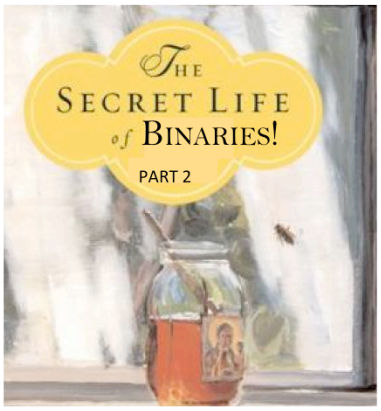


PE continued

Xeno Kovah – 2012
xkovah at gmail

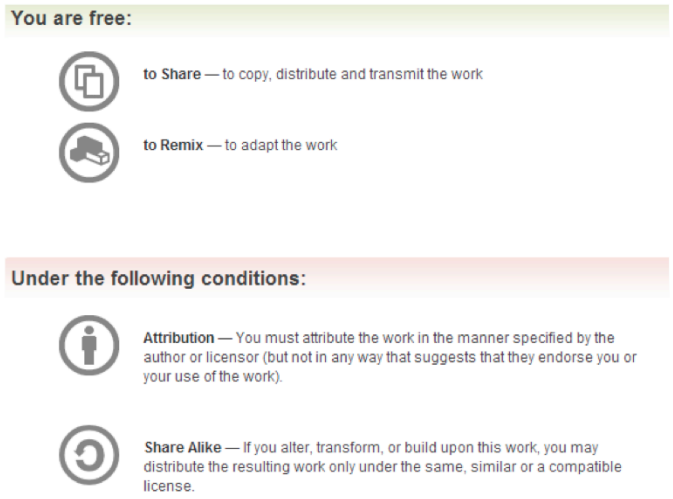
See notes for citation

1

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/



2

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

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

3

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.

4

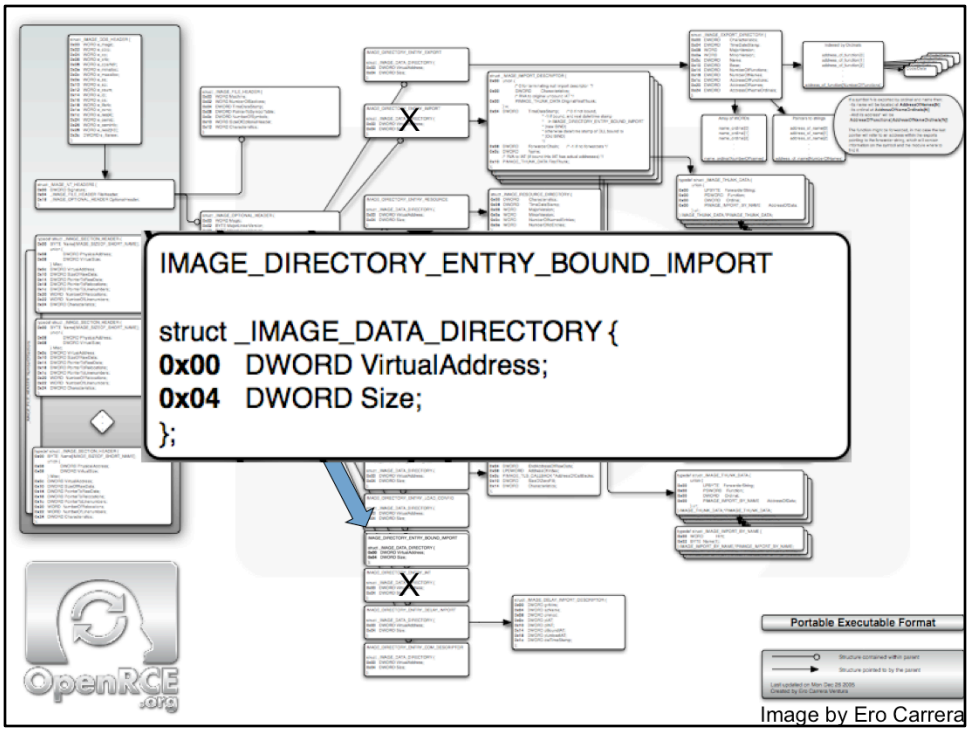


Image by Ero Carrera

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)

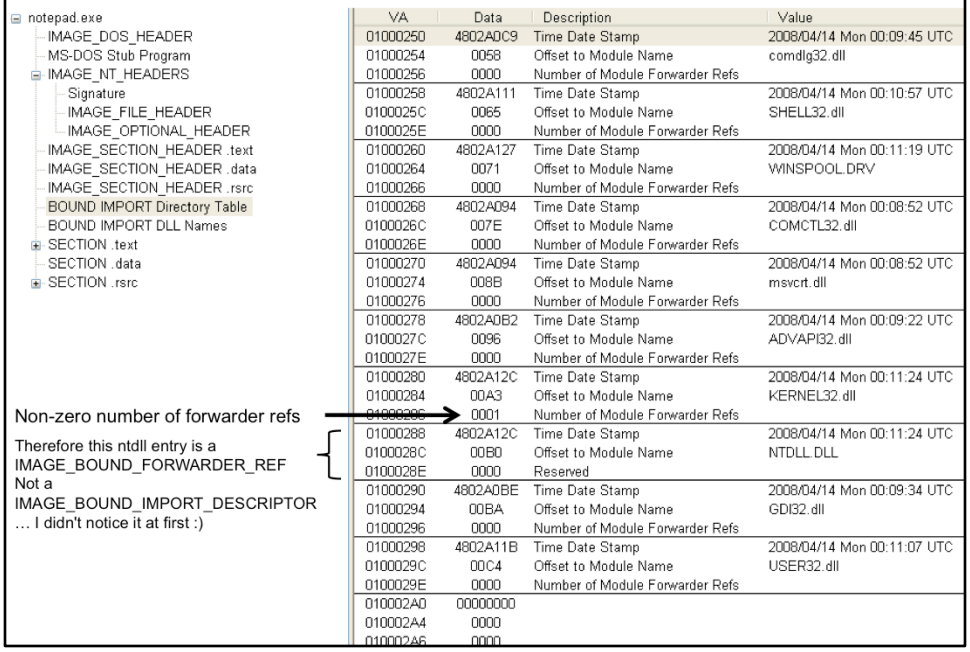
```
typedef struct _IMAGE_BOUND_IMPORT_DESCRIPTOR {
    DWORD   TimeDateStamp;
    WORD    OffsetModuleName;
    WORD    NumberOfModuleForwarderRefs;
// Array of zero or more IMAGE_BOUND_FORWARDER_REF follows
} IMAGE_BOUND_IMPORT_DESCRIPTOR,  *PIMAGE_BOUND_IMPORT_DESCRIPTOR;

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 Exports information of the DLL which is being imported from, as we will see later
- **OffsetModuleName** is *not* a base-relative 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.

7

Notepad.exe's IMAGE_BOUND_IMPORT_DESCRIPTOR array



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
[illegible]	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		

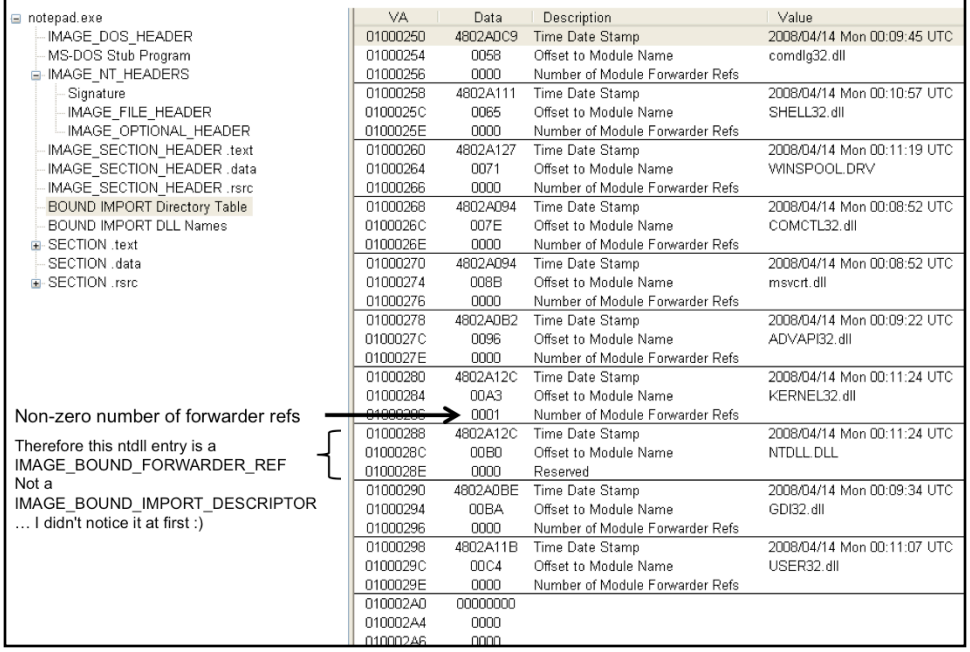
Teaching to the test

- Although I told you to ignore NumberOfModuleForwarderRefs…
- For purposes of the game later, when I ask you how many IMAGE_BOUND_IMPORT_DESCRIPTOR structures are in the bound import directory table, you should NOT count NumberOfModuleForwarderRefs worth of entries after a thing that has. Also don't count the null terminating entry

9

NEW 2012

So how many IMAGE_BOUND_IMPORT_DESCRIPTORs here?

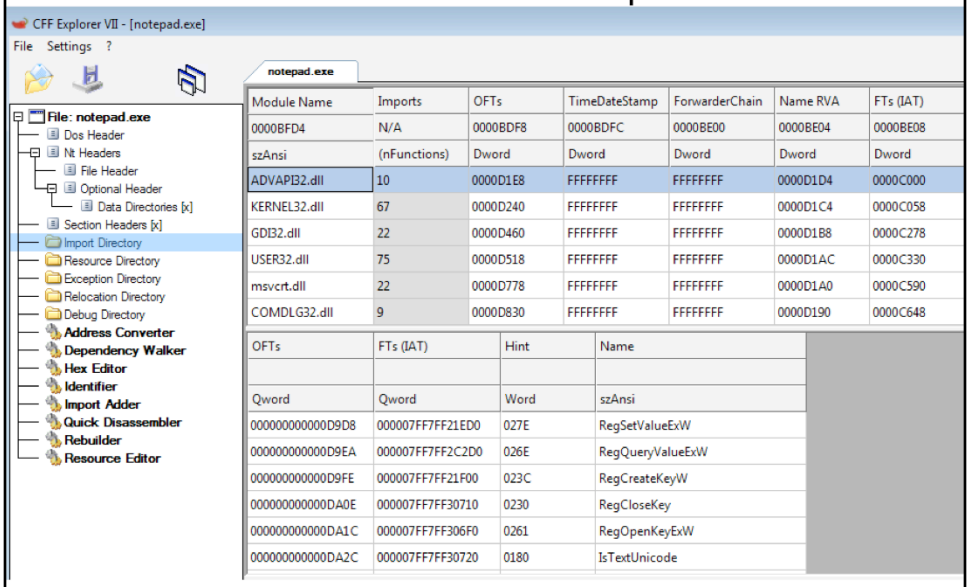


VA	Data	Description	Value
01000250	4802A0C9	Time Date Stamp	2008/04/14 Mon 00:09:45 UTC
01000254	0058	Offset to Module Name	comdlg32.dll
01000256	0000	Number of Module Forwarder Refs	
01000258	4802A111	Time Date Stamp	2008/04/14 Mon 00:10:57 UTC
0100025C	0065	Offset to Module Name	SHELL32.dll
0100025E	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
[illegible]	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		

NEW 2012

Answer = 9

Notepad.exe's IMAGE_IMPORT_DESCRIPTORs and IAT with bound imports



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
0000BFD4	N/A	0000BDF8	0000BDFC	0000BE00	0000BE04	0000BE08
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
ADVAPI32.dll	10	0000D1E8	FFFFFFFF	FFFFFFFF	0000D1D4	0000C000
KERNEL32.dll	67	0000D240	FFFFFFFF	FFFFFFFF	0000D1C4	0000C058
GDI32.dll	22	0000D460	FFFFFFFF	FFFFFFFF	0000D1B8	0000C278
USER32.dll	75	0000D518	FFFFFFFF	FFFFFFFF	0000D1AC	0000C330
msvcrt.dll	22	0000D778	FFFFFFFF	FFFFFFFF	0000D1A0	0000C590
COMDLG32.dll	9	0000D830	FFFFFFFF	FFFFFFFF	0000D190	0000C648

OFTs	FTs (IAT)	Hint	Name
Qword	Qword	Word	szAnsi
000000000000D9D8	000007FF7FF21ED0	027E	RegSetValueExW
000000000000D9EA	000007FF7FF2C2D0	026E	RegQueryValueExW
000000000000D9FE	000007FF7FF21F00	023C	RegCreateKeyW
000000000000DA0E	000007FF7FF30710	0230	RegCloseKey
000000000000DA1C	000007FF7FF306F0	0261	RegOpenKeyExW
000000000000DA2C	000007FF7FF30720	0180	IsTextUnicode

11

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.

12

Example things mentioning bind.exe
http://www.codeproject.com/KB/DLL/NeedBind.aspx
http://msdn.microsoft.com/en-us/magazine/cc301808.aspx

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.

13

See notes for citation

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

From winnt.h

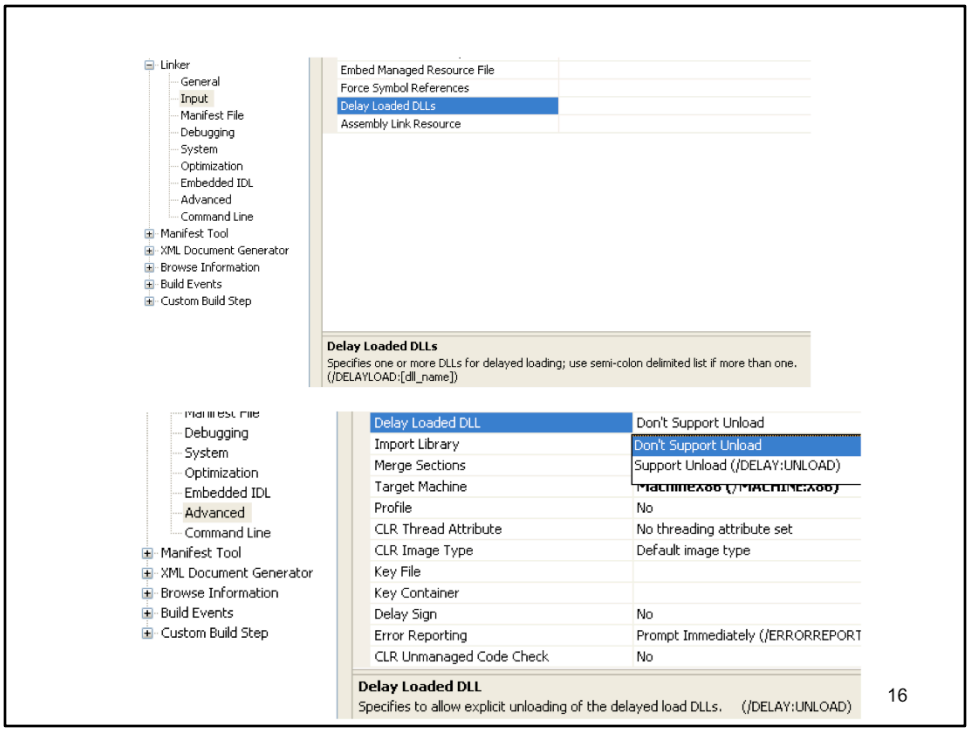
```
typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD    Magic;
    BYTE    MajorLinkerVersion;
    BYTE    MinorLinkerVersion;
    DWORD   SizeOfCode;
    DWORD   SizeOfInitializedData;
    DWORD   SizeOfUninitializedData;
    DWORD   AddressOfEntryPoint;
    DWORD   BaseOfCode;
    DWORD   BaseOfData;
    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?

14

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.


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
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])
Debugging
System
Optimization
Embedded IDL
Advanced
Command Line
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Delay Loaded DLL
Don't Support Unload
Import Library
Merge Sections
Target Machine
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
Don't Support Unload
Support Unload (/DELAY:UNLOAD)
Delay Loaded DLL
Specifies to allow explicit unloading of the delayed load DLLs. (/DELAY:UNLOAD)

16

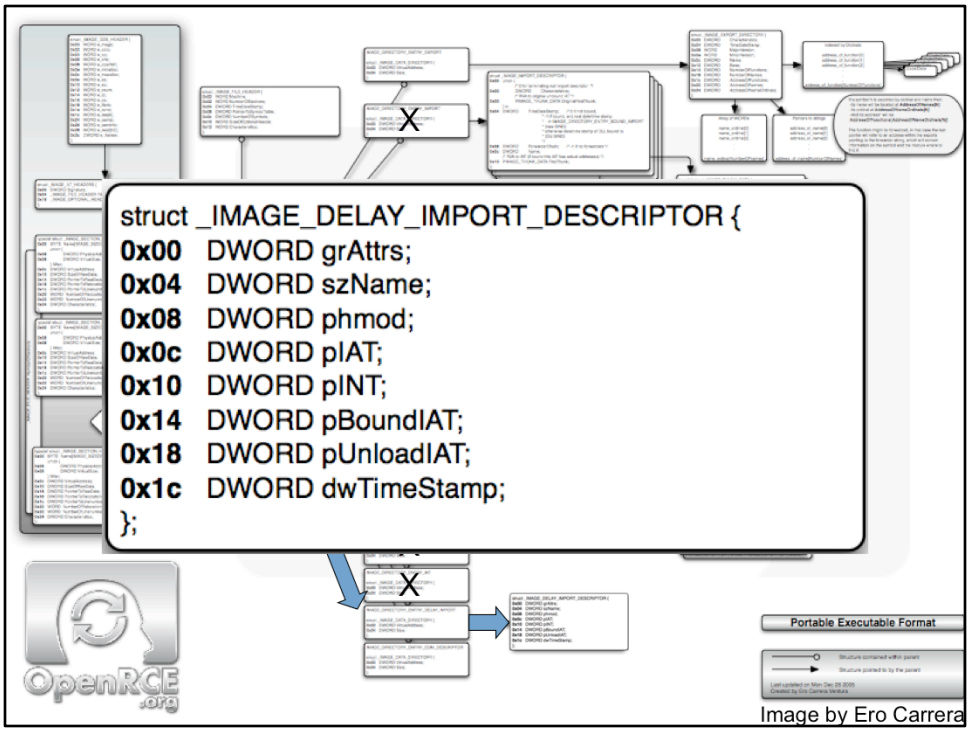


Image by Ero Carrera

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.

18

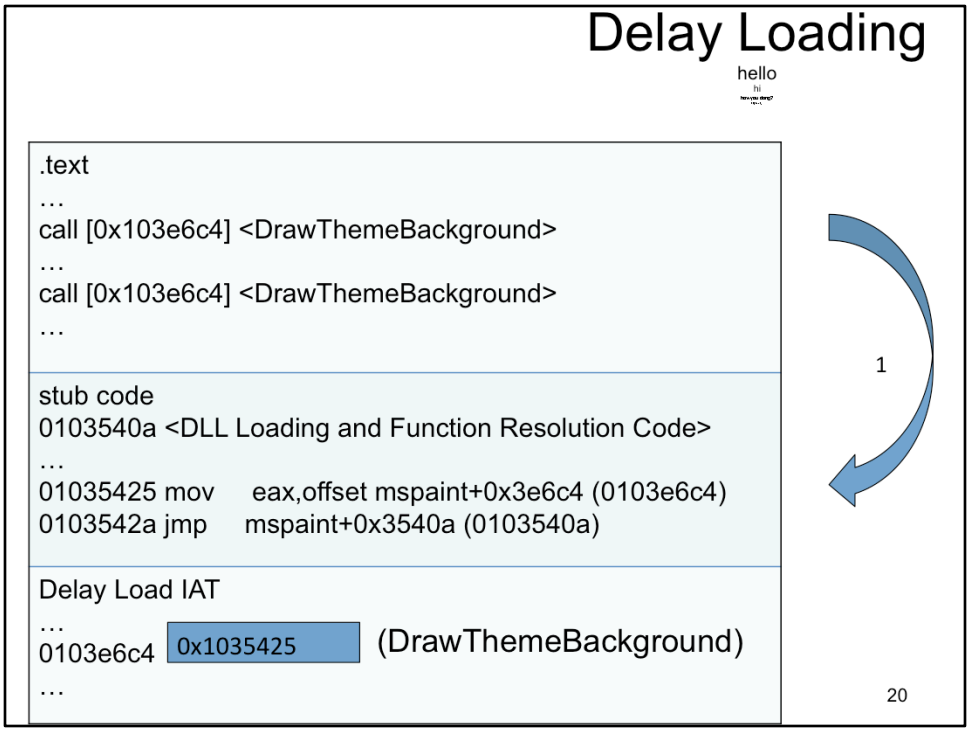
The Delay-Loaded IAT

- 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.

19

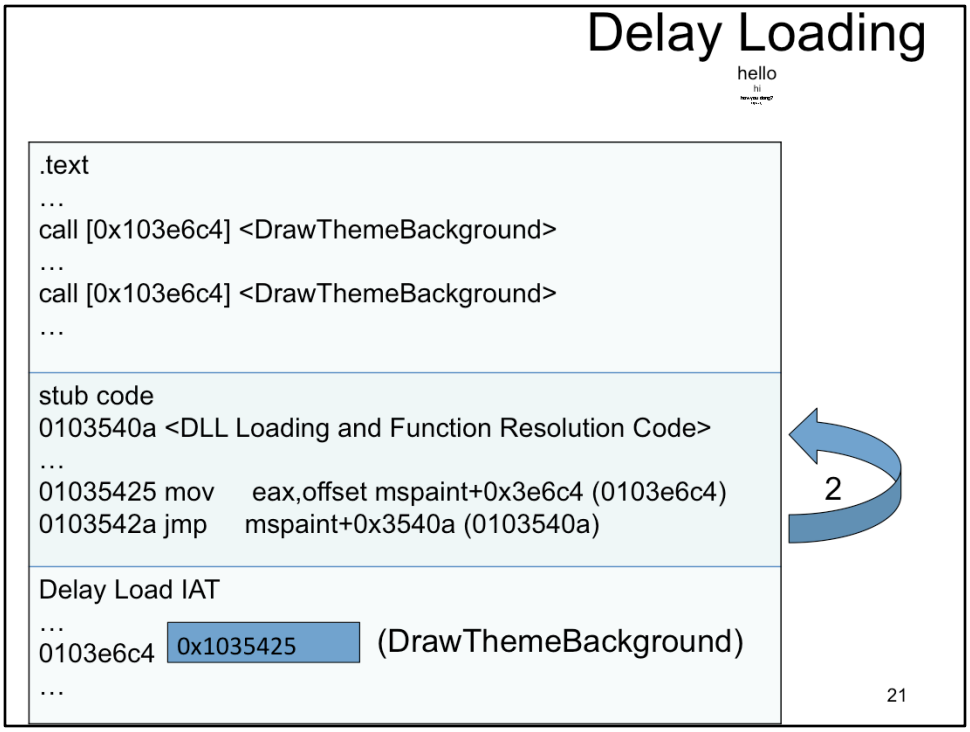
Delay Loading

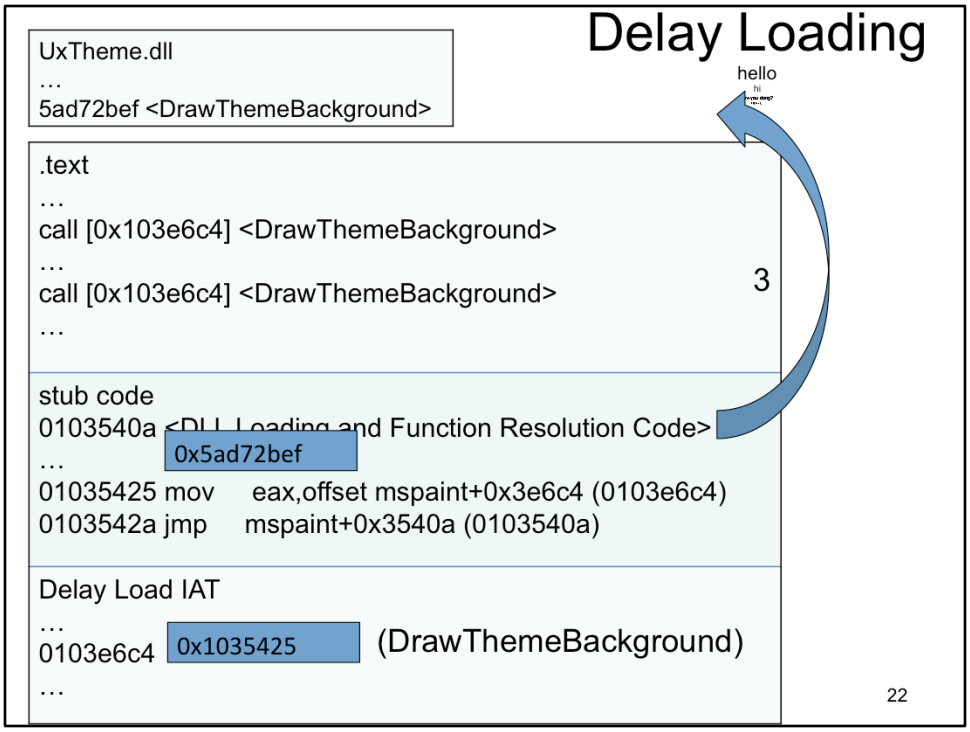
hello

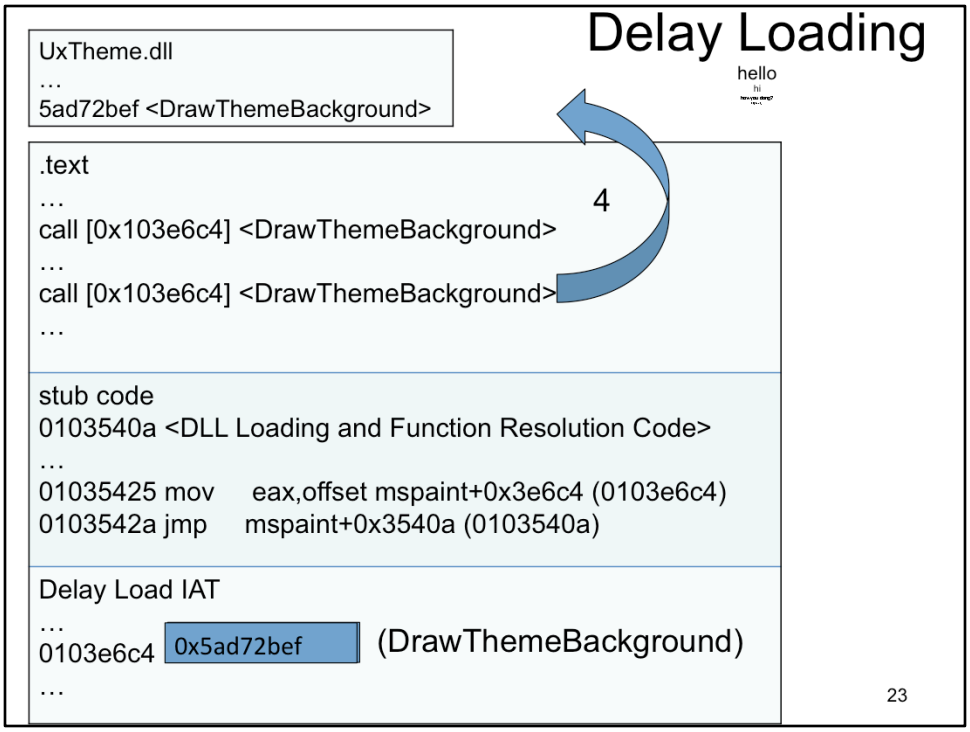


Delay Loading

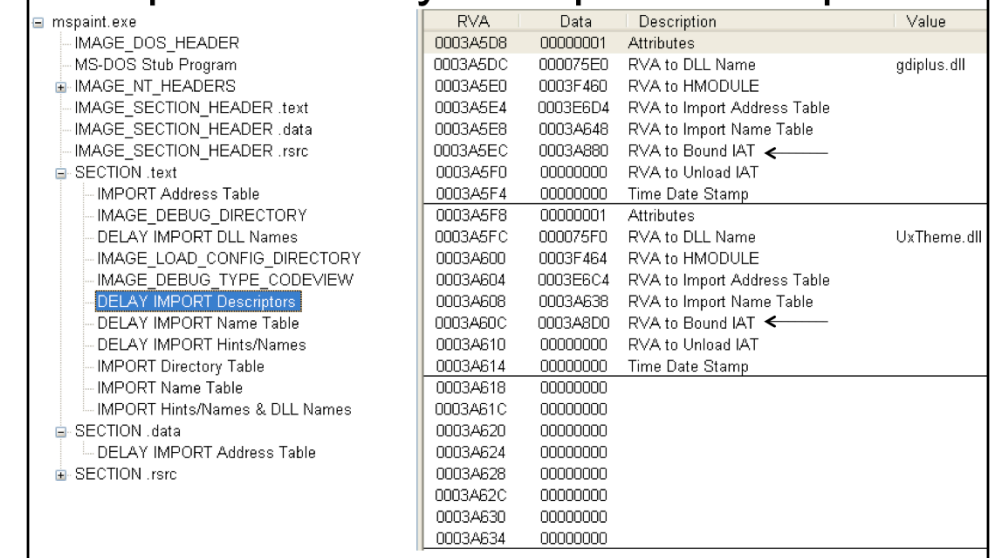
hello




Delay Loading
UxTheme.dll
…
5ad72bef <DrawThemeBackground>
hello
.text
…
call [0x103e6c4] <DrawThemeBackground>
…
call [0x103e6c4] <DrawThemeBackground>
…
3
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)
…
22


Delay Loading
hello
UxTheme.dll
…
5ad72bef <DrawThemeBackground>
.text
…
call [0x103e6c4] <DrawThemeBackground>
…
call [0x103e6c4] <DrawThemeBackground>
…
4
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)
…
23

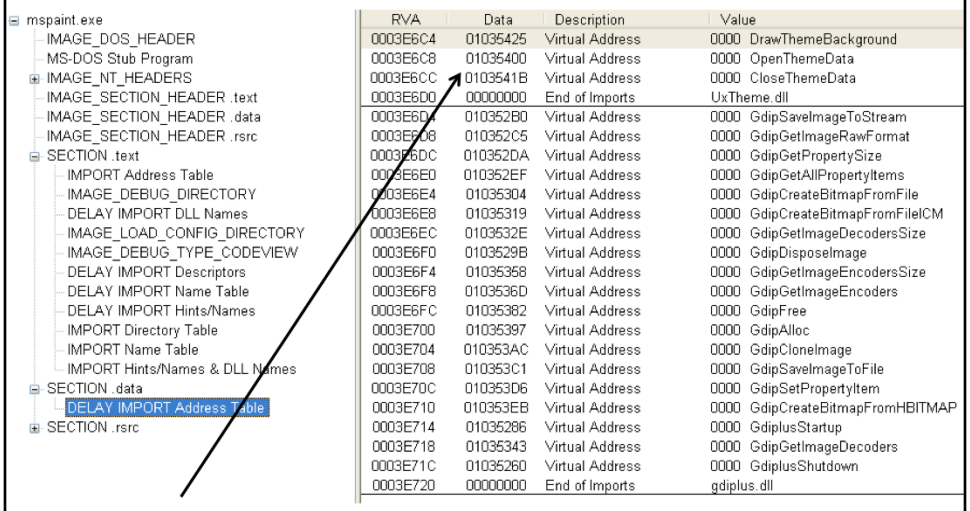
mspaint's delayed import descriptors



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.

24

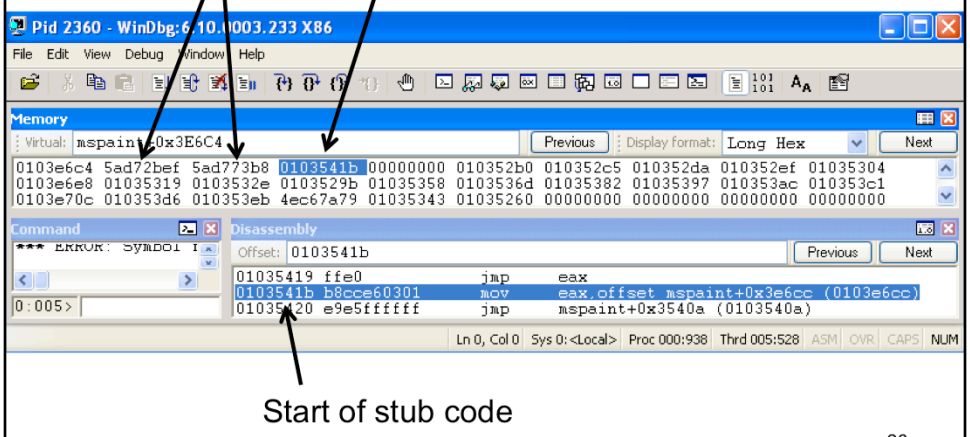
mspaint's delayed IAT



These are (absolute) 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.

25

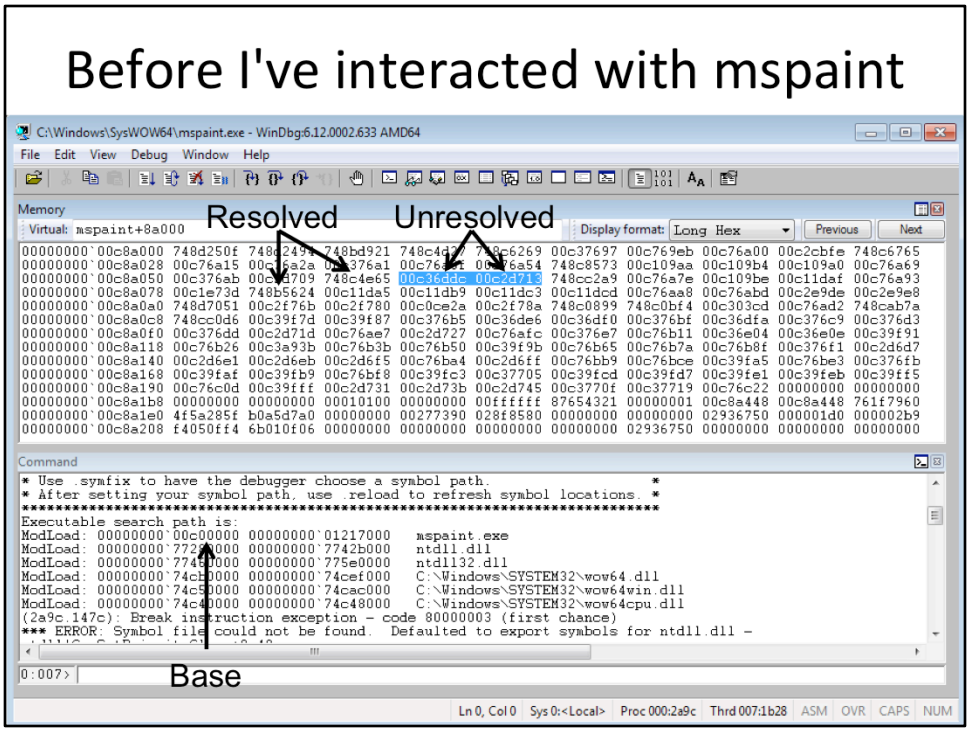
mspaint's delayed imports in memory (some resolved, some not)



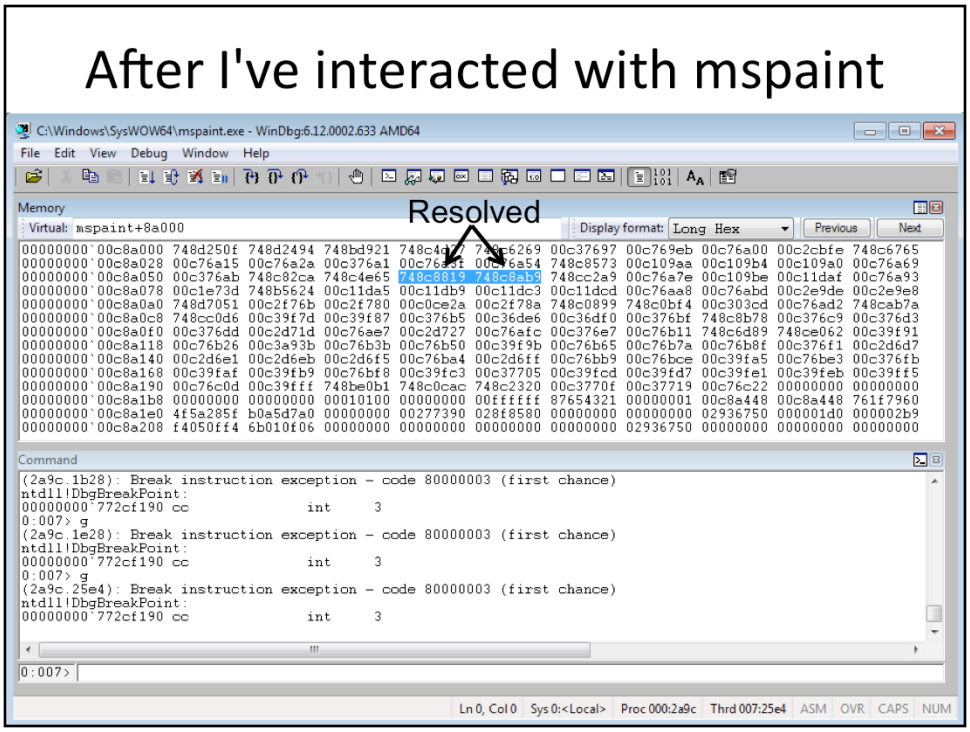
26

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

Before I've interacted with mspaint



After I've interacted with mspaint



To interact with mspaint, just mouse over the blank canvas

Get your geek on

- Play through round 5 on your own, and then wait for the seed for the class deathmatch
- This will be the hardest round so far, on account of the gotchas:
  - CFF Explorer doesn't expose delay-load DLL information, and PEView doesn't parse it (or bound import info) for 64 bit executables. You're going to have to manually interpret per the struct definitions

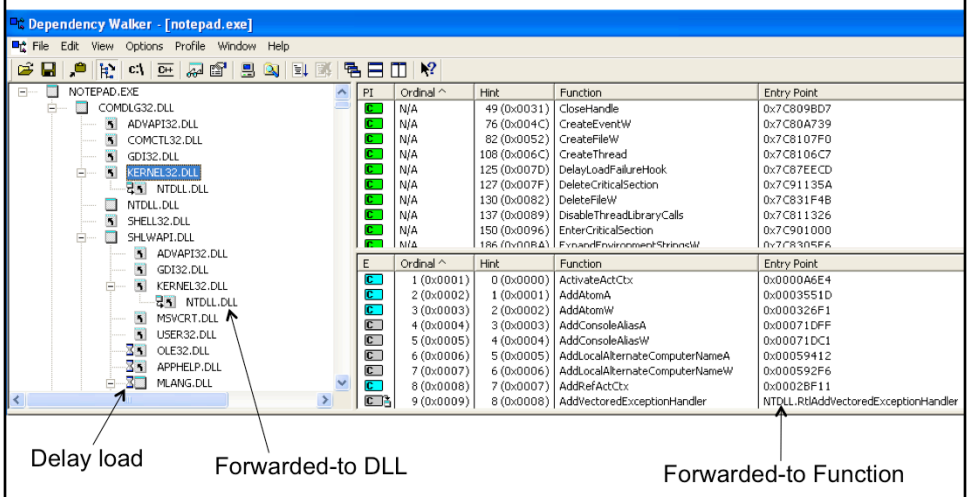
29

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

Dependency Walker, just 'cause

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



30

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.

31

TODO:

- Add picture & description of how rundll32.exe works

32

http://cheezburger.com/6678531328
https://i.chzbgr.com/completestore/12/10/16/TD8w0wqM5EetVkJaLdZROA2.jpg
https://i.chzbgr.com/completestore/12/10/12/SuxGUko-lUCvWyfpgOM6xQ2.jpg
https://i.chzbgr.com/completestore/12/10/7/C7UbJVtEHUmp-f9NS-A-Ag2.jpg

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.

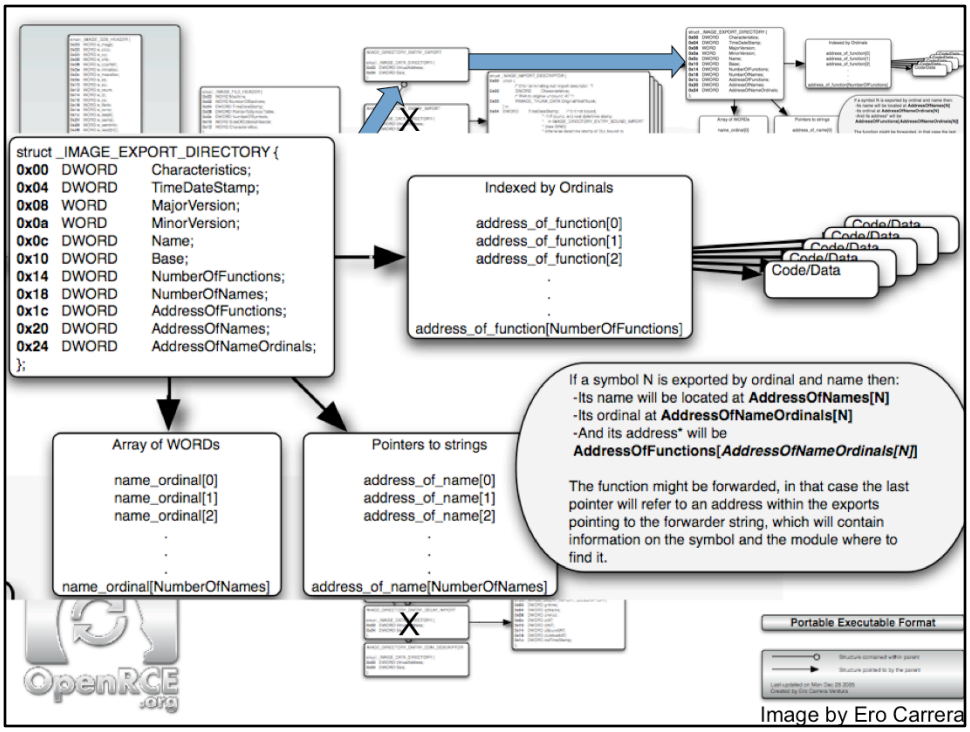


Image by Ero Carrera

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;
```

36

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** will be different from **NumberOfNames** when the file is exporting some functions by ordinal (talked about later). By knowing the number of names, when searching for an import by name, the loader can do a binary search.

37

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

38

Exports 4

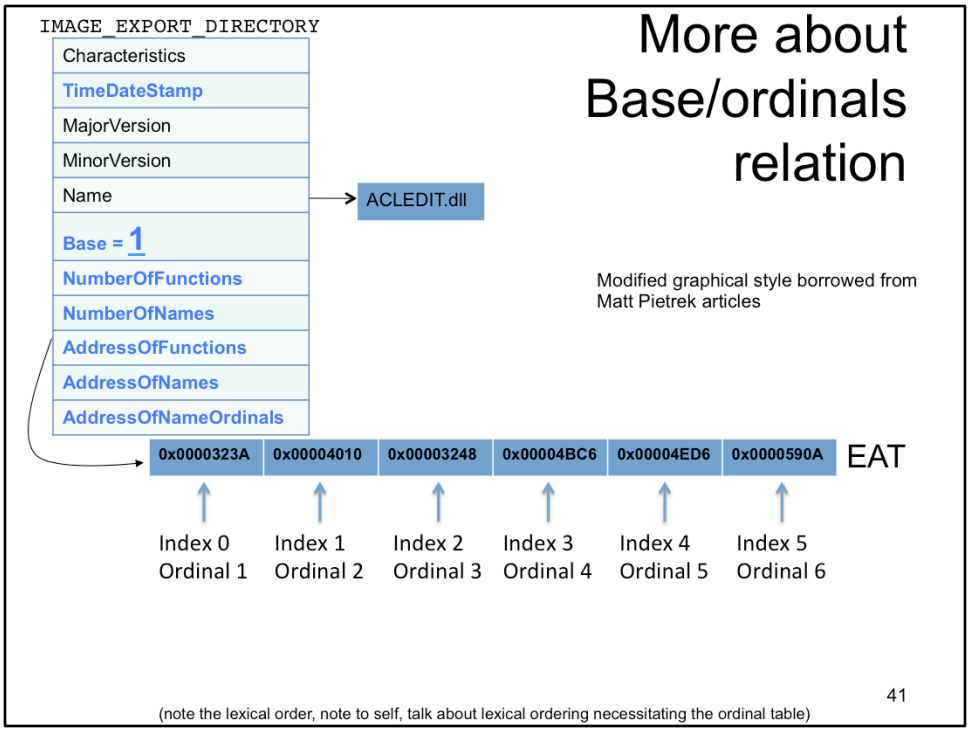
- **AddressOfNames** is an RVA which points to the beginning of an array which holds DWORD RVAs which point to the strings which specify function names. The pointed-to array should be NumberOfNames entries long. This would be the Export Names Table (ENT) like the flipside of the Import Names Table (INT).
- **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**.

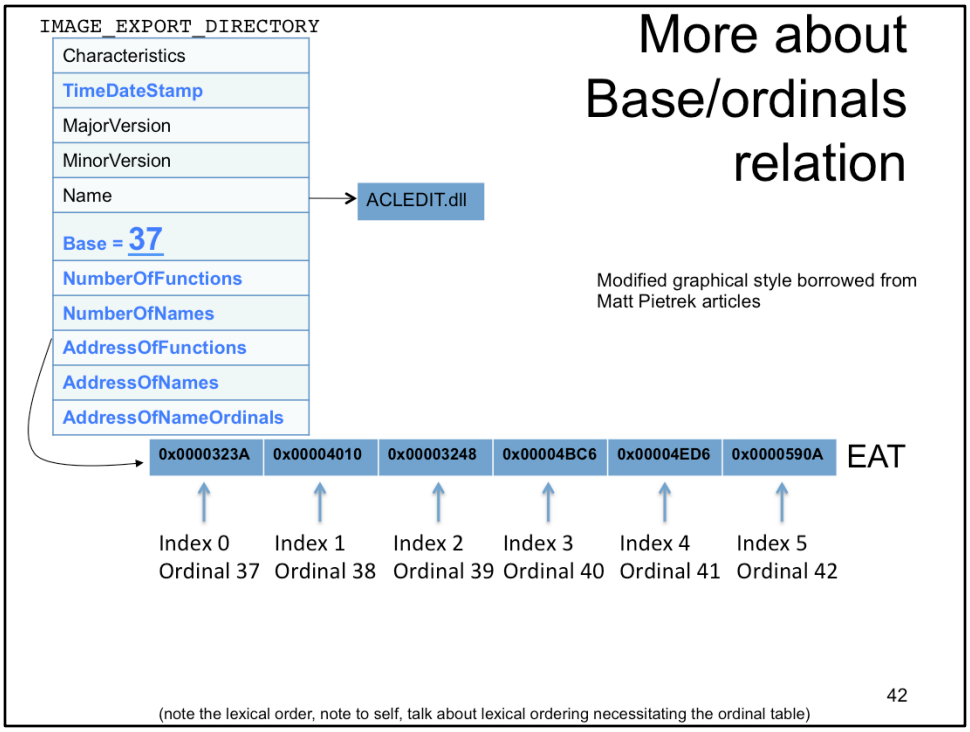
39

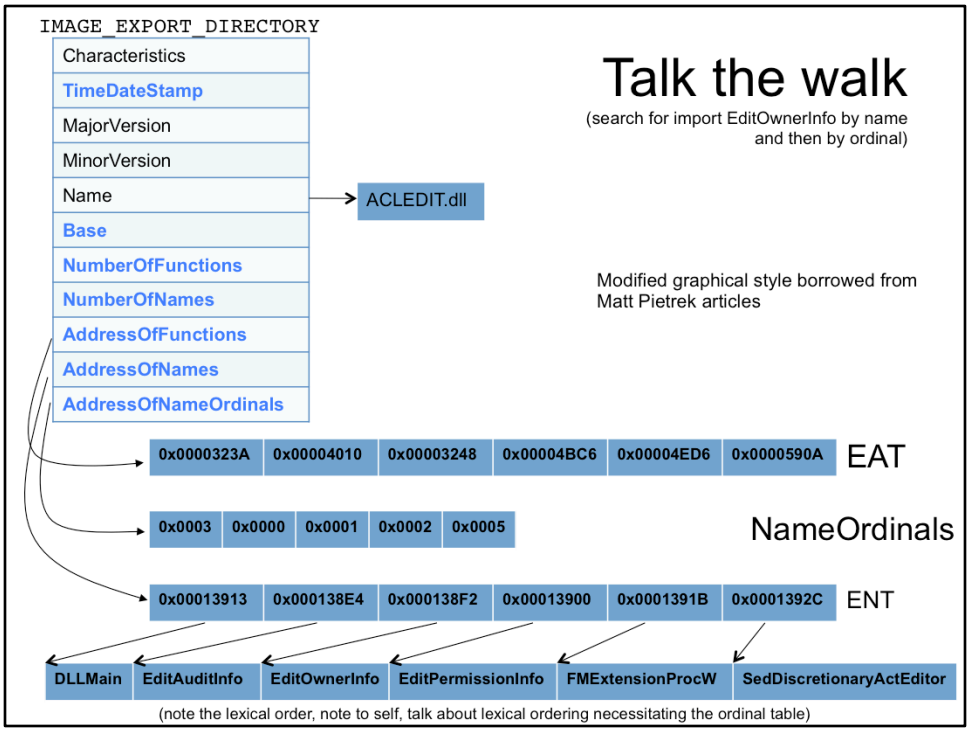
Ordinal says what?

- 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).

40


IMAGE_EXPORT_DIRECTORY
Characteristics
TimeDateStamp
MajorVersion
MinorVersion
Name
Base = 1
NumberOfFunctions
NumberOfNames
AddressOfFunctions
AddressOfNames
AddressOfNameOrdinals
ACLEDIT.dll
More about Base/ordinals relation
Modified graphical style borrowed from Matt Pietrek articles
0x0000323A
0x00004010
0x00003248
0x00004BC6
0x00004ED6
0x0000590A
EAT
Index 0 Ordinal 1
Index 1 Ordinal 2
Index 2 Ordinal 3
Index 3 Ordinal 4
Index 4 Ordinal 5
Index 5 Ordinal 6
41
(note the lexical order, note to self, talk about lexical ordering necessitating the ordinal table)


IMAGE_EXPORT_DIRECTORY
Characteristics
TimeDateStamp
MajorVersion
MinorVersion
Name
Base = 37
NumberOfFunctions
NumberOfNames
AddressOfFunctions
AddressOfNames
AddressOfNameOrdinals
ACLEDIT.dll
More about Base/ordinals relation
Modified graphical style borrowed from Matt Pietrek articles
0x0000323A
0x00004010
0x00003248
0x00004BC6
0x00004ED6
0x0000590A
EAT
Index 0 Ordinal 37
Index 1 Ordinal 38
Index 2 Ordinal 39
Index 3 Ordinal 40
Index 4 Ordinal 41
Index 5 Ordinal 42
42
(note the lexical order, note to self, talk about lexical ordering necessitating the ordinal table)


IMAGE_EXPORT_DIRECTORY
Characteristics
TimeDateStamp
MajorVersion
MinorVersion
Name
Base
NumberOfFunctions
NumberOfNames
AddressOfFunctions
AddressOfNames
AddressOfNameOrdinals
ACLEDIT.dll
Talk the walk
(search for import EditOwnerInfo by name and then by ordinal)
Modified graphical style borrowed from Matt Pietrek articles
0x0000323A
0x00004010
0x00003248
0x00004BC6
0x00004ED6
0x0000590A
EAT
0x0003
0x0000
0x0001
0x0002
0x0005
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?

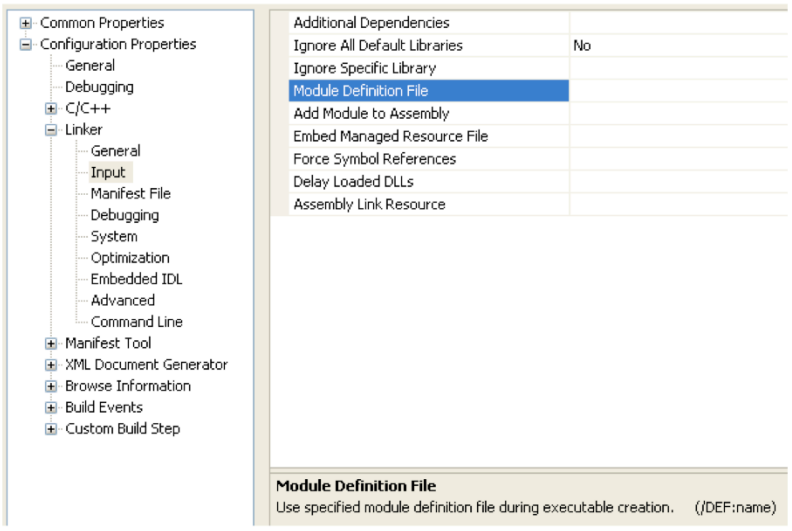
- http://msdn.microsoft.com/en-us/library/hyx1zcd3(VS.80).aspx
- “There are three methods for exporting a definition, listed in recommended order of use:
  - The __declspec(dllexport) keyword in the source code
  - An EXPORTS statement in a .def file
  - An /EXPORT specification in a LINK command”

44

From HelloWorldDLL's DllMain.c

```
__declspec(dllexport) void __cdecl SayHello(
    HWND hwnd,
    HINSTANCE hinst,
    LPSTR lpszCmdLine,
    int nCmdShow)
{
    MessageBox(0, "Hello World!", 0, 0);
}
```

Where to specify a .def file



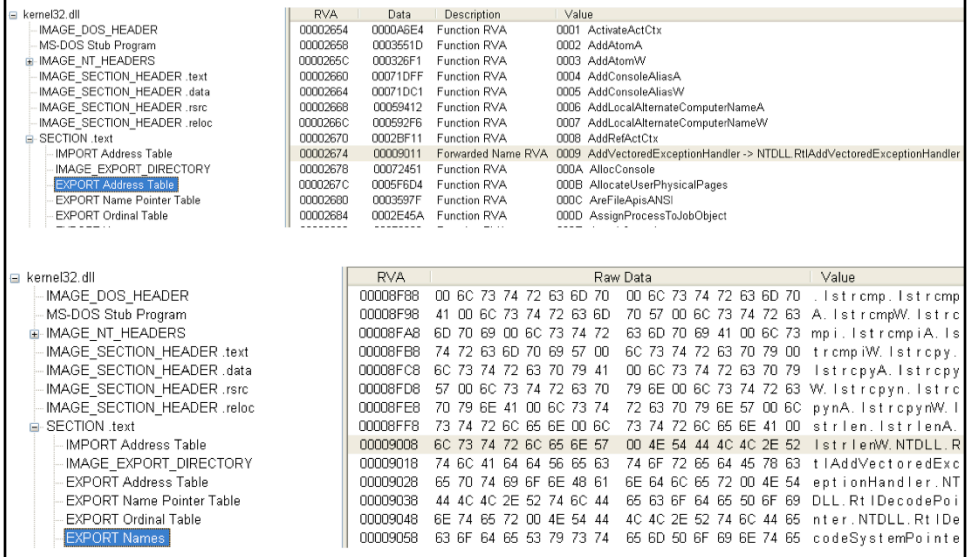
46

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

47

Kernel32.dll forwarded (to ntdll.dll) exports



48

How does one go about forwarding exports?

- Statement in .def file of the form

EXPORTS

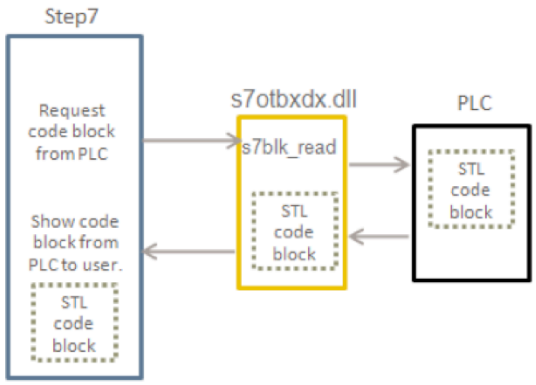
FunctionAlias=OtherDLLName.RealFunction

- 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



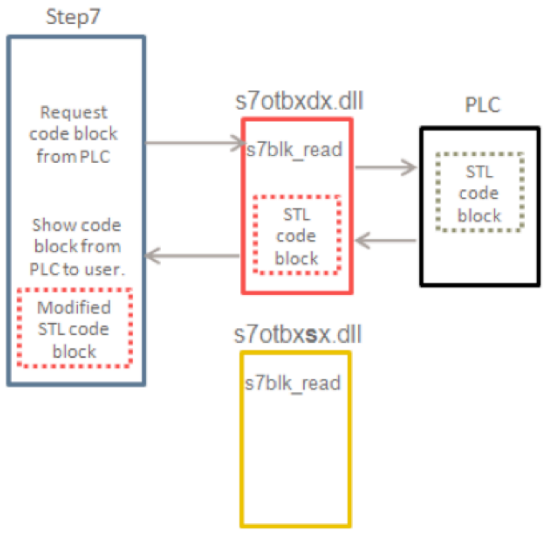
50

See notes for citation

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



See notes for citation

51

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.

52

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;
```

53

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?

See notes for citation

54

http://i43.tower.com/images/mm107041173/hot-shots-part-deux-charlie-sheen-dvd-cover-art.jpg

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.

55

EAT Hooking Lab

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

56

Get your geek on

- Play through round 6 on your own, and then wait for the seed for the class deathmatch

See notes for citation

57

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

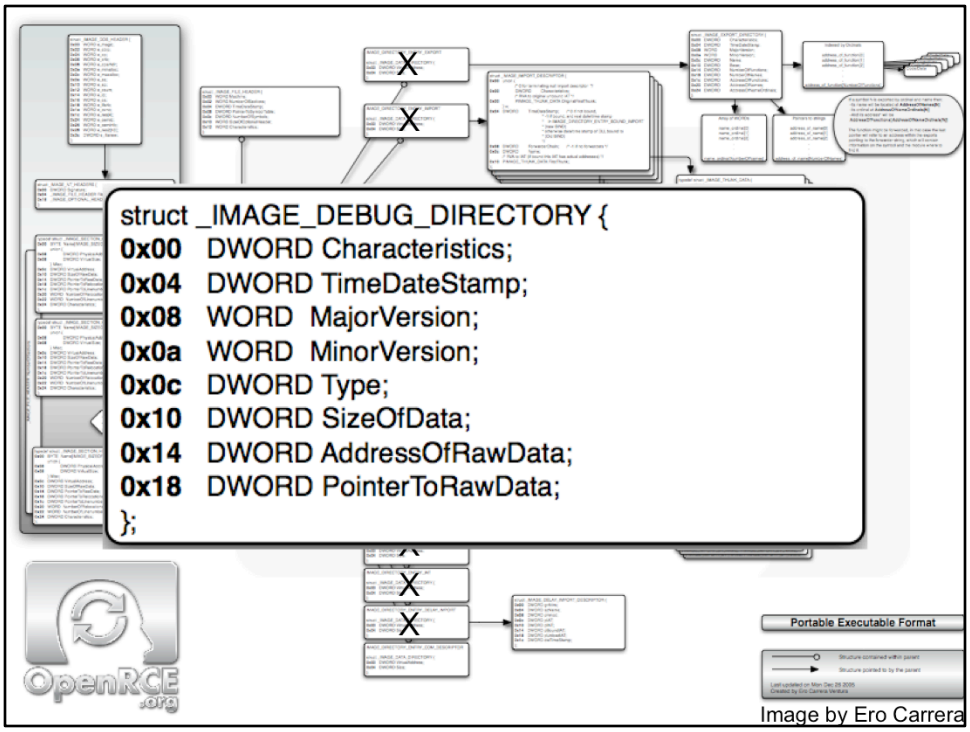


Image by Ero Carrera

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

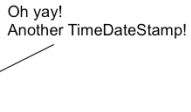
- **TimeDateStamp**, yet another to sanity check against. Should be the same as the one in the File Header I believe.
- **Type** and **SizeOfData** are what you would expect. The main Type we care about is IMAGE_DEBUG_TYPE_CODEVIEW as this is the common form now which points to a structure which holds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

60

Debug Info 3

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

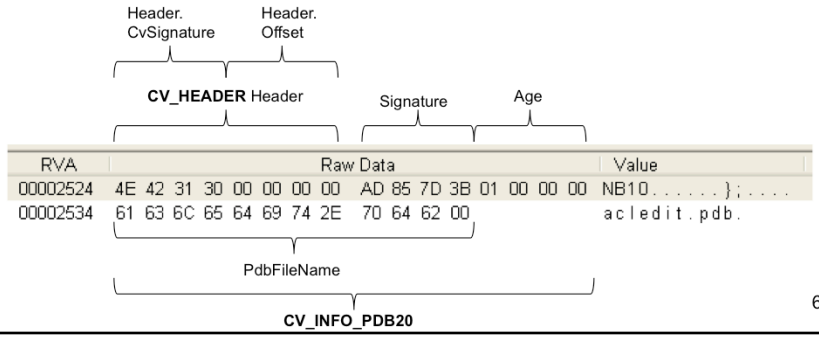
```
#define CV_SIGNATURE_NB10   '01BN'
#define CV_SIGNATURE_RSDS   'SDSR'
// CodeView header
struct CV_HEADER {
DWORD CvSignature; // NBxx
LONG  Offset;      // Always 0 for NB10
};
// CodeView NB10 debug information
// (used when debug information is stored in a PDB 2.00 file)
struct CV_INFO_PDB20 {
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
};
```



61

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	



62

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

63

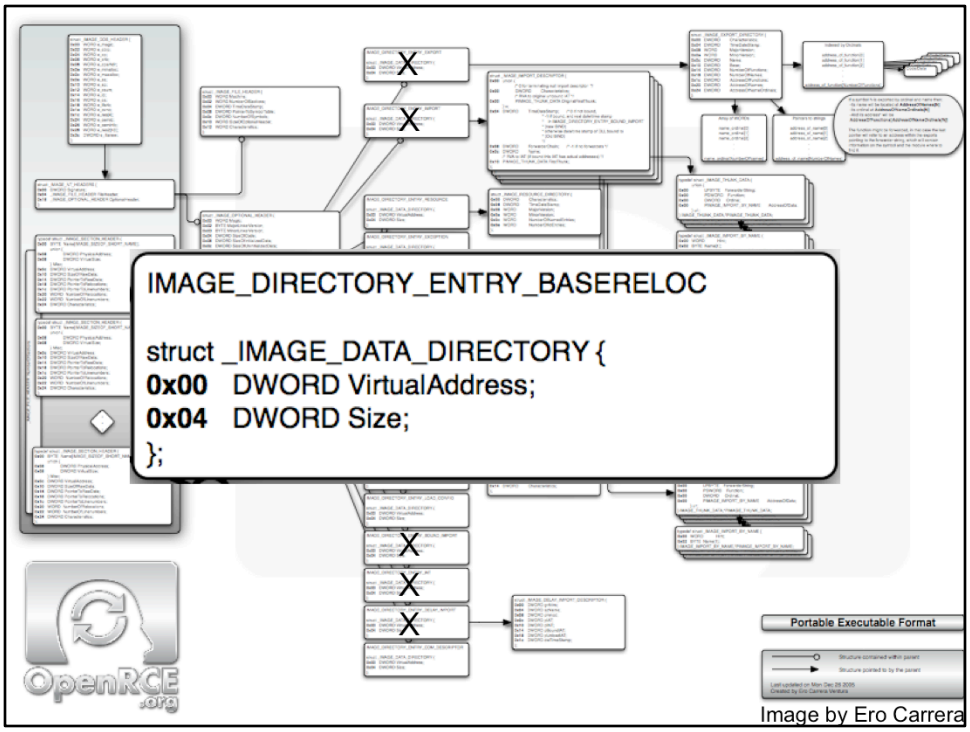


Image by Ero Carrera

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;
```

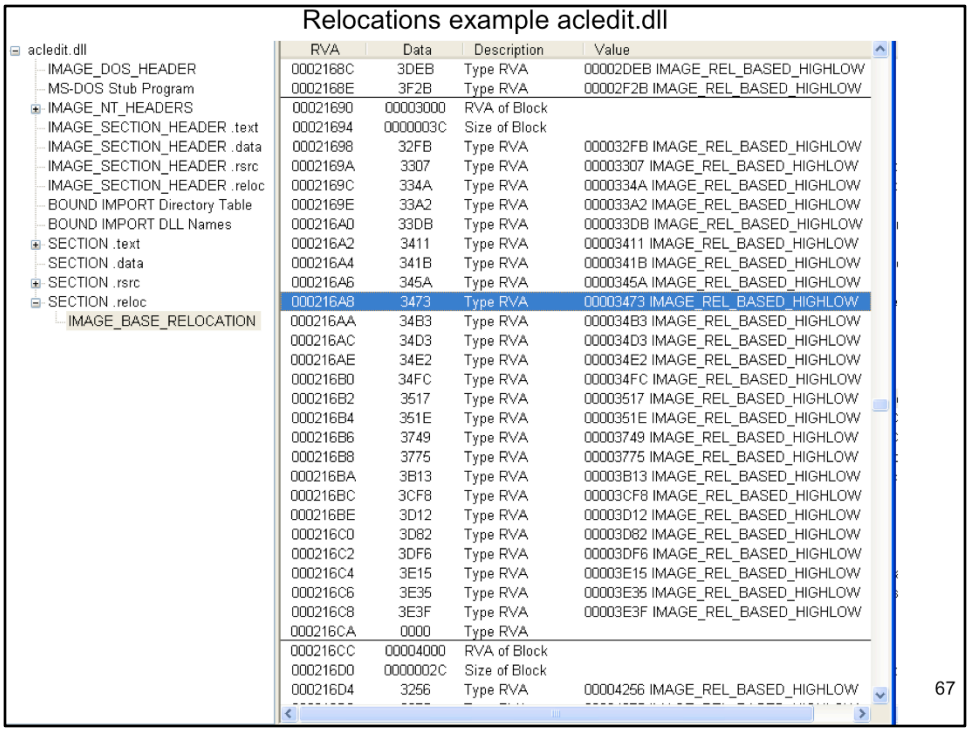
65

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).

66

Relocations example acledit.dll



67

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)

69

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.

71

Get your geek on

- Play through round 7 on your own, and then wait for the seed for the class deathmatch

See notes for citation

72

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

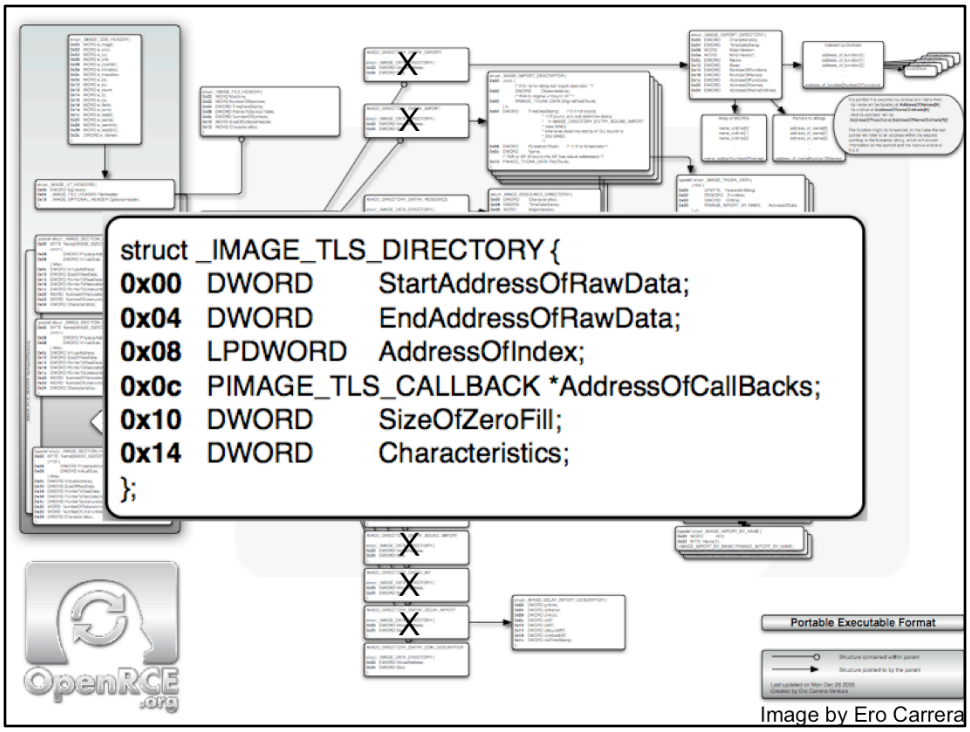


Image by Ero Carrera

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;
```

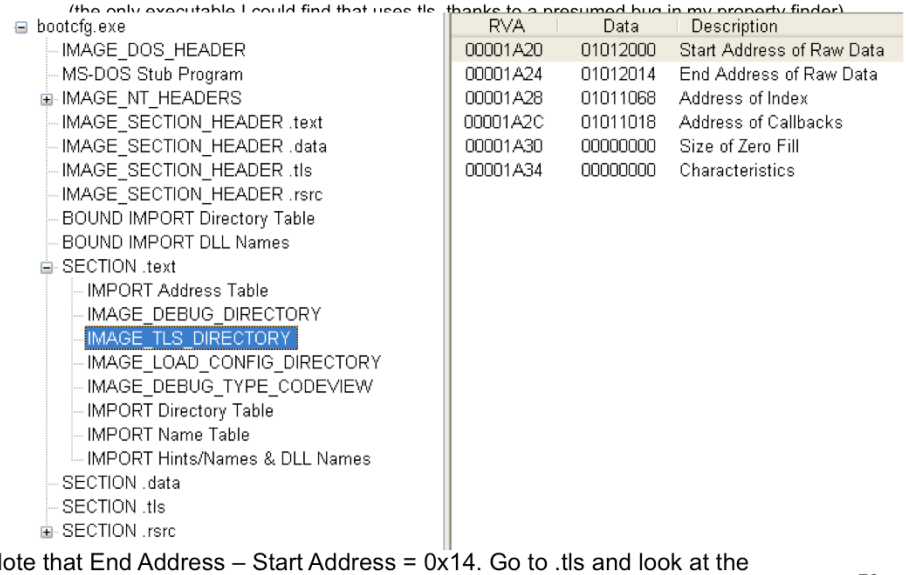
Thread Local Storage 2

- **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.

75

C:\WINDOWS\system32\bootcfg.exe

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



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

76

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

77

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 "uInfeat.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

78

Lab: TSL Callbacks

- Use Ilfak's example and Skywing's

79

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
  - In general you can find a ton of great documents at http://pferrie.tripod.com (I know, you thought tripod disappeared before geocities, right? ;))

Get your geek on

- Play through round 8 on your own, and then wait for the seed for the class deathmatch

See notes for citation

81

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

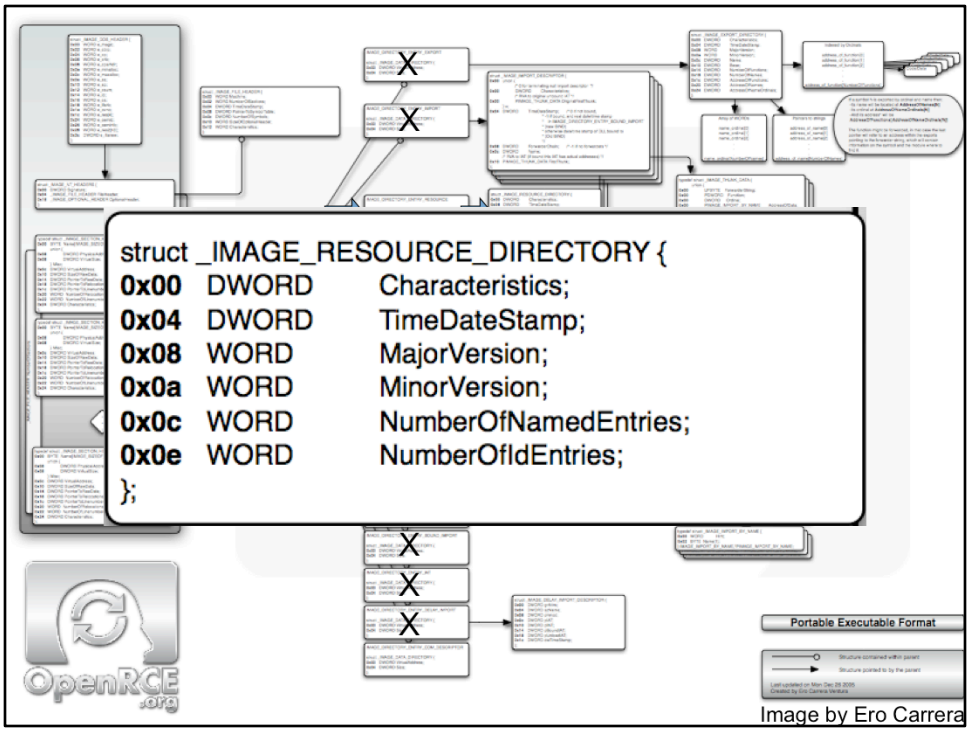


Image by Ero Carrera

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.

84

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

86

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.

87

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.

88

Get your geek on

- Play through round 9 on your own, and then wait for the seed for the class deathmatch

See notes for citation

89

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

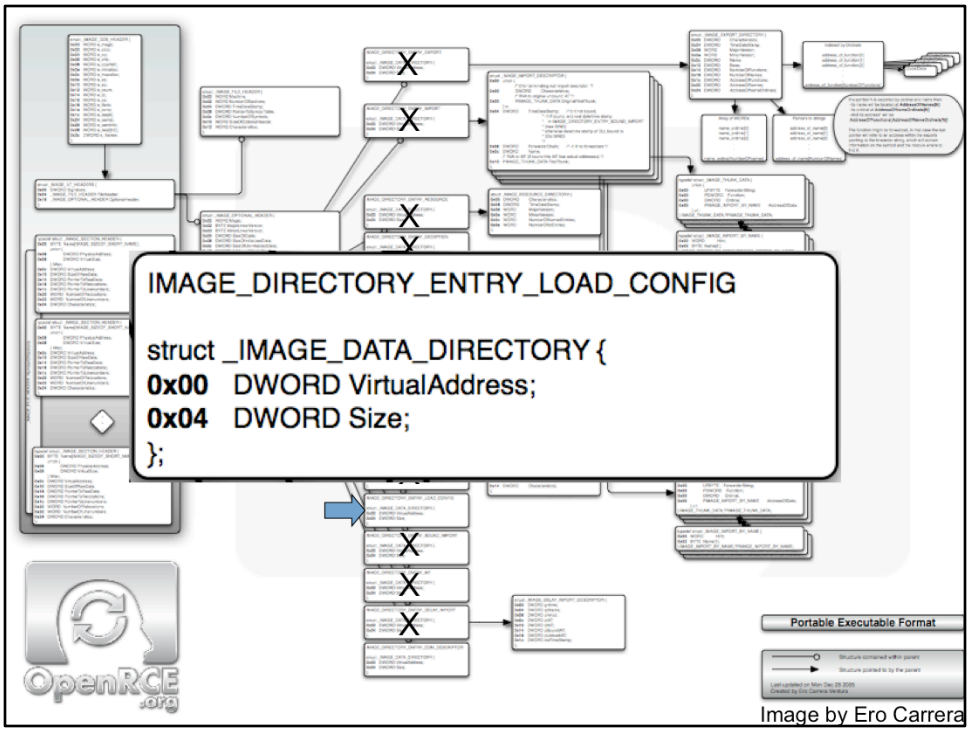


Image by Ero Carrera

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
```

91

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;
    ULONGLONG   DeCommitFreeBlockThreshold;
    ULONGLONG   DeCommitTotalFreeThreshold;
    ULONGLONG   LockPrefixTable;          // VA
    ULONGLONG   MaximumAllocationSize;
    ULONGLONG   VirtualMemoryThreshold;
    ULONGLONG   ProcessAffinityMask;
    DWORD       ProcessHeapFlags;
    WORD        CSDVersion;
    WORD        Reserved1;
    ULONGLONG   EditList;                 // VA
    ULONGLONG   SecurityCookie;           // VA
    ULONGLONG   SEHandlerTable;           // VA
    ULONGLONG   SEHandlerCount;
} IMAGE_LOAD_CONFIG_DIRECTORY64, *PIMAGE_LOAD_CONFIG_DIRECTORY64;
```

92

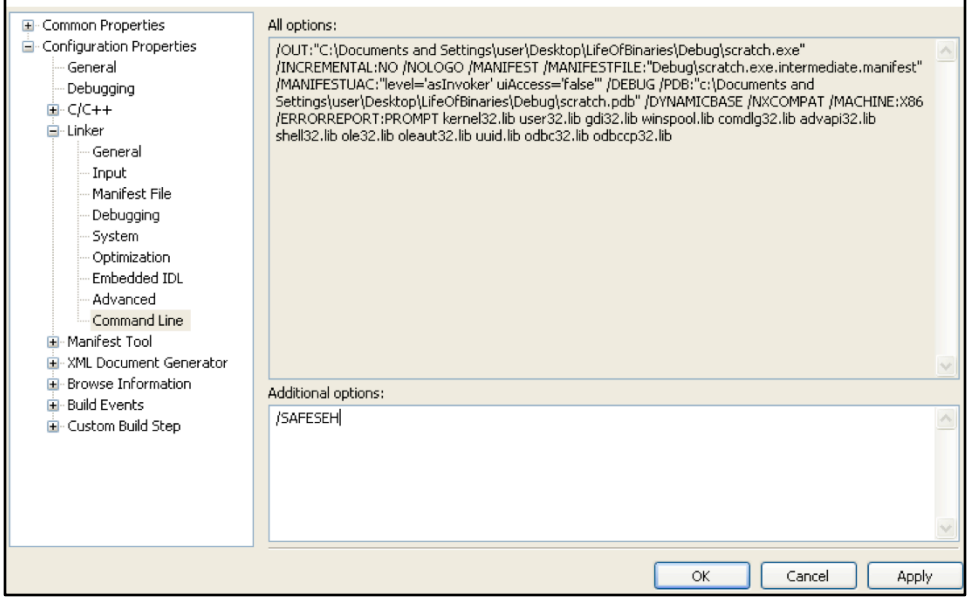
Load Config

- **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 (or fails to mitigate) 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

93

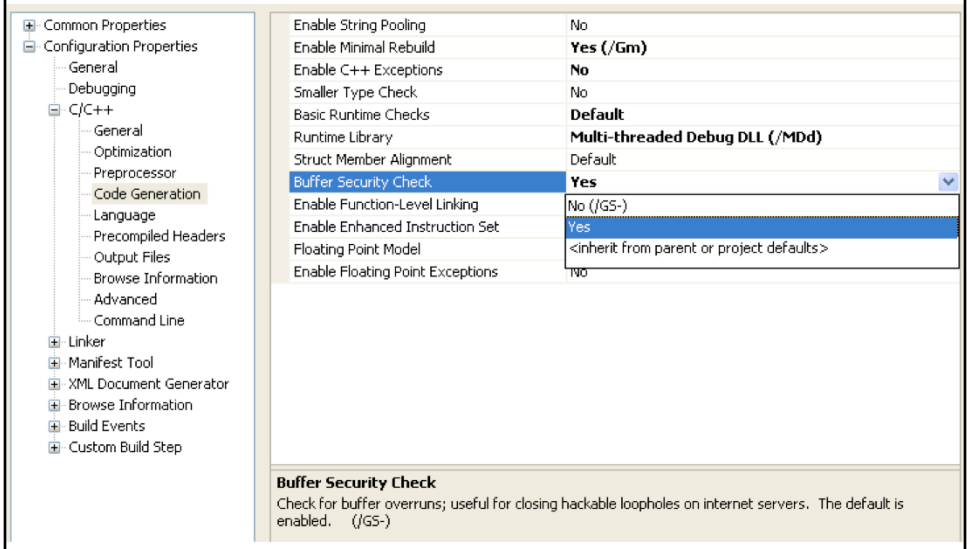
/SAFESEH

(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



/GS "stack cookie/canary" option

Helps detect stack buffer overflows



95

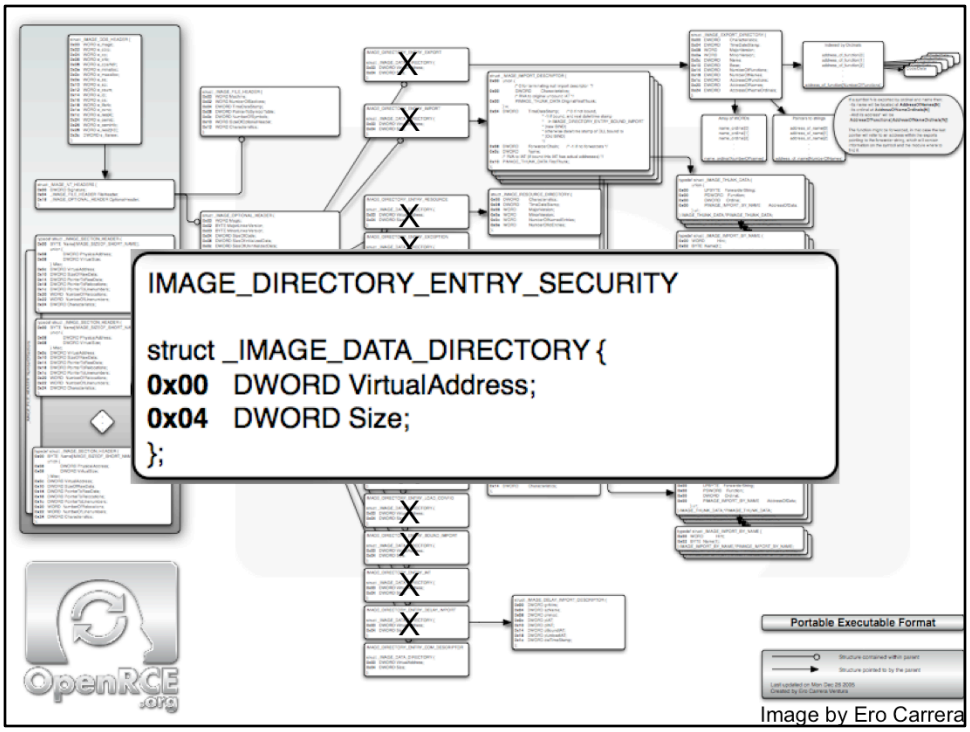


Image by Ero Carrera

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
- Remember way back on slide 30 we said there is that IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY characteristic which tells the loader to perform a digital signature check before it lets some code run? No? I can't believe you didn't remember that! It's like burned into my brain and I can't stop thinking about it no matter how hard I try!

Get your geek on

- Play through round 10 on your own, and then wait for the seed for the class deathmatch

See notes for citation

98

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

And the rest

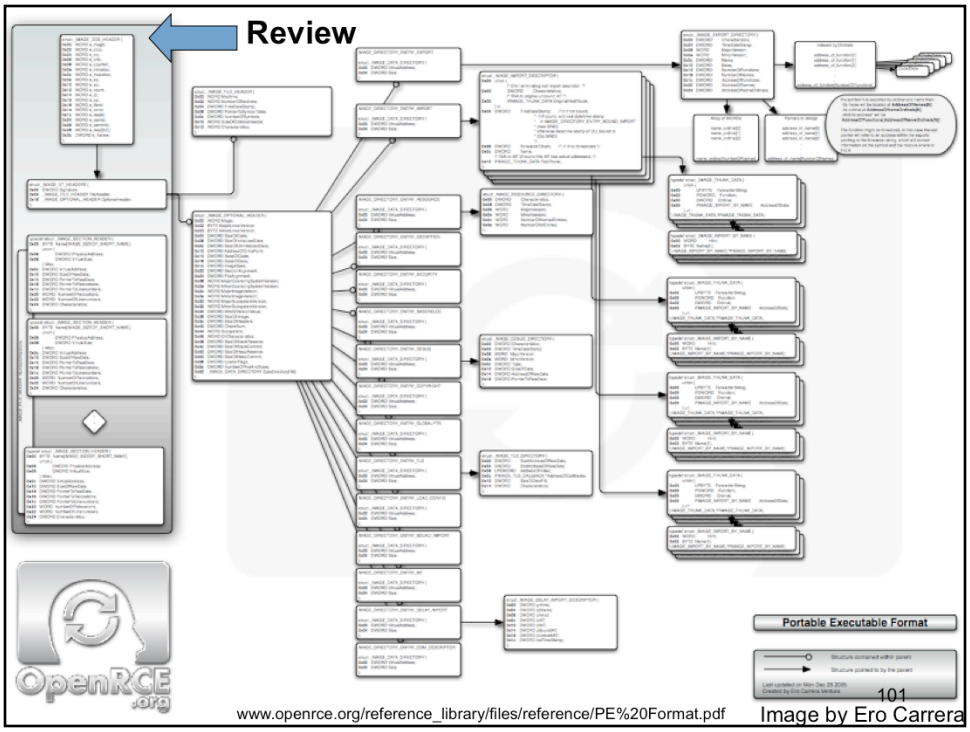
- Most of the rest of the DataDirectory[] entries don't even apply to x86, therefore they have been moved to the backup slides

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

100


Review
OpenRCE .org
Portable Executable Format
101
www.openrce.org/reference_library/files/reference/PE%20Format.pdf
Image by Ero Carrera