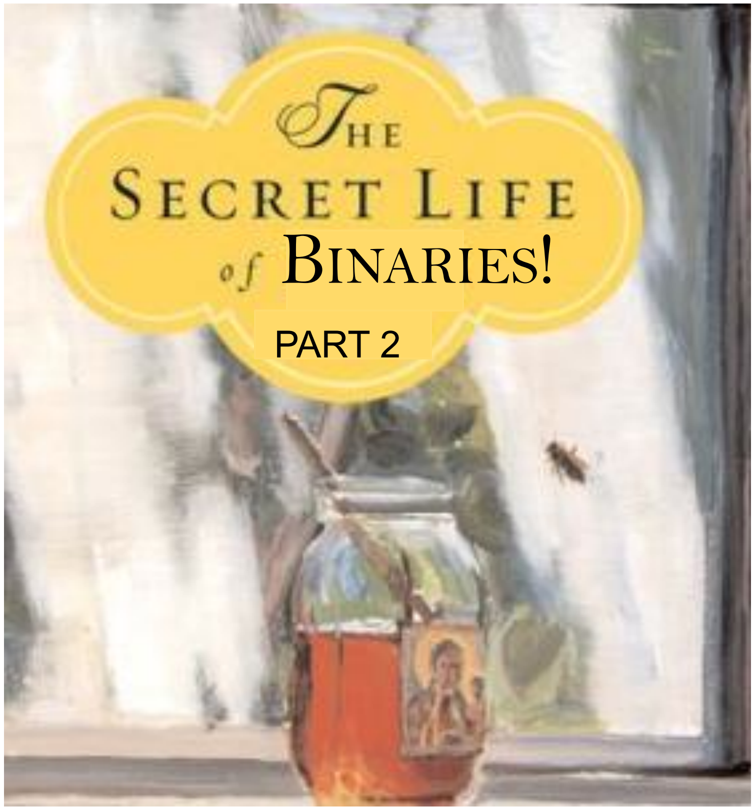


The Secret Life of Binaries!

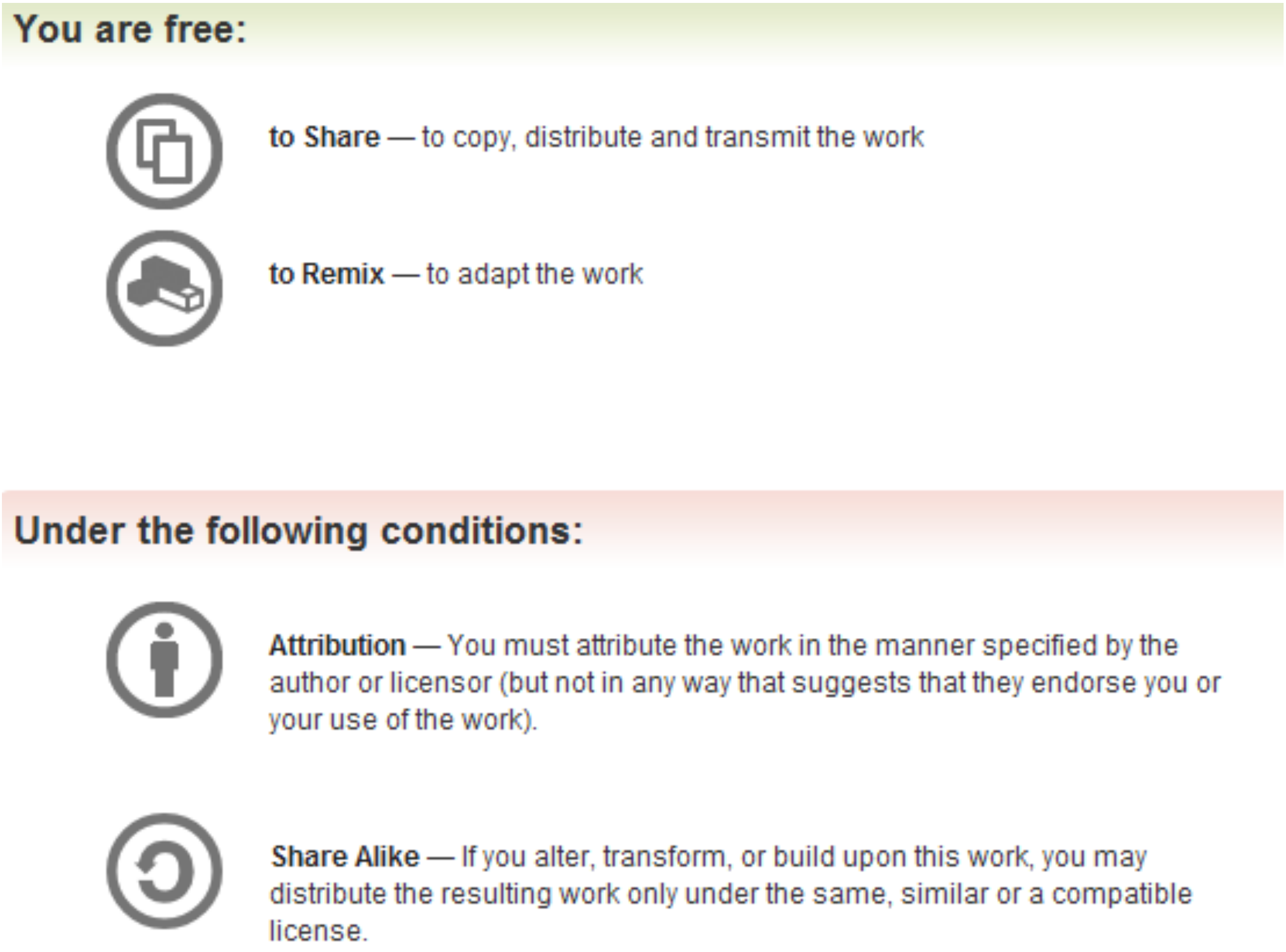
PART 2

Xeno Kovah - 2010

xkovah at gmail

All materials are licensed under a Creative Commons “Share Alike” license.

- http://creativecommons.org/licenses/by-sa/3.0/



You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

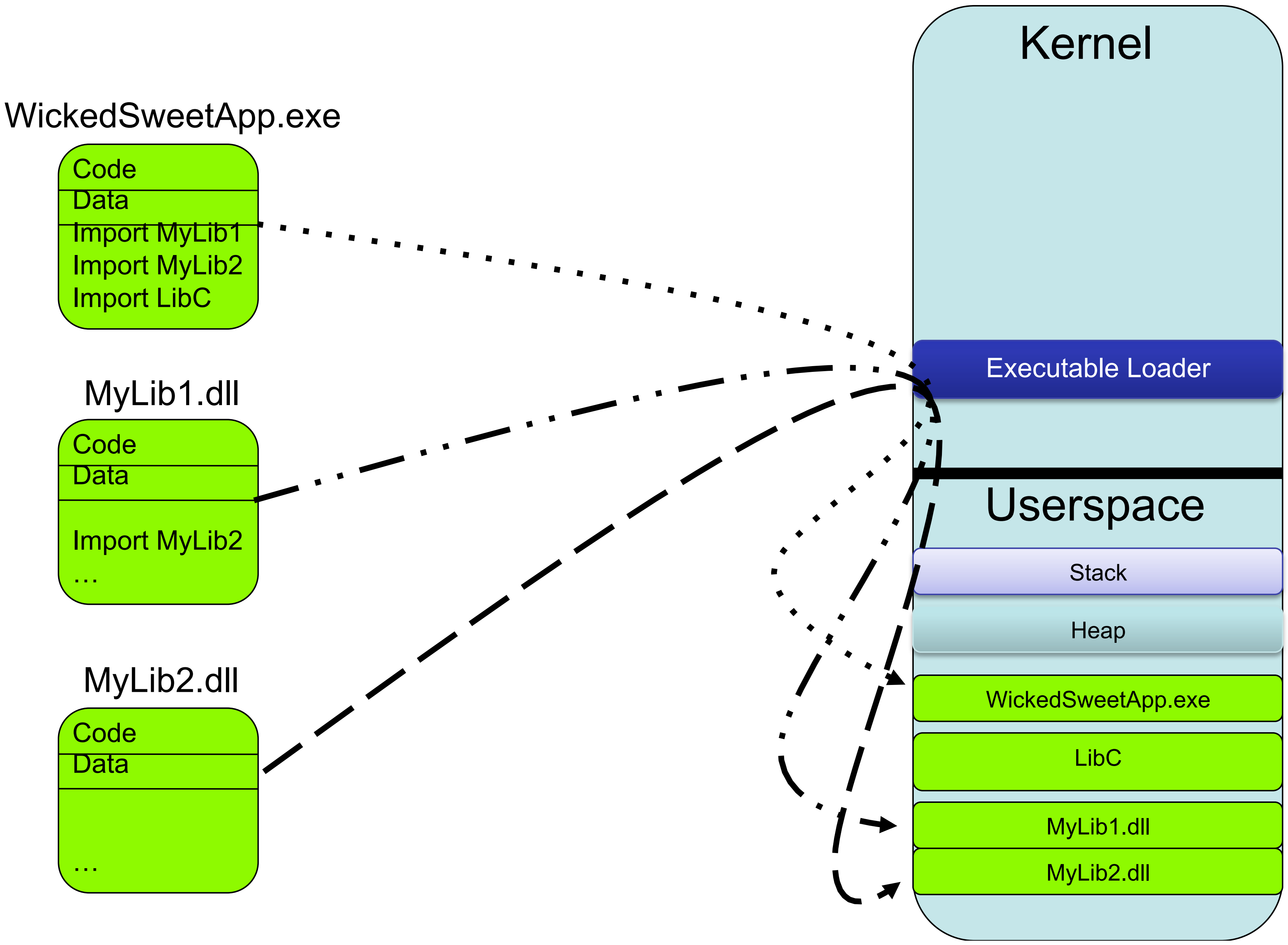
Executable Formats

- Common Object File Format (COFF) was introduced with UNIX System V.
- Windows has Portable Executable (PE) format. Derived from COFF.
- Modern unix derivatives tend to use the Executable and Linkable Format (ELF).
- Mac OS X uses the Mach Object (Mach-o) format.

Different target binary formats

- Executable (.exe on Windows, no suffix on Linux)
  - A program which will either stand completely on its own, containing all code necessary for its execution, or which will request external libraries that it will depend on (and which the loader must provide for the executable to run correctly)
- Dynamic Linked Library (.dll) on Windows == Shared Library aka Shared Object (.so) on Linux
  - Needs to be loaded by some other program in order for any of the code to be executed. The library *may* have some code which is automatically executed at load time (the DllMain() on windows or init() on Linux). This is as opposed to a library which executes none of its own code and only provides code to other programs.
- Static Library (.lib on Windows, .a on Linux)
  - Static libraries are just basically a collection of object files, with some specific header info to describe the organization of the files.

Loader Overview



Files on Disk

WickedSweetApp.exe

Code
Data
Import MyLib1
Import MyLib2
Import LibC

MyLib1.dll

Code
Data
Import MyLib2
…

MyLib2.dll

Code
Data
…

Virtual Memory
Address Space

Kernel

Executable Loader

Userspace

Stack

Heap

WickedSweetApp.exe

LibC

MyLib1.dll

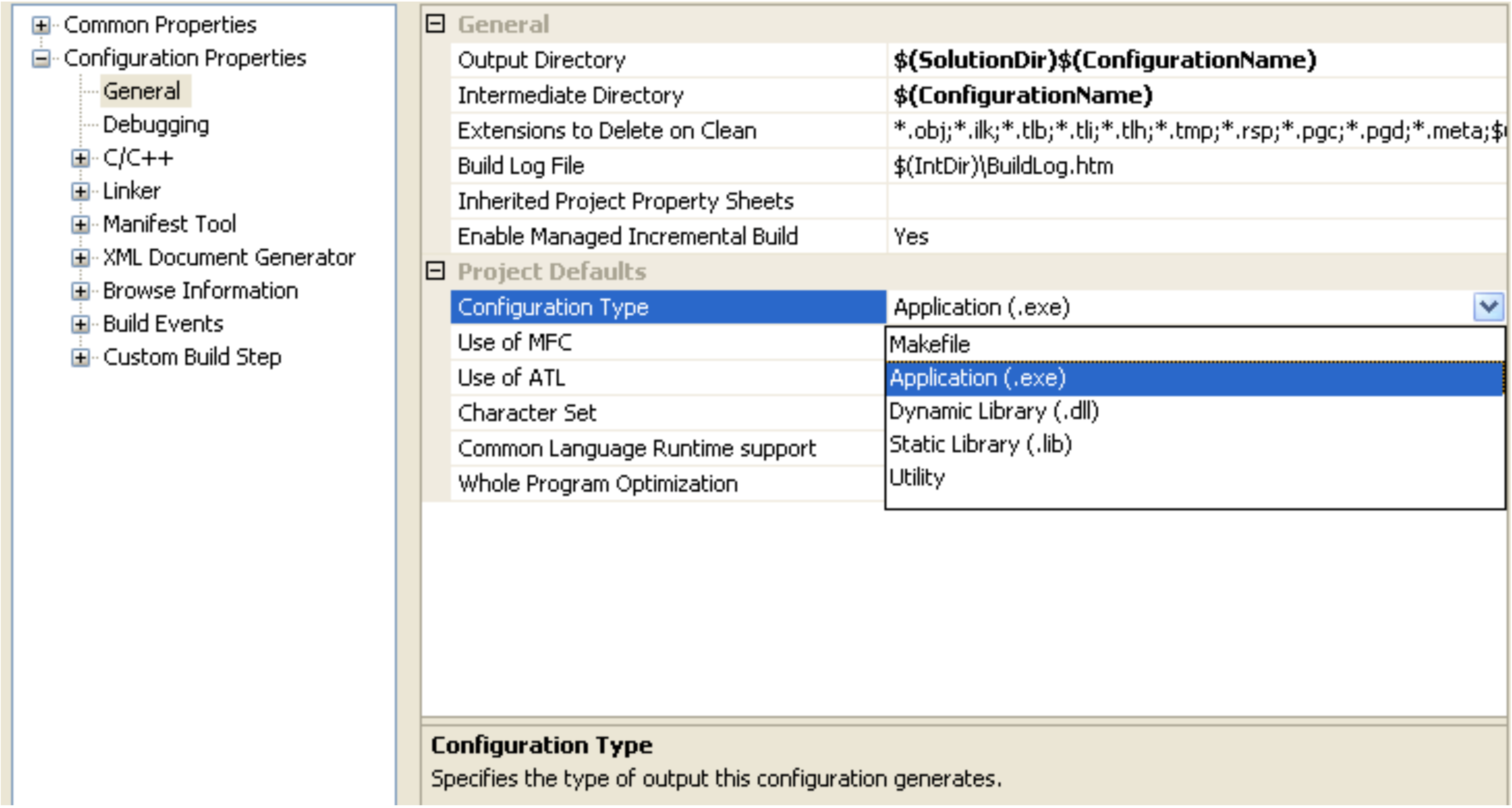
MyLib2.dll

Common Windows PE File Extensions

- .exe - Executable file
- .dll - Dynamic Link Library
- .sys/.drv - System file (Kernel Driver)
- .ocx - ActiveX control
- .cpl - Control panel
- .scr - Screensaver

- Note: .lib files (Static Libraries) don't have the same "DOS Header then PE Header" format that the rest of these do.

Building Windows Executable, Dynamic Linked Library, Static Library



General

Output Directory	**$(SolutionDir)$(ConfigurationName)**
Intermediate Directory	**$(ConfigurationName)**
Extensions to Delete on Clean	*.obj;*.ilk;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta;$
Build Log File	$(IntDir)\BuildLog.htm
Inherited Project Property Sheets	
Enable Managed Incremental Build	Yes

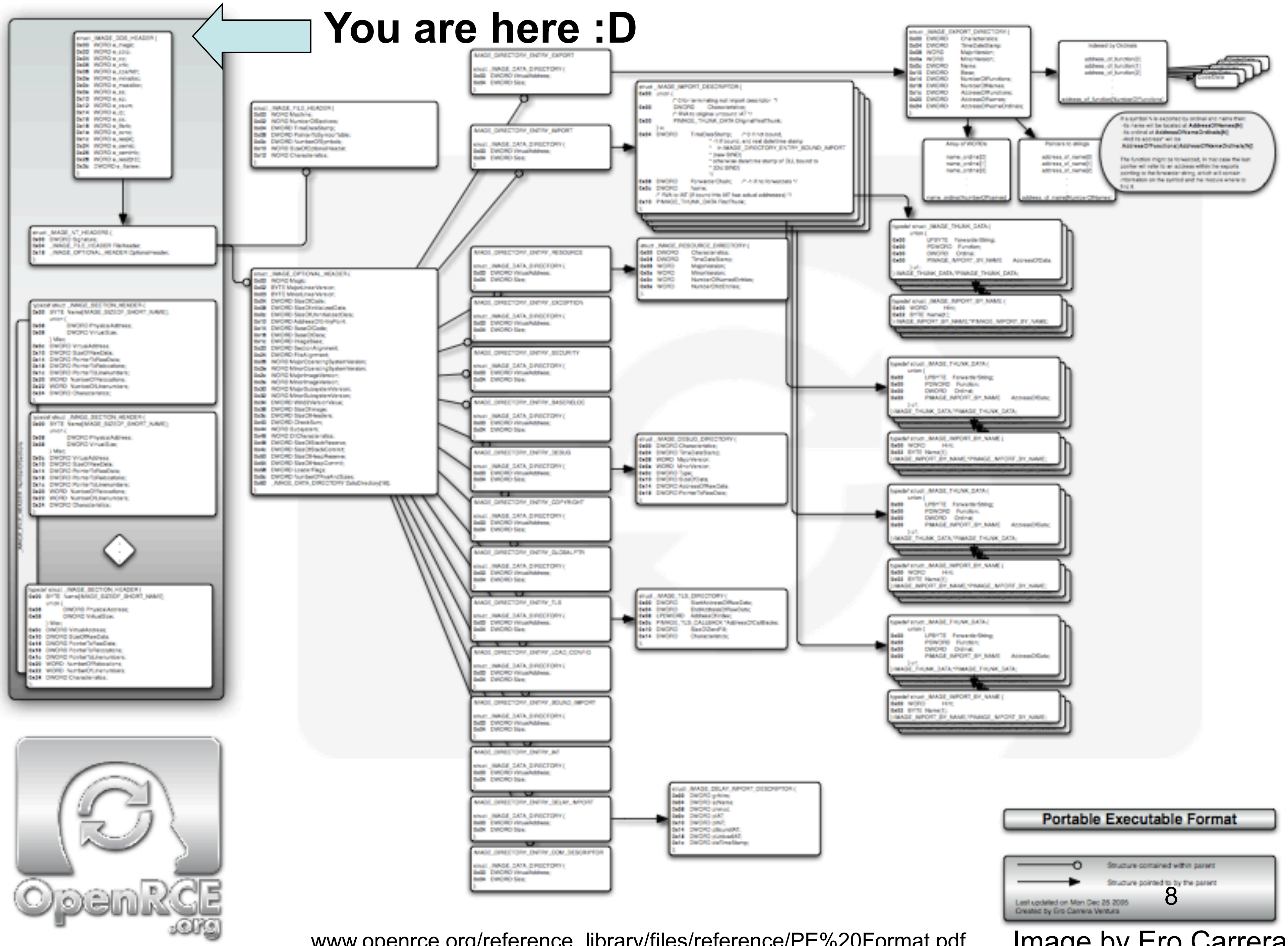
Project Defaults

Configuration Type	Application (.exe)
Use of MFC	
Use of ATL	
Character Set	
Common Language Runtime support	
Whole Program Optimization	

Configuration Type

Specifies the type of output this configuration generates.

You are here :D


Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE.org

www.openrce.org/reference_library/files/reference/PE%20Format.pdf

Image by Ero Carrera

Further Reading

- The definitions of all of the structures for a PE file are in WINNT.h
- An In-Depth Look into the Win32 Portable Executable File Format Part 1 & 2 – An excellent set of reference articles by Matt Pietrek (this is how I first learned) http://msdn.microsoft.com/en-us/magazine/cc301805.aspx, http://msdn.microsoft.com/en-us/magazine/cc301808.aspx
- The official spec:

  http://www.microsoft.com/whdc/system/platform/firmware/pecoff.mspx
- All the VisualStudio compiler options (note, some aren't in the GUI, you have to add them manually): http://msdn.microsoft.com/en-us/library/fwkeyyhe(v=VS.90).aspx
- All the VS linker options: http://msdn.microsoft.com/en-us/library/y0zzbyt4(v=VS.90).aspx

Your new best friends: PEView and CFF Explorer

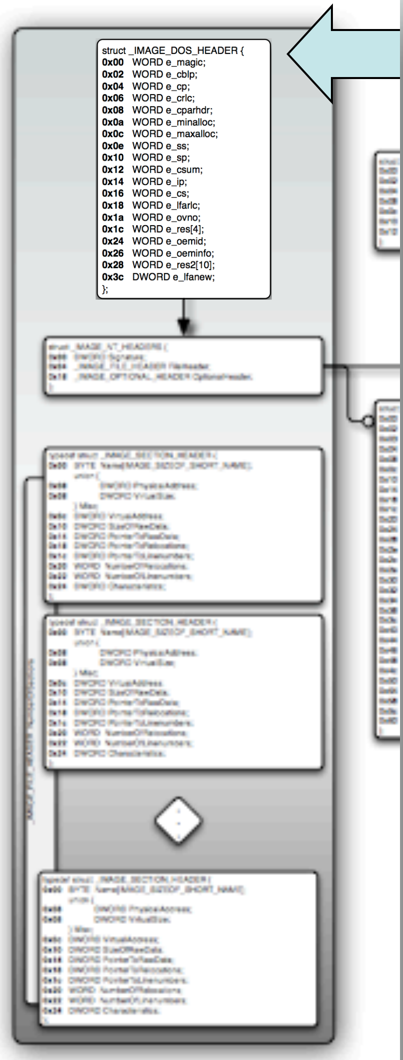
- I like PEView (http://www.magma.ca/~wjr/PEview.zip) by Wayne Radburn for looking at PE files. It's no frills and gives you a view very close to what you would see if you were looking at the structs in a program which was parsing the file.
- Once you've seen and understood stuff in PEView, you can graduate to the much more feature-full CFF Explorer by Daniel Pistelli (it lets you hex edit the file or disassemble code! :D) (http://www.ntcore.com/exsuite.php)

Tools: WinDbg

- We're going to be using WinDbg for basic userspace debugging (as opposed to kernel debugging like in the Intermediate x86 class)

Terminology

- RVA - Relative Virtual Address. This indicates some displacement relative to the start (base) of a binary in memory.
- So if the base is 0x80000000, and the (absolute) Virtual Address was 0x80001000, then the RVA would be 0x1000.
- If the base is 0x80000000, and the VA was 0xC123000f, then the RVA would be 0x4123000f.
- RVA = VA – Base
- Windows uses RVAs extensively in the PE format, unlike ELF which uses just absolute VAs



```
struct _IMAGE_DOS_HEADER {
0x00   WORD e_magic;
0x02   WORD e_cblp;
0x04   WORD e_cp;
0x06   WORD e_crlc;
0x08   WORD e_cparhdr;
0x0a   WORD e_minalloc;
0x0c   WORD e_maxalloc;
0x0e   WORD e_ss;
0x10   WORD e_sp;
0x12   WORD e_csum;
0x14   WORD e_ip;
0x16   WORD e_cs;
0x18   WORD e_lfarlc;
0x1a   WORD e_ovno;
0x1c   WORD e_res[4];
0x24   WORD e_oemid;
0x26   WORD e_oeminfo;
0x28   WORD e_res2[10];
0x3c   DWORD e_lfanew;
};
```

Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura

Image by Ero Carrera

The MS-DOS File Header

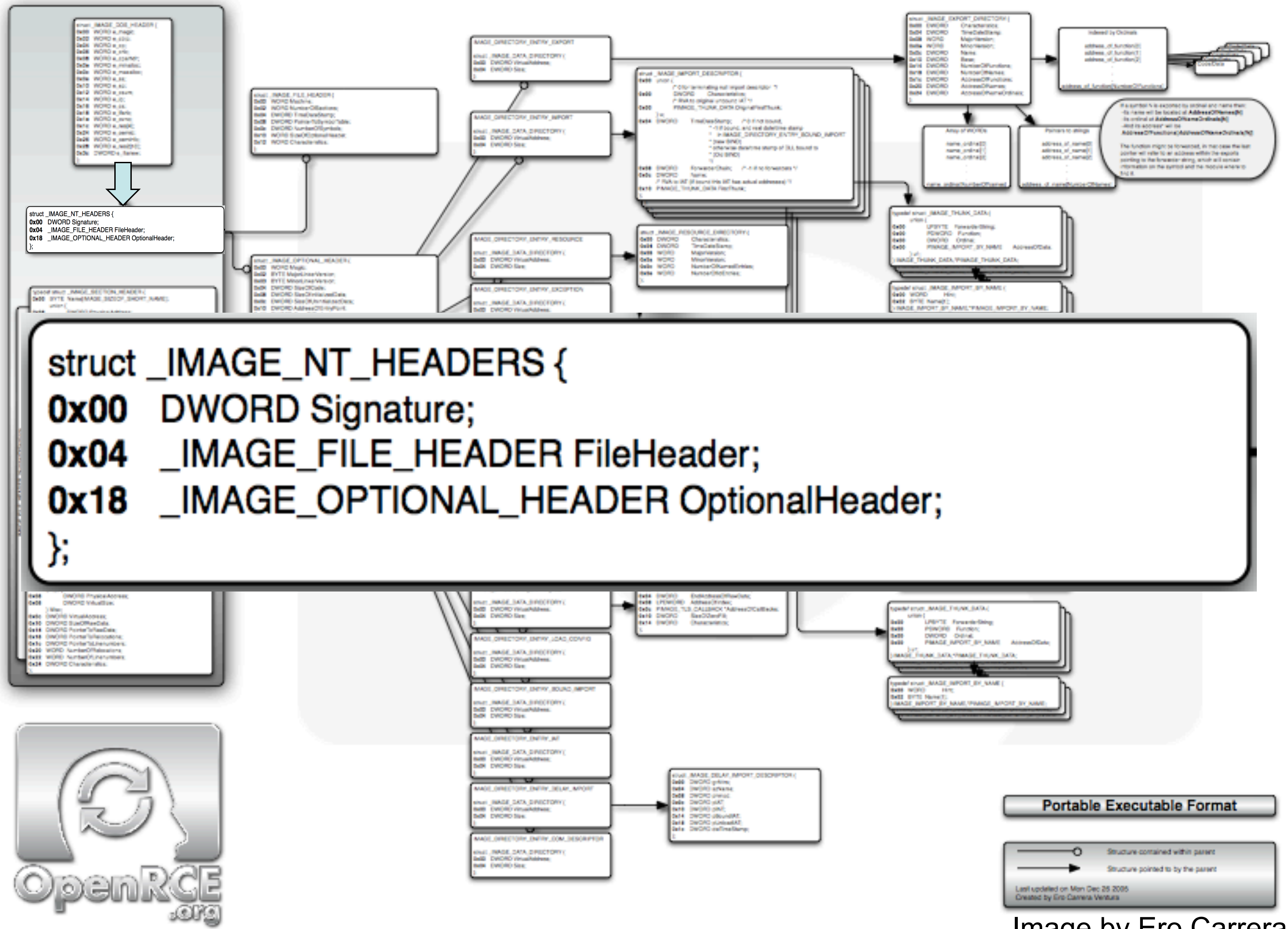
(from winnt.h)

BLUE means the stuff we actually care about

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

The DOS Header

- **e_magic** is set to ASCII ‘MZ’ which is from Mark Zbikowski who developed MS-DOS
- For most Windows programs the DOS header contains a stub DOS program which does nothing but print out “This program cannot be run in DOS mode”
- The main thing we care about is the **e_lfanew** field, which specifies a *file offset* where the PE header can be found (a file pointer if you will)


Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE.org

```
struct _IMAGE_NT_HEADERS {
0x00   DWORD Signature;
0x04   _IMAGE_FILE_HEADER FileHeader;
0x18   _IMAGE_OPTIONAL_HEADER OptionalHeader;
};
```

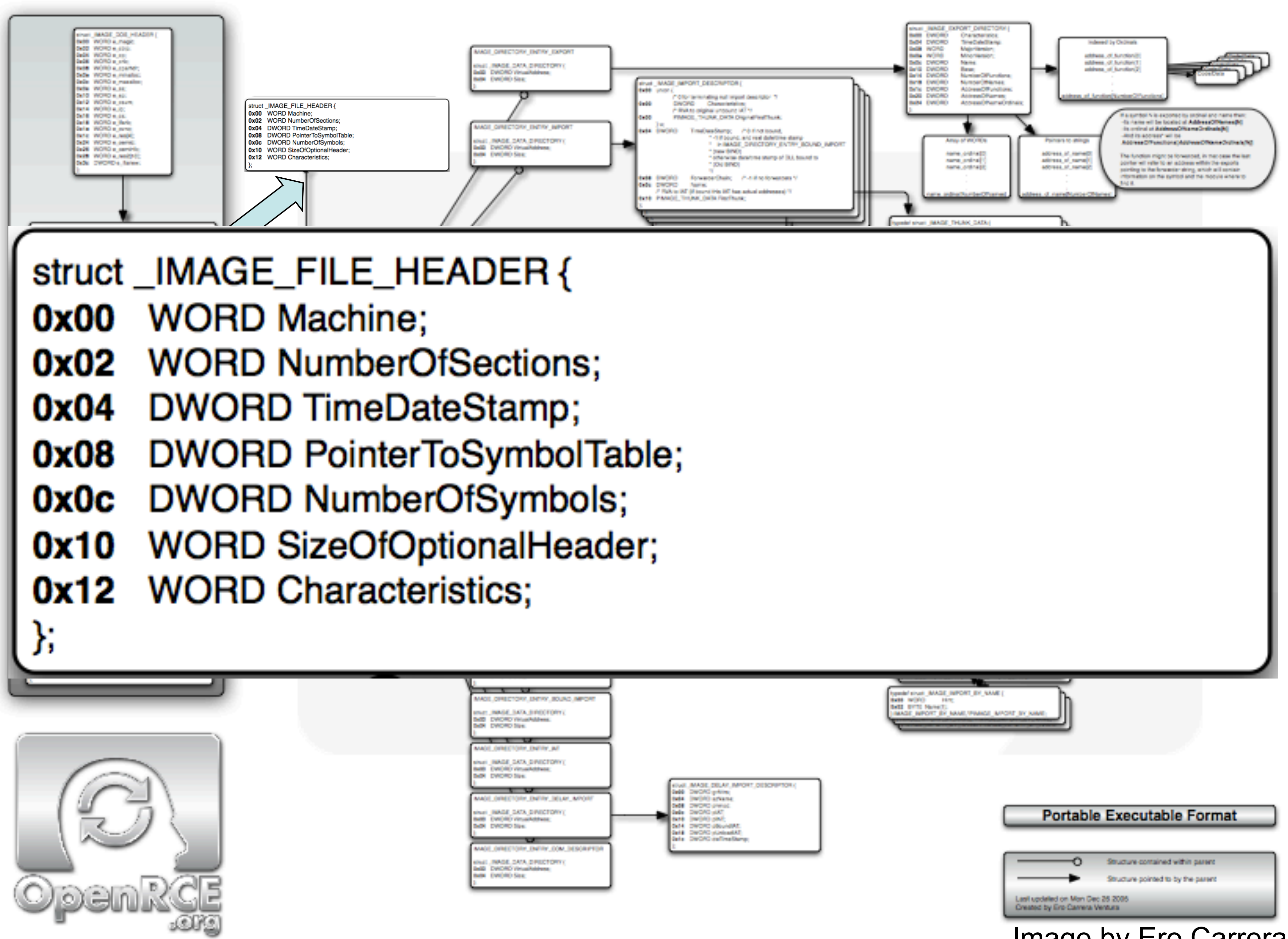
Image by Ero Carrera

NT Header or “PE Header”

(from winnt.h)

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;

} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

- Signature == 0x00004550 aka ASCII string “PE” in little endian order in a DWORD
- Otherwise, just a holder for two other *embedded* (not pointed to) structs


Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura

```
struct _IMAGE_FILE_HEADER {
0x00   WORD Machine;
0x02   WORD NumberOfSections;
0x04   DWORD TimeDateStamp;
0x08   DWORD PointerToSymbolTable;
0x0c   DWORD NumberOfSymbols;
0x10   WORD SizeOfOptionalHeader;
0x12   WORD Characteristics;
};
```

Image by Ero Carrera

File Header

(from winnt.h)

```
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;
```

File Header 2

- The **TimeDateStamp** field is pretty interesting. It's a Unix timestamp (seconds since epoc, where epoc is 00:00:00 UTC on January 1st 1970) and is set at link time.
  - Can be used as a "unique version" for the given file (the version compiled on Jan 1 2010 may or may not be meaningfully different than that compiled on Jan 2 2010)
  - Can be used to know when a file was linked (useful for determining whether an attacker tool is "fresh", or correlating with other forensic evidence, keeping in mind that attackers can manipulate it)

File Header 3

- Oh hay, Hoglund started using the TimeDateStamp as a characteristic for malware attribution (BlackHat Las Vegas 2010, slides not posted yet)
- **NumberOfSections** tells you how many section headers there will be later

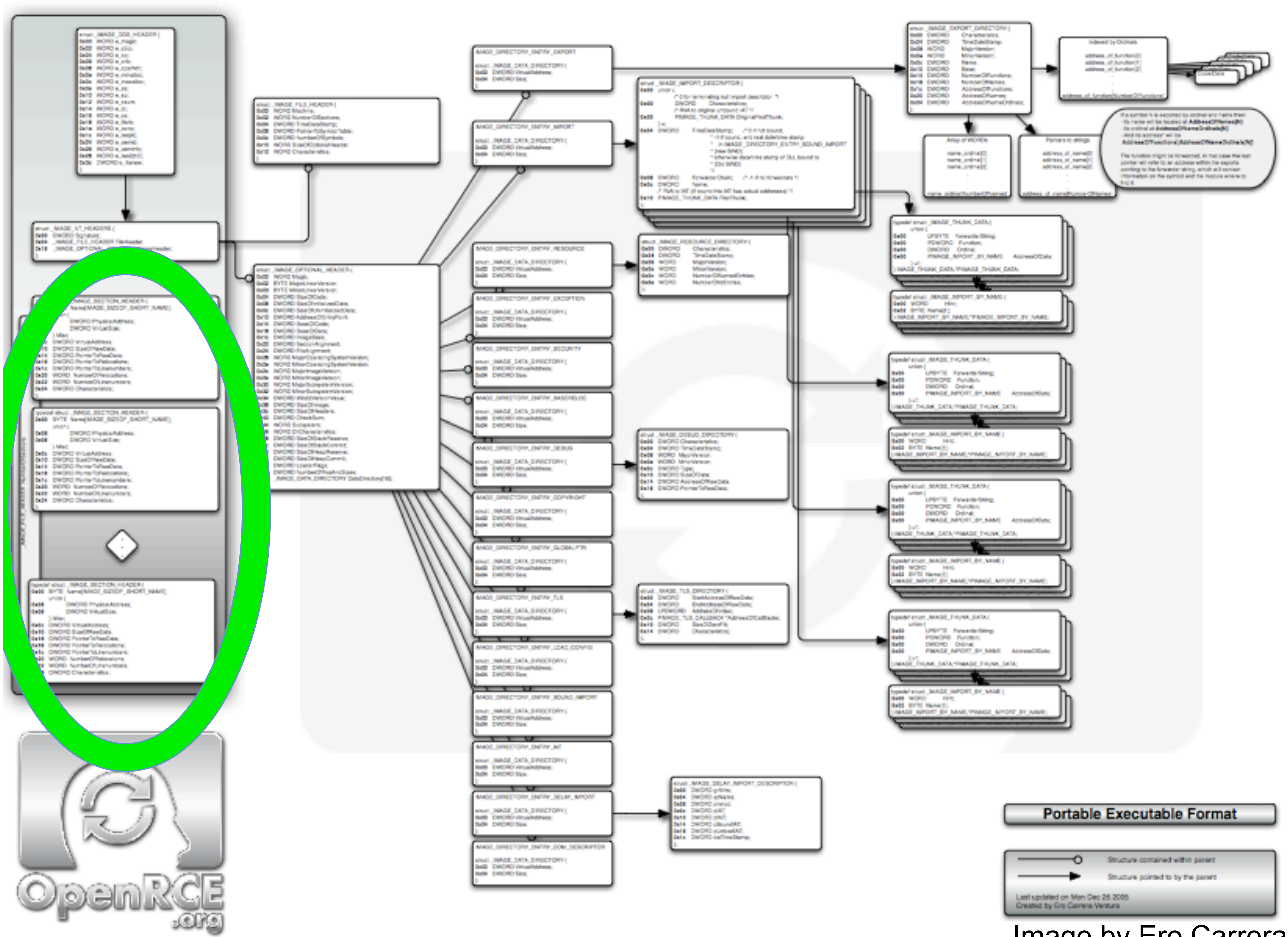

Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE.org

Image by Ero Carrera

File Header 4

(from winnt.h)

- The **Characteristics** field is used to specify things like:

(teeheehee)

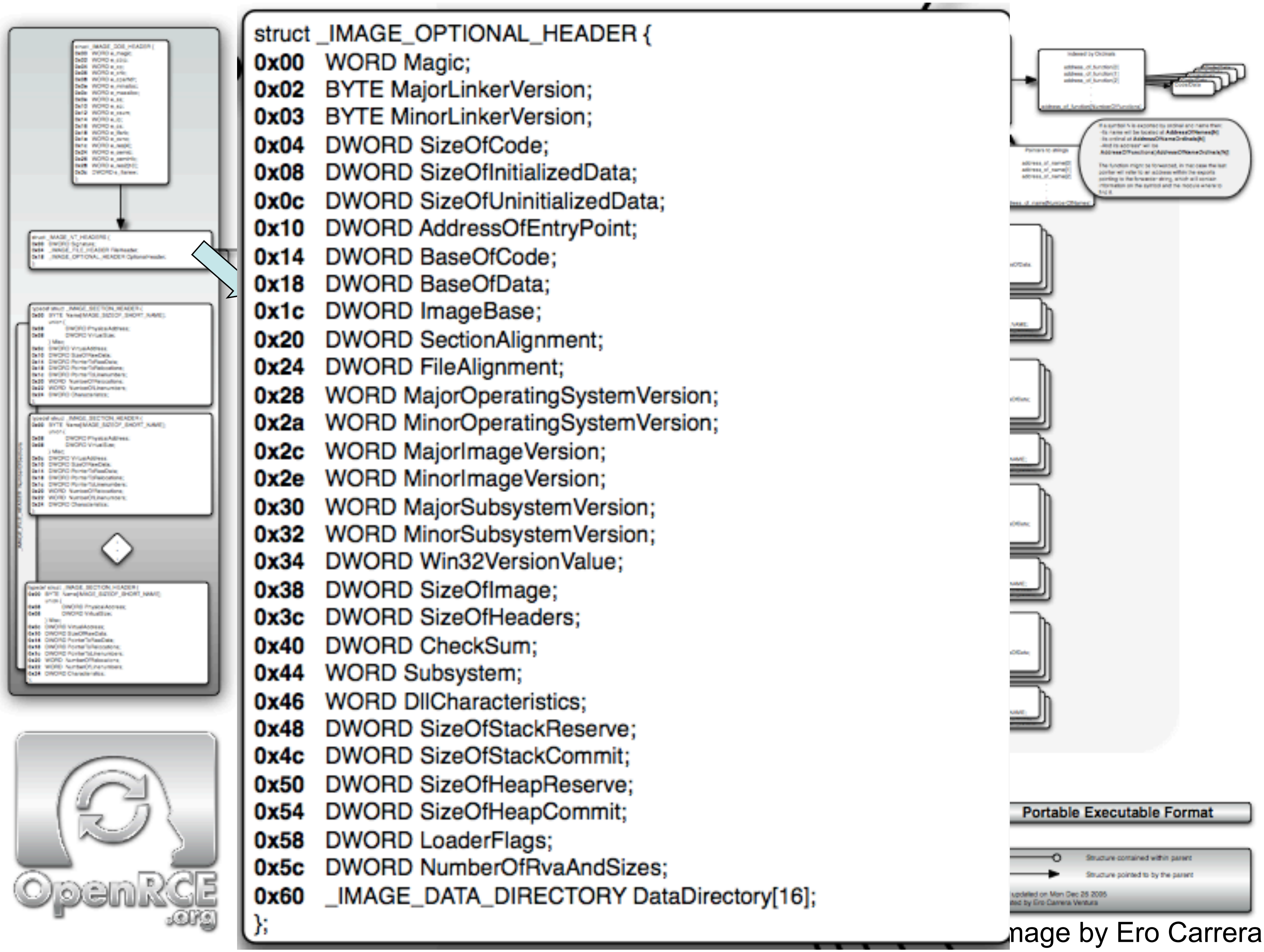
```
#define IMAGE_FILE_EXECUTABLE_IMAGE          0x0002
// File is executable  (i.e. no unresolved externel references).
#define IMAGE_FILE_LINE_NUMS_STRIPPED        0x0004
// Line nunbers stripped from file.
```

(teeheehee)

```
#define IMAGE_FILE_LARGE_ADDRESS_AWARE       0x0020
// App can handle >2gb addresses
#define IMAGE_FILE_32BIT_MACHINE             0x0100
// 32 bit word machine.
#define IMAGE_FILE_SYSTEM                    0x1000
// System File. (Xeno: I don’t see that set on .sys files)
#define IMAGE_FILE_DLL                       0x2000
// File is a DLL.
```

File Header 4

- SizeOfOptionalHeader can *theoretically* be shrunk to exclude “data directory” fields (talked about later) which the linker doesn’t need to include. But I don’t think it ever is in practice.
- PointerToSymbolTable, NumberOfSymbols not used anymore now that debug info is stored in separate file


Portable Executable Format
Structure contained within parent
Structure pointed to by the parent

```
struct _IMAGE_OPTIONAL_HEADER {
0x00   WORD Magic;
0x02   BYTE MajorLinkerVersion;
0x03   BYTE MinorLinkerVersion;
0x04   DWORD SizeOfCode;
0x08   DWORD SizeOfInitializedData;
0x0c   DWORD SizeOfUninitializedData;
0x10   DWORD AddressOfEntryPoint;
0x14   DWORD BaseOfCode;
0x18   DWORD BaseOfData;
0x1c   DWORD ImageBase;
0x20   DWORD SectionAlignment;
0x24   DWORD FileAlignment;
0x28   WORD MajorOperatingSystemVersion;
0x2a   WORD MinorOperatingSystemVersion;
0x2c   WORD MajorImageVersion;
0x2e   WORD MinorImageVersion;
0x30   WORD MajorSubsystemVersion;
0x32   WORD MinorSubsystemVersion;
0x34   DWORD Win32VersionValue;
0x38   DWORD SizeOfImage;
0x3c   DWORD SizeOfHeaders;
0x40   DWORD CheckSum;
0x44   WORD Subsystem;
0x46   WORD DllCharacteristics;
0x48   DWORD SizeOfStackReserve;
0x4c   DWORD SizeOfStackCommit;
0x50   DWORD SizeOfHeapReserve;
0x54   DWORD SizeOfHeapCommit;
0x58   DWORD LoaderFlags;
0x5c   DWORD NumberOfRvaAndSizes;
0x60   _IMAGE_DATA_DIRECTORY DataDirectory[16];
};
```

nage by Ero Carrera

From winnt.h

```
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;
    DWORD   Win32VersionValue;
    DWORD   SizeOfImage;
    DWORD   SizeOfHeaders;
    DWORD   CheckSum;
    WORD    Subsystem;
    WORD    DllCharacteristics;
    DWORD   SizeOfStackReserve;
    DWORD   SizeOfStackCommit;
    DWORD   SizeOfHeapReserve;
    DWORD   SizeOfHeapCommit;
    DWORD   LoaderFlags;
    DWORD   NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

Optional Header

- It’s not at all optional ;)
- **AddressOfEntryPoint** specifies the RVA of where the loader starts executing code once it’s completed loading the binary. Don’t assume it just points to the beginning of the .text section, or even the start of main().
- **SizeOfImage** is the amount of contiguous memory that must be reserved to load the binary into memory

Optional Header 2

- **SectionAlignment** specifies that sections (talked about later) must be aligned on boundaries which are multiples of this value. E.g. if it was 0x1000, then you might expect to see sections starting at 0x1000, 0x2000, 0x5000, etc.
- **FileAlignment** says that data was written to the binary in chunks no smaller than this value. Some common values are 0x200 (512, the size of a HD sector), and 0x80 (not sure what the significance is)

Optional Header 3

- **ImageBase** specifies the preferred virtual memory location where the beginning of the binary should be placed.
- Microsoft recommends developers "rebase" DLL files. That is, picking a non-default memory address which will not conflict with any of the other libraries which will be loaded into the same memory space.
- If the binary cannot be loaded at ImageBase (e.g. because something else is already using that memory), then the loader picks an unused memory range. Then, every location in the binary which was compiled assuming that the binary was loaded at ImageBase must be fixed by adding the difference between the actual ImageBase minus desired ImageBase.
- The list of places which must be fixed is kept in a special "relocations" (.reloc) section.
- This is because MS doesn't support position-independent code

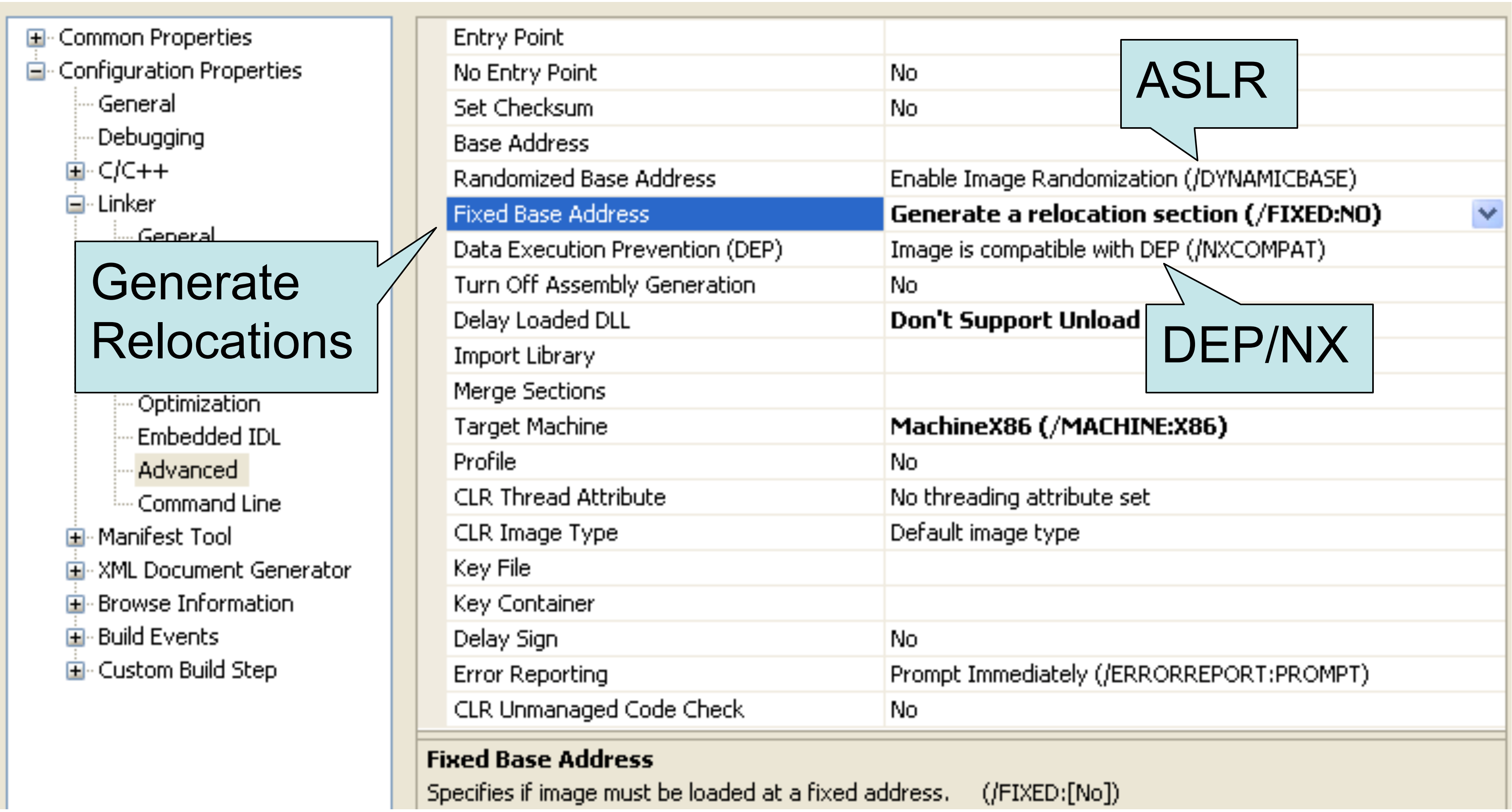
Optional Header 4

- **DLLCharacteristics** specifies some important security options like ASLR and non-executable memory regions for the loader, and the effects are not limited to DLLs.
- `#define IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE 0x0040     // DLL can move.`
- `#define IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY    0x0080     // Code Integrity Image`
- `#define IMAGE_DLLCHARACTERISTICS_NX_COMPAT    0x0100     // Image is NX compatible`
- `#define IMAGE_DLLCHARACTERISTICS_NO_SEH       0x0400     // Image does not use SEH.  No SE handler may reside in this image`
- IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE is set when linked with the /DYNAMICBASE option. This is the flag which tells the OS loader that this binary supports ASLR. Must be used with the /FIXED:NO option for .exe files otherwise they won't get relocation information.
- IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY says to check at load time whether the digitally signed hash of the binary matches.
- IMAGE_DLLCHARACTERISTICS_NX_COMPAT is set with the /NXCOMPAT linker option, and tells the loader that this image is compatible with Data Execution Prevention (DEP) and that non-executable sections should have the NX flag set in memory (we learn about NX in the Intermediate x86 class)
- IMAGE_DLLCHARACTERISTICS_NO_SEH says that this binary never uses structured exception handling, and therefore no default handler should be created (because in the absence of other options that SEH handler is potentially vulnerable to attack.)

Security-Relevant Linker Options

- /DYNAMICBASE – Mark the properties to indicate that this executable will work fine with Address Space Layout Randomization (ASLR)
- /FIXED:NO – This will force the linker to generate relocations information for an executable, so that it is capable of having its base address modified by ASLR (otherwise usually .exe files don't have relocations information, and therefore can't be moved around in memory)
- /NXCOMPAT – Mark the properties to indicate that this executable will work fine with Data Execution Protection (which marks data memory regions such as the stack and heap as non-executable). DEP is just MS's name for utilizing the NX/XD bit to mark memory pages as non-executable (Which we'll talk about more in the Intermediate x86 class)
- /SAFESEH – Safe Structured Exception Handling. Enforces that the only SEH things you can use are ones which are specified in the binary (it will automatically add any ones defined in your code to a list that will be talked about later)

ASLR & DEP/NX



Entry Point	
No Entry Point	No
Set Checksum	No
Base Address	
Randomized Base Address	Enable Image Randomization (/DYNAMICBASE)
Fixed Base Address	**Generate a relocation section (/FIXED:NO)**
Data Execution Prevention (DEP)	Image is compatible with DEP (/NXCOMPAT)
Turn Off Assembly Generation	No
Delay Loaded DLL	**Don't Support Unload**
Import Library	
Merge Sections	
Target Machine	**MachineX86 (/MACHINE:X86)**
Profile	No
CLR Thread Attribute	No threading attribute set
CLR Image Type	Default image type
Key File	
Key Container	
Delay Sign	No
Error Reporting	Prompt Immediately (/ERRORREPORT:PROMPT)
CLR Unmanaged Code Check	No

Fixed Base Address

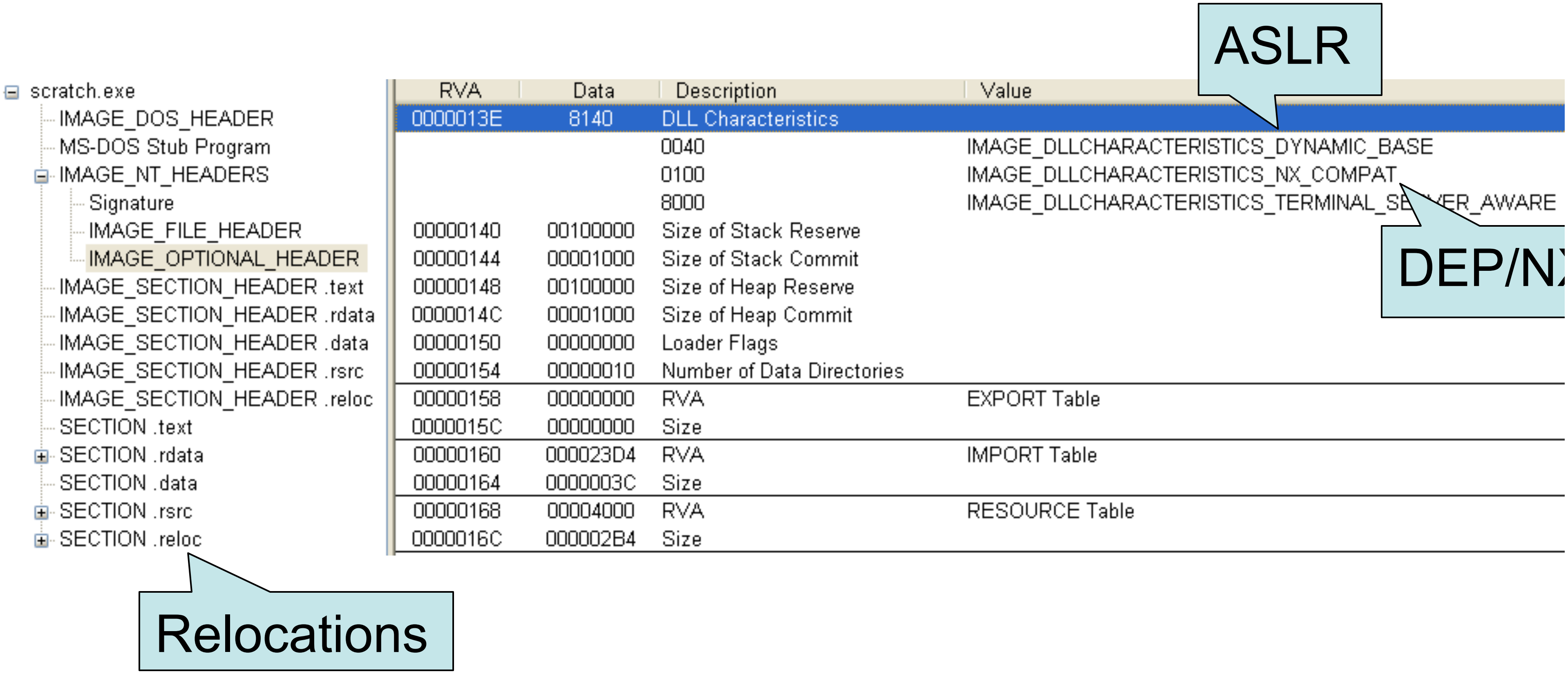
Specifies if image must be loaded at a fixed address. (/FIXED:[No])

ASLR

Generate Relocations

DEP/NX

ASLR & DEP/NX in the Binary



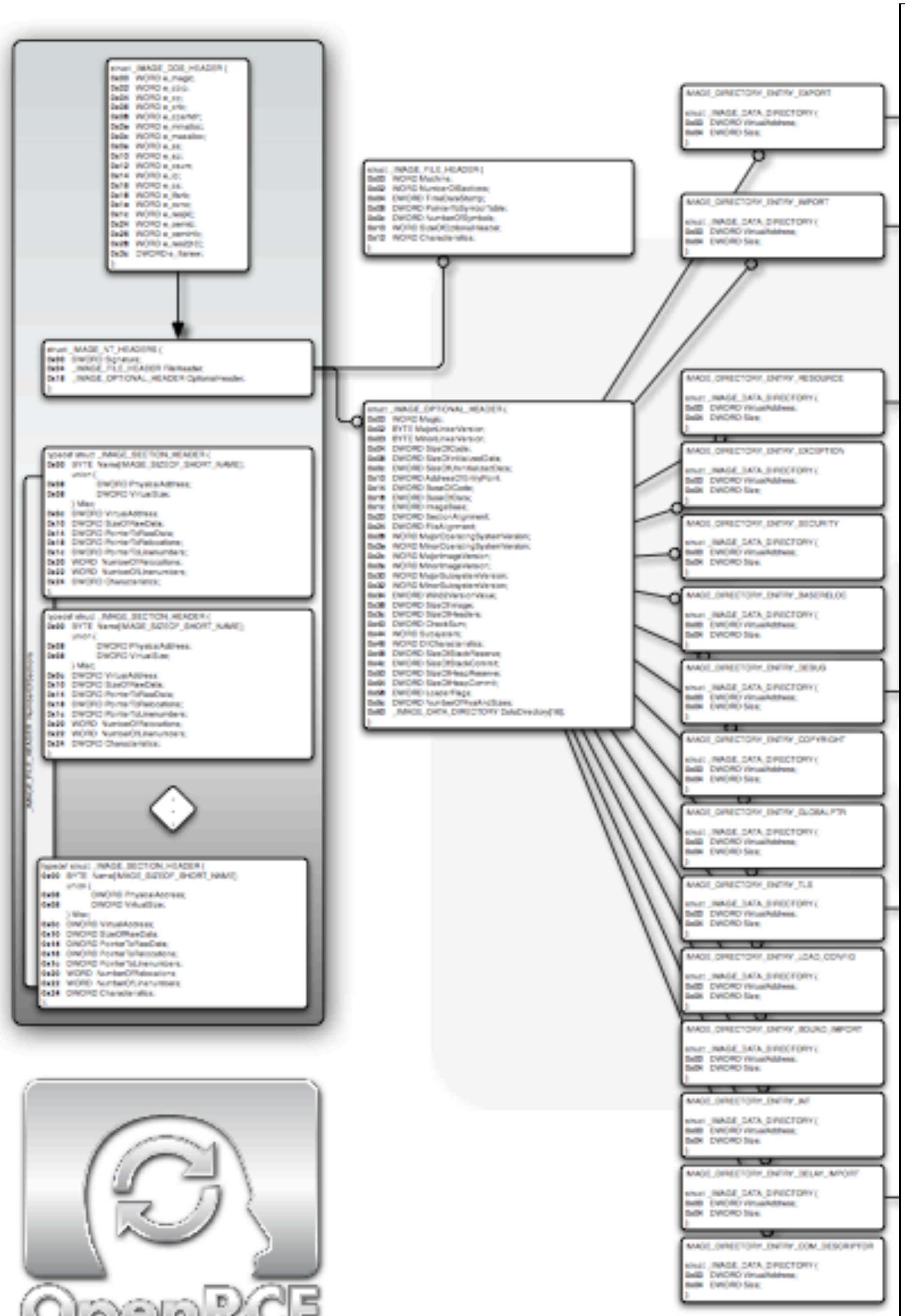
- scratch.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
    - Signature
    - IMAGE_FILE_HEADER
    - IMAGE_OPTIONAL_HEADER
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .rdata
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - IMAGE_SECTION_HEADER .reloc
  - SECTION .text
  - SECTION .rdata
  - SECTION .data
  - SECTION .rsrc
  - SECTION .reloc

RVA	Data	Description	Value
0000013E	8140	DLL Characteristics	
		0040	IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE
		0100	IMAGE_DLLCHARACTERISTICS_NX_COMPAT
		8000	IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE
00000140	00100000	Size of Stack Reserve	
00000144	00001000	Size of Stack Commit	
00000148	00100000	Size of Heap Reserve	
0000014C	00001000	Size of Heap Commit	
00000150	00000000	Loader Flags	
00000154	00000010	Number of Data Directories	
00000158	00000000	RVA	EXPORT Table
0000015C	00000000	Size	
00000160	000023D4	RVA	IMPORT Table
00000164	0000003C	Size	
00000168	00004000	RVA	RESOURCE Table
0000016C	000002B4	Size	

ASLR

DEP/N

Relocations



DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]

`#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES 16`
(from winnt.h)

Therefore, while FileHeader.SizeOfOptionalHeader could technically change, in practice it's fixed

Optional Header 3

- The type of **DataDirectory**[16] is IMAGE_DATA_DIRECTORY

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

- VirtualAddress is a RVA pointer to some other structure of the given Size

Optional Header 4

(from winnt.h)

- There is a predefined possible structure for each index in DataDirectory[]

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

- We will return to each entry in the DataDirectory[] later.
- Note that while the array is 16 elements, only 15 (0-14) are defined.

Pop quiz, hot shot. Which fields do we even care about, and why?

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

Sections

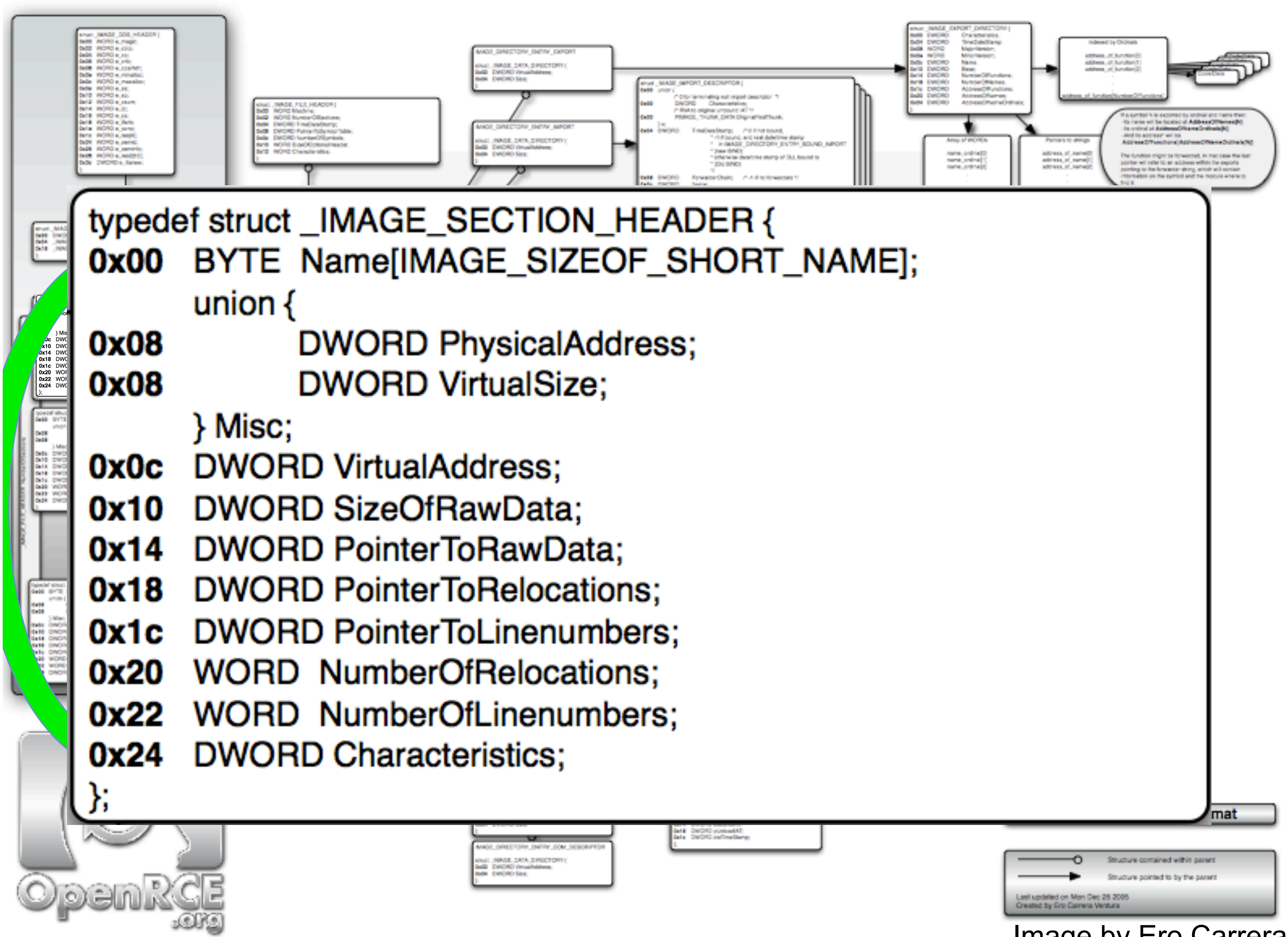
- Sections group portions of code or data (Von Neumann sez: “What’s the difference?! :P”) which have similar purpose, or should have similar memory permissions (remember the linking merge option? That would be for merging sections with "similar memory permissions")

Sections 2

- Common section names:
- .text = Code which should never be paged out of memory to disk
- .data = read/write data (globals)
- .rdata = read-only data (strings)
- .bss = (Block Started by Symbol or Block Storage Segment or Block Storage Start depending on who you ask (the CMU architecture book says the last one))
- MS spec says of .bss “Uninitialized data (free format)” which is the same as for ELF.
- In practice, the .bss seems to be merged into the .data section by the linker for the binaries I’ve looked at
- .idata = import address table (talked about later). In practice, seems to get merged with .text or .rdata

Sections 3

- PAGE* = Code/data which it’s fine to page out to disk if you’re running low on memory (not in the spec, seems to be used primarily for kernel drivers)
- .reloc = Relocation information for where to modify hardcoded addresses which assume that the code was loaded at its preferred base address in memory
- .rsrc = Resources. Lots of possible stuff from icons to other embedded binaries. The section has structures organizing it sort of like a filesystem.


OpenRCE.org
mat
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura

```
typedef struct _IMAGE_SECTION_HEADER {
0x00   BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
       union {
0x08           DWORD PhysicalAddress;
0x08           DWORD VirtualSize;
       } Misc;
0x0c   DWORD VirtualAddress;
0x10   DWORD SizeOfRawData;
0x14   DWORD PointerToRawData;
0x18   DWORD PointerToRelocations;
0x1c   DWORD PointerToLinenumbers;
0x20   WORD  NumberOfRelocations;
0x22   WORD  NumberOfLinenumbers;
0x24   DWORD Characteristics;
};
```

Image by Ero Carrera

Section Header

(from winnt.h)

```
#define IMAGE_SIZEOF_SHORT_NAME      8

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

Refresher: C Unions

```
union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;
    } Misc;
```

- Used to store multiple different interpretations of the same data in the same location.
- Accessed as if the union were a struct. So if you have

  `IMAGE_SECTION_HEADER sectHdr;`

  You don't access `sectHdr.VirtualSize`, you access `sectHdr.Misc.VirtualSize`
- We will only ever consider it as the VirtualSize field.

Section Header 2

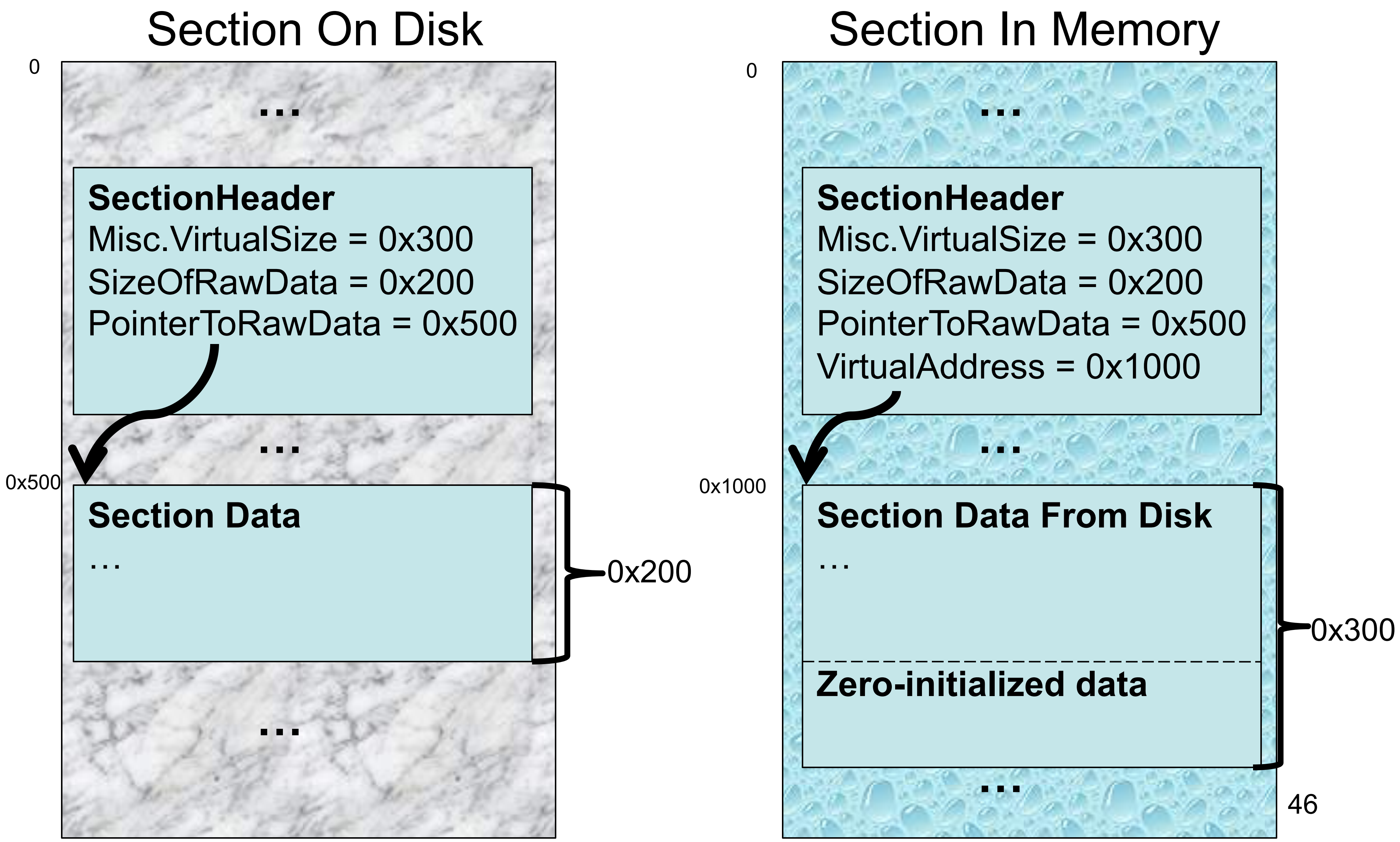
- **Name[8]** is a byte array of ASCII characters. It is **<u>NOT</u>** guaranteed to be null-terminated. So if you're trying to parse a PE file yourself you need to be aware of that.
- **VirtualAddress** is the RVA of the section relative to OptionalHeader.ImageBase
- **PointerToRawData** is a relative offset from the beginning of the file which says where the actual section data is stored.

Section Header 3

- There is an interesting interplay between **Misc.VirtualSize** and **SizeOfRawData**. Sometimes one is larger, and other times the opposite.
- Why would VirtualSize be greater than SizeOfRawData? This indicates that the section is allocating more memory space than it has data written to disk.
- Think about the .bss portion of the .rdata section. It just needs a bunch of space for variables. The variables are uninitialized, which is why they don't have to be in the file. Therefore the loader can just give a chunk of memory to store variables in, by just allocating VirtualSize worth of data. Thus you get a smaller binary.

VirtualSize > SizeOfRawData

(on your own slide, draw the correspondence between the 0x200 in the first picture and the 0x300 in the second)



Section On Disk

SectionHeader
Misc.VirtualSize = 0x300
SizeOfRawData = 0x200
PointerToRawData = 0x500

Section Data
…

Section In Memory

SectionHeader
Misc.VirtualSize = 0x300
SizeOfRawData = 0x200
PointerToRawData = 0x500
VirtualAddress = 0x1000

Section Data From Disk
…

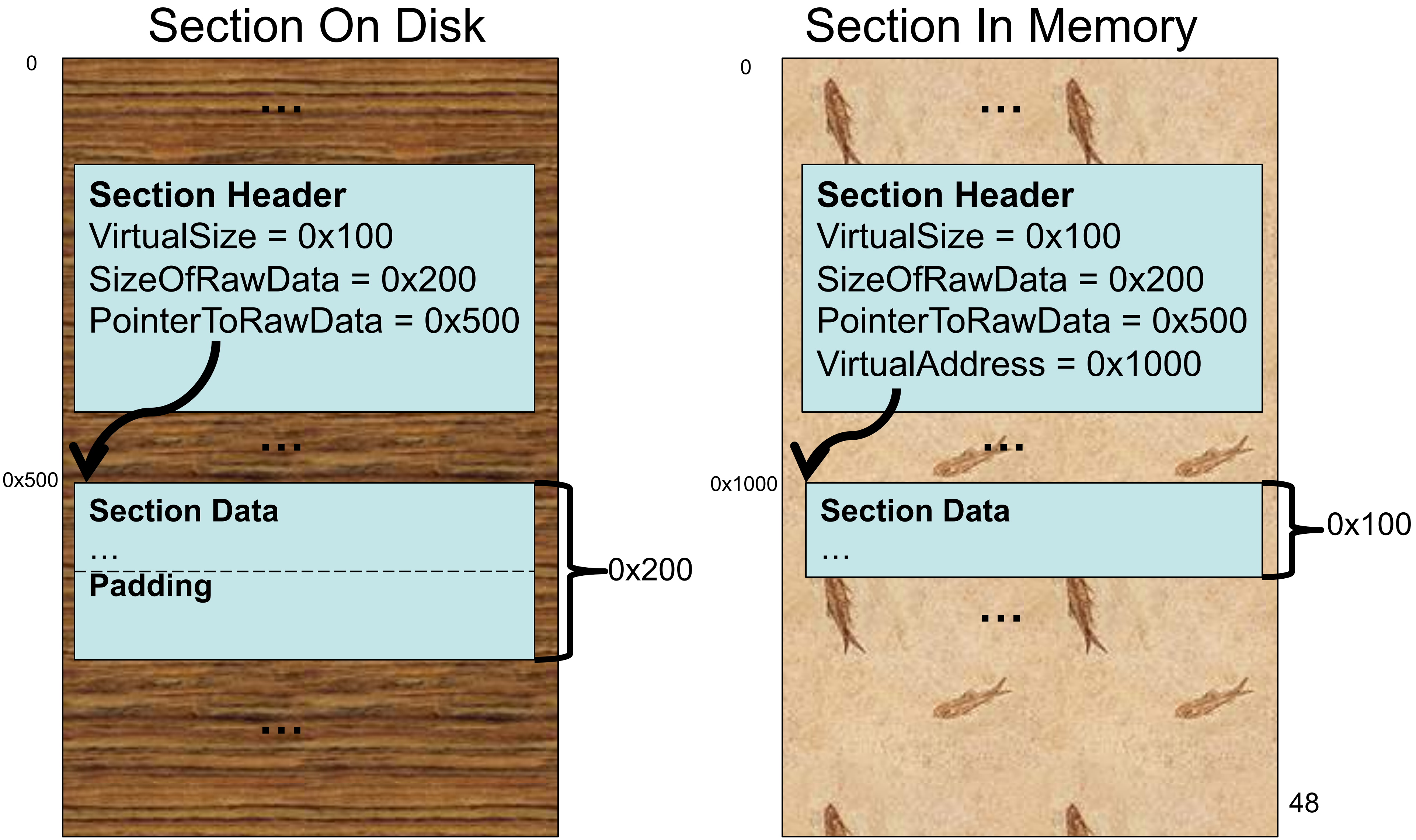
Zero-initialized data

Section Header 4

- Why would SizeOfRawData be greater than VirtualSize?
- Remember that PE has the notion of file alignment.(OptionalHeader.FileAlignment)Therefore, if you had a FileAlignment of 0x200, but you only had 0x100 bytes of data, the linker would have had to write 0x100 bytes of data followed by 0x100 bytes of padding.
- By having the VirtualSize < SizeOfRawData, the loader can say "ok, well I see I really only need to allocate 0x100 bytes of memory and read 0x100 bytes of data from disk."

VirtualSize < SizeOfRawData

(on your own slide, draw the correspondence between the 0x200 in the first picture and the 0x100 in the second))



Section On Disk

0

…

Section Header
VirtualSize = 0x100
SizeOfRawData = 0x200
PointerToRawData = 0x500

…

0x500

Section Data
…
Padding

0x200

…

Section In Memory

0

…

Section Header
VirtualSize = 0x100
SizeOfRawData = 0x200
PointerToRawData = 0x500
VirtualAddress = 0x1000

…

0x1000

Section Data
…

0x100

…

Section Header 5

(from winnt.h)

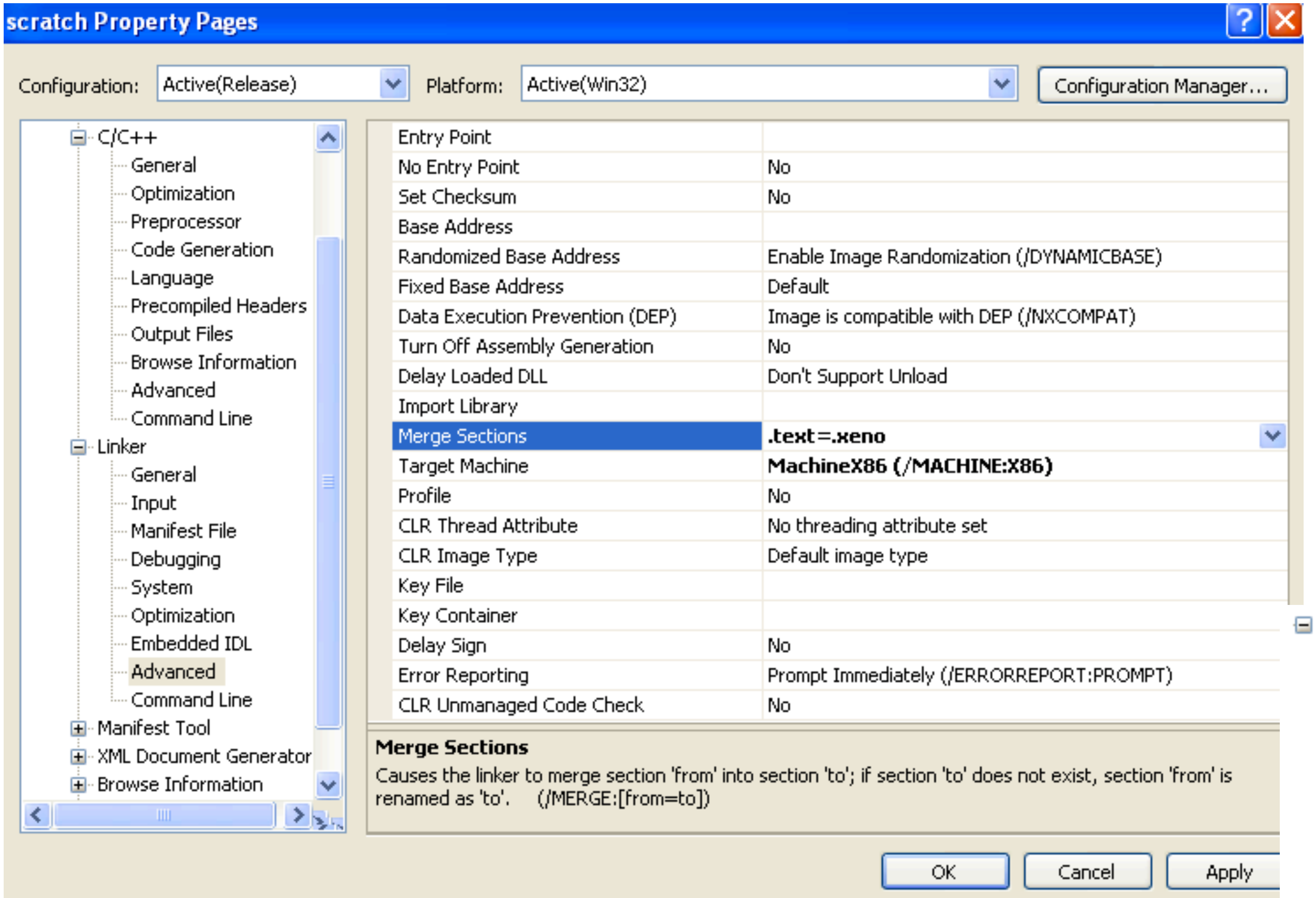
- **Characteristics** tell you something about the section. Examples:

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

Section Header

- PointerToRelocations, PointerToLinenumbers, NumberOfRelocations, NumberOfLinenumbers aren't used anymore

Renaming Sections



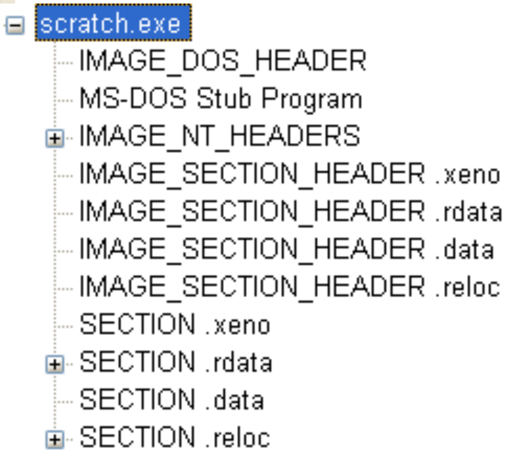
Configuration: Active(Release)

Platform: Active(Win32)

Entry Point	
No Entry Point	No
Set Checksum	No
Base Address	
Randomized Base Address	Enable Image Randomization (/DYNAMICBASE)
Fixed Base Address	Default
Data Execution Prevention (DEP)	Image is compatible with DEP (/NXCOMPAT)
Turn Off Assembly Generation	No
Delay Loaded DLL	Don't Support Unload
Import Library	
Merge Sections	**.text=.xeno**
Target Machine	**MachineX86 (/MACHINE:X86)**
Profile	No
CLR Thread Attribute	No threading attribute set
CLR Image Type	Default image type
Key File	
Key Container	
Delay Sign	No
Error Reporting	Prompt Immediately (/ERRORREPORT:PROMPT)
CLR Unmanaged Code Check	No

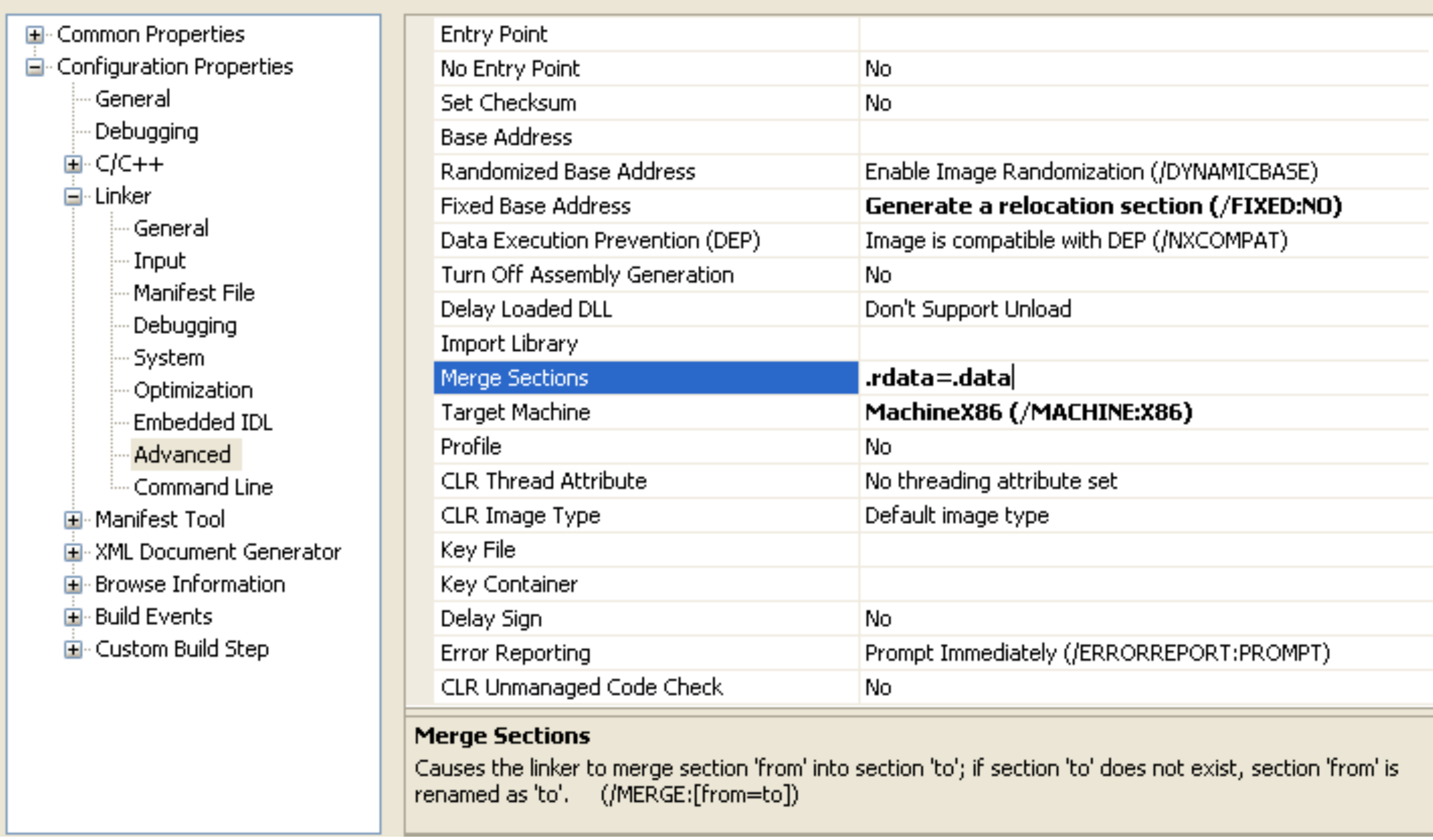
Merge Sections

Causes the linker to merge section 'from' into section 'to'; if section 'to' does not exist, section 'from' is renamed as 'to'. (/MERGE:[from=to])



- scratch.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .xeno
  - IMAGE_SECTION_HEADER .rdata
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .reloc
  - SECTION .xeno
  - SECTION .rdata
  - SECTION .data
  - SECTION .reloc

Merge Sections



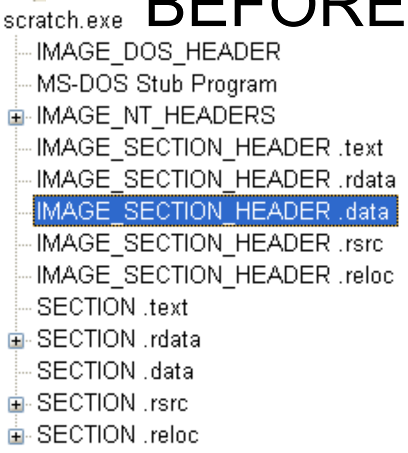
- Common Properties
- Configuration Properties
  - General
  - Debugging
  - C/C++
  - Linker
    - General
    - Input
    - Manifest File
    - Debugging
    - System
    - Optimization
    - Embedded IDL
    - Advanced
    - Command Line
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step

Entry Point	
No Entry Point	No
Set Checksum	No
Base Address	
Randomized Base Address	Enable Image Randomization (/DYNAMICBASE)
Fixed Base Address	**Generate a relocation section (/FIXED:NO)**
Data Execution Prevention (DEP)	Image is compatible with DEP (/NXCOMPAT)
Turn Off Assembly Generation	No
Delay Loaded DLL	Don't Support Unload
Import Library	
Merge Sections	**.rdata=.data**
Target Machine	**MachineX86 (/MACHINE:X86)**
Profile	No
CLR Thread Attribute	No threading attribute set
CLR Image Type	Default image type
Key File	
Key Container	
Delay Sign	No
Error Reporting	Prompt Immediately (/ERRORREPORT:PROMPT)
CLR Unmanaged Code Check	No

Merge Sections

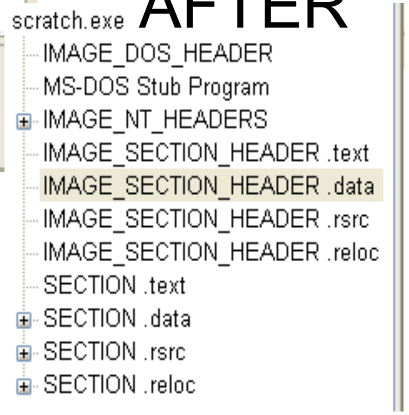
Causes the linker to merge section 'from' into section 'to'; if section 'to' does not exist, section 'from' is renamed as 'to'. (/MERGE:[from=to])

BEFORE



- scratch.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .rdata
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - IMAGE_SECTION_HEADER .reloc
  - SECTION .text
  - SECTION .rdata
  - SECTION .data
  - SECTION .rsrc
  - SECTION .reloc

AFTER



- scratch.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - IMAGE_SECTION_HEADER .reloc
  - SECTION .text
  - SECTION .data
  - SECTION .rsrc
  - SECTION .reloc

```
scratch.c
Linking...
LINK : warning LNK4254: section '.rdata' (40000040) merged into '.data' (C0000040) with different attributes
```

Which fields do we even care about, and why?

```
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;
```

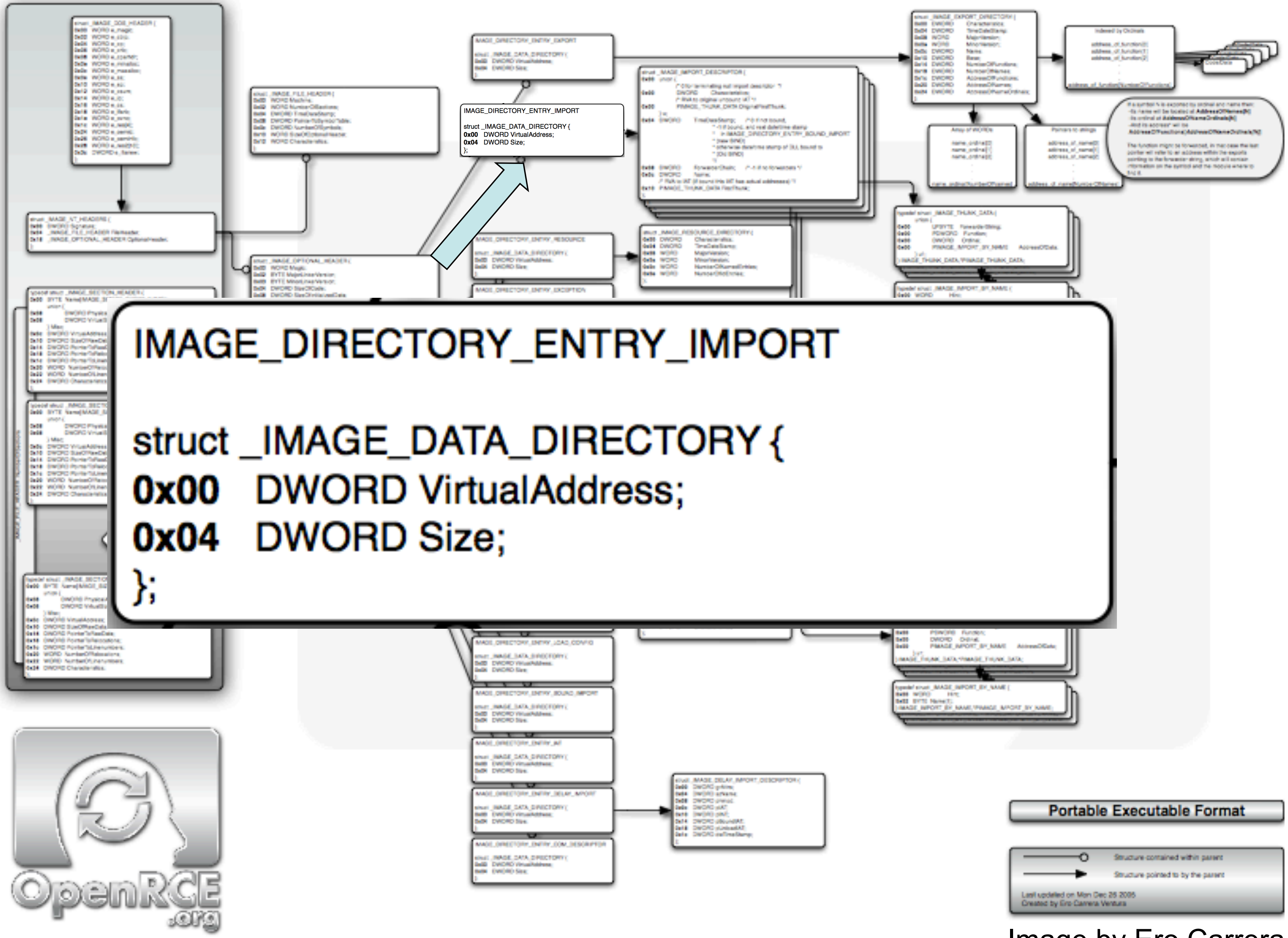
Static Linking vs Dynamic Linking

- With static linking, you literally just include a copy of every helper function you use inside the executable you're generating.
- Dynamic linking is when you resolve pointers to functions inside libraries at runtime.
- Needless to say, a statically linked executable is bloated compared to a dynamically linked one. But on the other hand, it's standalone, without outside dependencies. But on the other other hand, patches or fixes to libraries are not applied to the statically linked binary until it's re-linked, so it can potentially have vulnerable code long after a library vulnerability is patched.
- Going to learn a bunch about how dynamic linking works, in service to learning a bit about how it is abused.

Calling Imported Functions

- As a programmer, this is transparent to you, but what sort of assembly does the compiler actually generate when you call an imported function like printf()?
- We can use the handy-dandy HelloWorld.c to find out quickly.

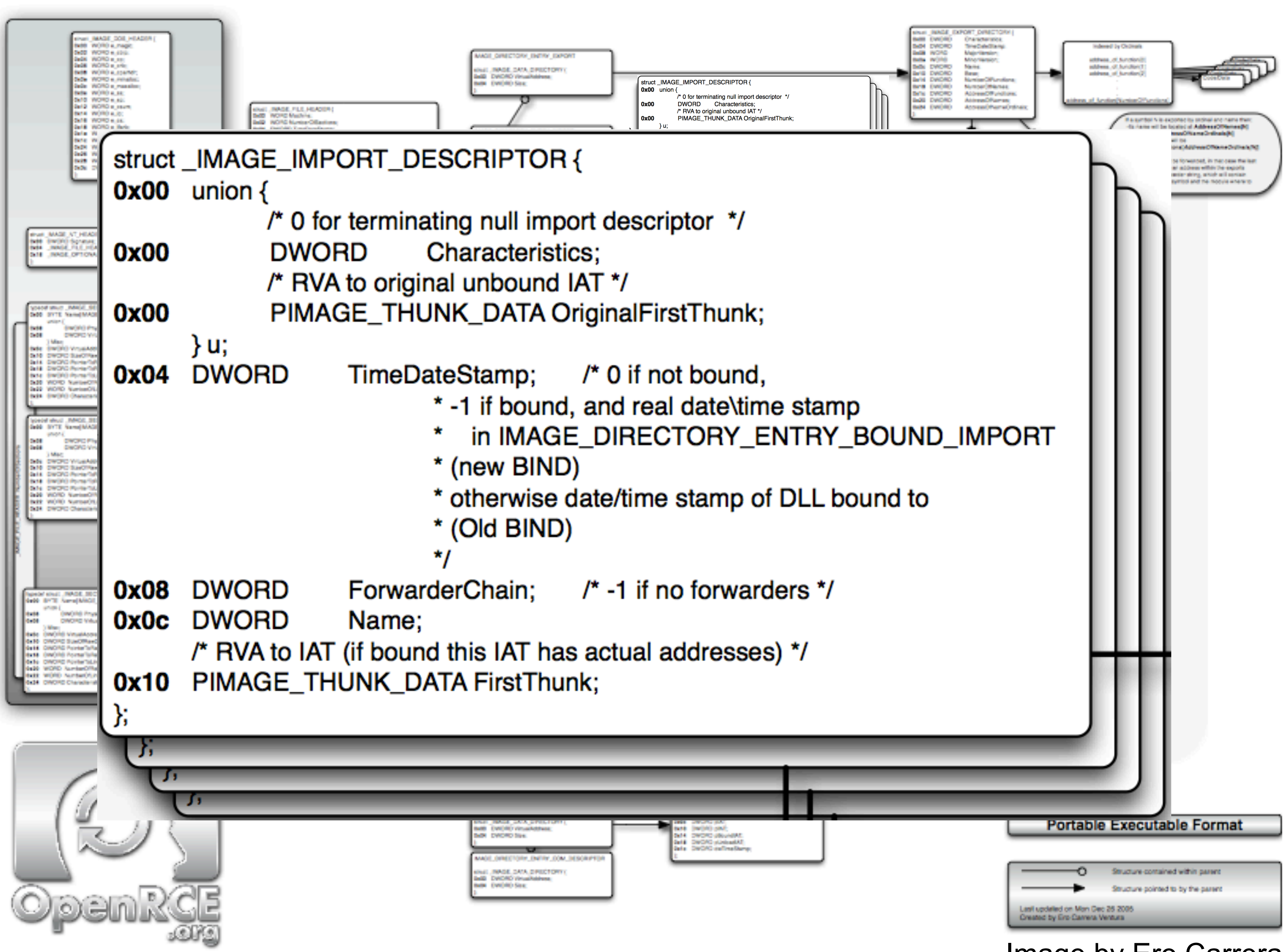
```
printf("Hello World!\n");
004113BE 8B F4                    mov         esi,esp
004113C0 68 3C 57 41 00           push        41573Ch
004113C5 FF 15 BC 82 41 00        call        dword ptr ds:[004182BCh]

(Note to self, show imports in PEView too)
```


IMAGE_DIRECTORY_ENTRY_IMPORT
Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE.org

IMAGE_DIRECTORY_ENTRY_IMPORT

```
struct _IMAGE_DATA_DIRECTORY {
0x00   DWORD VirtualAddress;
0x04   DWORD Size;
};
```

Image by Ero Carrera



```
struct _IMAGE_IMPORT_DESCRIPTOR {
0x00  union {
                /* 0 for terminating null import descriptor  */
0x00            DWORD       Characteristics;
                /* RVA to original unbound IAT */
0x00            PIMAGE_THUNK_DATA OriginalFirstThunk;
      } u;
0x04  DWORD        TimeDateStamp;      /* 0 if not bound,
                                   * -1 if bound, and real date\time stamp
                                   *    in IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT
                                   * (new BIND)
                                   * otherwise date/time stamp of DLL bound to
                                   * (Old BIND)
                                   */
0x08  DWORD        ForwarderChain;     /* -1 if no forwarders */
0x0c  DWORD        Name;
      /* RVA to IAT (if bound this IAT has actual addresses) */
0x10  PIMAGE_THUNK_DATA FirstThunk;
};
```

Image by Ero Carrera

Import Descriptor

(from winnt.h)

I think they meant “INT”

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;            // 0 for terminating null import descriptor
        DWORD   OriginalFirstThunk;         // RVA to original unbound IAT (PIMAGE_THUNK_DATA)
    };
    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;
```

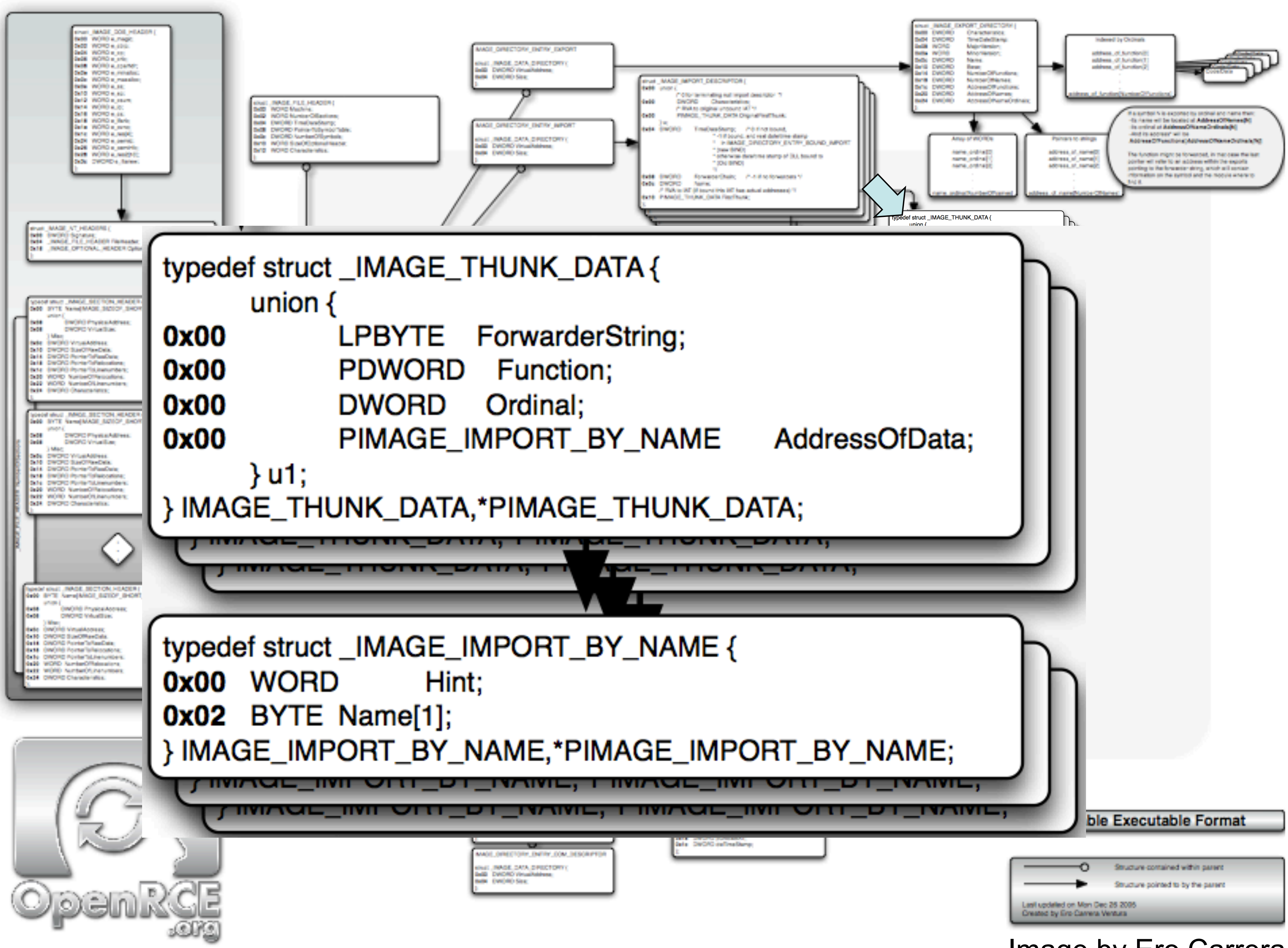
- While the things in blue are the fields filled in for the most common case, we will actually have to understand everything for this structure, because you could run into all the variations.

Import Descriptor 2

- **OriginalFirstThunk** ("is badly named" according to Matt Pietrek) is the RVA of the Import Name Table (INT). It' s so named because the INT is an array of IMAGE_THUNK_DATA structs. So this field of the import descriptor is trying to say that it' s pointing at the first entry in that array.

Import Descriptor 3

- **FirstThunk** like OriginalFirstThunk except that instead of being an RVA which points into the INT, it's pointing into the Import Address Table (IAT). The IAT is also an array of IMAGE_THUNK_DATA structures (they're heavily overloaded as we'll see).
- **Name** is just the RVA which will point at the specific name of the module which imports are taken from (e.g. hal.dll, ntdll.dll, etc)


ble Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE.org

```
typedef struct _IMAGE_THUNK_DATA {
        union {
0x00            LPBYTE    ForwarderString;
0x00            PDWORD    Function;
0x00            DWORD     Ordinal;
0x00            PIMAGE_IMPORT_BY_NAME     AddressOfData;
        } u1;
} IMAGE_THUNK_DATA,*PIMAGE_THUNK_DATA;
```

```
typedef struct _IMAGE_IMPORT_BY_NAME {
0x00   WORD        Hint;
0x02   BYTE  Name[1];
} IMAGE_IMPORT_BY_NAME,*PIMAGE_IMPORT_BY_NAME;
```

Image by Ero Carrera

IMAGE_THUNK_DATA

(from winnt.h)

```
typedef struct _IMAGE_THUNK_DATA32 {
    union {
        DWORD ForwarderString;      // PBYTE
        DWORD Function;             // PDWORD
        DWORD Ordinal;
        DWORD AddressOfData;        // PIMAGE_IMPORT_BY_NAME
    } u1;
} IMAGE_THUNK_DATA32;
```

- We just learned that both the INT (pointed to by OriginalFirstThunk) and the IAT (pointed to by FirstThunk) point at arrays of IMAGE_THUNK_DATA32s.
- The INT and IAT IMAGE_THUNK_DATA32 structures are all interpreted as pointing at IMAGE_IMPORT_BY_NAME structures ***to begin with***. That is they are **u1.AddressOfData**. This is actually the RVA of an IMAGE_IMPORT_BY_NAME structure.

IMAGE_IMPORT_BY_NAME

(from winnt.h)

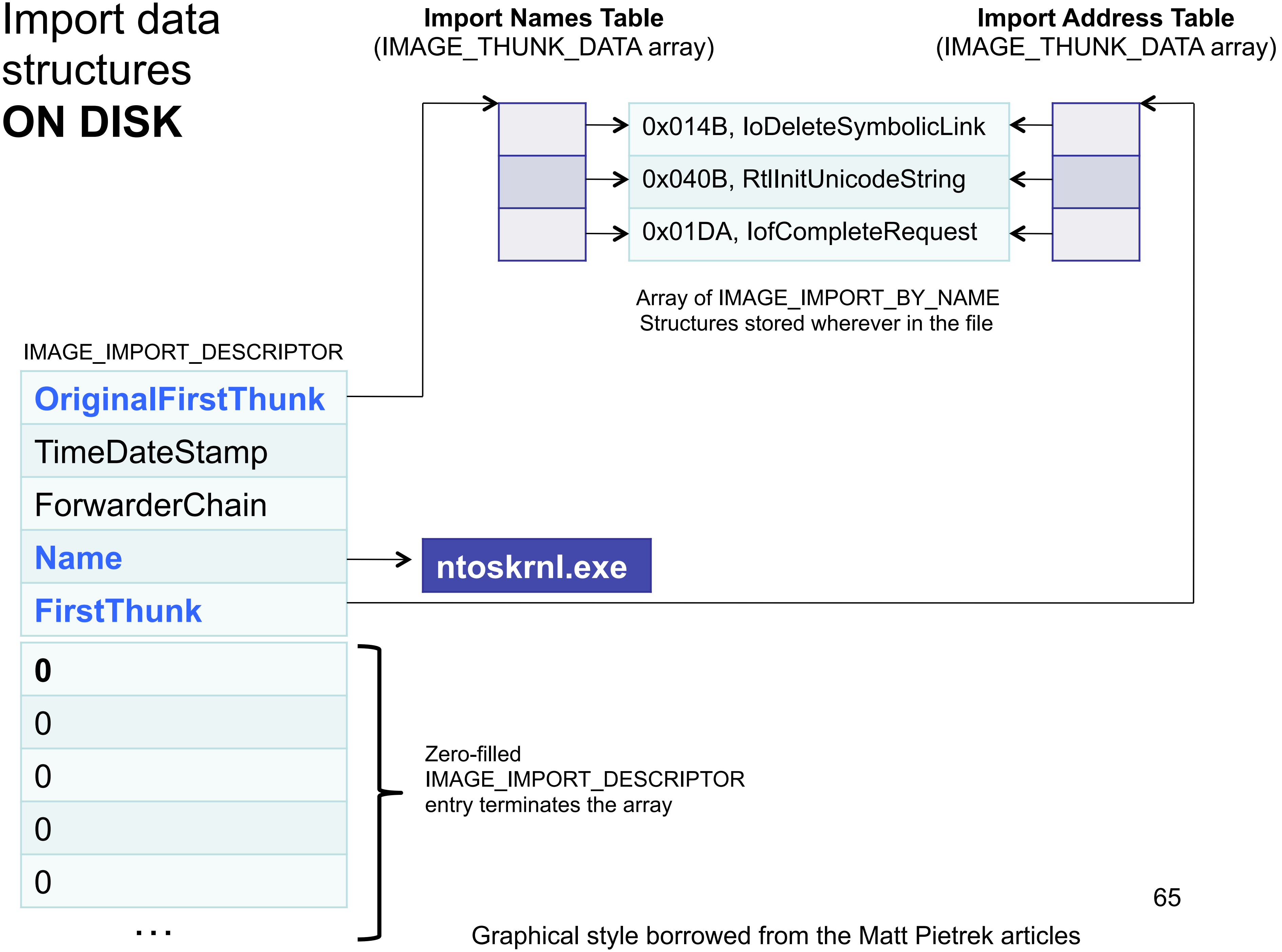
```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;
    BYTE    Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

- **Hint** specifies a possible “ordinal” of an imported function. Talked about later, when we talk about exports, but basically it’s just a way to look up the function by an index rather than a name.
- **Name** on the other hand is to look up the function by name. It’s not one byte long, it’s a null terminated ASCII string which follows the hint. But usually it's just null in our examples.

On the impersistence of being: INT vs IAT

- The **INT** IMAGE_THUNK_DATA structures are always interpreted as pointing at IMAGE_IMPORT_BY_NAME structures, that is they are **u1.AddressOfData**, the RVA of an IMAGE_IMPORT_BY_NAME.
- The **IAT** IMAGE_THUNK_DATA structures ***start out*** are all interpreted as the **u1.AddressOfData**, but once the OS loader resolves each import, it overwrites the IMAGE_THUNK_DATA structure with the actual virtual address of the start of the function. Therefore it is ***subsequently*** interpreted as **u1.Function**.

Import data structures
ON DISK



Import Names Table
(IMAGE_THUNK_DATA array)

Import Address Table
(IMAGE_THUNK_DATA array)

0x014B, IoDeleteSymbolicLink
0x040B, RtlInitUnicodeString
0x01DA, IofCompleteRequest

Array of IMAGE_IMPORT_BY_NAME
Structures stored wherever in the file

IMAGE_IMPORT_DESCRIPTOR

OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk

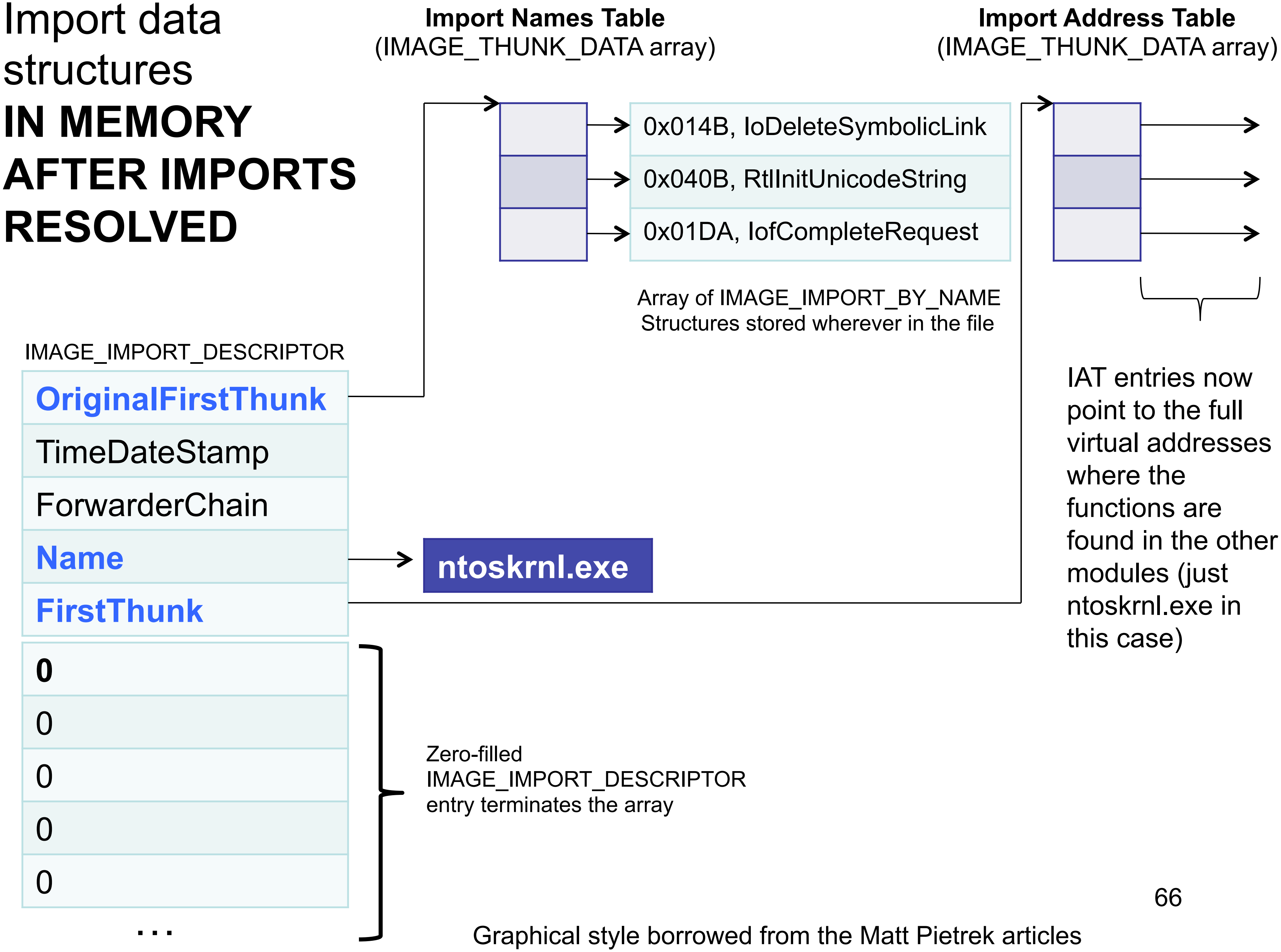
ntoskrnl.exe

0
0
0
0
0
. . .

Zero-filled
IMAGE_IMPORT_DESCRIPTOR
entry terminates the array

Graphical style borrowed from the Matt Pietrek articles

Import data structures
IN MEMORY AFTER IMPORTS RESOLVED



Import Names Table
(IMAGE_THUNK_DATA array)

0x014B, IoDeleteSymbolicLink
0x040B, RtlInitUnicodeString
0x01DA, IofCompleteRequest

Array of IMAGE_IMPORT_BY_NAME
Structures stored wherever in the file

Import Address Table
(IMAGE_THUNK_DATA array)

IAT entries now point to the full virtual addresses where the functions are found in the other modules (just ntoskrnl.exe in this case)

IMAGE_IMPORT_DESCRIPTOR

- **OriginalFirstThunk**
- TimeDateStamp
- ForwarderChain
- **Name**
- **FirstThunk**

ntoskrnl.exe

- **0**
- 0
- 0
- 0
- 0

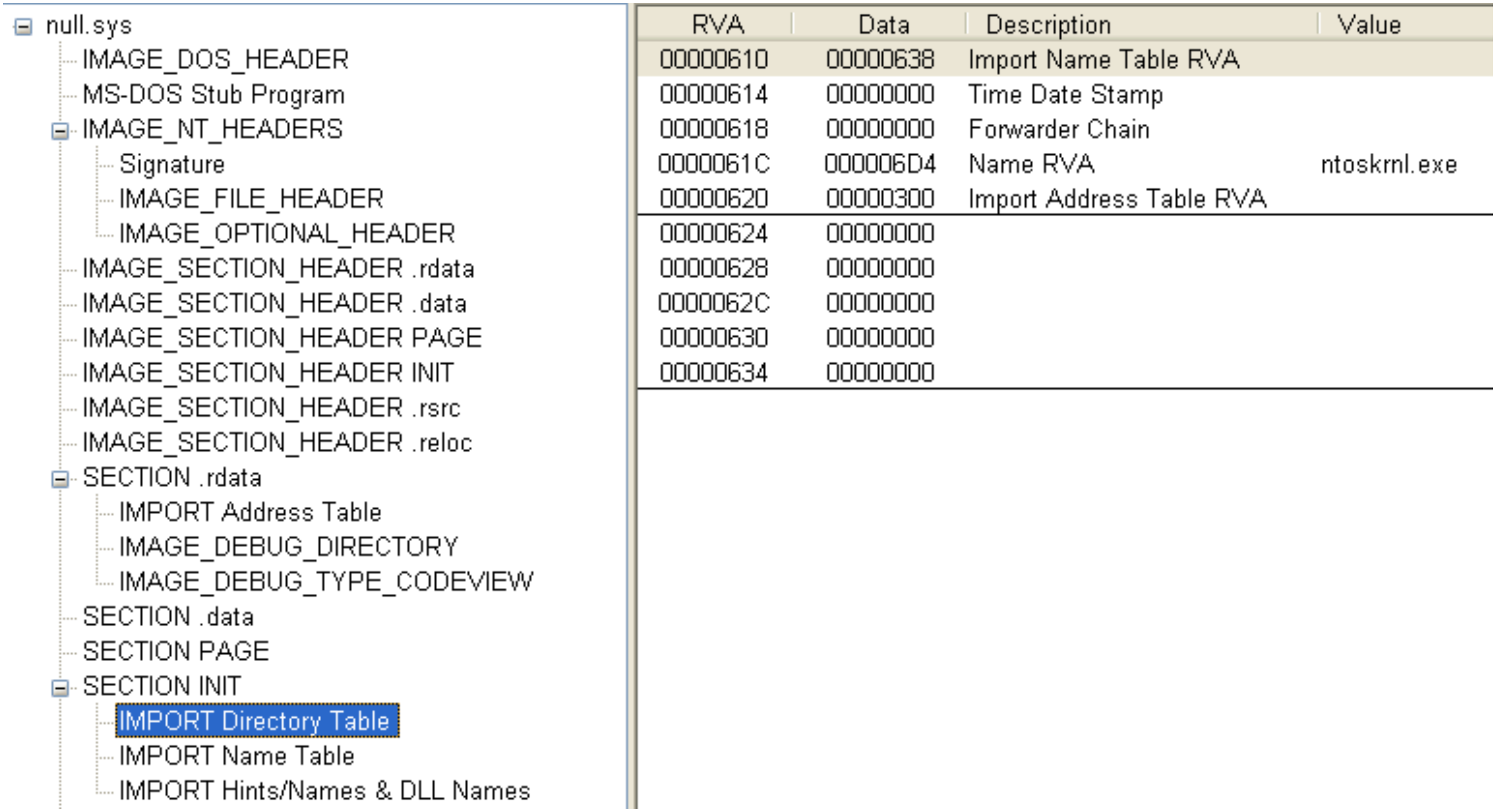
…

Zero-filled
IMAGE_IMPORT_DESCRIPTOR
entry terminates the array

Graphical style borrowed from the Matt Pietrek articles

Look through null.sys

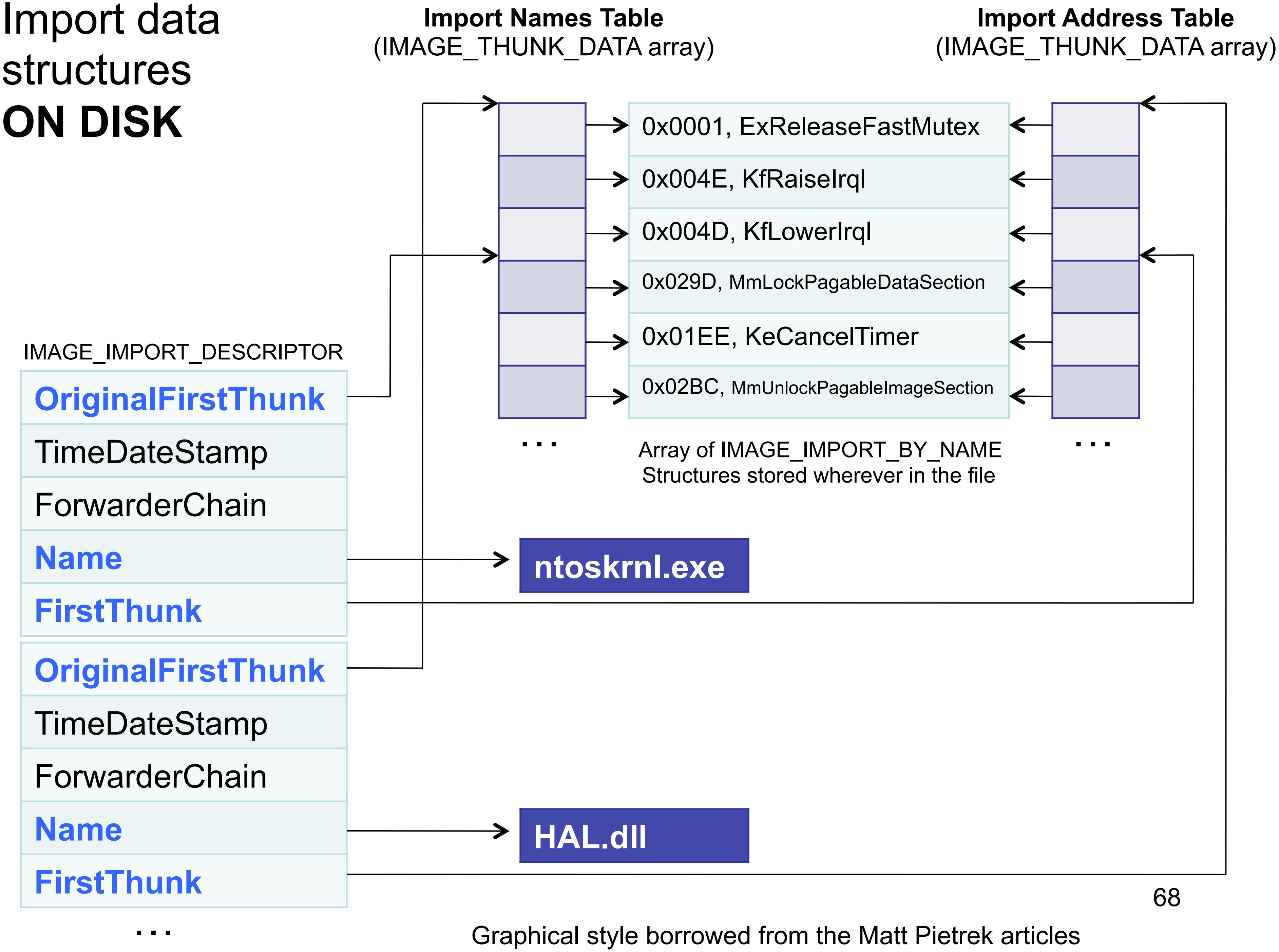
(note to self: start from the data directory)



- null.sys
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
    - Signature
    - IMAGE_FILE_HEADER
    - IMAGE_OPTIONAL_HEADER
  - IMAGE_SECTION_HEADER .rdata
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER PAGE
  - IMAGE_SECTION_HEADER INIT
  - IMAGE_SECTION_HEADER .rsrc
  - IMAGE_SECTION_HEADER .reloc
  - SECTION .rdata
    - IMPORT Address Table
    - IMAGE_DEBUG_DIRECTORY
    - IMAGE_DEBUG_TYPE_CODEVIEW
  - SECTION .data
  - SECTION PAGE
  - SECTION INIT
    - IMPORT Directory Table
    - IMPORT Name Table
    - IMPORT Hints/Names & DLL Names

RVA	Data	Description	Value
00000610	00000638	Import Name Table RVA	
00000614	00000000	Time Date Stamp	
00000618	00000000	Forwarder Chain	
0000061C	000006D4	Name RVA	ntoskrnl.exe
00000620	00000300	Import Address Table RVA	
00000624	00000000		
00000628	00000000		
0000062C	00000000		
00000630	00000000		
00000634	00000000		

Import data structures **ON DISK**



Import Names Table
(IMAGE_THUNK_DATA array)

Import Address Table
(IMAGE_THUNK_DATA array)

0x0001, ExReleaseFastMutex
0x004E, KfRaiseIrql
0x004D, KfLowerIrql
0x029D, MmLockPagableDataSection
0x01EE, KeCancelTimer
0x02BC, MmUnlockPagableImageSection

. . .

Array of IMAGE_IMPORT_BY_NAME
Structures stored wherever in the file

. . .

IMAGE_IMPORT_DESCRIPTOR

OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk

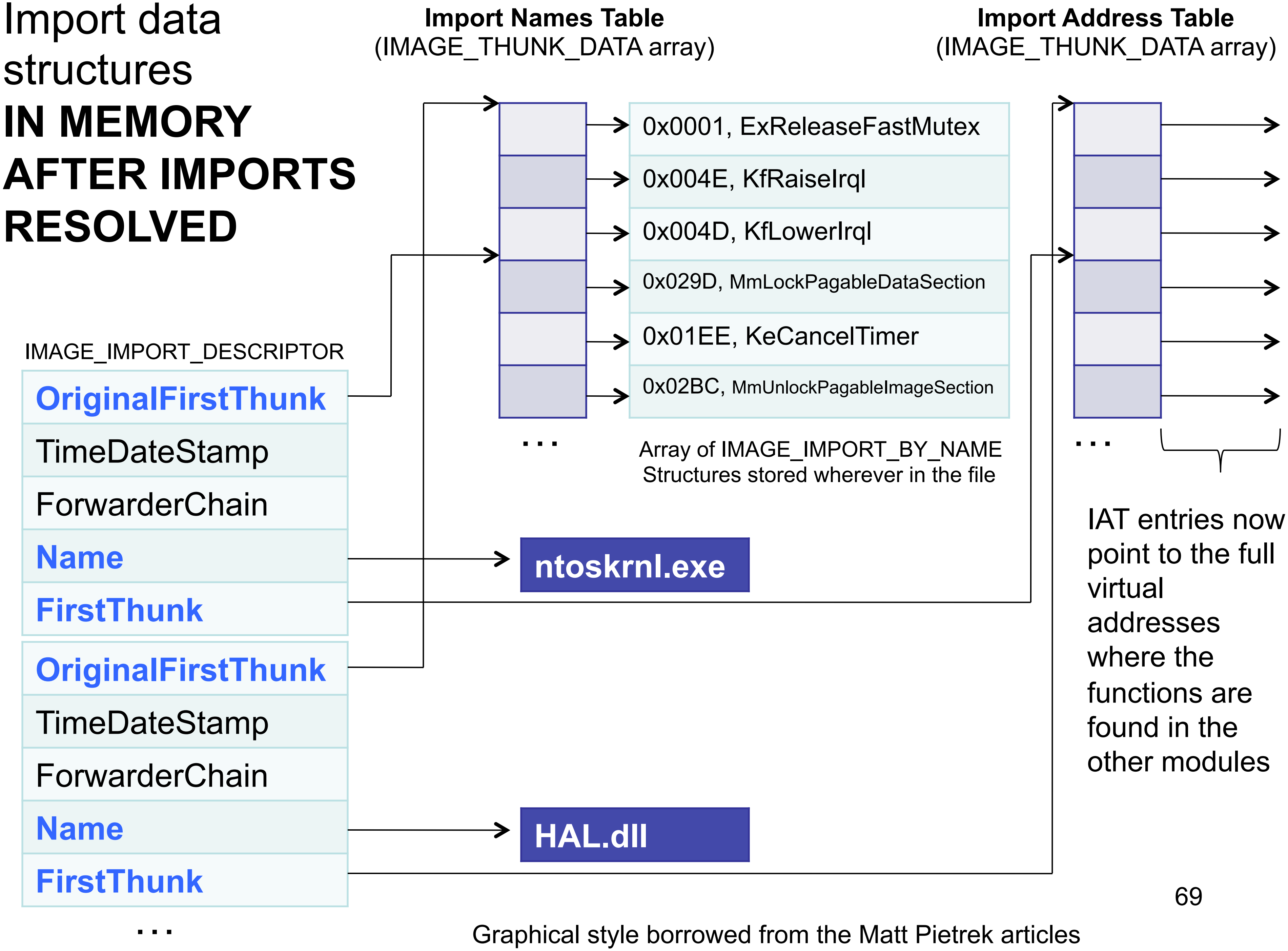
. . .

ntoskrnl.exe

HAL.dll

Graphical style borrowed from the Matt Pietrek articles

Import data structures **IN MEMORY AFTER IMPORTS RESOLVED**



Import Names Table
(IMAGE_THUNK_DATA array)

Import Address Table
(IMAGE_THUNK_DATA array)

IMAGE_IMPORT_DESCRIPTOR

OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk

. . .

ntoskrnl.exe

HAL.dll

0x0001, ExReleaseFastMutex
0x004E, KfRaiseIrql
0x004D, KfLowerIrql
0x029D, MmLockPagableDataSection
0x01EE, KeCancelTimer
0x02BC, MmUnlockPagableImageSection

. . .

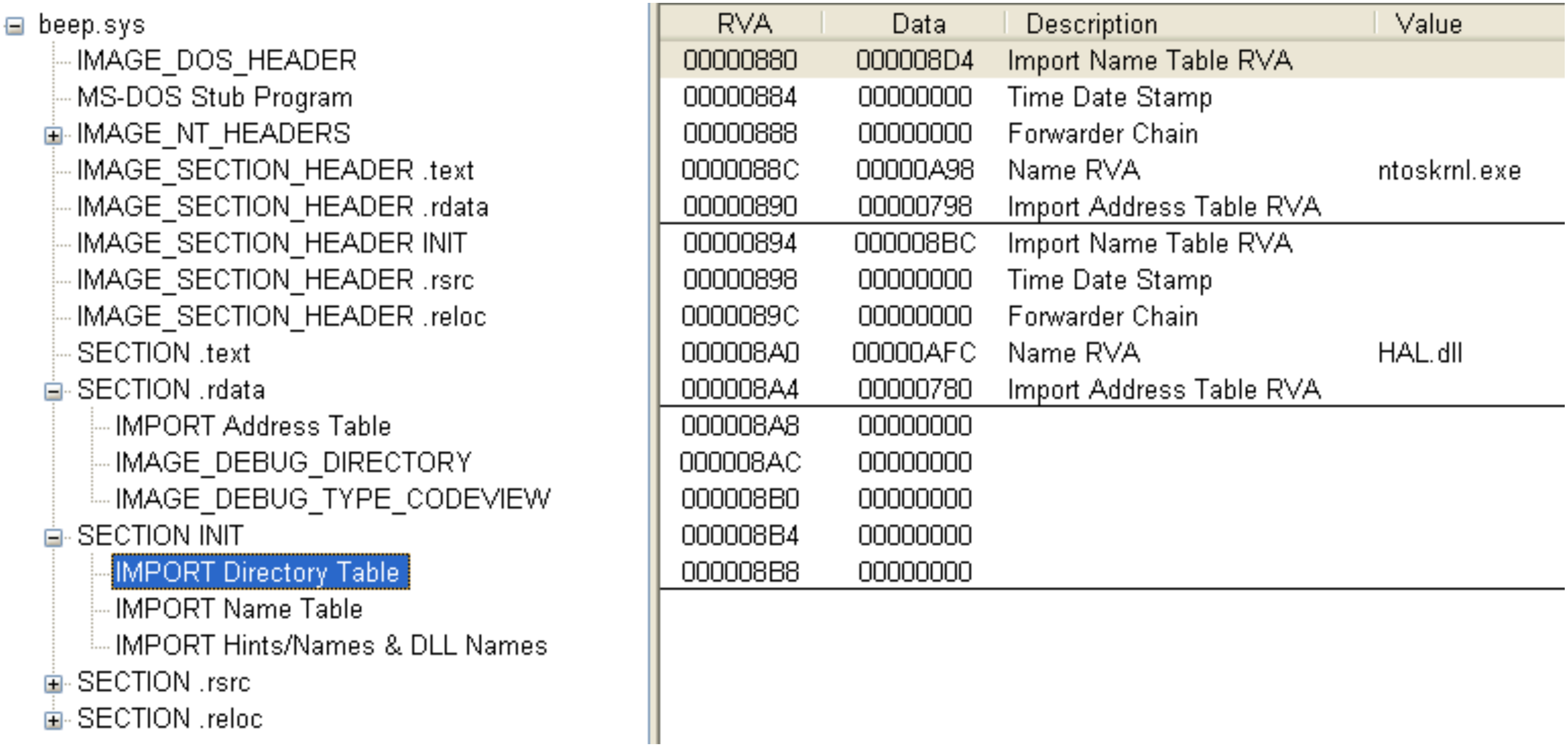
. . .

Array of IMAGE_IMPORT_BY_NAME Structures stored wherever in the file

IAT entries now point to the full virtual addresses where the functions are found in the other modules

Graphical style borrowed from the Matt Pietrek articles

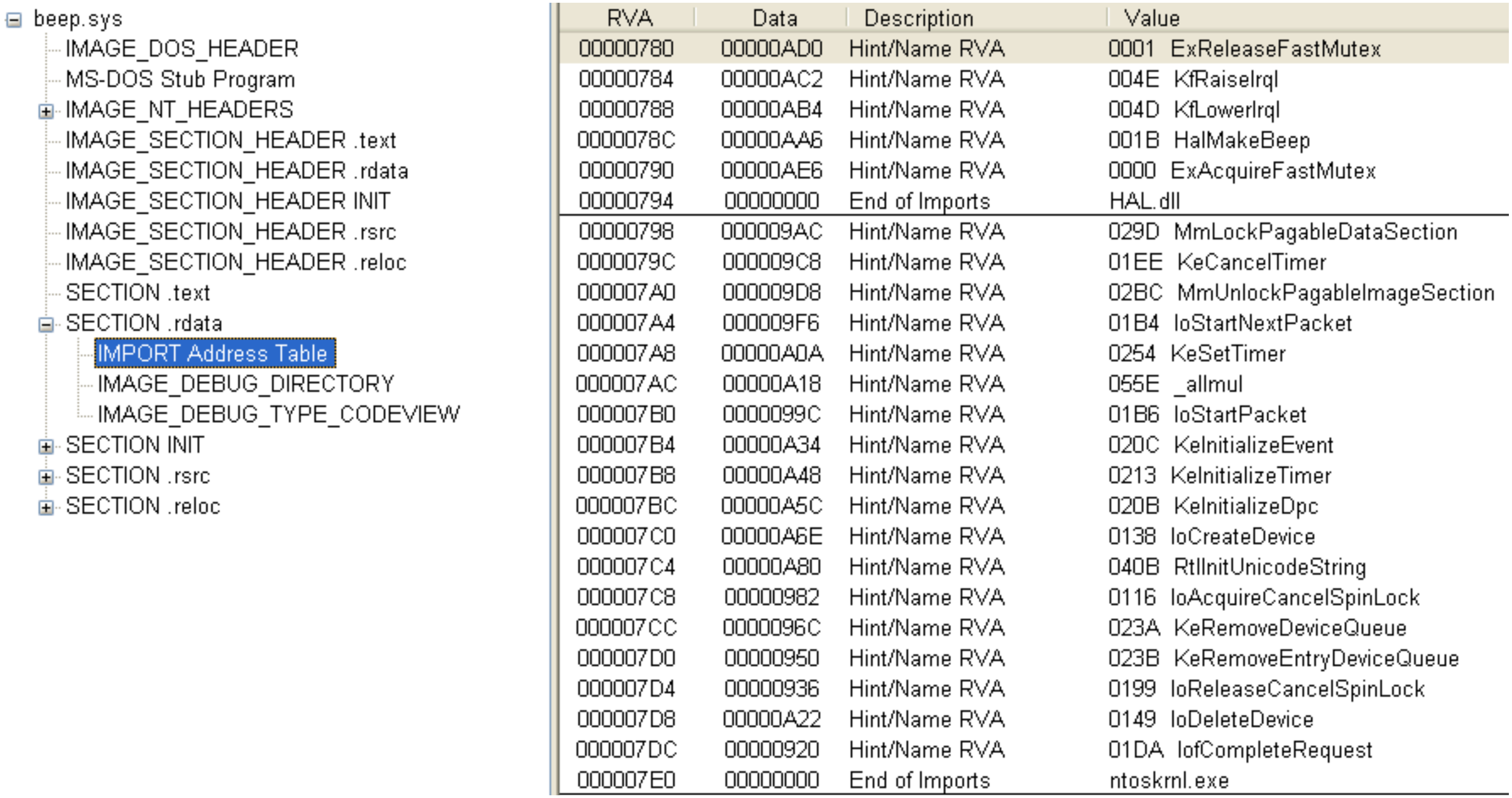
Look through beep.sys



RVA	Data	Description	Value
00000880	000008D4	Import Name Table RVA	
00000884	00000000	Time Date Stamp	
00000888	00000000	Forwarder Chain	
0000088C	00000A98	Name RVA	ntoskrnl.exe
00000890	00000798	Import Address Table RVA	
00000894	000008BC	Import Name Table RVA	
00000898	00000000	Time Date Stamp	
0000089C	00000000	Forwarder Chain	
000008A0	00000AFC	Name RVA	HAL.dll
000008A4	00000780	Import Address Table RVA	
000008A8	00000000		
000008AC	00000000		
000008B0	00000000		
000008B4	00000000		
000008B8	00000000		

nt then hal, no special significance, just sayin’

Look through beep.sys 2



- beep.sys
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .rdata
  - IMAGE_SECTION_HEADER INIT
  - IMAGE_SECTION_HEADER .rsrc
  - IMAGE_SECTION_HEADER .reloc
  - SECTION .text
  - SECTION .rdata
    - IMPORT Address Table
    - IMAGE_DEBUG_DIRECTORY
    - IMAGE_DEBUG_TYPE_CODEVIEW
  - SECTION INIT
  - SECTION .rsrc
  - SECTION .reloc

RVA	Data	Description	Value
00000780	00000AD0	Hint/Name RVA	0001 ExReleaseFastMutex
00000784	00000AC2	Hint/Name RVA	004E KfRaiseIrql
00000788	00000AB4	Hint/Name RVA	004D KfLowerIrql
0000078C	00000AA6	Hint/Name RVA	001B HalMakeBeep
00000790	00000AE6	Hint/Name RVA	0000 ExAcquireFastMutex
00000794	00000000	End of Imports	HAL.dll
00000798	000009AC	Hint/Name RVA	029D MmLockPagableDataSection
0000079C	000009C8	Hint/Name RVA	01EE KeCancelTimer
000007A0	000009D8	Hint/Name RVA	02BC MmUnlockPagableImageSection
000007A4	000009F6	Hint/Name RVA	01B4 IoStartNextPacket
000007A8	00000A0A	Hint/Name RVA	0254 KeSetTimer
000007AC	00000A18	Hint/Name RVA	055E _allmul
000007B0	0000099C	Hint/Name RVA	01B6 IoStartPacket
000007B4	00000A34	Hint/Name RVA	020C KeInitializeEvent
000007B8	00000A48	Hint/Name RVA	0213 KeInitializeTimer
000007BC	00000A5C	Hint/Name RVA	020B KeInitializeDpc
000007C0	00000A6E	Hint/Name RVA	0138 IoCreateDevice
000007C4	00000A80	Hint/Name RVA	040B RtlInitUnicodeString
000007C8	00000982	Hint/Name RVA	0116 IoAcquireCancelSpinLock
000007CC	0000096C	Hint/Name RVA	023A KeRemoveDeviceQueue
000007D0	00000950	Hint/Name RVA	023B KeRemoveEntryDeviceQueue
000007D4	00000936	Hint/Name RVA	0199 IoReleaseCancelSpinLock
000007D8	00000A22	Hint/Name RVA	0149 IoDeleteDevice
000007DC	00000920	Hint/Name RVA	01DA IofCompleteRequest
000007E0	00000000	End of Imports	ntoskrnl.exe

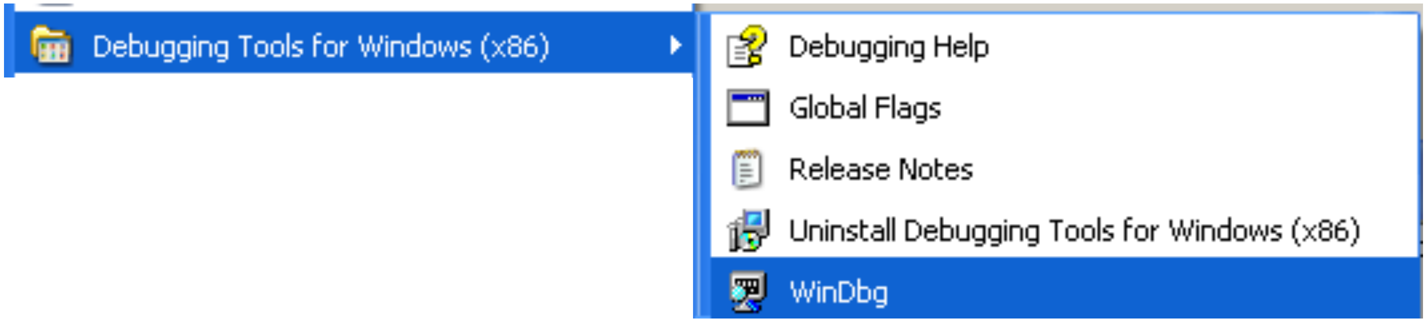
hal then nt, no special significance, just sayin’ it’s backwards from the previous

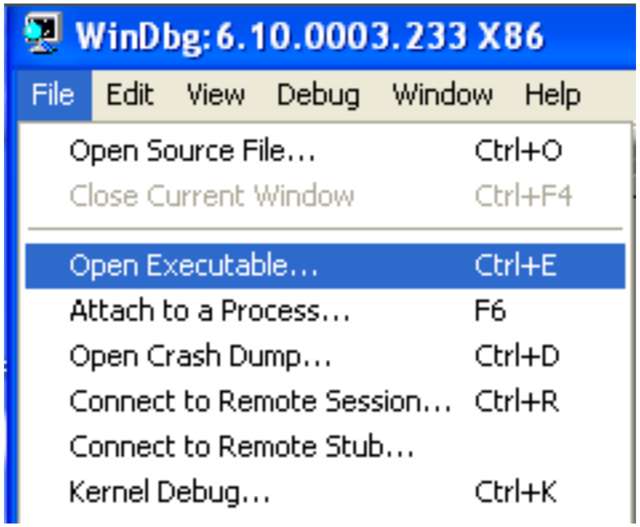
Lab: telnet.exe

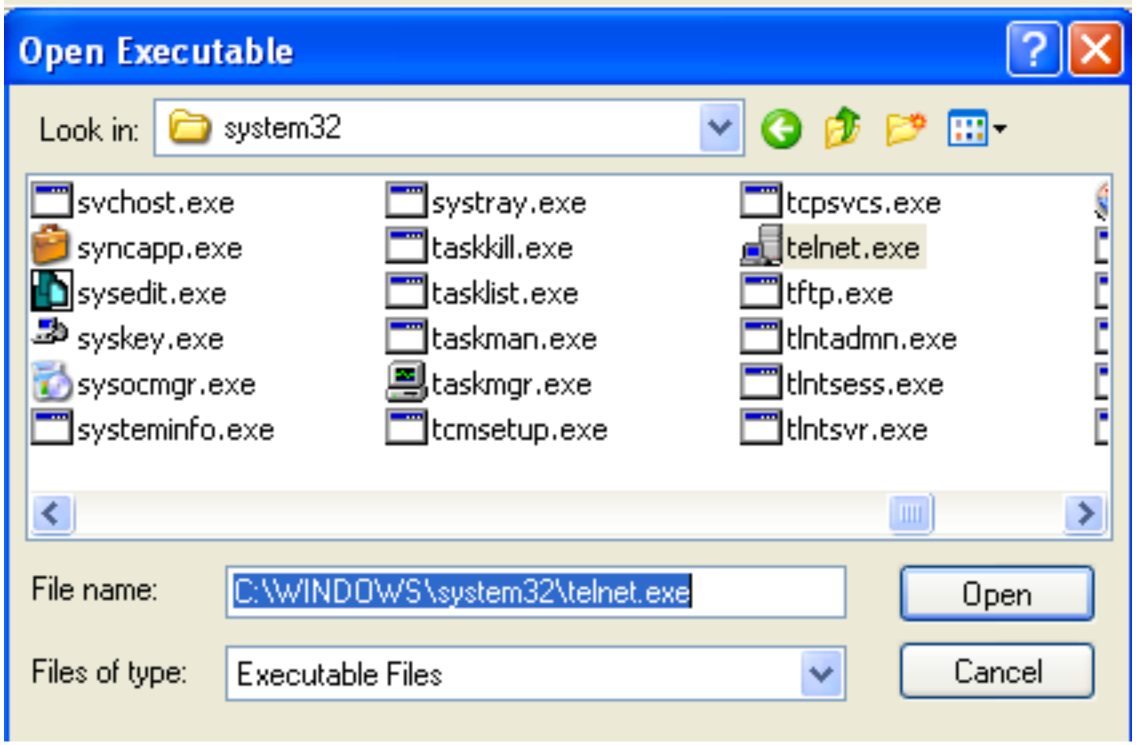
- telnet.exe was chosen because it has only normal imports; no “bound” or “delayed” imports as will be talked about later
- View imports with PEView
- Open telnet.exe
- View imports in memory by attaching with WinDbg

Open WinDbg

From Start Menu


Debugging Tools for Windows (x86)
Debugging Help
Global Flags
Release Notes
Uninstall Debugging Tools for Windows (x86)
WinDbg

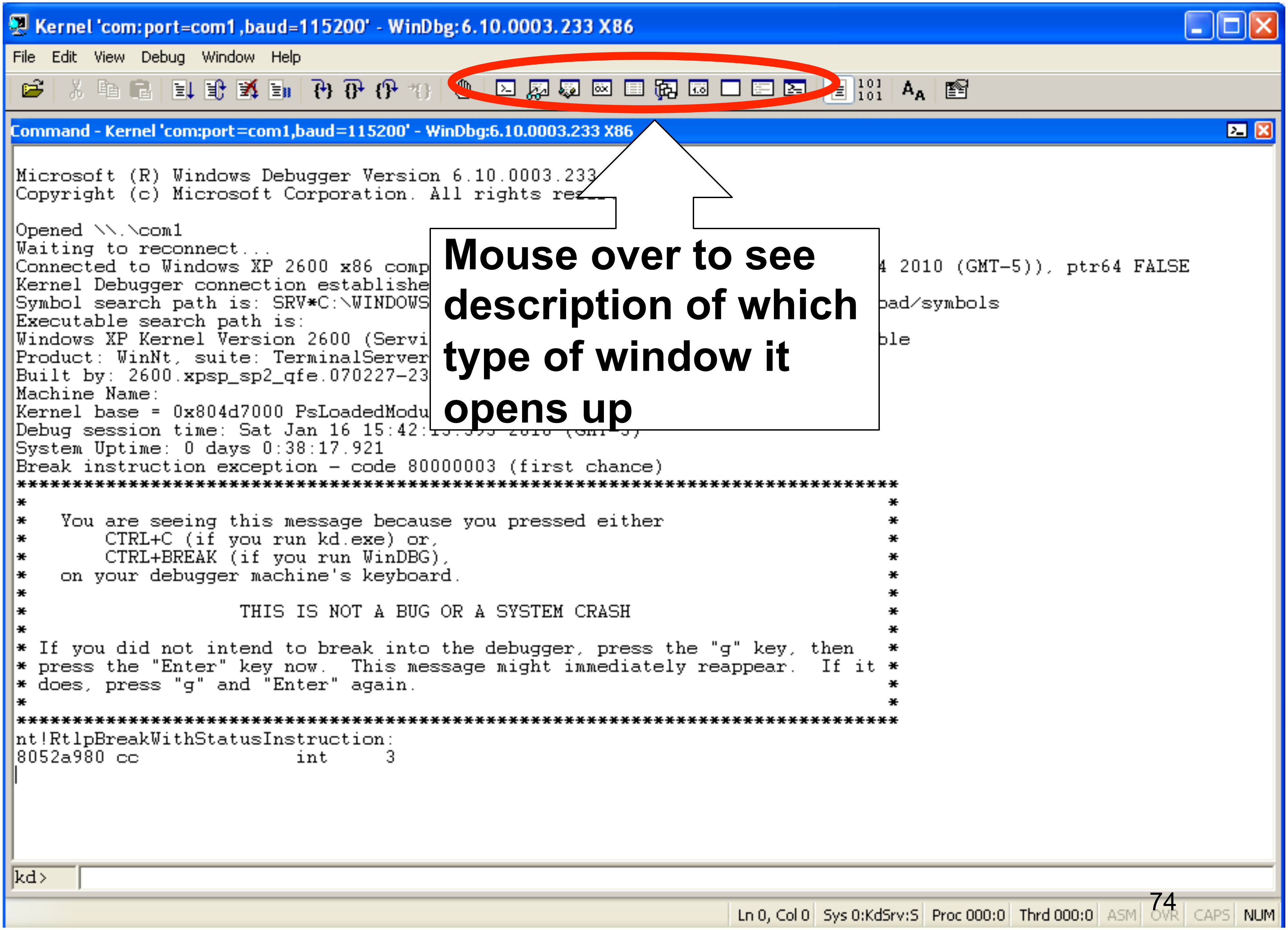

WinDbg:6.10.0003.233 X86
File
Edit
View
Debug
Window
Help
Open Source File... Ctrl+O
Close Current Window Ctrl+F4
Open Executable... Ctrl+E
Attach to a Process... F6
Open Crash Dump... Ctrl+D
Connect to Remote Session... Ctrl+R
Connect to Remote Stub...
Kernel Debug... Ctrl+K



Look in: system32

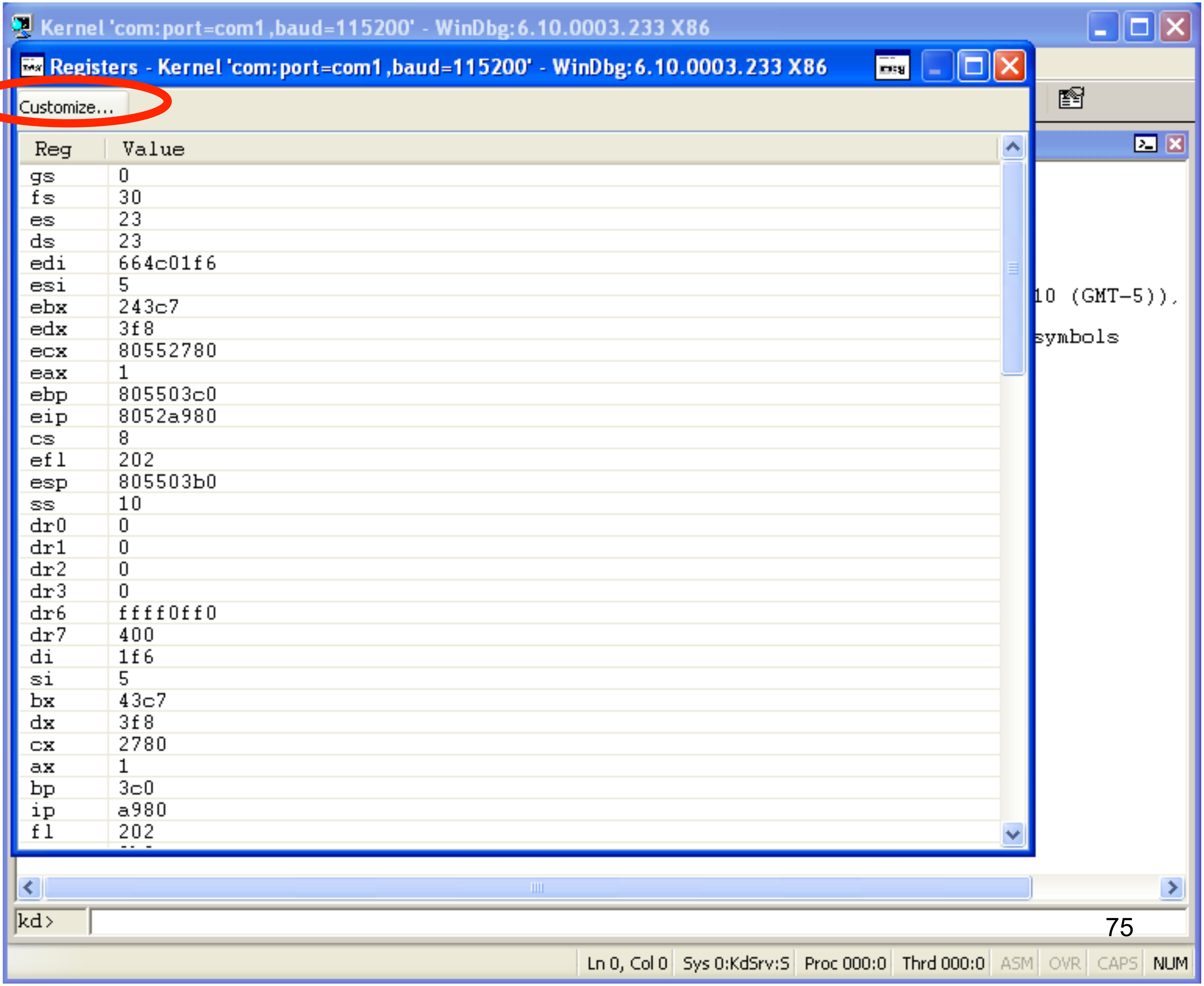
File name: C:\WINDOWS\system32\telnet.exe

Files of type: Executable Files


Kernel 'com:port=com1,baud=115200' - WinDbg:6.10.0003.233 X86
File Edit View Debug Window Help
Command - Kernel 'com:port=com1,baud=115200' - WinDbg:6.10.0003.233 X86
kd>
Ln 0, Col 0 Sys 0:KdSrv:S Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM

```
Microsoft (R) Windows Debugger Version 6.10.0003.233
Copyright (c) Microsoft Corporation. All rights res

Opened \\.\com1
Waiting to reconnect...
Connected to Windows XP 2600 x86 comp                                4 2010 (GMT-5)), ptr64 FALSE
Kernel Debugger connection establishe
Symbol search path is: SRV*C:\WINDOWS                                oad/symbols
Executable search path is:
Windows XP Kernel Version 2600 (Servi                                ble
Product: WinNt, suite: TerminalServer
Built by: 2600.xpsp_sp2_qfe.070227-23
Machine Name:
Kernel base = 0x804d7000 PsLoadedModu
Debug session time: Sat Jan 16 15:42:
System Uptime: 0 days 0:38:17.921
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
*                                                                             *
*   You are seeing this message because you pressed either                    *
*       CTRL+C (if you run kd.exe) or,                                        *
*       CTRL+BREAK (if you run WinDBG),                                       *
*   on your debugger machine's keyboard.                                      *
*                                                                             *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                       *
*                                                                             *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
nt!RtlpBreakWithStatusInstruction:
8052a980 cc              int     3
```

Mouse over to see description of which type of window it opens up



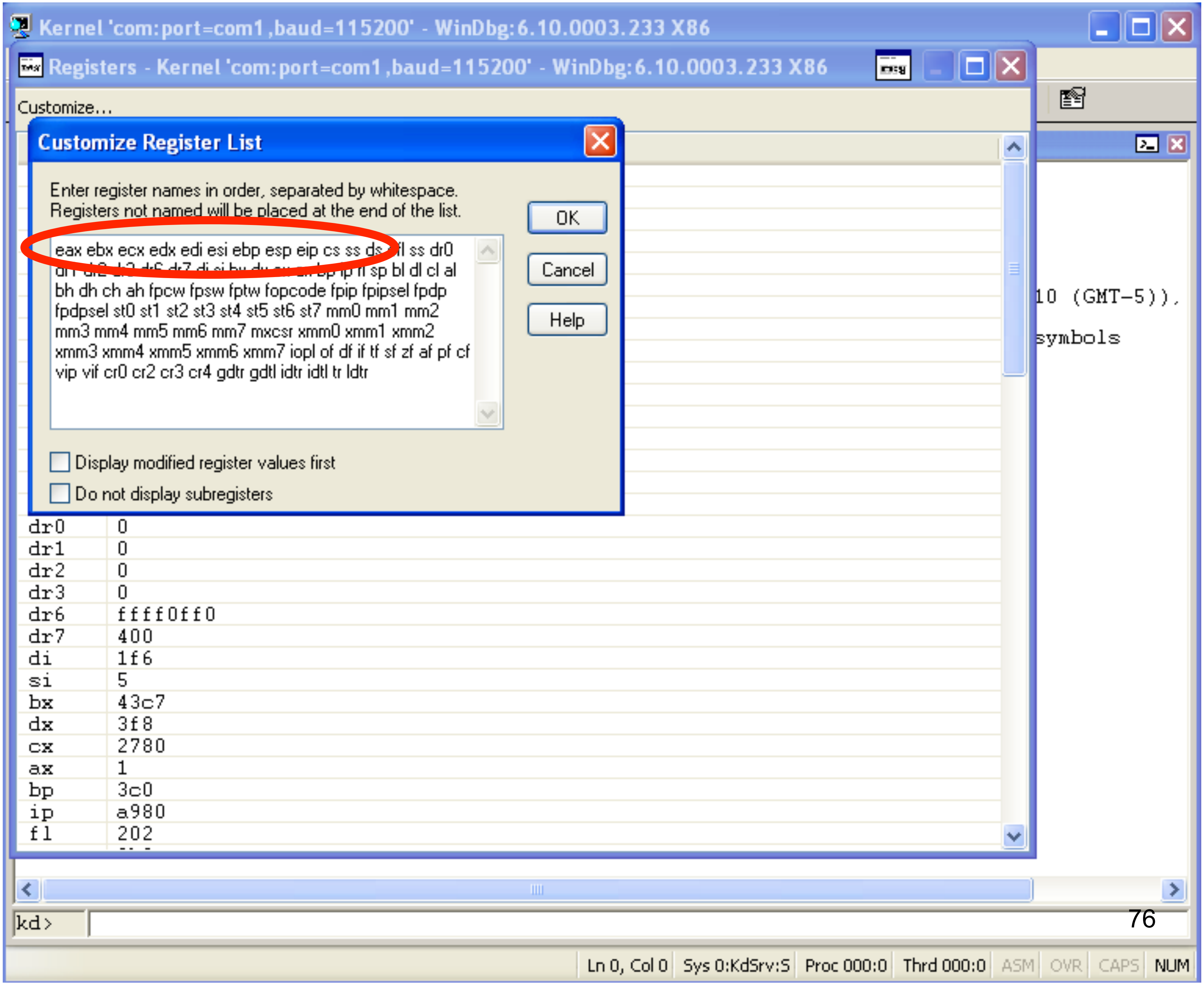
Kernel 'com:port=com1,baud=115200' - WinDbg:6.10.0003.233 X86

Registers - Kernel 'com:port=com1,baud=115200' - WinDbg:6.10.0003.233 X86

```
Reg   Value
gs    0
fs    30
es    23
ds    23
edi   664c01f6
esi   5
ebx   243c7
edx   3f8
ecx   80552780
eax   1
ebp   805503c0
eip   8052a980
cs    8
efl   202
esp   805503b0
ss    10
dr0   0
dr1   0
dr2   0
dr3   0
dr6   ffff0ff0
dr7   400
di    1f6
si    5
bx    43c7
dx    3f8
cx    2780
ax    1
bp    3c0
ip    a980
fl    202
```

```
10 (GMT-5)),
symbols
```

```
kd>
```


Kernel 'com:port=com1,baud=115200' - WinDbg:6.10.0003.233 X86
Registers - Kernel 'com:port=com1,baud=115200' - WinDbg:6.10.0003.233 X86
Customize...
Customize Register List
OK
Cancel
Help

Enter register names in order, separated by whitespace.
Registers not named will be placed at the end of the list.

```
eax ebx ecx edx edi esi ebp esp eip cs ss ds efl ss dr0
dr1 dr2 dr3 dr6 dr7 di si bx dx cx ax bp ip fl sp bl dl cl al
bh dh ch ah fpcw fpsw fptw fopcode fpip fpipsel fpdp
fpdpsel st0 st1 st2 st3 st4 st5 st6 st7 mm0 mm1 mm2
mm3 mm4 mm5 mm6 mm7 mxcsr xmm0 xmm1 xmm2
xmm3 xmm4 xmm5 xmm6 xmm7 iopl of df if tf sf zf af pf cf
vip vif cr0 cr2 cr3 cr4 gdtr gdtl idtr idtl tr ldtr
```

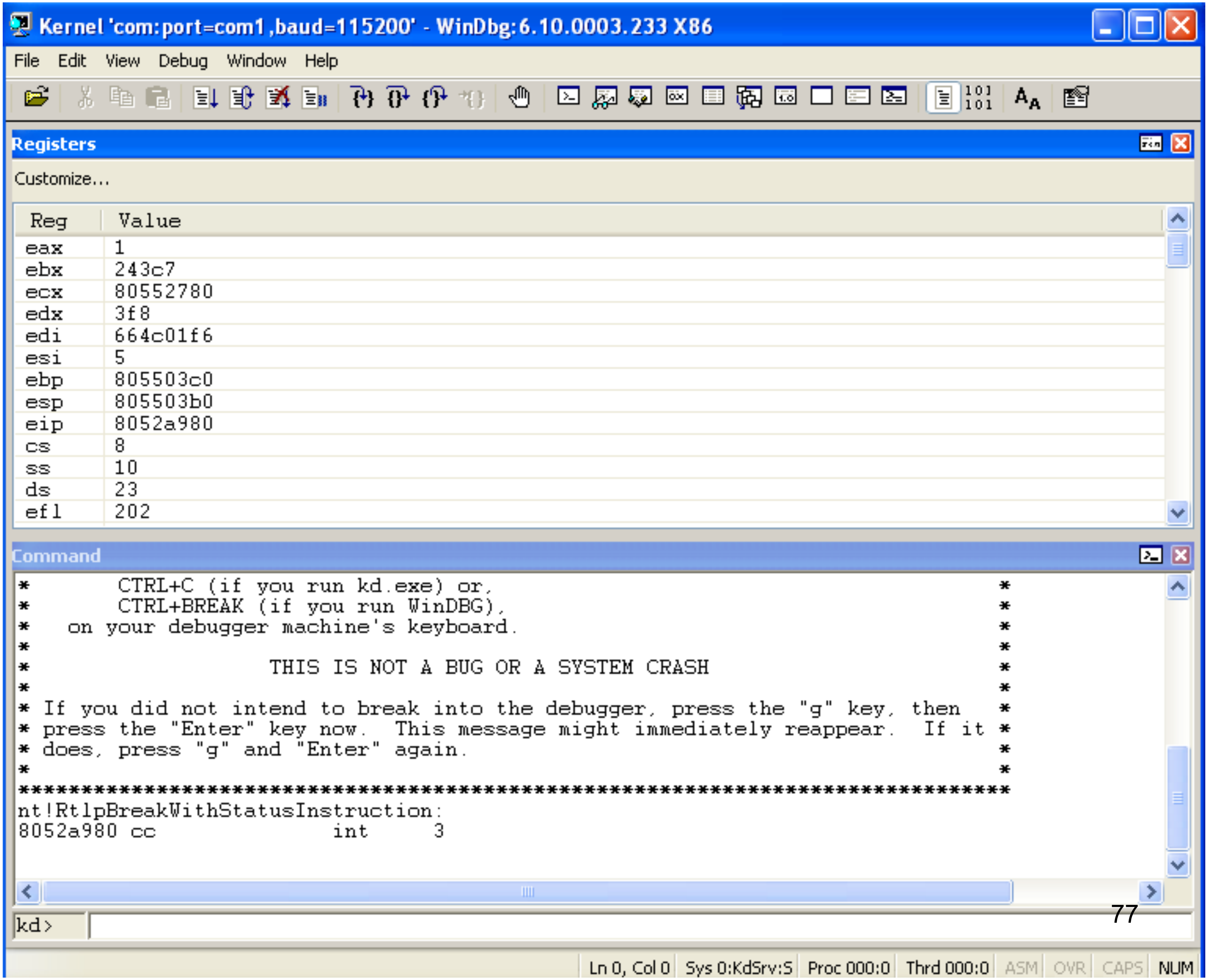
☐ Display modified register values first

☐ Do not display subregisters

```
dr0     0
dr1     0
dr2     0
dr3     0
dr6     ffff0ff0
dr7     400
di      1f6
si      5
bx      43c7
dx      3f8
cx      2780
ax      1
bp      3c0
ip      a980
fl      202
```

```
10 (GMT-5)),
symbols
```

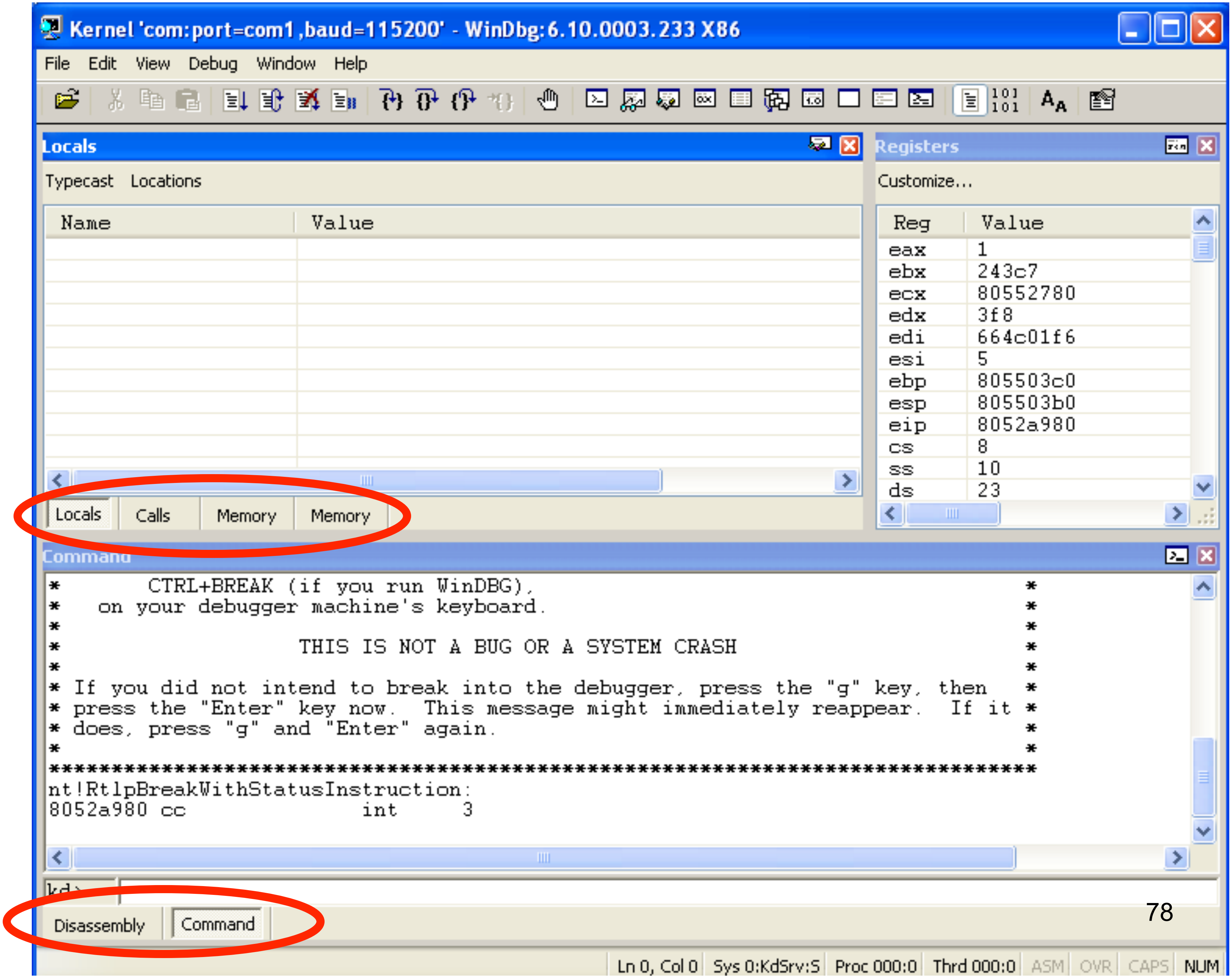
```
kd>
```



Reg	Value
eax	1
ebx	243c7
ecx	80552780
edx	3f8
edi	664c01f6
esi	5
ebp	805503c0
esp	805503b0
eip	8052a980
cs	8
ss	10
ds	23
efl	202

```
*       CTRL+C (if you run kd.exe) or,                                    *
*       CTRL+BREAK (if you run WinDBG),                                   *
*   on your debugger machine's keyboard.                                  *
*                                                                         *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                   *
*                                                                         *
* If you did not intend to break into the debugger, press the "g" key, then  *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                      *
*                                                                         *
*******************************************************************************
nt!RtlpBreakWithStatusInstruction:
8052a980 cc              int     3
```

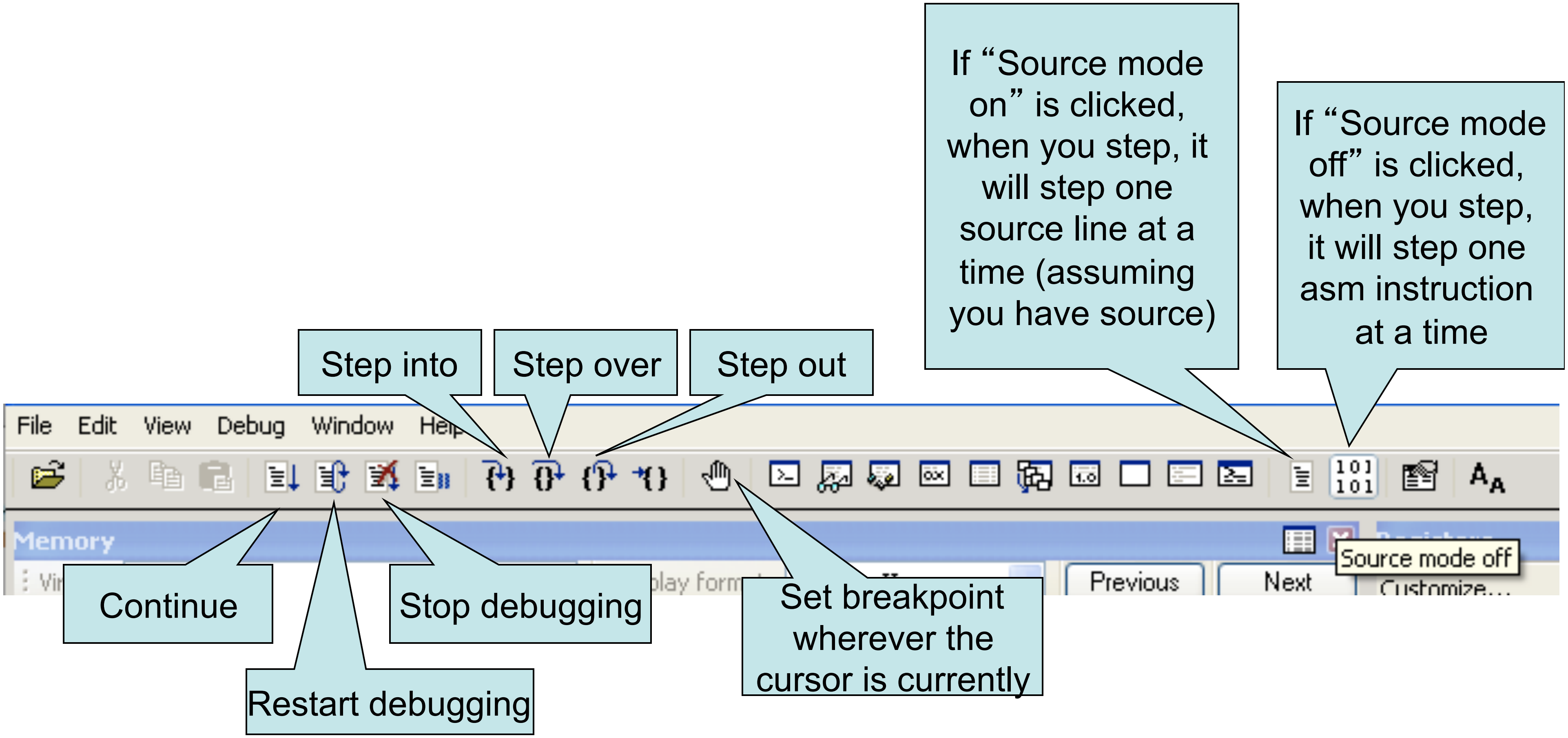
```
kd>
```



Reg	Value
eax	1
ebx	243c7
ecx	80552780
edx	3f8
edi	664c01f6
esi	5
ebp	805503c0
esp	805503b0
eip	8052a980
cs	8
ss	10
ds	23

```
*       CTRL+BREAK (if you run WinDBG),                                      *
*   on your debugger machine's keyboard.                                     *
*                                                                            *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                      *
*                                                                            *
* If you did not intend to break into the debugger, press the "g" key, then  *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                         *
*                                                                            *
******************************************************************************
nt!RtlpBreakWithStatusInstruction:
8052a980 cc              int     3
```

```
kd>
```



Step into

Step over

Step out

If “Source mode on” is clicked, when you step, it will step one source line at a time (assuming you have source)

If “Source mode off” is clicked, when you step, it will step one asm instruction at a time

Continue

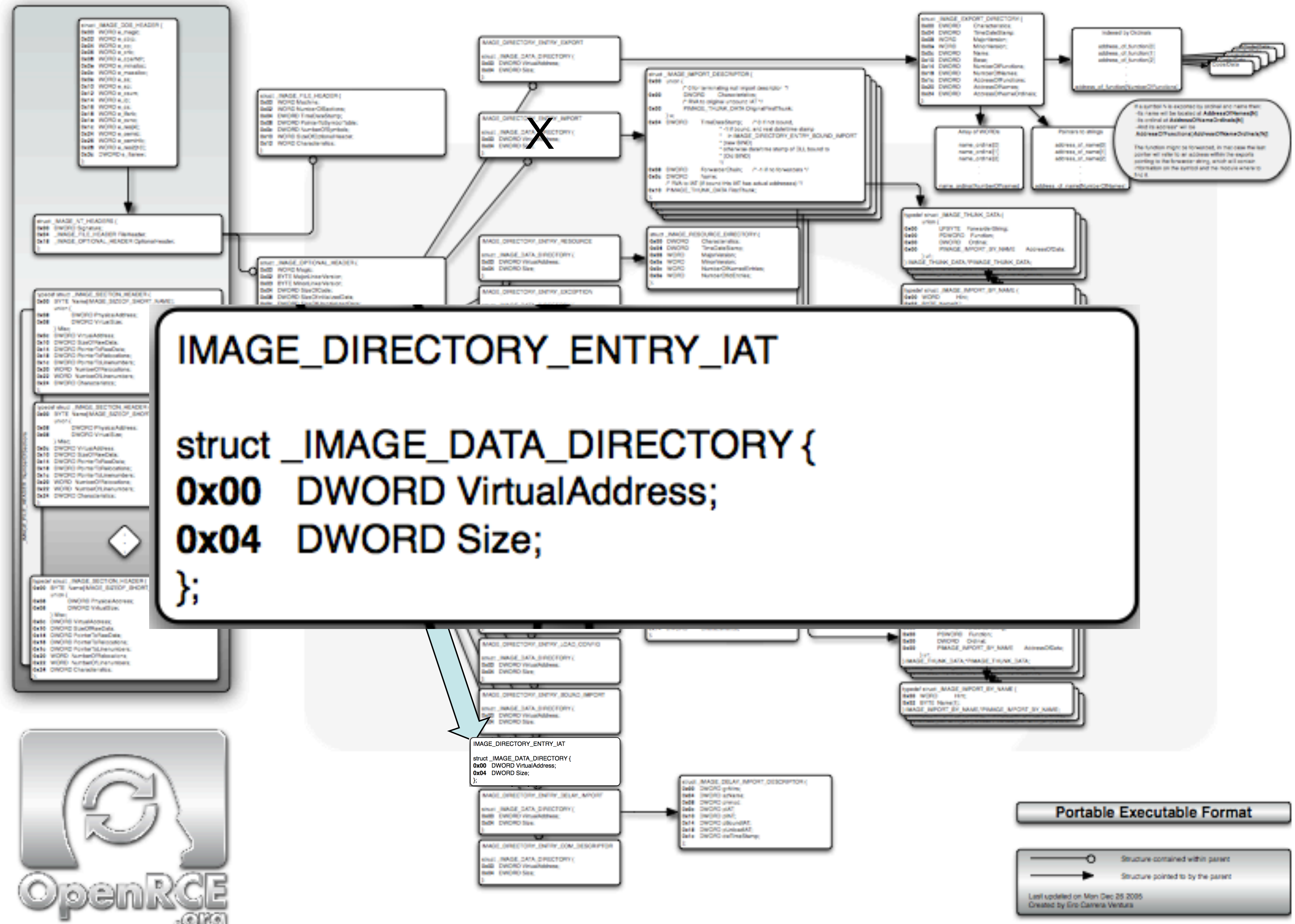
Restart debugging

Stop debugging

Set breakpoint wherever the cursor is currently

WinDbg breakpoints

- bp <address> : Set breakpoint
  - Address can be number or human readable input like “main” or “Example1:main”
- bl : Breakpoints list
- bd <bp ID> : Breakpoint disable
  - <bp ID> as given by first column of bl
- be <bp ID> : Breakpoint enable
  - <bp ID> as given by first column of bl
- bc <bp ID> : Breakpoint clear (delete)
  - Can do "bc *" to delete all breakpoints


X
IMAGE_DIRECTORY_ENTRY_IAT
Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE
.org

IMAGE_DIRECTORY_ENTRY_IAT

```
struct _IMAGE_DATA_DIRECTORY {
0x00   DWORD VirtualAddress;
0x04   DWORD Size;
};
```

Image by Ero Carrera

IAT Hooking

- When the IAT is fully resolved, it is basically an array of function pointers. Somewhere, in some code path, there’s something which is going to take an IAT address, and use whatever’s in that memory location as the destination of the code it should call.
- What if the “whatever’s in that memory location” gets changed after the OS loader is done? What if it points at attacker code?

IAT Hooking 2

- Well, that would mean the attacker's code would functionally be "man-in-the-middle"ing the call to the function. He can then change parameters before forwarding the call on to the original function, and filter results that come back from the function, or simply never call the original function, and send back whatever status he pleases.
  - Think rootkits. Say you're calling OpenFile. It looks at the file name and if you're asking for a file it wants to hide, it simply returns "no file found."
- But how does the attacker change the IAT entries? This is a question of assumptions about where the attacker is.

IAT Hooking 3

- In a traditional memory-corrupting exploit, the attacker is, by definition, in the memory space of the attacked process, upon successfully gaining arbitrary code execution. The attacker can now change memory such as the IAT for this process only, because remember (from OS class or Intermediate x86) each process has a separate memory space.
- If the attacker wants to change the IAT on other processes, he must be in their memory spaces as well. Typically the attacker will format some of his code as a DLL and then perform “DLL Injection” in order to get his code in other process’ memory space.
- The ability to do something like DLL injection is generally a prerequisite in order to leverage IAT hooking across many userspace processes. In the kernel, kernel modules are generally all sharing the same memory space with the kernel, and therefore one subverted kernel module can hook the IAT of any other modules that it wants.

DLL Injection

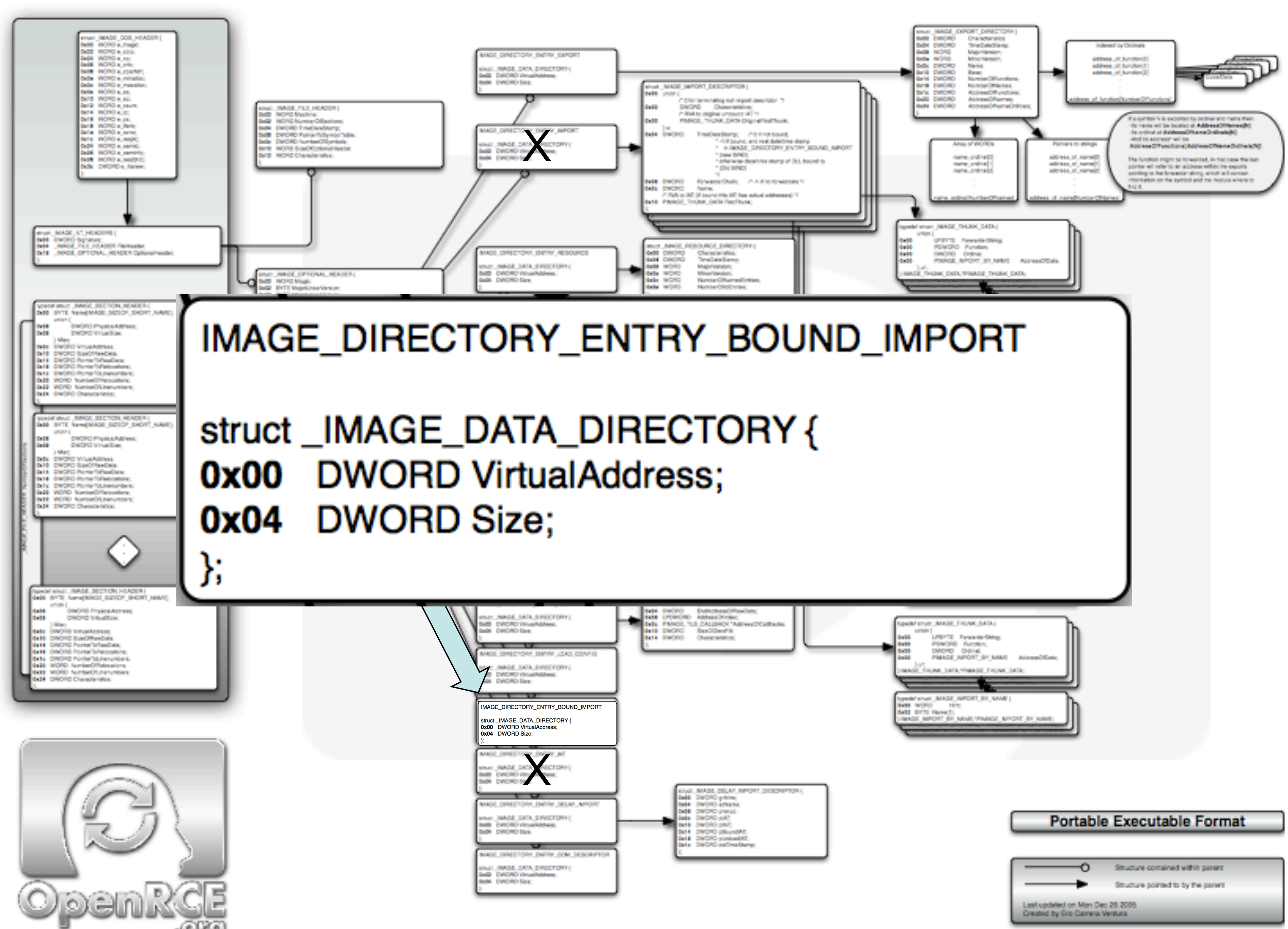
- See http://en.wikipedia.org/wiki/DLL_injection for more ways that this can be achieved on Windows/*nix
- We’re going to use the AppInit_DLLs way of doing this, out of laziness
- (Note: AppInit_DLLs' behavior has changed in releases > XP, it now has to be enabled with Administrator level permissions.)

Lab: IAT hooking

- http://www.codeproject.com/KB/vista/api-hooks.aspx
  - This will hook NtQuerySystemInformation(), which is what taskmgr.exe uses in order to list the currently running processes. It will replace this with HookedNtQuerySystemInformation(), which will hide calc.exe
  - I modified that code to use IAT hooking rather than inline (which is much simpler actually)
- Steps:
  - Compile AppInitHookIAT.dll
  - Place at C:\AppInitHookIAT.dll for simplicity
  - Use regedit.exe to add C:\AppInitHookIAT.dll as the value for the key **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs** (if there is already something there, separate the entries with a comma)
  - Start calc.exe, start taskmgr.exe, confirm that calc.exe doesn't show up in the list of running processes.
  - Remove C:\AppInitHookIAT.dll from AppInit_DLLs and restart taskmgr.exe.
  - Confirm calc.exe shows up in the list of running processes.
  - (This is a basic "userspace rootkit" technique. Because of this, all entries in this registry key should always be looked upon with suspicion.)

Bound Imports

- Import binding is a speed optimization. The addresses of the functions are resolved at link time, and then placed into the IAT.
- The binding is done under the assumption of specific versions of the DLL. If the DLL changes, then all the IAT entries will be invalid. But that just means you have to resolve them, so you’re not much worse off than if you had not used binding in the first place.
- notepad.exe and a bunch of other stuff in C:\WINDOWS\system32 are examples



IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT

```
struct _IMAGE_DATA_DIRECTORY {
0x00   DWORD VirtualAddress;
0x04   DWORD Size;
};
```

X
X
Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventures
OpenRCE.org

Image by Ero Carrera

Missing from the picture

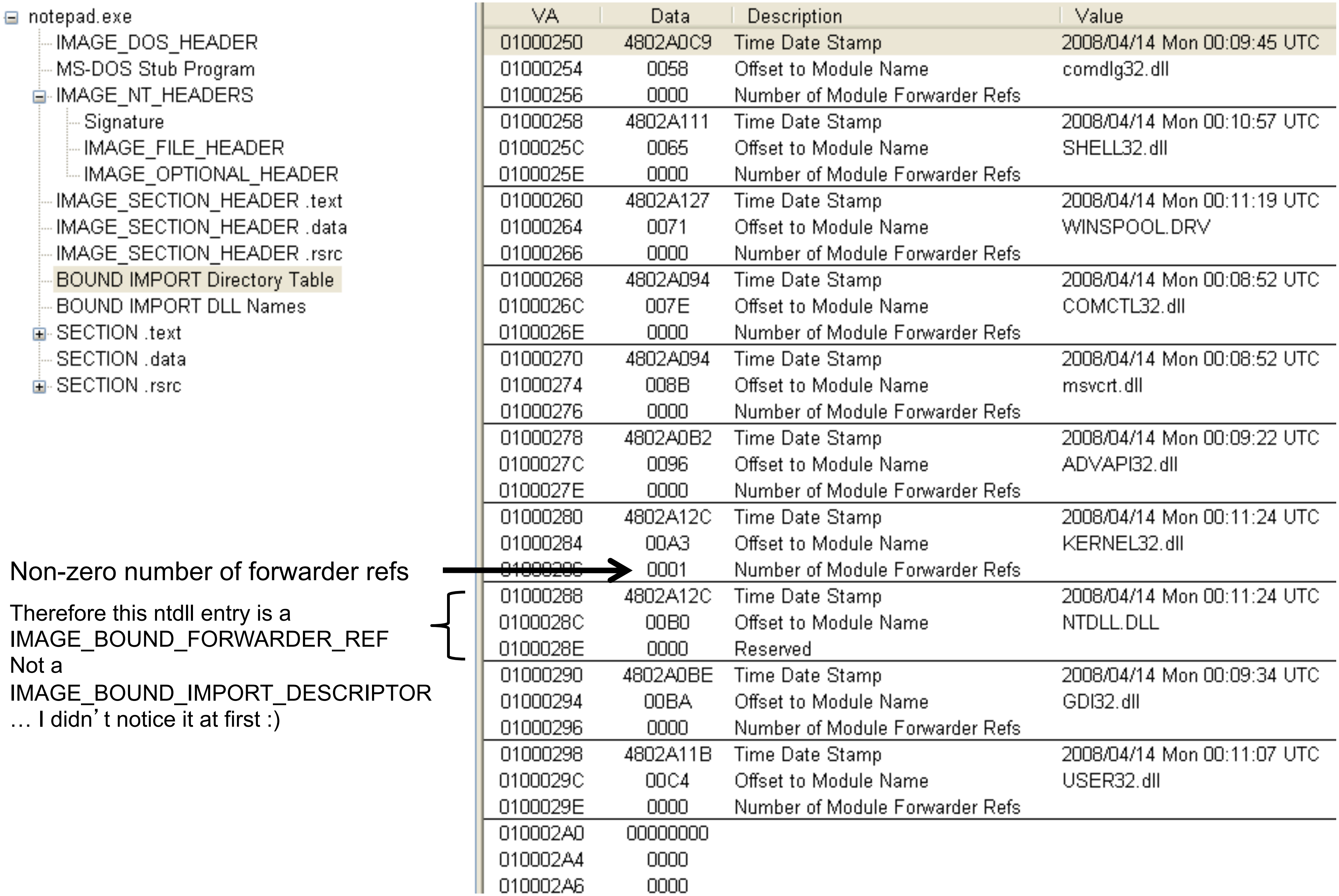
- The bound import data directory entry points at an array of IMAGE_BOUND_IMPORT_DESCRIPTORs, ending with an all-zeros IMAGE_BOUND_IMPORT_DESCRIPTOR (like what was done with IMAGE_IMPORT_DESCRIPTOR)

```
typedef struct _IMAGE_BOUND_IMPORT_DESCRIPTOR {
   DWORD   TimeDateStamp;
   WORD    OffsetModuleName;
   WORD    NumberOfModuleForwarderRefs;
// Array of zero or more IMAGE_BOUND_FORWARDER_REF follows
} IMAGE_BOUND_IMPORT_DESCRIPTOR,  *PIMAGE_BOUND_IMPORT_DESCRIPTOR;

typedef struct _IMAGE_BOUND_FORWARDER_REF {
   DWORD   TimeDateStamp;
   WORD    OffsetModuleName;
   WORD    Reserved;
} IMAGE_BOUND_FORWARDER_REF, *PIMAGE_BOUND_FORWARDER_REF;
```

IMAGE_BOUND_IMPORT_DESCRIPTOR

- **TimeDateStamp** is just the value from the FileHeader as we would expect.
- **OffsetModuleName** is not an RVA, it’s the offset from the beginning of the first IMAGE_BOUND_IMPORT_DESCRIPTOR
- We are going to return to NumberOfModuleForwarderRefs and IMAGE_BOUND_FORWARDER_REF after we learn about forwarded functions.

Notepad.exe’s IMAGE_BOUND_IMPORT_DESCRIPTOR array



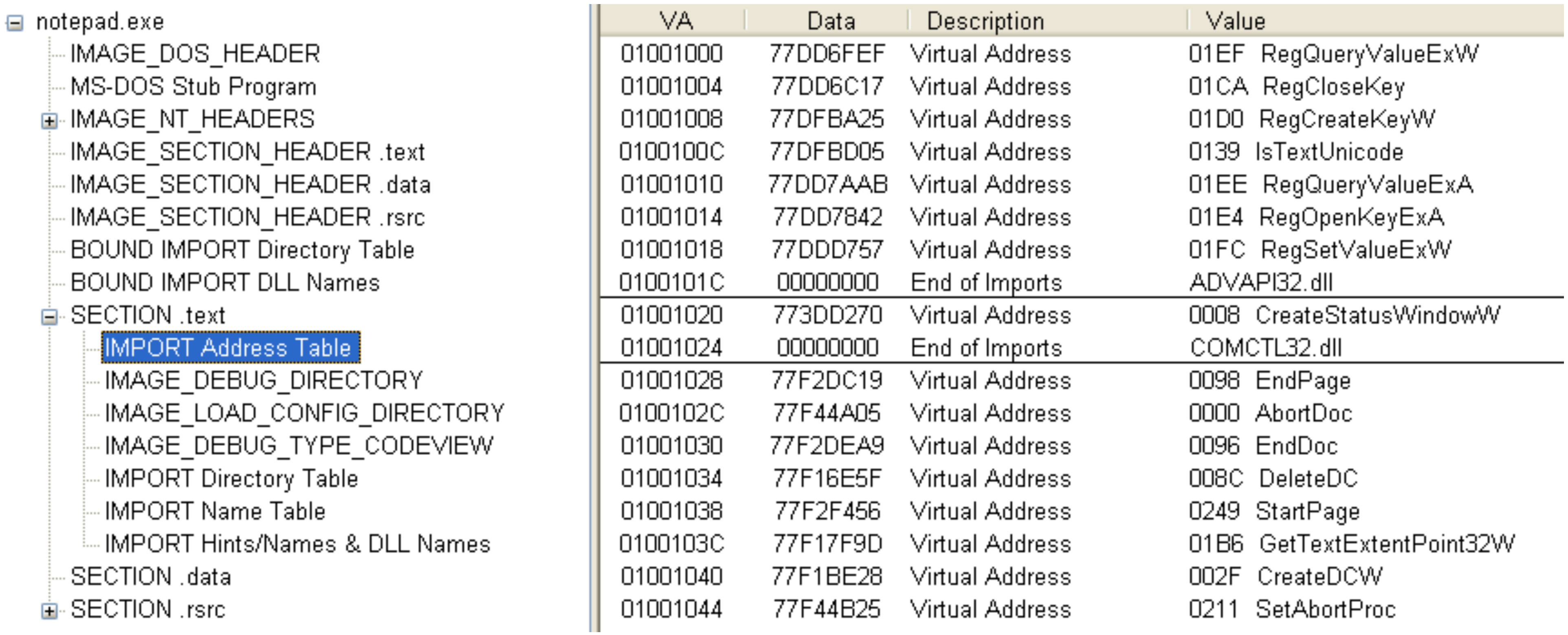
- notepad.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
    - Signature
    - IMAGE_FILE_HEADER
    - IMAGE_OPTIONAL_HEADER
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - BOUND IMPORT Directory Table
  - BOUND IMPORT DLL Names
  - SECTION .text
  - SECTION .data
  - SECTION .rsrc

VA	Data	Description	Value
01000250	4802A0C9	Time Date Stamp	2008/04/14 Mon 00:09:45 UTC
01000254	0058	Offset to Module Name	comdlg32.dll
01000256	0000	Number of Module Forwarder Refs	
01000258	4802A111	Time Date Stamp	2008/04/14 Mon 00:10:57 UTC
0100025C	0065	Offset to Module Name	SHELL32.dll
0100025E	0000	Number of Module Forwarder Refs	
01000260	4802A127	Time Date Stamp	2008/04/14 Mon 00:11:19 UTC
01000264	0071	Offset to Module Name	WINSPOOL.DRV
01000266	0000	Number of Module Forwarder Refs	
01000268	4802A094	Time Date Stamp	2008/04/14 Mon 00:08:52 UTC
0100026C	007E	Offset to Module Name	COMCTL32.dll
0100026E	0000	Number of Module Forwarder Refs	
01000270	4802A094	Time Date Stamp	2008/04/14 Mon 00:08:52 UTC
01000274	008B	Offset to Module Name	msvcrt.dll
01000276	0000	Number of Module Forwarder Refs	
01000278	4802A0B2	Time Date Stamp	2008/04/14 Mon 00:09:22 UTC
0100027C	0096	Offset to Module Name	ADVAPI32.dll
0100027E	0000	Number of Module Forwarder Refs	
01000280	4802A12C	Time Date Stamp	2008/04/14 Mon 00:11:24 UTC
01000284	00A3	Offset to Module Name	KERNEL32.dll
01000286	0001	Number of Module Forwarder Refs	
01000288	4802A12C	Time Date Stamp	2008/04/14 Mon 00:11:24 UTC
0100028C	00B0	Offset to Module Name	NTDLL.DLL
0100028E	0000	Reserved	
01000290	4802A0BE	Time Date Stamp	2008/04/14 Mon 00:09:34 UTC
01000294	00BA	Offset to Module Name	GDI32.dll
01000296	0000	Number of Module Forwarder Refs	
01000298	4802A11B	Time Date Stamp	2008/04/14 Mon 00:11:07 UTC
0100029C	00C4	Offset to Module Name	USER32.dll
0100029E	0000	Number of Module Forwarder Refs	
010002A0	00000000		
010002A4	0000		
010002A6	0000		

Non-zero number of forwarder refs

Therefore this ntdll entry is a
IMAGE_BOUND_FORWARDER_REF
Not a
IMAGE_BOUND_IMPORT_DESCRIPTOR
… I didn’t notice it at first :)

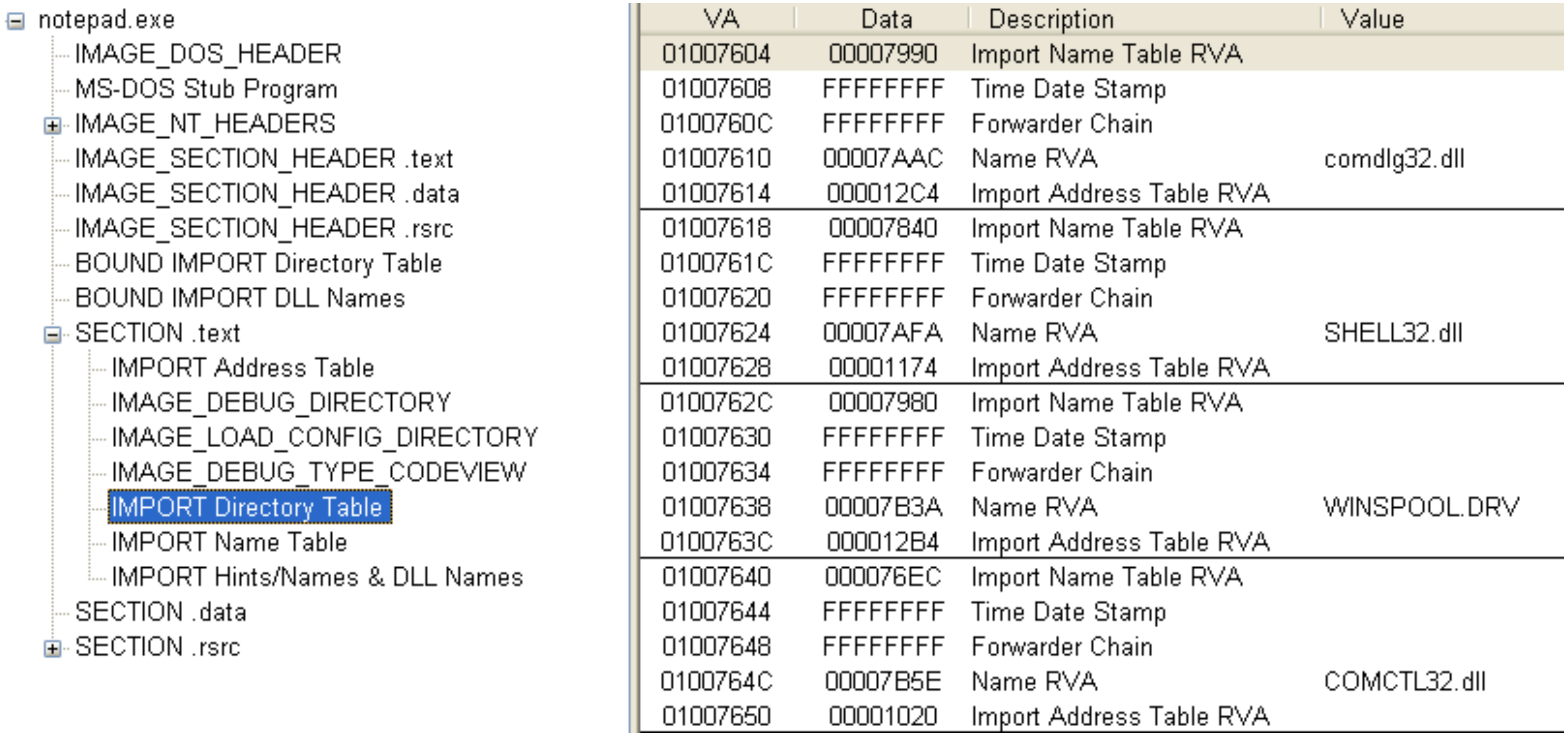
Notepad.exe’s IAT with bound imports



- notepad.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - BOUND IMPORT Directory Table
  - BOUND IMPORT DLL Names
  - SECTION .text
    - IMPORT Address Table
    - IMAGE_DEBUG_DIRECTORY
    - IMAGE_LOAD_CONFIG_DIRECTORY
    - IMAGE_DEBUG_TYPE_CODEVIEW
    - IMPORT Directory Table
    - IMPORT Name Table
    - IMPORT Hints/Names & DLL Names
  - SECTION .data
  - SECTION .rsrc

VA	Data	Description	Value
01001000	77DD6FEF	Virtual Address	01EF RegQueryValueExW
01001004	77DD6C17	Virtual Address	01CA RegCloseKey
01001008	77DFBA25	Virtual Address	01D0 RegCreateKeyW
0100100C	77DFBD05	Virtual Address	0139 IsTextUnicode
01001010	77DD7AAB	Virtual Address	01EE RegQueryValueExA
01001014	77DD7842	Virtual Address	01E4 RegOpenKeyExA
01001018	77DDD757	Virtual Address	01FC RegSetValueExW
0100101C	00000000	End of Imports	ADVAPI32.dll
01001020	773DD270	Virtual Address	0008 CreateStatusWindowW
01001024	00000000	End of Imports	COMCTL32.dll
01001028	77F2DC19	Virtual Address	0098 EndPage
0100102C	77F44A05	Virtual Address	0000 AbortDoc
01001030	77F2DEA9	Virtual Address	0096 EndDoc
01001034	77F16E5F	Virtual Address	008C DeleteDC
01001038	77F2F456	Virtual Address	0249 StartPage
0100103C	77F17F9D	Virtual Address	01B6 GetTextExtentPoint32W
01001040	77F1BE28	Virtual Address	002F CreateDCW
01001044	77F44B25	Virtual Address	0211 SetAbortProc

Notepad.exe’s IMAGE_IMPORT_DESCRIPTOR array with bound imports



- notepad.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - BOUND IMPORT Directory Table
  - BOUND IMPORT DLL Names
  - SECTION .text
    - IMPORT Address Table
    - IMAGE_DEBUG_DIRECTORY
    - IMAGE_LOAD_CONFIG_DIRECTORY
    - IMAGE_DEBUG_TYPE_CODEVIEW
    - IMPORT Directory Table
    - IMPORT Name Table
    - IMPORT Hints/Names & DLL Names
  - SECTION .data
  - SECTION .rsrc

VA	Data	Description	Value
01007604	00007990	Import Name Table RVA	
01007608	FFFFFFFF	Time Date Stamp	
0100760C	FFFFFFFF	Forwarder Chain	
01007610	00007AAC	Name RVA	comdlg32.dll
01007614	000012C4	Import Address Table RVA	
01007618	00007840	Import Name Table RVA	
0100761C	FFFFFFFF	Time Date Stamp	
01007620	FFFFFFFF	Forwarder Chain	
01007624	00007AFA	Name RVA	SHELL32.dll
01007628	00001174	Import Address Table RVA	
0100762C	00007980	Import Name Table RVA	
01007630	FFFFFFFF	Time Date Stamp	
01007634	FFFFFFFF	Forwarder Chain	
01007638	00007B3A	Name RVA	WINSPOOL.DRV
0100763C	000012B4	Import Address Table RVA	
01007640	000076EC	Import Name Table RVA	
01007644	FFFFFFFF	Time Date Stamp	
01007648	FFFFFFFF	Forwarder Chain	
0100764C	00007B5E	Name RVA	COMCTL32.dll
01007650	00001020	Import Address Table RVA	

How does one go about binding imports?

- BindImageEx API, if you want to make your own program to bind your other programs (why?)
- Windows Installer “BindImage” action – ideal case, you bind at install time, so it will be correct until the next update of Windows.
- Bind.exe? Can’t find it on my dev VM (VC++ 9.0, i.e. 2008 edition) but there’s plenty of references to it in older documents (e.g. VC++ 6.0). Seems to be deprecated.
- However, we can use CFF Explorer, so let's do that to our hello world quick:
  - Open HelloWorld.exe in CFF Explorer.exe
  - Goto Data Directories [x] and note the zeros for Bound Import Directory RVA/Size.
  - Goto Import Directory and select kernel32.dll. Note the values in the FTs(IAT) column.
  - Go to "Rebuilder" helper plugin, select "Bind Import Table" only and then select "Rebuild"
  - Go back to the Data Directories to see the non-zero Bound Import Directory RVA and go to the Import Directory area to see the absolute VAs for the imported function addresses.

Binding vs. ASLR: THERE CAN BE ONLY ONE!

- Address Space Layout Randomization makes binding pointless, because if the ASLR is doing its job, the bindings should be invalidated most of the time. So you end up being forced to resolve imports at load time anyway, and therefore any time you took to try and validate bound imports was pointless, so you may as well just not even use them.
- This is why I' m pretty sure binding is (going to be?) deprecated, and why bind.exe disappeared.

http://www.elfwood.com/
~tommartin/Highlander.
3294669.html

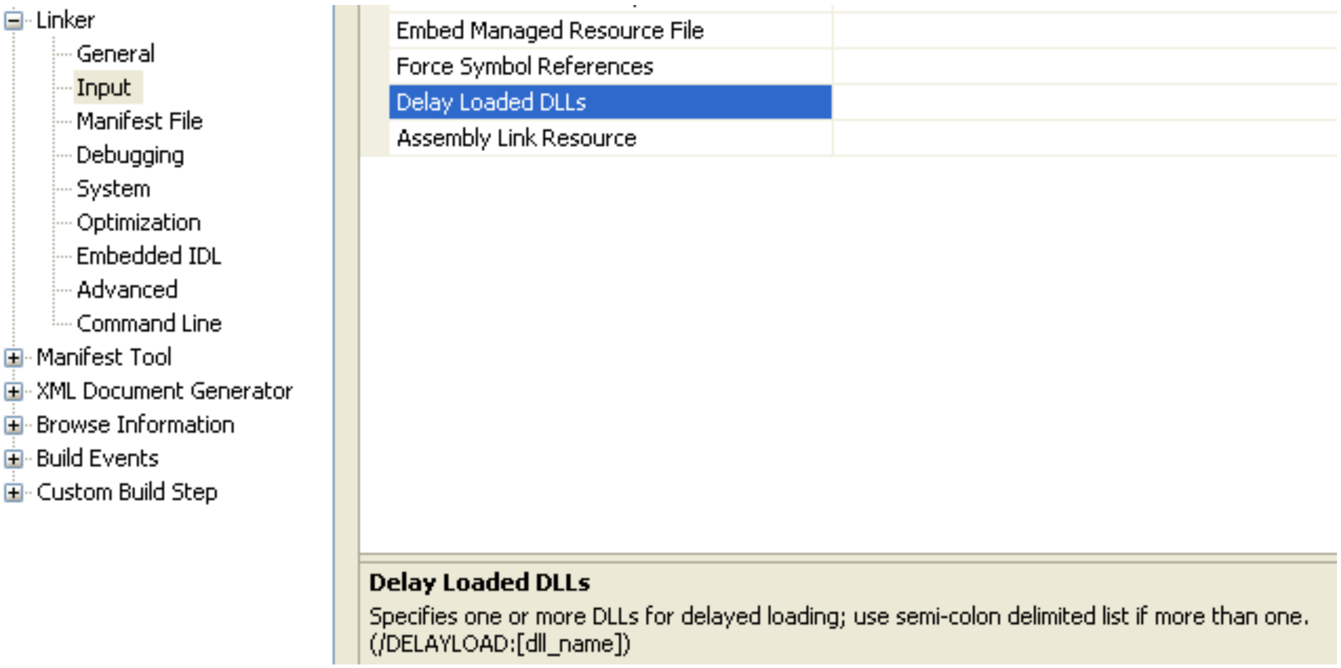
From winnt.h

```
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;
    DWORD   Win32VersionValue;
    DWORD   SizeOfImage;
    DWORD   SizeOfHeaders;
    DWORD   CheckSum;
    WORD    Subsystem;
    WORD    DllCharacteristics;
    DWORD   SizeOfStackReserve;
    DWORD   SizeOfStackCommit;
    DWORD   SizeOfHeapReserve;
    DWORD   SizeOfHeapCommit;
    DWORD   LoaderFlags;
    DWORD   NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

Which fields do we even care about, and why?

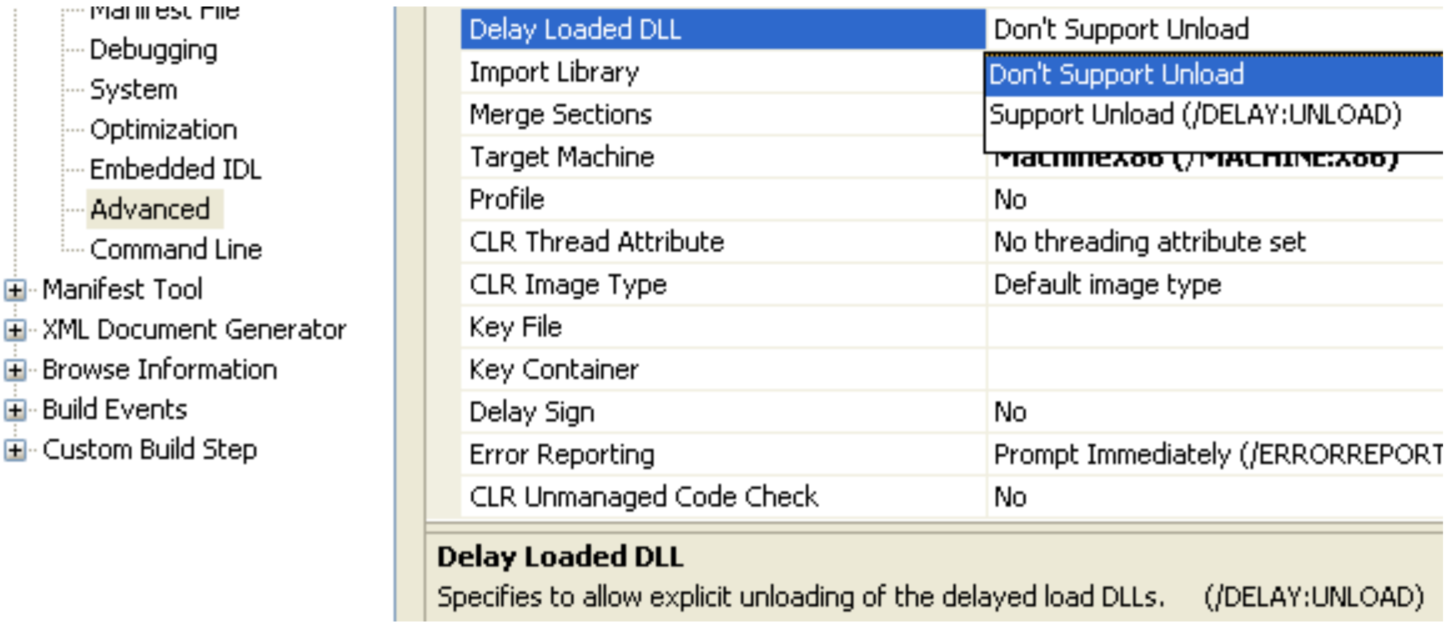
Delay Loaded DLLs

- Specifies that libraries will not even be loaded into the memory space until the first time they are used. This can potentially be a good thing to do for code
- Setting this option will generate extra information separate from normal DLL loading information to the support the delayed loading.
- Described in detail in the PE section



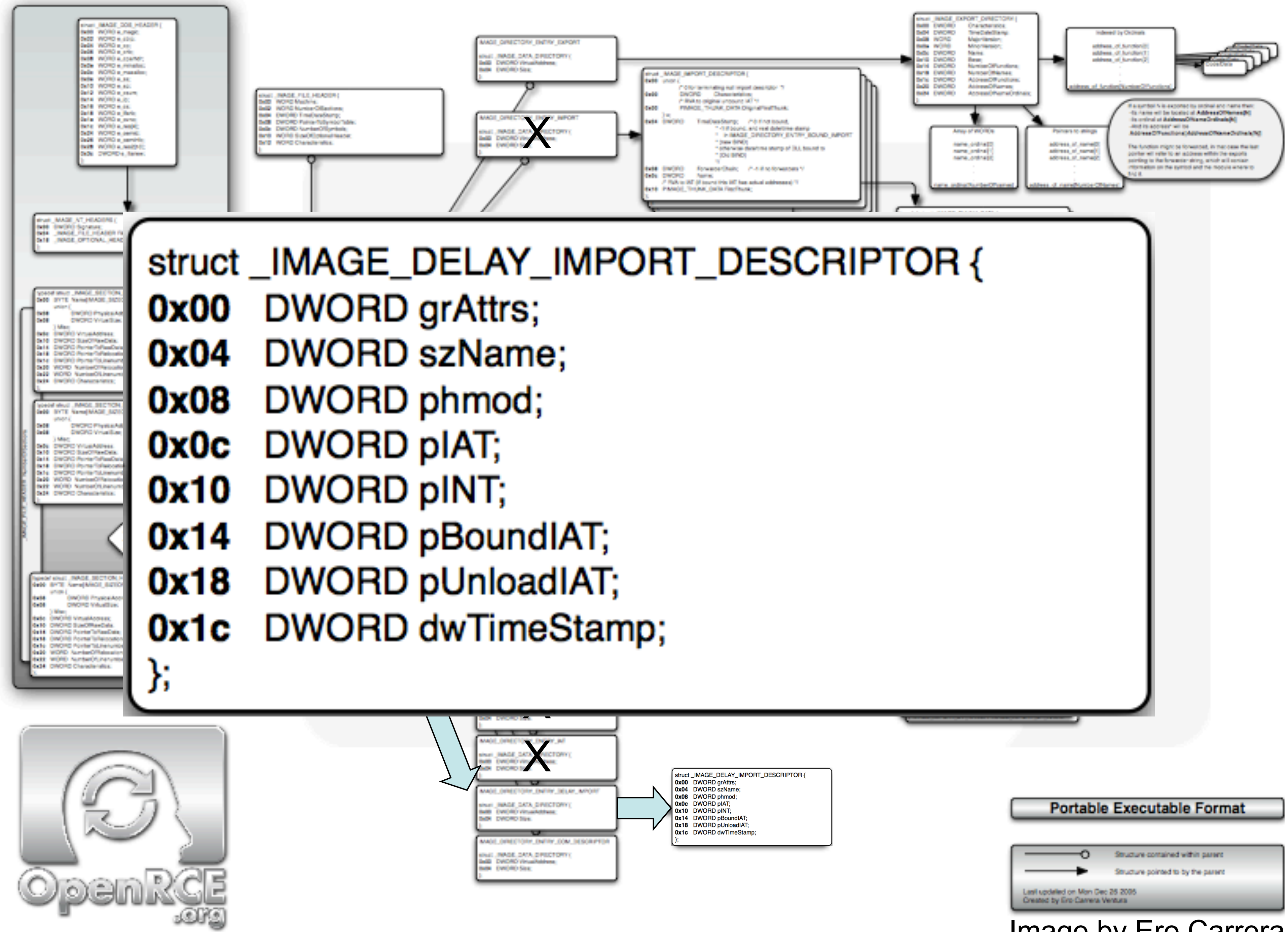
Embed Managed Resource File
Force Symbol References
Delay Loaded DLLs
Assembly Link Resource

Delay Loaded DLLs
Specifies one or more DLLs for delayed loading; use semi-colon delimited list if more than one. (/DELAYLOAD:[dll_name])



Delay Loaded DLL	Don't Support Unload
Import Library	
Merge Sections	
Target Machine	MachineX86 (/MACHINE:X86)
Profile	No
CLR Thread Attribute	No threading attribute set
CLR Image Type	Default image type
Key File	
Key Container	
Delay Sign	No
Error Reporting	Prompt Immediately (/ERRORREPORT
CLR Unmanaged Code Check	No

Delay Loaded DLL
Specifies to allow explicit unloading of the delayed load DLLs. (/DELAY:UNLOAD)


Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
X
X
OpenRCE.org

```
struct _IMAGE_DELAY_IMPORT_DESCRIPTOR {
0x00   DWORD grAttrs;
0x04   DWORD szName;
0x08   DWORD phmod;
0x0c   DWORD pIAT;
0x10   DWORD pINT;
0x14   DWORD pBoundIAT;
0x18   DWORD pUnloadIAT;
0x1c   DWORD dwTimeStamp;
};
```

Image by Ero Carrera

Delayed Imports

from DelayImp.H, dunno where he got _IMAGE_DELAY_IMPORT_DESCRIPTOR from

```
typedef struct ImgDelayDescr {
DWORD           grAttrs;        // attributes
RVA             rvaDLLName;     // RVA to dll name
RVA             rvaHmod;        // RVA of module handle
RVA             rvaIAT;         // RVA of the IAT
RVA             rvaINT;         // RVA of the INT
RVA             rvaBoundIAT;    // RVA of the optional bound IAT
RVA             rvaUnloadIAT;   // RVA of optional copy of original IAT
DWORD           dwTimeStamp;    // 0 if not bound,
                                // O.W. date/time stamp of DLL bound to (Old BIND)
} ImgDelayDescr, * PImgDelayDescr;
```

- We care about **rvaIAT** because it points at a separate IAT where stuff gets filled in as needed.
- Also **rvaDLLName** just because, you know, it tells us which DLL this is about.
- You can look up the rest on your own later (I recommend you check http://msdn.microsoft.com/en-us/magazine/cc301808.aspx), but really these fields are just there for the dynamic linker's benefit, so we don't care enough to go into any of them. The main takeaway will be about the procedure for resolving delayed imports.

The Delay-Loaded IAT

- We care about **rvaIAT** because this points to a ***separate*** IAT for delay-loaded functions only. But it's that IAT which is interesting.
- Initially the delay load IAT holds full virtual addresses of stub code. So the first time you call the delay-loaded function, it first calls the stub code.
- If necessary, the stub code loads the module which contains the function you want to call. Then it and resolves the address of the function within the module. It fills that address into the delay load IAT, and then calls the desired function. So the second time the code calls the function, it bypasses the dynamic resolution process, and just goes directly to the desired function.
- You can look up the rest on your own later (I recommend you check http://msdn.microsoft.com/en-us/magazine/cc301808.aspx), but these fields are mostly just there for the dynamic linker's benefit, so we don't care enough to go into them.

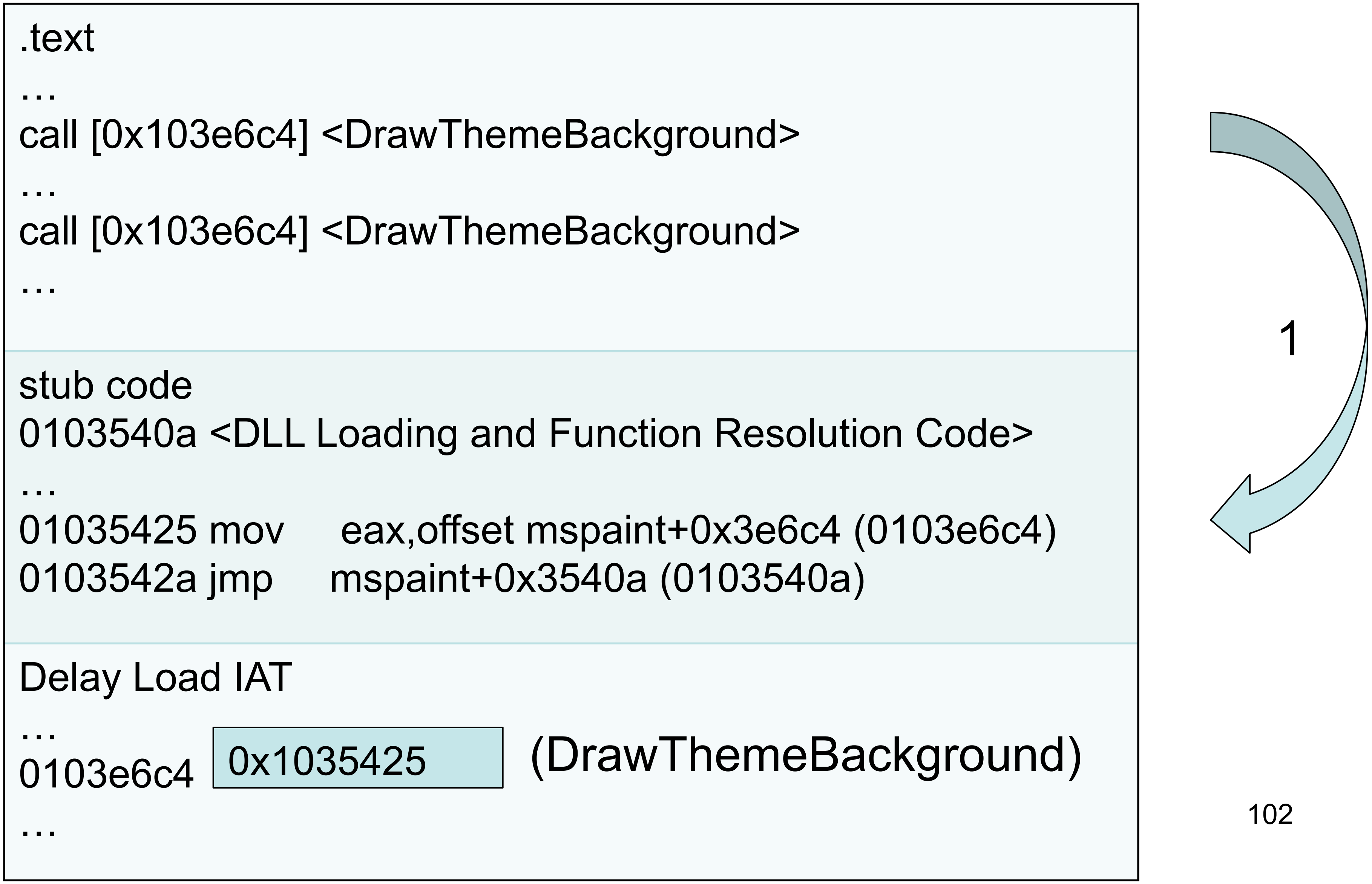
Delay Loading

hello

hi

how you doing?

fine, thanks.



```
.text
…
call [0x103e6c4] <DrawThemeBackground>
…
call [0x103e6c4] <DrawThemeBackground>
…

stub code
0103540a <DLL Loading and Function Resolution Code>
…
01035425 mov     eax,offset mspaint+0x3e6c4 (0103e6c4)
0103542a jmp     mspaint+0x3540a (0103540a)

Delay Load IAT
…
0103e6c4  0x1035425   (DrawThemeBackground)
…
```

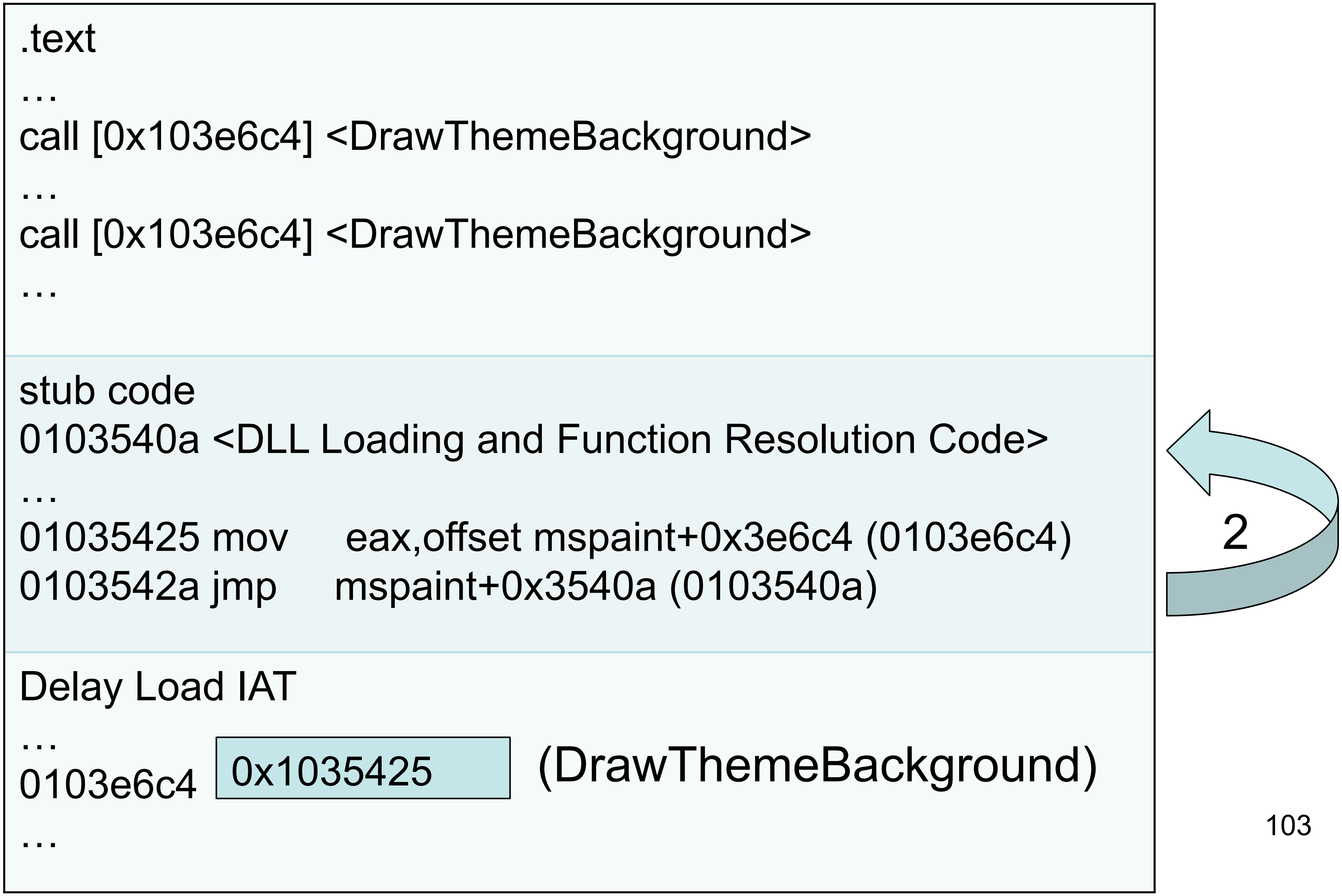
Delay Loading

hello

hi

how you doing?

fine, thanks.



.text

```
…
call [0x103e6c4] <DrawThemeBackground>
…
call [0x103e6c4] <DrawThemeBackground>
…
```

stub code

```
0103540a <DLL Loading and Function Resolution Code>
…
01035425 mov     eax,offset mspaint+0x3e6c4 (0103e6c4)
0103542a jmp     mspaint+0x3540a (0103540a)
```

Delay Load IAT

```
…
0103e6c4  0x1035425    (DrawThemeBackground)
…
```

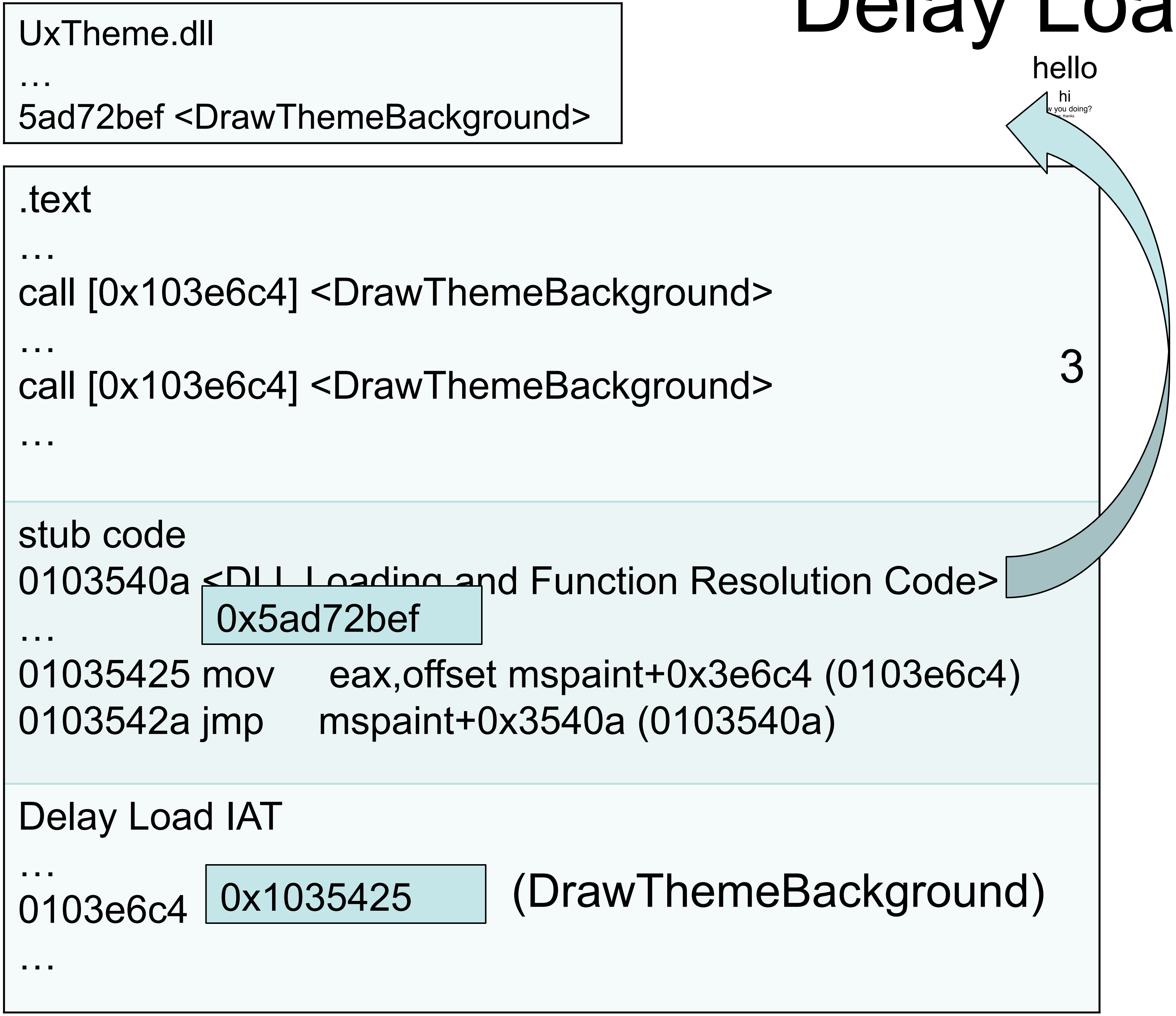
Delay Loading

hello

hi

w you doing?

fine, thanks.



UxTheme.dll
…
5ad72bef <DrawThemeBackground>

.text
…
call [0x103e6c4] <DrawThemeBackground>
…
call [0x103e6c4] <DrawThemeBackground>
…

stub code
0103540a <DLL Loading and Function Resolution Code>
0x5ad72bef
…
01035425 mov eax,offset mspaint+0x3e6c4 (0103e6c4)
0103542a jmp mspaint+0x3540a (0103540a)

Delay Load IAT
…
0103e6c4 0x1035425 (DrawThemeBackground)
…

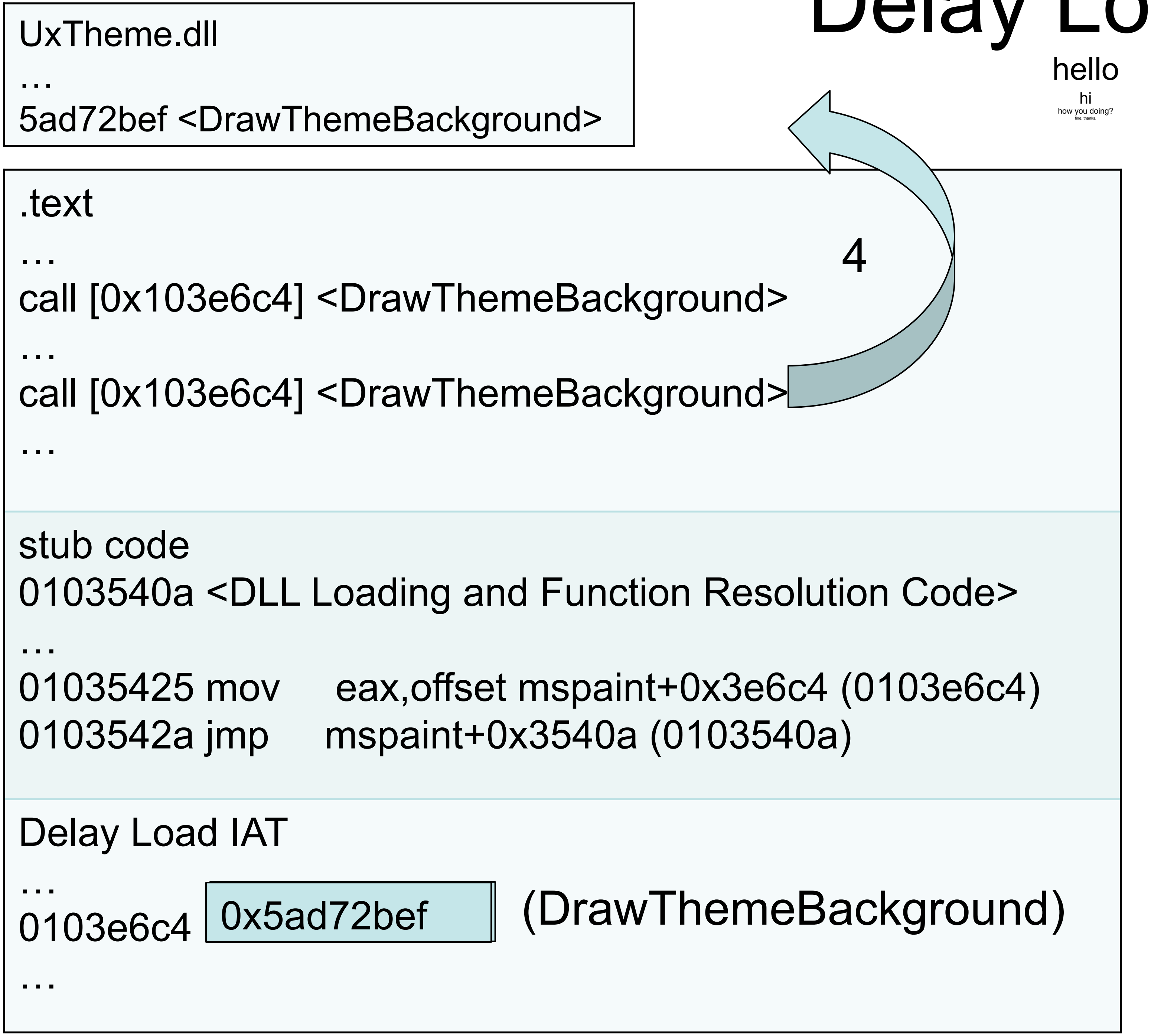
Delay Loading

hello

hi

how you doing?

fine, thanks.



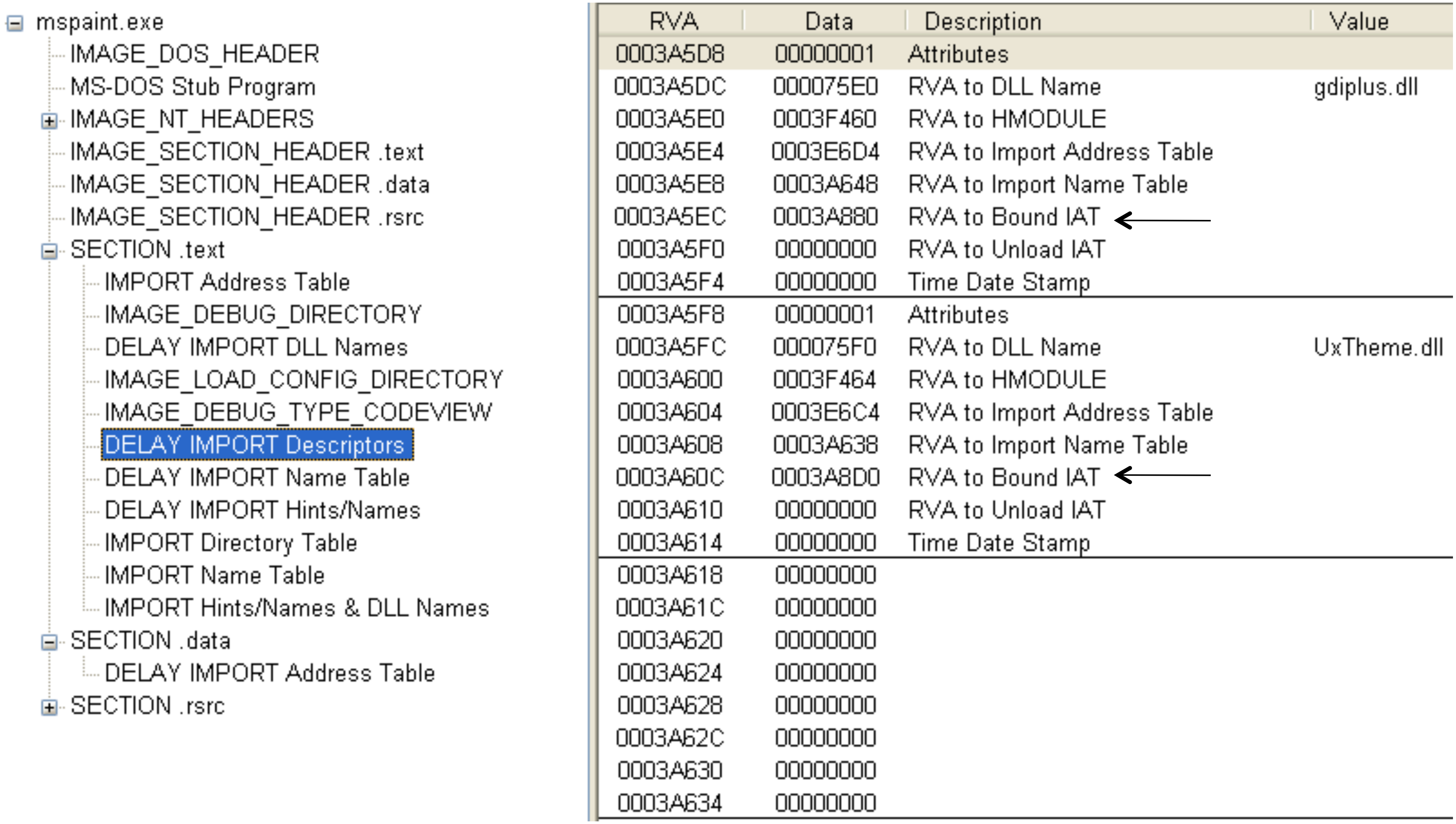
UxTheme.dll
…
5ad72bef <DrawThemeBackground>

.text
…
call [0x103e6c4] <DrawThemeBackground>
…
call [0x103e6c4] <DrawThemeBackground>
…

stub code
0103540a <DLL Loading and Function Resolution Code>
…
01035425 mov   eax,offset mspaint+0x3e6c4 (0103e6c4)
0103542a jmp   mspaint+0x3540a (0103540a)

Delay Load IAT
…
0103e6c4 0x5ad72bef (DrawThemeBackground)
…

mspaint’s delayed import descriptors

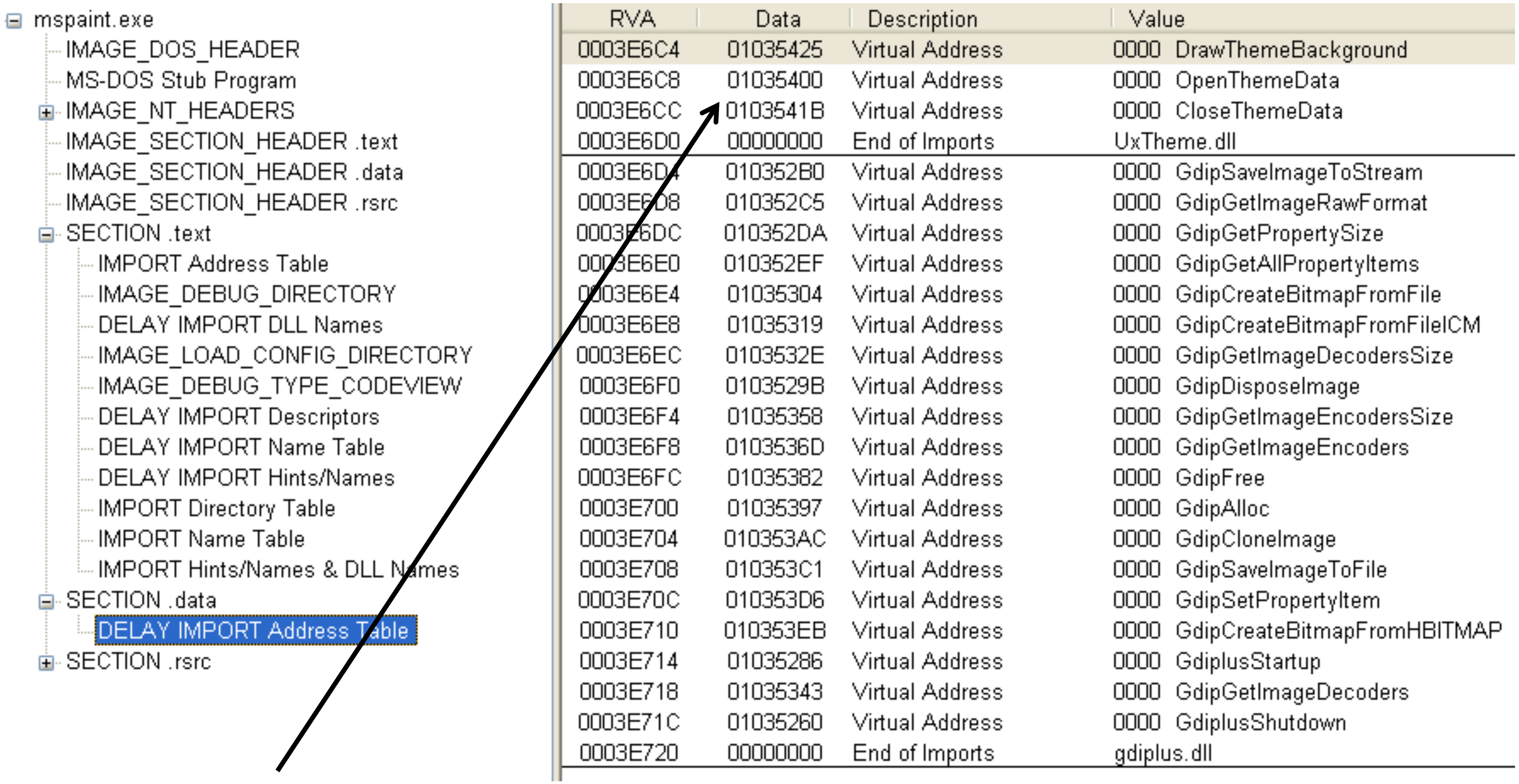


- mspaint.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - SECTION .text
    - IMPORT Address Table
    - IMAGE_DEBUG_DIRECTORY
    - DELAY IMPORT DLL Names
    - IMAGE_LOAD_CONFIG_DIRECTORY
    - IMAGE_DEBUG_TYPE_CODEVIEW
    - DELAY IMPORT Descriptors
    - DELAY IMPORT Name Table
    - DELAY IMPORT Hints/Names
    - IMPORT Directory Table
    - IMPORT Name Table
    - IMPORT Hints/Names & DLL Names
  - SECTION .data
    - DELAY IMPORT Address Table
  - SECTION .rsrc

RVA	Data	Description	Value
0003A5D8	00000001	Attributes	
0003A5DC	000075E0	RVA to DLL Name	gdiplus.dll
0003A5E0	0003F460	RVA to HMODULE	
0003A5E4	0003E6D4	RVA to Import Address Table	
0003A5E8	0003A648	RVA to Import Name Table	
0003A5EC	0003A880	RVA to Bound IAT	
0003A5F0	00000000	RVA to Unload IAT	
0003A5F4	00000000	Time Date Stamp	
0003A5F8	00000001	Attributes	
0003A5FC	000075F0	RVA to DLL Name	UxTheme.dll
0003A600	0003F464	RVA to HMODULE	
0003A604	0003E6C4	RVA to Import Address Table	
0003A608	0003A638	RVA to Import Name Table	
0003A60C	0003A8D0	RVA to Bound IAT	
0003A610	00000000	RVA to Unload IAT	
0003A614	00000000	Time Date Stamp	
0003A618	00000000		
0003A61C	00000000		
0003A620	00000000		
0003A624	00000000		
0003A628	00000000		
0003A62C	00000000		
0003A630	00000000		
0003A634	00000000		

Although the “RVA to Bound IAT” is filled in, this feature was reserved for a future version of bind, but I don’t think it ever got implemented before deprecation so it just points at some nulls.

mspaint’ s delayed IAT



- mspaint.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - SECTION .text
    - IMPORT Address Table
    - IMAGE_DEBUG_DIRECTORY
    - DELAY IMPORT DLL Names
    - IMAGE_LOAD_CONFIG_DIRECTORY
    - IMAGE_DEBUG_TYPE_CODEVIEW
    - DELAY IMPORT Descriptors
    - DELAY IMPORT Name Table
    - DELAY IMPORT Hints/Names
    - IMPORT Directory Table
    - IMPORT Name Table
    - IMPORT Hints/Names & DLL Names
  - SECTION .data
    - DELAY IMPORT Address Table
  - SECTION .rsrc

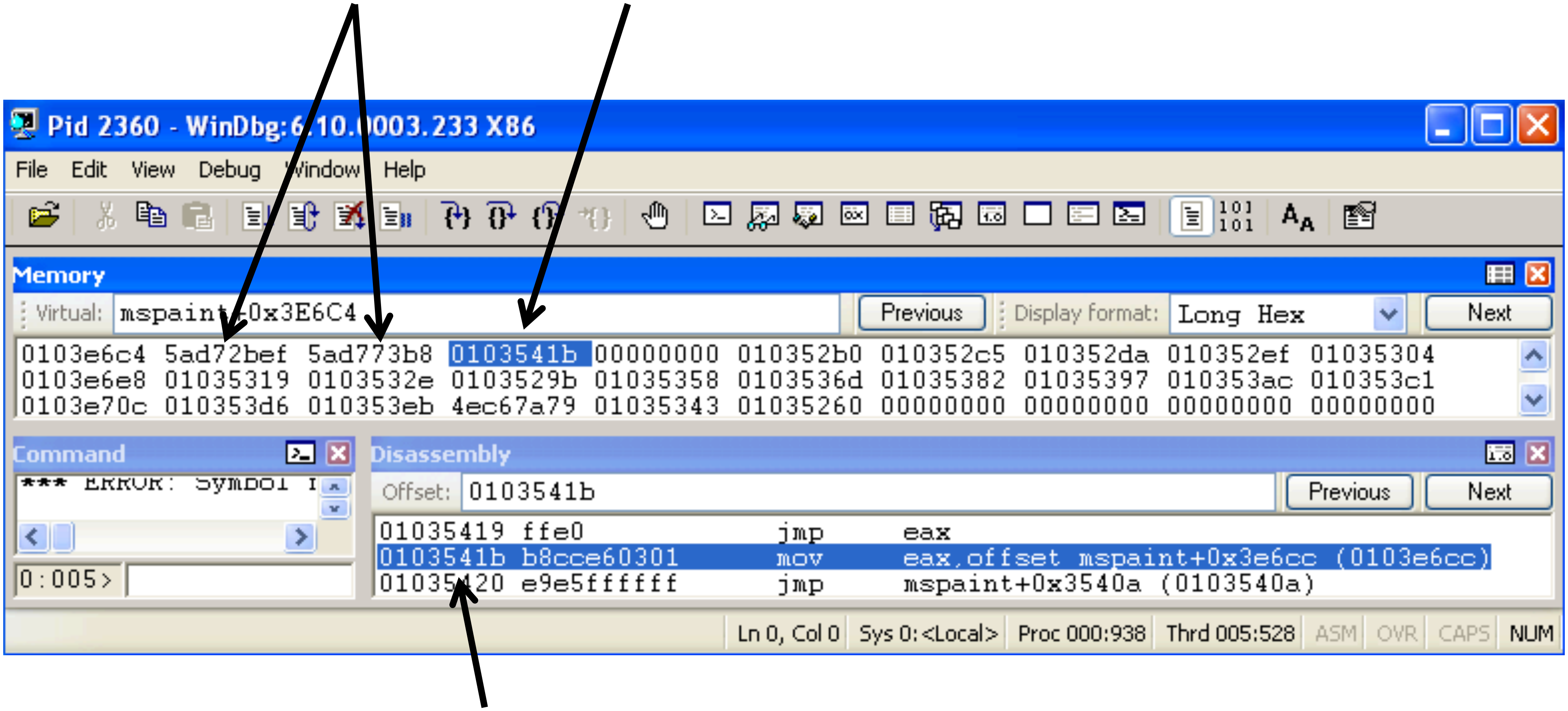
RVA	Data	Description	Value
0003E6C4	01035425	Virtual Address	0000 DrawThemeBackground
0003E6C8	01035400	Virtual Address	0000 OpenThemeData
0003E6CC	0103541B	Virtual Address	0000 CloseThemeData
0003E6D0	00000000	End of Imports	UxTheme.dll
0003E6D4	010352B0	Virtual Address	0000 GdipSaveImageToStream
0003E6D8	010352C5	Virtual Address	0000 GdipGetImageRawFormat
0003E6DC	010352DA	Virtual Address	0000 GdipGetPropertySize
0003E6E0	010352EF	Virtual Address	0000 GdipGetAllPropertyItems
0003E6E4	01035304	Virtual Address	0000 GdipCreateBitmapFromFile
0003E6E8	01035319	Virtual Address	0000 GdipCreateBitmapFromFileICM
0003E6EC	0103532E	Virtual Address	0000 GdipGetImageDecodersSize
0003E6F0	0103529B	Virtual Address	0000 GdipDisposeImage
0003E6F4	01035358	Virtual Address	0000 GdipGetImageEncodersSize
0003E6F8	0103536D	Virtual Address	0000 GdipGetImageEncoders
0003E6FC	01035382	Virtual Address	0000 GdipFree
0003E700	01035397	Virtual Address	0000 GdipAlloc
0003E704	010353AC	Virtual Address	0000 GdipCloneImage
0003E708	010353C1	Virtual Address	0000 GdipSaveImageToFile
0003E70C	010353D6	Virtual Address	0000 GdipSetPropertyItem
0003E710	010353EB	Virtual Address	0000 GdipCreateBitmapFromHBITMAP
0003E714	01035286	Virtual Address	0000 GdiplusStartup
0003E718	01035343	Virtual Address	0000 GdipGetImageDecoders
0003E71C	01035260	Virtual Address	0000 GdiplusShutdown
0003E720	00000000	End of Imports	gdiplus.dll

These are virtual addresses. Since the ImageBase for mspaint is 0x1000000 and the SizeOfImage is 0x57000, that means these virtual addresses start out **inside** mspaint itself. Each one just points at some stub code to call the dynamic linker.

mspaint’s delayed imports in memory (some resolved, some not)

Resolved

Not Resolved



Virtual: mspaint+0x3E6C4

```
0103e6c4  5ad72bef  5ad773b8  0103541b  00000000  010352b0  010352c5  010352da  010352ef  01035304
0103e6e8  01035319  0103532e  0103529b  01035358  0103536d  01035382  01035397  010353ac  010353c1
0103e70c  010353d6  010353eb  4ec67a79  01035343  01035260  00000000  00000000  00000000  00000000
```

```
*** ERROR: Symbol f
0:005>
```

Offset: 0103541b

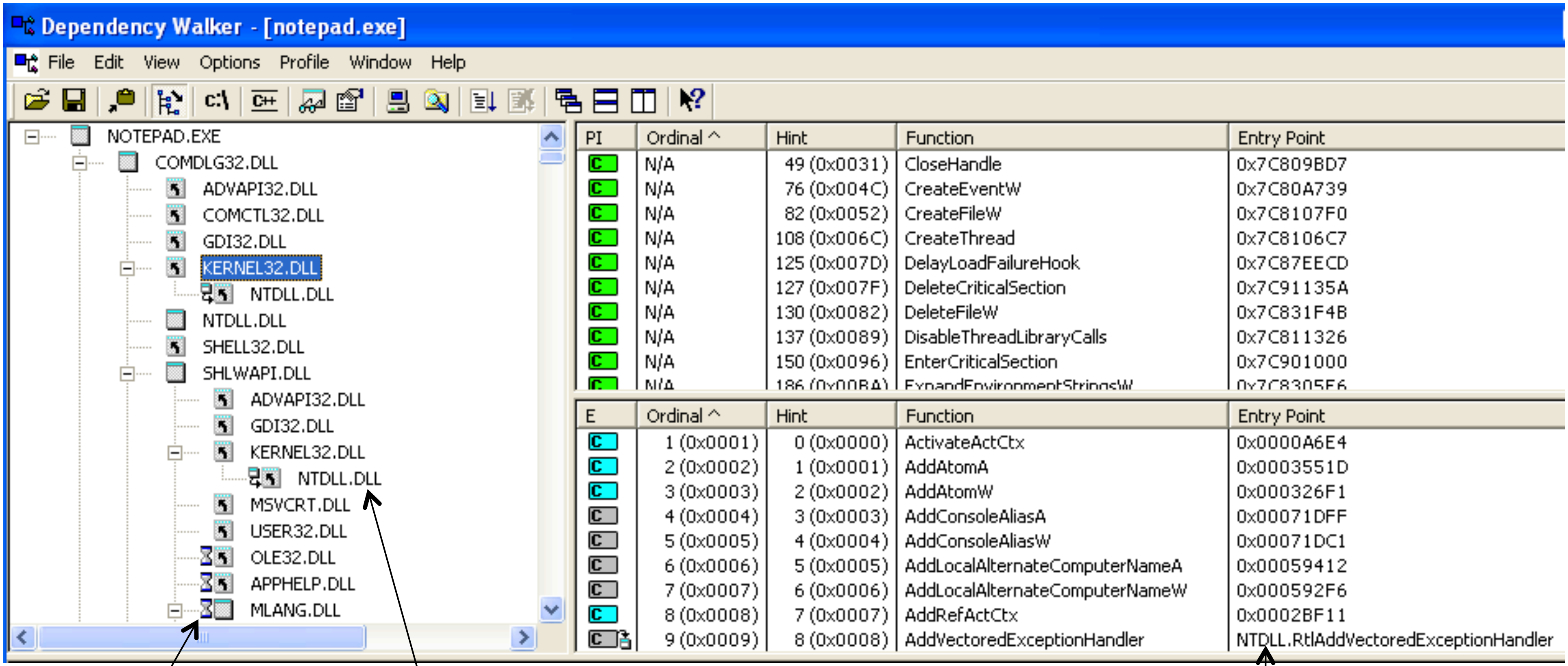
```
01035419 ffe0            jmp     eax
0103541b b8cce60301      mov     eax,offset mspaint+0x3e6cc (0103e6cc)
01035420 e9e5ffffff      jmp     mspaint+0x3540a (0103540a)
```

Start of stub code

Note to self, walk the stub code a bit in the debugger

Dependency Walker, just ‘cause

hehe depends.exe…that’s right, potty humor, I went there



```
NOTEPAD.EXE
    COMDLG32.DLL
        ADVAPI32.DLL
        COMCTL32.DLL
        GDI32.DLL
        KERNEL32.DLL
            NTDLL.DLL
        NTDLL.DLL
        SHELL32.DLL
        SHLWAPI.DLL
            ADVAPI32.DLL
            GDI32.DLL
            KERNEL32.DLL
                NTDLL.DLL
            MSVCRT.DLL
            USER32.DLL
            OLE32.DLL
            APPHELP.DLL
            MLANG.DLL
```

PI	Ordinal ^	Hint	Function	Entry Point
C	N/A	49 (0x0031)	CloseHandle	0x7C809BD7
C	N/A	76 (0x004C)	CreateEventW	0x7C80A739
C	N/A	82 (0x0052)	CreateFileW	0x7C8107F0
C	N/A	108 (0x006C)	CreateThread	0x7C8106C7
C	N/A	125 (0x007D)	DelayLoadFailureHook	0x7C87EECD
C	N/A	127 (0x007F)	DeleteCriticalSection	0x7C91135A
C	N/A	130 (0x0082)	DeleteFileW	0x7C831F4B
C	N/A	137 (0x0089)	DisableThreadLibraryCalls	0x7C811326
C	N/A	150 (0x0096)	EnterCriticalSection	0x7C901000
C	N/A	186 (0x00BA)	ExpandEnvironmentStringsW	0x7C8305F6

E	Ordinal ^	Hint	Function	Entry Point
C	1 (0x0001)	0 (0x0000)	ActivateActCtx	0x0000A6E4
C	2 (0x0002)	1 (0x0001)	AddAtomA	0x0003551D
C	3 (0x0003)	2 (0x0002)	AddAtomW	0x000326F1
C	4 (0x0004)	3 (0x0003)	AddConsoleAliasA	0x00071DFF
C	5 (0x0005)	4 (0x0004)	AddConsoleAliasW	0x00071DC1
C	6 (0x0006)	5 (0x0005)	AddLocalAlternateComputerNameA	0x00059412
C	7 (0x0007)	6 (0x0006)	AddLocalAlternateComputerNameW	0x000592F6
C	8 (0x0008)	7 (0x0007)	AddRefActCtx	0x0002BF11
C	9 (0x0009)	8 (0x0008)	AddVectoredExceptionHandler	NTDLL.RtlAddVectoredExceptionHandler

Delay load

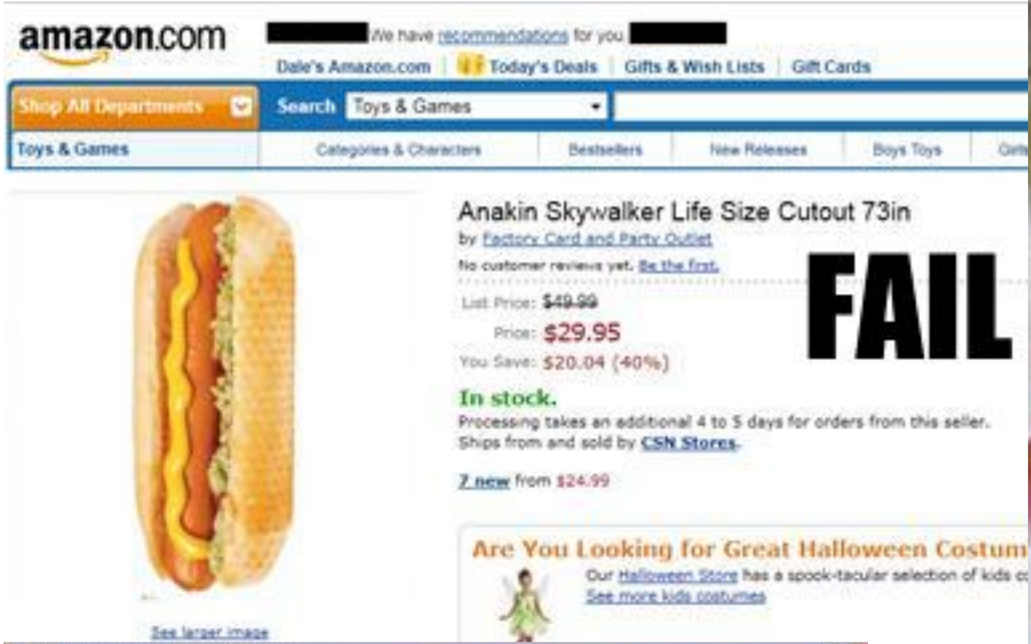
Forwarded-to DLL

Forwarded-to Function

Runtime Importing

- Just for completeness, I should mention LoadLibrary() and GetProcAddress().
- LoadLibrary() can be called to dynamically load a DLL into the memory space of the process
- GetProcAddress() gives the address of a function specified by name, or by ordinal (which we will talk about soon). This address can then be used as a function pointer.
- Remember when we were seeing delay-loaded DLLs, and the dynamic linker "somehow" loaded the DLL and then resolved the function address? It's actually using LoadLibrary() and GetProcAddress().
- These functions are often abused to make it so that which functions the malware actually uses cannot be determined simply by looking at the INT. Rather, the malware will have the names of the imported libraries and functions obfuscated somewhere in the data, and then will deobfuscate them and dynamically resolve them before calling the imported functions.

Uhg, *finally* done with imports. Treat yourself to some fail.



amazon.com [redacted] We have recommendations for you [redacted]

Dale's Amazon.com | Today's Deals | Gifts & Wish Lists | Gift Cards

Shop All Departments | Search Toys & Games

Toys & Games | Categories & Characters | Bestsellers | New Releases | Boys Toys

Anakin Skywalker Life Size Cutout 73in

by Factory Card and Party Outlet

No customer reviews yet. Be the first.

List Price: $49.99

Price: $29.95

You Save: $20.04 (40%)

In stock.

Processing takes an additional 4 to 5 days for orders from this seller.

Ships from and sold by CSN Stores.

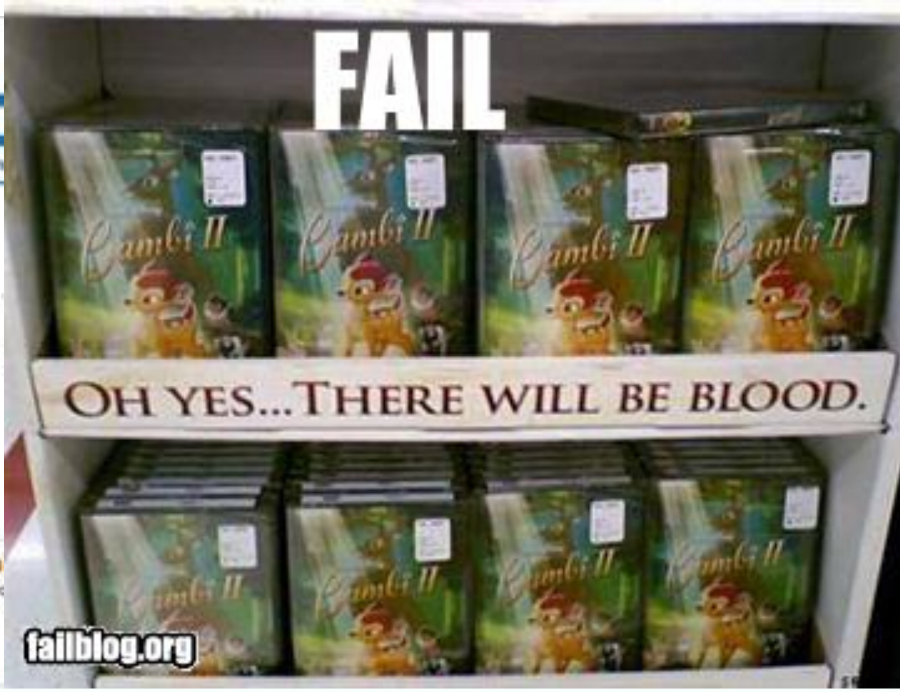
7 new from $24.99

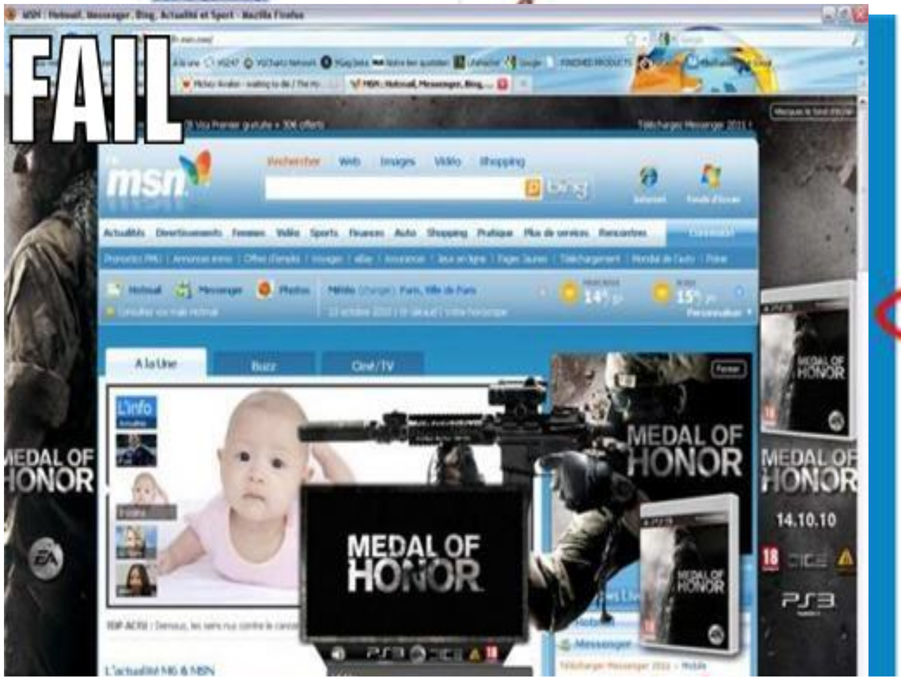
Are You Looking for Great Halloween Costum

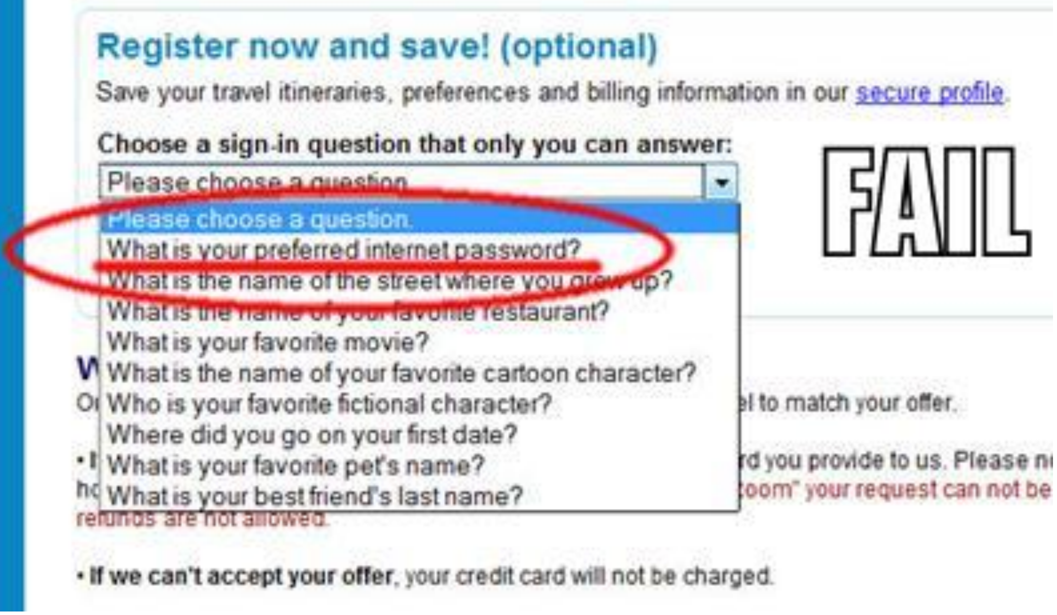
Our Halloween Store has a spook-tacular selection of kids c

See more kids costumes

See larger image







Register now and save! (optional)

Save your travel itineraries, preferences and billing information in our secure profile.

Choose a sign-in question that only you can answer:

Please choose a question

- Please choose a question.
- What is your preferred internet password?
- What is the name of the street where you grew up?
- What is the name of your favorite restaurant?
- What is your favorite movie?
- What is the name of your favorite cartoon character?
- Who is your favorite fictional character?
- Where did you go on your first date?
- What is your favorite pet's name?
- What is your best friend's last name?

l to match your offer.

rd you provide to us. Please n

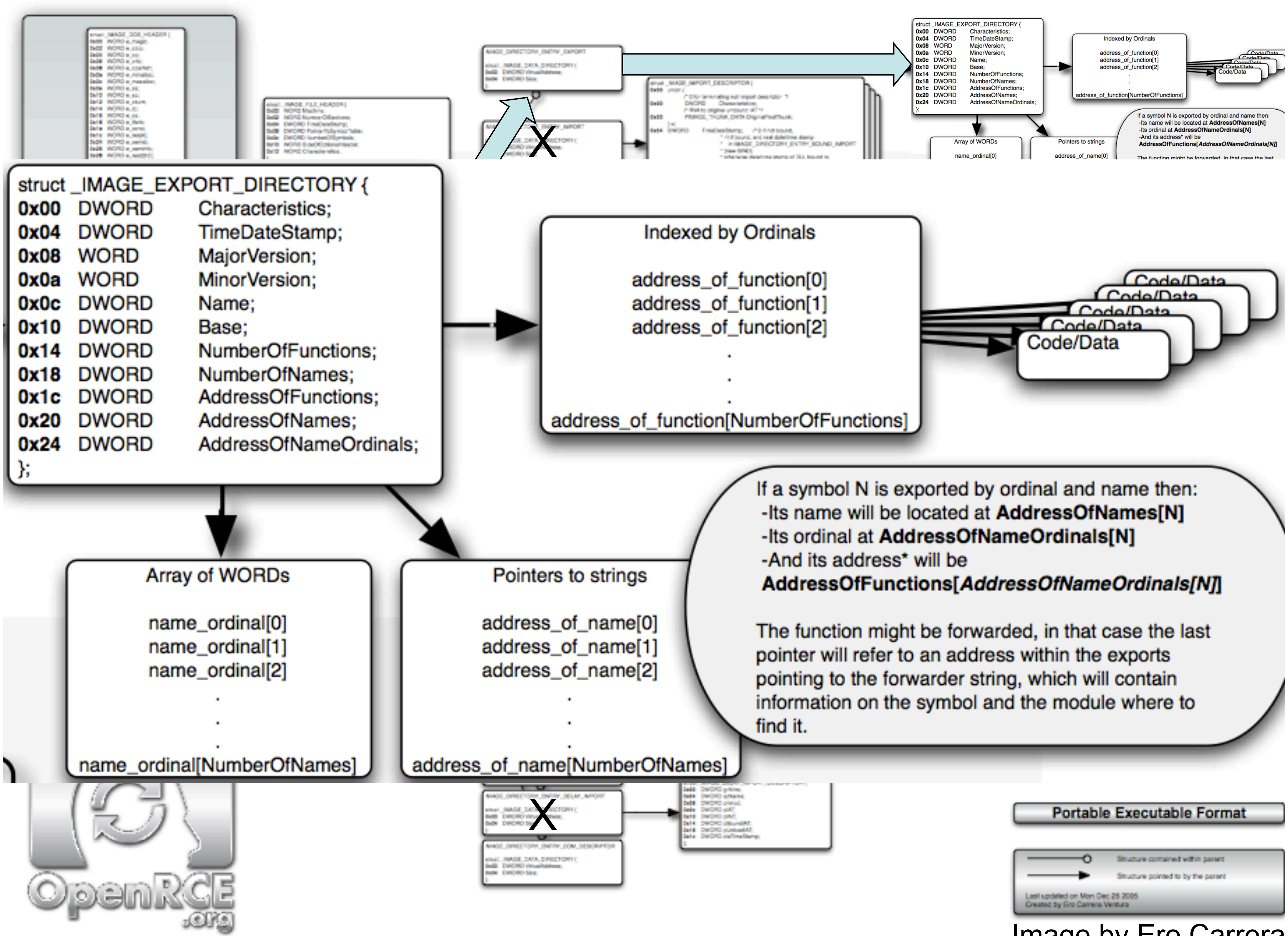
oom" your request can not be

refunds are not allowed.

• **If we can't accept your offer**, your credit card will not be charged.

Exporting Functions & Data

- For a library to be useful, other code which wants to use its functions must be able to import them, as already talked about.
- There are two options to export functions and data. They can be exported by name (where the programmer even has the option to call the exported name something different than he himself calls it), or they can be exported by ordinal.
- An ordinal is just an index, and if a function is exported by ordinal, it can only be imported by ordinal. While exporting by ordinal saves space, by not having extra strings for the names of symbols, and time by not having to search the strings, it also puts more work on the programmer which wants to import the export. But it can also be a way to make a private (undocumented) API more private.



```
struct _IMAGE_EXPORT_DIRECTORY {
0x00  DWORD     Characteristics;
0x04  DWORD     TimeDateStamp;
0x08  WORD      MajorVersion;
0x0a  WORD      MinorVersion;
0x0c  DWORD     Name;
0x10  DWORD     Base;
0x14  DWORD     NumberOfFunctions;
0x18  DWORD     NumberOfNames;
0x1c  DWORD     AddressOfFunctions;
0x20  DWORD     AddressOfNames;
0x24  DWORD     AddressOfNameOrdinals;
};
```

Indexed by Ordinals

address_of_function[0]
address_of_function[1]
address_of_function[2]
.
.
.
address_of_function[NumberOfFunctions]

Array of WORDs

name_ordinal[0]
name_ordinal[1]
name_ordinal[2]
.
.
.
name_ordinal[NumberOfNames]

Pointers to strings

address_of_name[0]
address_of_name[1]
address_of_name[2]
.
.
.
address_of_name[NumberOfNames]

If a symbol N is exported by ordinal and name then:
-Its name will be located at **AddressOfNames[N]**
-Its ordinal at **AddressOfNameOrdinals[N]**
-And its address* will be
AddressOfFunctions[*AddressOfNameOrdinals[N]*]

The function might be forwarded, in that case the last pointer will refer to an address within the exports pointing to the forwarder string, which will contain information on the symbol and the module where to find it.

Image by Ero Carrera

Exports

from winnt.h

```
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;
```

Exports 2

- The **TimeDateStamp** listed here is what's actually checked against when the loader is trying to determine if bound imports are out of date for instance. Can be different from the one in the File Header (see ntdll.dll). Presumably (wasn't able to confirm), the linker only updates this if there are meaningful changes to the RVAs or order for exported functions. That way, the TimeDateStamp "version" can stay backwards compatible as long as possible.
- **NumberOfFunctions** could theoretically be different from **NumberOfNames**, but in practice they should be the same. By knowing the number of names, when searching for an import by name, the loader can do a binary search.

Exports 3

- **Base** is the number to subtract from an ordinal to get the zero-indexed offset into the AddressOfFunctions array. Because ordinals start at 1 by default, this is usually 1. However ordinals could start at 10 if the programmer wants them to, and therefore Base would then be set to 10.
- **AddressOfFunctions** is an RVA which points to the beginning of an array which holds DWORD RVAs which point to the start of the exported functions. The pointed-to array should be NumberOfFunctions entries long. This would be the Export Address Table (EAT) like the flip side of the Import Address Table (IAT).
- Eat! I atè! :P

Exports 4

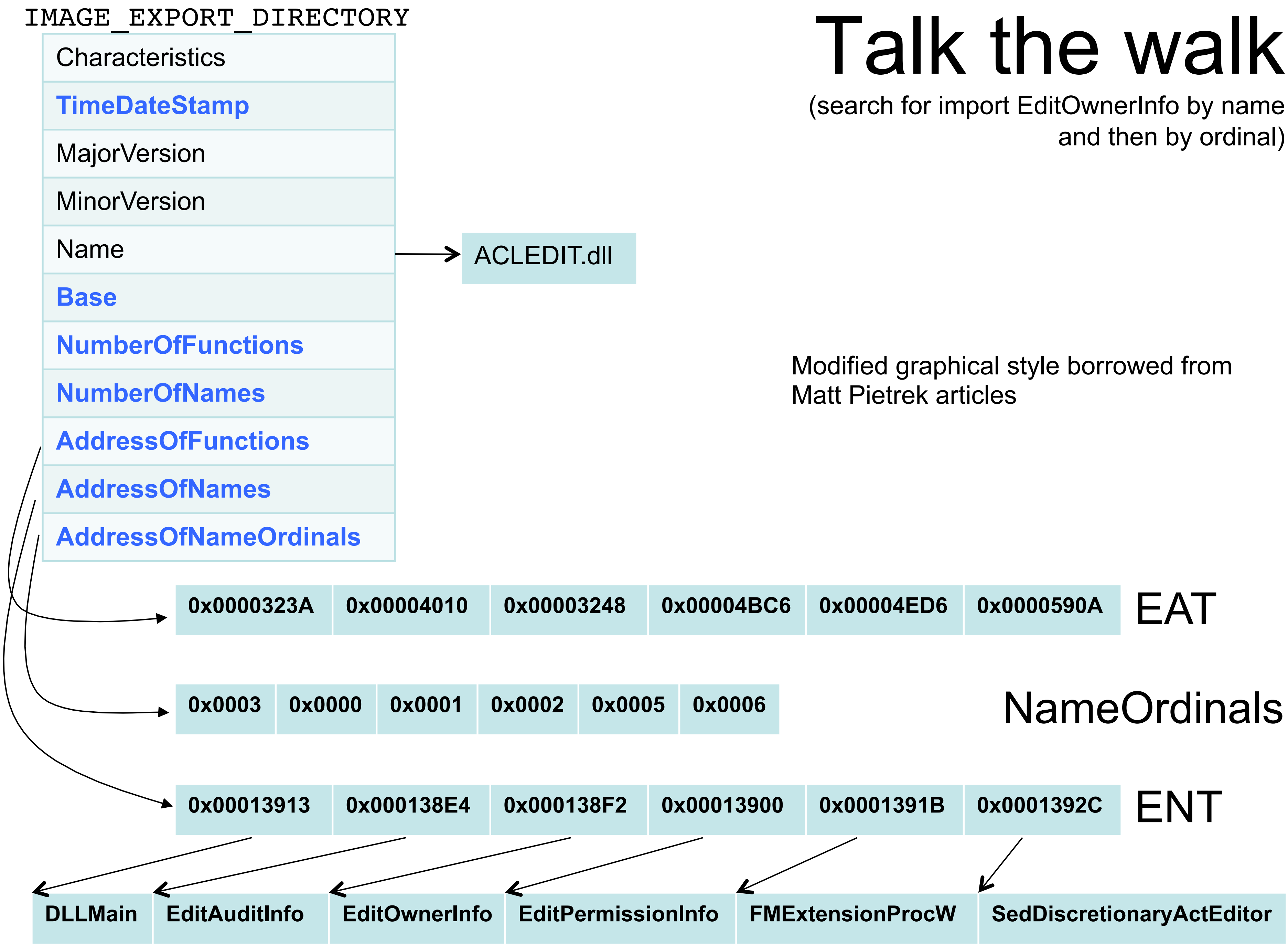
- **AddressOfNames** is an RVA which points to the beginning of an array which holds DWORD RVAs which point to the strings which specify function names. The pointed-to array should be NumberOfNames entries long. This would be the Export Names Table (ENT) like the flipside of the Import Names Table (INT).
- **AddressOfNameOrdinals** is an RVA which points to the beginning of an array which holds **WORD** (16 bit) sized ordinals. The entries in this array are already zero-indexed indices into the EAT, and therefore are unaffected by **Base**.

Ordinal says what?

- When importing by name, like I said, it can do a binary search over the strings in the ENT, because nowadays, they’re lexically sorted. “Back in the day” they weren’t sorted. Back then, it was strongly encouraged to “import by ordinal”, that is, you could specify “I want ordinal 5 in kernel32.dll” instead of “I want AddConsoleAliasW in kernel32.dll”, because if the names aren’t sorted, you’re doing a linear search. You can still import by ordinal if you choose, and that way your binary/library will load a bit faster.
- Even if you’re importing by name, it is actually just finding the index in the ENT, and then selecting the same index in the AddressOfNameOrdinals, and then reading the value from the AddressOfNameOrdinals to use as an index into the EAT.
- Generally speaking, the downside of importing by ordinal is that if the ordinals change, your app breaks. That said, the developer who’s exporting by ordinal has incentive to not change them, unless he wants those apps to break (e.g. to force a deprecated API to not be used any more).

Talk the walk

(search for import EditOwnerInfo by name
and then by ordinal)



`IMAGE_EXPORT_DIRECTORY`

Characteristics
TimeDateStamp
MajorVersion
MinorVersion
Name
Base
NumberOfFunctions
NumberOfNames
AddressOfFunctions
AddressOfNames
AddressOfNameOrdinals

ACLEDIT.dll

Modified graphical style borrowed from
Matt Pietrek articles

0x0000323A	**0x00004010**	**0x00003248**	**0x00004BC6**	**0x00004ED6**	**0x0000590A**	EAT

0x0003	**0x0000**	**0x0001**	**0x0002**	**0x0005**	**0x0006**	NameOrdinals

0x00013913	**0x000138E4**	**0x000138F2**	**0x00013900**	**0x0001391B**	**0x0001392C**	ENT

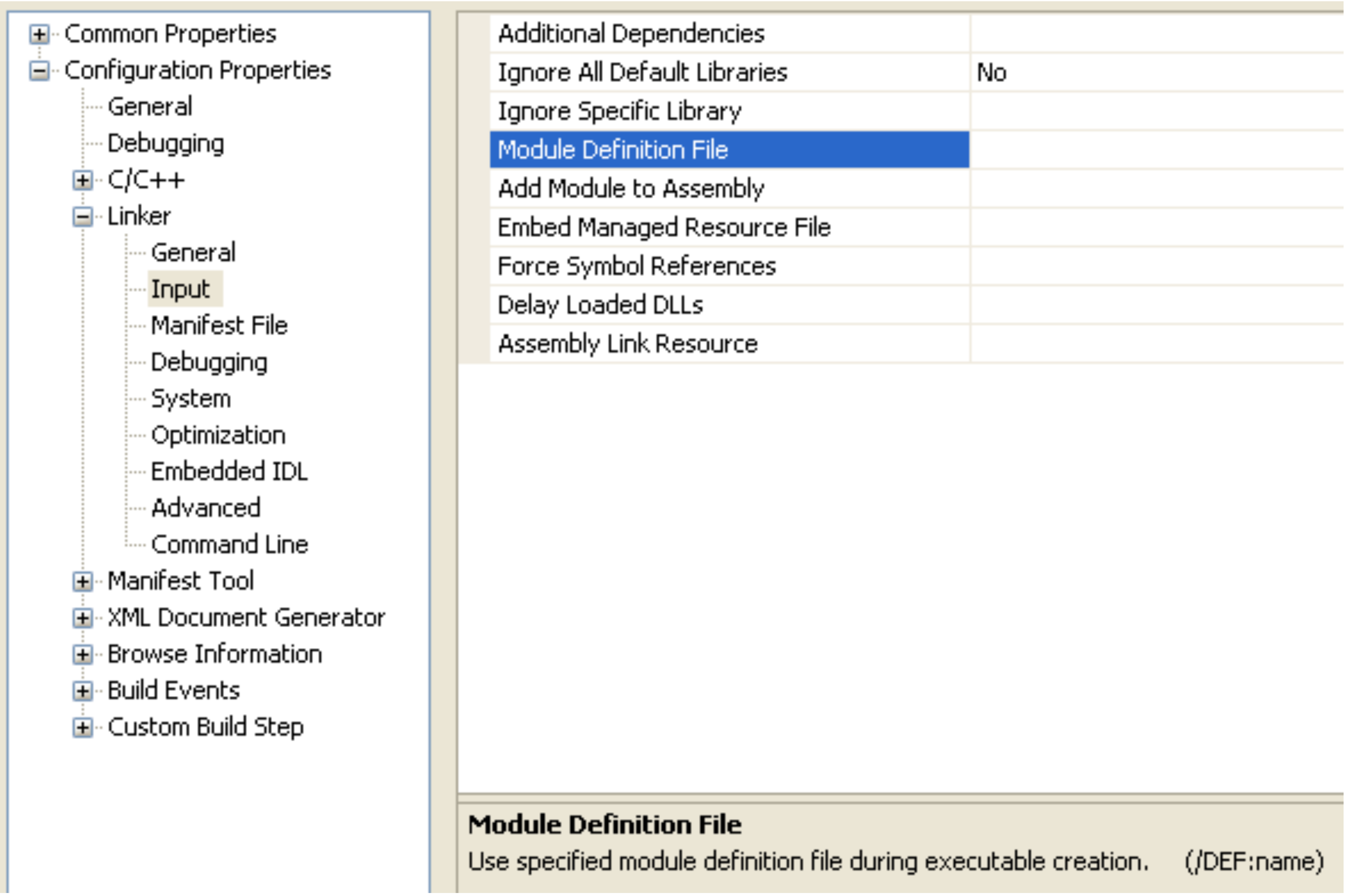
DLLMain	**EditAuditInfo**	**EditOwnerInfo**	**EditPermissionInfo**	**FMExtensionProcW**	**SedDiscretionaryActEditor**

(note the lexical order, note to self, talk about lexical ordering necessitating the ordinal table)

How does one go about specifying an export?

- http://msdn.microsoft.com/en-us/library/hyx1zcd3(VS.80).aspx
- “There are three methods for exporting a definition, listed in recommended order of use:
  - The __declspec(dllexport) keyword in the source code
  - An EXPORTS statement in a .def file
  - An /EXPORT specification in a LINK command”

Where to specify a .def file



Additional Dependencies	
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

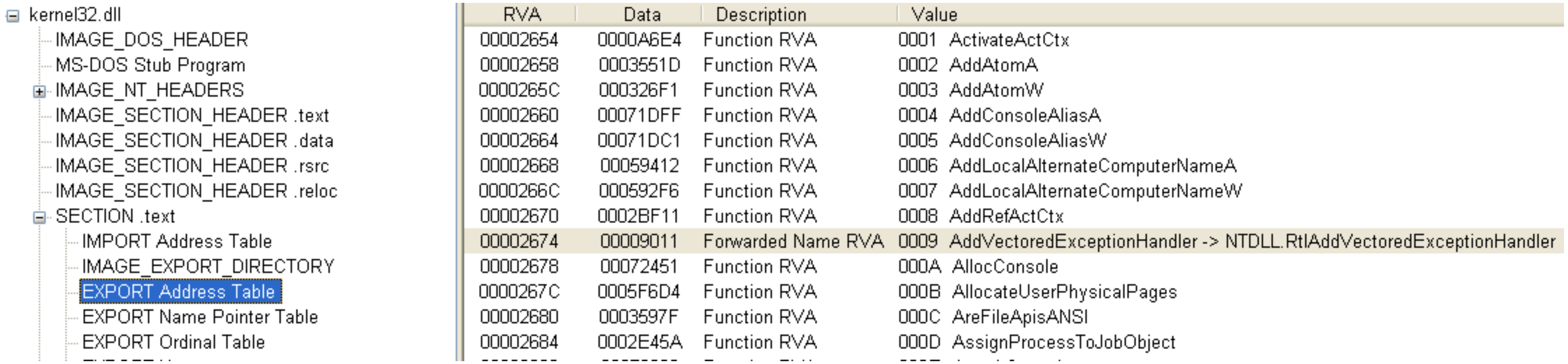
Module Definition File

Use specified module definition file during executable creation. (/DEF:name)

Forwarded Exports

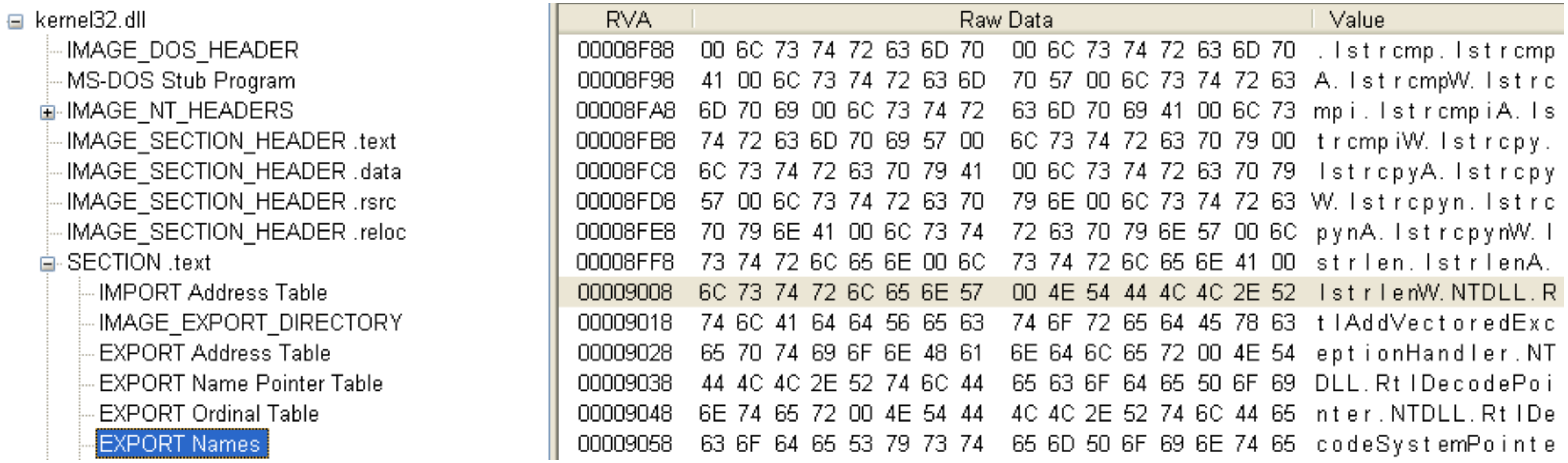
- There is an option to forward a function from one module to be handled by another one (e.g. it might be used if code was refactored to move a function to a different module, but you wanted to maintain backward compatibility.)
- As we just saw, normally **AddressOfFunctions** points to an array of RVAs which point at code. However, if a RVA in that array of RVAs points into the exports section (as defined by the base and size given in the data directory entry), then the RVA will actually be pointing at a string of the form DllToForwardTo.FunctionName

Kernel32.dll forwarded (to ntdll.dll) exports



- kernel32.dll
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - IMAGE_SECTION_HEADER .reloc
  - SECTION .text
    - IMPORT Address Table
    - IMAGE_EXPORT_DIRECTORY
    - EXPORT Address Table
    - EXPORT Name Pointer Table
    - EXPORT Ordinal Table

RVA	Data	Description	Value
00002654	0000A6E4	Function RVA	0001 ActivateActCtx
00002658	0003551D	Function RVA	0002 AddAtomA
0000265C	000326F1	Function RVA	0003 AddAtomW
00002660	00071DFF	Function RVA	0004 AddConsoleAliasA
00002664	00071DC1	Function RVA	0005 AddConsoleAliasW
00002668	00059412	Function RVA	0006 AddLocalAlternateComputerNameA
0000266C	000592F6	Function RVA	0007 AddLocalAlternateComputerNameW
00002670	0002BF11	Function RVA	0008 AddRefActCtx
00002674	00009011	Forwarded Name RVA	0009 AddVectoredExceptionHandler -> NTDLL.RtlAddVectoredExceptionHandler
00002678	00072451	Function RVA	000A AllocConsole
0000267C	0005F6D4	Function RVA	000B AllocateUserPhysicalPages
00002680	0003597F	Function RVA	000C AreFileApisANSI
00002684	0002E45A	Function RVA	000D AssignProcessToJobObject



- kernel32.dll
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - IMAGE_SECTION_HEADER .reloc
  - SECTION .text
    - IMPORT Address Table
    - IMAGE_EXPORT_DIRECTORY
    - EXPORT Address Table
    - EXPORT Name Pointer Table
    - EXPORT Ordinal Table
    - EXPORT Names

```
RVA       Raw Data                                          Value
00008F88  00 6C 73 74 72 63 6D 70  00 6C 73 74 72 63 6D 70  .lstrcmp.lstrcmp
00008F98  41 00 6C 73 74 72 63 6D  70 57 00 6C 73 74 72 63  A.lstrcmpW.lstrc
00008FA8  6D 70 69 00 6C 73 74 72  63 6D 70 69 41 00 6C 73  mpi.lstrcmpiA.ls
00008FB8  74 72 63 6D 70 69 57 00  6C 73 74 72 63 70 79 00  trcmpiW.lstrcpy.
00008FC8  6C 73 74 72 63 70 79 41  00 6C 73 74 72 63 70 79  lstrcpyA.lstrcpy
00008FD8  57 00 6C 73 74 72 63 70  79 6E 00 6C 73 74 72 63  W.lstrcpyn.lstrc
00008FE8  70 79 6E 41 00 6C 73 74  72 63 70 79 6E 57 00 6C  pynA.lstrcpynW.l
00008FF8  73 74 72 6C 65 6E 00 6C  73 74 72 6C 65 6E 41 00  strlen.lstrlenA.
00009008  6C 73 74 72 6C 65 6E 57  00 4E 54 44 4C 4C 2E 52  lstrlenW.NTDLL.R
00009018  74 6C 41 64 64 56 65 63  74 6F 72 65 64 45 78 63  tlAddVectoredExc
00009028  65 70 74 69 6F 6E 48 61  6E 64 6C 65 72 00 4E 54  eptionHandler.NT
00009038  44 4C 4C 2E 52 74 6C 44  65 63 6F 64 65 50 6F 69  DLL.RtlDecodePoi
00009048  6E 74 65 72 00 4E 54 44  4C 4C 2E 52 74 6C 44 65  nter.NTDLL.RtlDe
00009058  63 6F 64 65 53 79 73 74  65 6D 50 6F 69 6E 74 65  codeSystemPointe
```

How does one go about forwarding exports?

- Statement in .def file of the form

```
EXPORTS
FunctionAlias=OtherDLLName.RealFunction
```

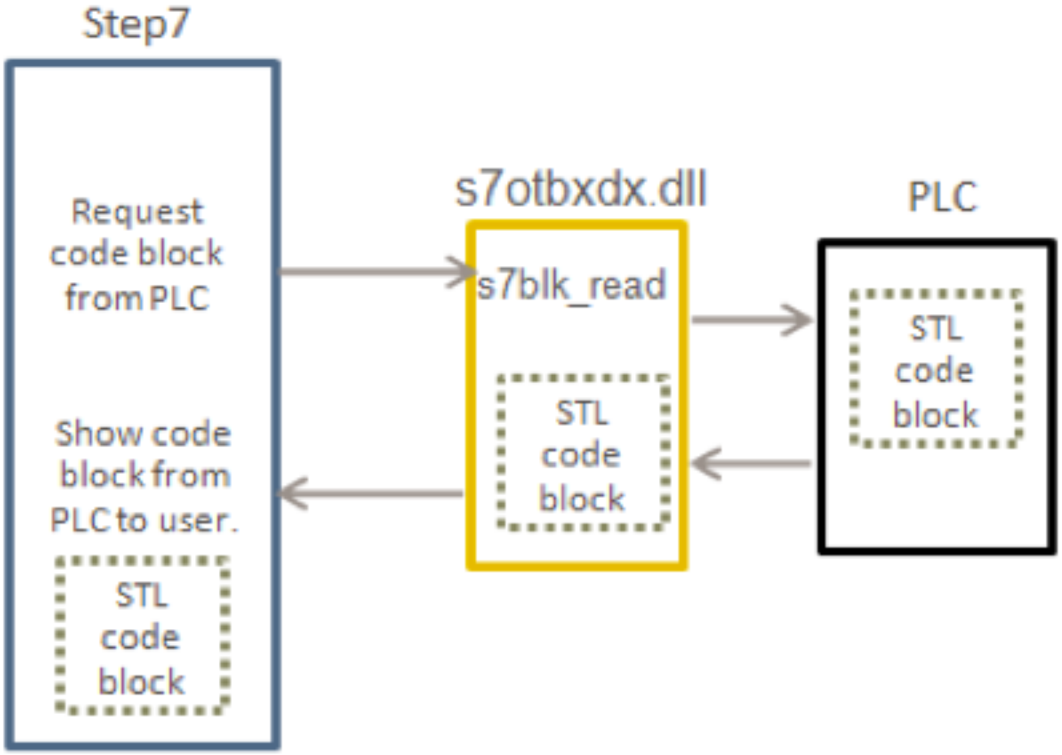
- or /export linker option
- `/export:FunctionAlias=OtherDLLName.RealFunction`
- Can even specify a linker comment in the code with
- `#pragma comment(linker, "/export:FunctionAlias=OtherDLLName.RealFunction")`

Relevance to Stuxnet

- Stuxnet used forwarded exports for the 93 of 109 exports in s7otbxdx.dll which it didn’t need to intercept.

Figure 18

Step7 and PCL communicating via s7otbxdx.dll

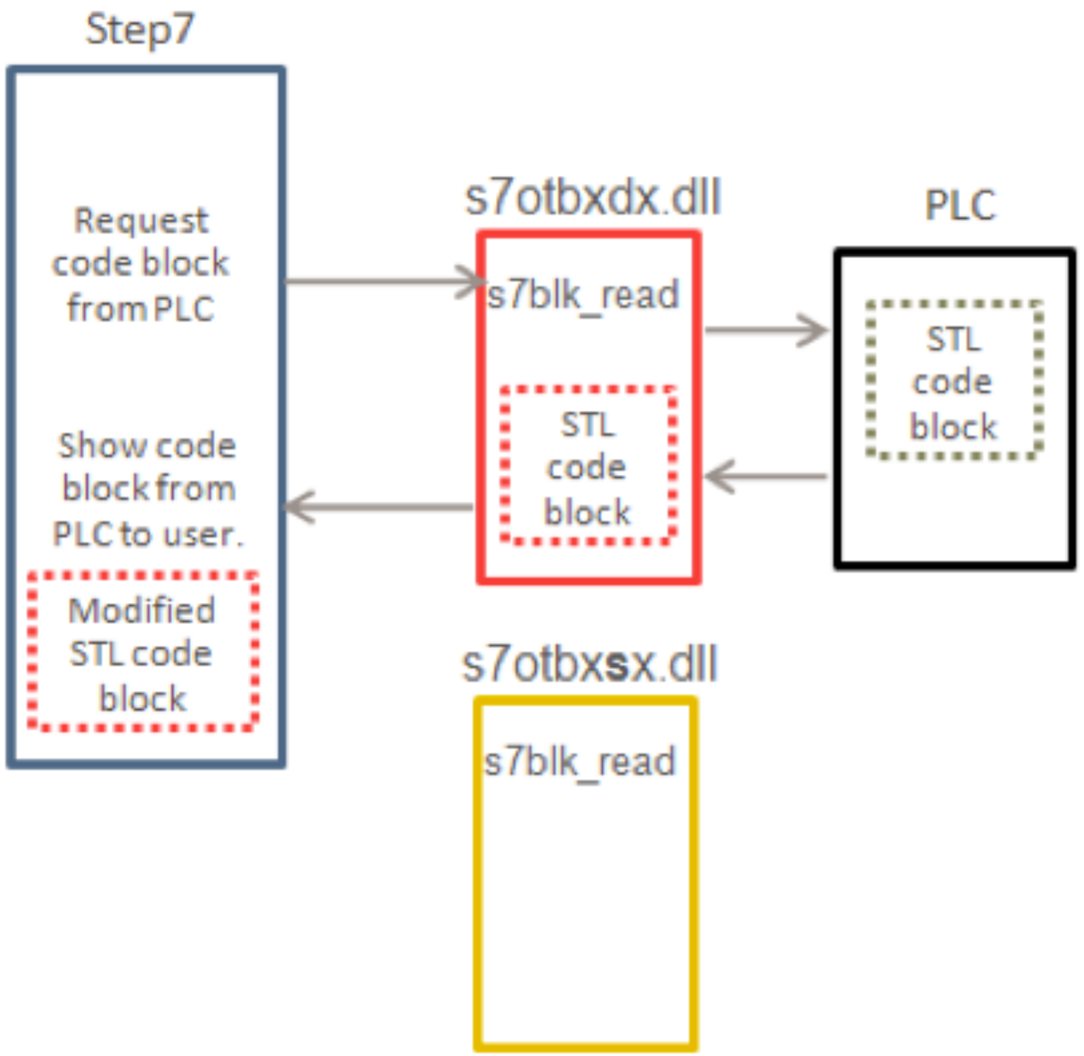


From http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Stuxnet trojaned DLL

Figure 19

Communication with malicious version of s7otbxdx.dll



From http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Function Redirection Tutorial

- http://packetstormsecurity.org/papers/win/intercept_apis_dll_redirection.pdf
- Basically talks about making a trojan DLL which hooks or reimplements some functions for the intercepted DLL, and then forwards the rest on to the original. Basically exactly what Stuxnet did for the trojan PLC accessing DLL.

Returning to Bound Imports

- Just to fill this in, now that we know about forwarded functions, the point of NumberOfModuleForwarderRefs and IMAGE_BOUND_FORWARDER_REF is that when the linker is trying to validate that none of the bound imports are changed, it needs to make sure none of the versions (TimeDateStamps) of imported modules has changed. Therefore if a module is bound to any modules which forward to other modules, those forwarded-to modules must be checked as well

```
typedef struct _IMAGE_BOUND_IMPORT_DESCRIPTOR {
   DWORD   TimeDateStamp;
   WORD    OffsetModuleName;
   WORD    NumberOfModuleForwarderRefs;
// Array of zero or more IMAGE_BOUND_FORWARDER_REF follows
} IMAGE_BOUND_IMPORT_DESCRIPTOR,  *PIMAGE_BOUND_IMPORT_DESCRIPTOR;

typedef struct _IMAGE_BOUND_FORWARDER_REF {
   DWORD   TimeDateStamp;
   WORD    OffsetModuleName;
   WORD    Reserved;
} IMAGE_BOUND_FORWARDER_REF, *PIMAGE_BOUND_FORWARDER_REF;
```

WHILE we’re thinking back…

- What are the three types of imports?
- What is the difference between importing by name vs. ordinal?
- Binding vs. ASLR: There can

be only one?

- What did the life-size cut out of

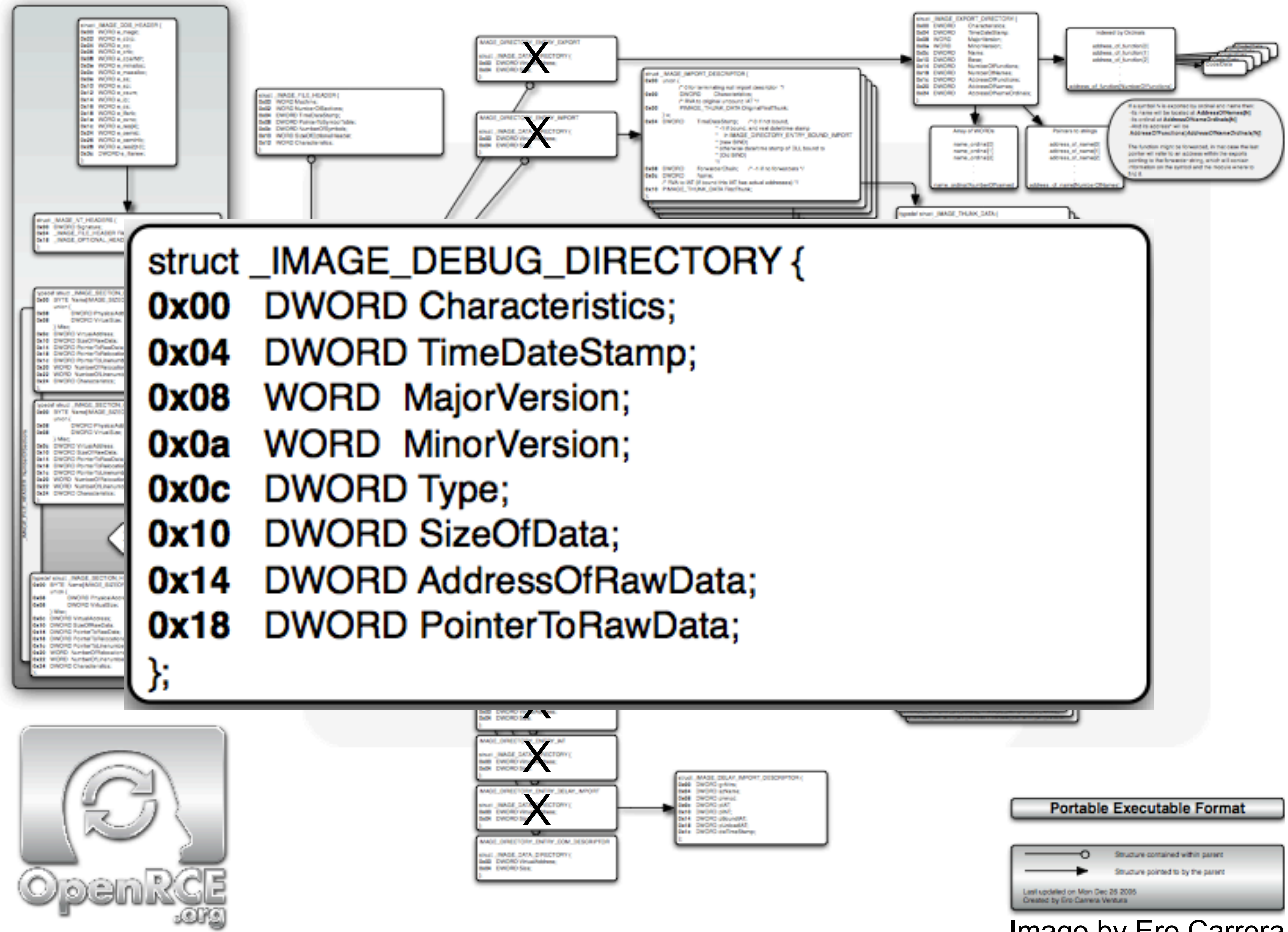
Anakin Skywalker look like?

EAT Hooking

- IAT hooking can modify all *currently loaded* modules in a process’ address space. If something new gets loaded (say, through LoadLibrary()), the attacker would need to be notified of this even to hook it’s IAT too.
- Instead, if the attacker modifies the EAT in the module which contains the the functions which he is intercepting, when a new module is loaded, he can just let the loaded do its thing, and the new module will point at the attacker’s code. Thus EAT hooking provides some “forward compatibility” assurance to the attacker that he will continue to hook the functions for all subsequently loaded modules.

EAT Hooking Lab

- beta: http://www.codeproject.com/KB/system/api_spying_hack.aspx


X
X
X
X
X
OpenRCE.org
Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura

```
struct _IMAGE_DEBUG_DIRECTORY {
0x00   DWORD Characteristics;
0x04   DWORD TimeDateStamp;
0x08   WORD  MajorVersion;
0x0a   WORD  MinorVersion;
0x0c   DWORD Type;
0x10   DWORD SizeOfData;
0x14   DWORD AddressOfRawData;
0x18   DWORD PointerToRawData;
};
```

Image by Ero Carrera

Debug Info

from winnt.h

```
typedef struct _IMAGE_DEBUG_DIRECTORY {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    DWORD   Type;
    DWORD   SizeOfData;
    DWORD   AddressOfRawData;
    DWORD   PointerToRawData;
} IMAGE_DEBUG_DIRECTORY, *PIMAGE_DEBUG_DIRECTORY;

#define IMAGE_DEBUG_TYPE_UNKNOWN          0
#define IMAGE_DEBUG_TYPE_COFF             1
#define IMAGE_DEBUG_TYPE_CODEVIEW         2
#define IMAGE_DEBUG_TYPE_FPO              3
#define IMAGE_DEBUG_TYPE_MISC             4
#define IMAGE_DEBUG_TYPE_EXCEPTION        5
#define IMAGE_DEBUG_TYPE_FIXUP            6
#define IMAGE_DEBUG_TYPE_OMAP_TO_SRC      7
#define IMAGE_DEBUG_TYPE_OMAP_FROM_SRC    8
#define IMAGE_DEBUG_TYPE_BORLAND          9
#define IMAGE_DEBUG_TYPE_RESERVED10       10
#define IMAGE_DEBUG_TYPE_CLSID            11
```

Debug Info 2

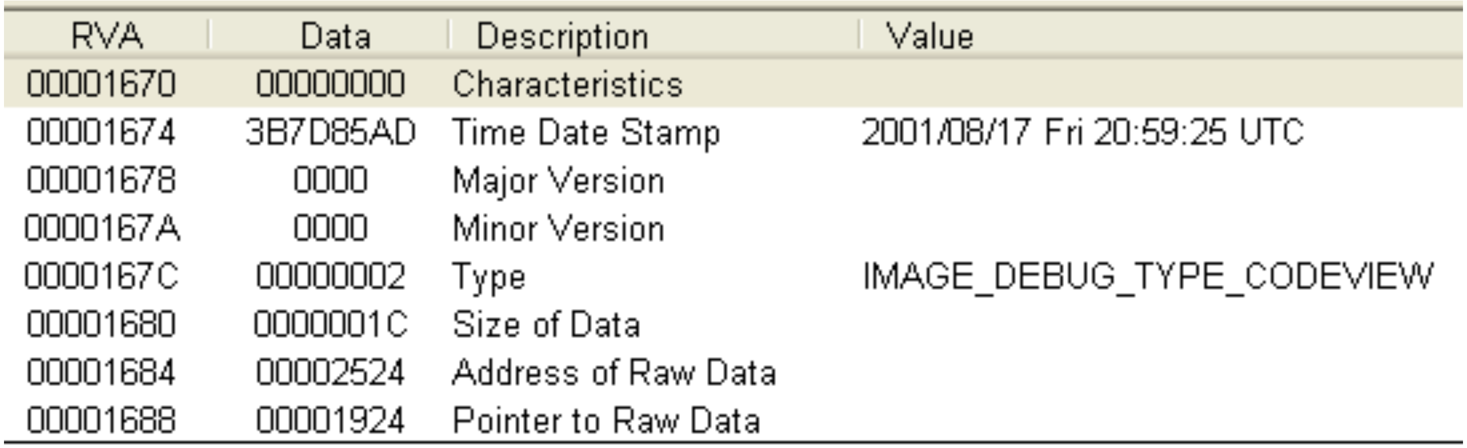
- **TimeDateStamp**, yet another to sanity check against. Should be the same as the one in the File Header I believe.
- **Type** and **SizeOfData** are what you would expect. The main Type we care about is IMAGE_DEBUG_TYPE_CODEVIEW as this is the common form now which points to a structure which holds a path to the pdb file which holds the debug symbols.
- **AddressOfRawData** is an RVA to the debug info.
- **PointerToRawData** is a file offset to the debug info.

Debug Info 3

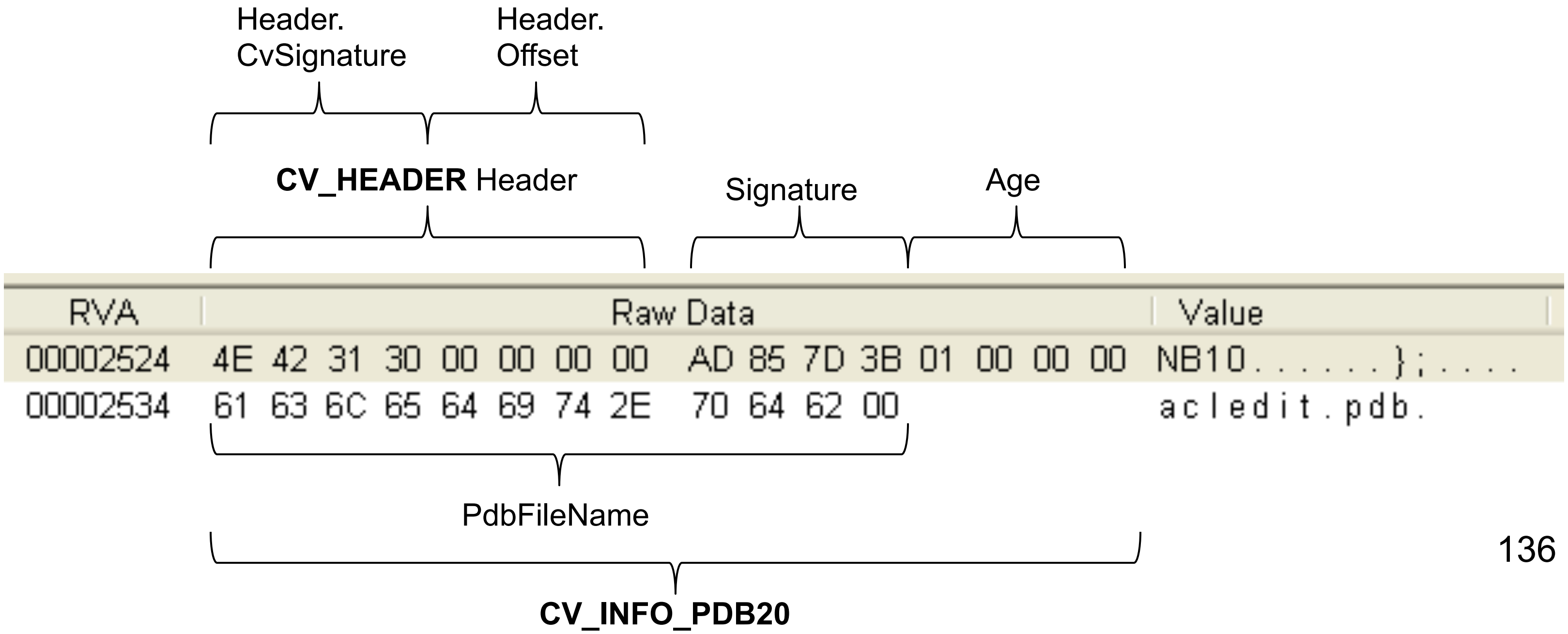
```
From http://www.debuginfo.com/examples/src/DebugDir.cpp

#define CV_SIGNATURE_NB10   '01BN’
#define CV_SIGNATURE_RSDS   'SDSR’
// CodeView header
struct CV_HEADER {
DWORD CvSignature; // NBxx
LONG  Offset;      // Always 0 for NB10
};
// CodeView NB10 debug information
// (used when debug information is stored in a PDB 2.00 file)
struct CV_INFO_PDB20 {
CV_HEADER  Header;
DWORD      Signature;       // seconds since 01.01.1970
DWORD      Age;             // an always-incrementing value
BYTE       PdbFileName[1];  // zero terminated string with the name of the PDB file
};

// CodeView RSDS debug information
// (used when debug information is stored in a PDB 7.00 file)
struct CV_INFO_PDB70 {
DWORD      CvSignature;
GUID       Signature;       // unique identifier
DWORD      Age;             // an always-incrementing value
BYTE       PdbFileName[1];  // zero terminated string with the name of the PDB file
};
```

Oh yay!
Another TimeDateStamp!

Therefore, how shall we interpret this?



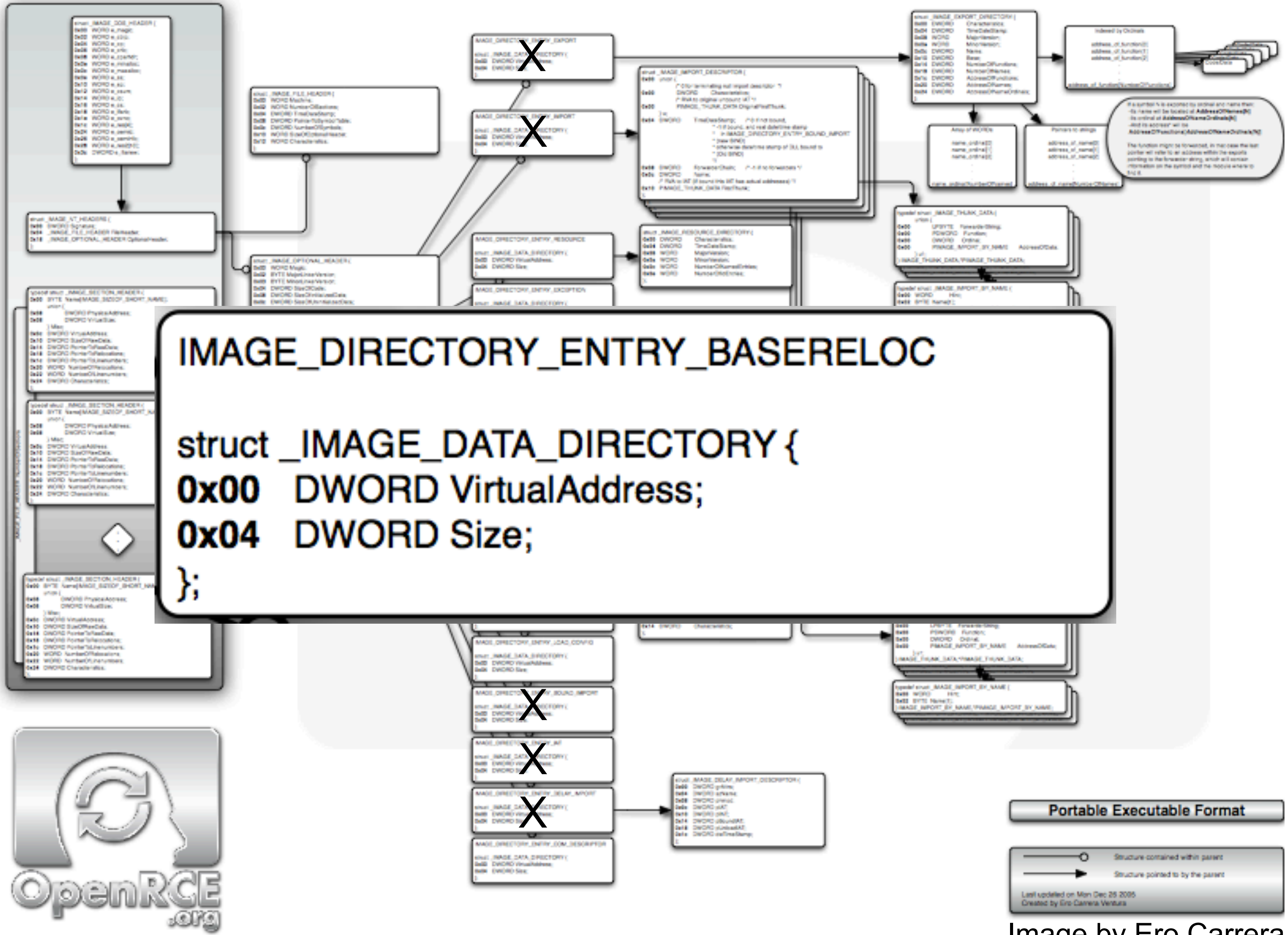
RVA	Data	Description	Value
00001670	00000000	Characteristics	
00001674	3B7D85AD	Time Date Stamp	2001/08/17 Fri 20:59:25 UTC
00001678	0000	Major Version	
0000167A	0000	Minor Version	
0000167C	00000002	Type	IMAGE_DEBUG_TYPE_CODEVIEW
00001680	0000001C	Size of Data	
00001684	00002524	Address of Raw Data	
00001688	00001924	Pointer to Raw Data	



RVA	Raw Data	Value
00002524	4E 42 31 30 00 00 00 00 AD 85 7D 3B 01 00 00 00	NB10.......};....
00002534	61 63 6C 65 64 69 74 2E 70 64 62 00	acledit.pdb.

A thing of the past?

- Between pulling a pdb path from high profile malware like GhostNet, Aurora, and Stuxnet malware, and Greg Hoglund starting to talk (at BlackHat LV 2010) about using pdb paths and TimeDateStamps to provide better attribution for malware authors, are we going to see any meaningful values here anymore? Time will tell.
- e:\gh0st\server\sys\i386\RESSDT.pdb
- \Aurora_Src\AuroraVNC\Avc\Release\AVC.pdb
- b:\myrtus\src\objfre_w2k_x86\i386\guava.pdb


X
X
X
X
X
Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE.org

IMAGE_DIRECTORY_ENTRY_BASERELOC

```
struct _IMAGE_DATA_DIRECTORY {
0x00   DWORD VirtualAddress;
0x04   DWORD Size;
};
```

Image by Ero Carrera

Relocations

from winnt.h

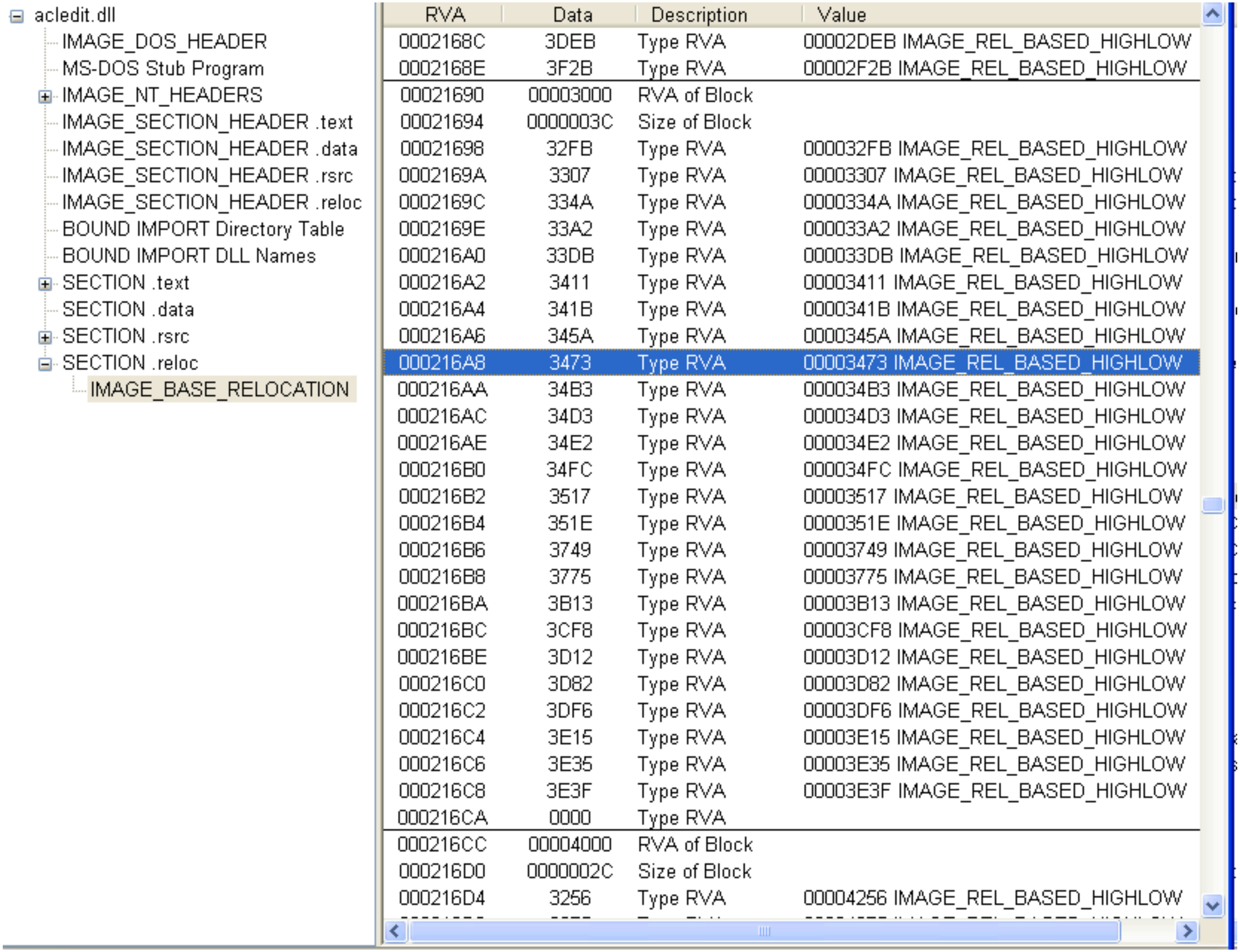
- Generally stored in the .reloc section
- Not shown on the picture the IMAGE_DIRECTORY_ENTRY_BASERELOC points at an array of IMAGE_BASE_RELOCATION structures.

```
typedef struct _IMAGE_BASE_RELOCATION {
    DWORD   VirtualAddress;
    DWORD   SizeOfBlock;
//  WORD    TypeOffset[1];
} IMAGE_BASE_RELOCATION;
```

Relocations 2

- **VirtualAddress** specifies the page-aligned virtual address that the specified relocation targets will be relative to.
- **SizeOfBlock** is the size of the IMAGE_BASE_RELOCATION itself + all of the subsequent relocation targets.
- Following SizeOfBlock are a variable number of WORD-sized relocation targets. The number of targets can be calculated as (SizeOfBlock – sizeof(`IMAGE_BASE_RELOCATION`)) / sizeof(WORD).

Relocations example acledit.dll



- acledit.dll
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - IMAGE_SECTION_HEADER .reloc
  - BOUND IMPORT Directory Table
  - BOUND IMPORT DLL Names
  - SECTION .text
  - SECTION .data
  - SECTION .rsrc
  - SECTION .reloc
    - IMAGE_BASE_RELOCATION

RVA	Data	Description	Value
0002168C	3DEB	Type RVA	00002DEB IMAGE_REL_BASED_HIGHLOW
0002168E	3F2B	Type RVA	00002F2B IMAGE_REL_BASED_HIGHLOW
00021690	00003000	RVA of Block	
00021694	0000003C	Size of Block	
00021698	32FB	Type RVA	000032FB IMAGE_REL_BASED_HIGHLOW
0002169A	3307	Type RVA	00003307 IMAGE_REL_BASED_HIGHLOW
0002169C	334A	Type RVA	0000334A IMAGE_REL_BASED_HIGHLOW
0002169E	33A2	Type RVA	000033A2 IMAGE_REL_BASED_HIGHLOW
000216A0	33DB	Type RVA	000033DB IMAGE_REL_BASED_HIGHLOW
000216A2	3411	Type RVA	00003411 IMAGE_REL_BASED_HIGHLOW
000216A4	341B	Type RVA	0000341B IMAGE_REL_BASED_HIGHLOW
000216A6	345A	Type RVA	0000345A IMAGE_REL_BASED_HIGHLOW
000216A8	3473	Type RVA	00003473 IMAGE_REL_BASED_HIGHLOW
000216AA	34B3	Type RVA	000034B3 IMAGE_REL_BASED_HIGHLOW
000216AC	34D3	Type RVA	000034D3 IMAGE_REL_BASED_HIGHLOW
000216AE	34E2	Type RVA	000034E2 IMAGE_REL_BASED_HIGHLOW
000216B0	34FC	Type RVA	000034FC IMAGE_REL_BASED_HIGHLOW
000216B2	3517	Type RVA	00003517 IMAGE_REL_BASED_HIGHLOW
000216B4	351E	Type RVA	0000351E IMAGE_REL_BASED_HIGHLOW
000216B6	3749	Type RVA	00003749 IMAGE_REL_BASED_HIGHLOW
000216B8	3775	Type RVA	00003775 IMAGE_REL_BASED_HIGHLOW
000216BA	3B13	Type RVA	00003B13 IMAGE_REL_BASED_HIGHLOW
000216BC	3CF8	Type RVA	00003CF8 IMAGE_REL_BASED_HIGHLOW
000216BE	3D12	Type RVA	00003D12 IMAGE_REL_BASED_HIGHLOW
000216C0	3D82	Type RVA	00003D82 IMAGE_REL_BASED_HIGHLOW
000216C2	3DF6	Type RVA	00003DF6 IMAGE_REL_BASED_HIGHLOW
000216C4	3E15	Type RVA	00003E15 IMAGE_REL_BASED_HIGHLOW
000216C6	3E35	Type RVA	00003E35 IMAGE_REL_BASED_HIGHLOW
000216C8	3E3F	Type RVA	00003E3F IMAGE_REL_BASED_HIGHLOW
000216CA	0000	Type RVA	
000216CC	00004000	RVA of Block	
000216D0	0000002C	Size of Block	
000216D4	3256	Type RVA	00004256 IMAGE_REL_BASED_HIGHLOW

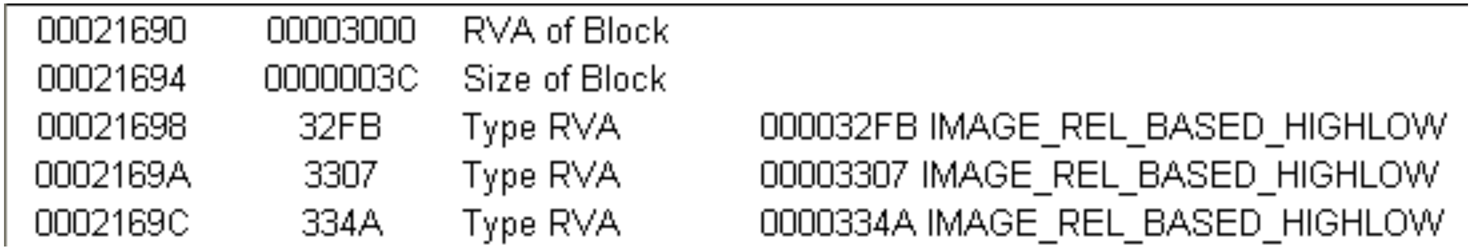
Relocations 3

- The upper 4 bits of the 16 bit relocation target specifies the type. The lower 12 bits specifies an offset, which will be used differently depending on the type. Types are:

```
#define IMAGE_REL_BASED_ABSOLUTE               0
#define IMAGE_REL_BASED_HIGH                   1
#define IMAGE_REL_BASED_LOW                    2
#define IMAGE_REL_BASED_HIGHLOW                3
#define IMAGE_REL_BASED_HIGHADJ                4
#define IMAGE_REL_BASED_MIPS_JMPADDR           5
#define IMAGE_REL_BASED_MIPS_JMPADDR16         9
#define IMAGE_REL_BASED_IA64_IMM64             9
#define IMAGE_REL_BASED_DIR64                  10
```

- We generally only care about IMAGE_REL_BASED_HIGHLOW, which when used says that the RVA for the data to be relocated is specified by **VirtualAddress** + the lower 12 bits.

Slice of life



```
00021690    00003000   RVA of Block
00021694    0000003C   Size of Block
00021698      32FB     Type RVA       000032FB IMAGE_REL_BASED_HIGHLOW
0002169A      3307     Type RVA       00003307 IMAGE_REL_BASED_HIGHLOW
0002169C      334A     Type RVA       0000334A IMAGE_REL_BASED_HIGHLOW
```

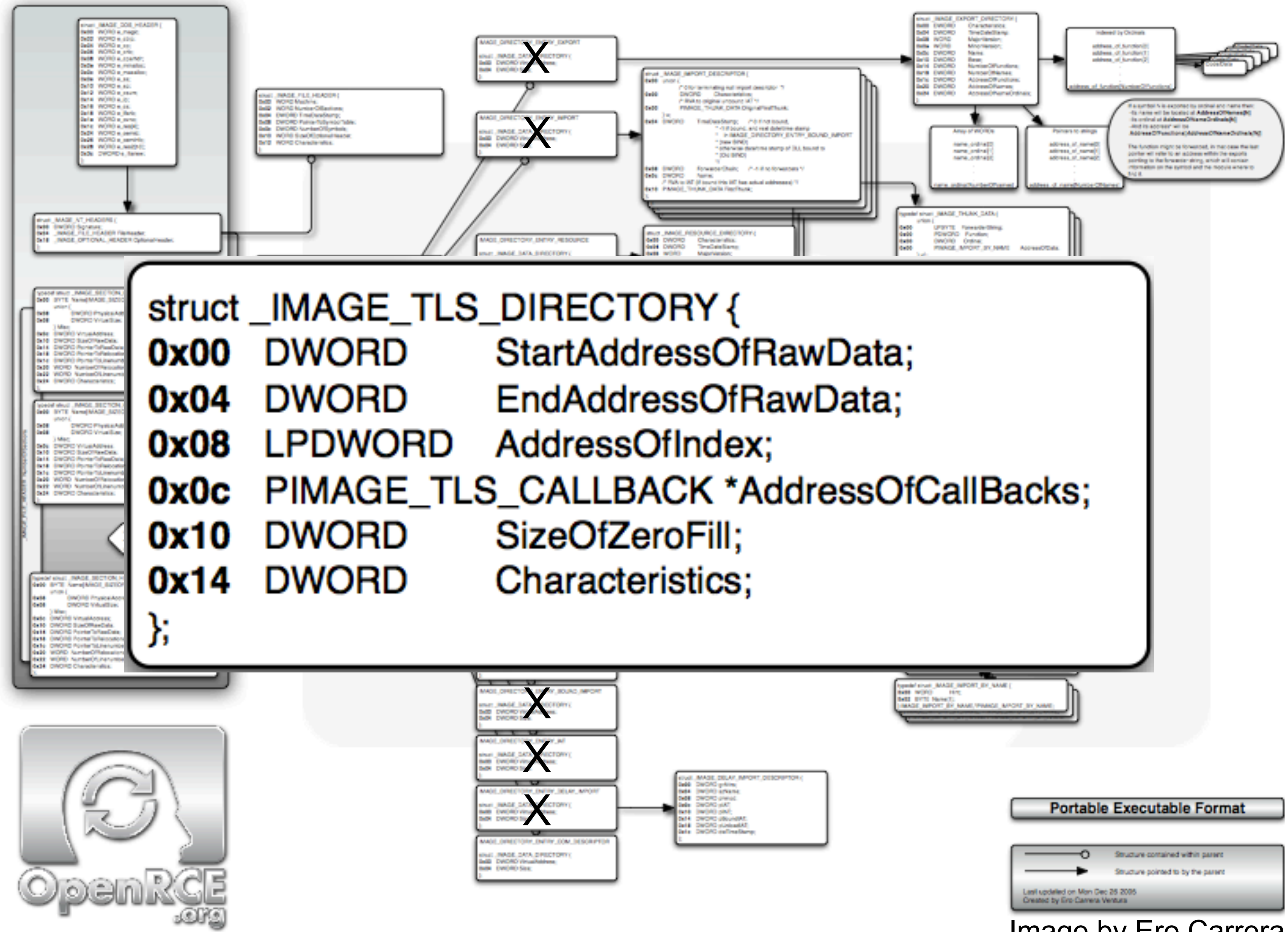
- So in the above if the file was being relocated, the loader would take the relocation target WORD 0x32FB, the upper 4 bits are 0x3 = IMAGE_REL_BASED_HIGHLOW. The lower 12 bits are 0x2FB. Given the type, we do (VirtualAddress (0x3000) + lower 12 bits (0x2FB)) == 0x32FB is the RVA of the location which needs to be fixed.
- Then the loaded would just add whatever the delta is between the file' s preferred load address and actual load address, and just add that delta to data at RVA 0x32FB.
- (Show example in WinDBG of what target for relocation can look like)

Memory Integrity Checking

- Let's say you want to make a memory integrity checker to look for inline hooks in running code. You know at this point that certain sections such as .text are marked as non-writable. Therefore you would think what is on disk should be the same as what's in memory. So to check for changes in memory, you should be able to hash the .text in memory, hash the .text read in from disk, and compare the hashes, right?
- Maybe. If the file isn't relocated when it's loaded into memory, yes that would work*. If the file is relocated when loaded, the application of the relocation fixups will change the bytes vs. what is on disk, and therefore change the hash. You can still compare hashes though if you now take the data read in from disk and apply relocations to it in the same way the loaded would have based on the delta between the preferred load address and the actual load address.
- *There are caveats such as the fact that things like the IAT can exist in "non-writable" memory, but it still gets written at load time, and thus differs from disk. That needs to be compensated for too.

Threads

- In modern OSes, processes generally have separate address spaces (as we talked about in the IAT/EAT hooking sections). Threads are distinct units of execution flow & context which are usually managed by the kernel, but which coexist within a single process address space. Therefore each thread can see the same global variables for instance, but care must be taken (mutual exclusion) to ensure they don’t incur race conditions where two threads access and modify some variable in a way which alters the other’s execution by screwing up its expectations.
- Therefore it is desirable sometimes to have variables (besides local (stack) variables) which are accessible only to a single thread. Thread Local Storage (TLS) is a mechanism which MS has provided in the PE spec to support this goal. They support both regular data as well as callback functions, which can initialize/destroy data on thread creation/destruction.



```
struct _IMAGE_TLS_DIRECTORY {
0x00   DWORD         StartAddressOfRawData;
0x04   DWORD         EndAddressOfRawData;
0x08   LPDWORD       AddressOfIndex;
0x0c   PIMAGE_TLS_CALLBACK *AddressOfCallBacks;
0x10   DWORD         SizeOfZeroFill;
0x14   DWORD         Characteristics;
};
```

Image by Ero Carrera

Thread Local Storage

from winnt.h

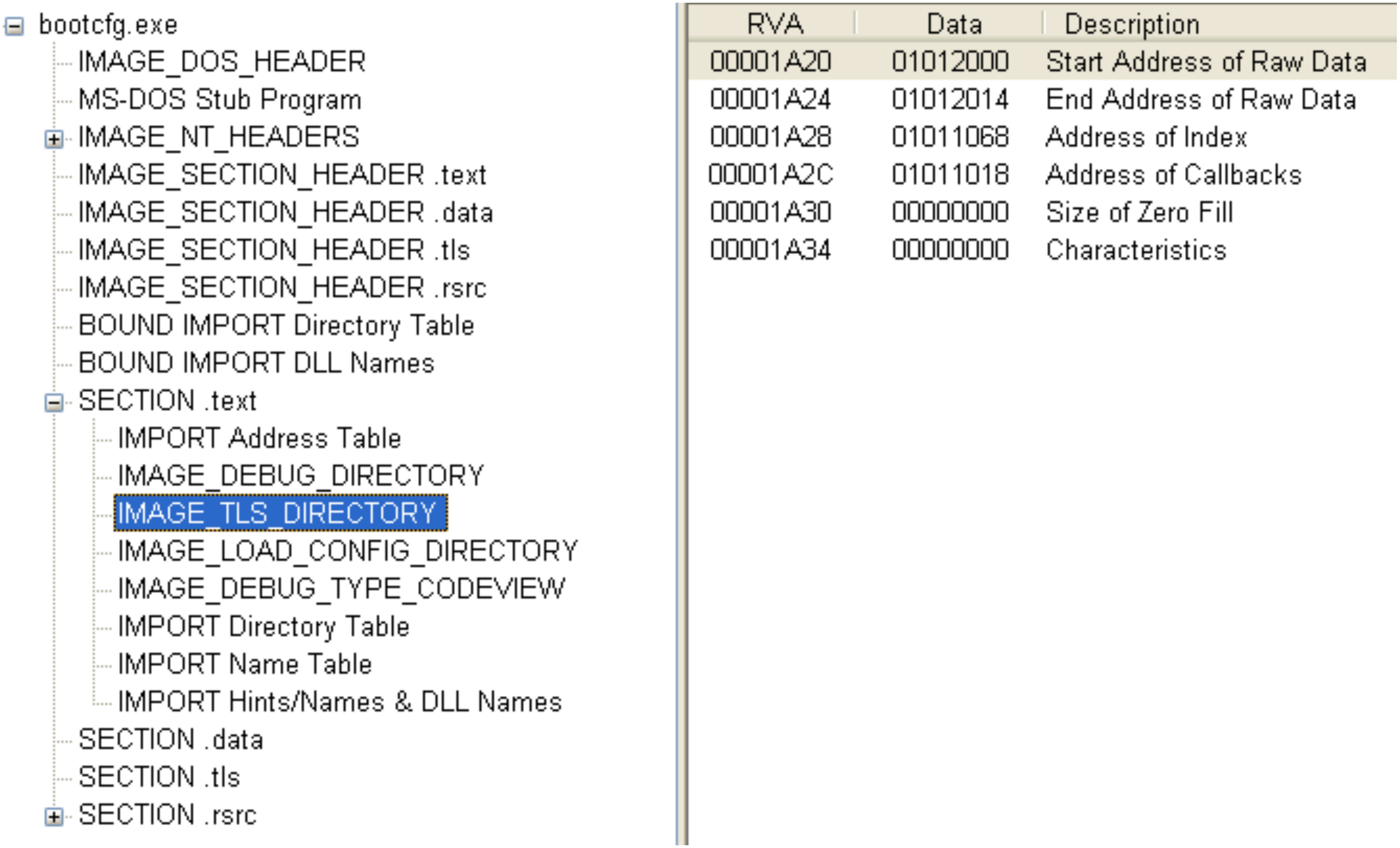
```
typedef struct _IMAGE_TLS_DIRECTORY32 {
    DWORD   StartAddressOfRawData;
    DWORD   EndAddressOfRawData;
    DWORD   AddressOfIndex;
    DWORD   AddressOfCallBacks;
    DWORD   SizeOfZeroFill;
    DWORD   Characteristics;
} IMAGE_TLS_DIRECTORY32;
```

Thread Local Storage 2

- **StartAddressOfRawData** is the absolute virtual address (not RVA, and therefore subject to relocations) where the data starts.
- **EndAddressOfRawData** is the absolute virtual address (not RVA, and therefore subject to relocations) where the data ends.
- **AddressOfCallbacks** absolute virtual address points to an array of PIMAGE_TLS_CALLBACK function pointers.
- SizeOfZeroFill is interesting just because it's like a .bss zeroed blob tacked on after the TLS data.

C:\WINDOWS\system32\bootcfg.exe

(the only executable I could find that uses tls, thanks to a presumed bug in my property finder)



- bootcfg.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .tls
  - IMAGE_SECTION_HEADER .rsrc
  - BOUND IMPORT Directory Table
  - BOUND IMPORT DLL Names
  - SECTION .text
    - IMPORT Address Table
    - IMAGE_DEBUG_DIRECTORY
    - IMAGE_TLS_DIRECTORY
    - IMAGE_LOAD_CONFIG_DIRECTORY
    - IMAGE_DEBUG_TYPE_CODEVIEW
    - IMPORT Directory Table
    - IMPORT Name Table
    - IMPORT Hints/Names & DLL Names
  - SECTION .data
  - SECTION .tls
  - SECTION .rsrc

RVA	Data	Description
00001A20	01012000	Start Address of Raw Data
00001A24	01012014	End Address of Raw Data
00001A28	01011068	Address of Index
00001A2C	01011018	Address of Callbacks
00001A30	00000000	Size of Zero Fill
00001A34	00000000	Characteristics

Note that End Address – Start Address = 0x14. Go to .tls and look at the likely file alignment padding resulting in a larger section.

How does one go about defining TLS?

- http://msdn.microsoft.com/en-us/library/6yh4a9k1(VS.80).aspx
- __declspec(thread) int tls_i = 1;
- More info http://msdn.microsoft.com/en-us/library/ms686749(VS.85).aspx
- Note: No way listed to create callbacks. For that we have to consult with unofficial sources:
- http://www.nynaeve.net/?p=183
- http://hype-free.blogspot.com/2008/10/playing-tricks-with-windows-pe-loader.html

Lab: TSL Callbacks

- Use Ilfak's example and Skywing's

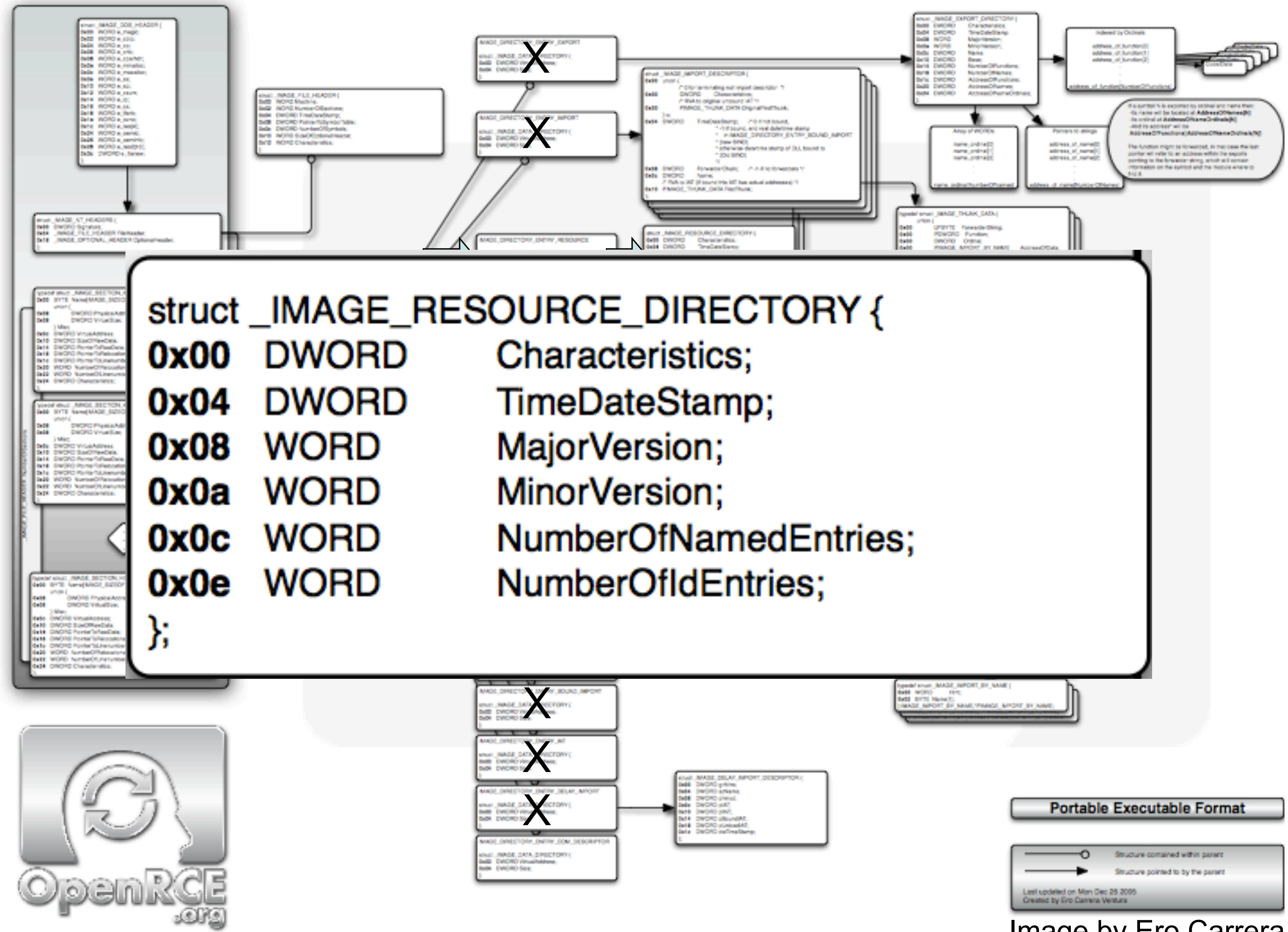
More TLS Anti-Debug Tricks

```
/*  TLS callback demonstration program.
    This program may be used to learn/illustrate the TLS callback concept.
    Copyright 2005 Ilfak Guilfanov <ig@hexblog.com>

    There is no standard way (from compiler vendors) of creating it.
    We use a special linker, UniLink, to create them.
    Please contact Yury Haron <yjh@styx.cabel.net> for more information
    about the linker.
*/

#include <windows.h>
#include <stdio.h>
#include "ulnfeat.h”
/* This is a TLS callback. It */
void __stdcall callback(void * /*instance*/,
                        DWORD reason,
                        void * /*reserved*/)
{
  if ( reason == DLL_PROCESS_ATTACH )
  {
    MessageBox(NULL, "Hello, world!", "Hidden message", MB_OK);
    ExitProcess(0);
  }
}
TLS_CALLBACK(c1, callback);     // Unilink trick to declare callbacks
/*  This is the main function.
    It will never be executed since the callback will call ExitProcess().
*/
int main(void)
{
  return 0;
}
```

From http://www.hexblog.com/?p=9

TLS misc

- TLS callbacks can be executed when a process or thread is started or stopped. (DLL_PROCESS_ATTACH, DLL_PROCESS_DETACH, DLL_THREAD_ATTACH, DLL_THREAD_DETACH), the thing being that despite the name, an exe is called with DLL_PROCESS_ATTACH.
- TLS data generally stored in the .tls section
- Self-modifying TLS callbacks: https://www.openrce.org/blog/view/1114/Self-modifying_TLS_callbacks
- Tls callbacks could also not just bypass a breakpoint, but remove it too! :) More descriptions of possible actions here: http://pferrie.tripod.com/papers/unpackers22.pdf


X
X
X
X
X
Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE
.org

```
struct _IMAGE_RESOURCE_DIRECTORY {
0x00   DWORD     Characteristics;
0x04   DWORD     TimeDateStamp;
0x08   WORD      MajorVersion;
0x0a   WORD      MinorVersion;
0x0c   WORD      NumberOfNamedEntries;
0x0e   WORD      NumberOfIdEntries;
};
```

Image by Ero Carrera

Resources

from winnt.h

- Generally stored in the .rsrc section

```
typedef struct _IMAGE_RESOURCE_DIRECTORY
  {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    WORD    NumberOfNamedEntries;
    WORD    NumberOfIdEntries;
} IMAGE_RESOURCE_DIRECTORY,
```

Resources 2

- Immediately following IMAGE_RESOURCE_DIRECTORY is an array of **NumberOfNamedEntries** + **NumberOfIdEntries** IMAGE_RESOURCE_DIRECTORY_ENTRY structs (with the Named entries first, followed by the ID entries.)
- A resource can be identified by a name or an ID, but not both.

Resources 3: What the…

```
typedef struct _IMAGE_RESOURCE_DIRECTORY_ENTRY {
    union {
        struct {
            DWORD NameOffset:31;
            DWORD NameIsString:1;
        };
        DWORD   Name;
        WORD    Id;
    };
    union {
        DWORD   OffsetToData;
        struct {
            DWORD   OffsetToDirectory:31;
            DWORD   DataIsDirectory:1;
        };
    };
} IMAGE_RESOURCE_DIRECTORY_ENTRY;
```

Resources 4

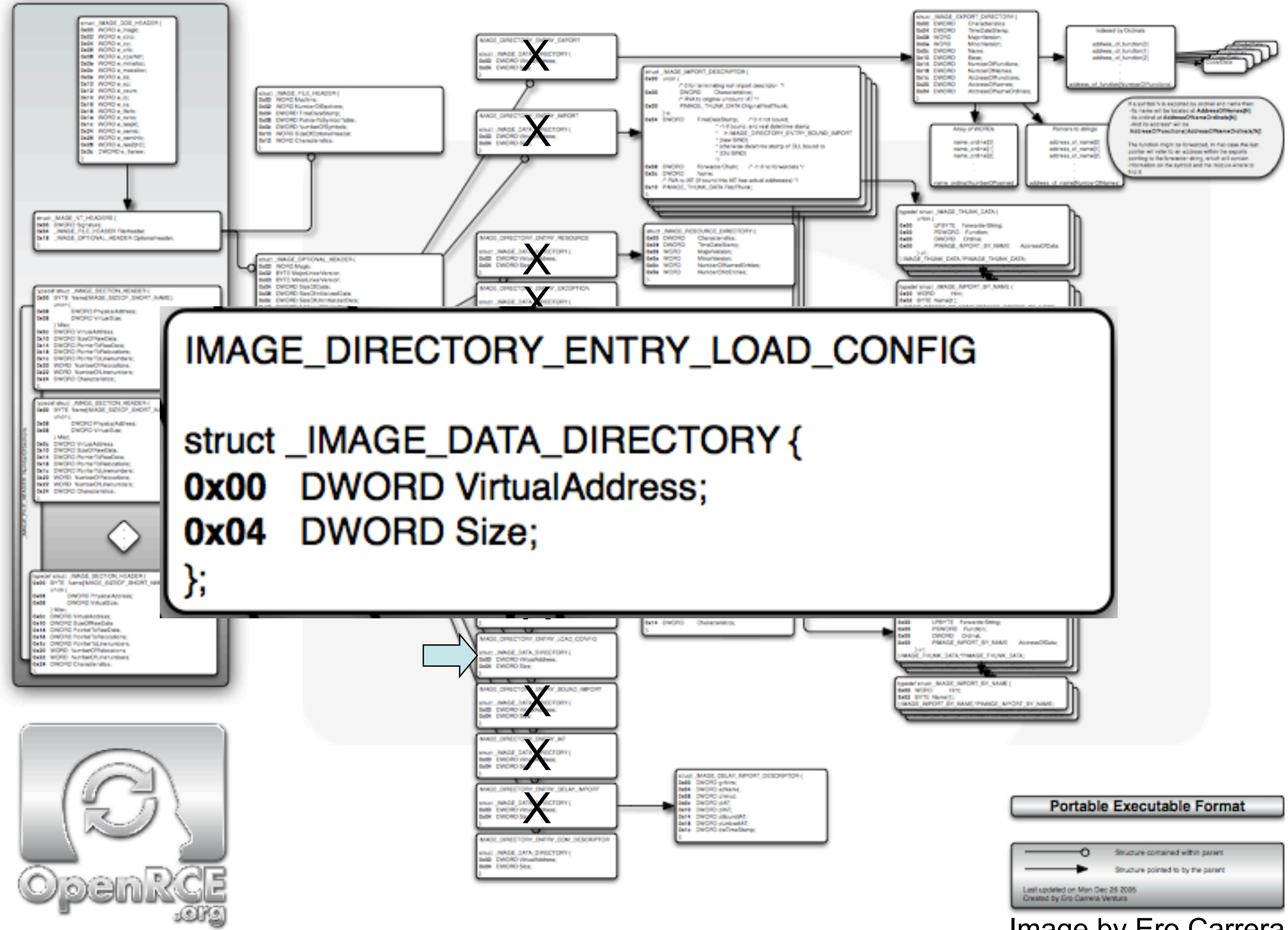
- It's actually simpler than it looks. If the first DWORD's MSB is set (and therefore it starts with 8), that means the lower 31 bits are an offset to a string which is the name of the resource (and is specified like a wide character pascal string…that is, instead of being null terminated, it starts with a length which specifies the number of characters which follow…haven't been able to find what the actual type is).
- If the MSB is not set, it's treated as a WORD sized ID.
- If the MSB of the second DWORD is set, that means the lower 31 bits are an offset to another IMAGE_RESOURCE_DIRECTORY.
- If the MSB is not set, that means it's an offset to the actual data.
- All offsets are relative to the start of resource section.
- Let's walk an example

Resources 5

- Using resources in Visual Studio: http://msdn.microsoft.com/en-us/library/7zxb70x7.aspx since I don't want to get into it.
- Both legitimate software and malware can embed additional binaries in the resources and then pull them out and execute them at runtime. E.g. ProcessExplorer and GMER .exes have kernel drivers embedded which they load on demand. Stuxnet also had numerous difference components such as kernel drivers, exploit code, dll injection templates, and config data embedded in resources.

ProcessExplorer.exe's resources

- Has embedded kernel drivers which it extracts and loads into memory on the fly. Different versions for x86 vs x86-64
- Look at the overloaded structs in PEView.


X
X
X
X
X
X
X
Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE.org

IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG

```
struct _IMAGE_DATA_DIRECTORY {
0x00   DWORD VirtualAddress;
0x04   DWORD Size;
};
```

Image by Ero Carrera

Load Configuration from winnt.h

- Another struct which doesn’t rate inclusion in the picture

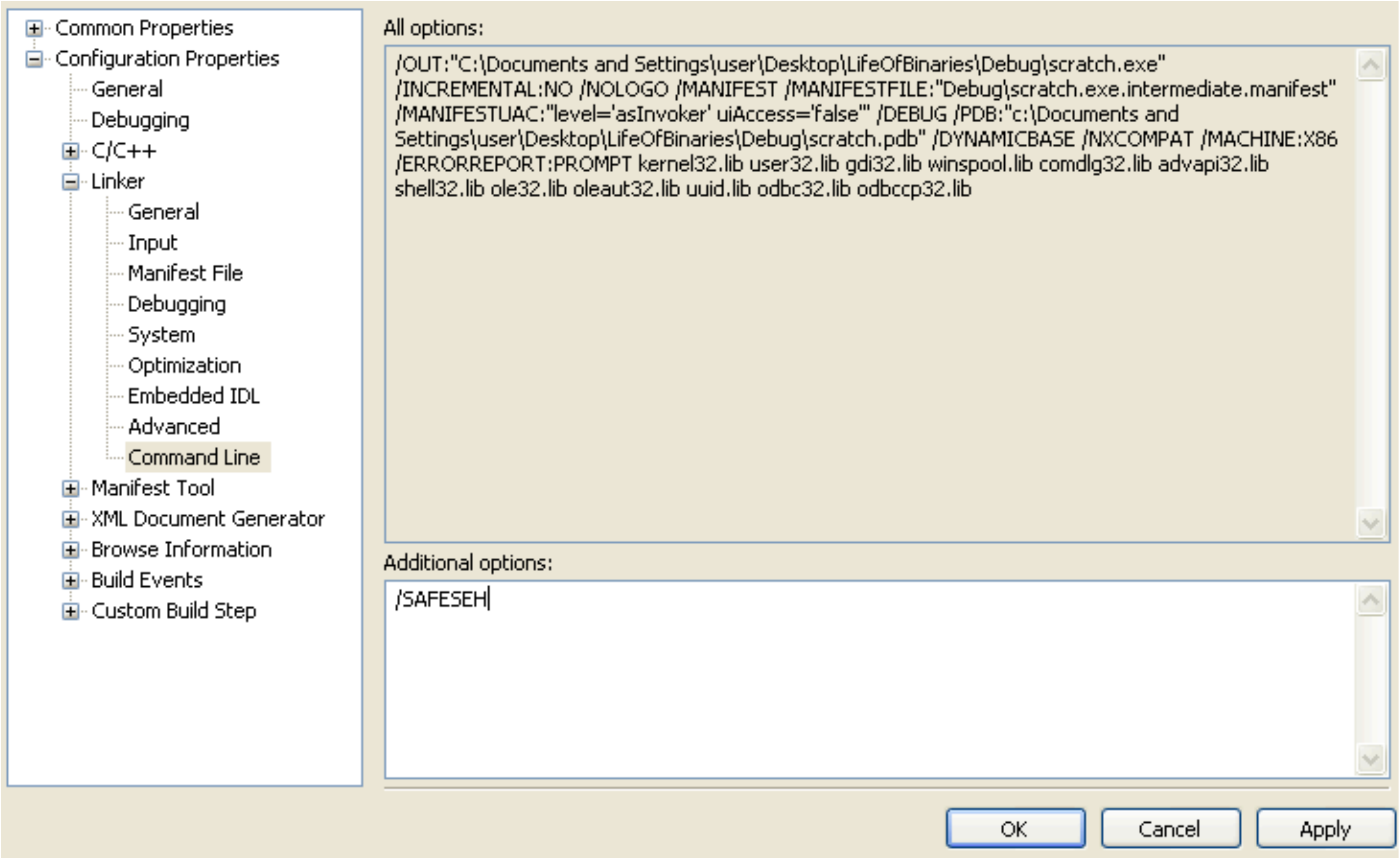
```
typedef struct {
    DWORD   Size;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    DWORD   GlobalFlagsClear;
    DWORD   GlobalFlagsSet;
    DWORD   CriticalSectionDefaultTimeout;
    DWORD   DeCommitFreeBlockThreshold;
    DWORD   DeCommitTotalFreeThreshold;
    DWORD   LockPrefixTable;            // VA
    DWORD   MaximumAllocationSize;
    DWORD   VirtualMemoryThreshold;
    DWORD   ProcessHeapFlags;
    DWORD   ProcessAffinityMask;
    WORD    CSDVersion;
    WORD    Reserved1;
    DWORD   EditList;                   // VA
    DWORD   SecurityCookie;             // VA
    DWORD   SEHandlerTable;             // VA
    DWORD   SEHandlerCount;
} IMAGE_LOAD_CONFIG_DIRECTORY32
```

Load Config

- **SecurityCookie** is a VA (not RVA, therefore subject to fixups) which points at the location where the stack cookie used with the /GS flag will be.
- **SEHandlerTable** is a VA (not RVA) which points to a table of RVAs which specify the only exception handlers which are valid for use with Structured Exception Handler (SEH). The placement of the pointers to these handlers is caused by the /SAFESEH linker options.
- Take Corey Kallenberg's exploits class to see how SafeSEH mitigates exploits.
- **SEHandlerCount** is then just the number of entries in the array pointed to by SEHandlerTable.
- See http://msdn.microsoft.com/en-us/library/ms680328(VS.85).aspx for a description of the rest of the fields

/SAFESEH

(There's no GUI option for this, and MS says to just set it manually)
http://msdn.microsoft.com/en-us/library/9a89h429(v=VS.100).aspx



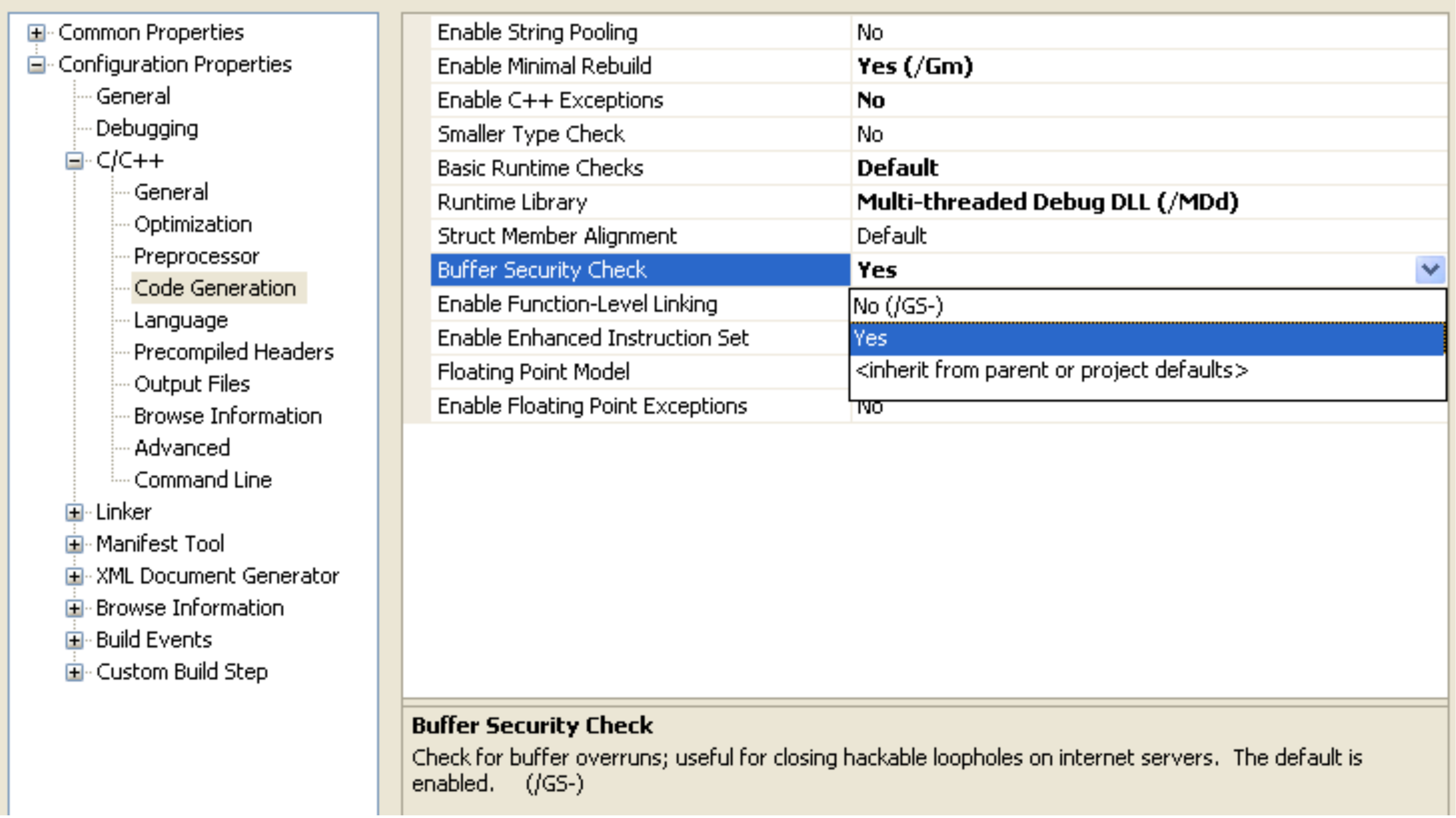
All options:

```
/OUT:"C:\Documents and Settings\user\Desktop\LifeOfBinaries\Debug\scratch.exe"
/INCREMENTAL:NO /NOLOGO /MANIFEST /MANIFESTFILE:"Debug\scratch.exe.intermediate.manifest"
/MANIFESTUAC:"level='asInvoker' uiAccess='false'" /DEBUG /PDB:"c:\Documents and
Settings\user\Desktop\LifeOfBinaries\Debug\scratch.pdb" /DYNAMICBASE /NXCOMPAT /MACHINE:X86
/ERRORREPORT:PROMPT kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
```

Additional options:

```
/SAFESEH
```

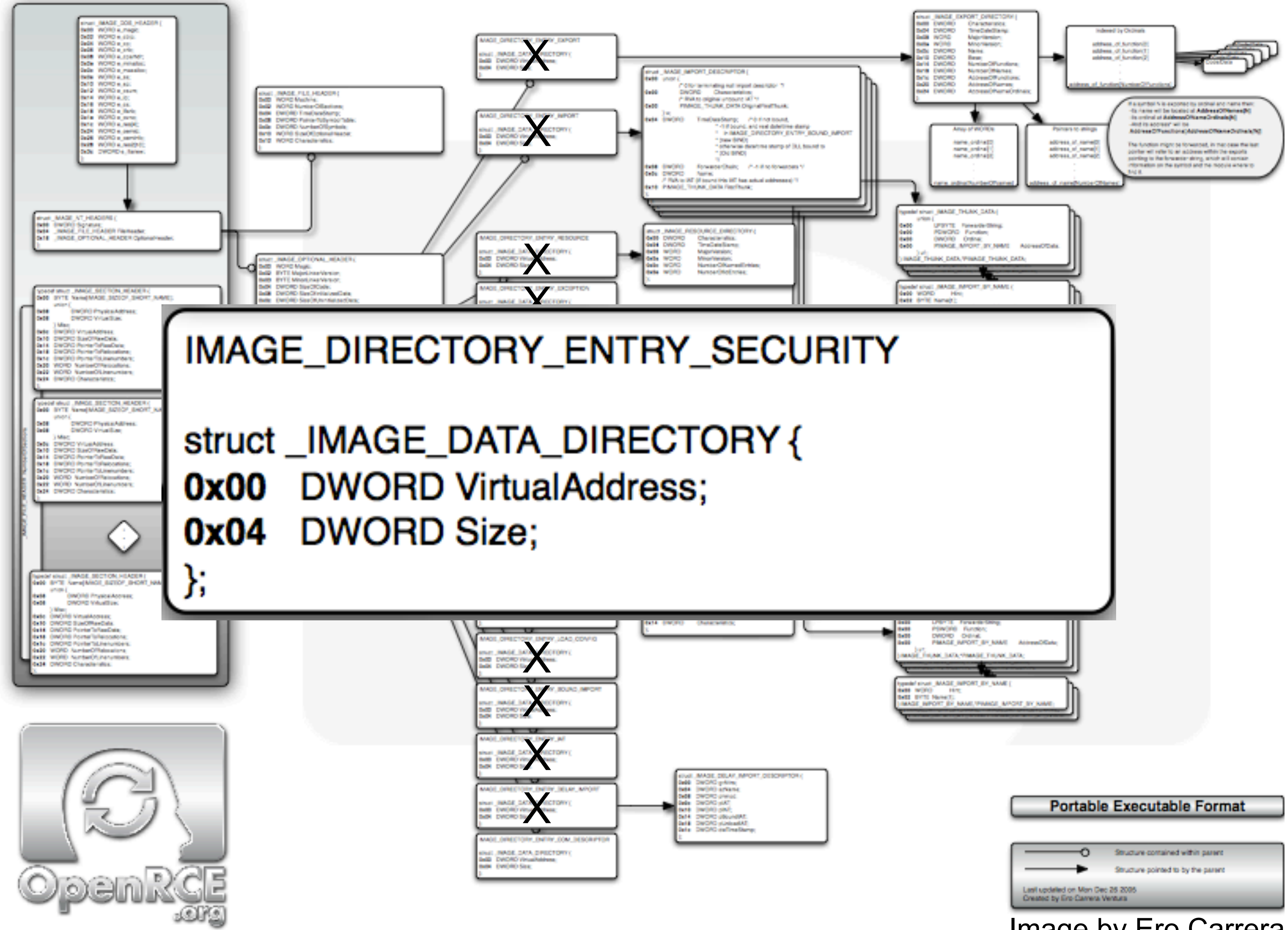
/GS "stack cookie/canary" option
Helps detect stack buffer overflows



Enable String Pooling	No
Enable Minimal Rebuild	**Yes (/Gm)**
Enable C++ Exceptions	**No**
Smaller Type Check	No
Basic Runtime Checks	**Default**
Runtime Library	**Multi-threaded Debug DLL (/MDd)**
Struct Member Alignment	Default
Buffer Security Check	**Yes**
Enable Function-Level Linking	
Enable Enhanced Instruction Set	
Floating Point Model	
Enable Floating Point Exceptions	No

Buffer Security Check

Check for buffer overruns; useful for closing hackable loopholes on internet servers. The default is enabled. (/GS-)


X
X
X
X
X
X
X
X
Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
OpenRCE.org

IMAGE_DIRECTORY_ENTRY_SECURITY

```
struct _IMAGE_DATA_DIRECTORY {
0x00   DWORD VirtualAddress;
0x04   DWORD Size;
};
```

Image by Ero Carrera

Digitally Signed Files (“Authenticode”)

- Where certificates are stored
- http://msdn.microsoft.com/en-us/library/ms537361(VS.85).aspx
- “The utility programs use the private key to generate a digital signature on a digest of the binary file and create a signature file containing the signed content of a public key certificate standard (PKCS) #7 signed-data object”
- ProcessExplorer as an example

And the rest

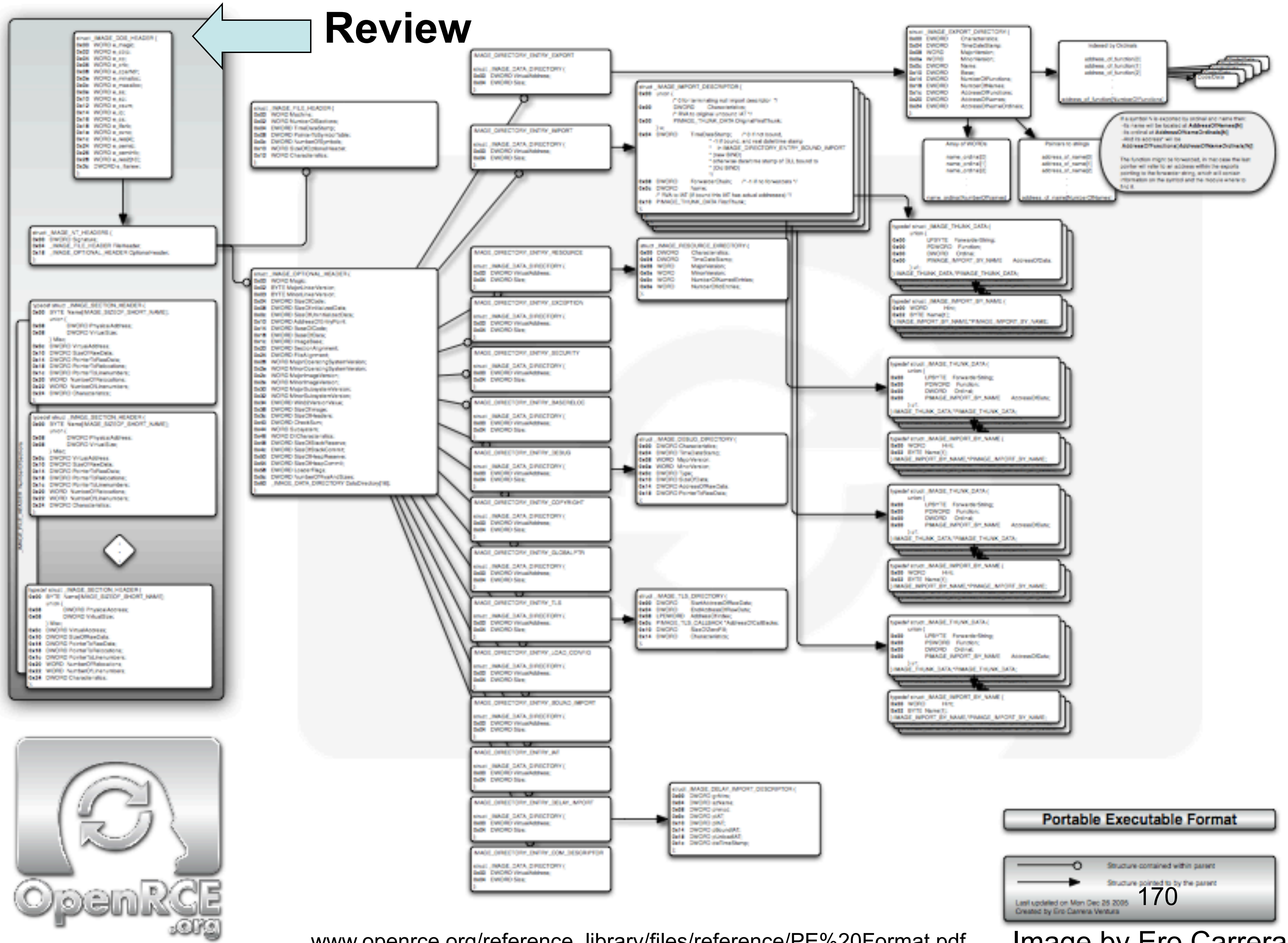
- Most of the rest of the DataDirectory[] entries don't even apply to x86, therefore they have been moved to the backup slides

OS Loader: Load Time

(roughly based on the description of the Win2k loader here:
http://msdn.microsoft.com/en-us/magazine/cc301727.aspx)

1. Copy file from disk to memory per the section headers' specification of file offsets being mapped to virtual addresses. Select randomized base virtual address if ASLR compatible. Set the backend RWX permissions on the virtual memory pages (with NX if asked for.)
2. Fix relocations (if any)
3. Recursively check whether a DLL is already loaded, and if not, load imported DLLs (and any forwarded-to DLLs) and resolve imported function addresses placing them into the IAT. After every DLL is imported, call each DLL's entry point.
4. Resolve any bound imports in the main executable which are out of date.
5. Transfer execution to any TLS callbacks
6. Transfer execution to the executable's entry point specified in the OptionalHeader

Review


Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 26 2005
Created by Ero Carrera Ventura
OpenRCE
.org

www.openrce.org/reference_library/files/reference/PE%20Format.pdf

Image by Ero Carrera