RVA [size] of Base Relocation Directory
  109A0 [      1C] RVA [size] of Debug Directory
     0 [       0] RVA [size] of Architecture Directory
     0 [       0] RVA [size] of Global Pointer Directory
     0 [       0] RVA [size] of Thread Storage Directory
  109C0 [      40] RVA [size] of Load Configuration Directory
     0 [       0] RVA [size] of Bound Import Directory
   C000 [     108] RVA [size] of Import Address Table Directory
     0 [       0] RVA [size] of Delay Import Directory
     0 [       0] RVA [size] of COM Descriptor Directory
     0 [       0] RVA [size] of Reserved Directory
```

# PE



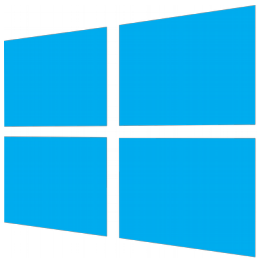
- Sections Table
  - Multiple entries that contain:
    - Name (byte[8])
    - Virtual size
      - in memory (zero padding if applies)
    - Virtual Address
      - relative to the base in executables (RVA)
    - Size of raw data
      - in file

# PE



- Sections Table
  - Raw address
    - Section offset in file
  - Section relocation offset in file (for objects)
  - Number of relocations in the section
  - Attributes
    - Is code? Is initialized data? Is executable? Can be written?

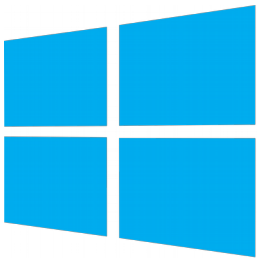
# PE



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

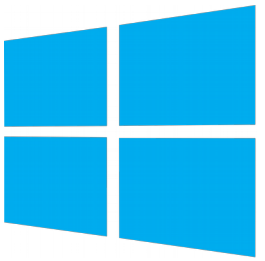
**WinNT.h**

# PE



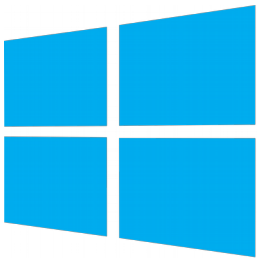
```
SECTION HEADER #3
    .data name
     11A8 virtual size
    12000 virtual address (00412000 to 004131A7)
      800 size of raw data
    10C00 file pointer to raw data (00010C00 to 000113FF)
        0 file pointer to relocation table
        0 file pointer to line numbers
        0 number of relocations
        0 number of line numbers
C0000040 flags
         Initialized Data
         Read Write
```

# PE



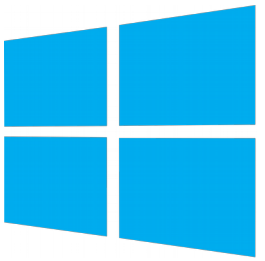
| Name | Virtual Size | Virtual Address | Raw Size | Raw Address |
|---|---|---|---|---|
| 00000238 | 00000240 | 00000244 | 00000248 | 0000024C |
| Byte[8] | Dword | Dword | Dword | Dword |
| .text | 0000AF67 | 00001000 | 0000B000 | 00000400 |
| .rdata | 0000562E | 0000C000 | 00005800 | 0000B400 |
| .data | 000011A8 | 00012000 | 00000800 | 00010C00 |
| .gfids | 000000AC | 00014000 | 00000200 | 00011400 |
| .reloc | 00000DE0 | 00015000 | 00000E00 | 00011600 |

# PE



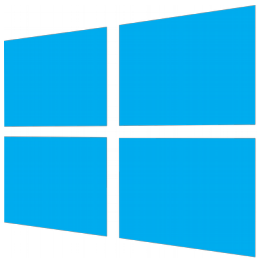
- Symbols Table (objects)
  - Sections names
  - Files names (I.e. imported DLL)
  - Variables (data)
  - Functions (code)

# PE



- Symbols Table (objects)
  - Symbol data
    - Name (if less than 8 bytes long, contained here; an offset to the Strings Table otherwise)
    - Value
      - Depends on the section and storage class but may be the virtual address for relocation
    - Section number
    - Type (I.e. function or not)
    - Storage class
      - EXTERNAL (externally defined), STATIC (section or within section), FUNCTION (beginning or end), etc.

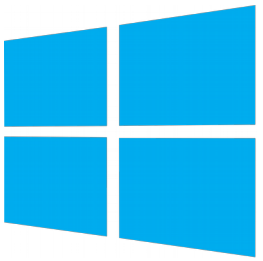
# PE



```
typedef struct {
  union {
    char e_name[E_SYMNMLEN];
    struct {
      unsigned long e_zeroes;
      unsigned long e_offset;
    } e;
  } e;
  unsigned long e_value;
  short e_scnum;
  unsigned short e_type;
  unsigned char e_sclass;
  unsigned char e_numaux;
} SYMENT;
```

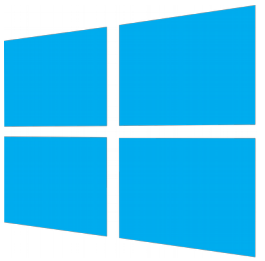
http://www.delorie.com/djgpp/doc/coff/symtab.html

# PE



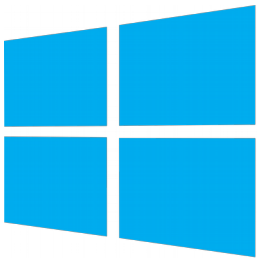
```
Dump of file main.obj

File Type: COFF OBJECT

COFF SYMBOL TABLE
000 01045E97 ABS    notype       Static       | @comp.id
001 80000191 ABS    notype       Static       | @feat.00
002 00000000 SECT1  notype       Static       | .drectve
    Section length   2F, #relocs    0, #linenums    0, checksum        0
004 00000000 SECT2  notype       Static       | .debug$S
    Section length   8C, #relocs    0, #linenums    0, checksum        0
006 00000000 SECT3  notype       Static       | .text$mn
    Section length    7, #relocs    0, #linenums    0, checksum 96F779C9
008 00000000 SECT3  notype ()    External     | _main

String Table Size = 0x0 bytes

  Summary

          8C .debug$S
          2F .drectve
           7 .text$mn
```

# PE



- Strings Table (objects)
  - Immediately after Symbols Table
  - Table size (4 bytes)
  - Null-terminated strings, referenced by symbols when 8 bytes length is exceeded

# PE



Symbol Table entry for “_printf”. String within the entry (less than 8 bytes)

```
000004E0  A0 71 00 00 02 00 00 00 2E 74 65 78 74 24 6D 6E   q.......text$mn
000004F0  00 00 00 00 06 00 00 00 03 01 29 00 00 00 02 00   ..........).....
00000500  00 00 7F B1 25 2B 00 00 02 00 00 00 2E 74 65 78   ...±%+.........tex
00000510  74 24 6D 6E 00 00 00 00 07 00 00 00 03 01 3A 00   t$mn..........:.
00000520  00 00 02 00 00 00 25 D6 E6 CA 00 00 02 00 00 00   ......%ÖæÊ......
00000530  00 00 00 00 04 00 00 00 00 00 00 00 05 00 20 00   .............. .
00000540  02 00 00 00 00 00 22 00 00 00 00 00 00 00 00 00   ......".........
00000550  20 00 02 00 00 00 00 00 33 00 00 00 00 00 00 00    .......3.......
00000560  00 00 20 00 02 00 00 00 00 00 4C 00 00 00 00 00   .. .......L.....
00000570  00 00 06 00 20 00 02 00 5F 70 72 69 6E 74 66 00   .... ..._printf.
00000580  00 00 00 00 07 00 20 00 02 00 00 00 00 00 59 00   ...... .......Y.
00000590  00 00 00 00 00 00 00 00 00 00 02 00 00 00 00 00   ................
000005A0  71 00 00 00 00 00 00 00 00 00 00 00 02 00 5F 66   q.............._f
000005B0  5F 61 00 00 00 00 00 00 00 00 00 00 20 00 02 00   _a.......... ...
000005C0  5F 6D 61 69 6E 00 00 00 00 00 00 00 04 00 20 00   _main.......... .
000005D0  02 00 00 00 00 00 87 00 00 00 08 00 00 00 00 00   ......‡.........
000005E0  00 00 02 00 BE 00 00 00 5F 5F 5F 6C 6F 63 61 6C   ....¾...___local
000005F0  5F 73 74 64 69 6F 5F 70 72 69 6E 74 66 5F 6F 70   _stdio_printf_op
00000600  74 69 6F 6E 73 00 5F 5F 5F 61 63 72 74 5F 69 6F   tions.___acrt_io
00000610  62 5F 66 75 6E 63 00 5F 5F 5F 73 74 64 69 6F 5F   b_func.___stdio_
00000620  63 6F 6D 6D 6F 6E 5F 76 66 70 72 69 6E 74 66 00   common_vfprintf.
00000630  5F 5F 76 66 70 72 69 6E 74 66 5F 6C 00 5F 5F 69   __vfprintf_l.__i
00000640  6D 70 5F 5F 47 65 74 50 72 6F 63 41 64 64 72 65   mp__GetProcAddre
00000650  73 73 40 38 00 5F 5F 69 6D 70 5F 5F 4C 6F 61 64   ss@8.__imp__Load
00000660  4C 69 62 72 61 72 79 41 40 34 00 3F 5F 4F 70 74   LibraryA@4.?_Opt
00000670  69 6F 6E 73 53 74 6F 72 61 67 65 40 3F 31 3F 3F   ionsStorage@?1??
00000680  5F 5F 6C 6F 63 61 6C 5F 73 74 64 69 6F 5F 70 72   __local_stdio_pr
00000690  69 6E 74 66 5F 6F 70 74 69 6F 6E 73 40 40 39 40   intf_options@@9@
```

Symbol Table entry for “__imp_GetProcAddress@8”

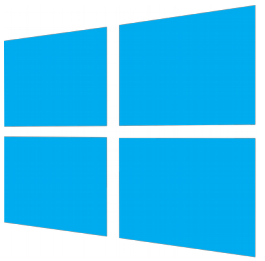
Offset to the Strings Table

Strings Table size

Offset 0x59: __imp_GetProcAddress@8

NULL terminated

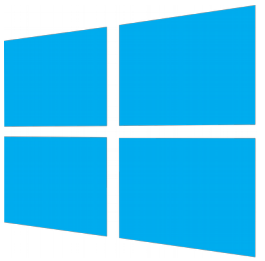
# PE



- .text section
  - Executable instructions (architecture dependent)

```
SECTION HEADER #1
    .text name
    F8DB virtual size
    1000 virtual address (00401000 to 004108DA)
    FA00 size of raw data
     400 file pointer to raw data (00000400 to 0000FDFF)
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
60000020 flags
         Code
         Execute Read
```

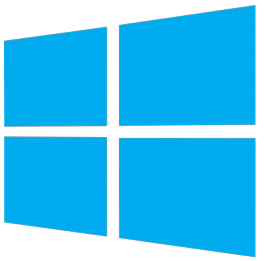
# PE



- .text section

```
Dump of file main.exe

File Type: EXECUTABLE IMAGE

  00401000: 55                 push        ebp
  00401001: 8B EC              mov         ebp,esp
  00401003: 83 EC 0C           sub         esp,0Ch
  00401006: 68 0C 80 41 00     push        41800Ch
  0040100B: FF 15 04 10 41 00  call        dword ptr ds:[00411004h]
  00401011: 89 45 FC           mov         dword ptr [ebp-4],eax
  00401014: 83 7D FC 00        cmp         dword ptr [ebp-4],0
  00401018: 74 2F              je          00401049
  0040101A: 68 18 80 41 00     push        418018h
  0040101F: 8B 45 FC           mov         eax,dword ptr [ebp-4]
  00401022: 50                 push        eax
```

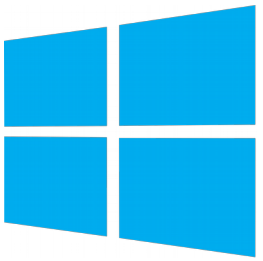
# PE



- .data and .rdata sections
  - Global variables (.data) and read-only global variables (.rdata)

```
SECTION HEADER #3
   .data name
    1210 virtual size
   18000 virtual address (00418000 to 0041920F)
     800 size of raw data
   16000 file pointer to raw data (00016000 to 000167FF)
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
C0000040 flags
         Initialized Data
         Read Write
```

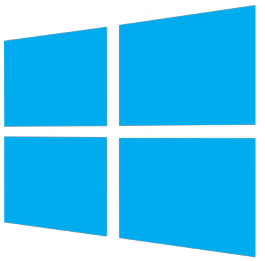
# PE



- .data section

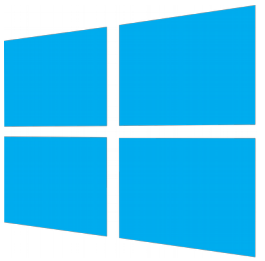
```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00016000   04 80 41 00 68 6F 6C 61 00 00 00 00 74 65 73 74  .€A.hola....test
00016010   2E 64 6C 6C 00 00 00 00 74 65 73 74 00 00 00 00  .dll....test....
00016020   52 65 74 75 72 6E 3A 20 25 64 0A 00 FF FF FF FF  Return: %d..ÿÿÿÿ
00016030   01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00016040   01 00 00 00 B1 19 BF 44 4E E6 40 BB 00 00 00 00  ....±.¿DNæ@»....
00016050   FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00  ÿÿÿÿ............
00016060   20 05 93 19 00 00 00 00 00 00 00 00 00 00 00 00   .“.............
00016070   00 00 00 00 00 00 00 00 00 00 00 00 01 20 00 00  ............. ..
00016080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00016090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000160A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

# PE



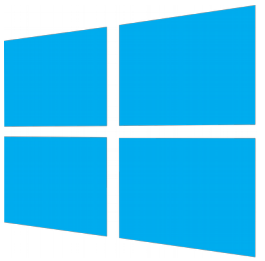
- Relocations section (.reloc)
  - Relocations in executable images (base address relocations)
  - The linker assumes a virtual address for each symbol location. If base address were different when the executable binary is loaded, it’s necessary to fix each place where the wrong assumption was done.

# PE



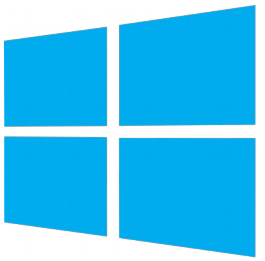
- Relocations section (.reloc)
  - How does a base address relocation work? (for an executable image)
    - Suppose that a binary has an assumed base address (when liked) of 0x400000. In RVA 0x1010 (0x401010 assumed virtual address) there is a pointer to a string in RVA 0x14002 (0x414002 assumed virtual address).

# PE



- How does a base address relocation work?
  - Suppose now that the binary was loaded to a base address of 0x600000. Thus, string is in 0x614002, 0x200000 addresses away from the assumed location. It’s necessary to update the pointer value to the correct address.
  - Relocation information allows to update in each required place the assumed address with the correct one.

# PE

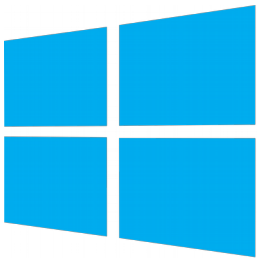


- Relocation information in .reloc section is packed in variable length blocks
- Each block has relocations for one memory page (4KB)

```
typedef struct _IMAGE_BASE_RELOCATION {
  DWORD   VirtualAddress;
  DWORD   SizeOfBlock;
} IMAGE_BASE_RELOCATION, *PIMAGE_BASE_RELOCATION;
```

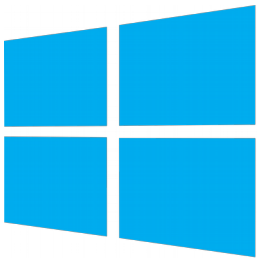
**Relocations block header**

# PE



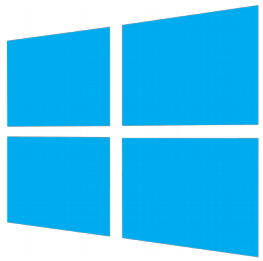
- VirtualAddress is block’s base RVA (Relative Virtual Address)
- After that, each relocation is specified with a “WORD TypeOffset”
  - 4 bits for relocation type
  - 12 bits for offset (added to the group base RVA)

# PE



- This information is enough to relocate:
  - If block's base RVA + offset + real base address are added, the exact target for relocation is obtained
  - Delta is calculated (real base – assumed base)
  - Delta is added to the value present in the relocation target

# PE



**assumed base + RVA**

```
.text:10001724
.text:10001724                          sub_10001724    proc near
.text:10001724 68 98 17 01 10                           push    offset ListHead
.text:10001729 FF 15 10 B0 00 10                        call    ds:InitializeSL
.text:1000172F C3                                       retn
.text:1000172F                          sub_10001724    endp
.text:1000172F
```

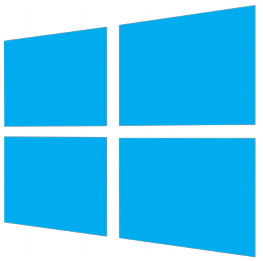
**.text (test.dll)**

**This byte's RVA is 0x1725**

```
.data:10011798                                          ; union
.data:10011798 00 00 00 00 00 00 00 00                    ListHead
.data:10011798
```

**.data (test.dll)**

# PE



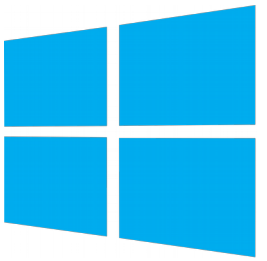
test.dll

| RVA | Size Of Block | Items |
|---|---|---|
| 00010600 | 00010604 | N/A |
| Dword | Dword | N/A |
| 00001000 | 0000010C | 130 |
| 00002000 | 000000F0 | 116 |
| 00003000 | 00000118 | 136 |
| 00004000 | 000000BC | 90 |
| 00005000 | 000000D0 | 100 |
| 00006000 | 00000070 | 52 |
| 00007000 | 000000AC | 82 |
| 00008000 | 000000B0 | 84 |
| 00009000 | 00000050 | 36 |
| 0000A000 | 00000058 | 40 |

| Item | RVA | Type |
|---|---|---|
| 0001066C | N/A | N/A |
| Word | N/A | N/A |
| 370C | 0000170C | HIGHLOW |
| 3714 | 00001714 | HIGHLOW |
| 3725 | 00001725 | HIGHLOW |
| 372B | 0000172B | HIGHLOW |
| 3731 | 00001731 | HIGHLOW |
| 373D | 0000173D | HIGHLOW |
| 3743 | 00001743 | HIGHLOW |
| 3766 | 00001766 | HIGHLOW |
| 3797 | 00001797 | HIGHLOW |
| 3842 | 00001842 | HIGHLOW |

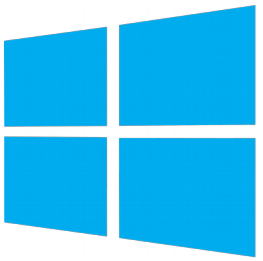
**test.dll relocations**

# PE



- Relocations in objects (COFF)
  - Different from base address relocations
  - Specific to objects
  - Useful to link objects
  - Has the following information:
    - Relocation target (VirtualAddress): section RVA + offset from the section beginning
    - Symbol index in Symbols Table
    - Type

# PE



## Relocations in objects

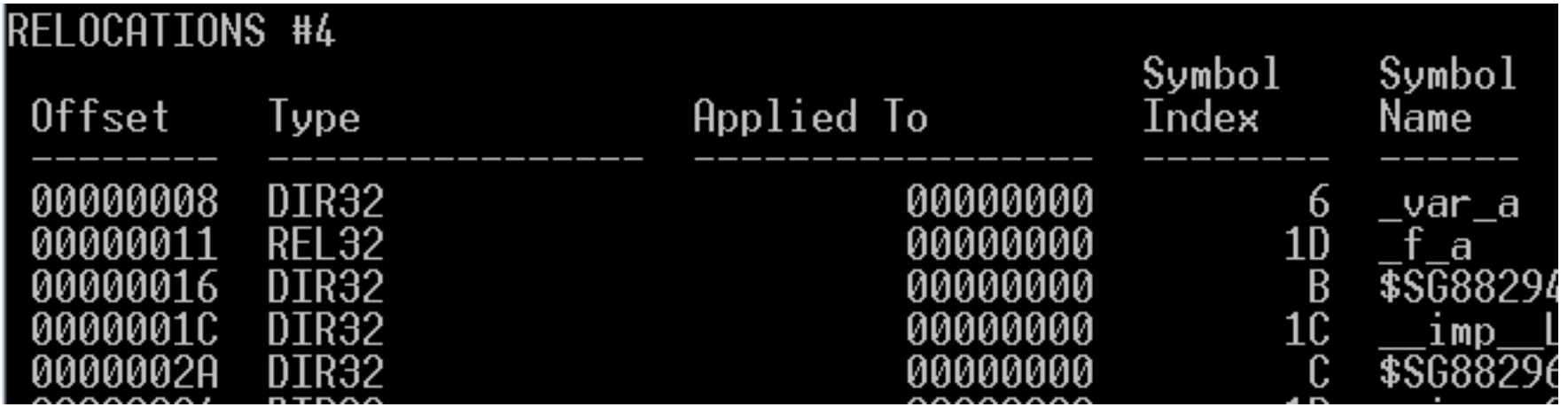
```
extern void f_a(void);

void main(void) {
    f_a();
}
```

main.c → main.obj

```
SECTION HEADER #4
.text$mn name
       0 physical address
       0 virtual address
      5E size of raw data
     257 file pointer to raw data (00000257
     2B5 file pointer to relocation table
       0 file pointer to line numbers
       8 number of relocations
       0 number of line numbers
60500020 flags
         Code
         16 byte align
         Execute Read

RAW DATA #4
  00000000: 55 8B EC 83 EC 0C C7 05 00 00 00
  00000010: E8 00 00 00 00 68 00 00 00 00 FF
  00000020: 89 45 FC 83 7D FC 00 74 2F 68 00
  00000030: FC 50 FF 15 00 00 00 00 89 45 F8
  00000040: 17 FF 55 F8 89 45 F4 8B 4D F4 51
  00000050: E8 00 00 00 00 83 C4 08 33 C0 8B
```

**CALL f_a → (0x10: E8...)**

# PE



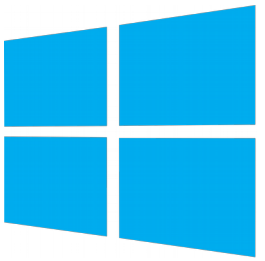
RELOCATIONS #4

| Offset | Type | Applied To | Symbol Index | Symbol Name |
|---|---|---|---|---|
| 00000008 | DIR32 | 00000000 | 6 | _var_a |
| 00000011 | REL32 | 00000000 | 1D | _f_a |
| 00000016 | DIR32 | 00000000 | B | $SG88294 |
| 0000001C | DIR32 | 00000000 | 1C | __imp__l |
| 0000002A | DIR32 | 00000000 | C | $SG88296 |

**main.obj**

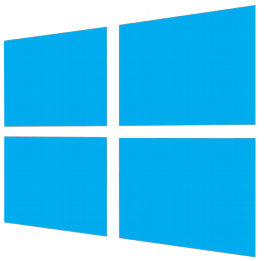
In offset 0x11 from .text section a relocation is required, when “_f_a” symbol is resolved (link time).

# PE



- Import Directory Table
  - When external variables or functions are used, it’s necessary to locate the corresponding DLLs is load time
  - Virtually every executable binary links external DLLs and have this information
  - One entry per imported DLL
    - DLL name RVA
    - DLL Import Lookup Table RVA
    - DLL Import Address Table RVA

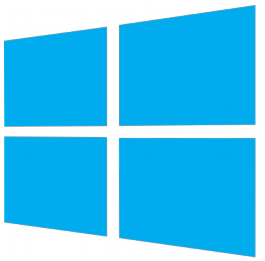
# PE



```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;            // 0 for terminating null import descriptor
        DWORD   OriginalFirstThunk;         // RVA to original unbound IAT (PIMAGE_THUNK_DATA)
    } DUMMYUNIONNAME;
    DWORD   TimeDateStamp;                  // 0 if not bound,
                                            // -1 if bound, and real date\time stamp
                                            //     in IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT (new BIND)
                                            // O.W. date/time stamp of DLL bound to (Old BIND)

    DWORD   ForwarderChain;                 // -1 if no forwarders
    DWORD   Name;
    DWORD   FirstThunk;                     // RVA to IAT (if bound this IAT has actual addresses)
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR UNALIGNED *PIMAGE_IMPORT_DESCRIPTOR;
```

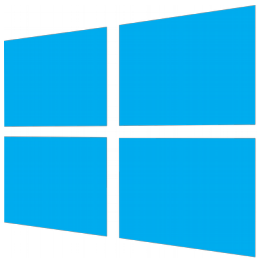
**WinNT.h**

# PE



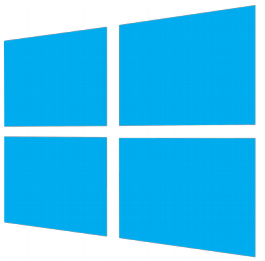
- Import Lookup Table
  - Function import by ordinal or name
  - Name RVA (if imported by name)
    - Import Name Table
  - 1 entry per imported symbol from the DLL (IMAGE_THUNK_DATA)
    - 4 bytes in x86
    - 8 bytes in x86_64

# PE



- Import Address Table (IAT)
  - Has virtual addresses of imported symbols (to be used in run time)
  - It’s filled in load time or through lazy initialization (Delay Import Address Table)
  - 1 entry per imported symbol (IMAGE_THUNK_DATA)
    - 4 bytes in x86
    - 8 bytes in x86_64

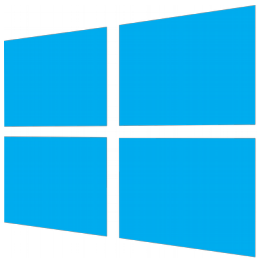
# PE



```
typedef struct _IMAGE_THUNK_DATA64 {
    union {
        ULONGLONG ForwarderString;  // PBYTE
        ULONGLONG Function;         // PDWORD
        ULONGLONG Ordinal;
        ULONGLONG AddressOfData;    // PIMAGE_IMPORT_BY_NAME
    } u1;
} IMAGE_THUNK_DATA64;
typedef IMAGE_THUNK_DATA64 * PIMAGE_THUNK_DATA64;
```

**WinNT.h**

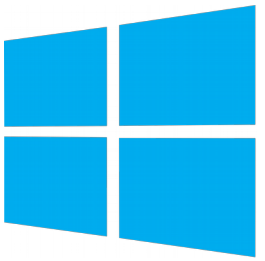
# PE



```
Section contains the following imports:

  KERNEL32.dll
              411000 Import Address Table
              416B64 Import Name Table
                   0 time date stamp
                   0 Index of first forwarder reference

                 29D GetProcAddress
                 3A5 LoadLibraryA
                 42D QueryPerformanceCounter
                 20A GetCurrentProcessId
                 20E GetCurrentThreadId
                 2D6 GetSystemTimeAsFileTime
                 34B InitializeSListHead
                 367 IsDebuggerPresent
                 582 UnhandledExceptionFilter
                 543 SetUnhandledExceptionFilter
```

**main.exe**

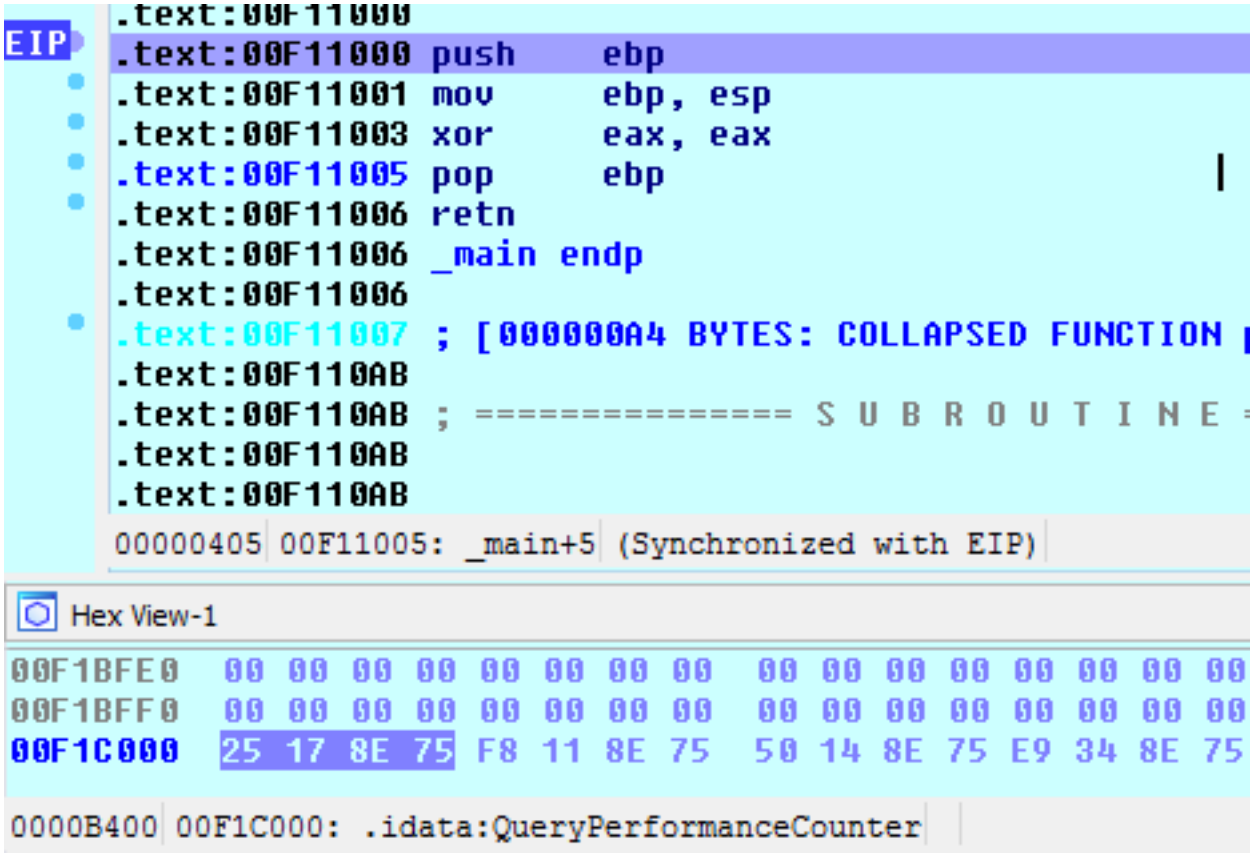
# PE



| Module Name | Imports | OFTs | TimeDateStamp | ForwarderChain | Name RVA | FTs (IAT) |
|---|---|---|---|---|---|---|
| 0001069A | N/A | 0001043C | 00010440 | 00010444 | 00010448 | 0001044C |
| szAnsi | (nFunctions) | Dword | Dword | Dword | Dword | Dword |
| KERNEL32.dll | 65 | 00011064 | 00000000 | 00000000 | 0001129A | 0000C000 |

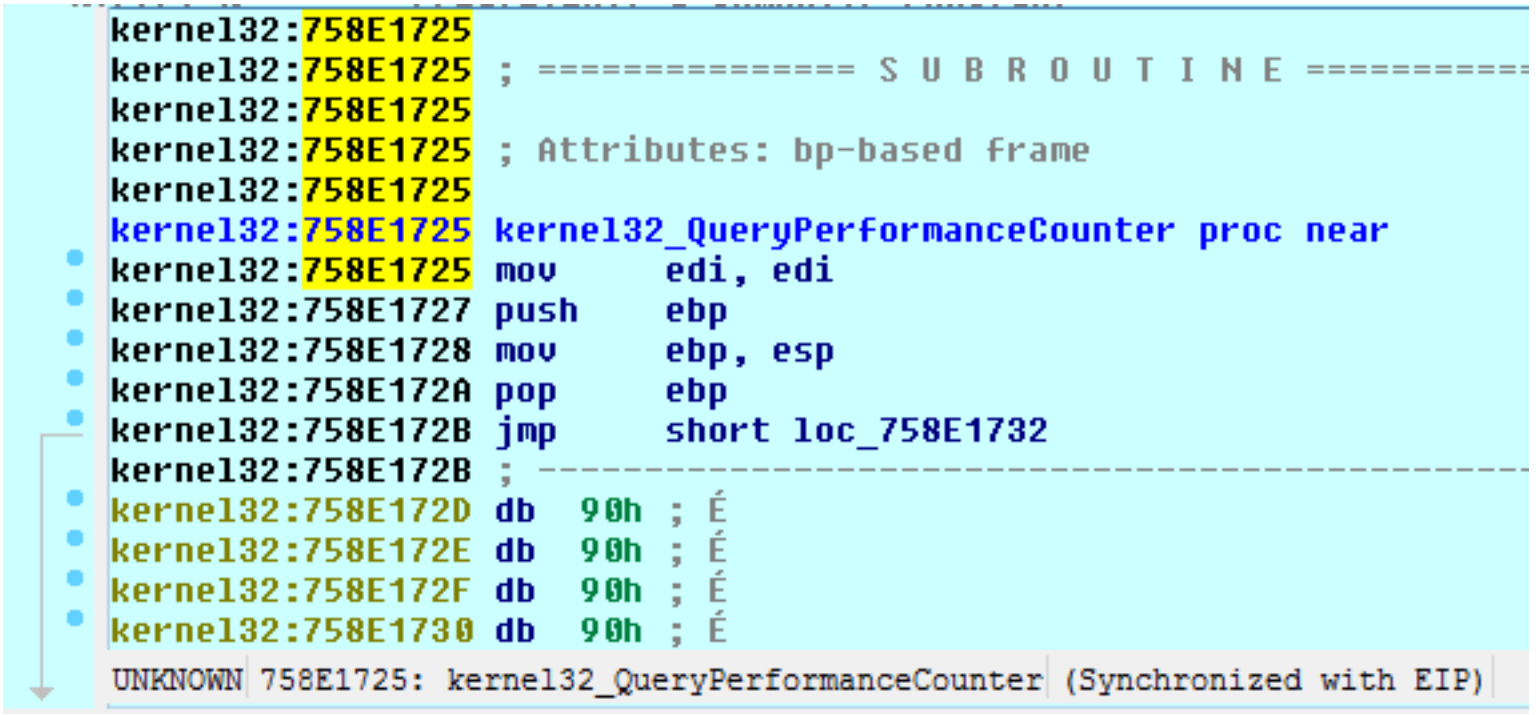
| OFTs | FTs (IAT) | Hint | Name |
|---|---|---|---|
| | | | |
| Dword | Dword | Word | szAnsi |
| 0001116C | 0001116C | 042D | QueryPerformanceCounter |
| 00011186 | 00011186 | 020A | GetCurrentProcessId |
| 0001119C | 0001119C | 020E | GetCurrentThreadId |
| 000111B2 | 000111B2 | 02D6 | GetSystemTimeAsFileTime |
| 000111CC | 000111CC | 034B | InitializeSListHead |
| 000111E2 | 000111E2 | 0367 | IsDebuggerPresent |
| 000111F6 | 000111F6 | 0582 | UnhandledExceptionFilter |
| 00011212 | 00011212 | 0543 | SetUnhandledExceptionFilter |
| 00011230 | 00011230 | 02BE | GetStartupInfoW |
| 00011242 | 00011242 | 036D | IsProcessorFeaturePresent |
| 0001125E | 0001125E | 0267 | GetModuleHandleW |
| 00011272 | 00011272 | 0209 | GetCurrentProcess |
| 00011286 | 00011286 | 0561 | TerminateProcess |
| 000112A8 | 000112A8 | 04AD | RtlUnwind |
| 000112B4 | 000112B4 | 0250 | GetLastError |

# PE



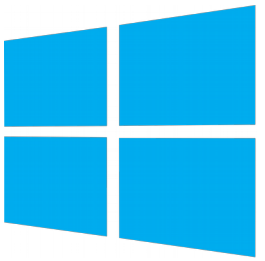

**kernel32.dll IAT**

# PE



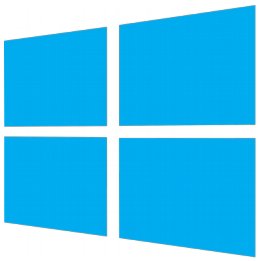

**IAT 1st entry for kernel32.dll in main.exe**

# PE



- .edata section
  - Export symbols from a DLL to be used by other executable images in run time
  - Export Directory Table
    - DLL name RVA
    - Export Address Table RVA and number of entries
    - Name Pointer Table RVA and number of entries
    - Ordinal Table RVA and number of entries
  - Export Address Table
    - Array with exported symbols RVAs (in .data and .text)

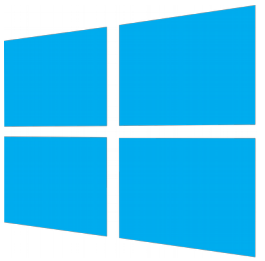
# PE



```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    DWORD   Name;
    DWORD   Base;
    DWORD   NumberOfFunctions;
    DWORD   NumberOfNames;
    DWORD   AddressOfFunctions;     // RVA from base of image
    DWORD   AddressOfNames;         // RVA from base of image
    DWORD   AddressOfNameOrdinals;  // RVA from base of image
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

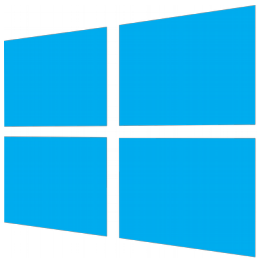
**WinNT.h**

# PE



- .edata section
  - Name Pointer Table
    - Array with pointers to exported symbols names (strings)
  - Ordinal table
    - Array with exported symbols indices in Export Address Table. Correspondence to Name Pointer Table by position
  - Export Name Table: null-terminated strings with exported symbols names

# PE

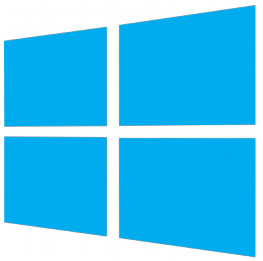


test.dll

| Member | Offset | Size | Value |
|---|---|---|---|
| Characteristics | 0000F410 | Dword | 00000000 |
| TimeDateStamp | 0000F414 | Dword | 5A52E591 |
| MajorVersion | 0000F418 | Word | 0000 |
| MinorVersion | 0000F41A | Word | 0000 |
| Name | 0000F41C | Dword | 00010042 |
| Base | 0000F420 | Dword | 00000001 |
| NumberOfFunctions | 0000F424 | Dword | 00000001 |
| NumberOfNames | 0000F428 | Dword | 00000001 |
| AddressOfFunctions | 0000F42C | Dword | 00010038 |

| Ordinal | Function RVA | Name Ordinal | Name RVA | Name |
|---|---|---|---|---|
| | | | | |
| (nFunctions) | Dword | Word | Dword | szAnsi |
| 00000001 | 00001000 | 0000 | 0001004B | test |

# PE



**test.dll**

Export Directory Table

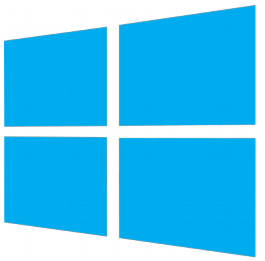
```
1000FFE0: FE FF FF FF 00 00 00 00 47 8F 00 10 00 00 00 00  þÿÿÿ....G.......
1000FFF0: FE FF FF FF 00 00 00 00 D8 FF FF FF 00 00 00 00  þÿÿÿ....Øÿÿÿ....
10010000: FE FF FF FF 19 9E 00 10 2C 9E 00 10 00 00 00 00  þÿÿÿ....,.......
10010010: 00 00 00 00 91 E5 52 5A 00 00 00 00 42 00 01 00  ......åRZ....B...
10010020: 01 00 00 00 01 00 00 00 01 00 00 00 38 00 01 00  ............8...
10010030: 3C 00 01 00 40 00 01 00 00 10 00 00 4B 00 01 00  <...@.......K...
10010040: 00 00 74 65 73 74 2E 64 6C 6C 00 74 65 73 74 00  ..test.dll.test.
10010050: 78 00 01 00 00 00 00 00 00 00 00 00 AA 02 01 00  x...........ª...
10010060: 00 B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .°..............
10010070: 00 00 00 00 00 00 00 00 7C 01 01 00 96 01 01 00  ........|.......
10010080: AC 01 01 00 C2 01 01 00 DC 01 01 00 F2 01 01 00  ¬...Â...Ü...ò...
```

Ordinal Table (1st entry → index 0x0)

Export Address Table (1st entry → RVA 0x1000)

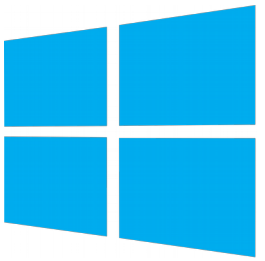
Name Pointer Table (1st entry → RVA 0x01004B)

# PE



- .edata section
  - If the dynamic linkers has to resolve “func_a” function address in “X” DLL, how can it leverage on previously described information to do it?

# PE



- .edata section
  - Iterate the Name Pointer Table
    - Compare each string with the looked string until there is a match
  - Use the position in Name Pointer Table where the match occurred to get the function index in the Ordinal Table
  - Use the function index get the function RVA from the Export Address Table

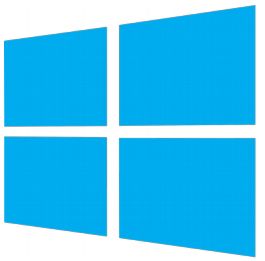
# Demo 2.1

Import Directory and Import Address Table

# Demo 2.2

Access to Export Address Table

# PE



- `(windbg) lm`
- `(windbg) !dl <dll_base> -f`
  - Locate IAT
- `(windbg) dps <dll_base>+<IAT_offset>`

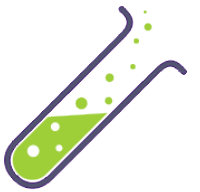
# Lab 2.1

- Develop a program in C/C++ that reads the Import Table and IAT from a loaded DLL to print in *stdout*:
  - Imported DLL name
  - An imported function name
  - Function address in IAT
  - Function value in IAT (*iat-address)

  Tip: PE structures are defined in winnt.h

# Lab 2.2

- Patch IAT and replace the function from Lab 1.1 with another one.
- Invoke the function before and after the patch.



# References


- https://support.microsoft.com/en-us/help/121460/common-object-file-format-coff
- https://msdn.microsoft.com/en-us/library/windows/desktop/ms680547(v=vs.85).aspx
- https://github.com/erocarrera/pefile
- https://msdn.microsoft.com/en-us/library/ms809762.aspx