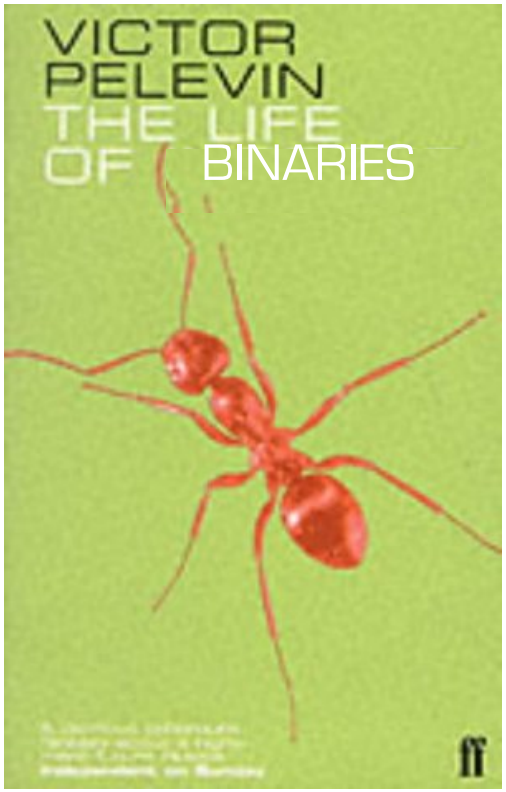


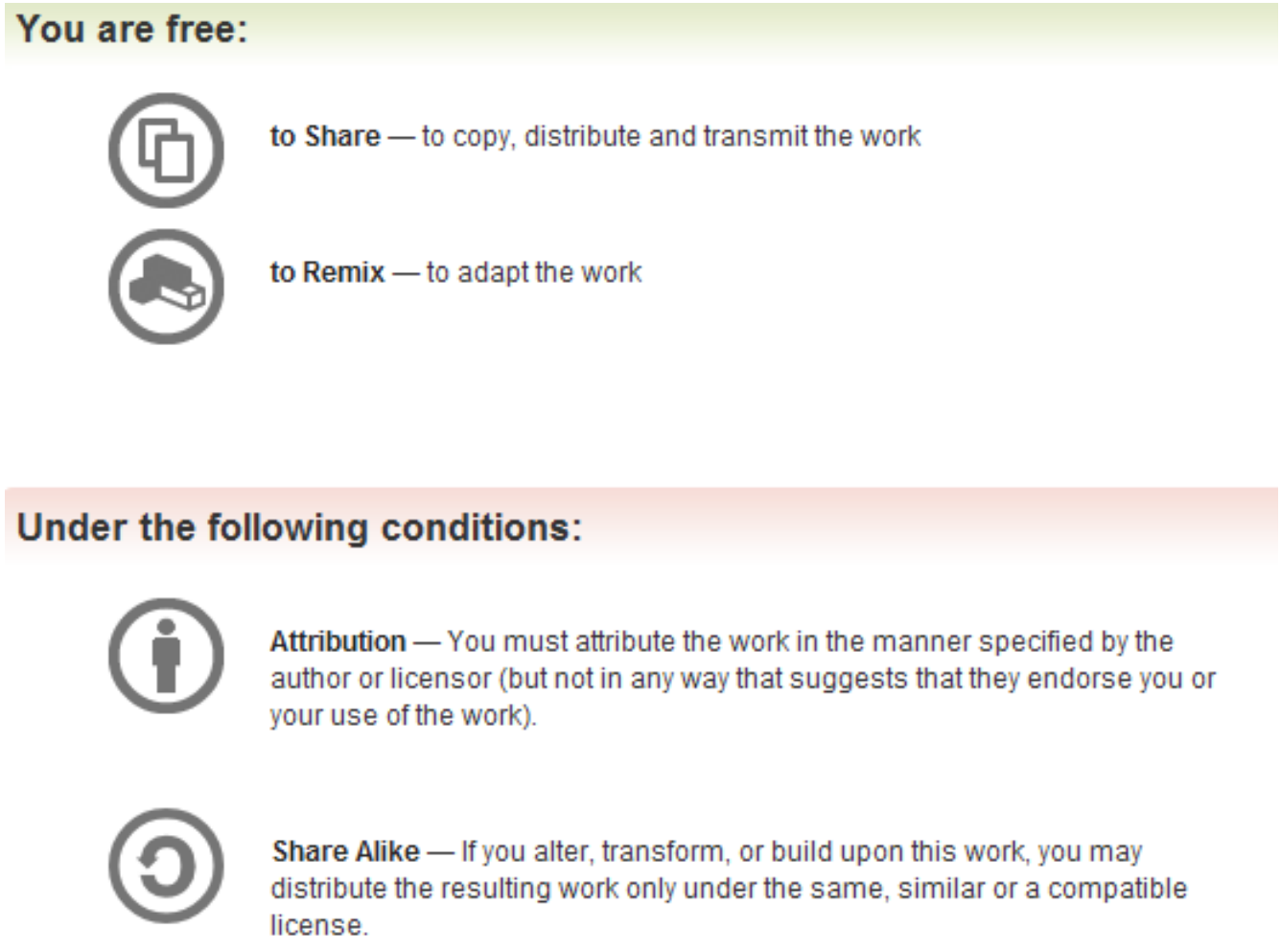
Xeno Kovah - 2010

xkovah at gmail

Approved for Public Release: 10-4654. Distribution Unlimited

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

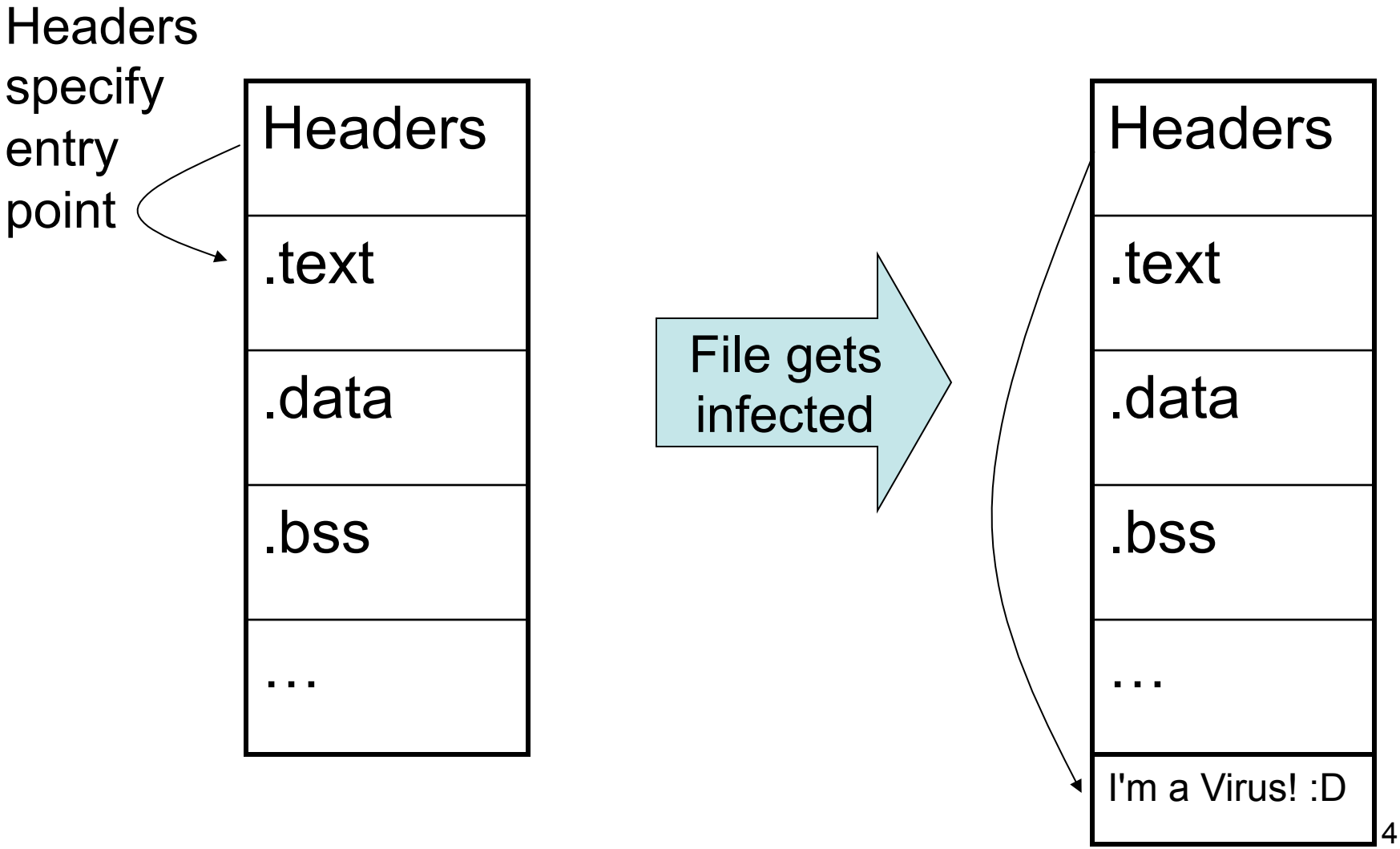
- http://creativecommons.org/licenses/by-sa/3.0/



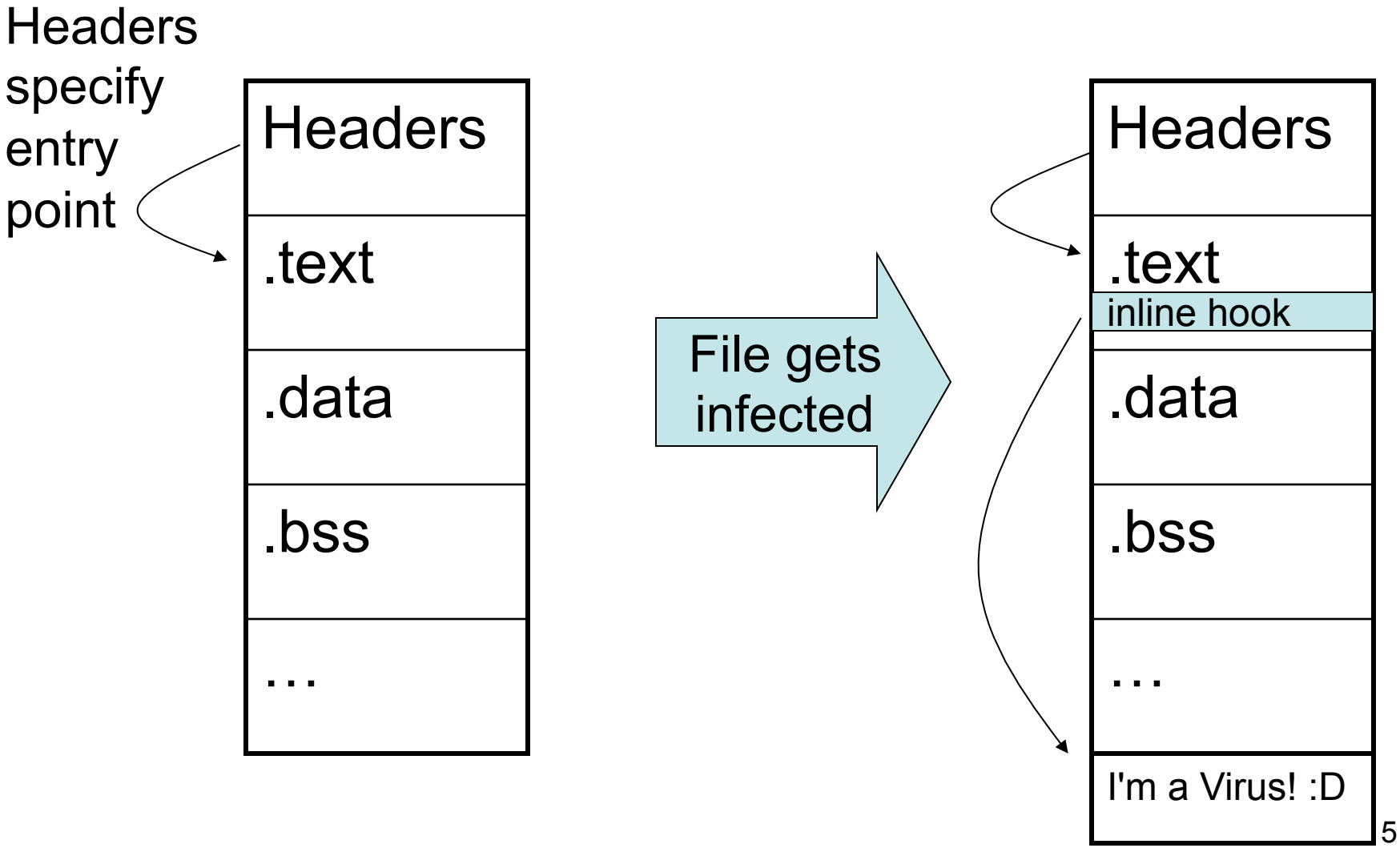
Viruses

- It's not Virii (http://en.wikipedia.org/wiki/Plural_form_of_words_ending_in_-us)
- Viruses are malware which self-replicate, but which need humans to initiate their execution. This is in contrast to worms which can self-replicate in the absence of humans.
- We're primarily interested in viruses which infect PE files, but they can infect other file types as well.

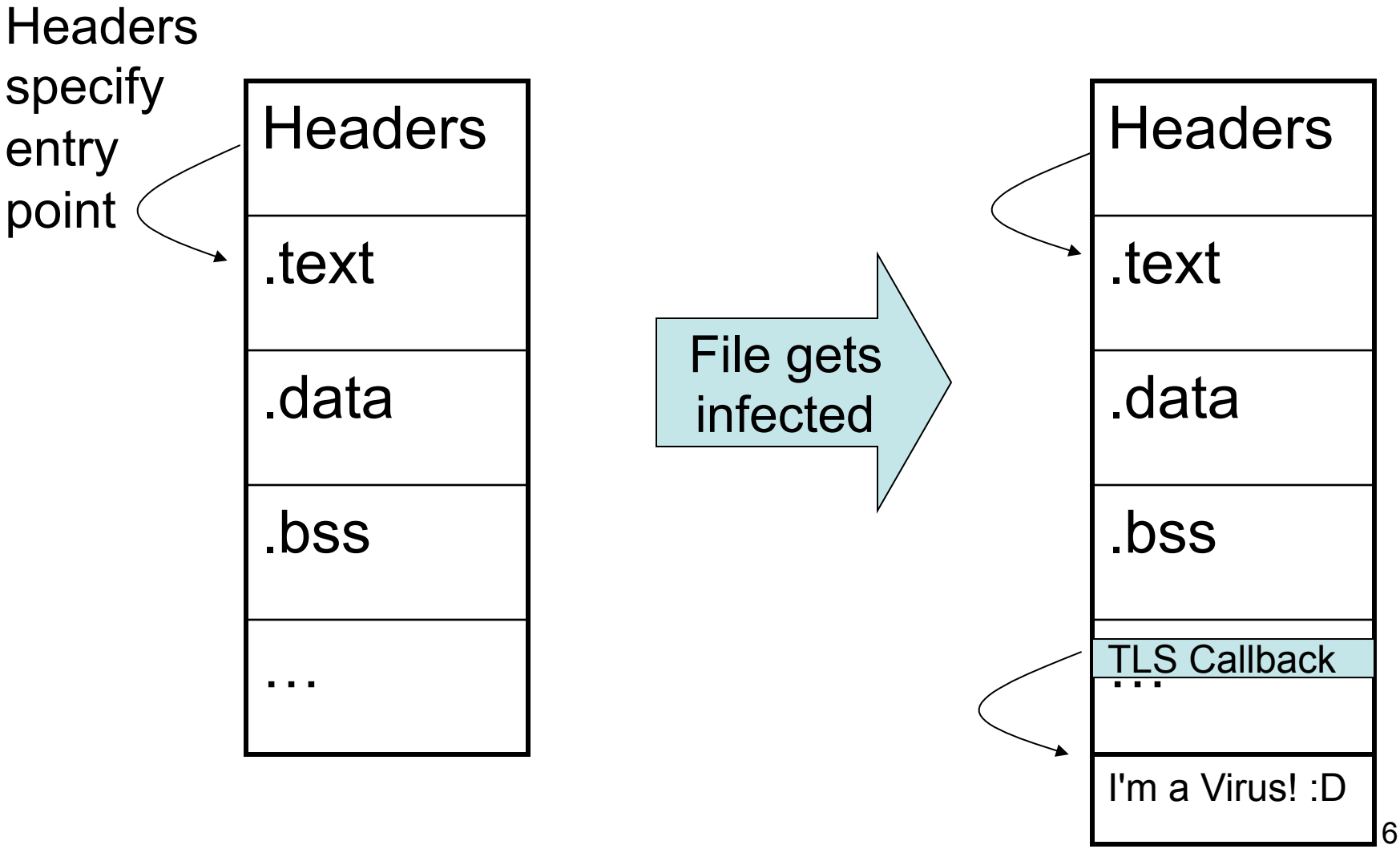
Conceptual: Executable file infection



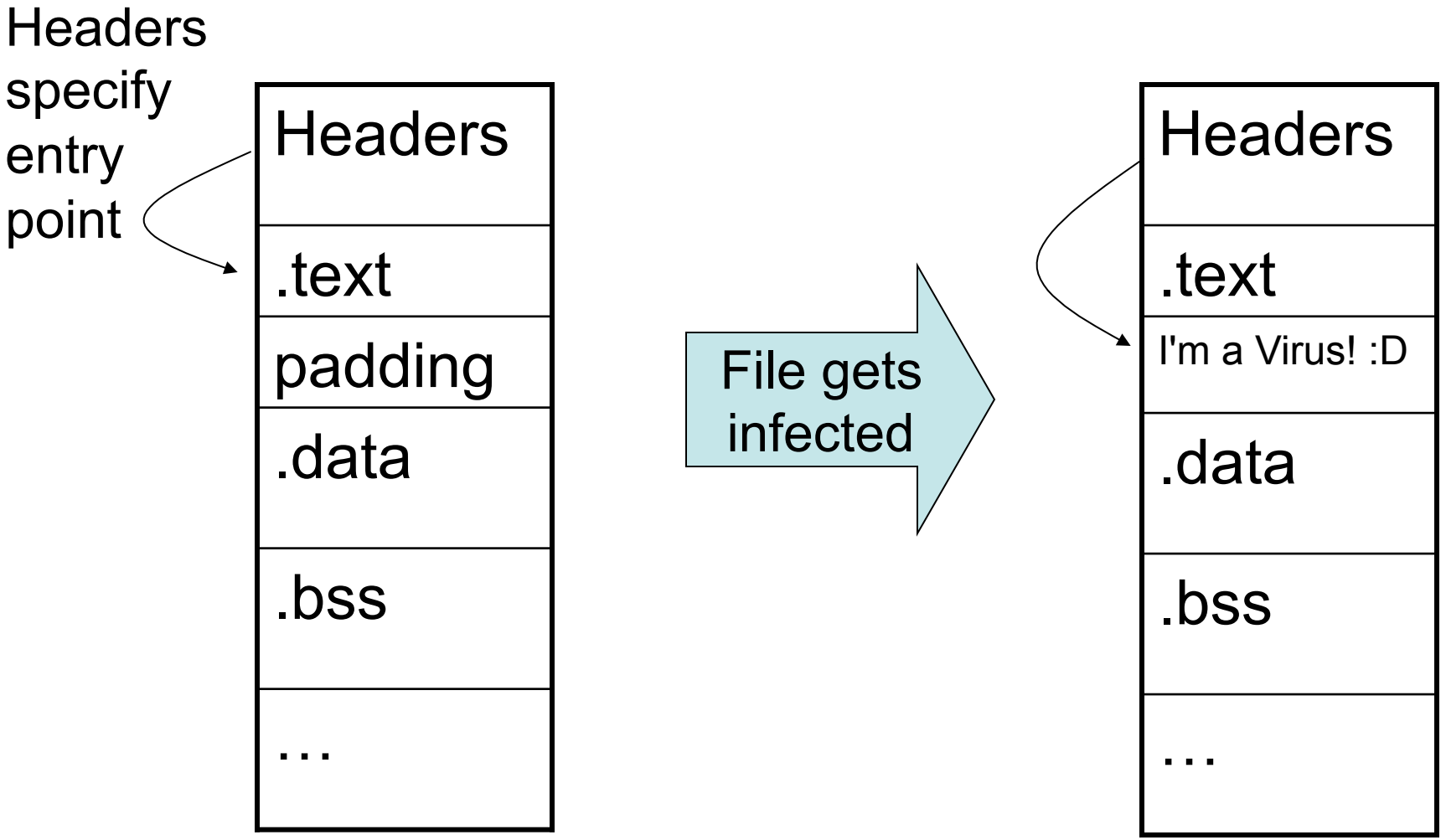
Or Maybe…


Headers specify entry point
Headers
.text
.data
.bss
…
File gets infected
Headers
.text
inline hook
.data
.bss
…
I'm a Virus! :D

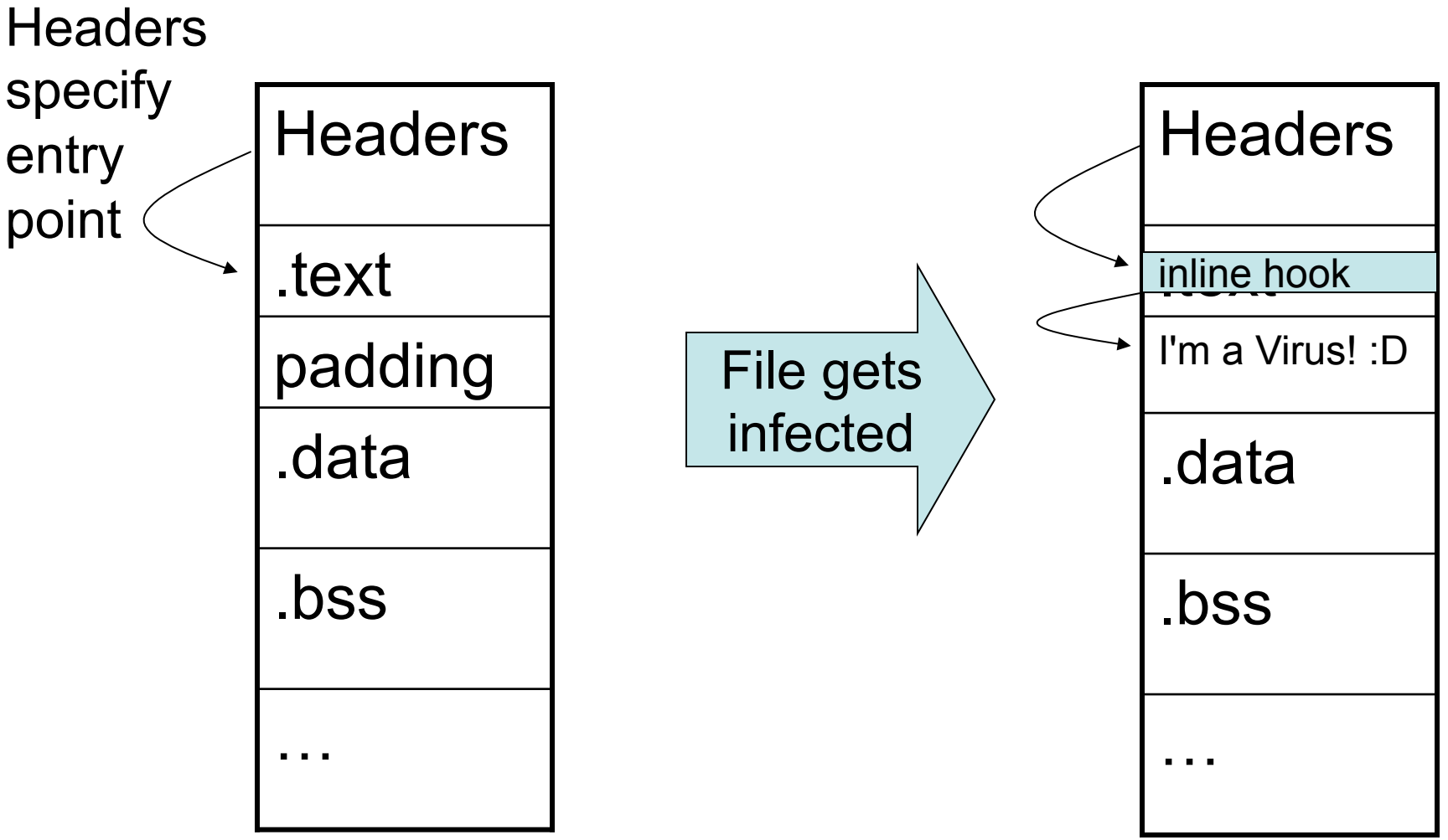
Or Maybe…


Headers specify entry point
Headers
.text
.data
.bss
…
File gets infected
Headers
.text
.data
.bss
TLS Callback
…
I'm a Virus! :D

Or Maybe…


Headers specify entry point
Headers
.text
padding
.data
.bss
…
File gets infected
Headers
.text
I'm a Virus! :D
.data
.bss
…

Or Maybe…


Headers specify entry point
Headers
.text
padding
.data
.bss
…
File gets infected
Headers
inline hook
I'm a Virus! :D
.data
.bss
…

Padding

- Show how between 0x4a4 and 0xf14 is all just padding in the hello ELF binary, and therefore it can be infected.
- But we're going to go for a virus which tacks itself to the end of the file, and alters the headers to make the extra data get loaded into memory.

Interlude

- For the PE infector I was sitting around thinking "How am I going to get the address of CreateFile/WriteFile in a way where I don't just do opportunistic infection of files which already have those or GetProcAddress in their IAT" and then Corey Kallenberg reminded me that Skape talked about EAT searching in his excellent shellcode paper: http://www.hick.org/code/skape/papers/win32-shellcode.pdf, so thanks go out to Corey and Skape.

Lab: BabysFirstPhageForPE.c

- Is like the conceptual picture 1
- Can only infect binaries in C:\VirusTarget
- Has built in kill switch logic so that the parents can infect binaries to create children, and children can infect binaries to make grandchildren, but the grandchildren are sterile and can't have children of their own.
- Also, it doesn't fix the OptionalHeader.Checksum. So that must be fixed before an infected file will run to infect another file.
- Doesn’t do anything malicious other than replicate, and whether that’s malicious or not is in the eye of the beholder

Steps

- Copy HelloWorld and any other executables to infect to C:\VirusTarget
- Run BabysFirstPhageForPE.exe in VS debug mode talking through the stages
- Open infected file up with CFF Explorer, go to Rebuilder section, and select only the "Update Checksum" option, then close it and save the changes.
- Open infected binary in WinDbg, set a breakpoint on the entry point in the PE header (which is the virus entry point), and step through the code
- NOTE: If you're doing this on your own, make sure you don't have any breakpoints set in the virus code at the time that it copies itself into the buffer, otherwise the breakpoints will get copied too! (We learn about how breakpoints modify the code in Intermediate x86)

Throwback…TO THE MAX!

- A virus in the days of botnets!: Virut
- (see what they did there? Virus+=1? They're so clever. Also called Virux…because X is awesome!)
- http://www.f-secure.com/v-descs/virus_w32_virut.shtml
- "Variants in the Virut family are polymorphic, memory-resident, appending file infectors that have Entry Point Obscuring (EPO) capabilities."
- http://www.symantec.com/connect/blogs/w32virutcf-collateral-damage
- "All of this sounds quite grim, but this threat can be removed from infected networks by following best practices."
- Of course! Best practices! In retrospect it's all so obvious!

Fortune favors the bold

- Xpaj: Another misc virus
- “It is not very common for a file infector to do more than simply introduce trivial modifications to the files it infects. Virus authors usually avoid complex modifications to the files because of the possibility of corruption. W32.Xpaj.B is one of exceptions.”
- http://www.symantec.com/connect/blogs/w32xpajb-upper-crust-file-infector
- Core principals the same, just another way of going about it, but the point is, now you can read and understand

Further Reading

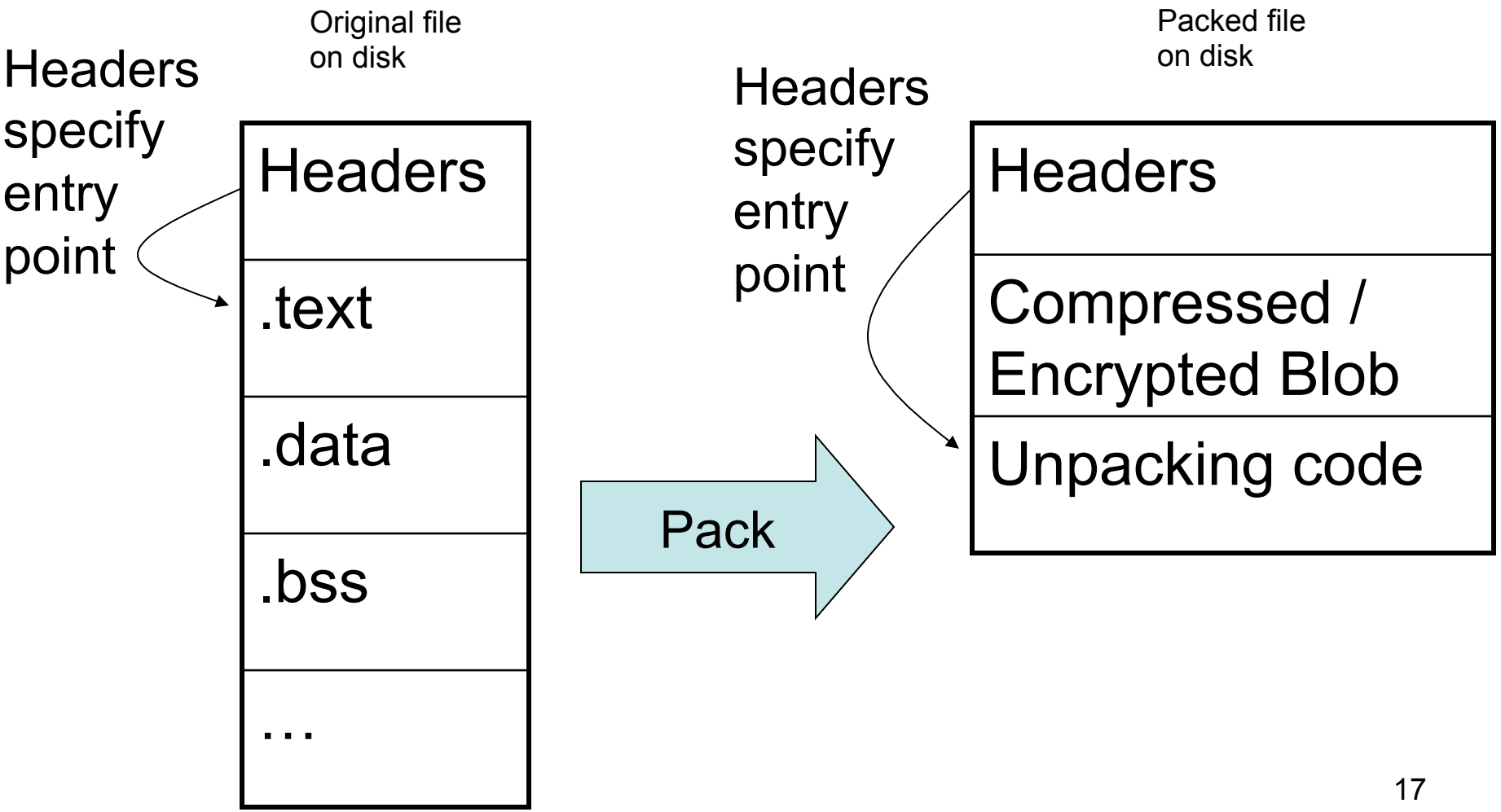
(with the exception of the ~nemo link these are just misc googling that I haven't read, so no guarantee of quality)

- PE infection
  - http://www.defcon.org/images/defcon-16/dc16-presentations/lclee/defcon-16-lclee_vx-wp.pdf (compares PE vs ELF infection)
  - http://vx.netlux.org/29a/29a-7/Articles/29A-7.023
- ELF infection
  - http://felinemenace.org/~mercy/slides/RUXCON2004-ELFfairytale.ppt
  - http://www.linuxsecurity.com/resource_files/documentation/virus-writing-HOWTO/_html/index.html
  - http://vxheavens.com/lib/static/vdat/tuunix02.htm
- Mach-O infection
  - http://felinemenace.org/~nemo/slides/mach-o_infection.ppt
  - http://vxheavens.com/lib/vrg01.html
- Old sk00l
  - http://www.textfiles.com/virus/
  - http://vx.netlux.org/29a/main.html
  - http://vx.netlux.org/lib/

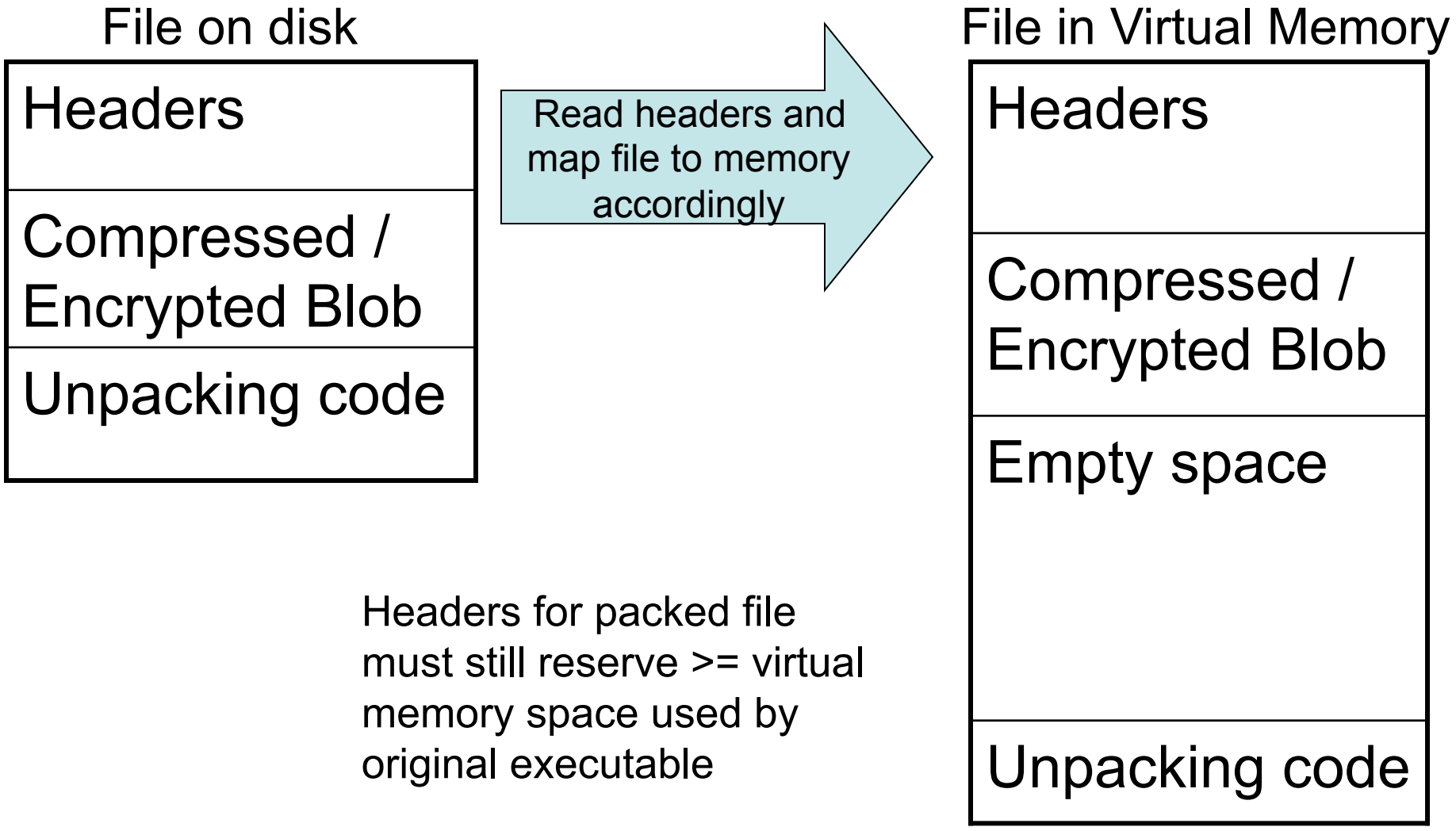
Packers

- Originally used to compress executables back when disk space was at a premium
- The executable would then decompress itself in memory and run as normal
- Nowadays they are mostly used for obfuscating binaries. Specifically since all the data for the original binary is compressed and/or encrypted, it prevents analysts from being able to infer things about the binary based on strings or function imports.

Conceptual Packing: File On Disk

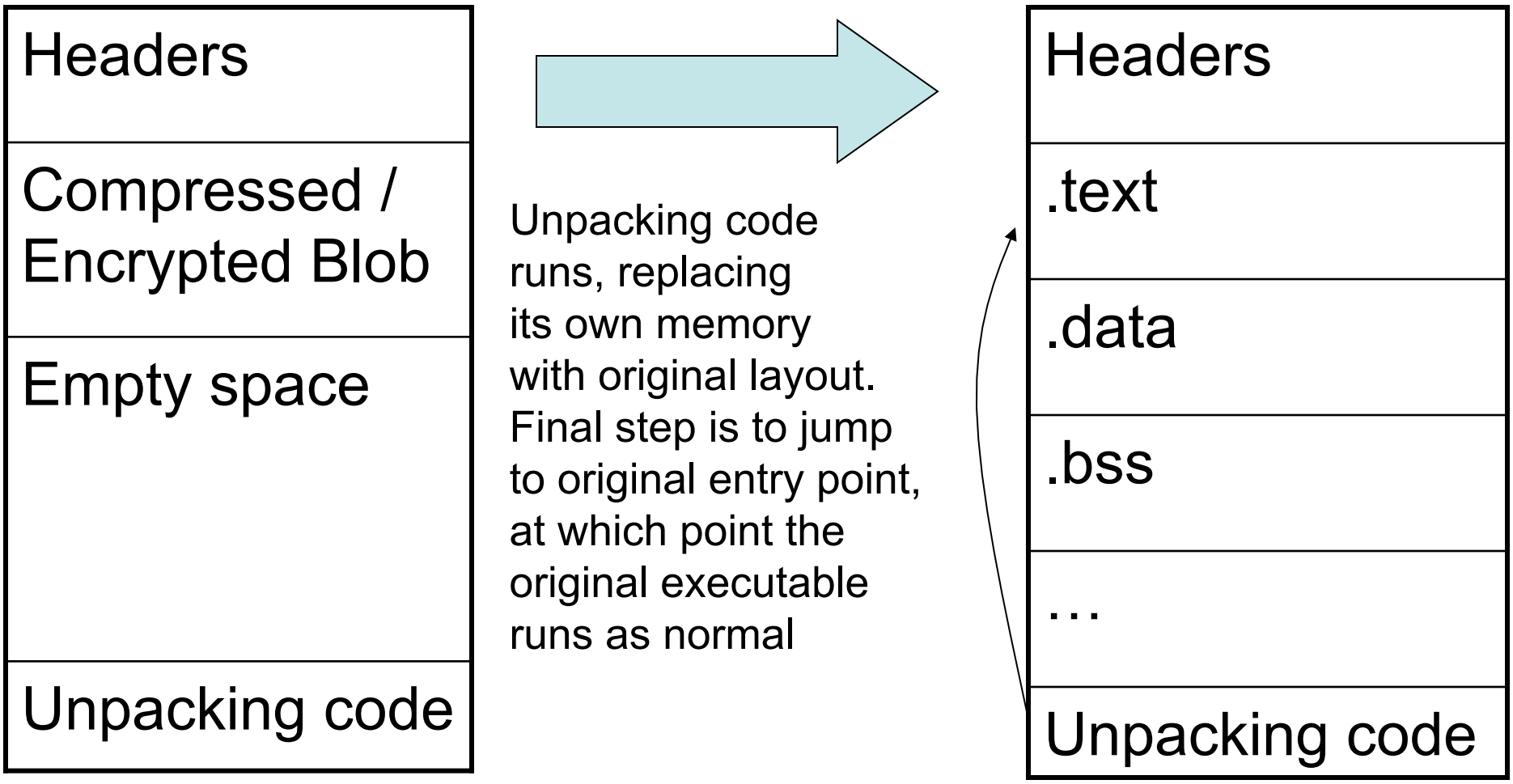


Conceptual Unpacking: Load Time



Headers for packed file must still reserve >= virtual memory space used by original executable

Conceptual Unpacking: Run Time



The Ultimate Packer for eXecutables (UPX)

- http://upx.sourceforge.net/
- Easy to understand, very cross-platform compatible, legitimate packer which also has an automatic unpacking ability as well.
- Run it as "upx File -o PackedFile" to pack, and "upx -d PackedFile –o File" to decompress.
- Demo the header changes made by UPX to both a PE and ELF file

UPX applied to PE hello.c

Before UPX

RVA	Data	Description
00000108	010B	Magic
0000010A	09	Major Linker Version
0000010B	00	Minor Linker Version
0000010C	00000A00	Size of Code
00000110	00000E00	Size of Initialized Data
00000114	00000000	Size of Uninitialized Data
00000118	000012C2	Address of Entry Point
0000011C	00001000	Base of Code
00000120	00002000	Base of Data
00000124	00400000	Image Base
00000128	00001000	Section Alignment
0000012C	00000200	File Alignment
00000130	0005	Major O/S Version
00000132	0000	Minor O/S Version
00000134	0000	Major Image Version
00000136	0000	Minor Image Version
00000138	0005	Major Subsystem Version
0000013A	0000	Minor Subsystem Version
0000013C	00000000	Win32 Version Value
00000140	00006000	Size of Image
00000144	00000400	Size of Headers
00000148	0000EF54	Checksum
0000014C	0003	Subsystem
0000014E	8140	DLL Characteristics
		0040
		0100
		8000

After UPX

RVA	Data	Description
00000108	010B	Magic
0000010A	09	Major Linker Version
0000010B	00	Minor Linker Version
0000010C	00001000	Size of Code
00000110	00001000	Size of Initialized Data
00000114	00006000	Size of Uninitialized Data
00000118	00007920	Address of Entry Point
0000011C	00007000	Base of Code
00000120	00008000	Base of Data
00000124	00400000	Image Base
00000128	00001000	Section Alignment
0000012C	00000200	File Alignment
00000130	0005	Major O/S Version
00000132	0000	Minor O/S Version
00000134	0000	Major Image Version
00000136	0000	Minor Image Version
00000138	0005	Major Subsystem Version
0000013A	0000	Minor Subsystem Version
0000013C	00000000	Win32 Version Value
00000140	00009000	Size of Image
00000144	00001000	Size of Headers
00000148	00000000	Checksum
0000014C	0003	Subsystem
0000014E	8140	DLL Characteristics
		0040
		0100
		8000

UPX applied to PE hello.c 2

- HelloWorld.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
    - 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
    - IMAGE_RESOURCE_DIRECTORY Type
    - IMAGE_RESOURCE_DIRECTORY Namell
    - IMAGE_RESOURCE_DIRECTORY Langua
    - IMAGE_RESOURCE_DATA_ENTRY
    - MANIFEST 0001 0409
  - SECTION .reloc
    - IMAGE_BASE_RELOCATION

RVA	Data	Description	Value
000001E8	2E 74 65 78	Name	.text
000001EC	74 00 00 00		
000001F0	0000080E	Virtual Size	
000001F4	00001000	RVA	
000001F8	00000A00	Size of Raw Data	
000001FC	00000400	Pointer to Raw Data	
00000200	00000000	Pointer to Relocations	
00000204	00000000	Pointer to Line Numbers	
00000208	0000	Number of Relocations	
0000020A	0000	Number of Line Numbers	
0000020C	60000020	Characteristics	
		00000020	IMAGE_SCN_CNT_CODE
		20000000	IMAGE_SCN_MEM_EXECUTE
		40000000	IMAGE_SCN_MEM_READ

RVA	Data	Description	Value
00000288	2E 72 65 6C	Name	.reloc
0000028C	6F 63 00 00		
00000290	0000018A	Virtual Size	
00000294	00005000	RVA	
00000298	00000200	Size of Raw Data	
0000029C	00001A00	Pointer to Raw Data	
000002A0	00000000	Pointer to Relocations	
000002A4	00000000	Pointer to Line Numbers	
000002A8	0000	Number of Relocations	
000002AA	0000	Number of Line Numbers	
000002AC	42000040	Characteristics	
		00000040	IMAGE_SCN_CNT_INITIALIZED_DATA
		02000000	IMAGE_SCN_MEM_DISCARDABLE
		40000000	IMAGE_SCN_MEM_READ

Total virtual size = 0x518A

Total file size = 0x1C00

UPX applied to PE hello.c 3

- UPXedHelloWorld.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
    - Signature
    - IMAGE_FILE_HEADER
    - IMAGE_OPTIONAL_HEADER
  - IMAGE_SECTION_HEADER UPX0
  - IMAGE_SECTION_HEADER UPX1
  - IMAGE_SECTION_HEADER .rsrc
  - SECTION UPX0
  - SECTION UPX1
    - IMAGE_LOAD_CONFIG_DIRECTORY
  - SECTION .rsrc
    - IMAGE_RESOURCE_DIRECTORY Type
    - IMAGE_RESOURCE_DIRECTORY NameID
    - IMAGE_RESOURCE_DIRECTORY Language
    - IMAGE_RESOURCE_DATA_ENTRY
    - MANIFEST 0001 0409
    - IMPORT Directory Table
    - IMPORT Address Table
    - IMPORT DLL Names
    - IMPORT Hints/Names
    - IMAGE_BASE_RELOCATION

RVA	Data	Description	Value
000001E8	55 50 58 30	Name	UPX0
000001EC	00 00 00 00		
000001F0	00006000	Virtual Size	
000001F4	00001000	RVA	
000001F8	00000000	Size of Raw Data	
000001FC	00000400	Pointer to Raw Data	
00000200	00000000	Pointer to Relocations	
00000204	00000000	Pointer to Line Numbers	
00000208	0000	Number of Relocations	
0000020A	0000	Number of Line Numbers	
0000020C	E0000080	Characteristics	
		00000080	IMAGE_SCN_CNT_UNINITIALIZED_DATA
		20000000	IMAGE_SCN_MEM_EXECUTE
		40000000	IMAGE_SCN_MEM_READ
		80000000	IMAGE_SCN_MEM_WRITE
00000210	55 50 58 31	Name	UPX1
00000214	00 00 00 00		
00000218	00001000	Virtual Size	
0000021C	00007000	RVA	
00000220	00000C00	Size of Raw Data	
00000224	00000400	Pointer to Raw Data	
00000228	00000000	Pointer to Relocations	
0000022C	00000000	Pointer to Line Numbers	
00000230	0000	Number of Relocations	
00000232	0000	Number of Line Numbers	
00000234	E0000040	Characteristics	
		00000040	IMAGE_SCN_CNT_INITIALIZED_DATA
		20000000	IMAGE_SCN_MEM_EXECUTE
		40000000	IMAGE_SCN_MEM_READ
		80000000	IMAGE_SCN_MEM_WRITE
00000238	2E 72 73 72	Name	.rsrc
0000023C	63 00 00 00		
00000240	00001000	Virtual Size	
00000244	00008000	RVA	
00000248	00000400	Size of Raw Data	
0000024C	00001000	Pointer to Raw Data	
00000250	00000000	Pointer to Relocations	
00000254	00000000	Pointer to Line Numbers	
00000258	0000	Number of Relocations	
0000025A	0000	Number of Line Numbers	
0000025C	C0000040	Characteristics	
		00000040	IMAGE_SCN_CNT_INITIALIZED_DATA
		40000000	IMAGE_SCN_MEM_READ
		80000000	IMAGE_SCN_MEM_WRITE

Covers the entire original memory space

No data from file

Total virtual size = 0x9000

Total file size = 0x1400

UPX applied to ELF hello.c

readelf -l hello-static

```
Elf file type is EXEC (Executable file)
Entry point 0x80481e0
There are 6 program headers, starting at offset 52
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x08048000 0x08048000 0x851df 0x851df R E 0x1000   (Total FileSize = 0x859B3)
  LOAD           0x085f8c 0x080cef8c 0x080cef8c 0x007d4 0x02388 RW  0x1000   (Total MemSize = 0x87567)
  NOTE           0x0000f4 0x080480f4 0x080480f4 0x00044 0x00044 R   0x4
  TLS            0x085f8c 0x080cef8c 0x080cef8c 0x00010 0x00028 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x085f8c 0x080cef8c 0x080cef8c 0x00074 0x00074 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00     .note.ABI-tag .note.gnu.build-id .rel.plt .init .plt .text __libc_freeres_fn .fini .rodata
    __libc_atexit __libc_subfreeres .eh_frame .gcc_except_table
   01     .tdata .ctors .dtors .jcr .data.rel.ro .got .got.plt .data .bss __libc_freeres_ptrs
   02     .note.ABI-tag .note.gnu.build-id
   03     .tdata .tbss
   04
   05     .tdata .ctors .dtors .jcr .data.rel.ro .got
```

readelf -l hello-static-packed

```
Elf file type is EXEC (Executable file)
Entry point 0xc40708
There are 2 program headers, starting at offset 52
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x00c01000 0x00c01000 0x3ff08 0x3ff08 R E 0x1000   (Total FileSize = 0x3FF08)
  LOAD           0x000314 0x080d1314 0x080d1314 0x00000 0x00000 RW  0x1000   (Total MemSize = 0x3FF08)
```

Original executable 2nd LOAD segment VirtAddr + MemSiz

Pseudo Execution Flow Description

my wife pointed out if I don't obfuscate the addresses you could skip straight to step 4 ;)

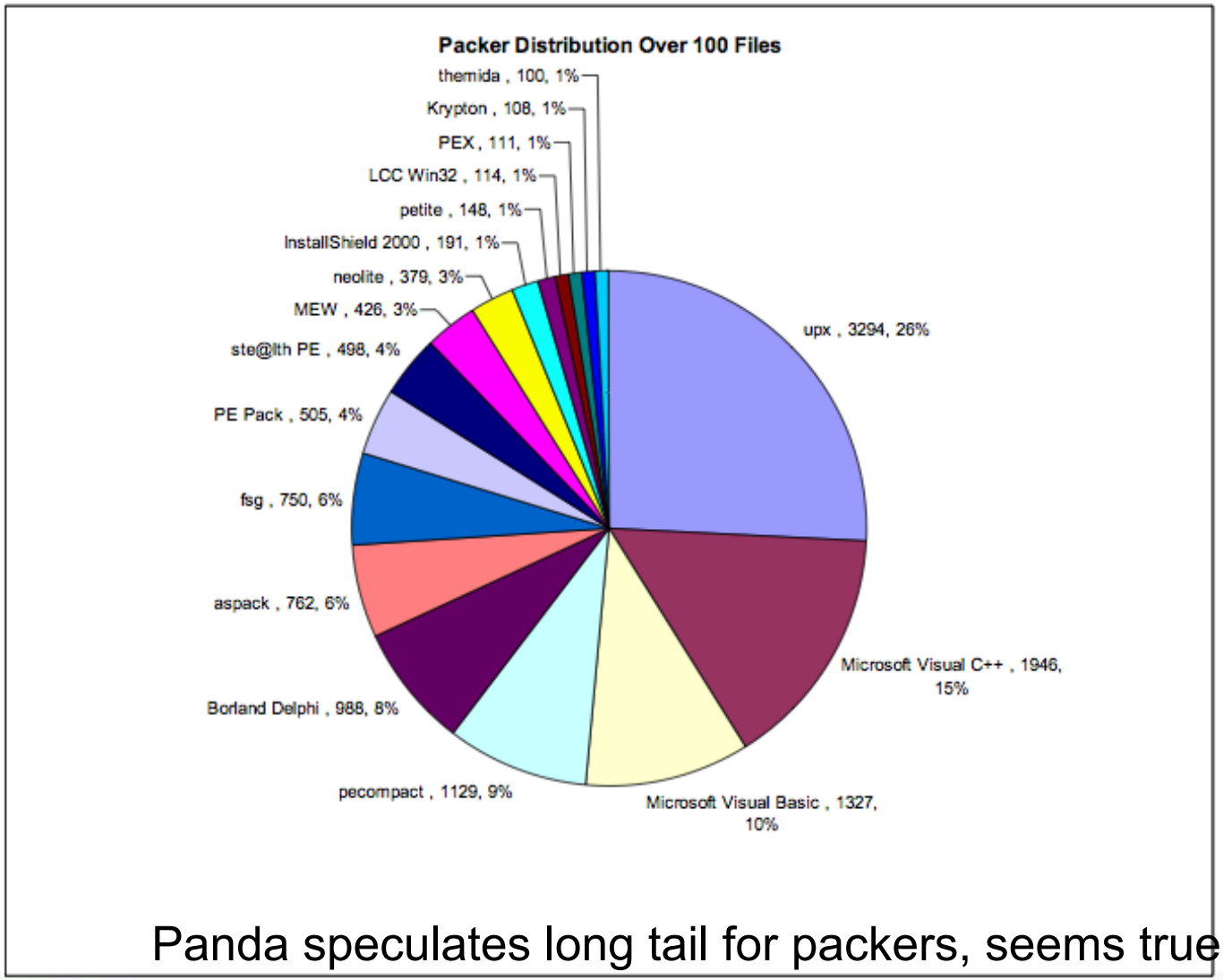
- 0. Starts out running at its entry point
- 1. mmaps a page <SOMEWHERE>
- 2. Copies some of its code to <SOMEWHERE>
- 3. You will be stuck in loops the range between <SOMEWHERE> and <SOMEWHERE> for a while. Find the escape hatch!
- 4. Transfers control flow to <SOMEWHERE>
- 5. mmap allocates memory space sufficient to cover original memory space
- 6. Decompresses data into original location
- 7. munmaps the original compressed memory area (but leaves the 1 page at <SOMEWHERE>) immediately before going to the original entry point (OEP)

Other packers

- Other packers introduce extensive anti-debug tricks into the unpacking code.
- Themida/VMProtect will take x86 code and convert it into a series of equivalent instructions in a custom bytecode language which is subsequently interpreted into machine code (which may not necessarily be the exact same code that existed in the application pre-modification.)
  - Therefore the "VM" in VMProtect is a "software virtual machine" in the "java virtual machine" sense. Java interprets bytecode in order to eventually execute equivalent machine code, these VMs interpret bytecode to execute equivalent machine code.

http://www.defcon.org/images/defcon-15/dc15-presentations/dc-15-valsmith_and_delchi.pdf

Packers



More than 100 files detected with packer listed

Panda speculates long tail for packers, seems true

Further Reading

- PE
  - Lots and lots of good stuff - http://pferrie.tripod.com/
  - http://www.codebreakers-journal.com/downloads/cbj/2006/CBM_1_2_2006_BigBoote_Own_Packer.pdf
  - Automatic unpacking - http://www.joestewart.org/ollybone/tutorial.html
  - Automatic unpacking - http://bitblaze.cs.berkeley.edu/renovo.html
  - http://securitylabs.websense.com/content/Assets/HistoryofPackingTechnology.pdf
  - http://www.lmgtfy.com/?q=PE+packer+filetype%3Apdf
- ELF
  - Shiva - advanced packer w/ anti-debug tricks: http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-mehta/bh-us-03-mehta.pdf
  - Reversing shiva: http://www.blackhat.com/presentations/bh-federal-03/bh-federal-03-eagle/bh-fed-03-eagle.pdf
  - Burneye packer - http://packetstormsecurity.org/groups/teso/burneye-1.0.1-src.tar.bz2
  - Userland Exec - Just another name for what unpacking does as far as I'm concerned, but still worth a read - http://www.securityfocus.com/archive/1/348638
  - http://www.lmgtfy.com/?q=ELF+packer+filetype%3Apdf

Other self-decompressors

- The concept of self-decompression at runtime is also used in other areas. For instance Cisco's IOS is stored on the router in a compressed form which decompresses in memory. (http://www.coresecurity.com/files/attachments/Killing_the_myth_of_Cisco_IOS_rootkits.pdf.rar)
- BIOS can play the same game. (http://www.coresecurity.com/files/attachments/Persistent_BIOS_Infection_CanSecWest09.pdf)
- Both of these are obviously doing this because storage space is expected to be smaller than memory.

Reflective DLL Injection

- http://www.harmonysecurity.com/ReflectiveDllInjection.html
- In contrast with normal DLL injection, where you might rely on the OS to load the DLL for you, a reflective DLL is self-sufficient. The code within it will handle the necessary initialization (that the OS loader would normally do) in order to ensure it can execute normally. The benefit is that this DLL will not be registered anywhere by the OS as being a DLL, it will just be some blob of code in memory somewhere.
- Note, that this property can also potentially be used to find it. That is, if you are doing memory analysis and you see something that indicates control flow eventually transfers to "some blob of code in memory somewhere", that is sort of suspicious, and therefore bears investigation. (Of course then the question becomes, how did you find this control flow divergence ;))
- That said, there are "legitimate" reasons there may be code blobs running around in memory; see the Adobe Flash ActionScript Just-In-Time (JIT) code generation engine…and the subsequent utilization for exploits ("JIT Spray" - http://www.semantiscope.com/research/BHDC2010/BHDC-2010-Paper.pdf)

Hot Patching Running Binaries

(topic added under threat of physical violence by my wife!
"No! You have to add it! Patching is part of a binary's life! Don't make me give you to the back of my hand!")

- Good talk here covering all the various ways you can hotpatch, including microsoft's way: http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Sotirov.pdf
- /hotpatch option can be added manually to the additional compiler options. Then it will generate a "mov edi, edi" instruction before the normal "push ebp; mov ebp, esp" function prolog which will make your functions easier to hotpatch in the future if necessary.
- The "mov edi, edi" obviously does nothing, but it's 2 bytes, which when combined with the 3 bytes for the "push ebp; mov ebp, esp" == 5 bytes. If some code needs to hotpatch a function, the first 5 bytes can then be overwritten with a jmp instruction to jump to the new implementation of the function, and that function knows it only needs to execute the "push ebp; mov ebp, esp" instructions before it jumps back to the original code.

5-Byte JMP Overwrite

determina

Before overwriting the function prologue, we need to save the overwritten instructions. The hook routine should execute the saved instructions before returning to the patched function.

Patched function:

```
jmp     hook
mov     esi, [ebp+arg_0]
...
ret
```

Hook routine:

```
// do some work

// saved instructions
push    ebp
mov     ebp, esp


// return
jmp     patched_function+5
```

modified from
http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Sotirov.pdf

Games Nerds Play: TinyPE/TeenyELF/TinyMach-O

- http://www.phreedom.org/solar/code/tinype/ - 97 bytes
- http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html - 45 bytes
- http://www.osxbook.com/blog/2009/03/15/crafting-a-tiny-mach-o-executable - 165 bytes

Explore your world!
Read the Mach-O Spec! :D

- http://developer.apple.com/mac/library/documentation/DeveloperTools/Conceptual/MachORuntime/Reference/reference.html
- How is it similar to the binary formats we've covered in this class?
- How is it different?
- Where's the 0xbeef?

Bad Moon Rising

- Unified Extensible Firmware Interface (UEFI) is the replacement for BIOS. It's going to bring BIOS into the modern day by making people not have to program as much 16 bit x86.
- It's also standardizing on the PE format for the binaries. That means, now your firmware can have modules which look a lot like Windows executables and are a lot easier to programs.
- Does anyone see why this might start to create more problems for security at the firmware level?

Binject

- http://www.rnicrosoft.net/tools/binject_v0.1.zip
- https://media.blackhat.com/bh-us-10/presentations/Harbour/BlackHat-USA-2010-Harbour-Black-Art-of-Binary-Hijacking-slides.pdf
- Can break apart a binary and put it back together in a convenient trojaned form.
  - The only thing which is going to catch that sort of thing are filesystem integrity checkers.
- DLL Entry Point Redirection, Import Table DLL Additions, TLS Callback and more, made easy

Teardown - What did we learn about?

- Compilation
  - Lexical Analysis - turning characters into lexemes, which can be grouped into tokens)
  - Syntax Analysis - Context Free Grammars in Backus-Naur form (BNF), and Parse Trees (aka Concrete Syntax Trees)
  - Abstract Syntax Trees (ASTs), Abstract Assembly Trees(AATs), tiling on AATs to generate assembly
- Linking
  - Splicing output object files together into a final binary which may be standalone or may depend on the loader to find external libraries for it

Teardown - What did we learn about?

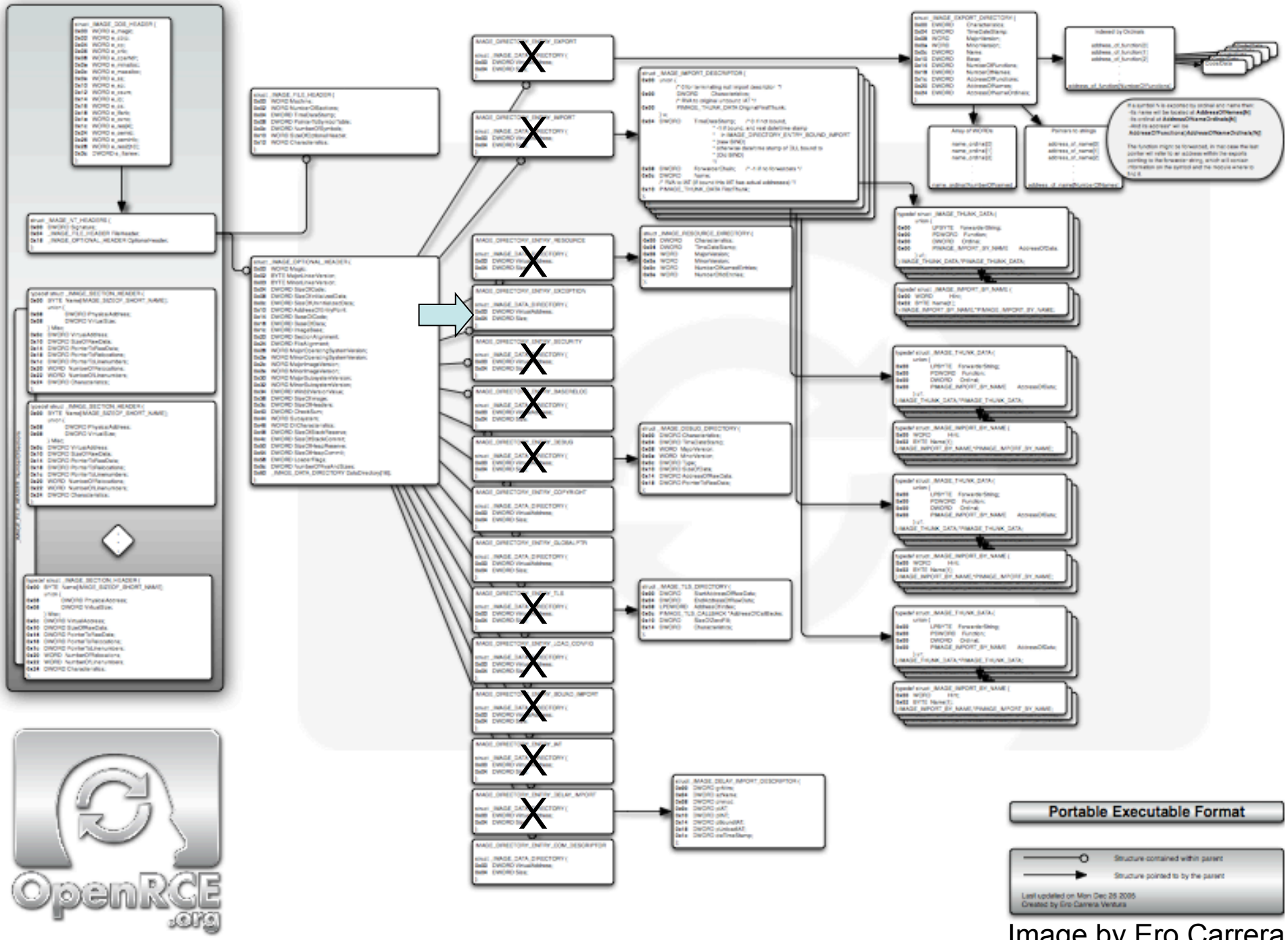
- Portable Executable (PE) binary format used on Windows systems
  - 3 flavors of imports + hooking, exports + hooking & export forwarding, relocations w/ relavance to memory integrity checks, thread local storage (TLS) & TLS callbacks, resources & file embedding, digital signature files
- Executable and Linking Format (ELF) binary format used on *nix systems
  - imports & dynamic linking, exports, relocations

Teardown - What did we learn about?

- F*cking viruses, how do they work?!
- Packing/Unpacking and the effects on binary format and memory contents
- Reflective DLL injection
- Smallest possible binaries
- And the rest!

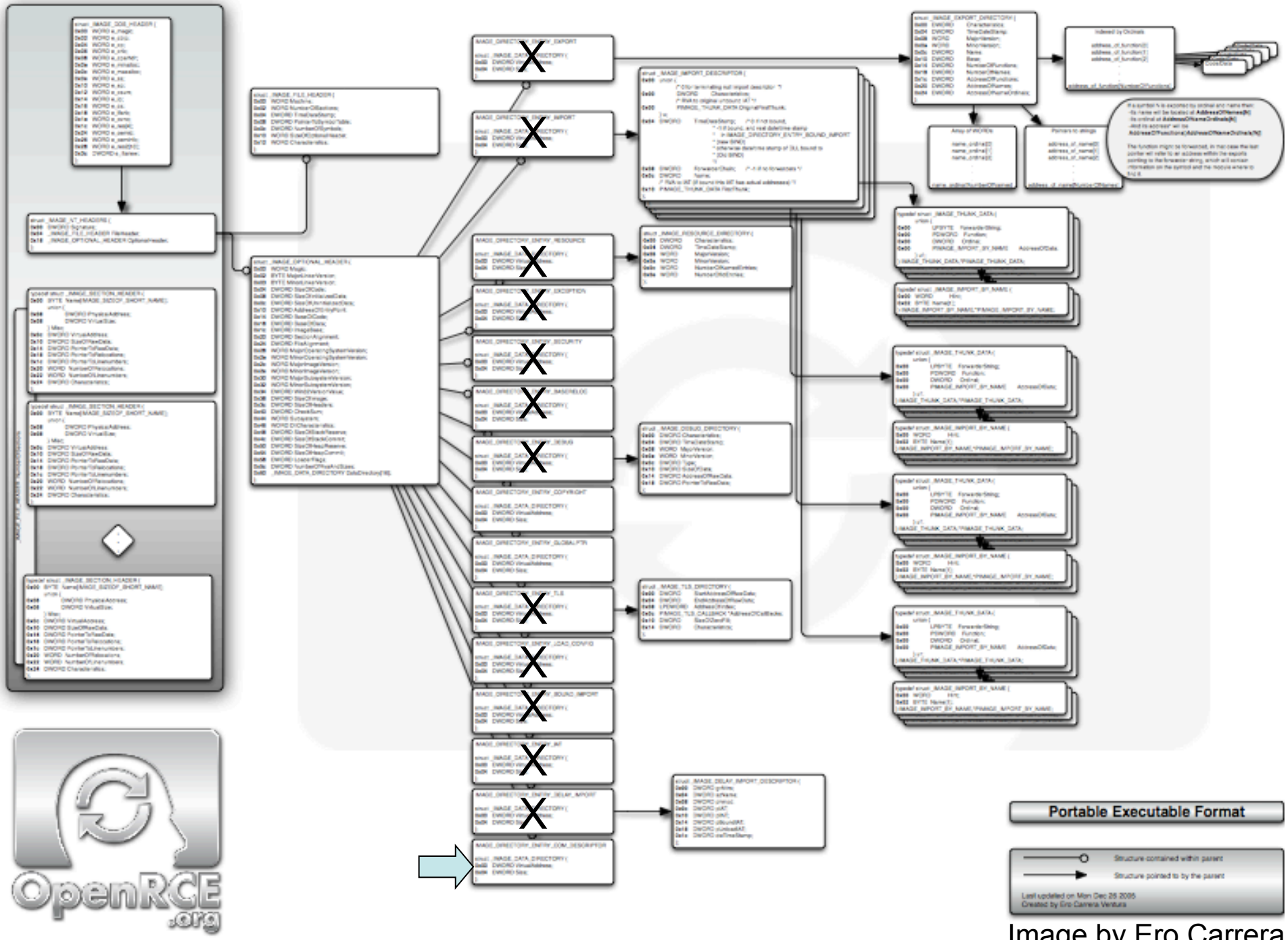
- Fly little birdie fly! It's time for you to explore on your own. Go back through and re-read explanations, read cited materials, etc.

Extra Slides


Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 28 2009
Created by Ero Carrera Ventura
OpenRCE .org
Image by Ero Carrera

IMAGE_DIRECTORY_ENTRY_EXCEPTION

- http://msdn.microsoft.com/en-us/magazine/cc301808.aspx
- “array of IMAGE_RUNTIME_FUNCTION_ENTRY structures, which are CPU-specific. Pointed to by the IMAGE_DIRECTORY_ENTRY_EXCEPTION slot in the DataDirectory. Used for architectures with table-based exception handling, such as the IA-64. The only architecture that doesn't use table-based exception handling is the x86.”


Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 28 2009
Created by Ero Carrera Ventura
OpenRCE .org
Image by Ero Carrera

IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR

- C:\WINDOWS\System32\TsWpfWrp.exe has a non-zero instance of this
- From Matt Pietrek's PE part 2 figures: http://msdn.microsoft.com/en-us/magazine/bb985997.aspx
- "This value has been renamed to IMAGE_DIRECTORY_ENTRY_COMHEADER in more recent updates to the system header files. It points to the top-level information for .NET information in the executable, including metadata. This information is in the form of an IMAGE_COR20_HEADER structure."

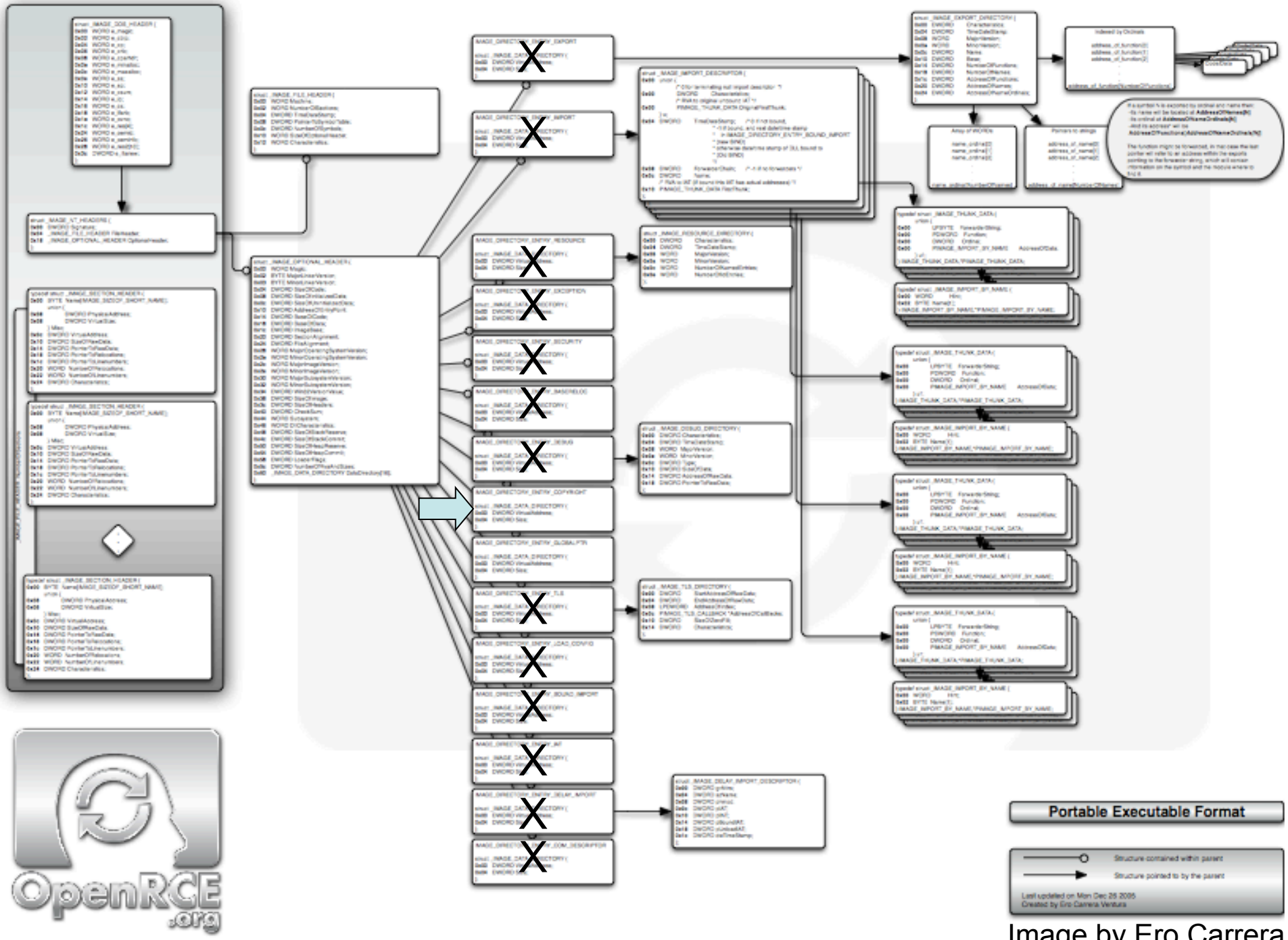
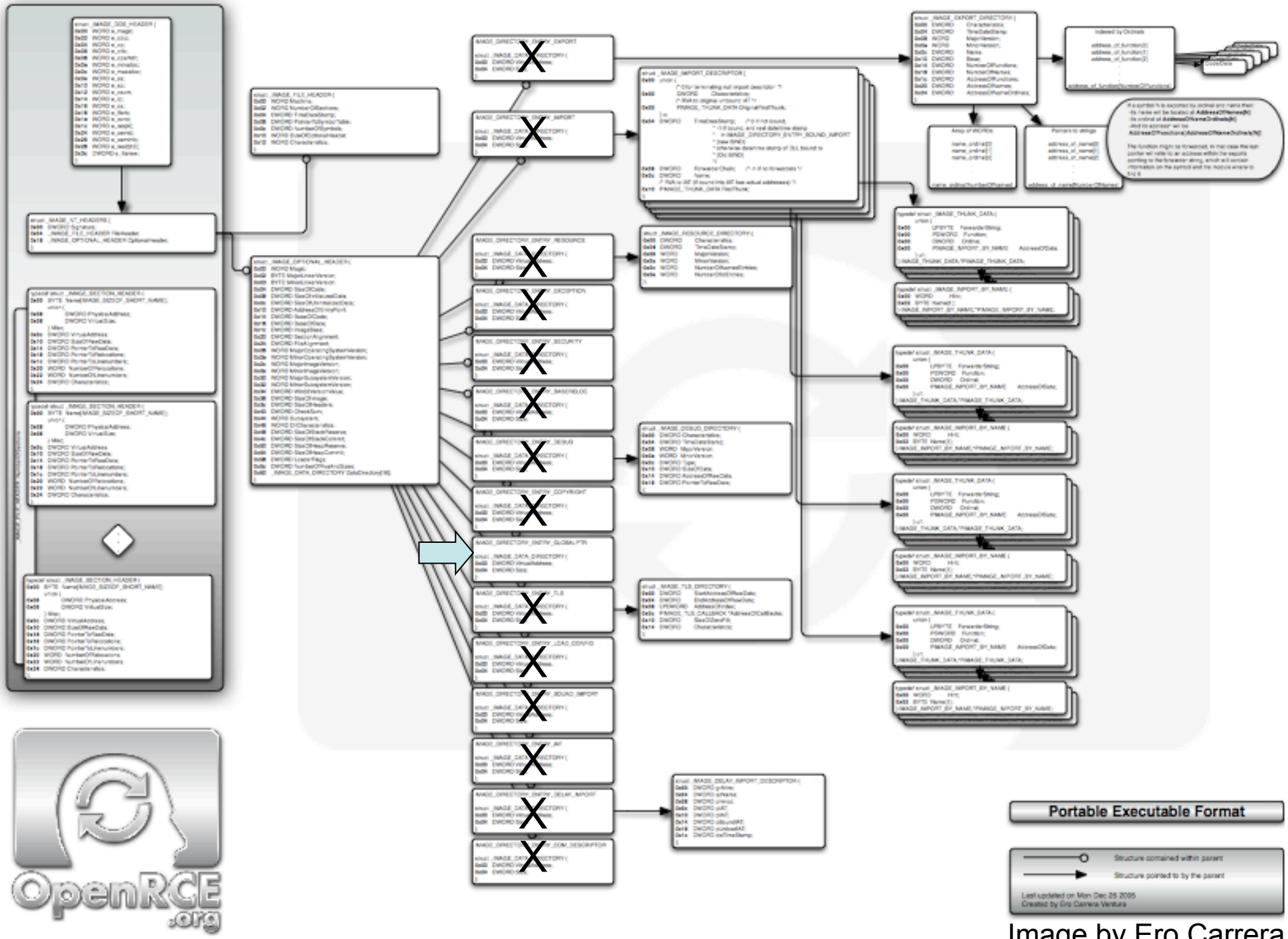


Image by Ero Carrera

IMAGE_DIRECTORY_ENTRY_ARCHITECTURE aka IMAGE_DIRECTORY_ENTRY_COPYRIGHT

- From Matt Pietrek's PE part 2 figures: http://msdn.microsoft.com/en-us/magazine/bb985997.aspx
- "Points to architecture-specific data, which is an array of IMAGE_ARCHITECTURE_HEADER structures. Not used for x86 or IA-64, but appears to have been used for DEC/Compaq Alpha."


Portable Executable Format
Structure contained within parent
Structure pointed to by the parent
Last updated on Mon Dec 28 2009
Created by Ero Carrera Ventura
OpenRCE.org
Image by Ero Carrera

IMAGE_DIRECTORY_ENTRY_GLOBALPTR

- From Matt Pietrek's PE part 2 figures: http://msdn.microsoft.com/en-us/magazine/bb985997.aspx
- "The VirtualAddress field is the RVA to be used as the global pointer (gp) on certain architectures. Not used on x86, but is used on IA-64. The Size field isn't used. See the November 2000 Under The Hood column for more information on the IA-64 gp."