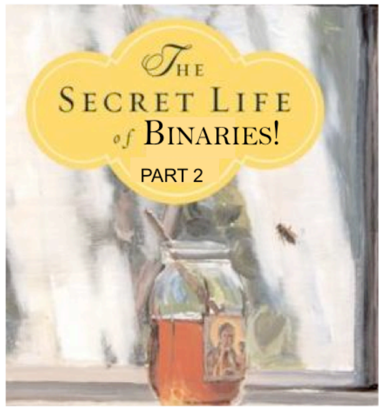


Xeno Kovah – 2012

xkovah at gmail

1

See notes for citation

Image from: http://upload.moldova.org/movie/2007/dec/bee.jpg

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.

2

Attribution conditions: Just state author name and where the slides were obtained from

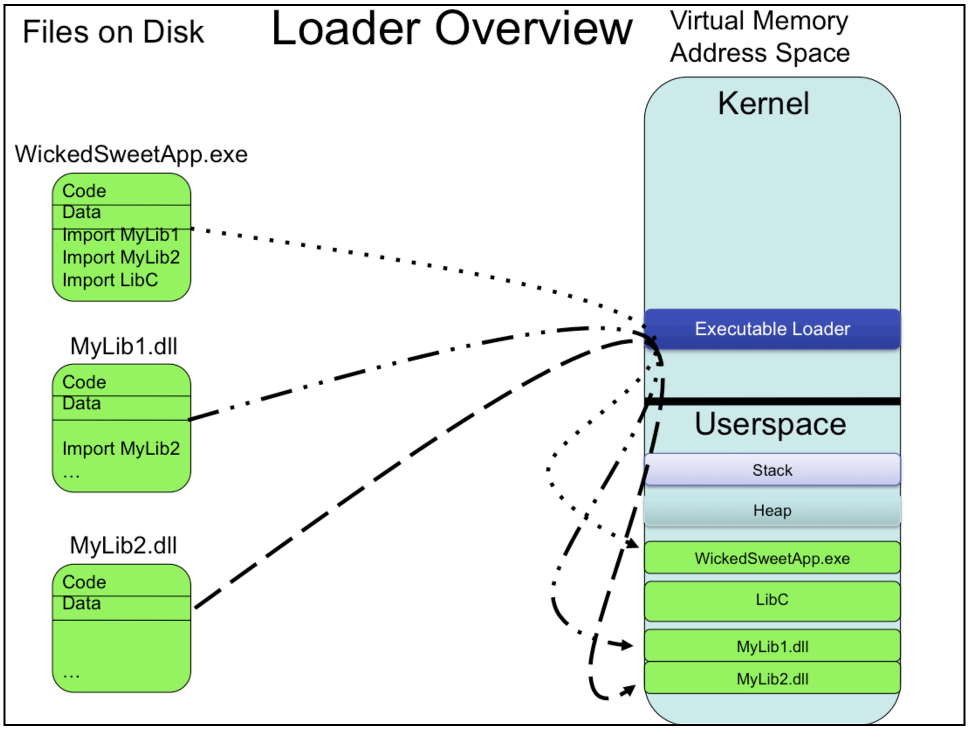
Executable Formats

- 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.

3

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.


Files on Disk
Loader Overview
Virtual Memory Address Space
Kernel
WickedSweetApp.exe
Code
Data
Import MyLib1
Import MyLib2
Import LibC
Executable Loader
MyLib1.dll
Code
Data
Import MyLib2
…
Userspace
Stack
Heap
MyLib2.dll
Code
Data
…
WickedSweetApp.exe
LibC
MyLib1.dll
MyLib2.dll

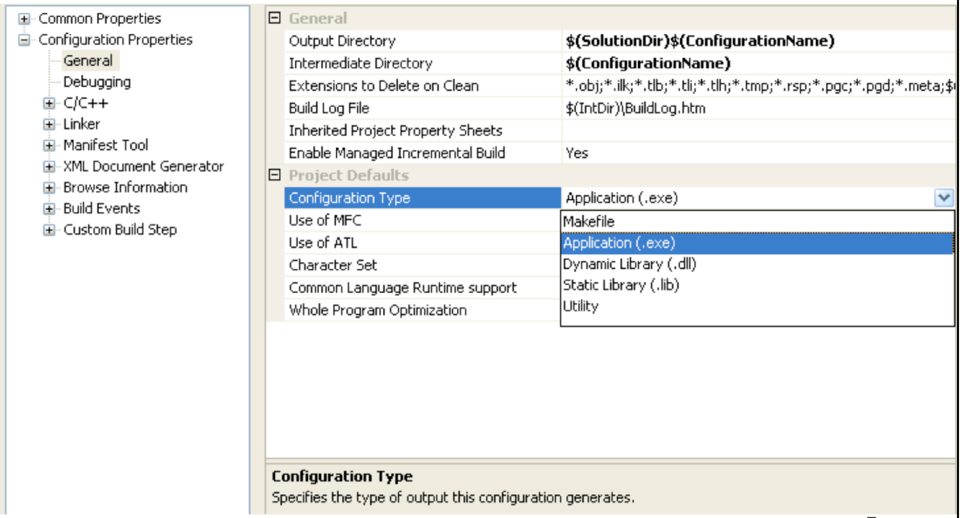
Common Windows PE File Extensions

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

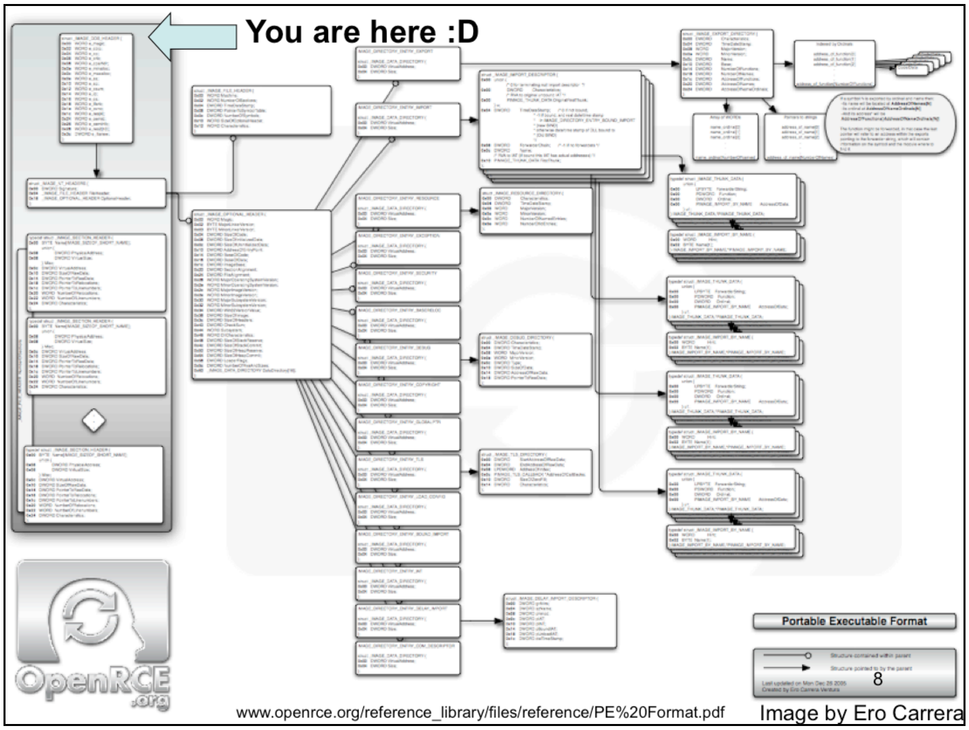
- Note: .lib files (Static Libraries) don't have the same "DOS Header then PE Header" format that the rest of these do.

6

Building Windows Executable, Dynamic Linked Library, Static Library



7



You are here :D
Portable Executable Format
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)

10

Tools: WinDbg

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

11

Terminology

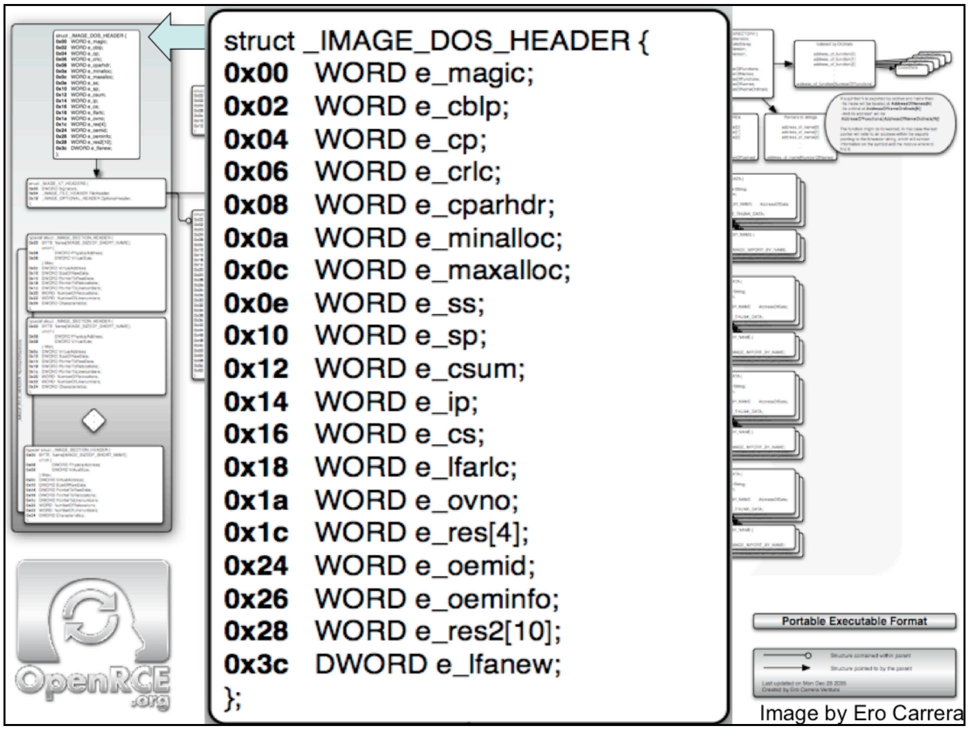
- RVA - Relative Virtual Address. This indicates some displacement relative to the start (base) of a binary in memory.
- AVA – Absolute Virtual Address, more often just "Virtual Address", but I want to be exact. This is a specific address memory where something can be found.
- So if the base is 0x80000000, and the AVA was 0x80001000, then the RVA would be 0x1000.
- If the base is 0x80000000, and the AVA was 0xC123000f, then the RVA would be 0x4123000f.
- RVA = VA – Base
- Windows uses RVAs extensively in the PE format, unlike ELF which uses just AVAs

Terminology 2

- Windows uses the following variable size names:
- CHAR = character = 1 byte
- WORD = word = 2 bytes
  - SHORT = short integer = 2 bytes
- DWORD = double-word = 4 bytes
  - LONG = long integer = 4 bytes
- QWORD = quad-word = 8 bytes
  - LONGLONG = long long integer = 8 bytes

13

NEW 2012



New 2012 -> Q: Ask students what the next offset after 0x3C would be. A: 0x40 (ensures they get what I just said about sizes, and they have their hex math down)

The MS-DOS File Header

(from winnt.h)

BLUE means the stuff we actually care about

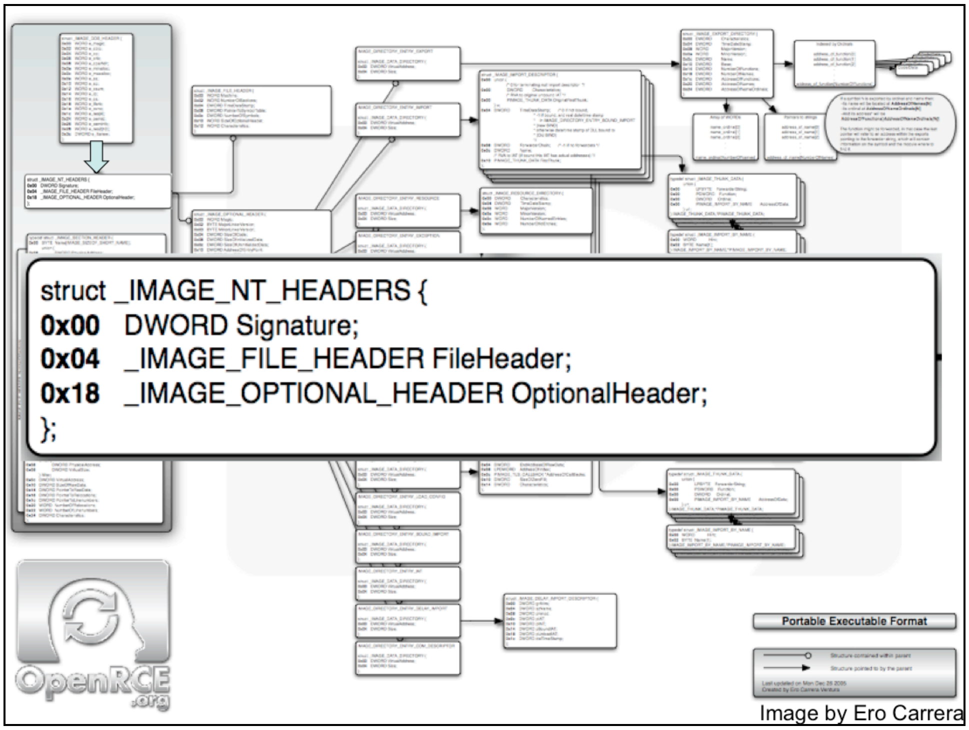
```
typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE header
    WORD   e_magic;                     // Magic number
    WORD   e_cblp;                      // Bytes on last page of file
    WORD   e_cp;                        // Pages in file
    WORD   e_crlc;                      // Relocations
    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;
```

15

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)

16

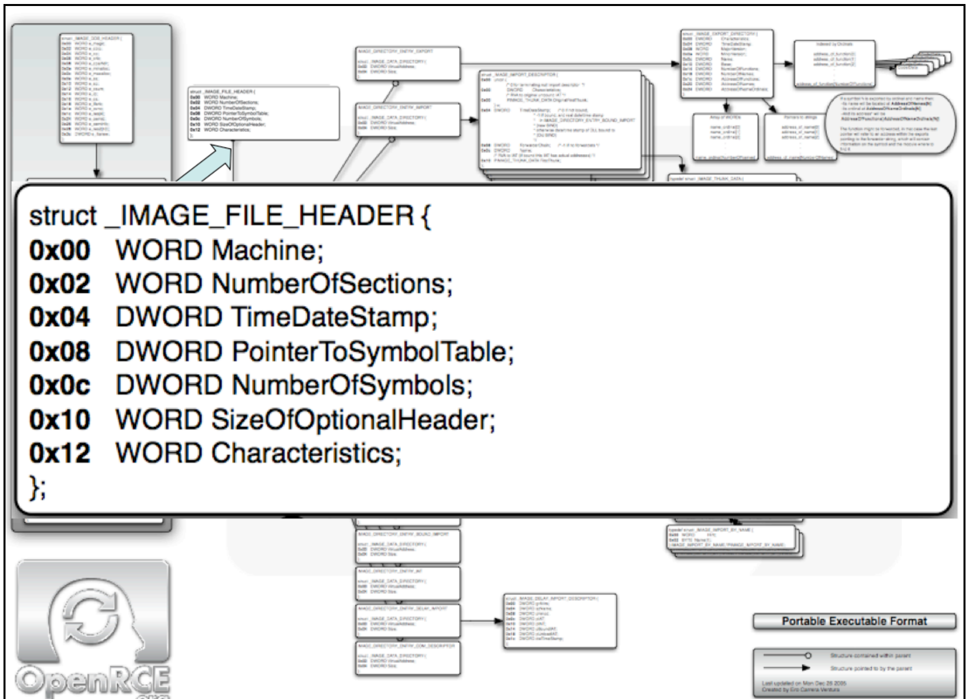

struct _IMAGE_NT_HEADERS {
0x00 DWORD Signature;
0x04 _IMAGE_FILE_HEADER FileHeader;
0x18 _IMAGE_OPTIONAL_HEADER OptionalHeader;
};
Portable Executable Format
Image by Ero Carrera

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




```
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;
```

20

File Header 2

- **Machine** specifies what architecture this is supposed to run on. This is our first indication about 32 or 64 bit binary
- Value of **014C** = x86 binary, aka 32 bit binary, aka **PE32** binary
- Value of **8664** = x86-64 binary, aka AMD64 binary, aka 64 bit binary, aka **PE32+** binary

21

File Header 3

- 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 4

- 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

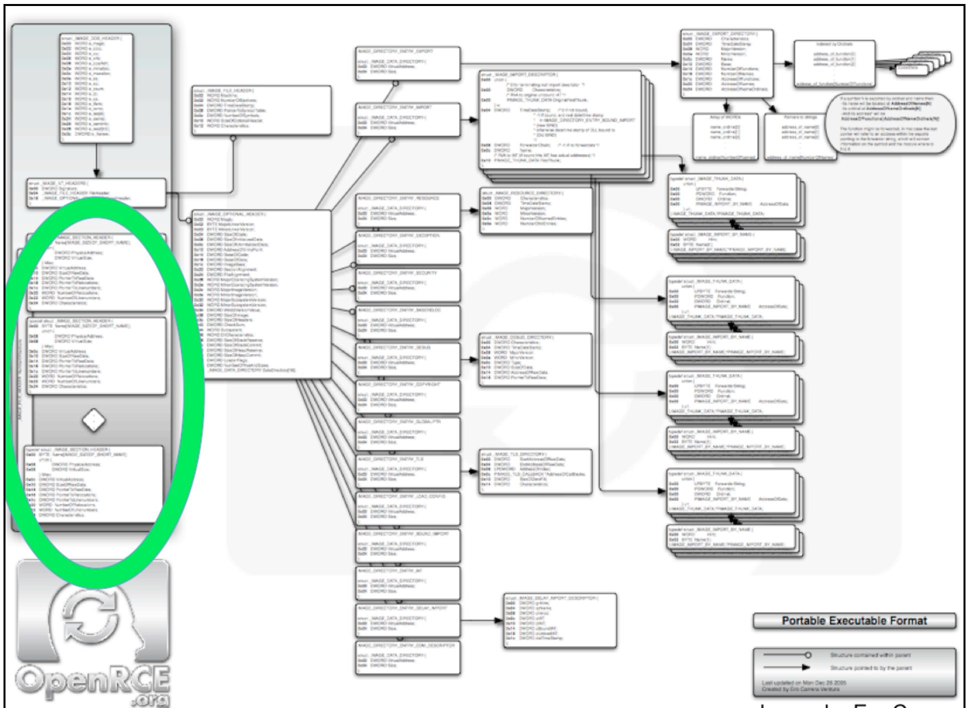


Image by Ero Carrera

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.
#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.
```

(teeheehee)

25

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

26

New 2012 – changed this to a screen shot to save size

From

http://www.defensereview.com/stories/predatorcamo/Predator%20Camo_Large.jpg

http://cognitive-edge.com/uploads/blog/predator-3.jpg

http://media.moddb.com/images/mods/1/12/11314/00004.jpg

http://remingtons.files.wordpress.com/2010/07/arnold-predator.jpg

http://www.trespassmag.com/wp-content/uploads/2010/07/Predators.jpg

http://www.pcgameshardware.com/screenshots/medium/2009/06/aliens-vs-predator-screenshot-02.jpg

http://images.alphacoders.com/178/178993.jpg

http://www.iamexpat.nl/app/webroot/upload/files/Topics/Lifestyle/Whats-on/Guns-n-Roses-guns-n-roses-589484_655_475(1).jpg

<- I spent way too much time on that. Appreciate it ;)

How to play

- Open 2 instances of cmd.exe
  - One will be for independent work, one will be for class-competition
- Start the game in both instances by doing "python BinHunt.py"
- In the independent work one, enter 0 for the mode
- In the class one, enter 2 for the mode

28

See notes for citation

New 2012 – NOTE: I spent way more time on that token than I should have, so you must love and cherish it

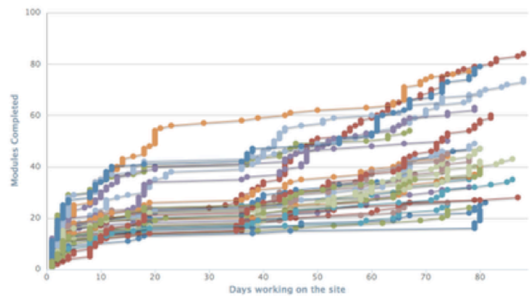
From

http://www.classicplastic.net/dvgi/pics-tokenstilt02.jpg

http://www.classicplastic.net/dvgi/pics-tokensgeneric02.jpg

About This Game

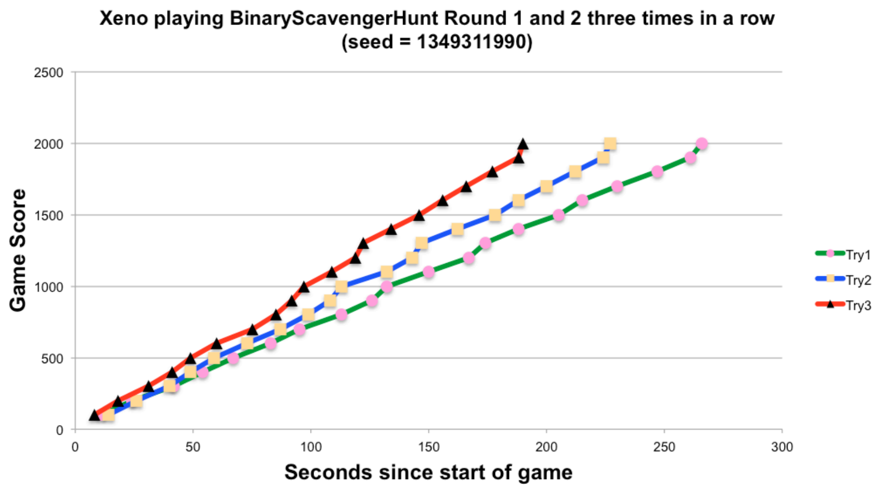
- Part of a larger effort to create games to reinforce material from security classes
- http://code.google.com/p/roxor-arcade/
- Allows for interesting data collection. Inspired by this picture from khanacademy.org/about:



29

Example of me playing the same round multiple times and getting better each time

(I tried not to memorize any of the answers, and go through the motions of looking them up with the tools, I basically just got faster with the tools

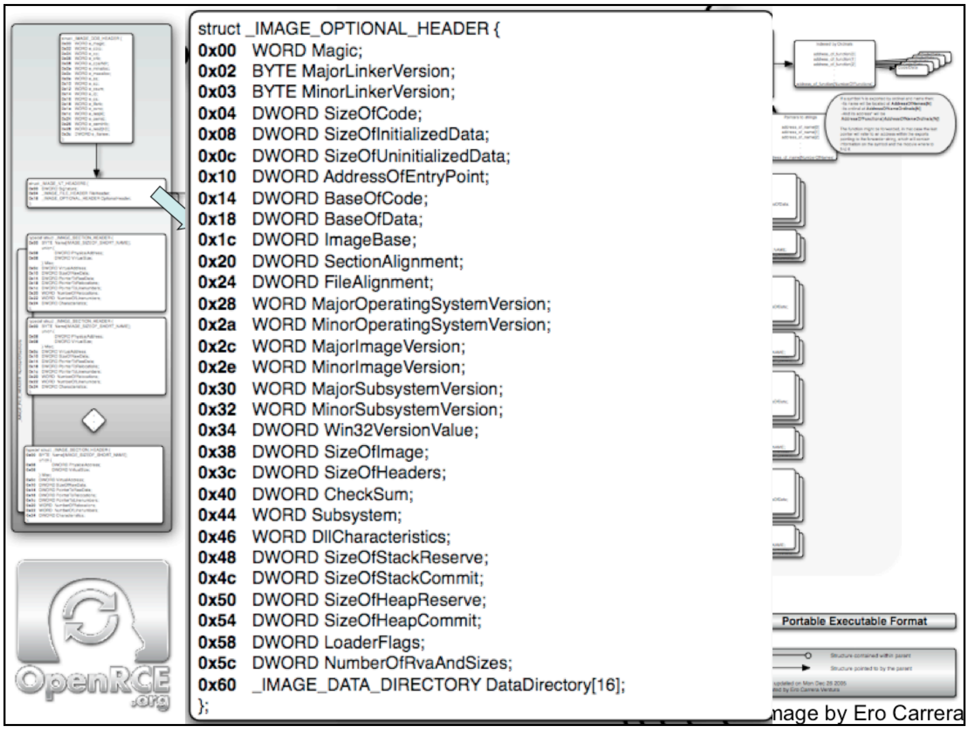


30

Example of Entire First Class to Beta Test BinaryScavengerHunt

- TODO

31




```
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];
};
```

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;
```

33

From winnt.h

```
typedef struct _IMAGE_OPTIONAL_HEADER64 {
    WORD        Magic;
    BYTE        MajorLinkerVersion;
    BYTE        MinorLinkerVersion;
    DWORD       SizeOfCode;
    DWORD       SizeOfInitializedData;
    DWORD       SizeOfUninitializedData;
    DWORD       AddressOfEntryPoint;
    DWORD       BaseOfCode;
    ULONGLONG   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;
    ULONGLONG   SizeOfStackReserve;
    ULONGLONG   SizeOfStackCommit;
    ULONGLONG   SizeOfHeapReserve;
    ULONGLONG   SizeOfHeapCommit;
    DWORD       LoaderFlags;
    DWORD       NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER64, *PIMAGE_OPTIONAL_HEADER64;
```

34

Optional Header 0

- It's not at all optional ;)
- **Magic** is the true determinant of whether this is a PE32 or PE32+ binary
- Depending on the value, the optional header will be interpreted as having a couple 32 or 64 bit fields.
- 0x10C = 32 bit, PE32
- 0x20B = 64 bit, PE32+

35

Optional Header 1

- 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

36

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

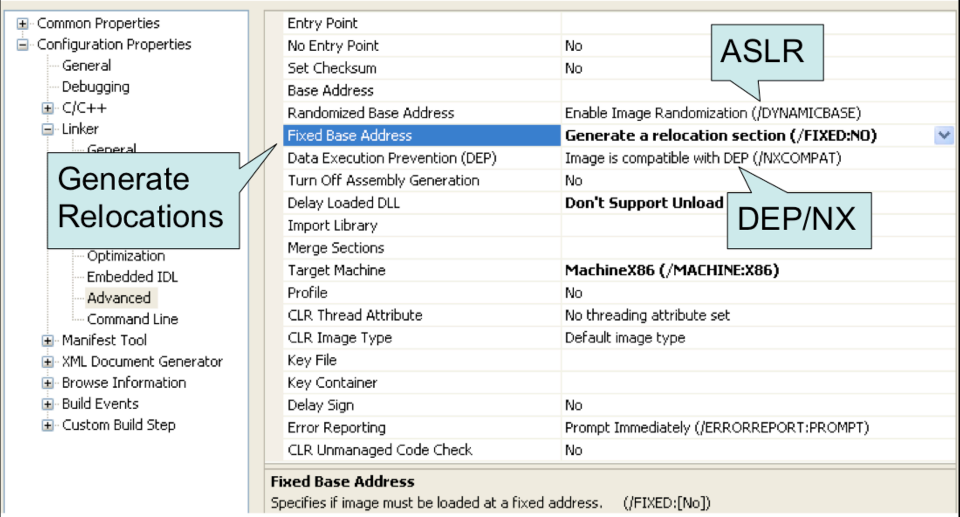
Optional Header 4

- **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

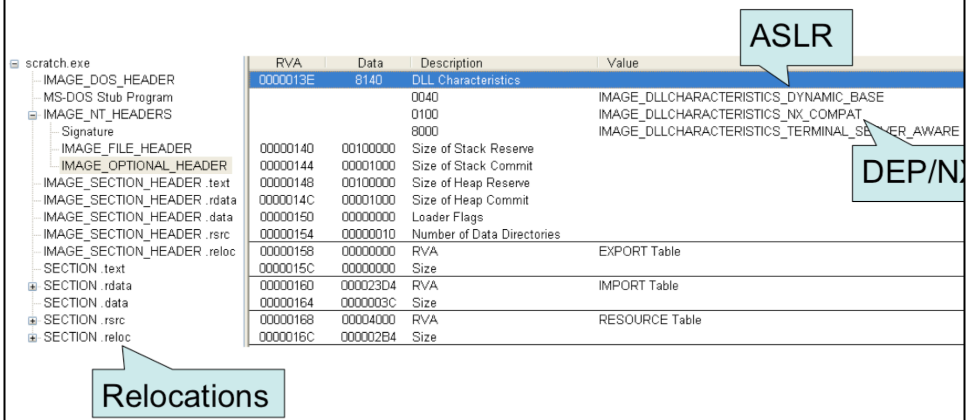
- /DYNAMICBASE – Mark the properties to indicate that this executable will work fine with Address Space Layout Randomization (ASLR)
- /FIXED:NO – This will force the linker to generate relocations information for an executable, so that it is capable of having its base address modified by ASLR (otherwise 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

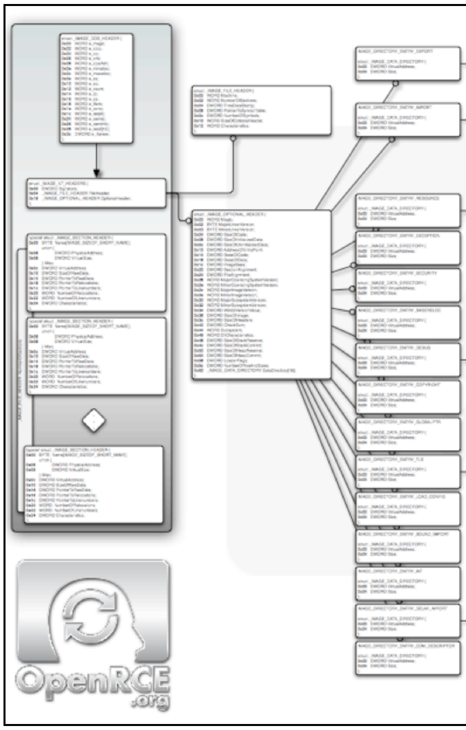


41

ASLR & DEP/NX in the Binary



42



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

44

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.

45

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;
```

Get your geek on

- Play through round 2 on your own, and then wait for the seed for the class deathmatch
- You can skip to level 2 by starting the game with "python BinHunt.py 2"

47

See notes for citation

New 2012 – NOTE: I spent way more time on that token than I should have, so you must love and cherish it

From

http://www.classicplastic.net/dvgi/pics-tokenstilt02.jpg

http://www.classicplastic.net/dvgi/pics-tokensgeneric02.jpg

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")

48

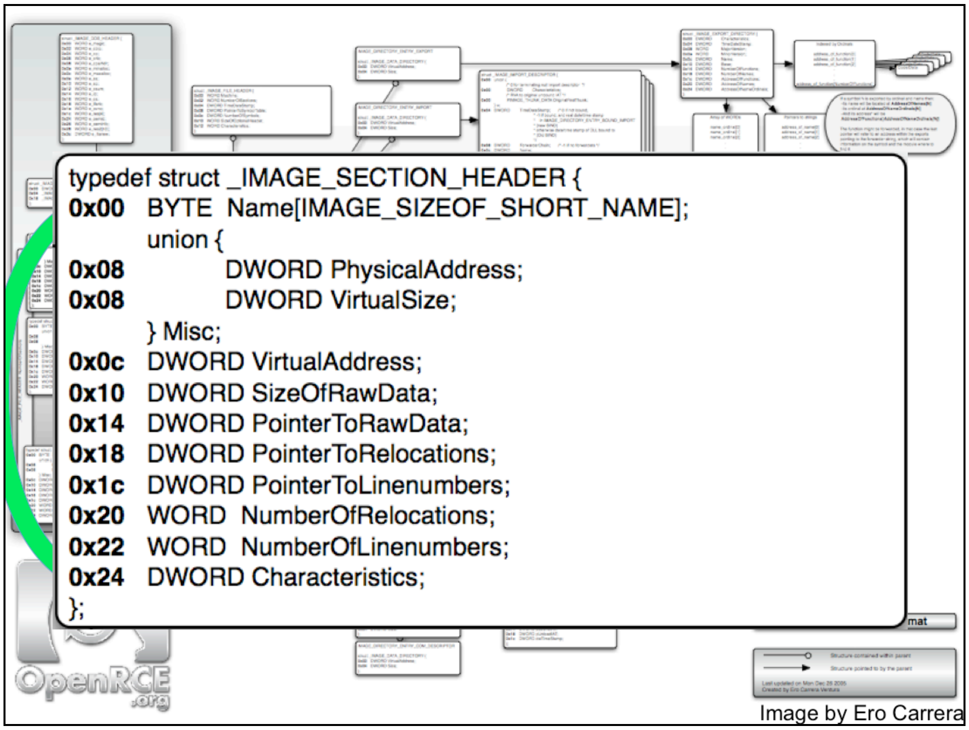
Sections 2

- 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
- .edata = export information

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.

50


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;
```

52

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.

53

Section Header 2

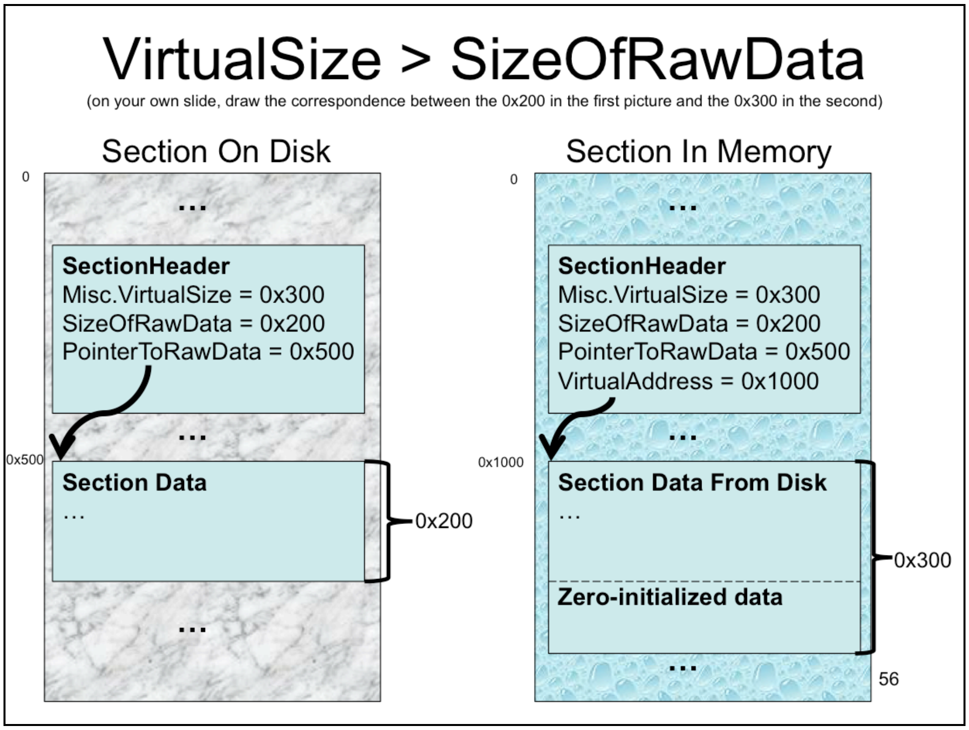
- **Name[8]** is a byte array of ASCII characters. It is **NOT** guaranteed to be null-terminated. So if you're trying to parse a PE file yourself you need to be aware of that.
- **VirtualAddress** is the RVA of the section relative to OptionalHeader.ImageBase
- **PointerToRawData** is a relative offset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

54

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.

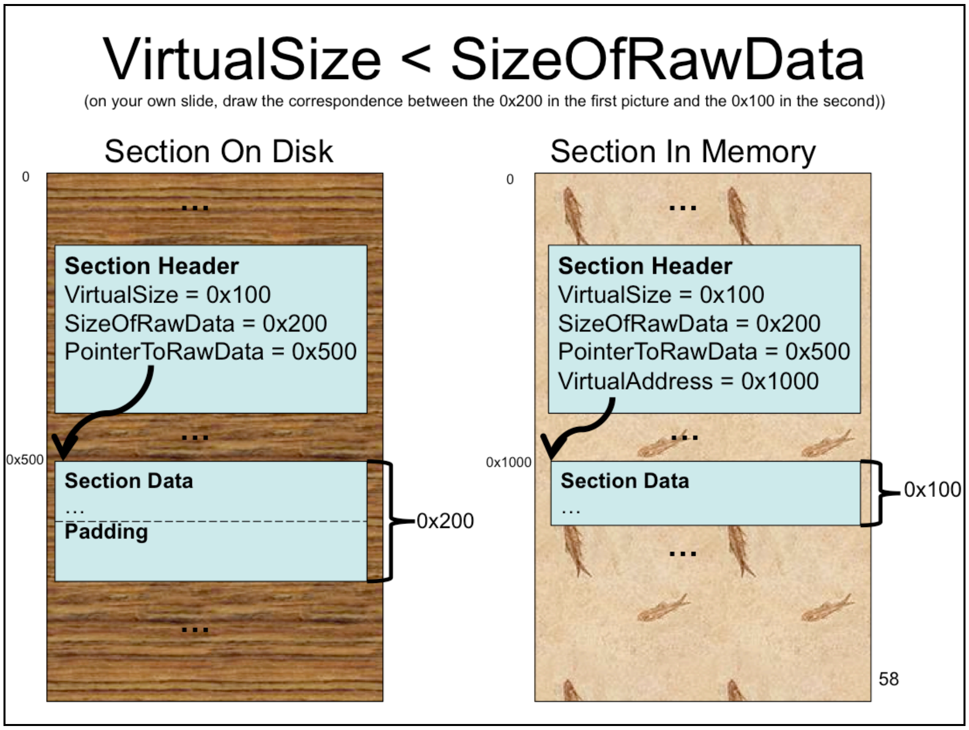
55



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
Section In Memory
0
…
SectionHeader
Misc.VirtualSize = 0x300
SizeOfRawData = 0x200
PointerToRawData = 0x500
0x500
Section Data
0x200
0
SectionHeader
Misc.VirtualSize = 0x300
SizeOfRawData = 0x200
PointerToRawData = 0x500
VirtualAddress = 0x1000
0x1000
Section Data From Disk
Zero-initialized data
0x300
56

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
…
58

Section Header 5

(from winnt.h)

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

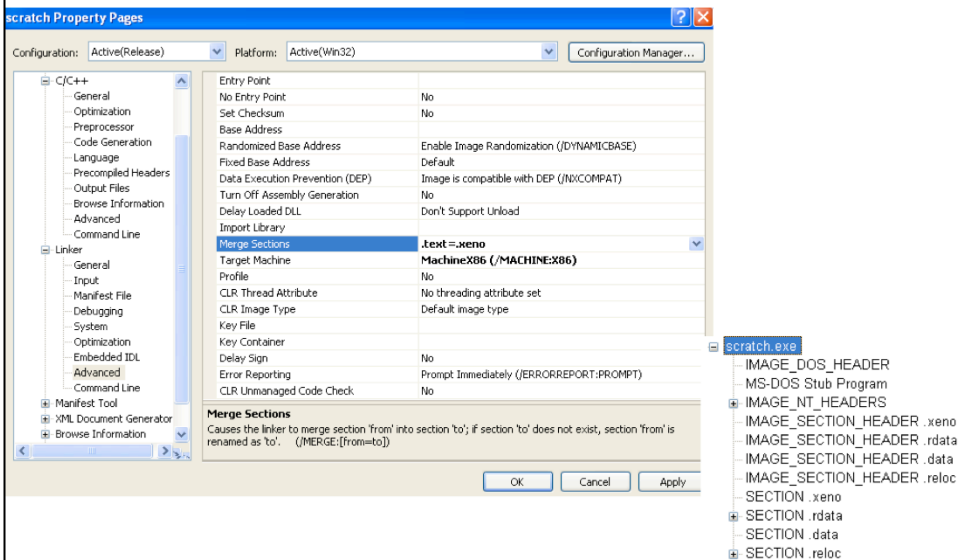
```
#define IMAGE_SCN_CNT_CODE                    0x00000020
// Section contains code.
#define IMAGE_SCN_CNT_INITIALIZED_DATA        0x00000040
// Section contains initialized data.
#define IMAGE_SCN_CNT_UNINITIALIZED_DATA      0x00000080
// Section contains uninitialized data.
#define IMAGE_SCN_MEM_DISCARDABLE             0x02000000
//Do not cache this section
#define IMAGE_SCN_MEM_NOT_CACHED                 0x04000000
// 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.
```

59

Section Header

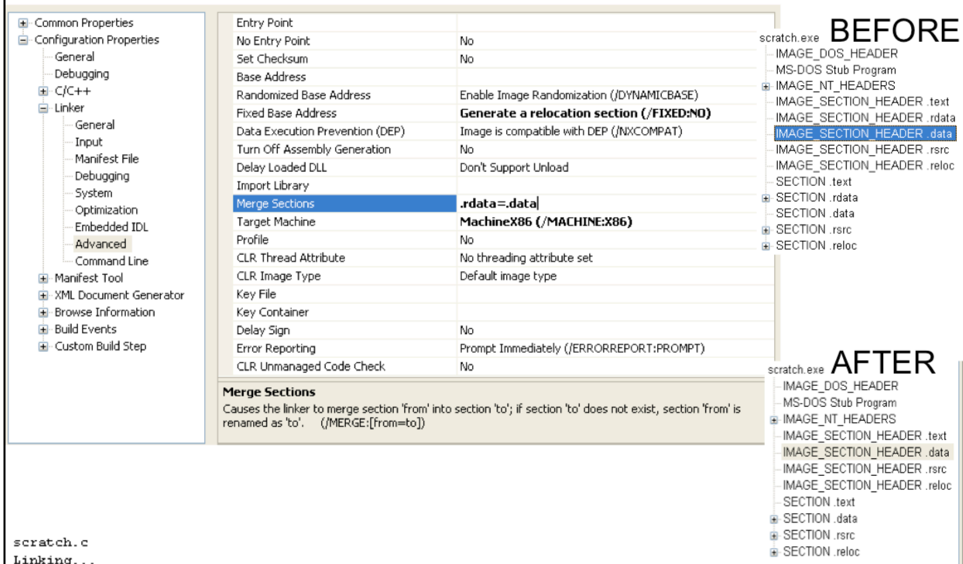
- PointerToRelocations, PointerToLinenumbers, NumberOfRelocations, NumberOfLinenumbers aren't used anymore

Renaming Sections



61

Merge Sections




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

62

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;
```

Get your geek on

- Play through round 3 on your own, and then wait for the seed for the class deathmatch
- You can skip to level 3 by starting the game with "python BinHunt.py 3"

64

See notes for citation

New 2012 – NOTE: I spent way more time on that token than I should have, so you must love and cherish it

From

http://www.classicplastic.net/dvgi/pics-tokenstilt02.jpg

http://www.classicplastic.net/dvgi/pics-tokensgeneric02.jpg

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.

65

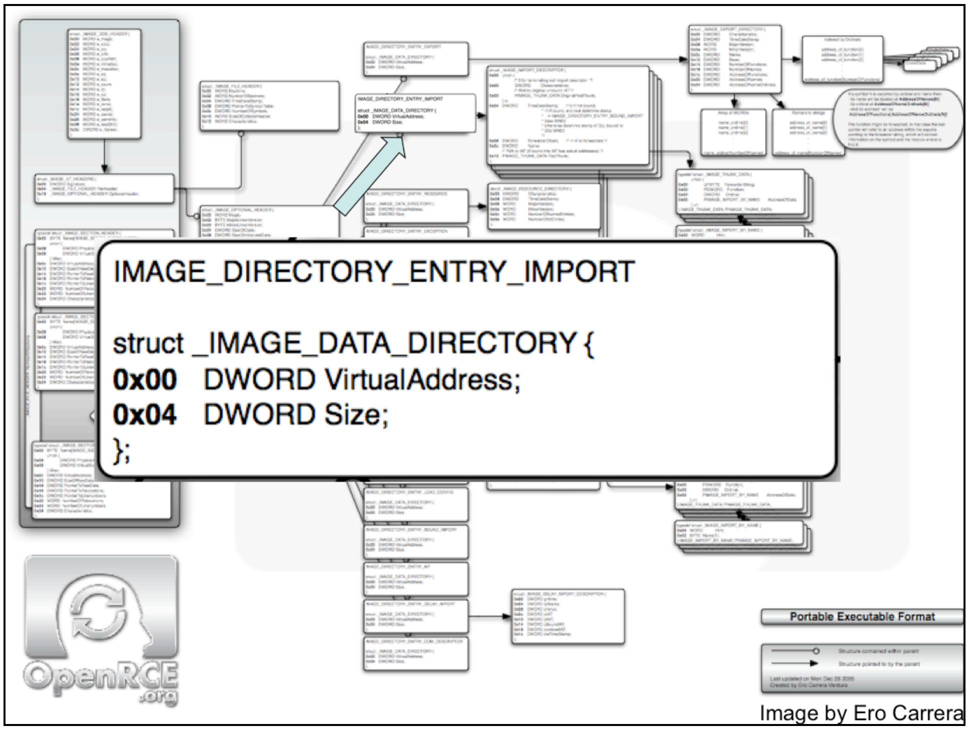
Calling Imported Functions

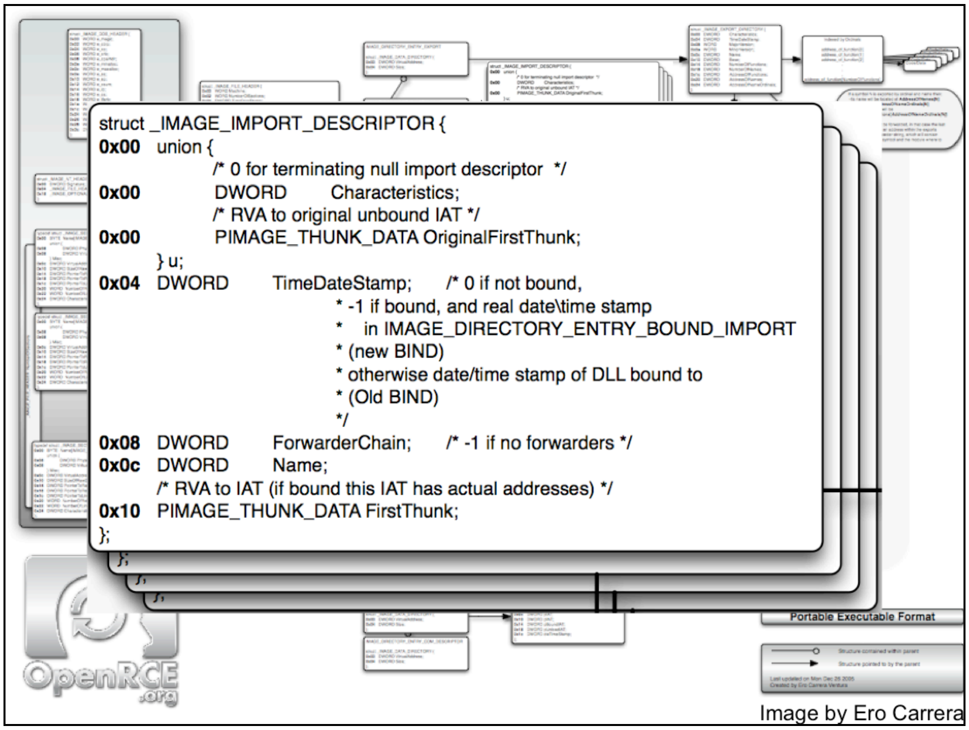
- As a programmer, this is transparent to you, but what sort of assembly does the compiler actually generate when you call an imported function like printf()?
- We can use the handy-dandy HelloWorld.c to find out quickly.

```
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)

66


IMAGE_DIRECTORY_ENTRY_IMPORT
struct _IMAGE_DATA_DIRECTORY {
0x00 DWORD VirtualAddress;
0x04 DWORD Size;
};
Portable Executable Format
OpenRCE.org
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;
};
Portable Executable Format
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.

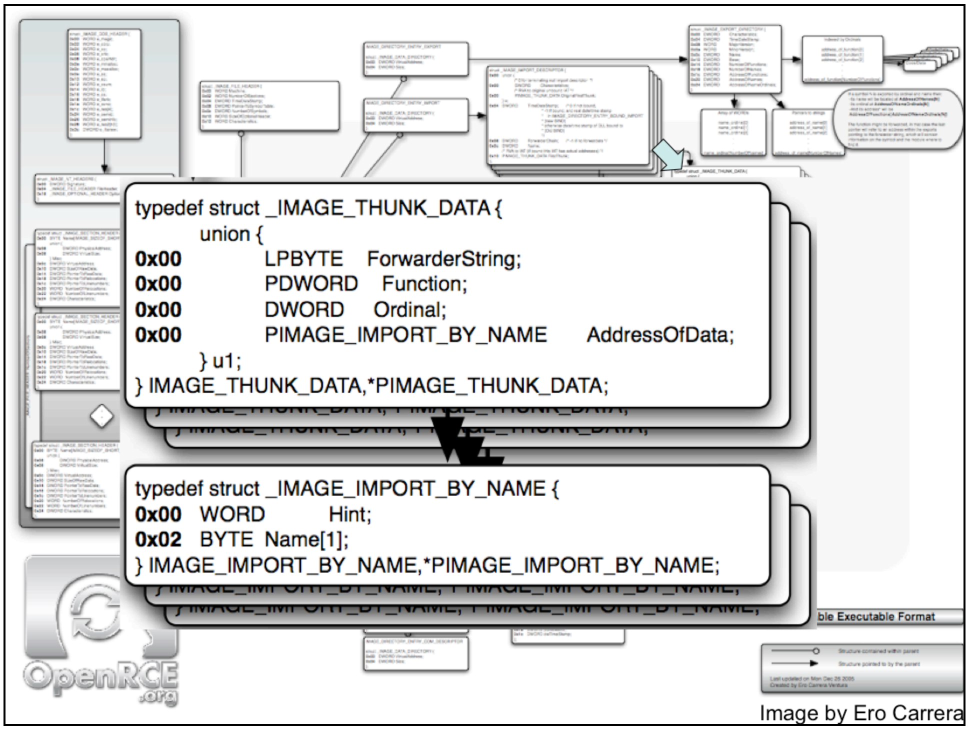
Import Descriptor 2

- **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)

71


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)

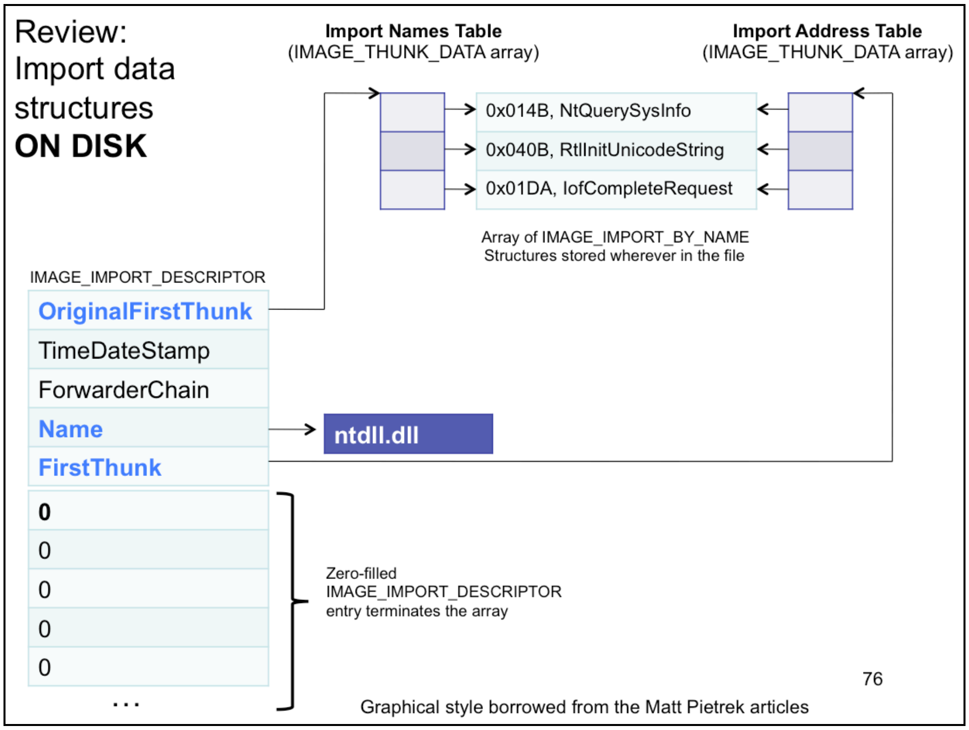
```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;
    BYTE    Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

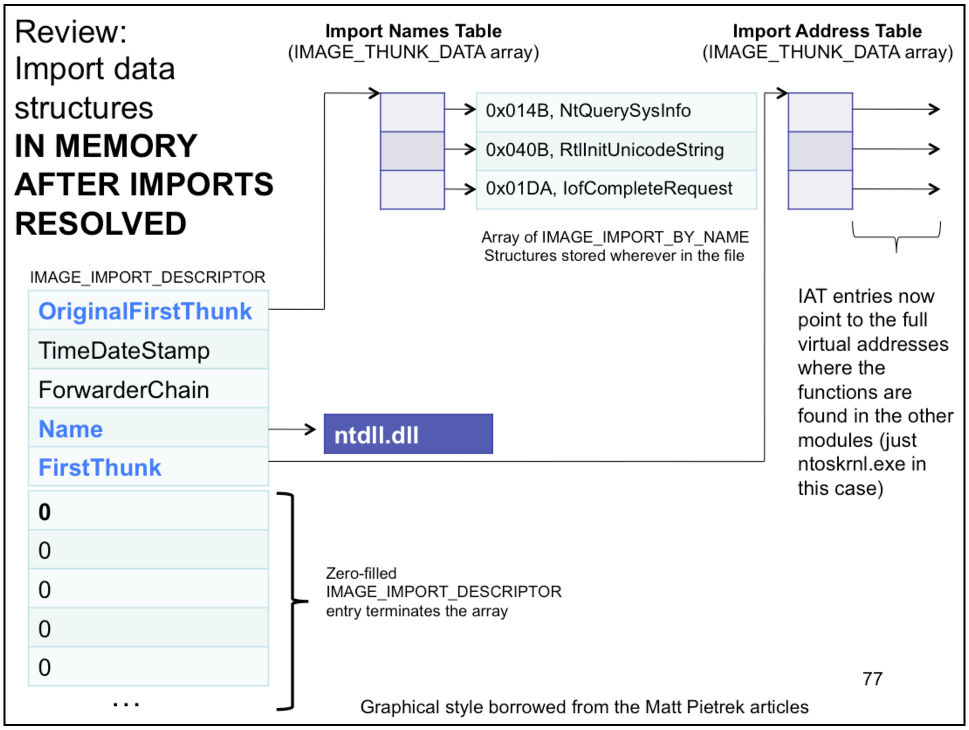
- **Hint** specifies a possible “ordinal” of an imported function. Talked about later, when we talk about exports, but basically it's just a way to look up the function by an index rather than a name.
- **Name** on the other hand is to look up the function by name. It's not one byte long, it's a null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

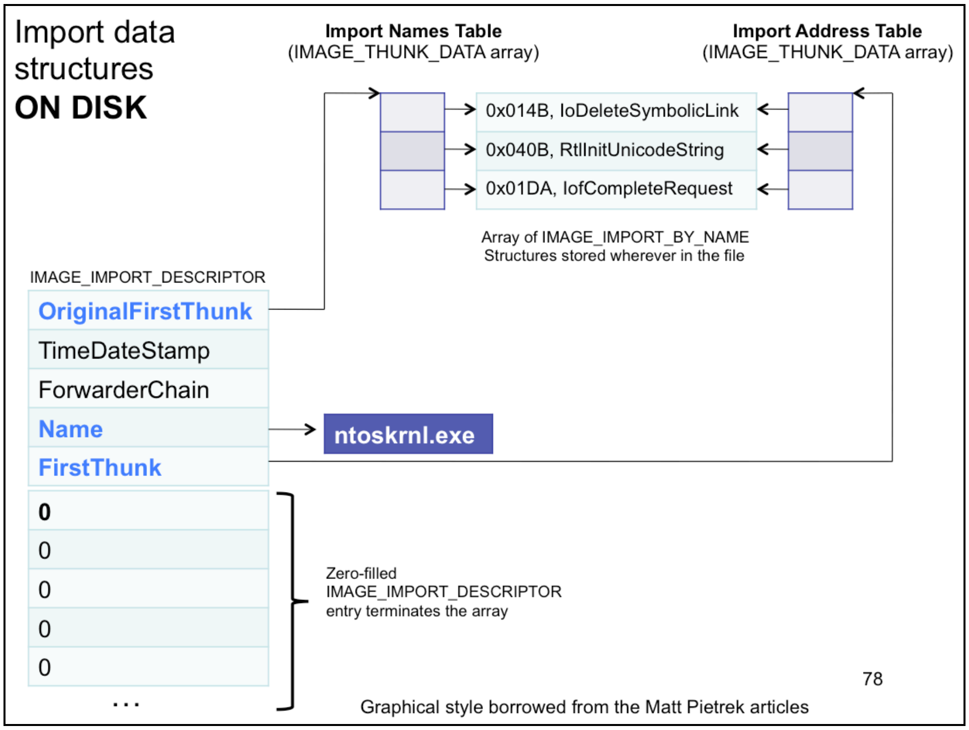
74

On the impersistence of being: INT vs IAT

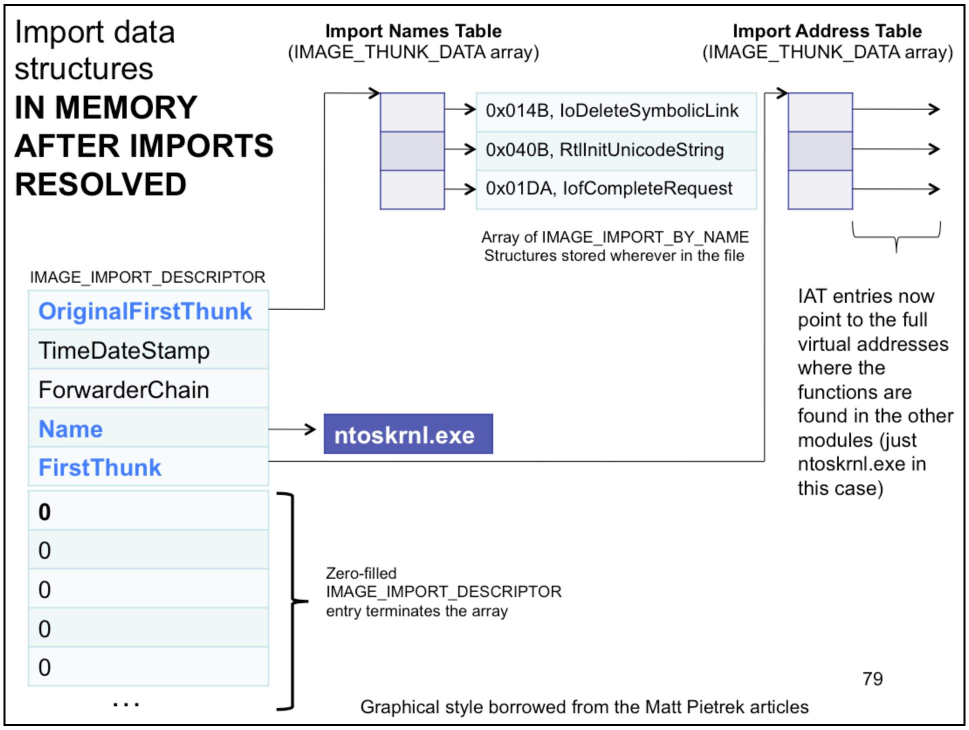
- The **INT** IMAGE_THUNK_DATA structures are always interpreted as pointing at IMAGE_IMPORT_BY_NAME structures, that is they are **u1.AddressOfData**, the RVA of an IMAGE_IMPORT_BY_NAME.
- The **IAT** IMAGE_THUNK_DATA structures ***start out*** are all interpreted as the **u1.AddressOfData**, but once 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**.


Review:
Import data
structures
ON DISK
Import Names Table
(IMAGE_THUNK_DATA array)
Import Address Table
(IMAGE_THUNK_DATA array)
0x014B, NtQuerySysInfo
0x040B, RtlInitUnicodeString
0x01DA, IofCompleteRequest
Array of IMAGE_IMPORT_BY_NAME
Structures stored wherever in the file
IMAGE_IMPORT_DESCRIPTOR
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
ntdll.dll
0
0
0
0
0
...
Zero-filled
IMAGE_IMPORT_DESCRIPTOR
entry terminates the array
76
Graphical style borrowed from the Matt Pietrek articles


Review:
Import data
structures
IN MEMORY
AFTER IMPORTS
RESOLVED
Import Names Table
(IMAGE_THUNK_DATA array)
Import Address Table
(IMAGE_THUNK_DATA array)
0x014B, NtQuerySysInfo
0x040B, RtlInitUnicodeString
0x01DA, IofCompleteRequest
Array of IMAGE_IMPORT_BY_NAME
Structures stored wherever in the file
IMAGE_IMPORT_DESCRIPTOR
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
ntdll.dll
IAT entries now point to the full virtual addresses where the functions are found in the other modules (just ntoskrnl.exe in this case)
0
0
0
0
0
...
Zero-filled
IMAGE_IMPORT_DESCRIPTOR
entry terminates the array
77
Graphical style borrowed from the Matt Pietrek articles



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
78
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)
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
IAT entries now point to the full virtual addresses where the functions are found in the other modules (just ntoskrnl.exe in this case)
0
0
0
0
0
...
Zero-filled
IMAGE_IMPORT_DESCRIPTOR
entry terminates the array
79
Graphical style borrowed from the Matt Pietrek articles

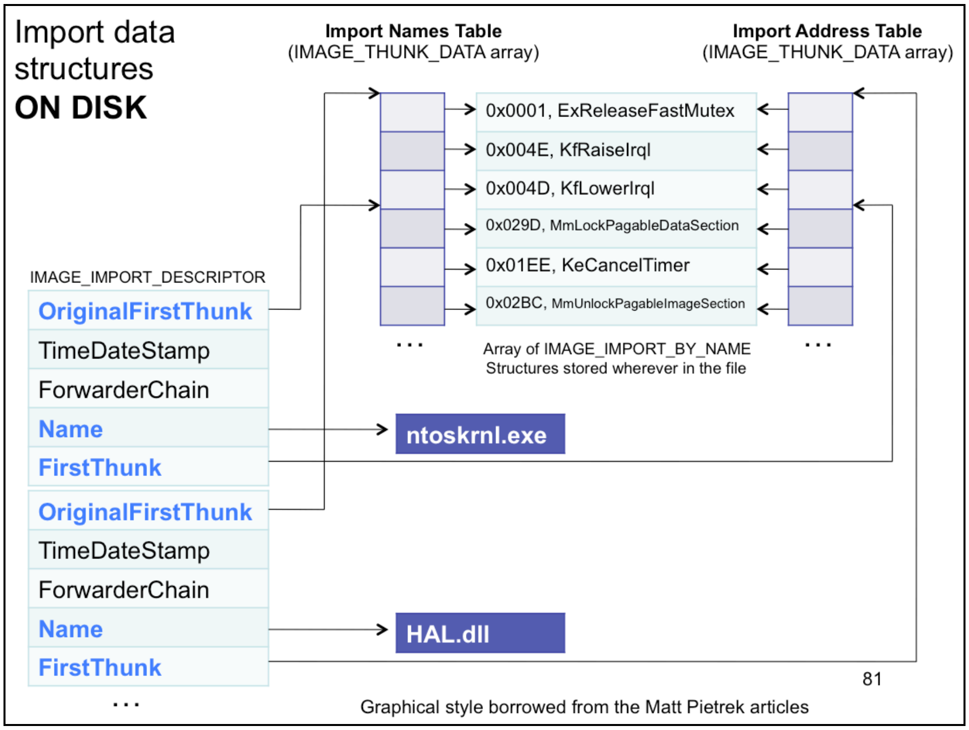
Look through null.sys

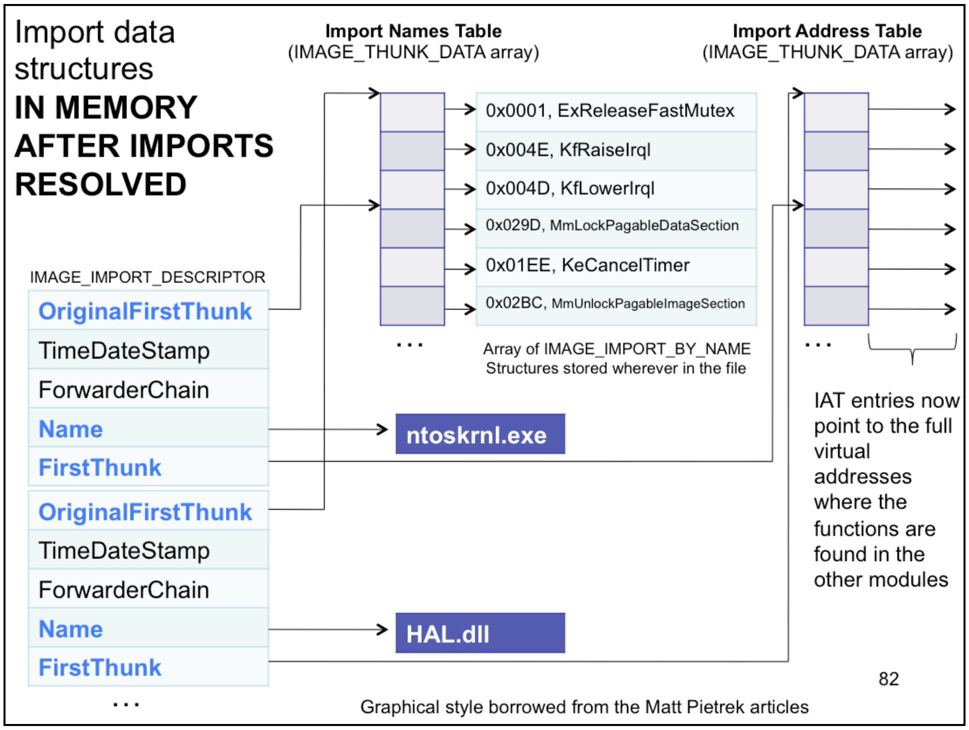
(note to self: start from the data directory)

- 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		

80


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
ntoskrnl.exe
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
HAL.dll
81
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)
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
IAT entries now point to the full virtual addresses where the functions are found in the other modules
82
Graphical style borrowed from the Matt Pietrek articles

Look through beep.sys

- 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
    - IMPORT Directory Table
    - IMPORT Name Table
    - IMPORT Hints/Names & DLL Names
  - SECTION .rsrc
  - SECTION .reloc

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		

83

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

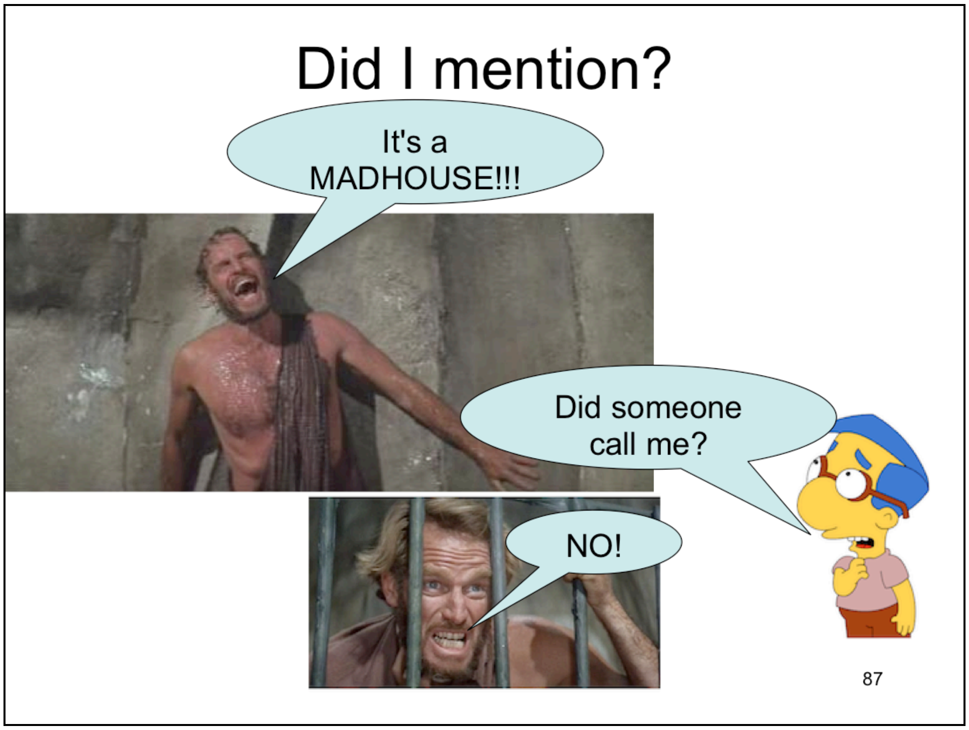
hal then nt, no special significance, just sayin' it's backwards from the previous

Lab: appverif.exe

- appverif.exe was chosen because it has only "normal" imports; no “bound” or “delayed” imports as will be talked about later
- View Imports of C:\Windows \SysWOW64\appverif.exe with PEView
- View imports in memory by attaching with WinDbg

The WOW Effect

- On Win 7 x64…
  - C:\Windows\System32 = where the 64 bit binaries are stored
  - C:\Windows\SysWOW64 = where the 32 bit binaries are stored.
    - Try opening C:\Windows\SysWOW
    - 32 bit executables, like PEView currently is, will open SysWOW64 instead of System32
  - C:\Windows\Sysnative = how you can force 32 bit executables to find the 64 bit executables to find the 64 bit executables
- For more: http://www.cert.at/static/downloads/papers/cert.at-the_wow_effect.pdf

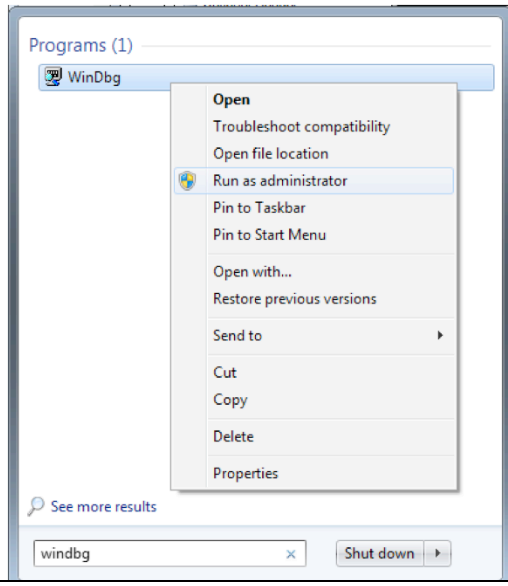


http://s1.picofile.com/file/6417096576/mad_house.jpg

http://www.staceyreid.com/news/wp-content/uploads/2011/08/milhouse.gif

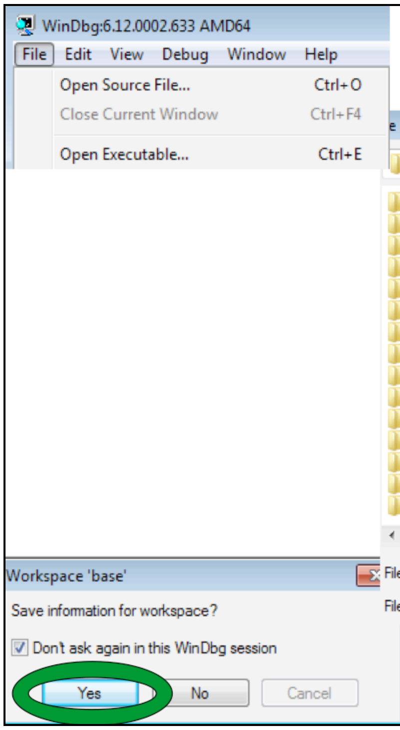
http://mimg.ugo.com/201111/9/0/1/214109/cuts/brighteyes_528x297.jpg

Open WinDbg as Administrator

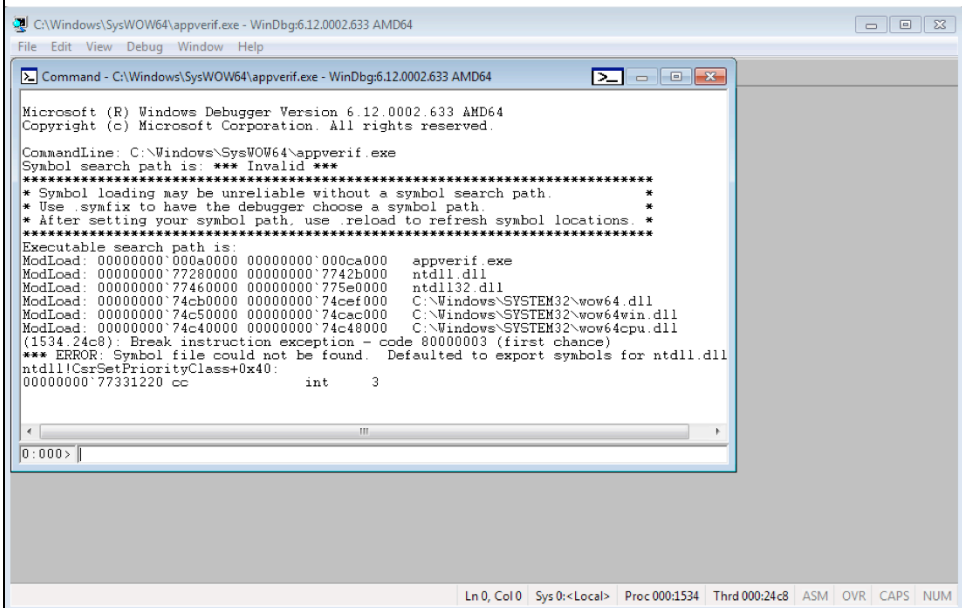


88

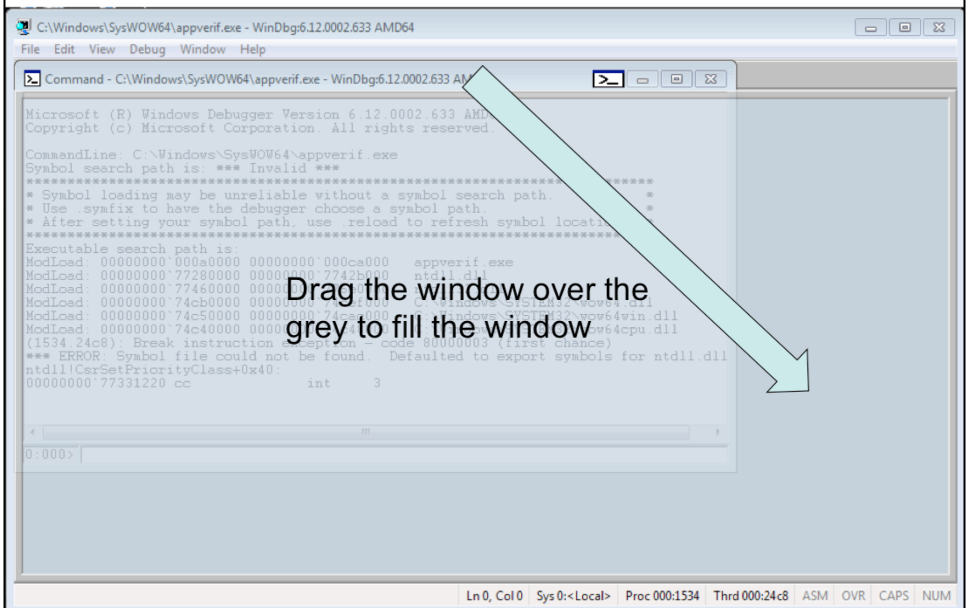
Open executable



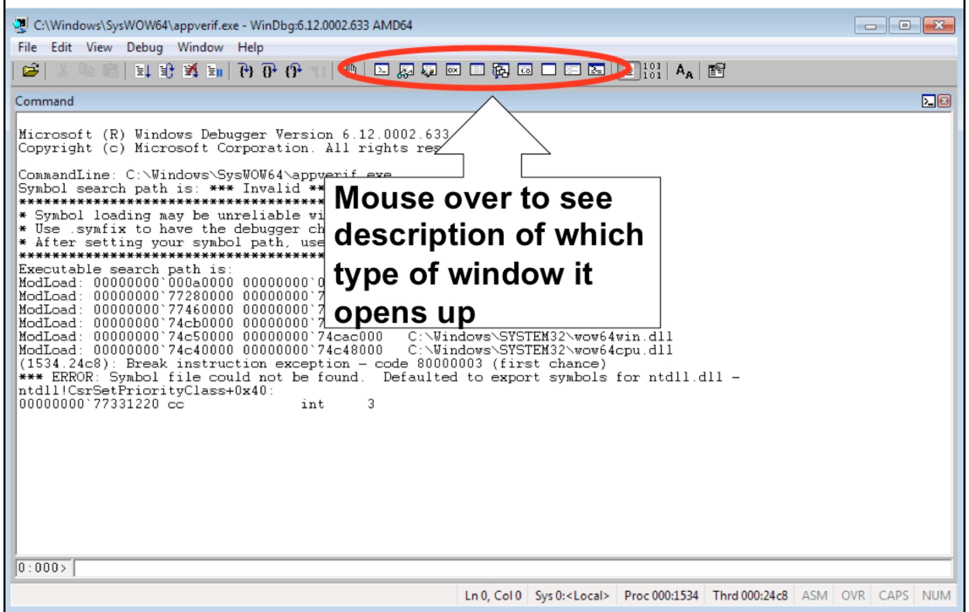
89


C:\Windows\SysWOW64\appverif.exe - WinDbg:6.12.0002.633 AMD64
File Edit View Debug Window Help
Command - C:\Windows\SysWOW64\appverif.exe - WinDbg:6.12.0002.633 AMD64
Microsoft (R) Windows Debugger Version 6.12.0002.633 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.
CommandLine: C:\Windows\SysWOW64\appverif.exe
Symbol search path is: *** Invalid ***
****
* Symbol loading may be unreliable without a symbol search path. *
* Use .symfix to have the debugger choose a symbol path. *
* After setting your symbol path, use .reload to refresh symbol locations. *
****
Executable search path is:
ModLoad: 00000000`000a0000 00000000`000ca000 appverif.exe
ModLoad: 00000000`77280000 00000000`7742b000 ntdll.dll
ModLoad: 00000000`77460000 00000000`775e0000 ntdll32.dll
ModLoad: 00000000`74cb0000 00000000`74cef000 C:\Windows\SYSTEM32\wow64.dll
ModLoad: 00000000`74c50000 00000000`74cac000 C:\Windows\SYSTEM32\wow64win.dll
ModLoad: 00000000`74c40000 00000000`74c48000 C:\Windows\SYSTEM32\wow64cpu.dll
(1534.24c8): Break instruction exception - code 80000003 (first chance)
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntdll.dll
ntdll!CsrSetPriorityClass+0x40:
00000000`77331220 cc int 3
0:000>
Ln 0, Col 0 Sys 0:<Local> Proc 000:1534 Thrd 000:24c8 ASM OVR CAPS NUM

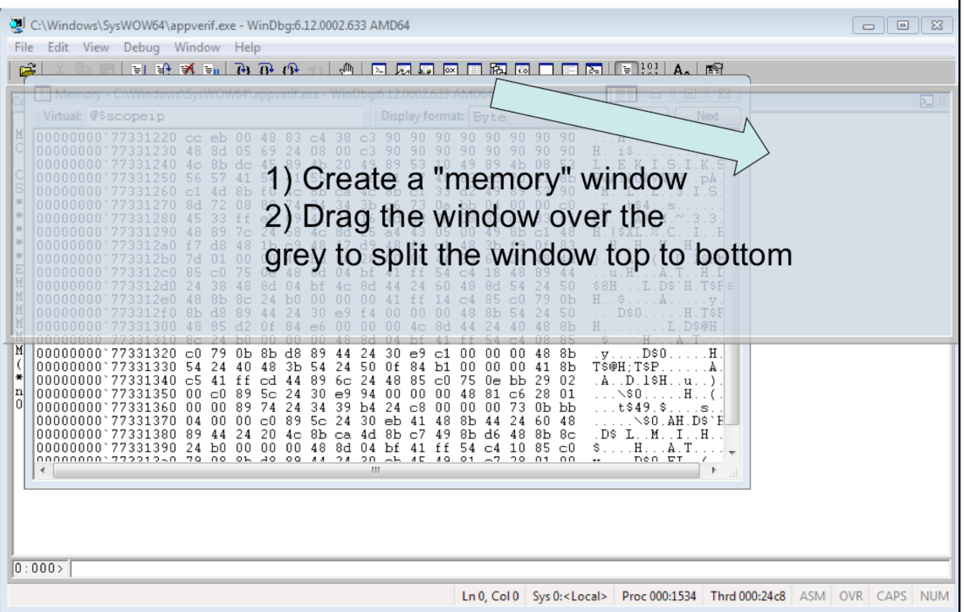
90


C:\Windows\SysWOW64\appverif.exe - WinDbg:6.12.0002.633 AMD64
File Edit View Debug Window Help
Command - C:\Windows\SysWOW64\appverif.exe - WinDbg:6.12.0002.633 AMD64
Drag the window over the
grey to fill the window
Ln 0, Col 0
Sys 0:<Local>
Proc 000:1534
Thrd 000:24c8
ASM
OVR
CAPS
NUM

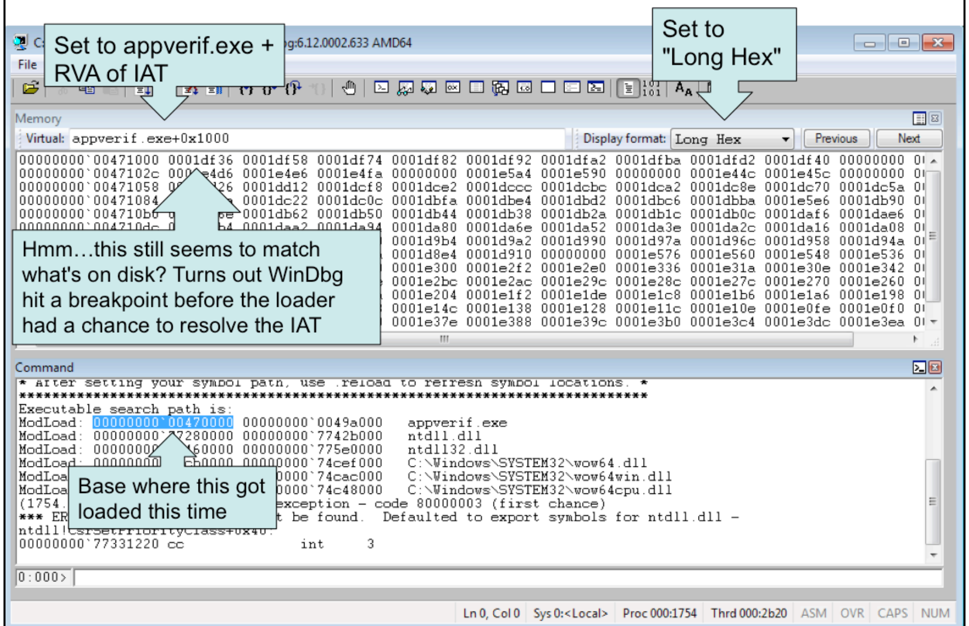
91



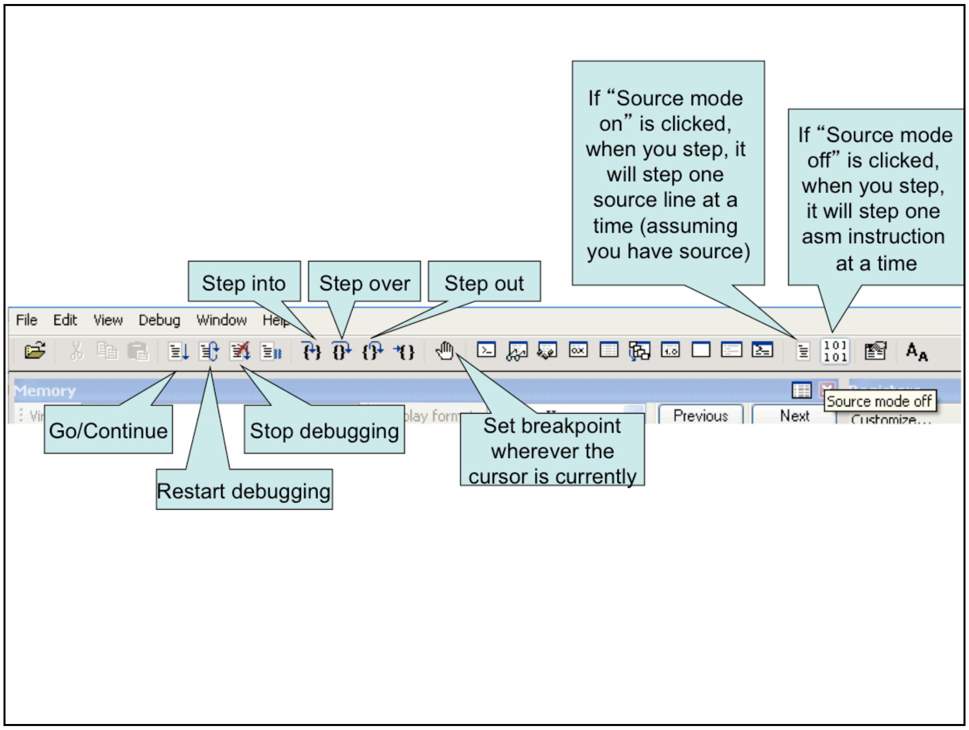
C:\Windows\SysWOW64\appverif.exe - WinDbg:6.12.0002.633 AMD64
File Edit View Debug Window Help
Command
Microsoft (R) Windows Debugger Version 6.12.0002.633
Copyright (c) Microsoft Corporation. All rights res
CommandLine: C:\Windows\SysWOW64\appverif.exe
Symbol search path is: *** Invalid **
* Symbol loading may be unreliable wi
* Use .symfix to have the debugger ch
* After setting your symbol path, use
Executable search path is:
ModLoad: 00000000`000a0000 00000000`0
ModLoad: 00000000`77280000 00000000`7
ModLoad: 00000000`77460000 00000000`7
ModLoad: 00000000`74cb0000 00000000`7
ModLoad: 00000000`74c50000 00000000`74cac000 C:\Windows\SYSTEM32\wow64win.dll
ModLoad: 00000000`74c40000 00000000`74c48000 C:\Windows\SYSTEM32\wow64cpu.dll
(1534.24c8): Break instruction exception - code 80000003 (first chance)
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntdll.dll -
ntdll!CsrSetPriorityClass+0x40:
00000000`77331220 cc int 3
0:000>
Ln 0, Col 0 Sys 0:<Local> Proc 000:1534 Thrd 000:24c8 ASM OVR CAPS NUM
Mouse over to see description of which type of window it opens up


1) Create a "memory" window
2) Drag the window over the
grey to split the window top to bottom

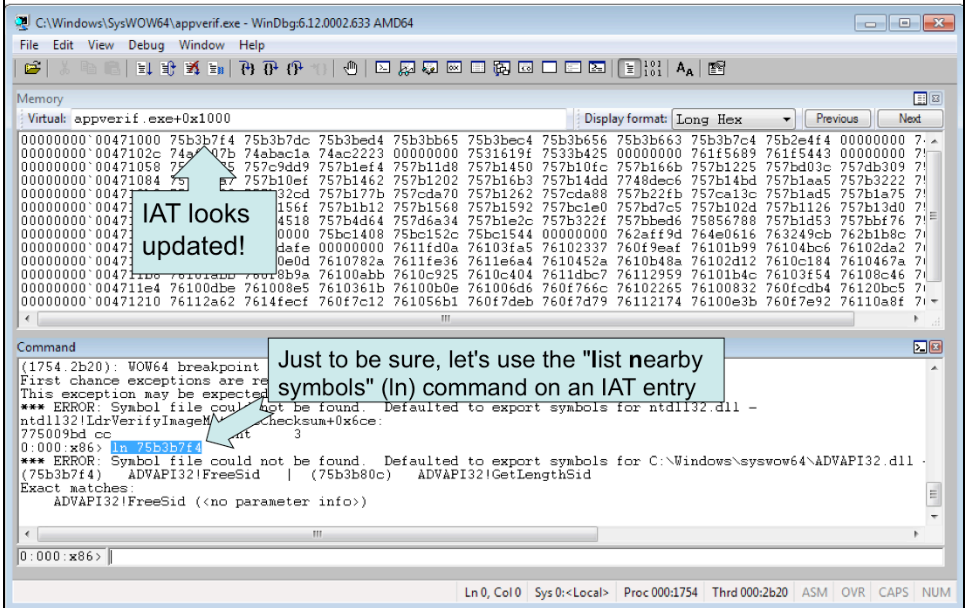
93


Set to appverif.exe + RVA of IAT
Set to "Long Hex"
Hmm…this still seems to match what's on disk? Turns out WinDbg hit a breakpoint before the loader had a chance to resolve the IAT
Base where this got loaded this time

94


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
Step into
Step over
Step out
File Edit View Debug Window
Memory
Previous
Next
Source mode off
Go/Continue
Stop debugging
Set breakpoint wherever the cursor is currently
Restart debugging

Hit Go and you will see this



96

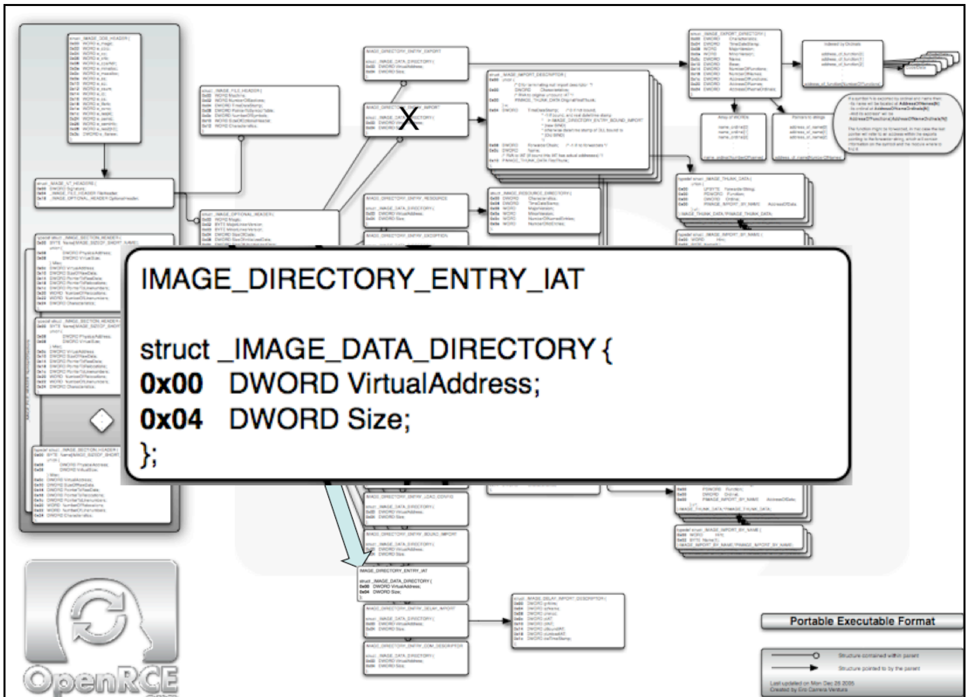


Image by Ero Carrera

Get your geek on

- Play through round 4 on your own, and then wait for the seed for the class deathmatch
- If you see something like the following: "user32.dll!Foofus" that means the function Foofus() in user32.dll
- You can skip to level 4 by starting the game with "python BinHunt.py 4"

98

See notes for citation

New 2012 – NOTE: I spent way more time on that token than I should have, so you must love and cherish it

From

http://www.classicplastic.net/dvgi/pics-tokenstilt02.jpg

http://www.classicplastic.net/dvgi/pics-tokensgeneric02.jpg

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.

101

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.)

102

TODO:

- First thing tomorrow, show the fastjump r0x0r

103

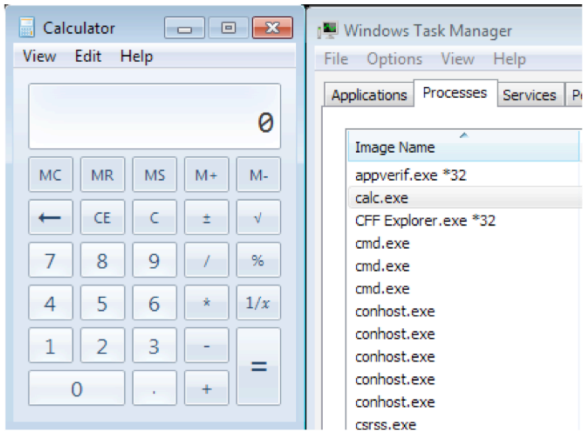
FIXME: Lab: IAT hooking

- 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:\tmp\AppInitHookIAT.dll for simplicity
  – Use regedit.exe to set **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft \Windows NT \CurrentVersion\Windows\LoadAppInit_DLLs = 1**
  – Use regedit.exe to add C:\tmp\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 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.)

104

Can also read more here: http://www.codeproject.com/KB/system/api_spying_hack.aspx

Before IAT hooking



105

After IAT hooking

http://knowyourmeme.com/memes/oh-crap-omg-rage-face