CSec15233
Malicious Software Analysis

Basic Static Analysis

Qasem Abu Al-Haija

Readings

1. Chapter 1. Basic Static Analysis
2. Notes:
   - PracticalMalwareAnalysis-Labs.7z, an encrypted 7-zip file containing the labs.

     The password is “malware”.

Static Analysis

- Static analysis describes the process of analyzing the code or structure of a program to determine its function.

- The program itself is not run at this time.

- You'll use several techniques to gather as much information as possible

Techniques

- Antivirus scanning
  - Using antivirus tools to confirm maliciousness
- Hashes
  - Using hashes to identify malware
- A file's strings, functions, and headers
  - Gleaning information from a file's strings, functions, & headers

Antivirus Scanning

Antivirus Scanning: Only a First Step

- Good first step to analyze malware is to run it through antivirus programs.

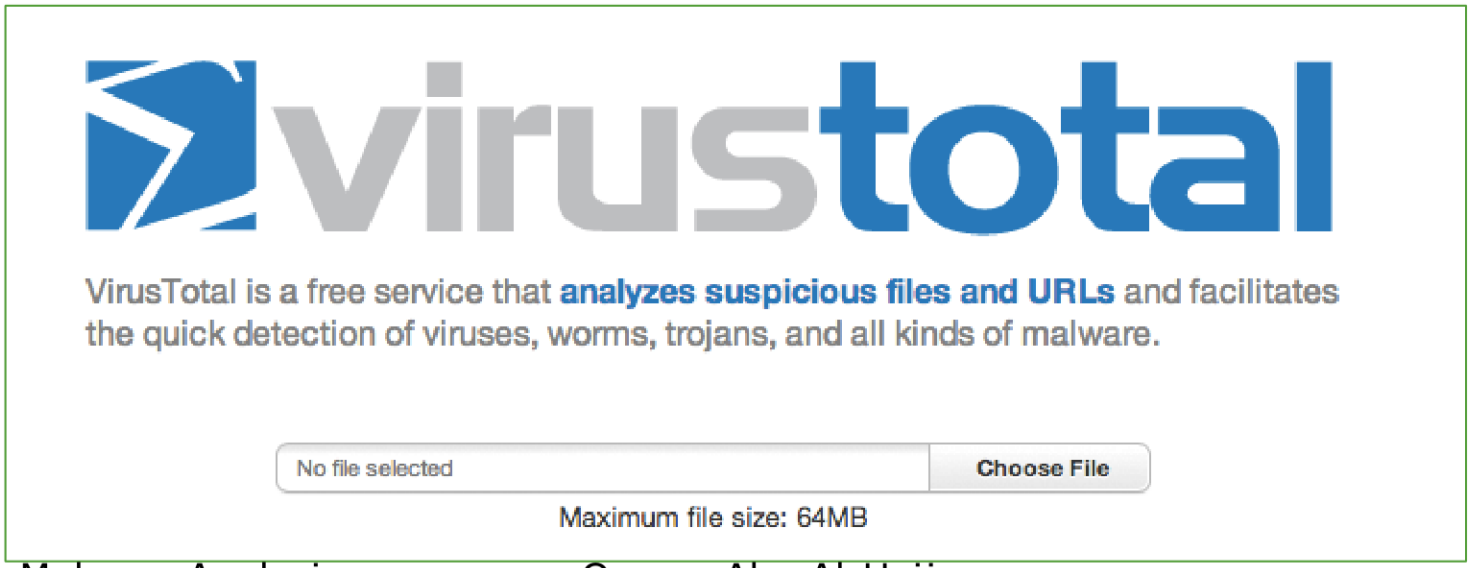
- Antivirus tools are certainly not perfect since they rely on:
  - A database of identifiable pieces of known suspicious code (file signatures).
  - A behavioral pattern-matching analysis (heuristics) to identify suspect files.

- Main Problems of Antivirus tools:
  - Malware can easily change its signature and fool the antivirus.
  - Rare malware is often undetected because it's simply not in the database.
  - Heuristics can be bypassed by new and unique malware.

- Since various antivirus programs use different signatures and heuristics.
  - it's better to run several different antivirus programs against the suspected file.
  - Example: VirusTotal website for scanning using several different antivirus programs.

Antivirus Scanning: Only a First Step

- **VirusTotal** allows you to upload a file for scanning by multiple antivirus engines.
  - http://www.virustotal.com
- **VirusTotal** generates a report that provides:
  - Total number of engines that mark the file as malicious,
  - The malware name and other information about the malware.
- **VirusTotal** is convenient, but using it may alert attackers that they've been caught



Hashing

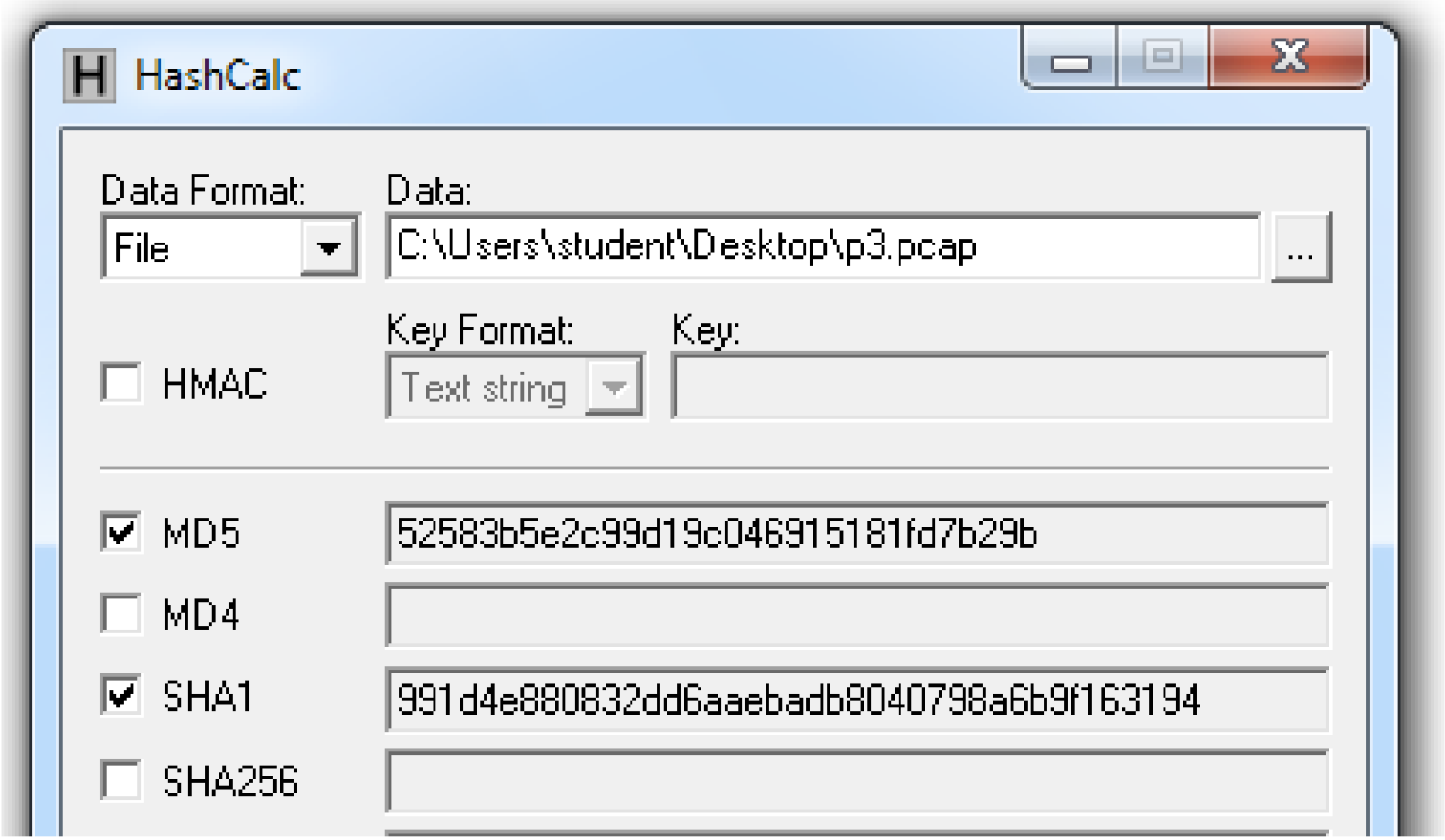
A fingerprint for malware

Hashes

- common method used to identify malware uniquely.
  - Condenses a file of any size down to a fixed-length fingerprint.
  - Produces a unique hash that identifies malware (fingerprint).

- MD5 or SHA-1 (or SHA-2), most common in MA.
  - Uniquely identifies a file well in practice
  - There are MD5 collisions, but they are not common
  - Collision: two different files with the same hash

HashCalc


HashCalc
Data Format:
File
Data:
C:\Users\student\Desktop\p3.pcap
...
HMAC
Key Format:
Text string
Key:
MD5
52583b5e2c99d19c046915181fd7b29b
MD4
SHA1
991d4e880832dd6aaebadb8040798a6b9f163194
SHA256

Hash Uses

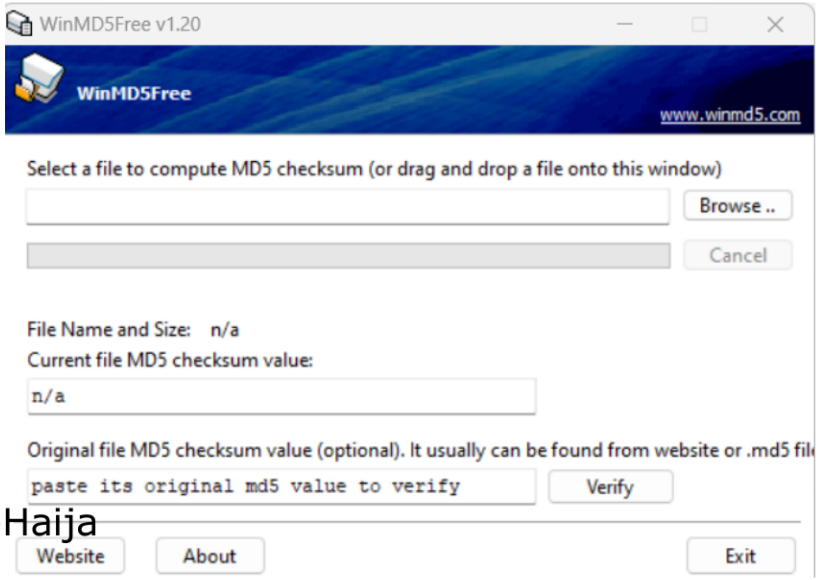
- Once you have a unique hash for a piece of malware, you can use it as follows:
  - Label a malware file
  - Share the hash with other analysts to identify malware
  - Search the hash online to see if someone else has already identified the file

Hashing: A Fingerprint for Malware

- Some other tools:
  - Command line: md5deep

```
C:\>md5deep c:\WINDOWS\system32\sol.exe
373e7a863a1a345c60edb9e20ec3231  c:\WINDOWS\system32\sol.exe
```

  - GUI tools: WinMD5Free



Finding Strings

Finding Strings

- Any sequence of printable characters is a **string.**
  - A program contains strings if it: Prints a message, connects to a URL, or copies a file to a specific location.
- Searching through the strings can be a simple way to get hints about the functionality of a program.
  - For example, if the program accesses a URL, you will see the URL accessed stored as a string in the program.

Finding Strings

- Strings program (http://bit.ly/ic4plL) to search an executable for strings
  - Typically stored in either ASCII or Unicode format.
  - Strings are terminated by a **null** (0x00)
- ASCII characters are 8 bits long
  - Now called ANSI
- Unicode characters are 16 bits long
  - Microsoft calls them "wide characters"

Finding Strings

- Both ASCII and Unicode formats store characters in sequences that end with a NULL terminator.
  - NULL terminator indicates that the string is complete.
- ASCII strings use 1 byte per character, and Unicode uses 2 bytes per character.
  - Figure 2-2 shows the string BAD stored as ASCII.
  - Figure 2-3 shows the string BAD stored as Unicode.

- **ASCII**

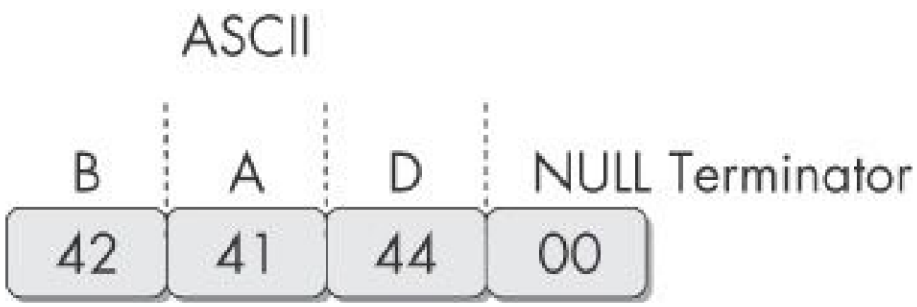


Figure 2-2. ASCII representation of the string BAD

- **UNICODE**

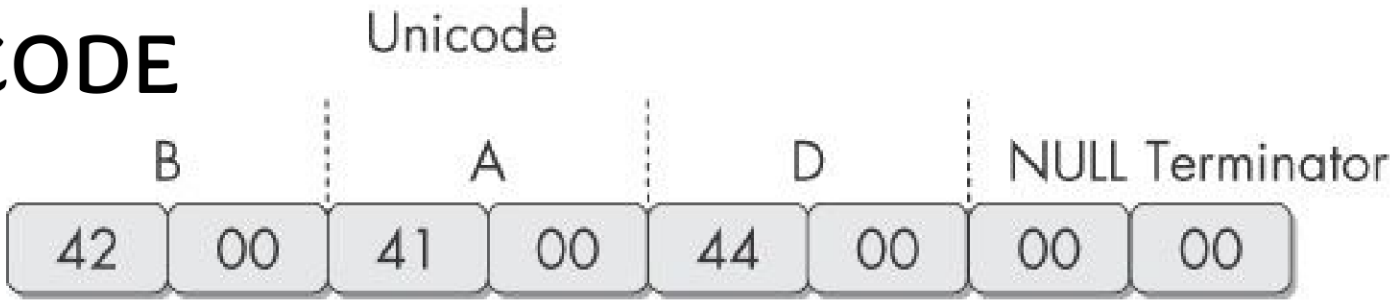


Figure 2-3. Unicode representation of the string BAD

How Strings Work

- **Strings** is Native to Linux and also available for Windows.
- When it searches an executable for ASCII/Unicode strings:
  - It ignores context and formatting
  - It analyzes any file type and detects strings across an entire file.
  - Finds all strings in a file 3 or more characters long.
  - Every string should be followed by a string terminator.

How Strings Work

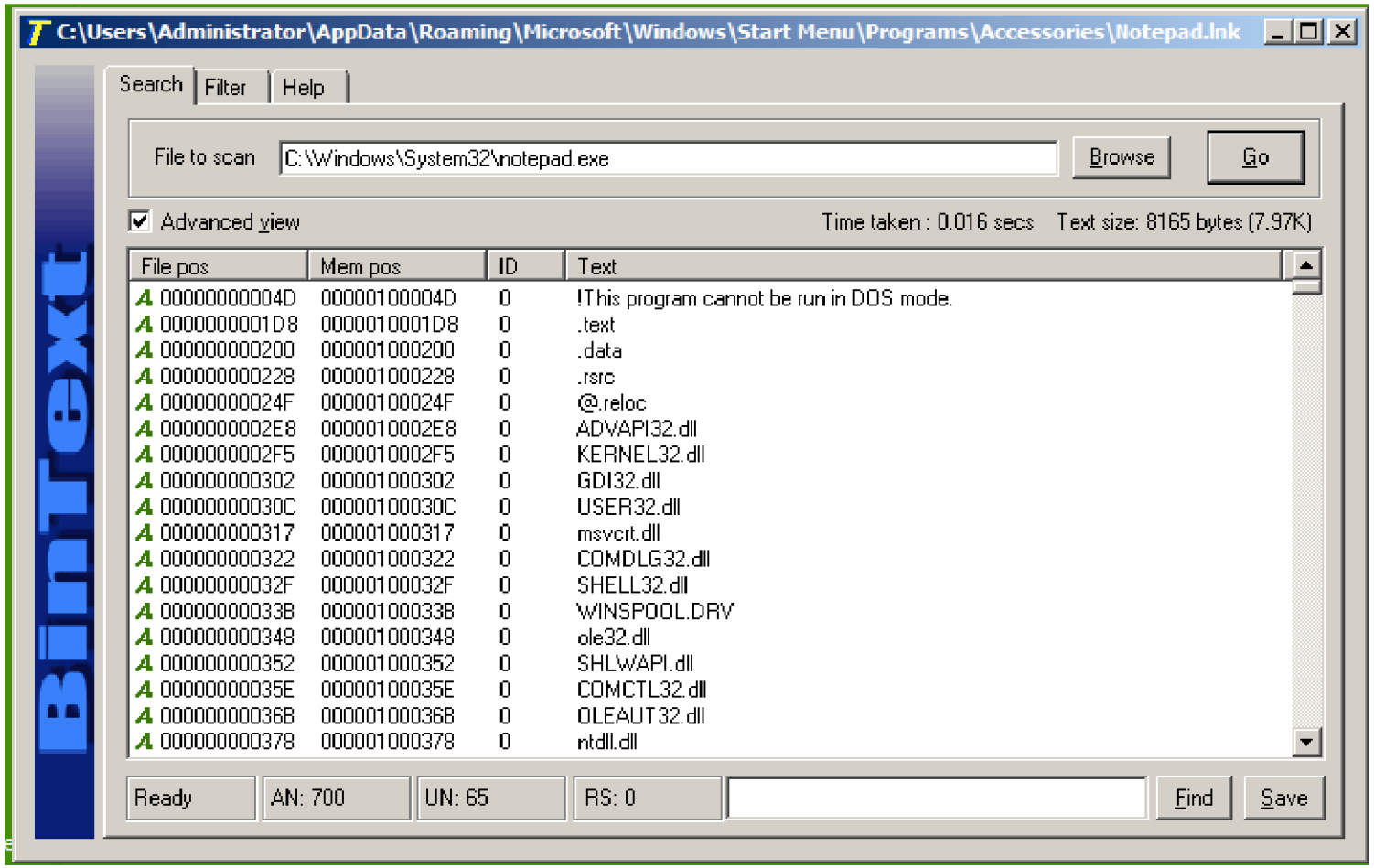
- Sometimes, the strings detected are not actual strings.
  - Example: the byte sequence 0x56, 0x50, 0x33, 0x00.
  - It will be interpreted to VP3.
  - But those bytes may be a mem address, CPU inst., or data.
  - The user needs to filter out invalid strings.

The strings Command: Example

- Bold items can be ignored
- GetLayout and SetLayout are Windows functions
- GDI32.DLL is a Dynamic Link Library

```
C:>strings bp6.ex_
VP3
VW3
t$@
D$4
99.124.22.1 4
e-@
GetLayout 1
GDI32.DLL 3
SetLayout 2
M}C
Mail system DLL is invalid.!Send Mail failed to
send message. 5
```

BinText


C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Accessories\Notepad.lnk
BinText
Search
Filter
Help
File to scan
C:\Windows\System32\notepad.exe
Browse
Go
Advanced view
Time taken : 0.016 secs
Text size: 8165 bytes (7.97K)
File pos
Mem pos
ID
Text
A 00000000004D 00000100004D 0 !This program cannot be run in DOS mode.
A 0000000001D8 0000010001D8 0 .text
A 000000000200 000001000200 0 .data
A 000000000228 000001000228 0 .rsrc
A 00000000024F 00000100024F 0 @.reloc
A 0000000002E8 0000010002E8 0 ADVAPI32.dll
A 0000000002F5 0000010002F5 0 KERNEL32.dll
A 000000000302 000001000302 0 GDI32.dll
A 00000000030C 00000100030C 0 USER32.dll
A 000000000317 000001000317 0 msvcrt.dll
A 000000000322 000001000322 0 COMDLG32.dll
A 00000000032F 00000100032F 0 SHELL32.dll
A 00000000033B 00000100033B 0 WINSPOOL.DRV
A 000000000348 000001000348 0 ole32.dll
A 000000000352 000001000352 0 SHLWAPI.dll
A 00000000035E 00000100035E 0 COMCTL32.dll
A 00000000036B 00000100036B 0 OLEAUT32.dll
A 000000000378 000001000378 0 ntdll.dll
Ready
AN: 700
UN: 65
RS: 0
Find
Save

Packed and Obfuscated Malware

Packed and Obfuscated Programs

- Malware writers often use packing or obfuscation to make their files more difficult to detect or analyze.

- Obfuscated programs:
  - Ones whose execution the malware author has attempted to hide.

- Packed programs:
  - Malicious program is compressed (obfuscated) and cannot be analyzed.

- Both techniques severely limit static malware analysis capabilities

Packed and Obfuscated Programs

- Legitimate programs almost always include many strings.
- Malware that is packed or obfuscated contains very few strings.
- If upon searching for a program with Strings:
  - you find that it has only a few strings
  - it is probably either obfuscated or packed
  - Suggesting that it may be malicious.
  - You'll likely need dynamic analysis to investigate further.

NOTE *Packed and obfuscated code will often include at least the functions* `LoadLibrary` *and* `GetProcAddress`*, which are used to load and gain access to additional functions.*

Packing Files

- The code is compressed, like a Zip file
- This makes the strings and instructions unreadable
- All you'll see is the **wrapper** – small code that unpacks the file when it is run

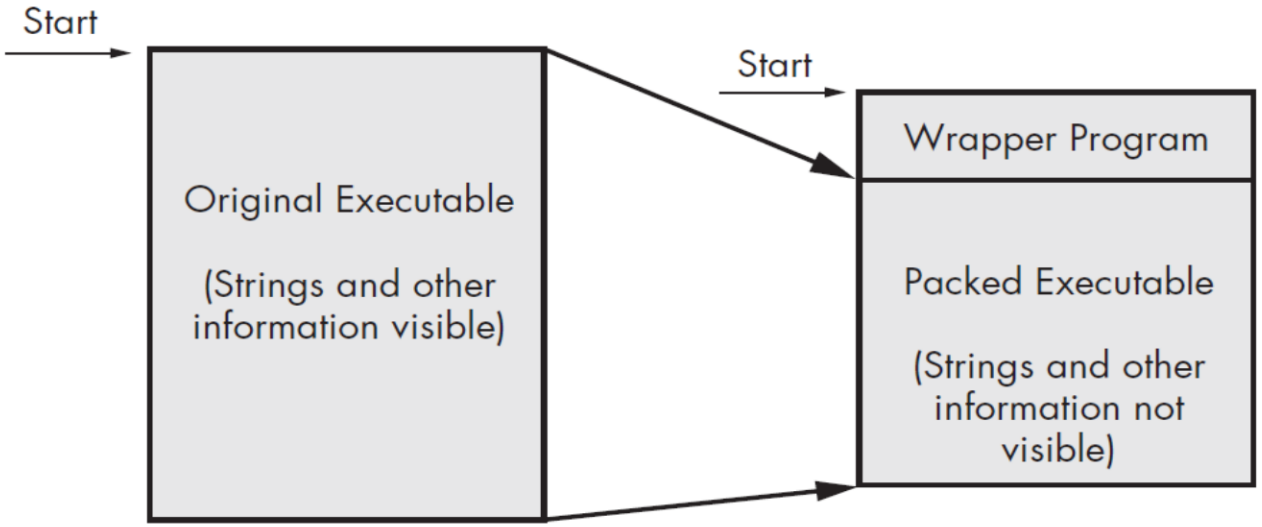


Figure 1-4: The file on the left is the original executable, with all strings, imports, and other information visible. On the right is a packed executable. All of the packed file's strings, imports, and other information are compressed and invisible to most static analysis tools.

Detecting Packers with PEiD

PEiD used to detect the type of packer or compiler employed to build an application, which makes analyzing the packed file much easier

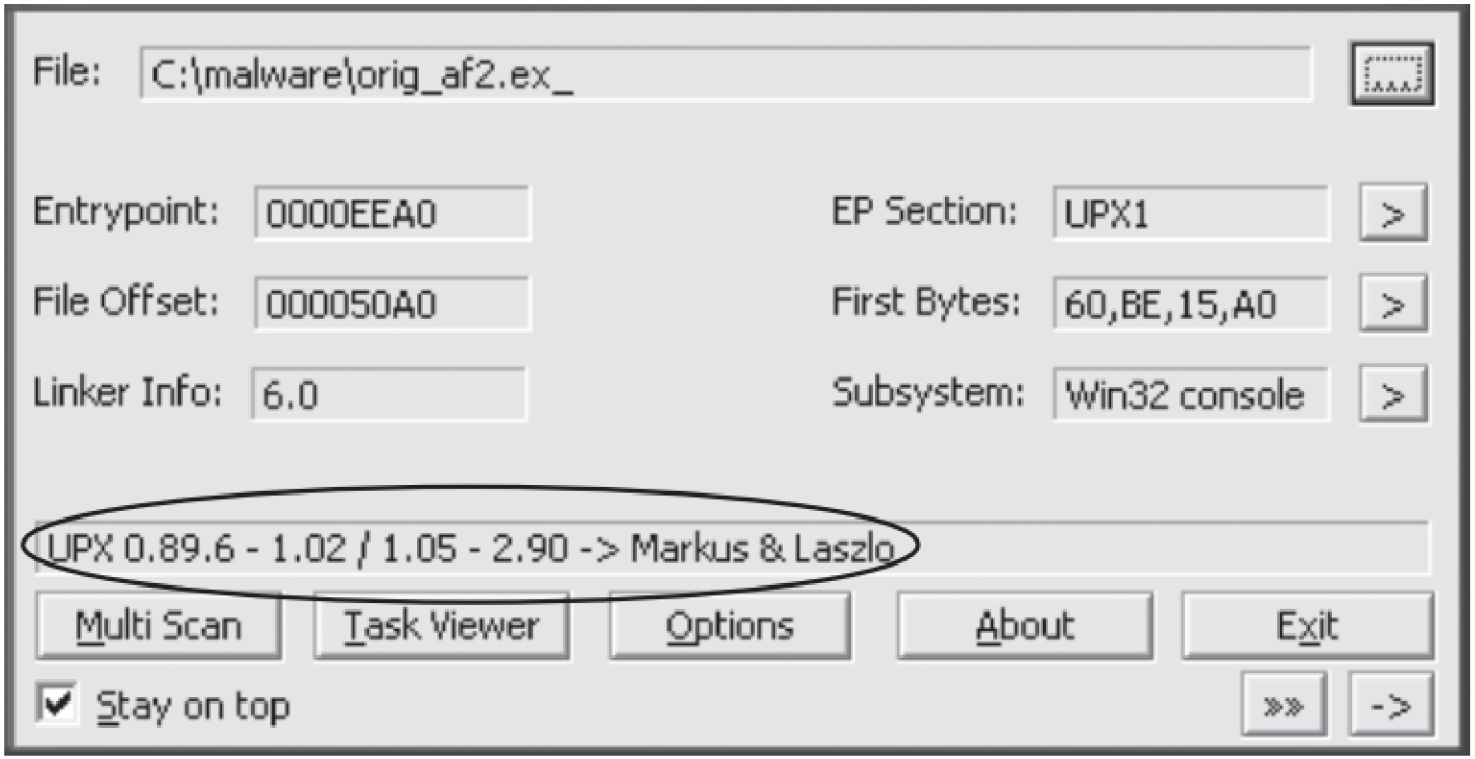


Figure 1-5: The PEiD program

NOTE *Development and support for PEiD has been discontinued since April 2011, but it's still the best tool available for packer and compiler detection. In many cases, it will also identify which packer was used to pack the file.*

UPX packing program

- When a program is packed, you must unpack it in order to be able to perform any analysis.
  - Unpacking process is often complex (covered in Ch. 18).

- UPX packing program is so popular and easy to use for unpacking.
  - For example, to unpack malware packed with UPX, download UPX (http://upx.sourceforge.net/)
  - Run it like so, using the packed program as input:

```
upx -d PackedProgram.exe
```

NOTE:

Many PEiD plug-ins will run the malware executable without warning! (See Next Chapter to learn how to set up a safe environment for running malware.)

Also, like all programs, especially those used for malware analysis, PEiD can be subject to vulnerabilities. For example, PEiD version 0.92 contained a buffer overflow that allowed an attacker to execute arbitrary code.

This would have allowed a clever malware writer to write a program to exploit the malware analyst's machine. Be sure to use the latest version of PEiD.

Portable Executable File Format

Portable Executable File Format

- All previous tools scan executables regardless of their format.
  - But file format can reveal much about the program's functionality.
- PE used by Windows executable files, object code, and DLLs
- PE is a data structure that contains the information necessary for Windows to load the file.
  - Almost every file executed on Windows is in PE format.

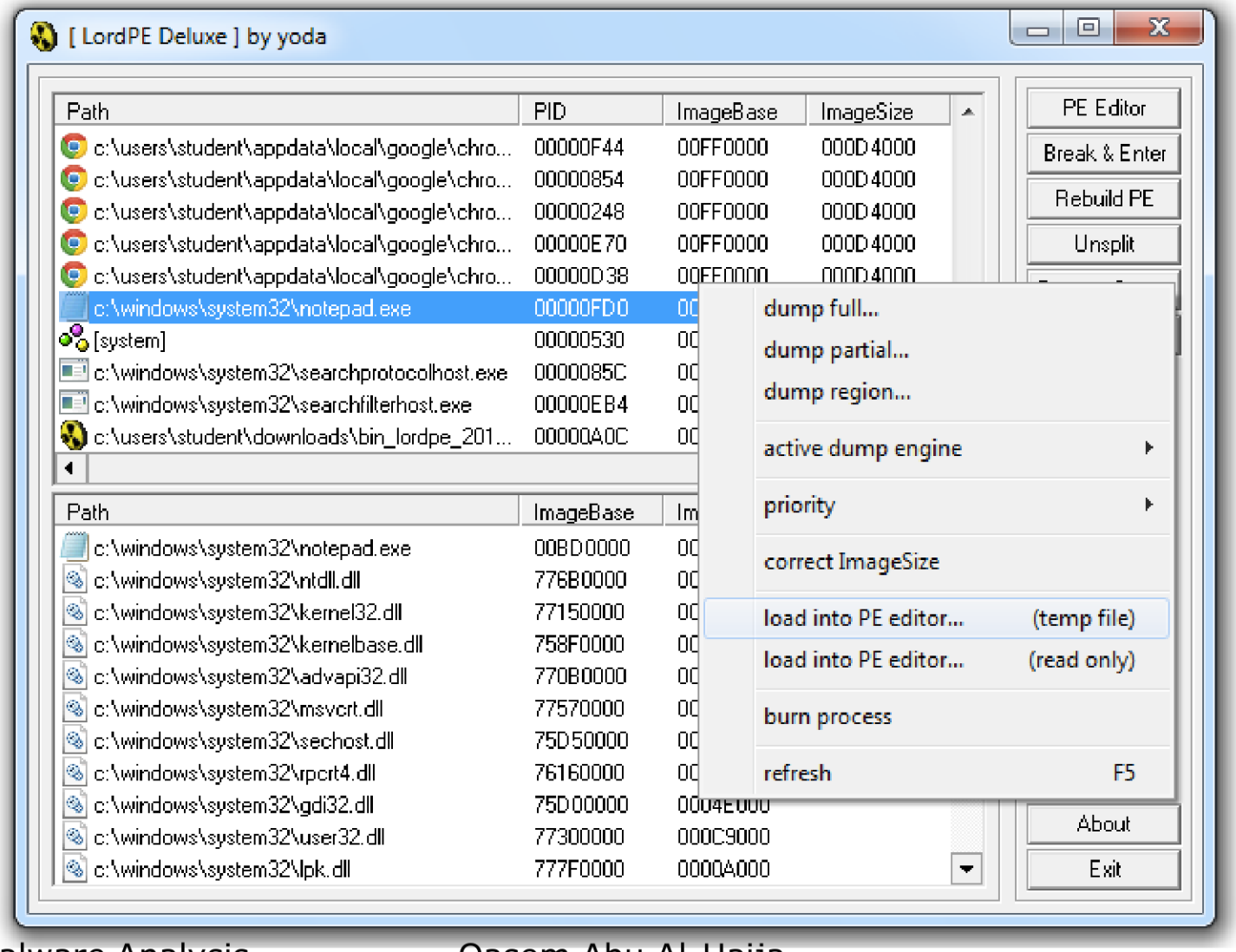
What is PE File Format?

- Used by Win 32-bit /64-bit Oss for executables.
  - DLLs, COM files, .NET executables, .FON Font files, … etc.
- Info. for Win. OS loader to manage wrapped executable code.
  - COFF(Common Object File Format) was used in Win NT before PE.
- The different extensions used to recognize that file format is:
  - .cpl, .dll, .drv, .efi, .exe, .ocx, .scr, .sys.

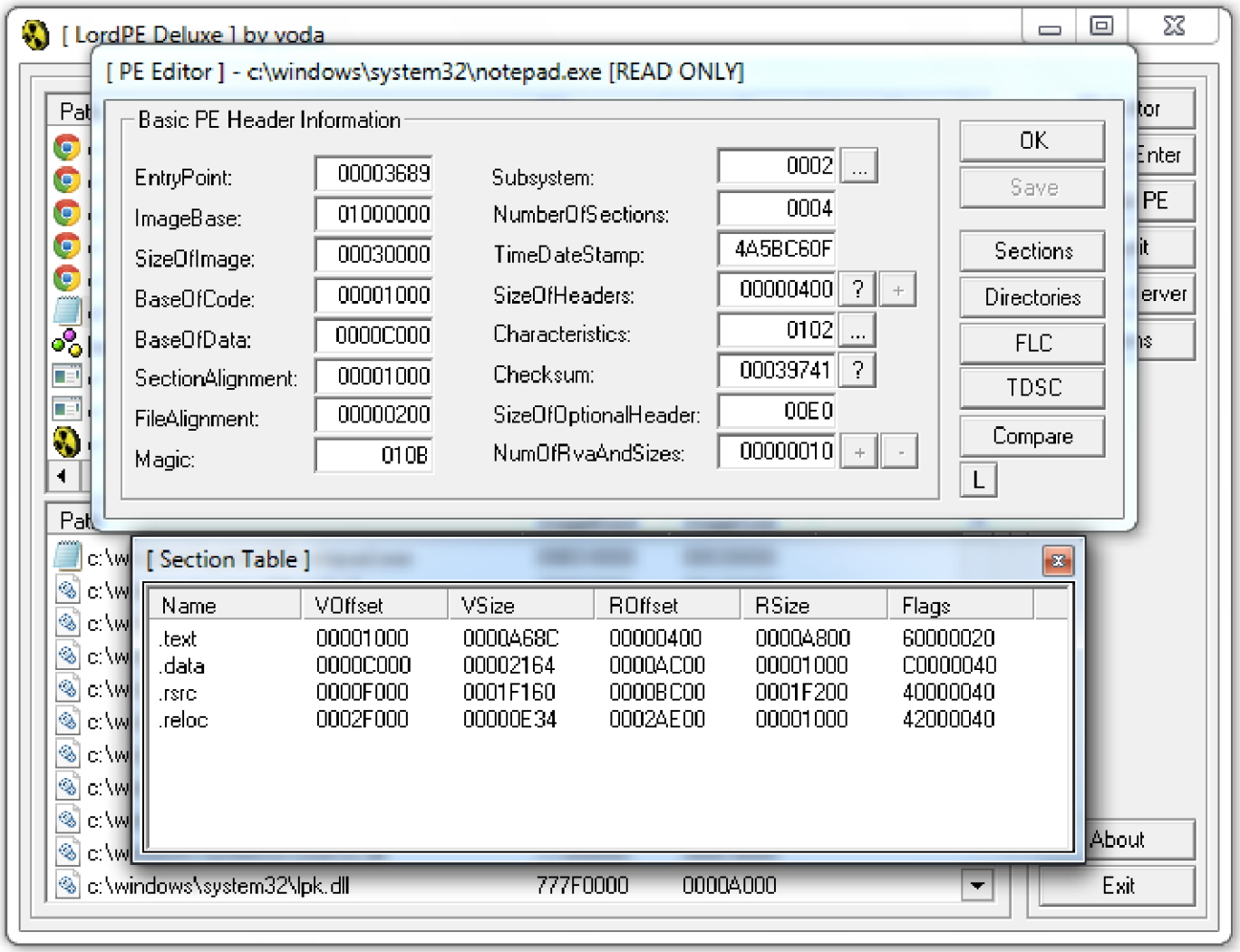
PE Header

- PE files begin with a header that includes:
  - Information about the code
  - Type of application
  - Required library functions
  - Space requirements
- The information in the PE header is of great value to the malware analyst.

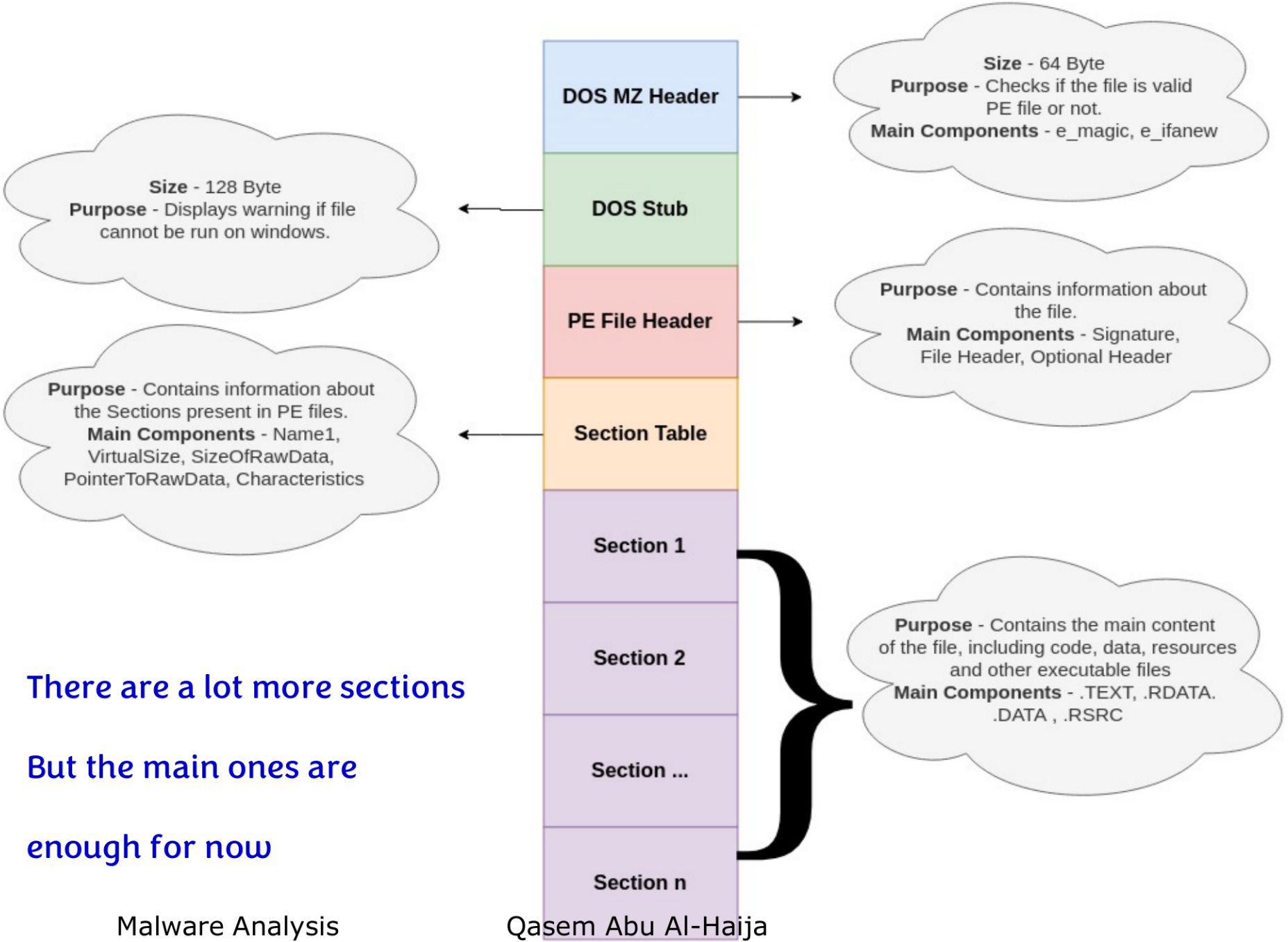
LordPE Demo


[LordPE Deluxe] by yoda
Path
PID
ImageBase
ImageSize
c:\users\student\appdata\local\google\chro...
00000F44
00FF0000
000D4000
c:\users\student\appdata\local\google\chro...
00000854
00FF0000
000D4000
c:\users\student\appdata\local\google\chro...
00000248
00FF0000
000D4000
c:\users\student\appdata\local\google\chro...
00000E70
00FF0000
000D4000
c:\users\student\appdata\local\google\chro...
00000D38
c:\windows\system32\notepad.exe
00000FD0
[system]
00000530
c:\windows\system32\searchprotocolhost.exe
0000085C
c:\windows\system32\searchfilterhost.exe
00000EB4
c:\users\student\downloads\bin_lordpe_201...
00000A0C
Path
ImageBase
c:\windows\system32\notepad.exe
00BD0000
c:\windows\system32\ntdll.dll
776B0000
c:\windows\system32\kernel32.dll
77150000
c:\windows\system32\kernelbase.dll
758F0000
c:\windows\system32\advapi32.dll
770B0000
c:\windows\system32\msvcrt.dll
77570000
c:\windows\system32\sechost.dll
75D50000
c:\windows\system32\rpcrt4.dll
76160000
c:\windows\system32\gdi32.dll
75D00000
c:\windows\system32\user32.dll
77300000
000C9000
c:\windows\system32\lpk.dll
777F0000
0000A000
dump full...
dump partial...
dump region...
active dump engine
priority
correct ImageSize
load into PE editor... (temp file)
load into PE editor... (read only)
burn process
refresh F5
PE Editor
Break & Enter
Rebuild PE
Unsplit
About
Exit

Main Sections



Basic Structure of PE FIle



- There are a lot more sections
- But the main ones are enough for now

PE File format

offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00000000	0x5A4D (MZ)		lastsize		PagesInFile		relocations		headerSizeInParagraph		MinExtraParagraphNeeded		MaxExtraParagraphNeeded		Initial (relative) SS	
0x00000010	Initial (relative) SP		checksum		Initial IP		Initial (relative) CS		FileAddOfRelocTable		OverlayNumber		reserved		reserved	
0x00000020	reserved		reserved		OEMIdentifier		OEMInformation		reserved		reserved		reserved		reserved	
0x00000030	reserved		reserved		reserved		reserved		reserved		reserved		0x80 (offset to PE signature)			
0x00000040	This block contains instructions to display the message "This program cannot be run in DOS mode" when run in MS-DOS															
0x00000050																
0x00000060																
0x00000070																
0x00000080	0x00004550 (PE\0\0 - PE Signature)				Target Machine		NumberOfSections		TimeDateStamp				PointerToSymbolTable (0 for image)			
0x00000090	NumberOfSymbols (0 for image)				SizeOfOptionalHeaders		Characteristics		0x10B (exe)		lnMajVer	lnMnrVer	SizeOfCode			
0x000000A0	SizeOfInitializedData				SizeOfUninitializedData				AddressOfEntryPoint				BaseOfCode			
0x000000B0	BaseOfData				ImageBase				SectionAlignment				FileAlignment			
0x000000C0	MajorOSVersion	MinorOSVersion			MajorImageVersion		MinorImageVersion		MajorSubSystemVersion		MinorSubsystemVersion		Win32VersionValue			
0x000000D0	SizeOfImage				SizeOfHeaders				CheckSum				CheckSum		DllCharacteristics	
0x000000E0	SizeOfStackReserve				SizeOfStackCommit				SizeOfHeapReserve				SizeOfHeapCommit			
0x000000F0	LoaderFlags				NumberOfRVAandSizes				.edata offset				.edata size			
0x00000100	.idata offset				.idata size				.rsrc offset				.rsrc size			
0x00000110	.pdata offset				.pdata size				attribute certificate offset (image)				attribute certificate size (image)			
0x00000120	.reloc offset (image)				.reloc size (image)				.debug offset				.debug size			
0x00000130	Architecture (reserved - 0x0)				Architecture (reserved - 0x0)				Global Ptr offset				must be 0x0			
0x00000140	.tls offset				.tls size				Load config table offset (image)				Load Config table size (image)			
0x00000150	Bound import table offset				Bound import table size				IAT (Import address table) offset				IAT (Import address table) size			
0x00000160	Delay import descriptor offset (image)				Delay import descriptor size (image)				CLR runtime header offset (object)				CLR runtime header size (object)			
0x00000170	Reserved (must be 0x0)				Reserved (must be 0x0)				Section header - Name							
0x00000180	VirtualSize				VirtualAddress				SizeOfRawData				PointerToRawData			
0x00000190	PointerToRelocations				PointerToLineNumbers				NumberOfRelocations		NumberOfLineNumbers		Characteristics			
0x000001A0	Section header - Name								VirtualSize				VirtualAddress			
0x000001B0	SizeOfRawData				PointerToRawData				PointerToRelocations				PointerToLineNumbers			
0x000001C0	NumberOfRelocations		NumberOfLineNumbers		Characteristics				Section header - Name..							

			Size in bytes
	MS-DOS header	File header	64
	PE Signature		4
	COFF header		20
	Standard fields	Optional header	28
	Windows-Specific fields		68
	Data directories		variable
	Section table (each section header is 40 bytes)		variable

Linked Libraries and Functions

Linked Libraries and Functions

- One of the most useful pieces of information that we can gather about an executable is the list of functions it imports.

- IMPORTS:
  - Functions used by one program but stored in a different program.
    - Such as code libraries that contain functionality common to many programs.
  - Connected to the main EXE by **Linking**
  - No need to re-implement certain functionality in multiple programs.
  - Code libraries (Imports) can be linked in three ways
    - **Statically**
    - At Runtime
    - **Dynamically**

Static Linking

- Least common method of linking libraries in Windows.
  - But it is common in UNIX and Linux programs (?)

- When a library is statically linked to an executable:
  - All code from that library is copied into the executable.
  - Which makes the executable grow in size.

- Therefore, when analyzing code:
  - it's difficult to distinguish linked code and original code.
  - Nothing in PE header indicates that the file has linked code.

Runtime Linking

- Commonly used in malware, especially when packed or obfuscated (unpopular in friendly programs).

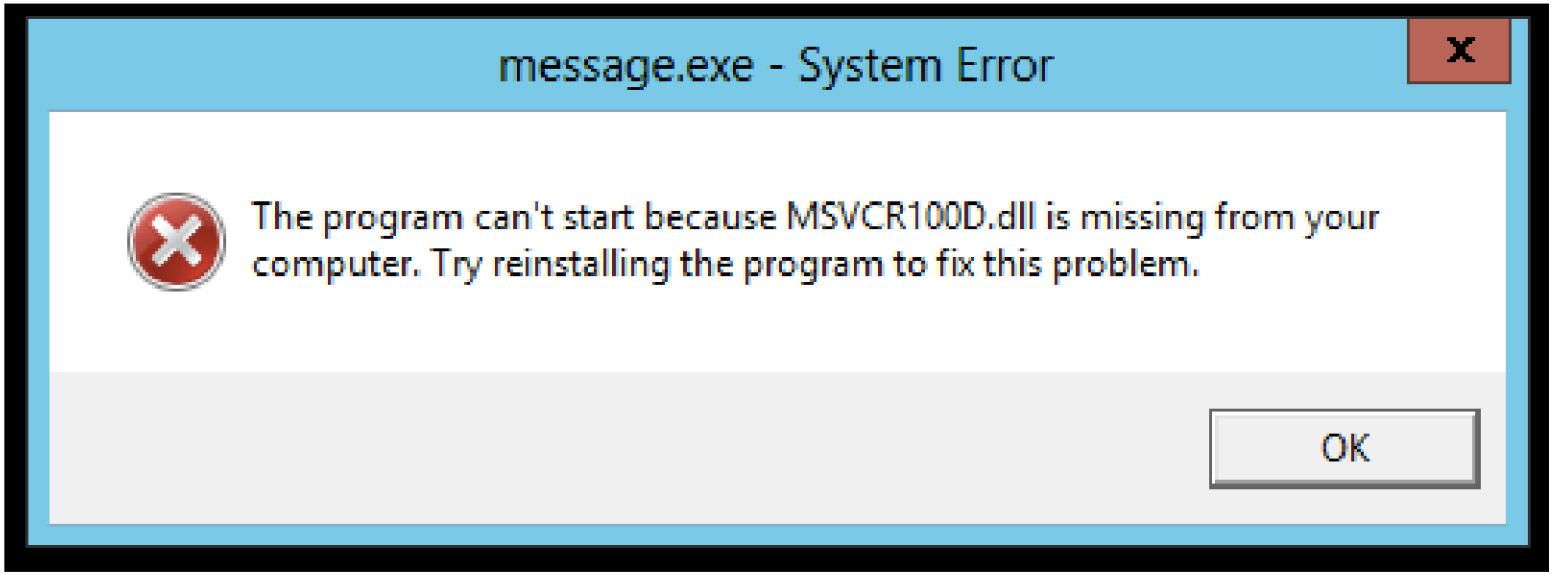
- Executables connect to libraries only when function is needed.
  - not at program start, as with dynamically linked programs.

- Several Windows functions allows importing linked functions.
  - Two most commonly used: LoadLibrary and GetProcAddress.

Dynamic Linking

- The most common and the most interesting for malware analysts.

- Host OS searches for the necessary libraries when the program is loaded.



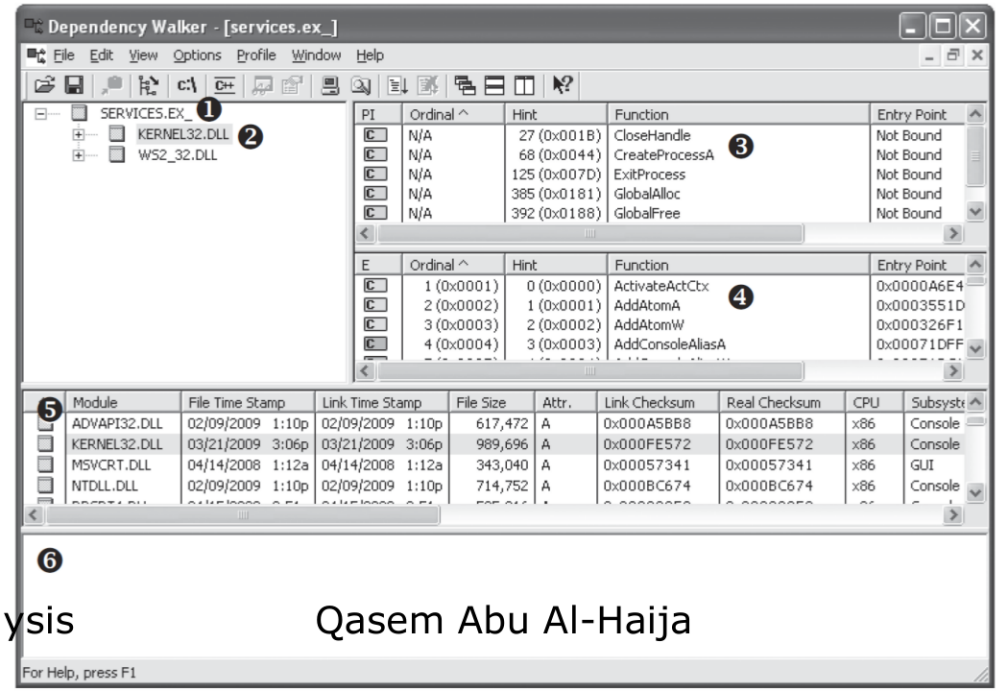
Clues in Libraries and Functions

- PE header lists every library and function that will be loaded.
  - Libraries & functions are very important parts of a program.
  - Identifying them helps guess what the program does.
  - Their names can reveal what the program does
  - **URLDownloadToFile** indicates that the program downloads something

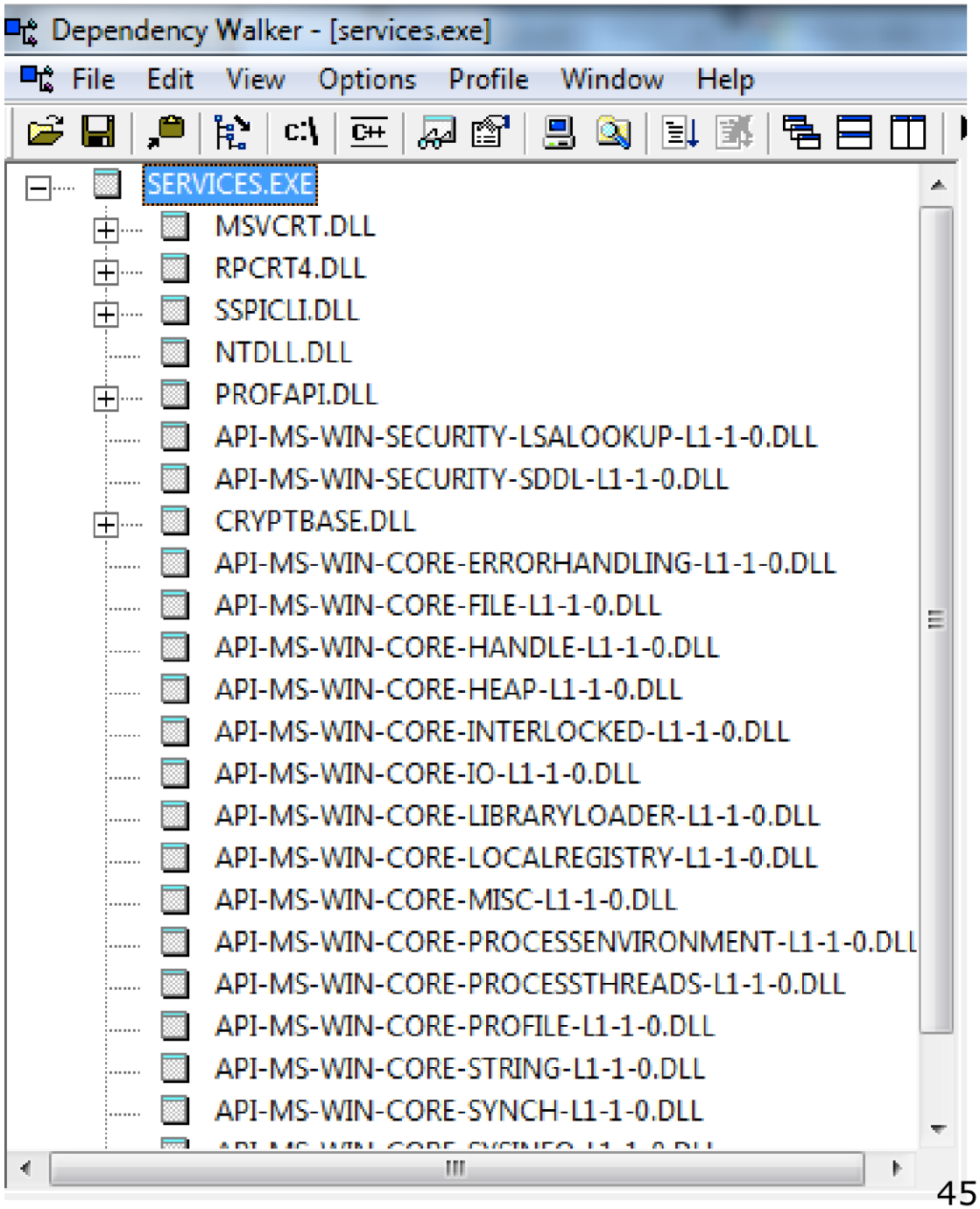
Dependency Walker

Dependency Walker program

- http://www.dependencywalker.com/
- Shows Dynamically Linked Functions in an executable
  - *Normal programs have a lot of DLLs*
  - *Malware often has very few DLLs*

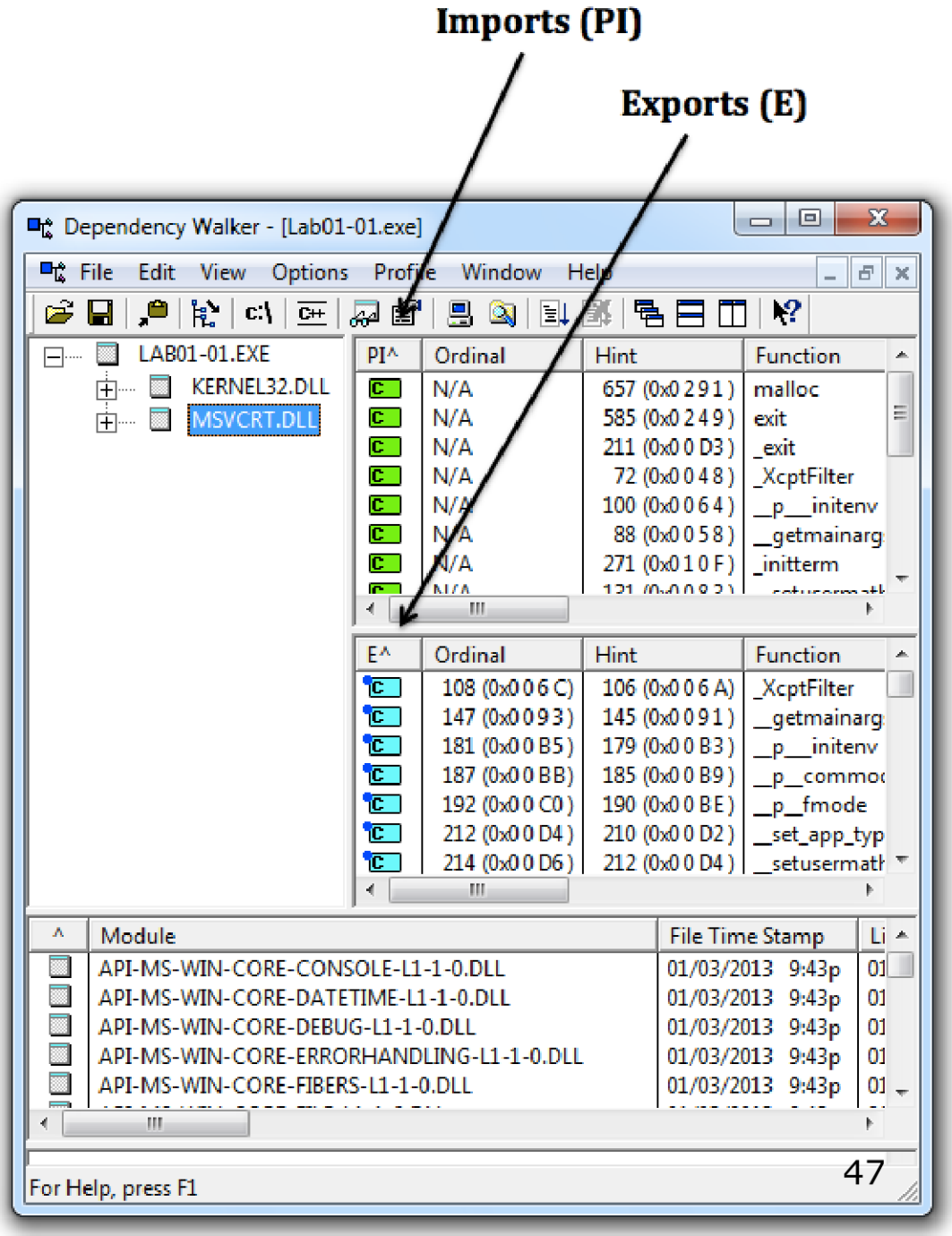


Services.exe



Services.ex_ (malware)

Imports & Exports in Dependency Walker



Malware Analysis
Qasem Abu Al-Haija

Imported Functions

- The PE file header also includes information about specific functions used by an executable.
- The names of these Windows functions can give a good idea about what the executable does.
- Microsoft does an excellent job of documenting the Windows API
  - Microsoft Developer Network (MSDN) library.

Exported Functions

- Like imports, executables export functions to interact with other programs and code.
  - DLLs export functions and EXEs import functions
  - DLLs are implemented to provide functionality used by EXEs.

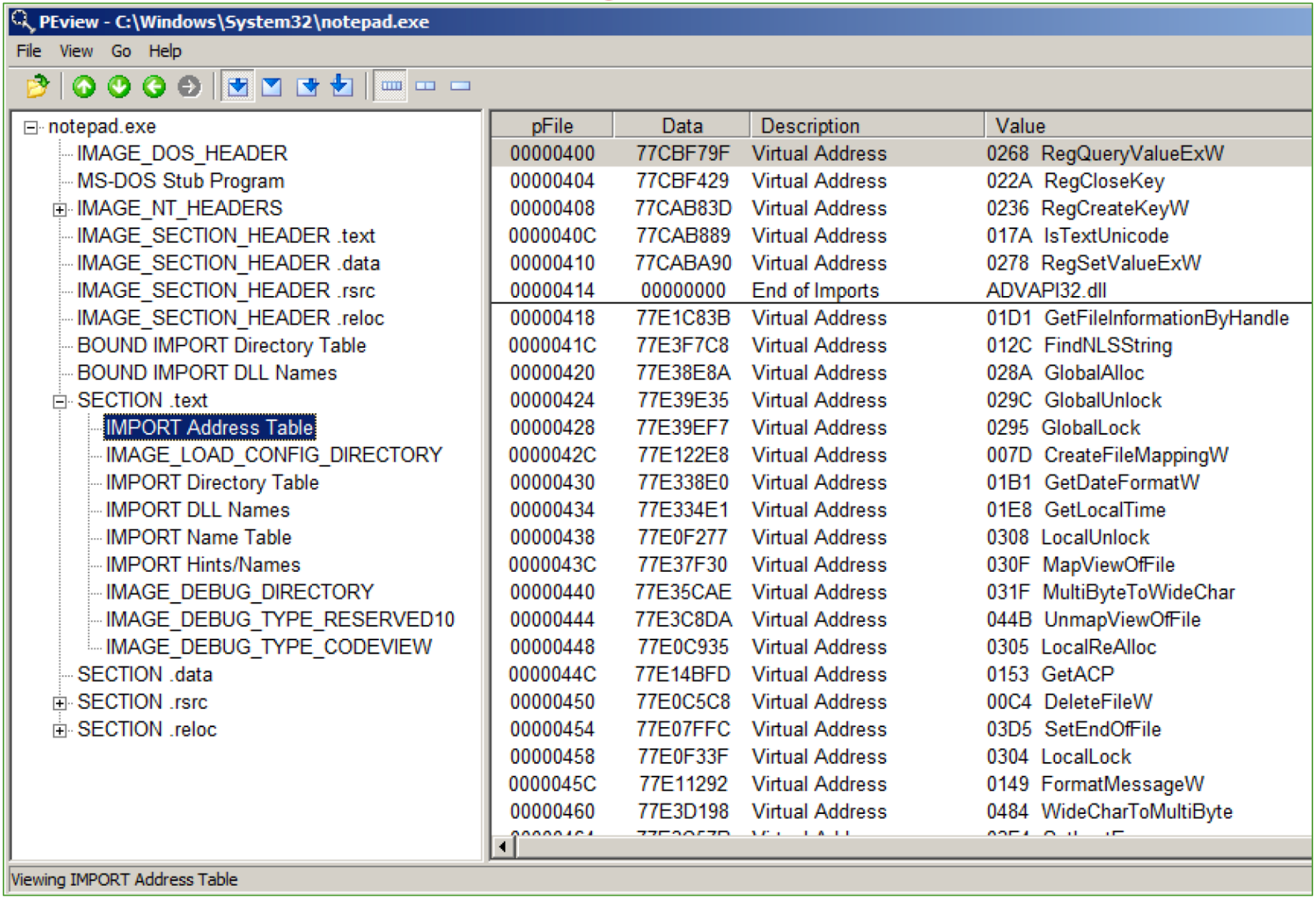
- Both exports and imports are listed in the PE header
  - If you discover exports in an executable, they often will provide useful information.

Common DLLs

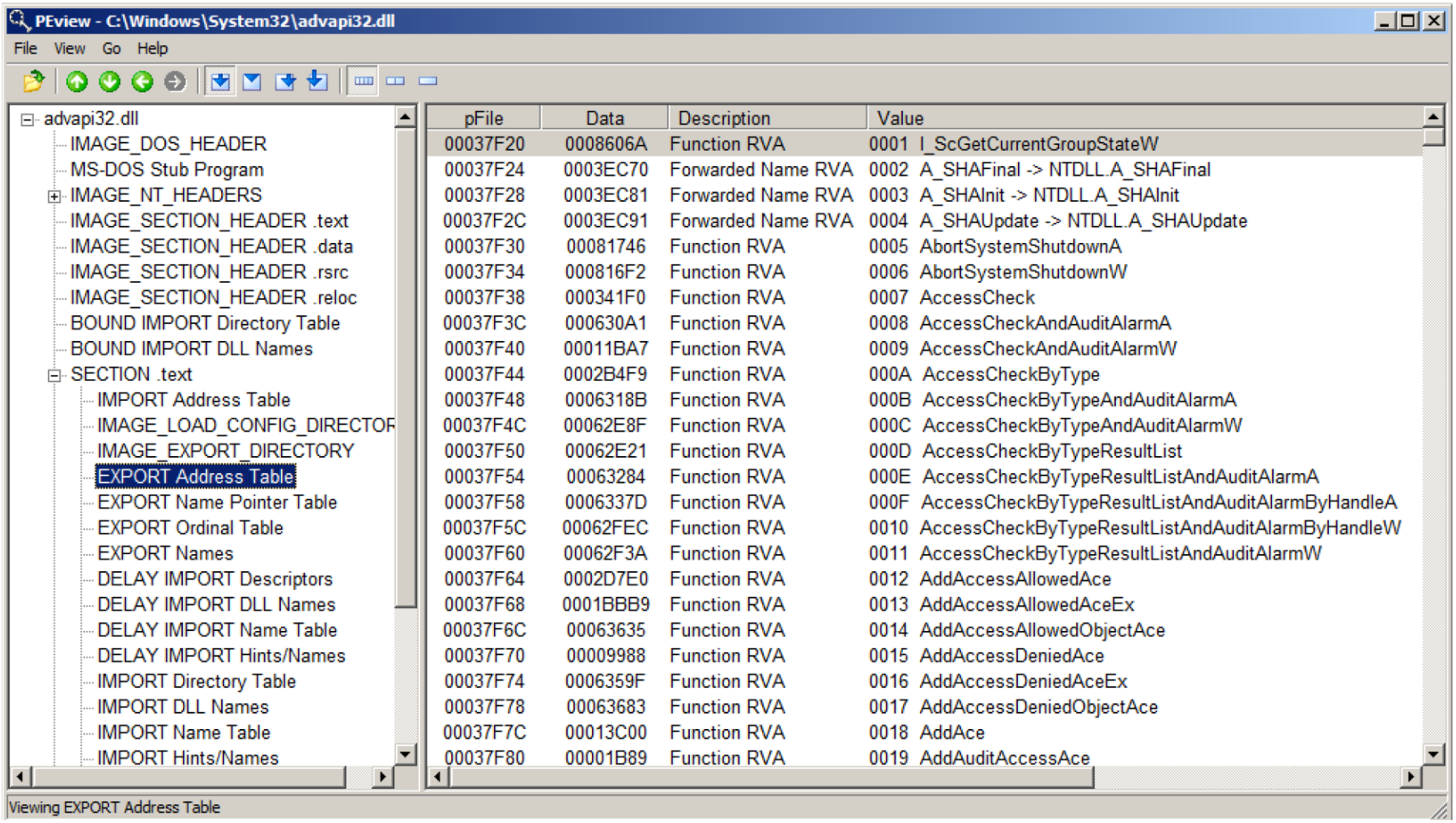
Table 1-1: Common DLLs

DLL	Description
Kernel32.dll	This is a very common DLL that contains core functionality, such as access and manipulation of memory, files, and hardware.
Advapi32.dll	This DLL provides access to advanced core Windows components such as the Service Manager and Registry.
User32.dll	This DLL contains all the user-interface components, such as buttons, scroll bars, and components for controlling and responding to user actions.
Gdi32.dll	This DLL contains functions for displaying and manipulating graphics.
Ntdll.dll	This DLL is the interface to the Windows kernel. Executables generally do not import this file directly, although it is always imported indirectly by *Kernel32.dll*. If an executable imports this file, it means that the author intended to use functionality not normally available to Windows programs. Some tasks, such as hiding functionality or manipulating processes, will use this interface.
WSock32.dll and *Ws2_32.dll*	These are networking DLLs. A program that accesses either of these most likely connects to a network or performs network-related tasks.
Wininet.dll	This DLL contains higher-level networking functions that implement protocols such as FTP, HTTP, and NTP.

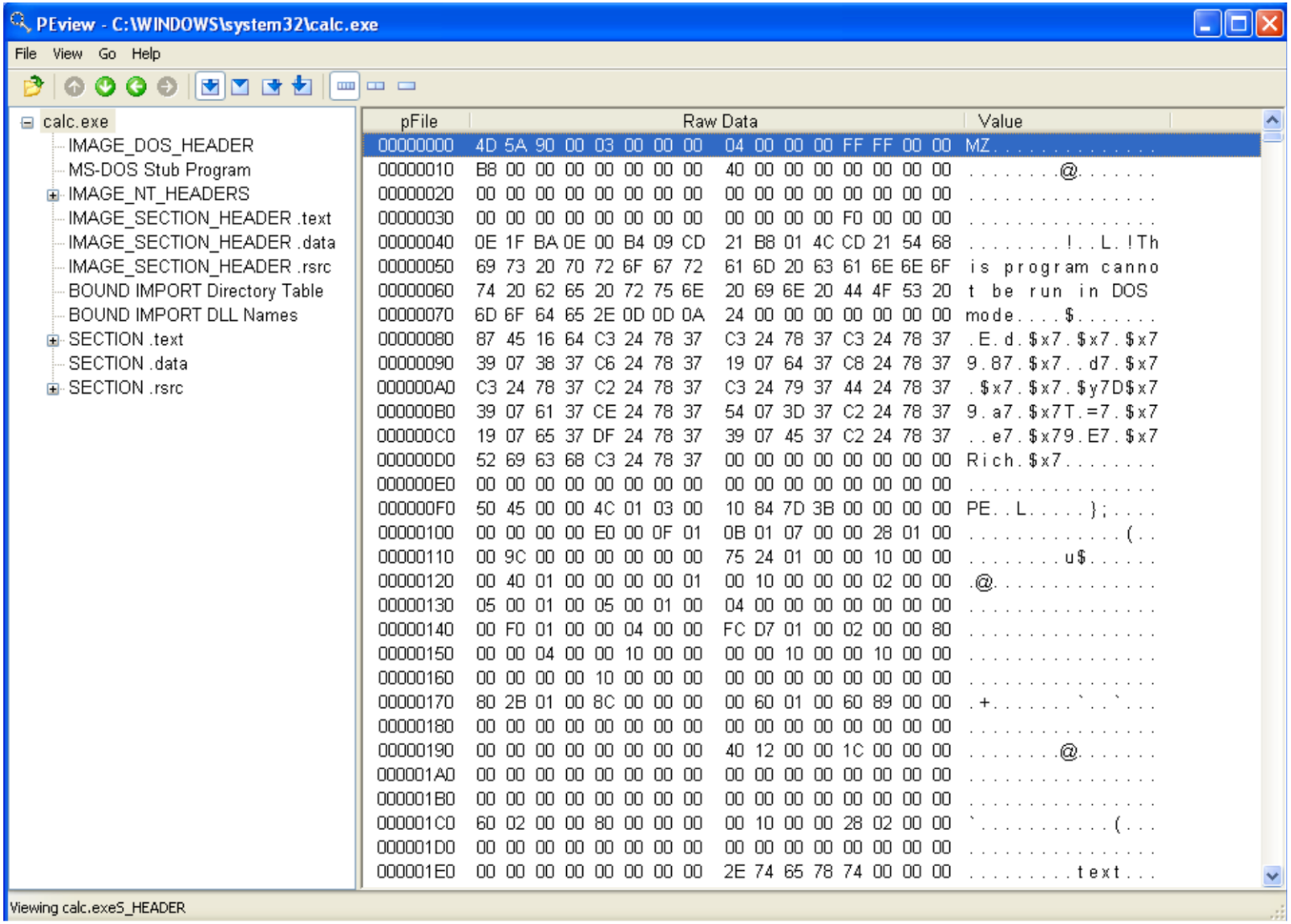
Notepad.exe


PEview - C:\Windows\System32\notepad.exe
File View Go Help
notepad.exe
IMAGE_DOS_HEADER
MS-DOS Stub Program
IMAGE_NT_HEADERS
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .rsrc
IMAGE_SECTION_HEADER .reloc
BOUND IMPORT Directory Table
BOUND IMPORT DLL Names
SECTION .text
IMPORT Address Table
IMAGE_LOAD_CONFIG_DIRECTORY
IMPORT Directory Table
IMPORT DLL Names
IMPORT Name Table
IMPORT Hints/Names
IMAGE_DEBUG_DIRECTORY
IMAGE_DEBUG_TYPE_RESERVED10
IMAGE_DEBUG_TYPE_CODEVIEW
SECTION .data
SECTION .rsrc
SECTION .reloc
pFile Data Description Value
00000400 77CBF79F Virtual Address 0268 RegQueryValueExW
00000404 77CBF429 Virtual Address 022A RegCloseKey
00000408 77CAB83D Virtual Address 0236 RegCreateKeyW
0000040C 77CAB889 Virtual Address 017A IsTextUnicode
00000410 77CABA90 Virtual Address 0278 RegSetValueExW
00000414 00000000 End of Imports ADVAPI32.dll
00000418 77E1C83B Virtual Address 01D1 GetFileInformationByHandle
0000041C 77E3F7C8 Virtual Address 012C FindNLSString
00000420 77E38E8A Virtual Address 028A GlobalAlloc
00000424 77E39E35 Virtual Address 029C GlobalUnlock
00000428 77E39EF7 Virtual Address 0295 GlobalLock
0000042C 77E122E8 Virtual Address 007D CreateFileMappingW
00000430 77E338E0 Virtual Address 01B1 GetDateFormatW
00000434 77E334E1 Virtual Address 01E8 GetLocalTime
00000438 77E0F277 Virtual Address 0308 LocalUnlock
0000043C 77E37F30 Virtual Address 030F MapViewOfFile
00000440 77E35CAE Virtual Address 031F MultiByteToWideChar
00000444 77E3C8DA Virtual Address 044B UnmapViewOfFile
00000448 77E0C935 Virtual Address 0305 LocalReAlloc
0000044C 77E14BFD Virtual Address 0153 GetACP
00000450 77E0C5C8 Virtual Address 00C4 DeleteFileW
00000454 77E07FFC Virtual Address 03D5 SetEndOfFile
00000458 77E0F33F Virtual Address 0304 LocalLock
0000045C 77E11292 Virtual Address 0149 FormatMessageW
00000460 77E3D198 Virtual Address 0484 WideCharToMultiByte
Viewing IMPORT Address Table

Advapi32.dll



Calc.exe


PEview - C:\WINDOWS\system32\calc.exe
File View Go Help
calc.exe
IMAGE_DOS_HEADER
MS-DOS Stub Program
IMAGE_NT_HEADERS
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .rsrc
BOUND IMPORT Directory Table
BOUND IMPORT DLL Names
SECTION .text
SECTION .data
SECTION .rsrc
pFile Raw Data Value
00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ
00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 @
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00
00000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 !..L.!Th
00000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
00000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
00000070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.....
00000080 87 45 16 64 C3 24 78 37 C3 24 78 37 C3 24 78 37 .E.d.$x7.$x7.$x7
00000090 39 07 38 37 C6 24 78 37 19 07 64 37 C8 24 78 37 9.87.$x7..d7.$x7
000000A0 C3 24 78 37 C2 24 78 37 C3 24 79 37 44 24 78 37 .$x7.$x7.$y7D$x7
000000B0 39 07 61 37 CE 24 78 37 54 07 3D 37 C2 24 78 37 9.a7.$x7T.=7.$x7
000000C0 19 07 65 37 DF 24 78 37 39 07 45 37 C2 24 78 37 ..e7.$x79.E7.$x7
000000D0 52 69 63 68 C3 24 78 37 00 00 00 00 00 00 00 00 Rich.$x7
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0 50 45 00 00 4C 01 03 00 10 84 7D 3B 00 00 00 00 PE..L.....};....
00000100 00 00 00 00 E0 00 0F 01 0B 01 07 00 00 28 01 00 .....(..
00000110 00 9C 00 00 00 00 00 00 75 24 01 00 00 10 00 00 .....u$.....
00000120 00 40 01 00 00 00 00 01 00 10 00 00 00 02 00 00 @.....
00000130 05 00 01 00 05 00 01 00 04 00 00 00 00 00 00 00
00000140 00 F0 01 00 00 04 00 00 FC D7 01 00 02 00 00 80
00000150 00 00 04 00 00 10 00 00 00 00 10 00 00 10 00 00
00000160 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
00000170 80 2B 01 00 8C 00 00 00 00 60 01 00 60 89 00 00 .+.....`..`...
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000190 00 00 00 00 00 00 00 00 40 12 00 00 1C 00 00 00 .....@.....
000001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C0 60 02 00 00 80 00 00 00 00 10 00 00 28 02 00 00 `.....(...
000001D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001E0 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00 .....text...
Viewing calc.exe5_HEADER

Example: Keylogger

- Imports User32.dll and uses the function **SetWindowsHookEx**, which is a popular way keyloggers receive keyboard inputs

- It exports **LowLevelKeyboardProc** and **LowLevelMouseProc** to send the data elsewhere

- It uses **RegisterHotKey** to define a special keystroke like Ctrl+Shift+P to harvest the collected data

Ex: A Packed Program

- Very few functions
- All you see is the unpacker

Table 2-3. DLLs and Functions Imported from PackedProgram.exe

Kernel32.dll	User32.dll
GetModuleHandleA	MessageBoxA
LoadLibraryA	
GetProcAddress	
ExitProcess	
VirtualAlloc	
VirtualFree	

The PE File Headers and Sections

PE File Headers

- PE file headers can provide considerably more information than just imports.
- PE file format contains a header followed by a series of sections.
- The header contains metadata about the file itself.
- Following the header are the actual sections of the file, each of which contains useful information.

Important PE Sections (1)

- **.text** -- instructions for the CPU to execute
- **.rdata** -- imports & exports
- **.data** - global data
- **.rsrc** - strings, icons, images, menus

Important PE Sections(2)

.text

- instructions for the CPU to execute
- This is the only section that can execute,
- It should be the only section that includes code.

.rdata

- Import and export information.
- Also store other read-only data used by the program.
- Some file contains: .idata & .edata sections
  - which store the import and export information

Important PE Sections (3)

.data

- Program's global data
  - accessible from anywhere in the program.
- No Local data stored here or in the PE file.

.rsrc

- Resources used by the executable.
  - Such as icons, images, menus, and strings.

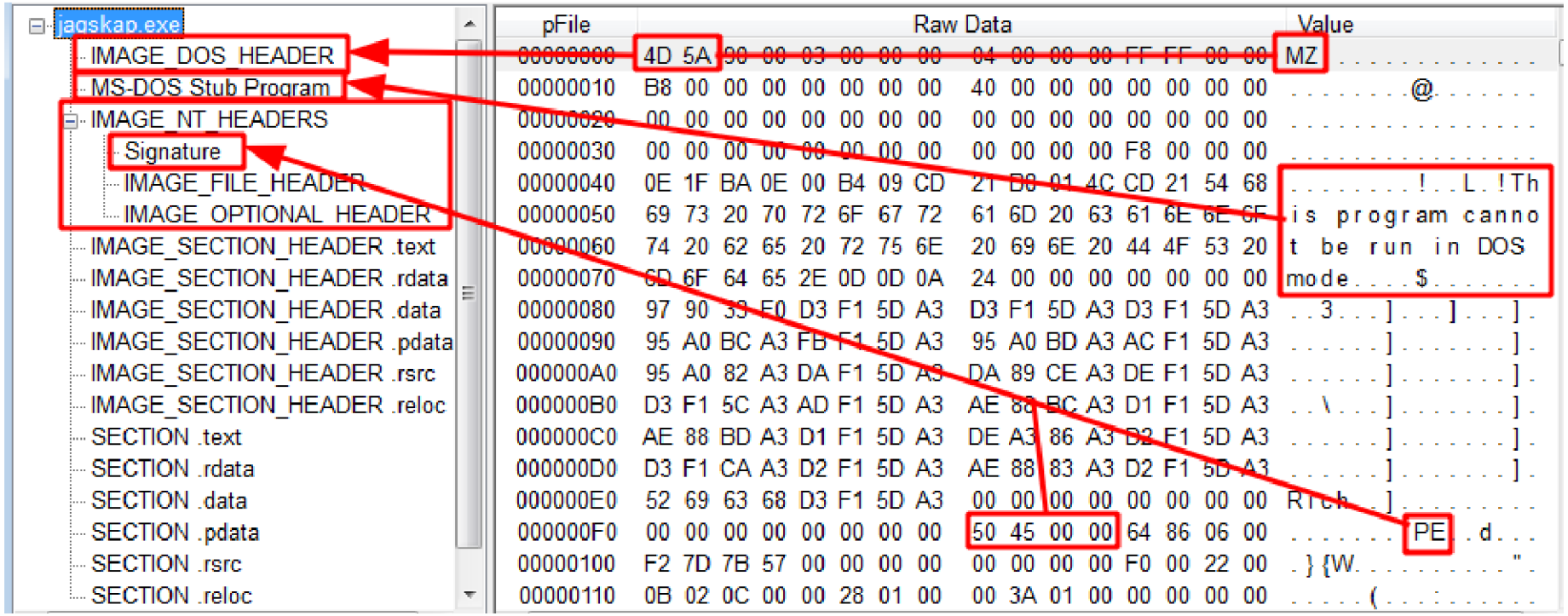
Important File Sections (4)

- Section names are often consistent across a compiler but vary across different compilers.
  - For example, Visual Studio uses .text for executable code, but Borland Delphi uses CODE.
- Windows doesn't care about the actual name.
  - it uses other info. in the PE header to determine how a section is used.
- Furthermore, the section names are sometimes obfuscated to make analysis more difficult.
  - Luckily, the default names are used most of the time.

Examining PE Files with PEview

PE File Components

- The first two parts of the PE header:
  - IMAGE_DOS_HEADER and MS-DOS Stub Program
  - Historical and offer no information of our interest.



PE File Components

- IMAGE_NT_HEADERS shows the NT headers.
  - The signature is always the same and can be ignored.
  - IMAGE_FILE_HEADER contains basic info. about the file.
  - Time Date Stamp shows when this executable was compiled.
  - IMAGE_OPTIONAL_HEADER includes several important info.

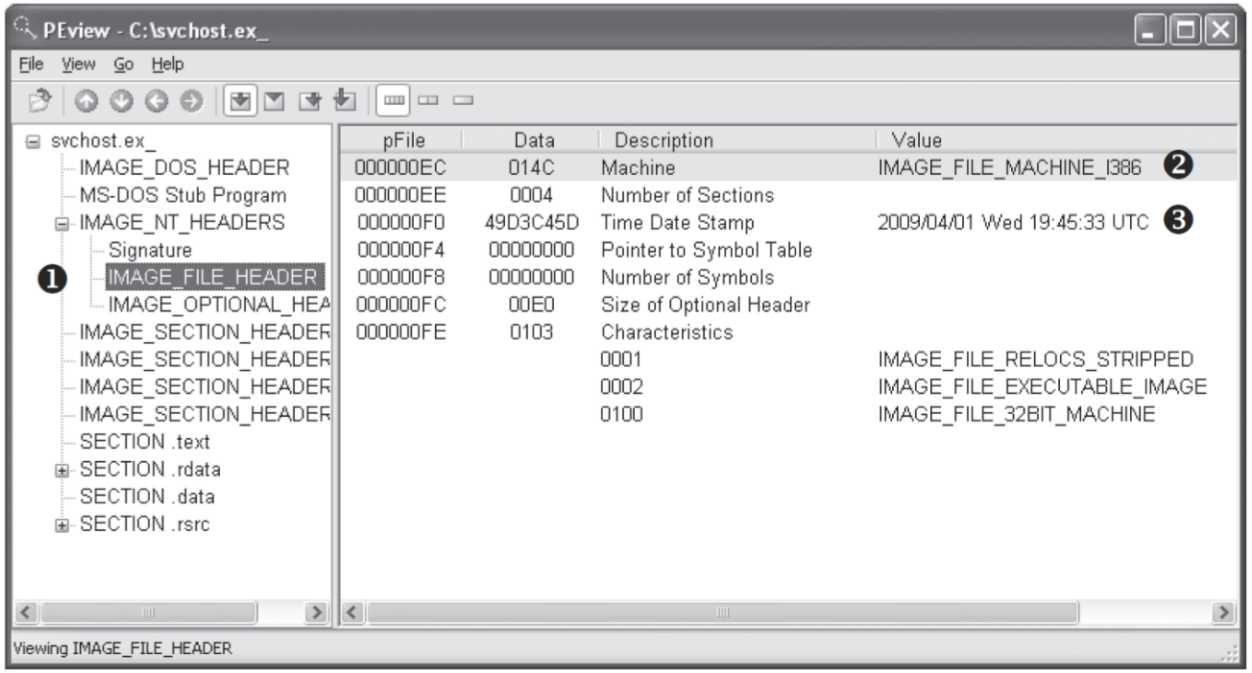


Figure 1-7: Viewing the IMAGE_FILE_HEADER in the PEview program

Time Date Stamp

- Shows when this executable was compiled
- Older programs are more likely to be known to antivirus software
- But sometimes the date is wrong
  - All Delphi programs show June 19, 1992
  - Date can also be faked

IMAGE_OPTIONAL_HEADER

- Includes several important pieces of information.
  - Subsystem indicates whether this is a console or GUI program.
    - Console programs have the value IMAGE_SUBSYSTEM_WINDOWS_CUI and run inside a command window.
    - GUI programs have the value IMAGE_ SUBSYSTEM_WINDOWS_GUI and run within the Windows system.

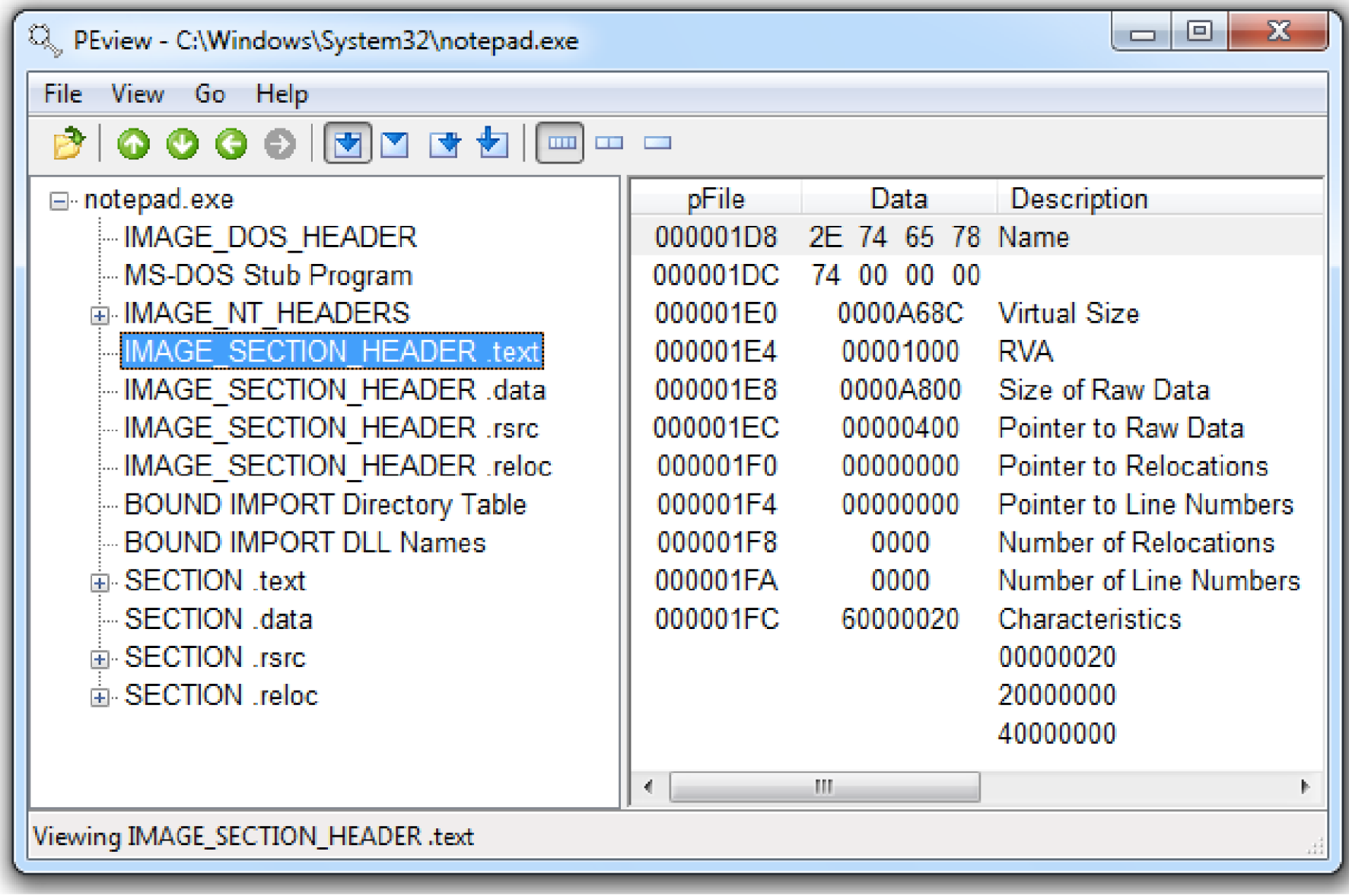
IMAGE_SECTION_HEADER(1)

- Most interesting info. about PE section headers.
  - The compiler generally creates and names PE sections, and the user has little control over these names.
  - Thus, the sections are usually consistent from executable to executable, and any deviations may be **suspicious**.
  - Most important information to look at:
    - Virtual Size – RAM
    - Size of Raw Data – DISK

IMAGE_SECTION_HEADER(2)

- Virtual Size
  - Space is allocated for a section during the loading process
- Size of Raw Data
  - How big the section is on disk.
- For the **.text** section, normally equal or nearly equal.
- Packed executables show a Virtual Size much larger than the Size of Raw Data for .text section

Not Packed


PEview - C:\Windows\System32\notepad.exe
File View Go Help
notepad.exe
IMAGE_DOS_HEADER
MS-DOS Stub Program
IMAGE_NT_HEADERS
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .rsrc
IMAGE_SECTION_HEADER .reloc
BOUND IMPORT Directory Table
BOUND IMPORT DLL Names
SECTION .text
SECTION .data
SECTION .rsrc
SECTION .reloc
pFile Data Description
000001D8 2E 74 65 78 Name
000001DC 74 00 00 00
000001E0 0000A68C Virtual Size
000001E4 00001000 RVA
000001E8 0000A800 Size of Raw Data
000001EC 00000400 Pointer to Raw Data
000001F0 00000000 Pointer to Relocations
000001F4 00000000 Pointer to Line Numbers
000001F8 0000 Number of Relocations
000001FA 0000 Number of Line Numbers
000001FC 60000020 Characteristics
00000020
20000000
40000000
Viewing IMAGE_SECTION_HEADER .text

Packed

Table 2-6. Section Information for PackedProgram.exe

Name	Virtual size	Size of raw data
.text	A000	0000
.data	3000	0000
.rdata	4000	0000
.rsrc	19000	3400
Dijfpds	20000	0000
.sdfuok	34000	3313F
Kijijl	1000	0200

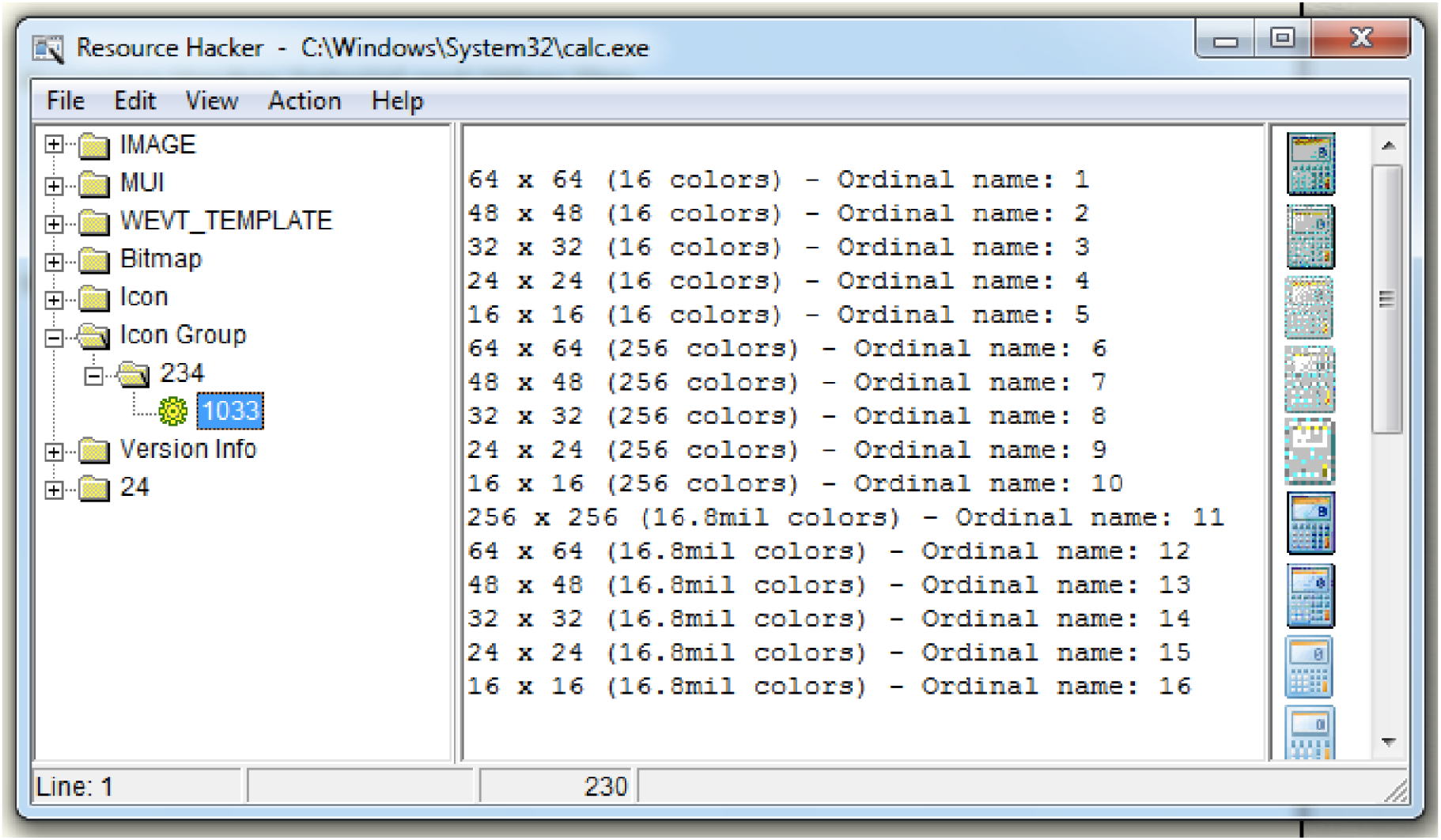
Viewing the Resource Section with Resource Hacker

Resource Hacker

- Lets you browse the **.rsrc** section
- Strings, icons, and menus
- Resource Hacker: http://www.angusj.com/

NOTE *Malware, and occasionally legitimate software, often store an embedded program or driver here and, before the program runs, they extract the embedded executable or driver. Resource Hacker lets you extract these files for individual analysis.*

Resource Hacker



Resource Section with Resource Hacker

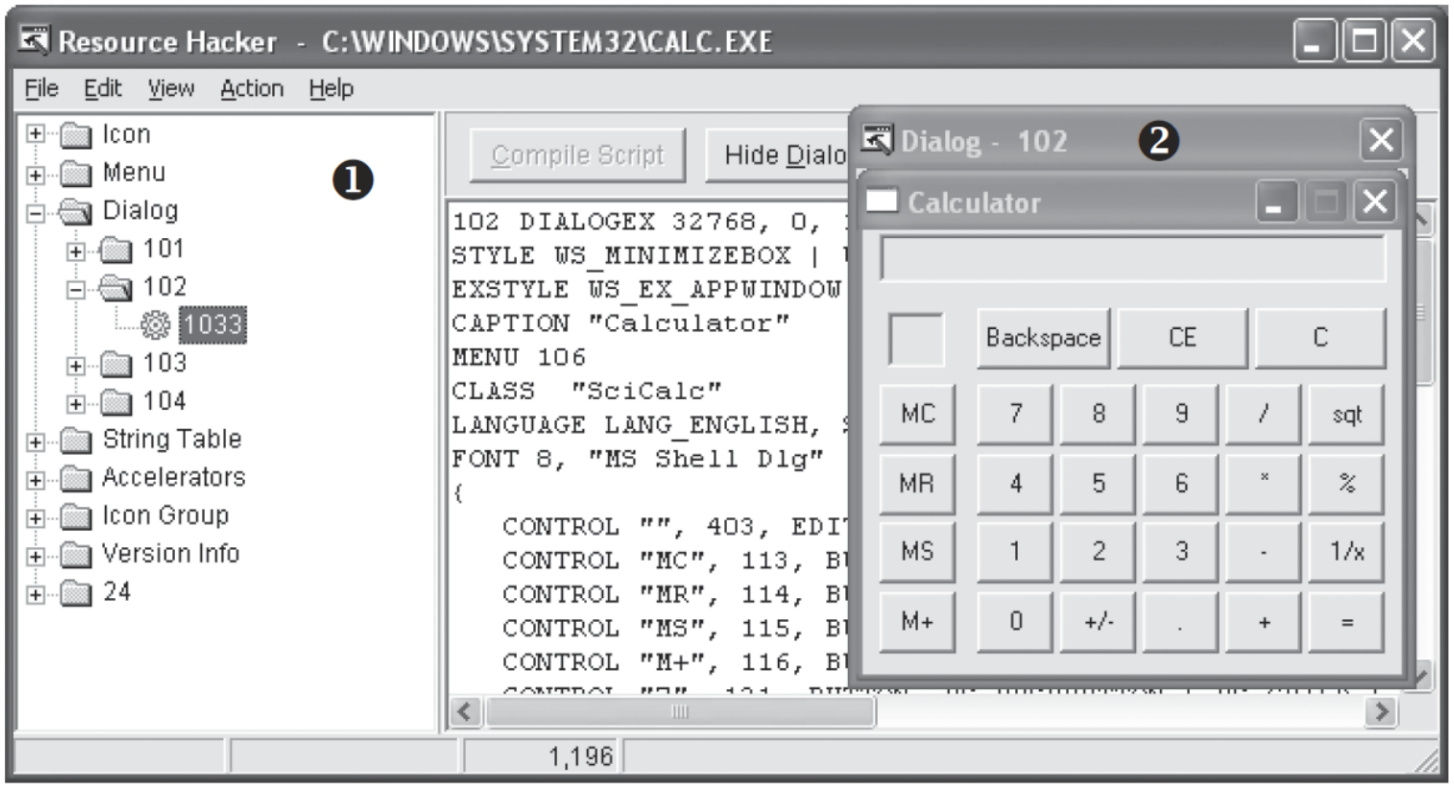


Figure 1-9: The Resource Hacker tool display for calc.exe

PE Header Summary

PE Header Summary

- PE header contains useful info for MA, and we will continue to examine it in subsequent chapters.

- Table 1-7 reviews the key information that can be obtained from a PE header

Table 1-7: Information in the PE Header

Field	Information revealed
Imports	Functions from other libraries that are used by the malware
Exports	Functions in the malware that are meant to be called by other programs or libraries
Time Date Stamp	Time when the program was compiled
Sections	Names of sections in the file and their sizes on disk and in memory
Subsystem	Indicates whether the program is a command-line or GUI application
Resources	Strings, icons, menus, and other information included in the file

Main Sources for these slides

- *Michael Sikorski and Andrew Honig, "Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software"; ISBN-10: 1593272901.*
- *Xinwen Fu, "Introduction to Malware Analysis," University of Central Florida*
- *Sam Bowne, "Practical Malware Analysis," City College San Francisco*
- *Abhijit Mohanta and Anoop Saldanha, "Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware," ISBN: 1484261925.*

Thank you