Info 499

Practical Malware Analysis
Lecture 1 | Basic Static Techniques

Static Analysis

Determine functionality without running the program

1. Antivirus (AV) scan can determine maliciousness
   a. But sometimes you shouldn't...
2. Use hash to identify malware
3. Look at strings, functions, and headers

Static Analysis:

Analyze the code or structure of a program to determine its function
The program is not run at this time

Confirm maliciousness with a preliminary Antivirus scan
There are scenarios where you want to avoid uploading a sample to a public tool
Use the file hash to identify the malware
Look for information in the file's strings, functions, and headers

Antivirus Scanning

Has this been seen before?

- Most AV uses
  - File Signatures
  - Heuristics

Can you fool AV?

- Change or obfuscate code
- New or rare samples

Check to see if the sample has been seen before and identified
Most AV works using
    File Signatures - known pieces of code
    Heuristics - pattern matching
How could you get around AV?
    Change or obfuscate code
    New or rare samples

Fingerprinting via Hash

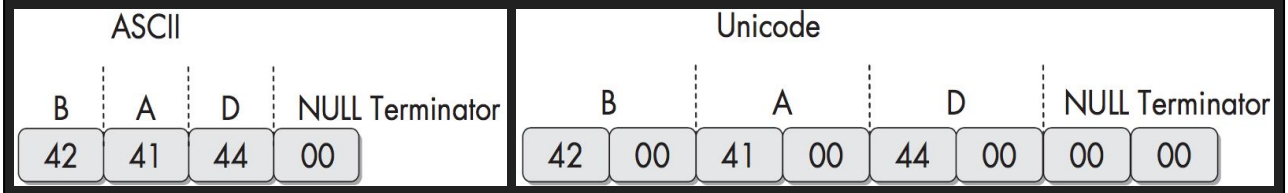
File => hash

- MD5 / SHA-1
  - 9e107d9d372bb6826bd81d3542a419d6 / 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
- Same input => same output
  - Fingerprint
  - Identify change
- Collision-resistant, quick, one-way
- Little change in input => completely different output

Message-Digest Algorithm 5 (MD5) or Secure Hash Algorithm 1 (SHA-1) are popular hashing algorithms that accept a file as input and produce a (mostly) unique hash of the file as output
A given input will always hash to the same output - can be used to fingerprint, check for modifications
Hashing algorithms are collision-resistant, quick, one-way
A small modification to the input produces a vastly different output

Finding Strings

- Hint towards functionality
- Stored as ASCII or Unicode with NULL terminator
  - ASCII uses 1 byte per character
  - Unicode uses 2 bytes per character



Sequences of “human readable” characters
Can give hints about the functionality of code
Typically stored as ASCII or Unicode character encodings the end with a NULL terminator

- ASCII uses 1 byte per character
- Unicode uses 2 bytes per character

Strings

- Searches for 3+ characters + NULL
- Also outputs
  - Memory addresses
  - CPU instructions
  - Data, garbage

```
C:>strings bp6.ex_
VP3
VW3
t$@
D$4
99.124.22.1 ❹
e-@
GetLayout ❶
GDI32.DLL ❸
SetLayout ❷
M}C
Mail system DLL is invalid.!Send Mail failed to send message. ❺
```

Searches for sequences of 3+ characters, followed by a terminator
Sometimes turns up things that aren't strings
- Memory addresses
- CPU instructions
- Data, garbage

Packing and Obfuscation

Hide program execution

Packing compresses, cannot analyze

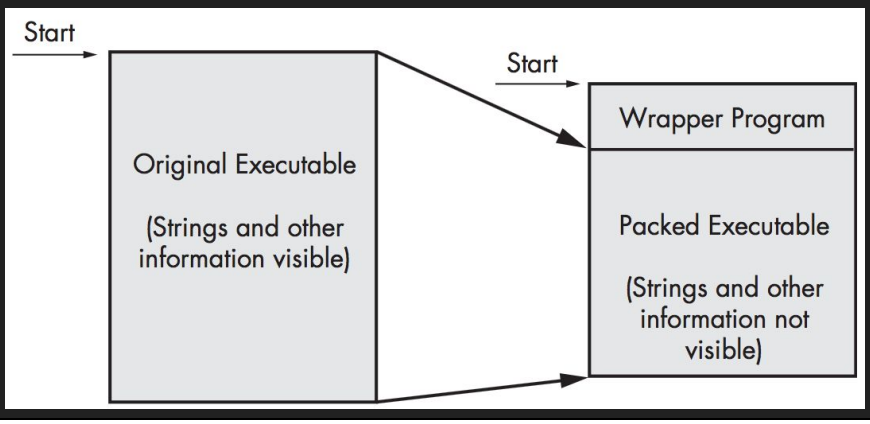
- Small strings output
- Indicates maliciousness

LoadLibrary and **GetProcAddress** functions allow access to more functions

Obfuscation - the execution of the program is hidden by the author
Packing (a type of obfuscation) - the program is compressed and cannot be analyzed
    Strings output will contain very few strings
    Obfuscation is an indicator of malicious intent
Obfuscated code usually includes at least the LoadLibrary and GetProcAddress functions to gain access to more functions

Packed Files

- Wrapper decompresses and runs
- Analyze wrapper statically



Wrapper program decompresses and runs the unpacked file
Usually can only analyze the wrapper statically if a program is packed

Detecting Packers

Identify packers / compilers with **PEiD**

Download and use the packer

*Always run malware in a secure **sandbox** environment

*Know your tools' vulnerabilities

PEiD is a program which can identify the type of packer / compiler used to build an application
Often you will be able to download the packer used and unpack the program and run it
*Always run malware in a secure sandbox environment
*Be wary of tools that may have vulnerabilities that a malware author can exploit

Portable Executable File Format

- Windows executables, object code, and dynamic-link libraries (DLLs)
- A data structure used by the Windows OS loader to manage the wrapped executable code
- Begins with a header including
  - Information about the code
  - Type of application
  - Required library functions
  - Space requirements

File format can reveal information about a program’s functionality
The PE file format is
Used for Windows executables, object code, and dynamic-link libraries (DLLs)
A data structure containing the information necessary for the Windows OS loader to manage the wrapped executable code
Begins with a header including
    Information about the code
    Type of application
    Required library functions
    Space requirements

Linked Libraries and Functions

Borrowing code

- Imports - functions stored in another program
  - Functions shared by many programs
  - Code libraries
- Linking - connecting libraries to the main executable
  - Static or dynamic
  - Type affects information in PE header

Borrowing code
Imports - functions stored in another program
    Functions shared by many programs
    Code libraries
Linking - connecting libraries to the main executable
    Can be static or dynamic
    Linking method affects information in PE header

Linking

- Static linking
  - Least common, used in UNIX/Linux
  - Code copied into executable
  - PE file header does not indicate linked code
- Runtime linking
  - Commonly used in malware
  - Connect to function only when needed
  - PE file header does not indicate linked code
  - Windows - **LoadLibrary** and **GetProcAddress** allow access to any function in any library on the system

Static linking
	Least common, often used in UNIX/Linux programs
	All code is copied into executable
	PE file header does not indicate linked code
Runtime linking
	Commonly used in malware
	Connect to function only when needed
	PE file header does not indicate linked code
	Windows - LoadLibrary and GetProcAddress allow access to any function in any library on the system

Linking

- Dynamic Linking
  - Most common
  - OS finds libraries on program load
  - When the linked function is called, executes within the library
  - Every library and function used shows up in the PE file header
  - Use **Dependency Walker** to view linked functions

Dynamic Linking
Most common
OS finds libraries on program load
When the linked function is called, executes within the library
Every library and function used shows up in the PE file header
Use Dependency Walker to view linked functions

Possible to import functions by Ordinal

- Names of functions won't show up in the executable
- Match the ordinal number in the top pane with its corresponding function in the exports pane.

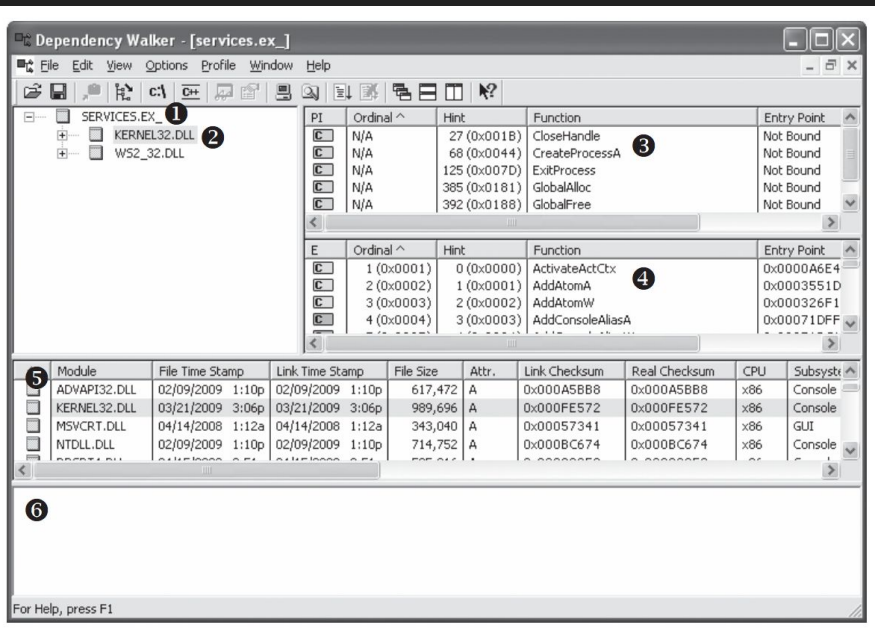


Figure 1-6: The Dependency Walker program

Possible to import functions by Ordinal

Names of functions won't show up in the exe, possible to match the ordinal number in the top pane with its corresponding function in the exports pane.

Common DLLs

- Kernel32
  - Core functionality: Access/change memory, files, hardware
- Advapi32
  - Advanced Windows access: Service manager, registry
- User32
  - UI components
- Gdi32
  - Display/change graphics

Kernel32
	Core functionality: Access/change memory, files, hardware
Advapi32
	Advanced Windows access: Service manager, registry
User32
	UI components
Gdi32
	Display/change graphics

Common DLLs

- Ntdll
  - Interface to Windows kernel
  - Imported indirectly by Kernel32
  - Explicit import => access to uncommon functions
- WSock32 / Ws2_32
  - Networking
- Wininet
  - High-level networking functions

Ntdll

Interface to Windows kernel

Imported indirectly by Kernel32

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 / Ws2_32

Networking

Wininet

High-level networking functions that implement protocols such as HTTP, NTP, FTP

Notable Naming Conventions

- -Ex (**CreateWindowEx**)
  - Updated function, incompatible with old
  - Twice-updated will have -ExEx
- -A or -W (**CreateDirectoryW**)
  - Accepts a string as a parameter
  - Not in function documentation
  - ASCII or wide (ex. UTF-16)

-Ex (CreateWindowEx)

Significantly updated function, incompatible with old but Microsoft still support the old one

Twice-updated will have -ExEx

-A or -W (CreateDirectoryW)

Accepts a string as a parameter

Not in function documentation so drop the suffix if you are looking for documentation

ASCII or wide character strings

Imported Functions

- Included in PE file header
- Windows API documented via Microsoft Developer Network (MSDN) library
- Give you a good idea of what the executable does

Included in PE file header
Windows API documented via Microsoft Developer Network (MSDN) library
Give you a good idea of what the executable does

Exported Functions

- Listed in PE file
- DLL implements and exports functions
- Executables typically import, exports are rare
- Authors name exports following MS docs
  - But they don’t have to (obfuscation)
- Can also be viewed with Dependency Walker

Listed in PE file
DLL implements and exports functions
Executables typically import, exports are rare
Authors name exports following MS docs
But they don’t have to (obfuscation)
Can also be viewed with Dependency Walker

Example: PotentialKeylogger.exe

- Is it packed?
- What interesting functionality can be derived from the functions?

Table 1-2: An Abridged List of DLLs and Functions Imported from *PotentialKeylogger.exe*

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

Is it packed?

We can see a lot of imported functions so it's not packed

What interesting functionality can be derived from the functions?

- Open and modify processes and files
  - OpenProcess
  - GetCurrentProcess
  - GetProcessHeap
  - Read/Create/WriteFile
- Search directories
  - FindFirst/FindNextFile
- GUI manipulation
  - RegisterClassEx
  - SetWindowText
  - ShowWindow
- Hook events ಠ_ಠ
  - SetWindowsHookEx
- Notified on hotkey press
  - RegisterHotKey

Table 1-2: An Abridged List of DLLs and Functions Imported from *PotentialKeylogger.exe*

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

We can tell from the imports that this executable has the ability to

Open and modify processes and files, search directories

There is functionality for GUI manipulation so it may have a GUI (doesn't have to be displayed to the user)

The SetWindowsHookEx function has legitimate uses, but is suspicious and commonly used in spyware and keyloggers to receive keyboard input

Set a hotkey that notifies the application when the user presses key(s), regardless of which application is active

- More GUI-related imports
  - GDI32.dll
- Launch other programs
  - Shell32.dll
- Use the registry
  - Advapi32.dll

Searching for strings that look like registry keys turns up:

Software\Microsoft\Windows\CurrentVersion\Run

which controls Windows startup programs ಠ_ಠ

Table 1-2: An Abridged List of DLLs and Functions Imported from *PotentialKeylogger.exe*

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

The imports from GDI32.dll are graphics-related and are further indication that the program has a GUI

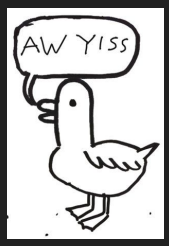
The imports from Shell32.dll tell us this program can launch other programs, a function common to malware and legitimate programs

The registry-related imports may suggest persistence mechanisms, look for registry key strings
Software\Microsoft\Windows\CurrentVersion\Run -> startup programs -> commonly used by malware for persistence

PotentialKeylogger.exe Exports

- LowLevelKeyboardProc and LowLevelMouseProc
  - Used with SetWindowsHookEx (MSDN) to connect an event to a function
  - Function called on low-level keyboard events

- Author named export after MS function



LowLevelKeyboardProc and LowLevelMouseProc
Used with SetWindowsHookEx (MSDN) to connect an event to a function
Function called on low-level keyboard events

Author named export after MS function
This makes our job a lot easier because we can more easily draw conclusions about what the program does

PotentialKeylogger.exe Hypothesis

- Probably a keylogger
  - Uses SetWindowsHookEx to log keystrokes
  - Has a GUI for the author
  - Author uses hotkey to access GUI
  - Uses a registry key to run the keylogger on startup

Probably a keylogger
Uses SetWindowsHookEx to log keystrokes
Has a GUI for the author
Author uses hotkey to access GUI
Uses a registry key to run the keylogger on startup

Example: PackedProgram.exe

Is it packed?

- Short import list
- No readable strings
- Requires dynamic

Table 1-3: DLLs and Functions Imported from *PackedProgram.exe*

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

Short import list, probably packed
Running strings would show that there are no human-readable strings
Nothing further we can do with this, requires dynamic analysis

PE File Headers and Sections

Header => metadata about file

Sections => pieces of the file

- .text
  - CPU instructions
  - Only section with code
- .rdata
  - Import/export information
  - Globally accessible read-only data
  - Sometimes split into .idata and .edata

PE file headers contain more than just imports
The PE file format specifies headers followed by sections
    Header => metadata about file
    Sections => pieces of the file

Some common interesting sections are
.text
    CPU instructions
    Only section with code
.rdata
    Import/export information
    Globally accessible **read**-only data
    Sometimes split into .idata and .edata

PE File Sections

- .data
  - Global data
- .pdata (x64 only)
  - Exception-handling info
- .rsrc
  - Resources (icons, images, menus, strings)
- .reloc
  - Info regarding relocation of library files

.data

Global data accessible from anywhere in the program

Local data is not stored here, or anywhere else in the PE file

.pdata (x64 only)

Exception-handling info

.rsrc

Resources (icons, images, menus, strings)

Strings can be stored here or in the main program, but are usually here for multilanguage support

.reloc

Info regarding relocation of library files

PE File Sections

- Section names may vary per compiler
- Windows doesn’t care
- Section names can be obfuscated

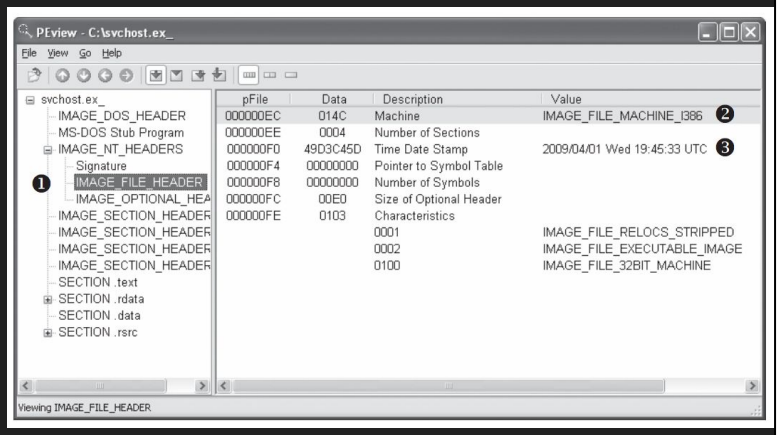
Section names usually stay the same for one compiler but may vary across compilers

Windows uses other info in the PE header to figure out how a section should be used

Section names can be obfuscated but are usually default

PEview

1. Main parts
2. Basic info
3. Time Date Stamp



IMAGE_OPTIONAL_HEADER (not shown)

Use PEview to browse PE header information

1. The main parts of a PE header
   a. IMAGE_DOS_HEADER & MS-DOS Stub Program are historical
   b. IMAGE_NE_HEADERS signature doesn't change, ignore it
2. IMAGE_FILE_HEADER contains basic info about the file
3. The time stamp tells when the executable was compiled (can be faked)
   a. Delphi programs use June 19, 1992 always

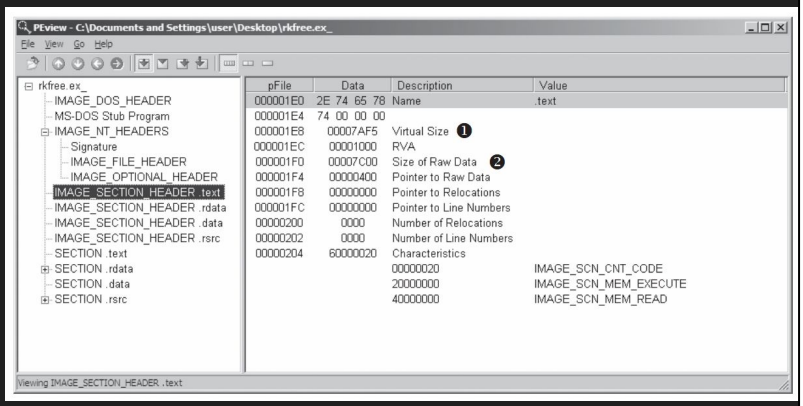
IMAGE_OPTIONAL_HEADER

Subsystem indicates console or GUI

IMAGE_SUBSYSTEM_WINDOWS_CUI = console
IMAGE_SUBSYSTEM_WINDOWS_GUI = GUI
Native / Xbox also used

PEview

1. Virtual Size
2. Size of Raw Data



IMAGE_SECTION_HEADERs describe each section of the PE file - dictated by compiler, deviations are uncommon and suspect

1. Virtual Size tells how much space is allocated for a section during the loading process
2. Size of Raw Data tells how big the section is on disk
   a. Should usually be about equal, data should take up as much space on disk as in memory
   b. Small differences are normal due to alignment

Virtual Size > Raw Data = packed, especially if .text is larger in memory than on disk

PEview

Section Information for PotentialKeylogger.exe

Section	Virtual size	Size of raw data
.text	7AF5	7C00
.data	17A0	0200
.rdata	1AF5	1C00
.rsrc	72B8	7400

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

PotentialKeylogger.exe has sections with similar sizes in memory and on disk

Except for .data, though this is normal for .data in Windows programs

Doesn't mean it's not malicious, just that it probably isn't packed and that the PE file header is compiler-generated

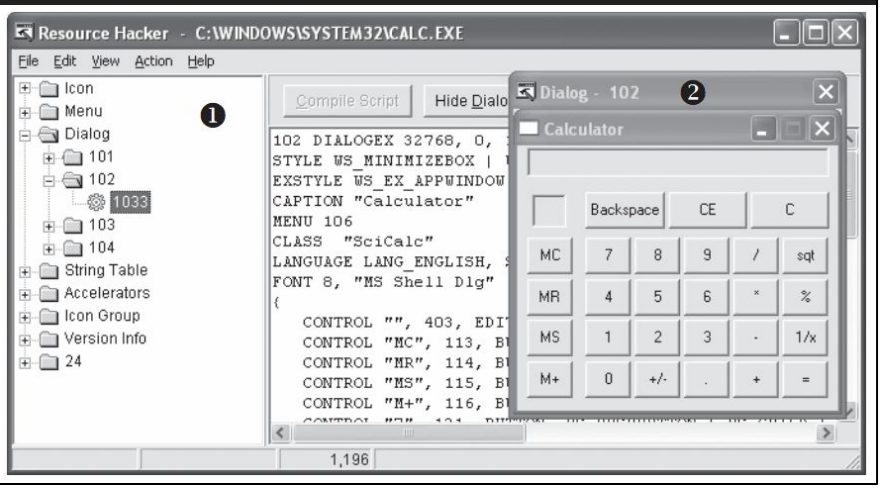
PackedProgram.exe has oddly named sections, sections that take up zero space on disk, and the .text section has a Virtual Size of A000 meaning that space will be allocated for that segment.

Indicative of a packer which will unpack the executable code that is allocated to the .text section.

Resource Hacker

Only section accessible thus far is .rsrc

- Icon
- Menu
- Dialog
- String Table
- Version Info



With our current knowledge, we can examine the .rsrc section using Resource Hacker to view the strings, icons, and menus

Menus displayed are identical to what the program uses.

1. The resources in the executable
    a. Each root folder contains a different type of resource
        i. Icon section lists images displayed when the executable is in a file listing
        ii. Menu section stores all menus that appear, i.e. File, Edit, and View.
        iii. Dialog shows dialog menus (2) shows what is displayed to the user on run
        iv. String Table stores strings
        v. Version Info contains the version number, company name, and copyright

NOTE Software (mostly Malware) can store an embedded program or driver here and extract it before the program runs. Resource Hacker can extract these files.

Other PE File Tools

PEBrowse Professional

- Similar to PEview, better view of .rsrc

PE Explorer

- GUI for navigating / editing, including a resource editor
- $$$

Summary

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

Static analysis is useful, but only the first step...

Labs

- Given little or no information about the samples
- Generically named
- Malicious file provided
- Short questions, short answers
- Detailed analysis
- Answers in Appendix C

Resources

Use VirusTotal to scan or search for information on files, URLs, domains, IPs, hashes, etc.

Use a public sandbox such as Hybrid Analysis or Malwr to perform static (and some dynamic) analysis for you

Set up a private sandbox such as cuckoo for private analysis

Strings for Windows

Dependency Walker (Windows)

Resources

PEview (Windows)

Resource Hacker (Windows)

PEBrowse Professional (Windows)

PE Explorer (Windows)

PEiD (Windows)