Practical Malware Analysis

Ch 1: Malware Analysis Primer

Updated 1-15-16

The Goals of Malware Analysis

Incident Response

- Case history
  - A medical clinic with 10 offices found malware on one of their workstations
  - Hired a consultant to clean & re-image that machine
- All done—case closed?

Incident Response

- After malware is found, you need to know
  - Did an attacker implant a rootkit or trojan on your systems?
  - Is the attacker really gone?
  - What did the attacker steal or add?
  - How did the attack get in
    - Root-cause analysis

Breach clean-up cost LinkedIn nearly $1 million, another $2-3 million in upgrades

Summary: *LinkedIn executives reveal on quarterly earnings call just what the June theft of 6.5 million passwords cost the company in forensic work and on-going security updates.*

By John Fontana for Identity Matters | August 3, 2012 -- 17:10 GMT (10:10 PDT)

Follow @johnfontana

LinkedIn spent nearly $1 million investigating and unraveling the theft of 6.5 million passwords in June and plans to spend up to $3 million more updating security on its social networking site.

- Link Ch 1a

Malware Analysis

- Dissecting malware to understand
  - How it works
  - How to identify it
  - How to defeat or eliminate it
- A critical part of incident response

The Goals of Malware Analysis

- Information required to respond to a network intrusion
  - Exactly what happened
  - Ensure you've located all infected machines and files
  - How to measure and contain the damage
  - Find signatures for intrusion detection systems

Signatures

- Host-based signatures
  - Identify files or registry keys on a victim computer that indicate an infection
  - Focus on what the malware did to the system, not the malware itself
    - Different from antivirus signature
- Network signatures
  - Detect malware by analyzing network traffic
  - More effective when made using malware analysis

False Positives

Malware Analysis Techniques

Static v. Dynamic Analysis

- Static Analysis
  - Examines malware without running it
  - Tools: VirusTotal, strings, a disassembler like IDA Pro
- Dynamic Analysis
  - Run the malware and monitor its effect
  - Use a virtual machine and take snapshots
  - Tools: RegShot, Process Monitor, Process Hacker, CaptureBAT
  - RAM Analysis: Mandant Redline and Volatility

Basic Analysis

- Basic static analysis
  - View malware without looking at instructions
  - Tools: VirusTotal, strings
  - Quick and easy but fails for advanced malware and can miss important behavior
- Basic dynamic analysis
  - Easy but requires a safe test environment
  - Not effective on all malware

Advanced Analysis

- Advanced static analysis
  - Reverse-engineering with a disassembler
  - Complex, requires understanding of assembly code
- Advanced Dynamic Analysis
  - Run code in a debugger
  - Examines internal state of a running malicious executable

```
#include <stdlib.h>
int sub(int x, int y){
        return 2*x+y;
}

int main(int argc, char ** argv){
        int a;
        a = atoi(argv[1]);
        return sub(argc,a);
}
```

```
.text:00000000 _sub:      push   ebp
.text:00000001            mov    ebp, esp
.text:00000003            mov    eax, [ebp+8]
.text:00000006            mov    ecx, [ebp+0Ch]
.text:00000009            lea    eax, [ecx+eax*2]
.text:0000000C            pop    ebp
.text:0000000D            retn
.text:00000010 _main:     push   ebp
.text:00000011            mov    ebp, esp
.text:00000013            push   ecx
.text:00000014            mov    eax, [ebp+0Ch]
.text:00000017            mov    ecx, [eax+4]
.text:0000001A            push   ecx
.text:0000001B            call   dword ptr ds:__imp__atoi
.text:00000021            add    esp, 4
.text:00000024            mov    [ebp-4], eax
.text:00000027            mov    edx, [ebp-4]
.text:0000002A            push   edx
.text:0000002B            mov    eax, [ebp+8]
.text:0000002E            push   eax
.text:0000002F            call   _sub
.text:00000034            add    esp, 8
.text:00000037            mov    esp, ebp
.text:00000039            pop    ebp
.text:0000003A            retn
```

```
01 .MODEL SMALL
02 .STACK 100H
03 .CODE
04
05 MOV AX, 0x3C
06 MOV BX, 0000000000001010B
07 ADD AX, BX
08 MOV BX, 14
09 SUB AX, BX
10
11 MOV AH, 04CH
12 INT 21H
```

Assembly vs. machine code

Machine code bytes

```
B8 22 11 00 FF
01 CA
31 F6
53
8B 5C 24 04
8D 34 48
39 C3
72 EB
C3
```

Assembly language statements

```
foo:
movl $0xFF001122, %eax
addl %ecx, %edx
xorl %esi, %esi
pushl %ebx
movl 4(%esp), %ebx
leal (%eax,%ecx,2), %esi
cmpl %eax, %ebx
jnae foo
retl
```

Instruction stream

```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

Types of Malware

Types of Malware

- Backdoor
  - Allows attacker to control the system
- Botnet
  - All infected computers receive instructions from the same Command-and-Control (C&C) server
- Downloader
  - Malicious code that exists only to download other malicious code
  - Used when attacker first gains access

Types of Malware

- Information-stealing malware
  - Sniffers, keyloggers, password hash grabbers
- Launcher
  - Malicious program used to launch other malicious programs
  - Often uses nontraditional techniques to ensure stealth or greater access to a system
- Rootkit
  - Malware that conceals the existence of other code
  - Usually paired with a backdoor

Types of Malware

- Scareware
  - Frightens user into buying something
  - Link Ch 1b

Fake FBI warning tricks man into surrendering himself for possession of child porn

29 Jul, 2013 | by Nishtha Kanal | Like 3 +1 0 Tweet 3 Share

Here's a weird one. We've heard of viruses and malware bringing harm to computers but in a rare instance, a "ransomware" has brought a positive outcome. A man in the US turned himself in to the police after a pop-up caused by a ransomware informed him that child porn had been identified on his machine.

Jay Matthew Riley, a 21-year-old from Virginia was browsing the Internet, when a pop-up containing an "FBI warning" informed him that it had detected child pornography on his machine. The message went on to tell Riley to pay up a fine online or face the consequences.

Types of Malware

- Spam-sending malware
  - Attacker rents machine to spammers
- Worms or viruses
  - Malicious code that can copy itself and infect additional computers

Mass v. Targeted Malware

- Mass malware
  - Intended to infect as many machines as possible
  - Most common type
- Targeted malware
  - Tailored to a specific target
  - Very difficult to detect, prevent, and remove
  - Requires advanced analysis
  - Ex: Stuxnet

General Rules for Malware Analysis

General Rules for Malware Analysis

- Don’t Get Caught in Details
  - You don’t need to understand 100% of the code
  - Focus on key features
- Try Several Tools
  - If one tool fails, try another
  - Don’t get stuck on a hard issue, move along
- Malware authors are constantly raising the bar

Ch 2: Basic Static Analysis

Techniques

- Antivirus scanning
- Hashes
- A file’s strings, functions, and headers

Antivirus Scanning

Only a First Step

- Malware can easily change its signature and fool the antivirus
- VirusTotal is convenient, but using it may alert attackers that they've been caught
  - Link Ch 2a

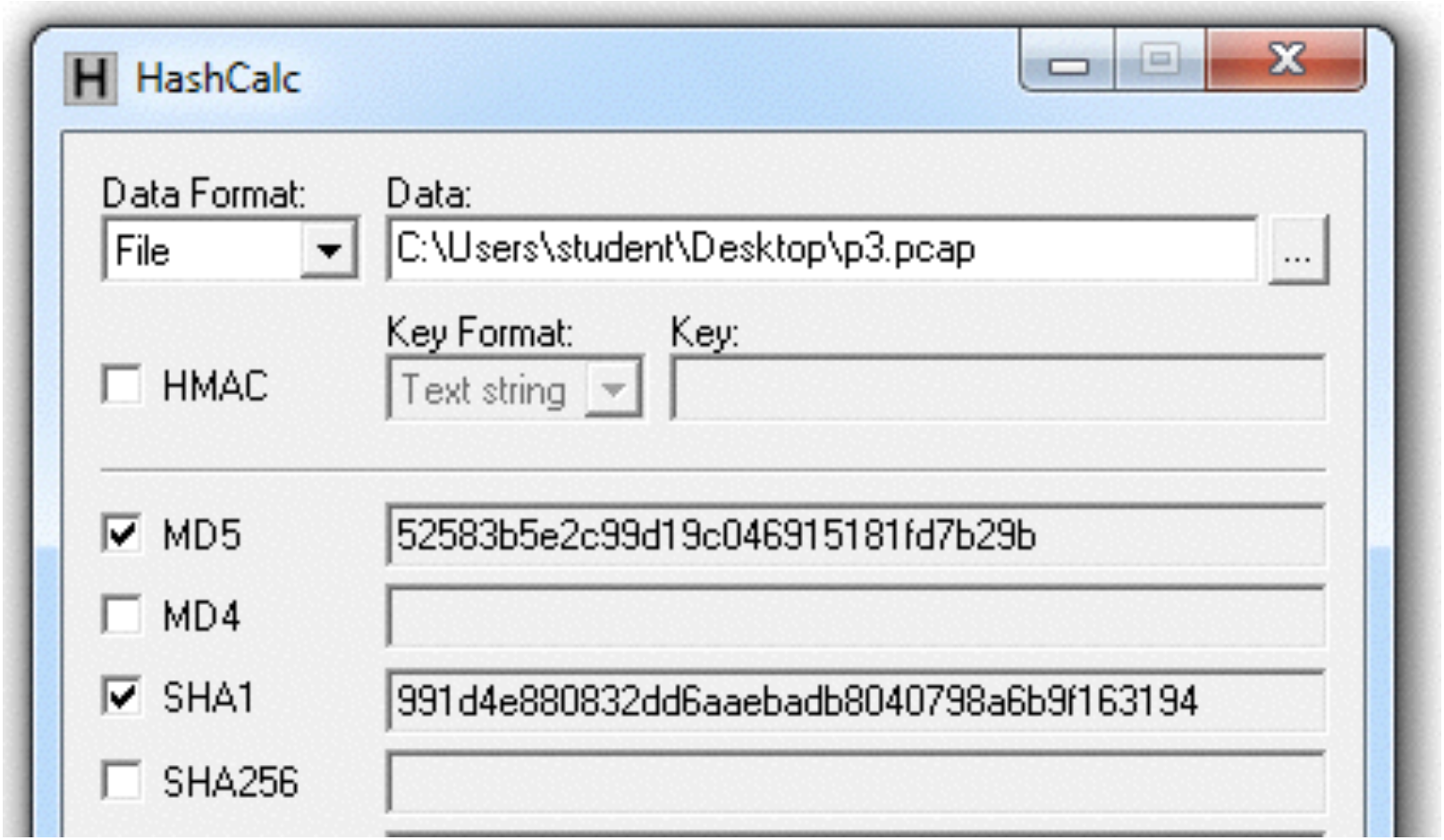
Hashing

A fingerprint for malware

Hashes

- MD5 or SHA-1
- Condenses a file of any size down to a fixed-length fingerprint
- 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

- 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

Finding Strings

Strings

- Any sequence of printable characters is a **string**
- 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"

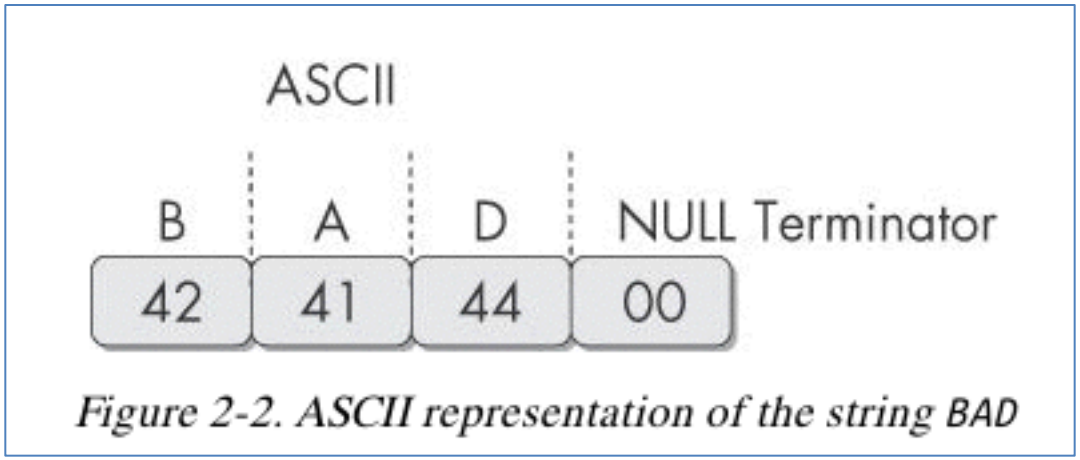


Figure 2-2. ASCII representation of the string BAD

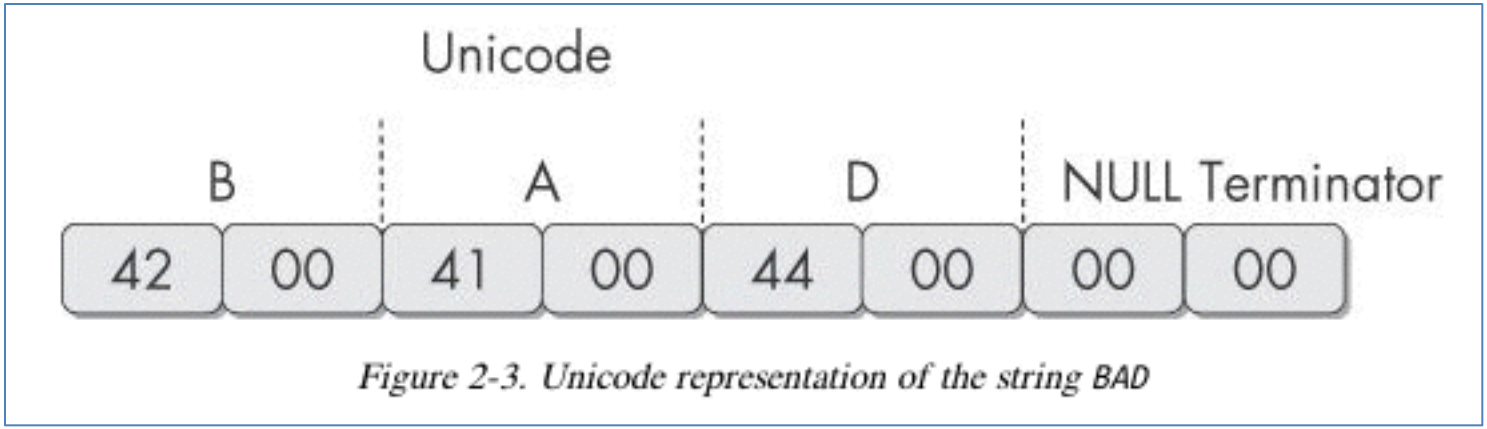


Figure 2-3. Unicode representation of the string BAD

The strings Command

- Native in Linux, also available for Windows
- Finds all strings in a file 3 or more characters long

- strings --> Unix, Linux, WinVista, Win10
- more --> Win7, WinXp, ...

```
more < FilePath.exe | findstr "."
more < FilePath.exe | findstr "....."
```

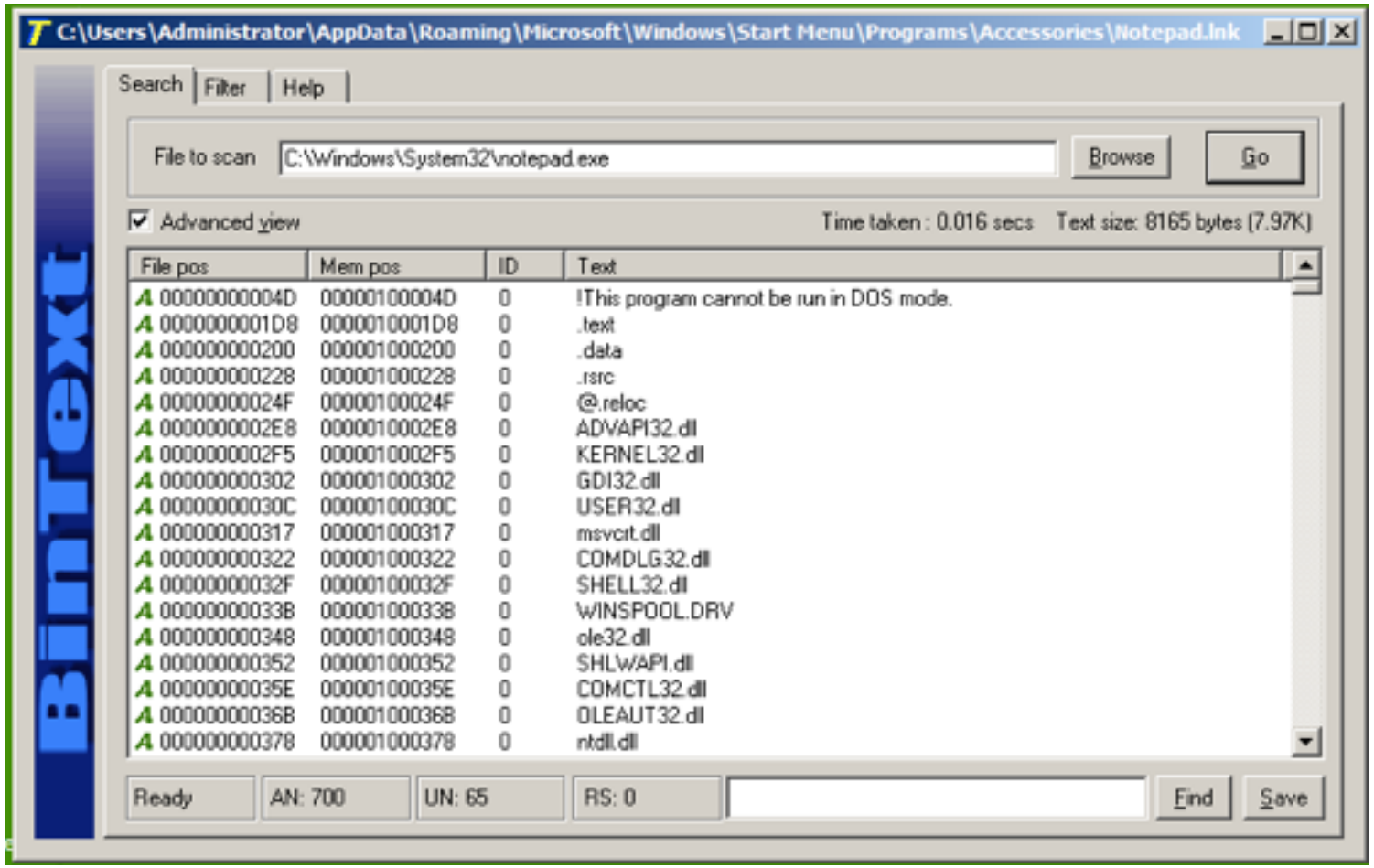
Win 7 için

The strings Command

- 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

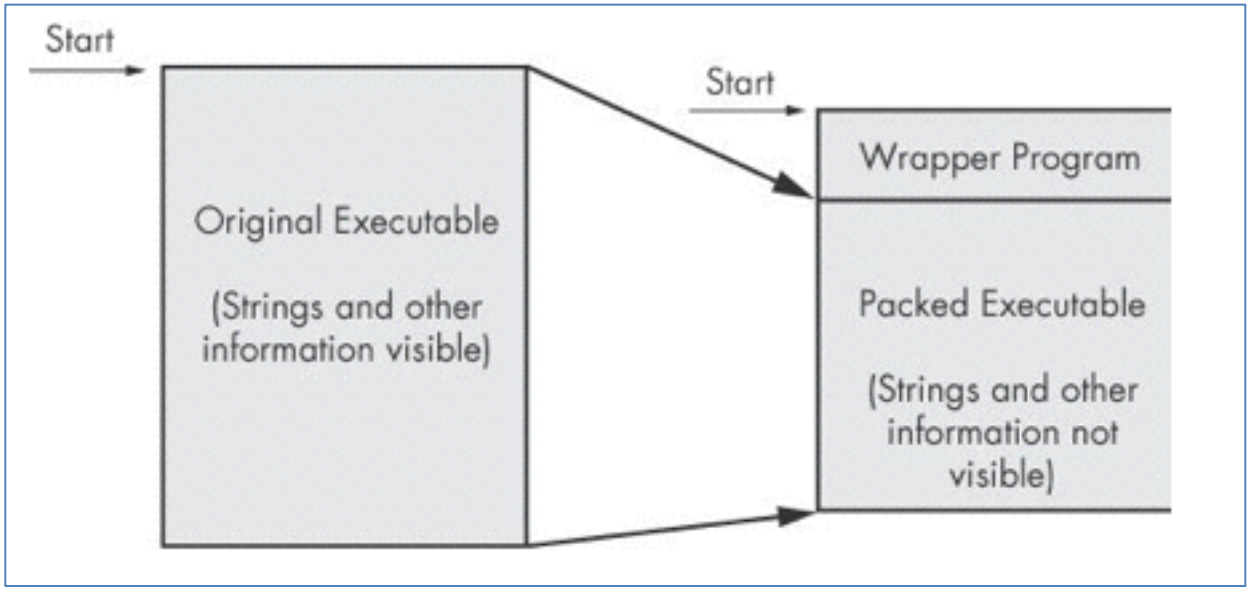


- Link Ch 2i

Packed and Obfuscated Malware

Packing Files

- The code is compressed, like 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



Detecting Packers with PEiD

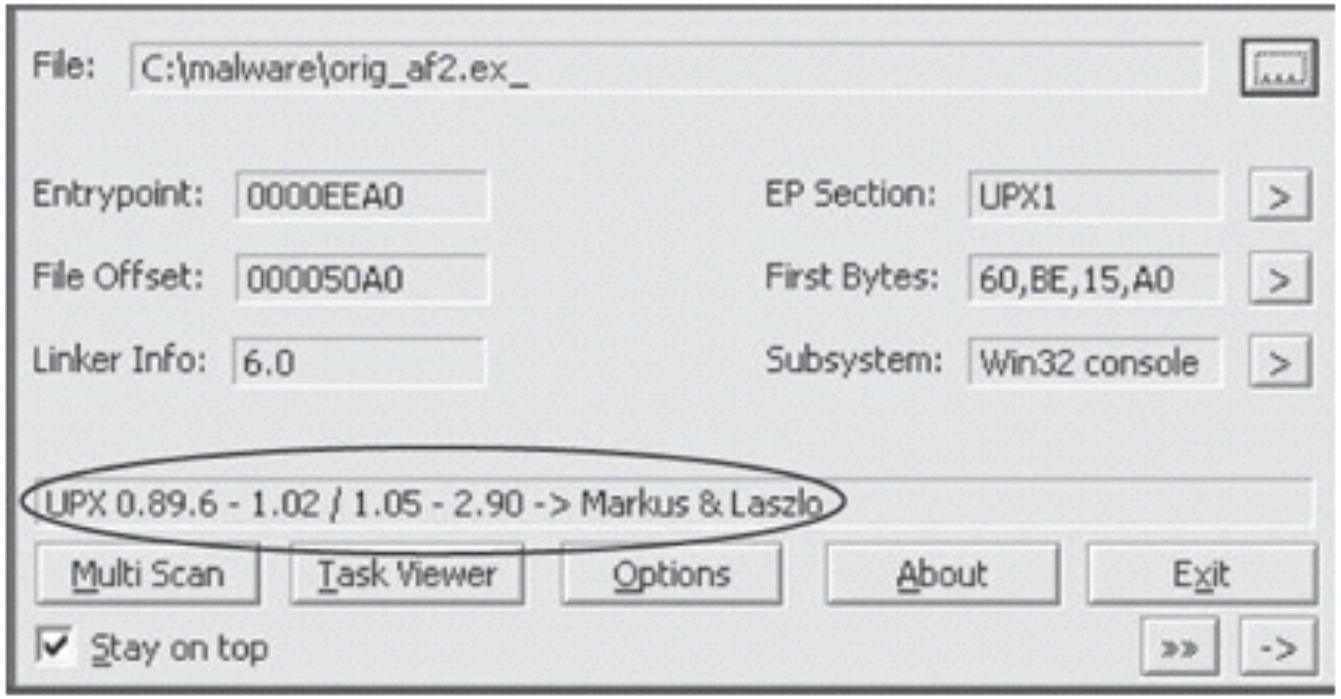
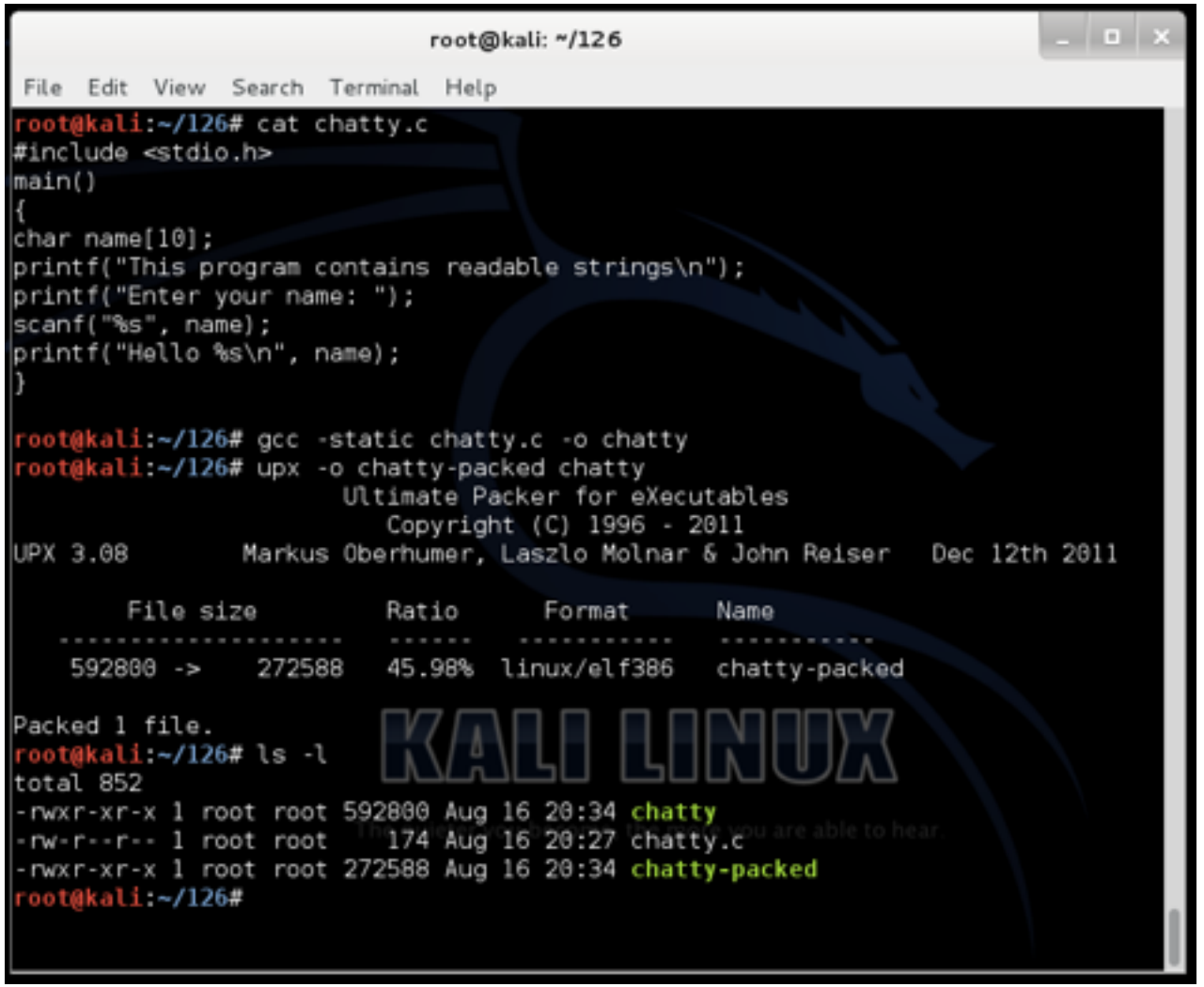


Figure 2-5. The PEiD program

Demo: UPX


root@kali: ~/126
File Edit View Search Terminal Help
root@kali:~/126# cat chatty.c
#include <stdio.h>
main()
{
char name[10];
printf("This program contains readable strings\n");
printf("Enter your name: ");
scanf("%s", name);
printf("Hello %s\n", name);
}
root@kali:~/126# gcc -static chatty.c -o chatty
root@kali:~/126# upx -o chatty-packed chatty
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2011
UPX 3.08 Markus Oberhumer, Laszlo Molnar & John Reiser Dec 12th 2011
File size Ratio Format Name
592800 -> 272588 45.98% linux/elf386 chatty-packed
Packed 1 file.
root@kali:~/126# ls -l
total 852
-rwxr-xr-x 1 root root 592800 Aug 16 20:34 chatty
-rw-r--r-- 1 root root 174 Aug 16 20:27 chatty.c
-rwxr-xr-x 1 root root 272588 Aug 16 20:34 chatty-packed
root@kali:~/126#
KALI LINUX

Packing Obfuscates Strings

NOTE

Many PEiD plug-ins will run the malware executable without warning! (See Chapter 3 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

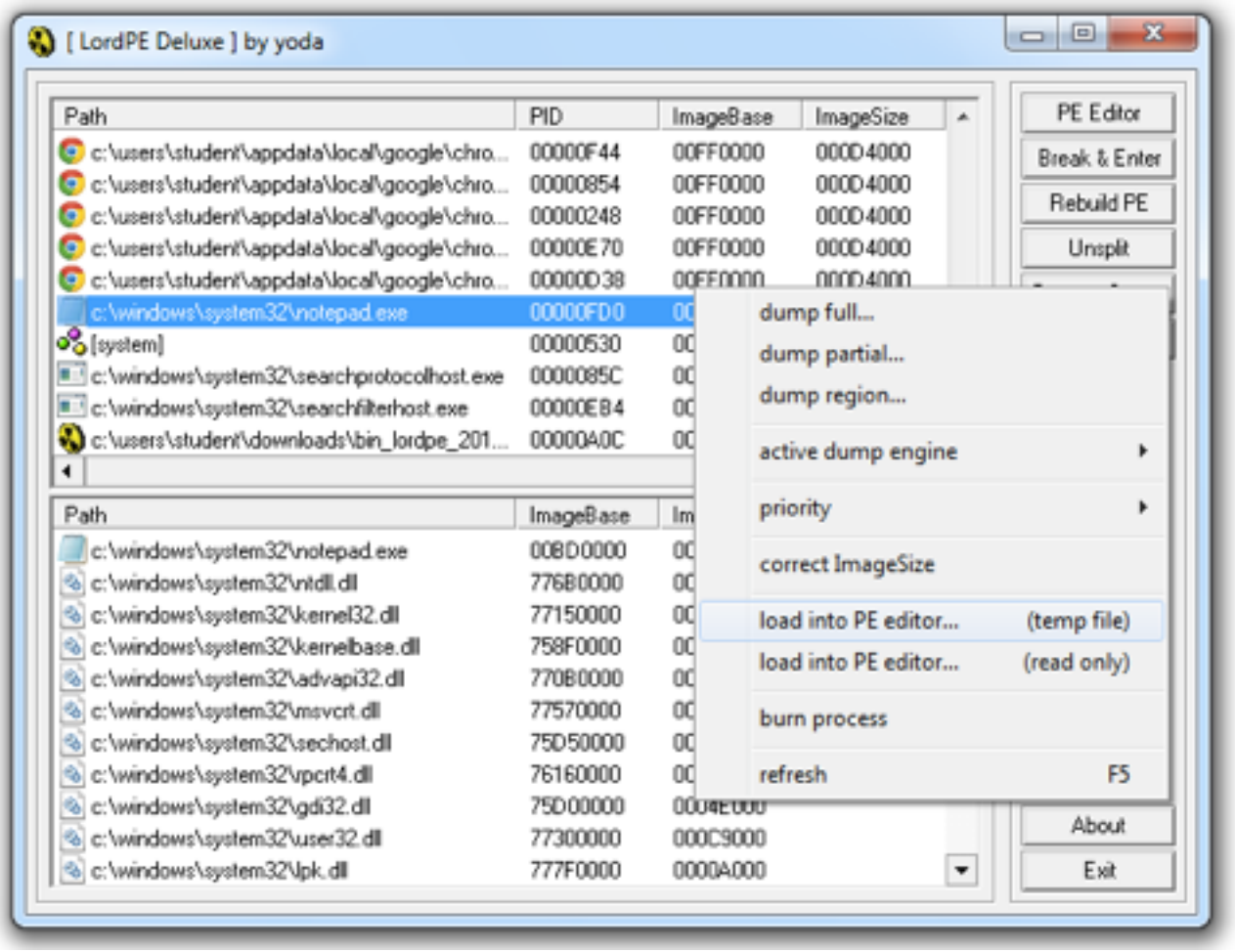
PE Files

- Used by Windows executable files, object code, and DLLs
- A data structure that contains the information necessary for Windows to load the file
- Almost every file executed on Windows is in PE format

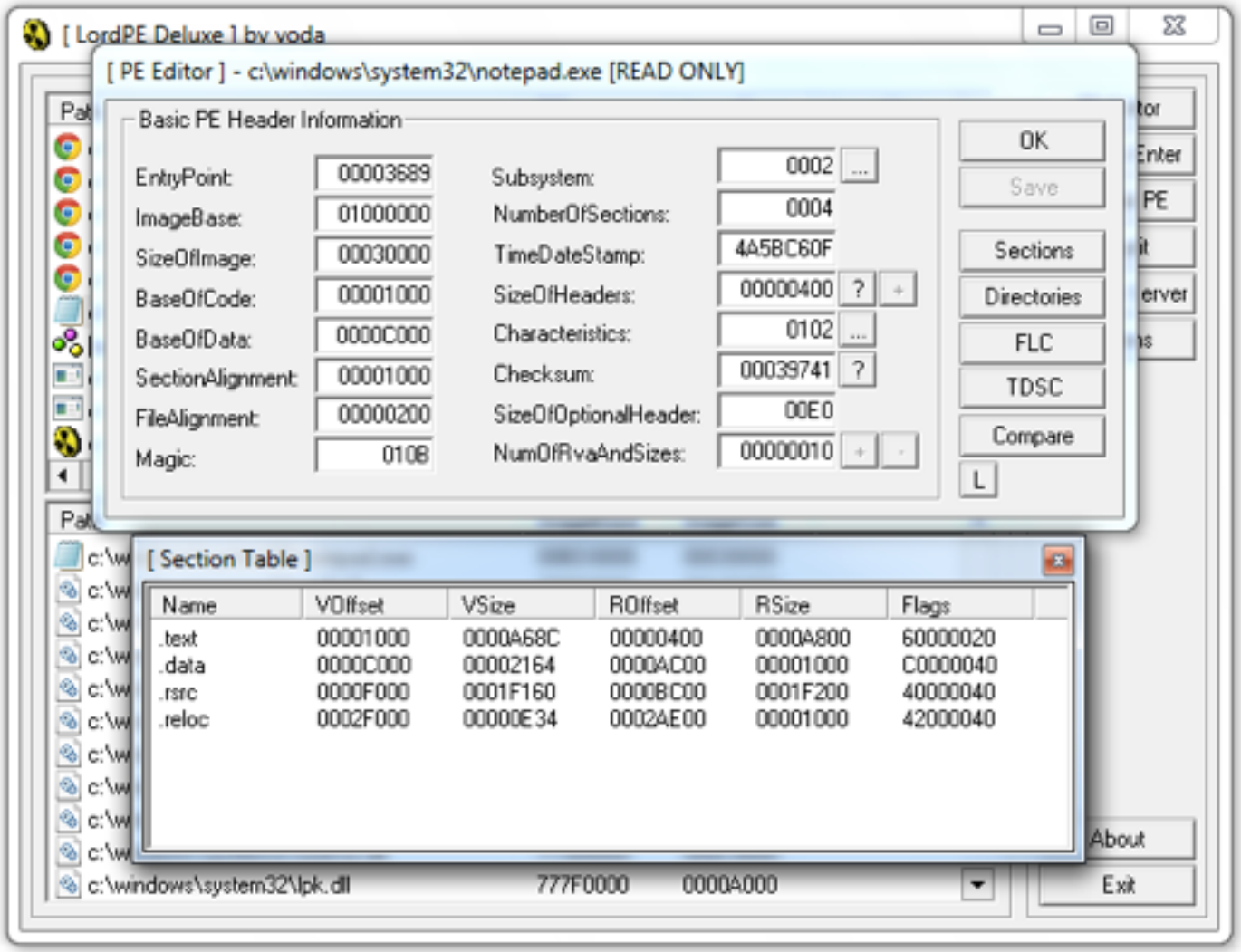
PE Header

- Information about the code
- Type of application
- Required library functions
- Space requirements

LordPE Demo


[LordPE Deluxe] by yoda
Path
PID
ImageBase
ImageSize
c:\users\student\appdata\local\google\chro...
00000F44
00FF0000
000D4000
00000854
00000248
00000E70
00000D38
c:\windows\system32\notepad.exe
00000FD0
[system]
00000530
c:\windows\system32\searchprotocolhost.exe
0000085C
c:\windows\system32\searchfilterhost.exe
00000E84
c:\users\student\downloads\bin_lordpe_201...
00000A0C
ImageBase
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



There are a lot more sections

- But the main ones are enough for now
- Link Ch 2c

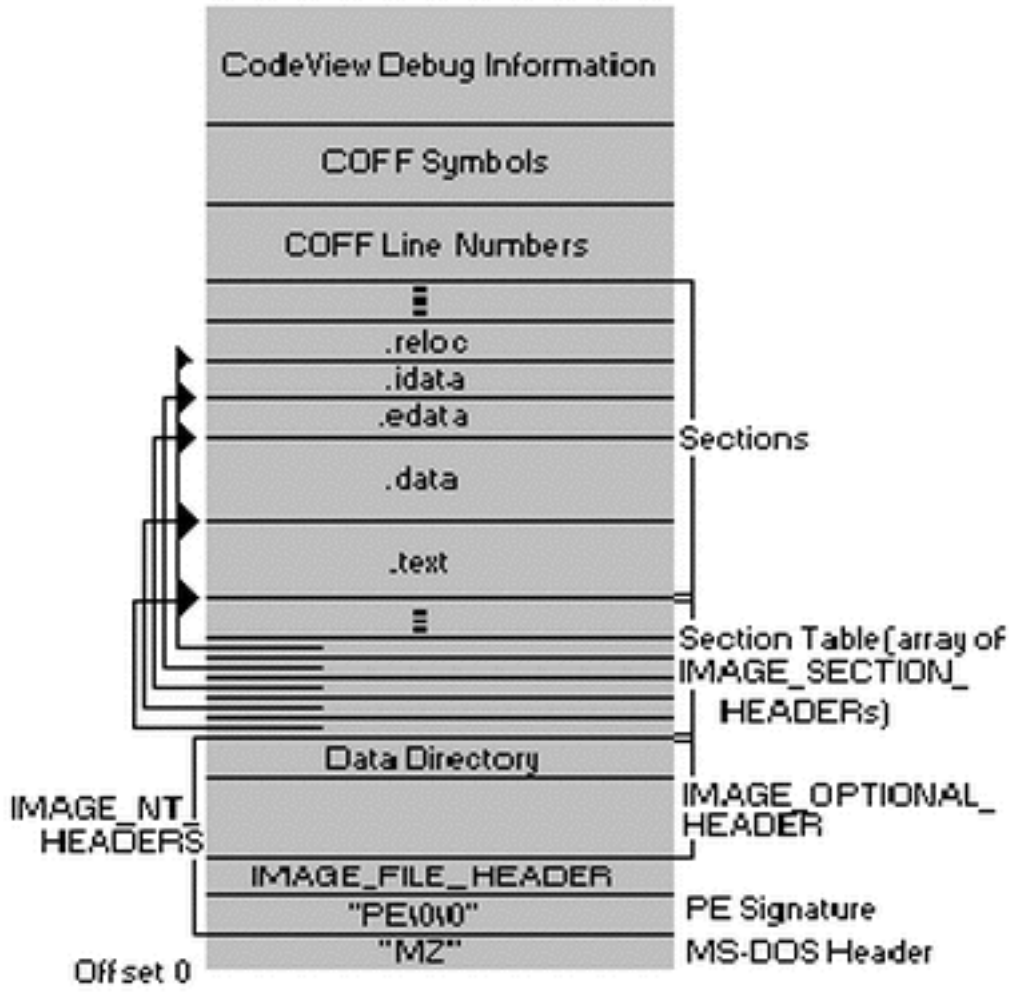


Figure 1. The PE file format

Linked Libraries and Functions

Imports

- Functions used by a program that are stored in a different program, such as library
- Connected to the main EXE by **Linking**
- Can be linked three ways
  - **Statically**
  - At **Runtime**
  - **Dynamically**

Static Linking

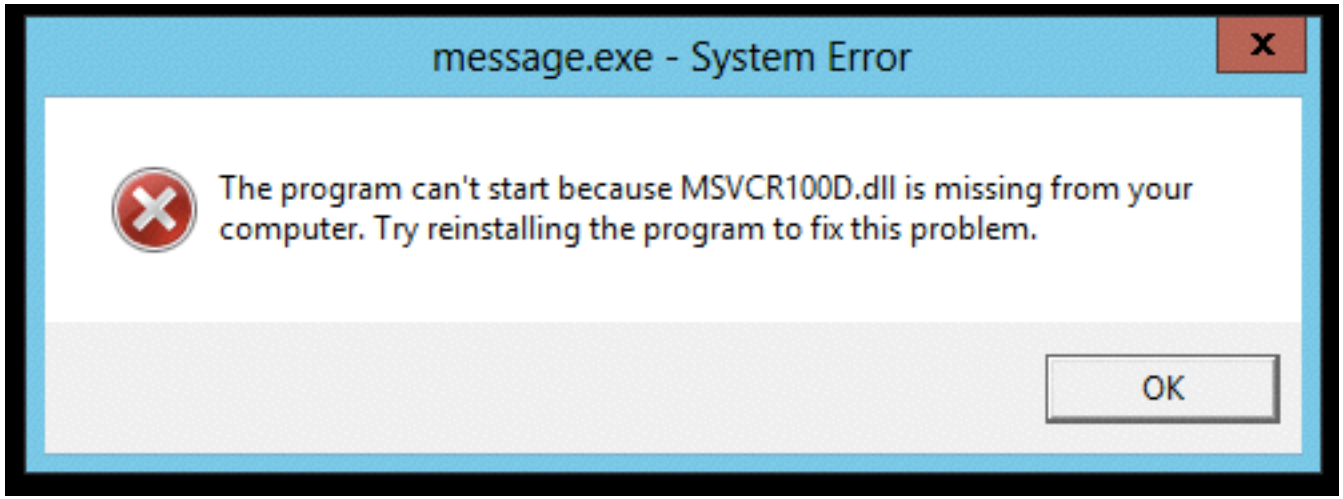
- Rarely used for Windows executables
- Common in Unix and Linux
- All code from the library is copied into the executable
- Makes executable large in size

Runtime Linking

- Unpopular in friendly programs
- Common in malware, especially packed or obfuscated malware
- Connect to libraries only when needed, not when the program starts
- Most commonly done with the **LoadLibrary** and **GetProcAddress** functions

Dynamic Linking

- Most common method
- Host OS searches for necessary libraries when the program is loaded



Clues in Libraries

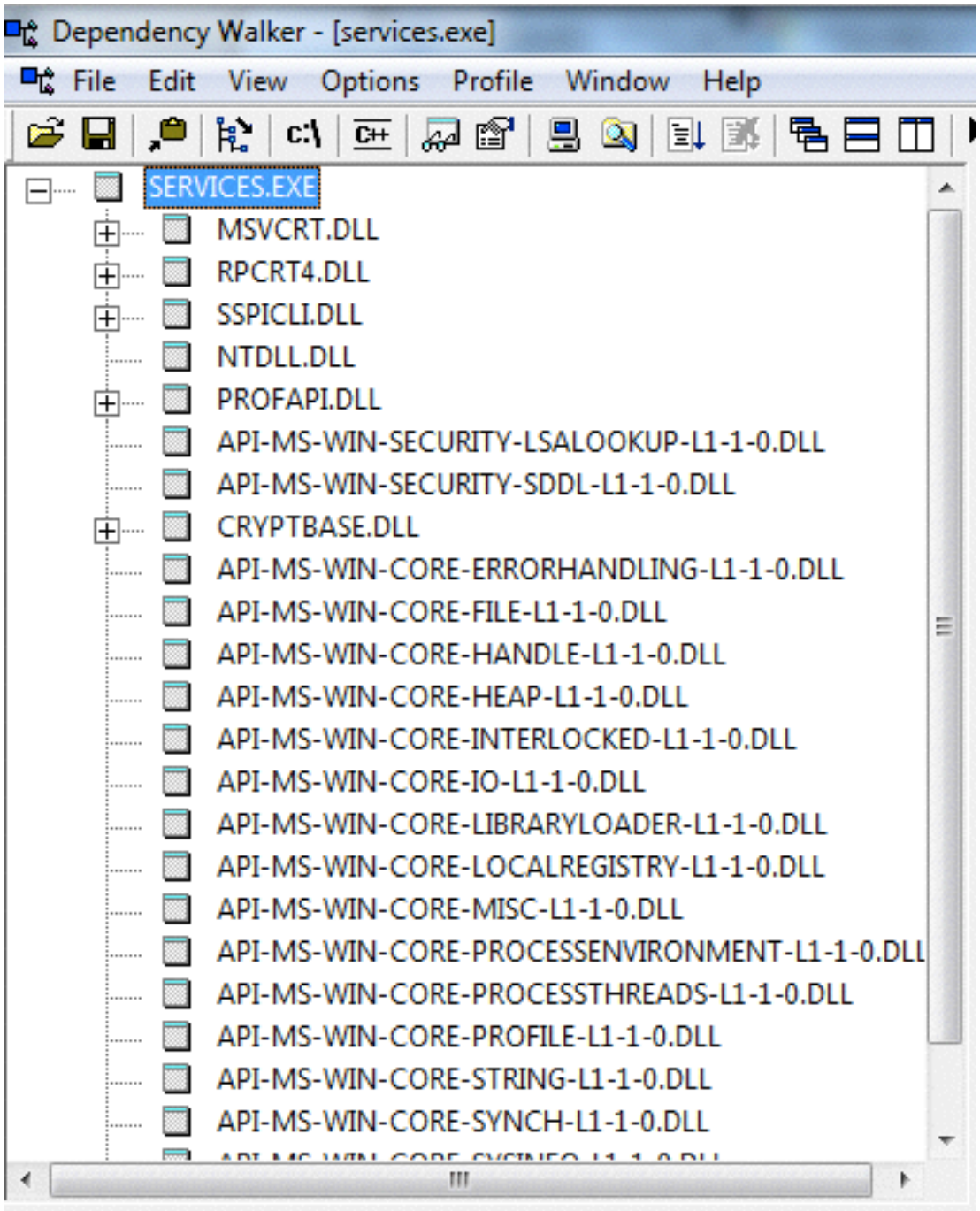
- The PE header lists every library and function that will be loaded
- Their names can reveal what the program does
- **URLDownloadToFile** indicates that the program downloads something

Dependency Walker

Shows Dynamically Linked Functions

- Normal programs have a lot of DLLs
- Malware often has very few DLLs

Services.exe



Services.ex_ (malware)

Imports & Exports in Dependency Walker

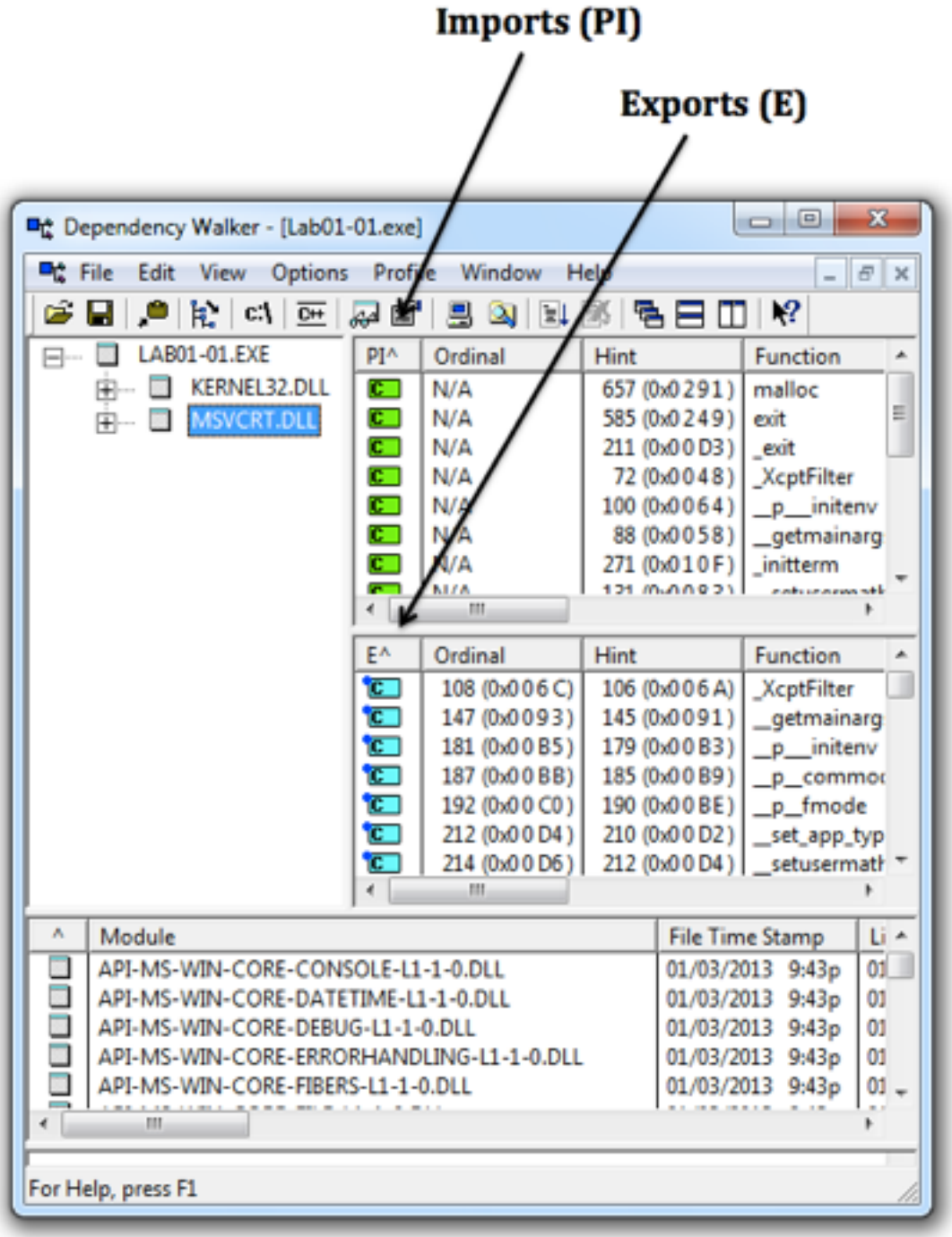


Table 2-1. Common DLLs

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.

Exports

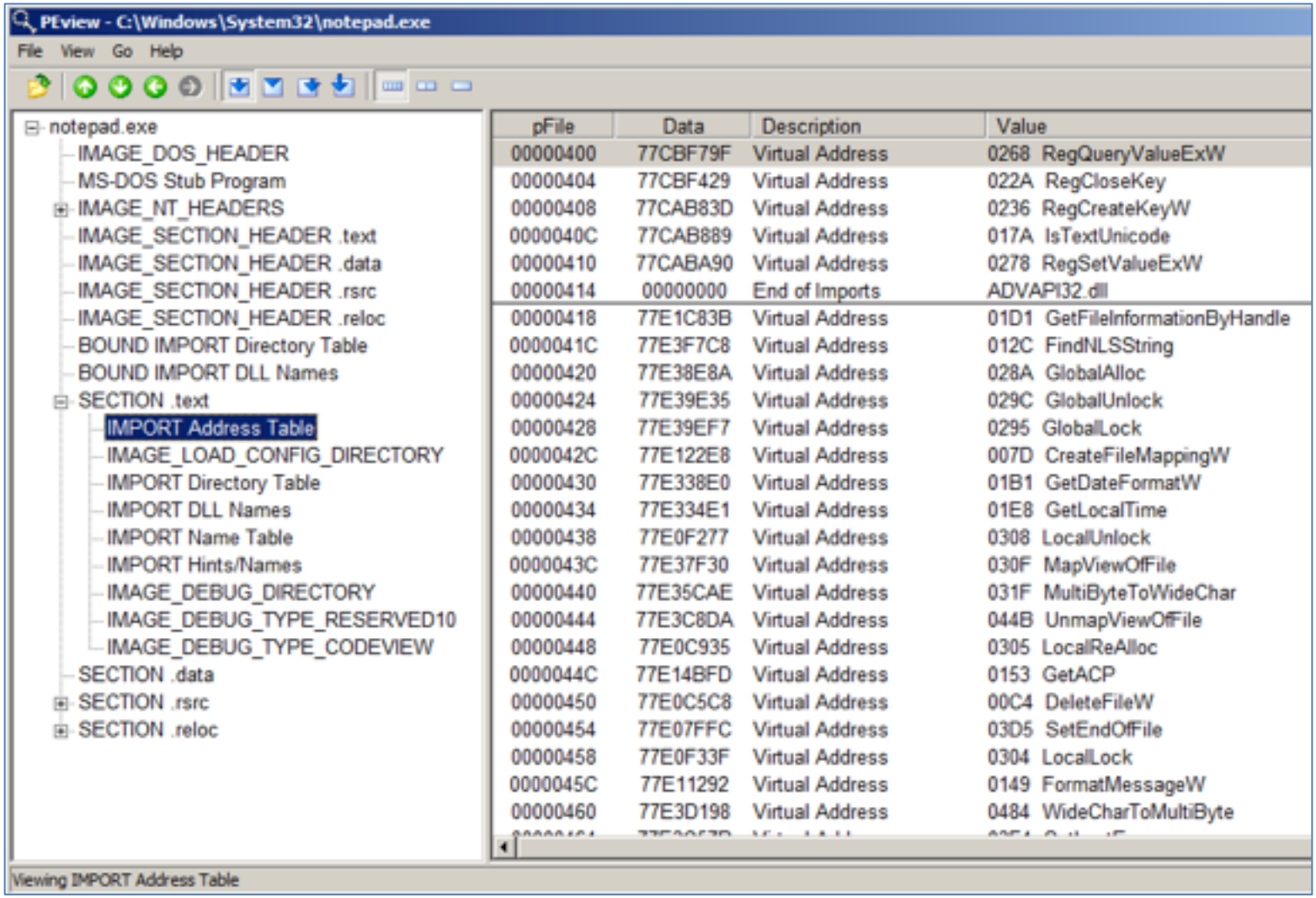
- DLLs **export** functions
- EXEs **import** functions
- Both exports and imports are listed in the PE header

FUNCTION NAMING CONVENTIONS

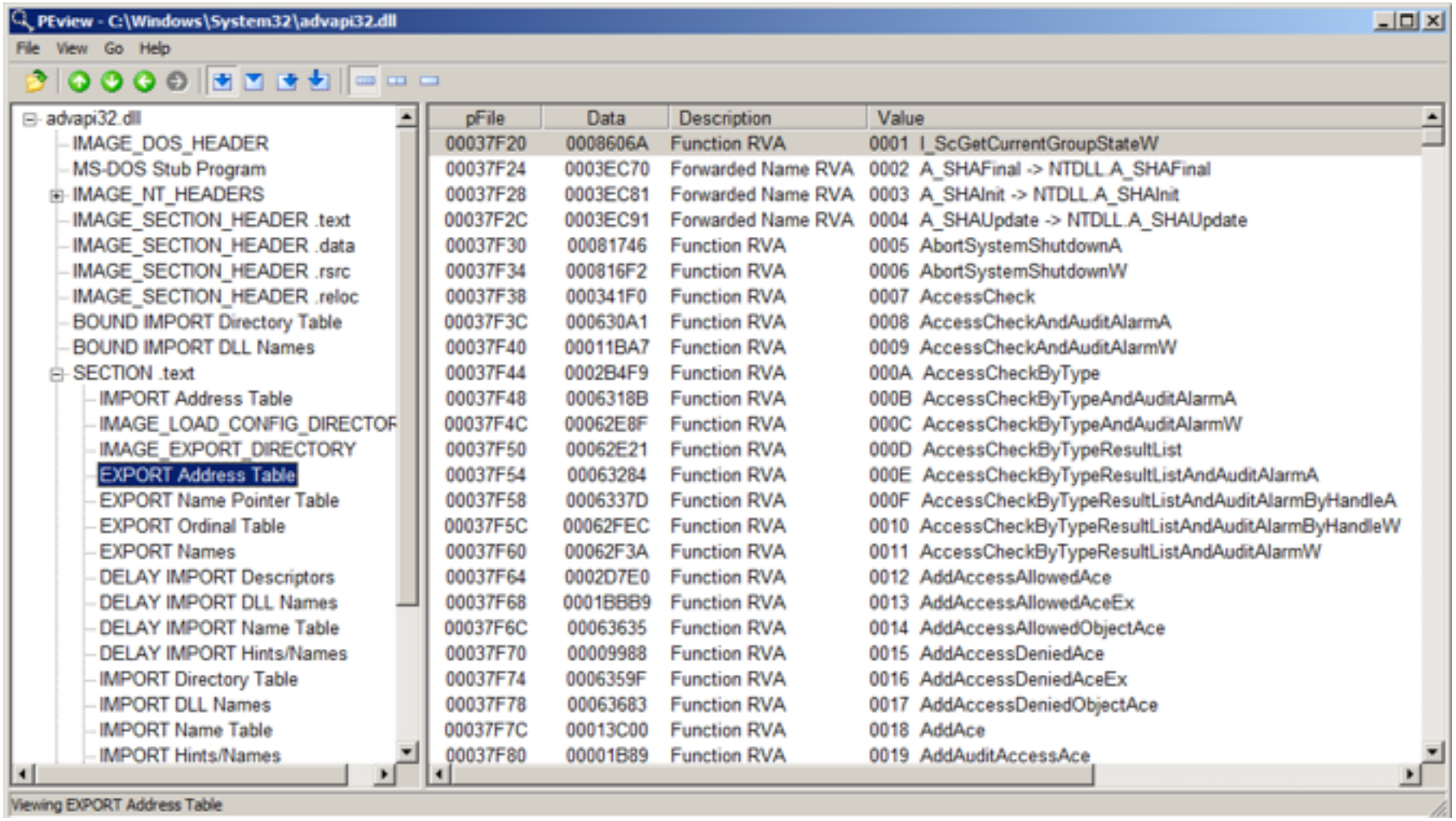
When evaluating unfamiliar Windows functions, a few naming conventions are worth noting because they come up often and might confuse you if you don't recognize them. For example, you will often encounter function names with an `Ex` suffix, such as `CreateWindowEx`. When Microsoft updates a function and the new function is incompatible with the old one, Microsoft continues to support the old function. The new function is given the same name as the old function, with an added `Ex` suffix. Functions that have been significantly updated twice have two `Ex` suffixes in their names.

Many functions that take strings as parameters include an `A` or a `W` at the end of their names, such as `CreateDirectoryW`. This letter does *not* appear in the documentation for the function; it simply indicates that the function accepts a string parameter and that there are two different versions of the function: one for ASCII strings and one for wide character strings. Remember to drop the trailing `A` or `W` when searching for the function in the Microsoft documentation.

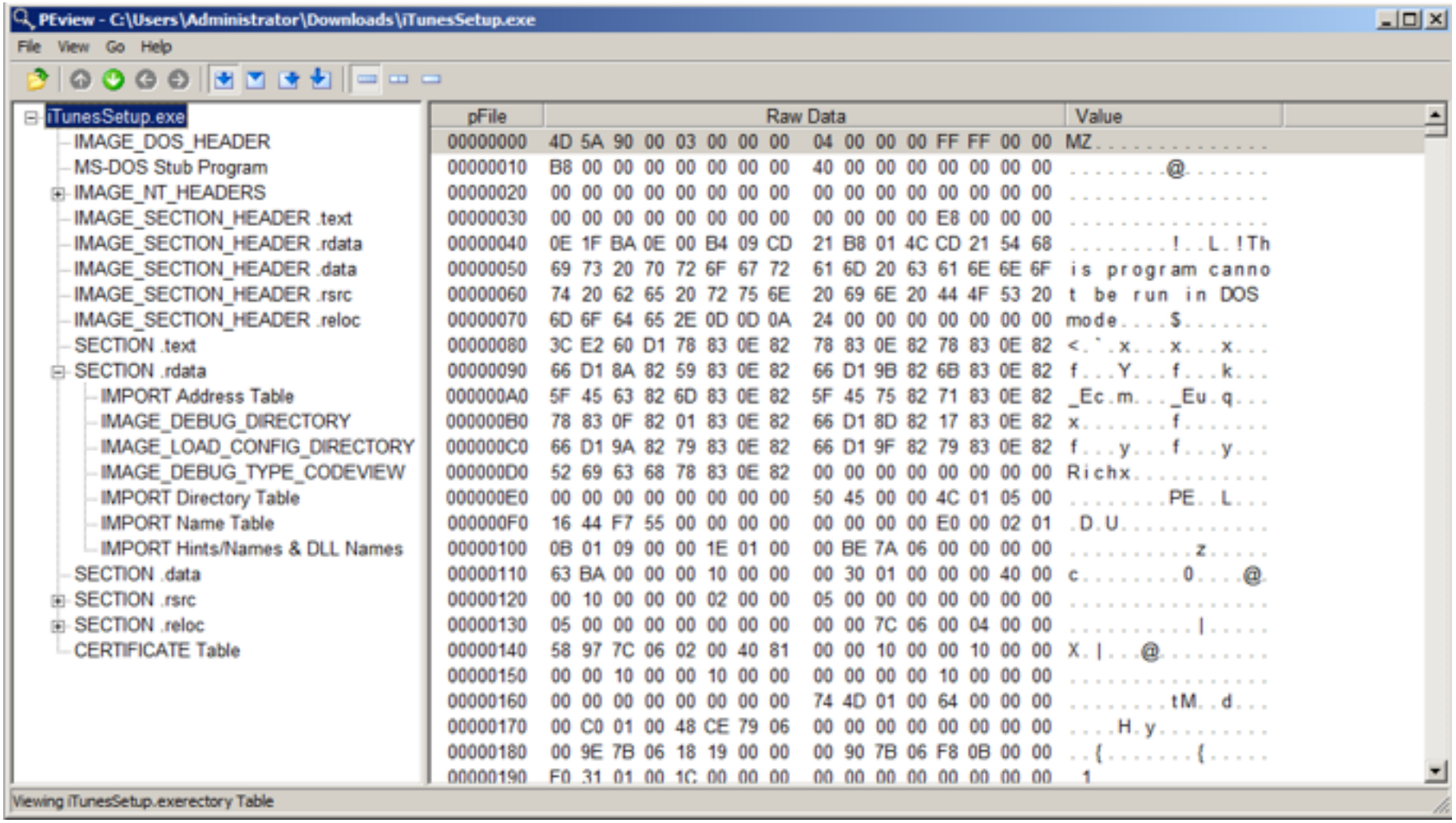
Notepad.exe


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



iTunesSetup.exe



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

Kernel32.dll	User32.dll	User32.dll (continued)
CreateDirectoryW	BeginDeferWindowPos	**ShowWindow**
CreateFileW	CallNextHookEx	ToUnicodeEx
CreateThread	CreateDialogParamW	TrackPopupMenu
DeleteFileW	CreateWindowExW	TrackPopupMenuEx
ExitProcess	DefWindowProcW	TranslateMessage
FindClose	DialogBoxParamW	UnhookWindowsHookEx
FindFirstFileW	EndDialog	UnregisterClassW
FindNextFileW	GetMessageW	UnregisterHotKey
GetCommandLineW	GetSystemMetrics	
GetCurrentProcess	GetWindowLongW	**GDI32.dll**
GetCurrentThread	GetWindowRect	GetStockObject
GetFileSize	GetWindowTextW	SetBkMode
GetModuleHandleW	InvalidateRect	SetTextColor
GetProcessHeap	IsDlgButtonChecked	
GetShortPathNameW	IsWindowEnabled	**Shell32.dll**
HeapAlloc	LoadCursorW	CommandLineToArgvW
HeapFree	LoadIconW	SHChangeNotify
IsDebuggerPresent	LoadMenuW	SHGetFolderPathW
MapViewOfFile	MapVirtualKeyW	ShellExecuteExW
OpenProcess	MapWindowPoints	ShellExecuteW
ReadFile	MessageBoxW	
SetFilePointer	**RegisterClassExW**	**Advapi32.dll**
WriteFile	**RegisterHotKey**	RegCloseKey
	SendMessageA	RegDeleteValueW
	SetClipboardData	RegOpenCurrentUser
	SetDlgItemTextW	RegOpenKeyExW
	SetWindowTextW	RegQueryValueExW
	SetWindowsHookExW	RegSetValueExW

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

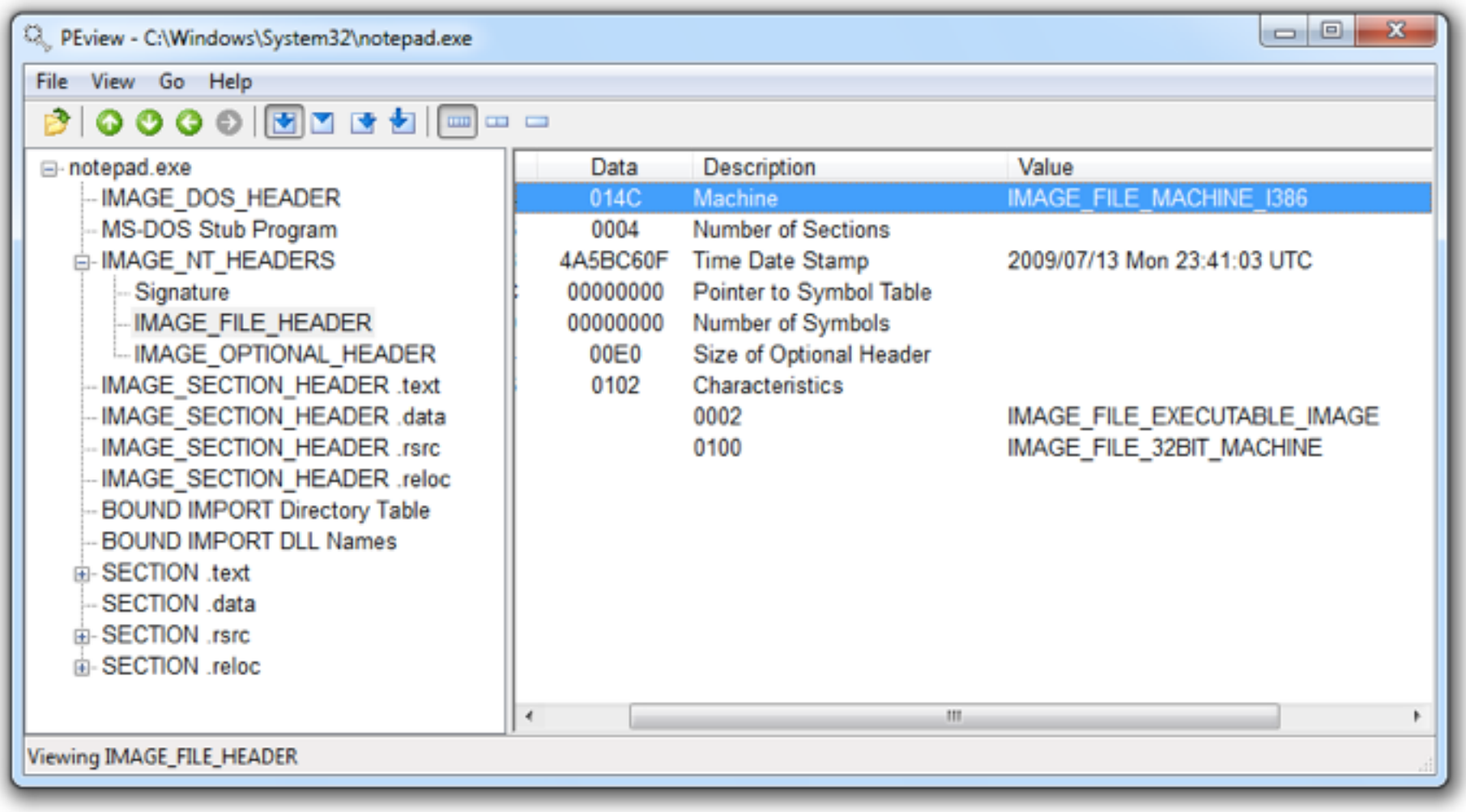
Important PE Sections

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

Table 1-4: Sections of a PE File for a Windows Executable

Executable	Description
`.text`	Contains the executable code
`.rdata`	Holds read-only data that is globally accessible within the program
`.data`	Stores global data accessed throughout the program
`.idata`	Sometimes present and stores the import function information; if this section is not present, the import function information is stored in the `.rdata` section
`.edata`	Sometimes present and stores the export function information; if this section is not present, the export function information is stored in the `.rdata` section
`.pdata`	Present only in 64-bit executables and stores exception-handling information
`.rsrc`	Stores resources needed by the executable
`.reloc`	Contains information for relocation of library files

PEView (Link Ch 2e)



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_SECTION_HEADER

- Virtual Size – RAM
- Size of Raw Data – DISK
- For **.text** section, normally equal, or nearly equal
- Packed executables show Virtual Size much larger than Size of Raw Data for **.text** section

Not Packed

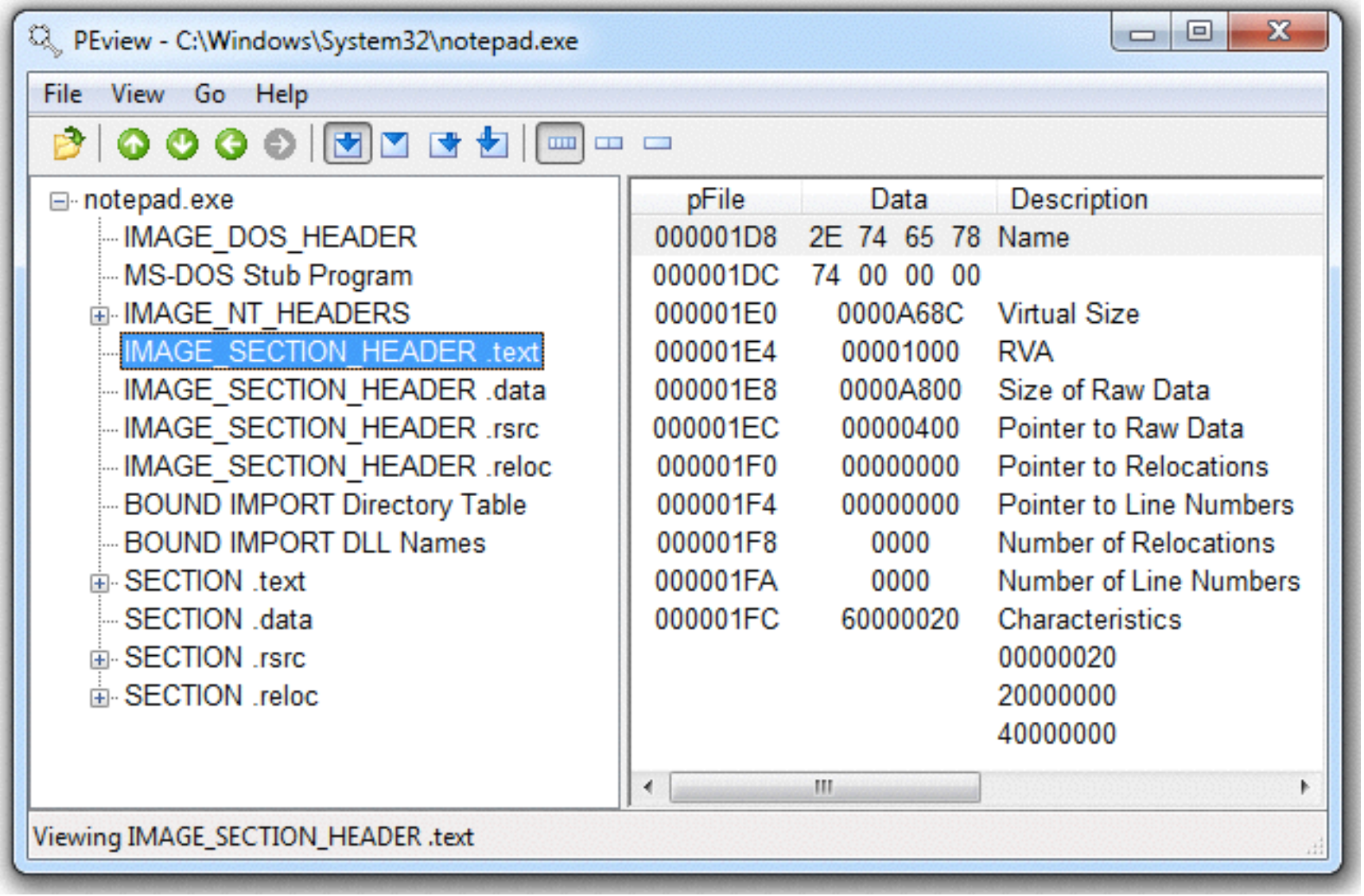

PEview - C:\Windows\System32\notepad.exe
File View Go Help
notepad.exe
IMAGE_DOS_HEADER
MS-DOS Stub Program
IMAGE_NT_HEADERS
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .rsrc
IMAGE_SECTION_HEADER .reloc
BOUND IMPORT Directory Table
BOUND IMPORT DLL Names
SECTION .text
SECTION .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

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

Resource Hacker

- Lets you browse the **.rsrc** section
- Strings, icons, and menus
- Link Ch 2f

Resource Hacker