BASIC STATIC ANALYSIS

MALWARE ANALYSIS AND INCIDENT FORENSICS
M.Sc. in Cyber Security

SYSTEMS AND ENTERPRISE SECURITY
M.Sc. in Engineering in Computer Science

SAPIENZA
UNIVERSITÀ DI ROMA

CIS SAPIENZA
CYBER INTELLIGENCE AND INFORMATION SECURITY

THE GOAL OF MALWARE ANALYSIS

Depp understanding on how malware works is crucial for:

Monitoring and attack prevention

- Identify recurring attack patterns
- Build rules for monitoring and intrusion detection
- Design IT systems that more resilient to forthcoming attacks

Incident response

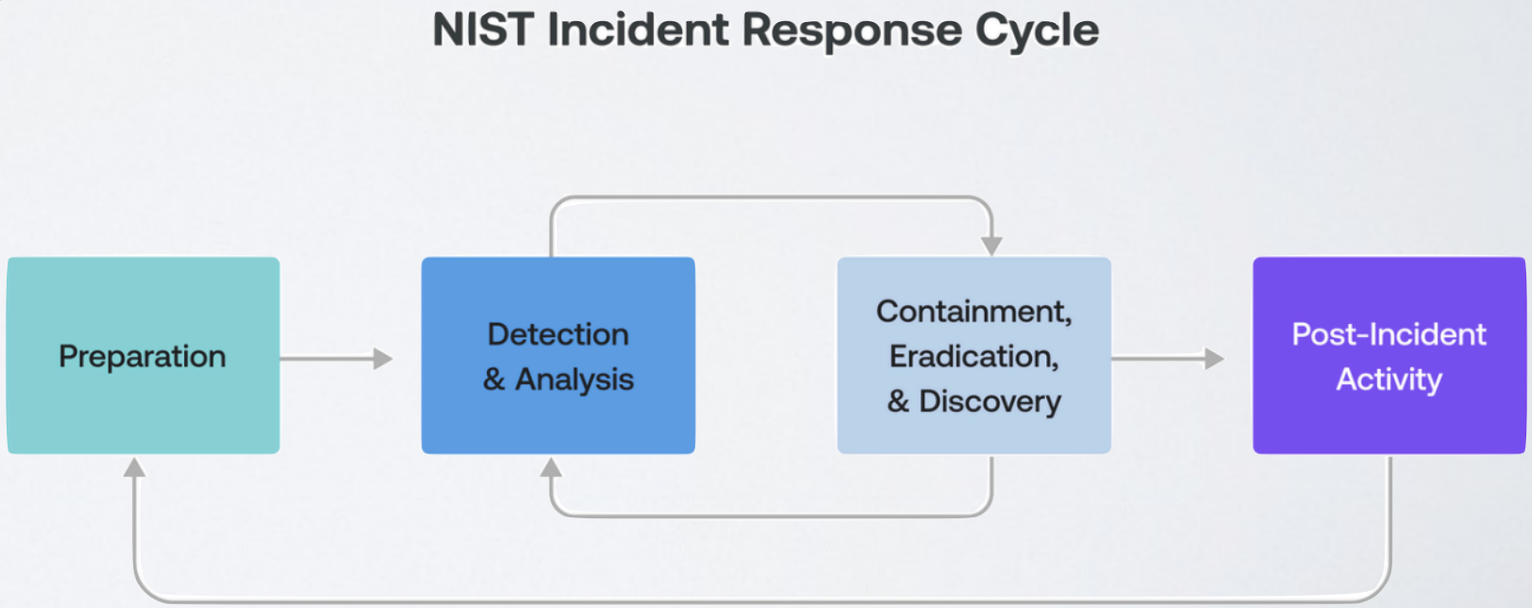
- Understand what exactly happened
- Identify all compromised assets (machines, files, etc.)
- Contain further expansion of the attack
- Eradicate the attack
- Recover efficiently

THE GOAL OF MALWARE ANALYSIS

Dissecting malware to understand

- How it works
  - Know your enemy's strategies
- How to identify it
  - Extract signatures/IoCs (Indicators of compromise)
  - Prevent further similar infections
- How to defeat or eliminate it
  - Rollback (if possible) its effects
  - Clean the infected system

A critical part of incident response!



SIGNATURES

Host-based signatures

- Identify files/processes/registry keys on a victim computer that indicate an infection

Network signatures

- Detect malware by analyzing network traffic
  - Does the malware inspect the surroundings?
  - Does the malware contact an external Command & Control (C&C or C2) server?
- More effective when coupled with malware analysis

Behavioral signatures

- Focus on what the malware does to the system, not the malware itself
  - Behavioral analysis. Different from your standard antivirus signature!

!!!!! WARNING !!!!!

Be conscious about the risks you are going to face

- You will be asked to dissect malicious software that is designed to harm computers (possibly your one)
- You may need to run such malicious software in order to understand how it works.
- Some of the code you'll inspect have been designed by criminals
- Some IPs referenced in some malware may possibly be live. You must assume that behind that IP there is a criminal

!!!!! WARNING !!!!!

Stay safe:

- Keep a safe **offline backup** of your important data
- Always perform analysis on a virtual machine isolated from your main host
- Keep a safe **offline backup** of your important data
- Disconnect the virtual machine from the network unless netcomms are necessary
- Keep a safe **offline backup** of your important data
- If you find references to external IP addresses NEVER EVER try to directly contact them
  - If someone rob your house, and you find the robber home address, would you try to go there to ask him “how did you enter in my house”?
- and…**keep a safe offline backup of your important data**

STATIC V. DYNAMIC ANALYSIS

Static Analysis

- **Examine the malware without running it**
- Tools: strings, PE inspection tools, a disassembler like IDA Pro, etc.

Dynamic Analysis

- **Run the malware and monitor its effects**
- Use a virtual machine and take snapshots
- Tools: RegShot, Process Monitor, Process Hacker, CaptureBAT, debuggers,…
- RAM Analysis: Mandant Redline and Volatility

Other forms of analysis exist that are beyond the scope of this course.

BASIC ANALYSIS

Basic static analysis

- Observe the malware without looking at its code
- Tools: strings, PE inspection, etc.
- Quick and easy but fails for advanced malware and can miss important behaviour

Basic dynamic analysis

- Easy but requires a safe test environment
- Not effective on all malware

ADVANCED ANALYSIS

Advanced static analysis

- Reverse-engineering the malware code with a disassembler.
- Complex, requires understanding of assembly code and how such code interacts with the OS.

Advanced Dynamic Analysis

- Run code in a debugger.
- Examines internal state of a running malicious executable and how such state evolves.

MALWARE ANALYSIS PROCESS

EXPECTED BEHAVIORS

- **Persistence**
  - The malware installs itself in the system
  - It performs actions to guarantee its survivability to
    - reboots
    - inspections
    - etc.
  - Often modifies the OS configuration
  - Changes are hidden

EXPECTED BEHAVIORS

- **Uniqueness**
  - Malware is often incompatible with itself
  - Multiple concurrent executions are meaningless
    - Or sometimes dangerous for the malware itself
    - Think about two ransomware instances competing for the same resources
  - Checks at startup guarantee the execution of a single copy

EXPECTED BEHAVIORS

- **Environment checks / Targeting**
  - Malware is designed to work in appropriate environment
  - Checks at startup if specific execution conditions are met
    - Time/date
    - Localization
    - Availability of specific resources
    - Presence of other processes
    - etc.

EXPECTED BEHAVIORS

- **Obfuscation and evasion**
  - Malware will try to hide its presence and its effects to stay undetected as long as possible
  - It may inject code in other processes to conceal its real identity
  - It may mangle its internal code to hide easily discoverable hints of its malicious nature
  - It may leverage techniques to hamper the possibility to correctly analyse its behaviour

EXPECTED BEHAVIORS

- **Fingerprinting & beaconing**
  - Malware will inspect the infected system and collect information about it
  - Information can be used to “publicize” its presence to a remote server

EXPECTED BEHAVIORS

- **Communication**
  - Malware often interacts with external command & control services to
    - receive instructions and commands
    - update its internal components
    - exfiltrate data
  - Communication may use heterogeneous links types
  - Data on channels is often obfuscated

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 one
- Don't get stuck on a hard issue, move along

Malware authors are constantly raising the bar

BASIC STATIC ANALYSIS

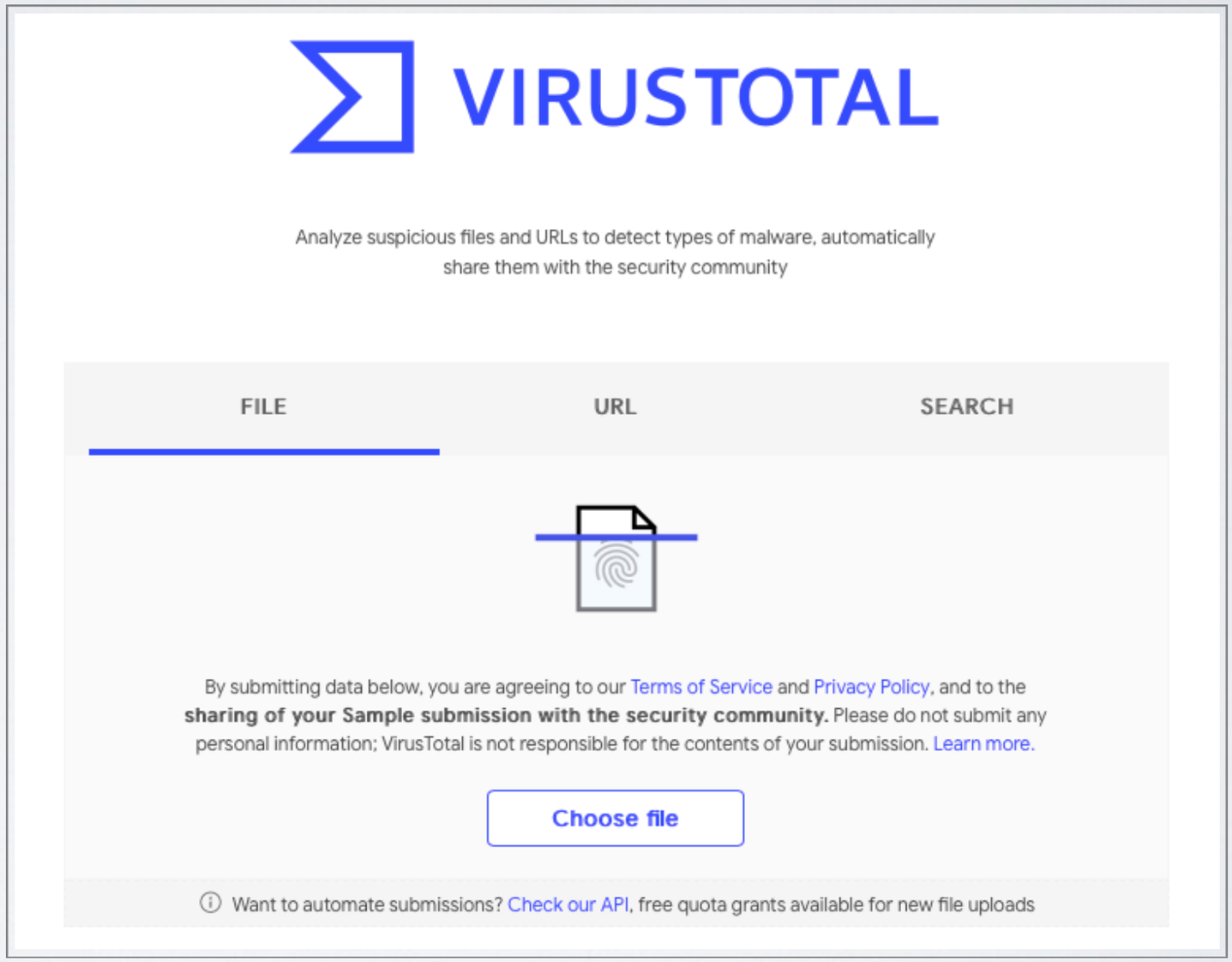
Techniques:

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

AN OBVIOUS 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



HASHING SAMPLES

MD5 or SHA-1

Condenses a file of any size down to a fixed-length fingerprint

Uniquely identifies a file

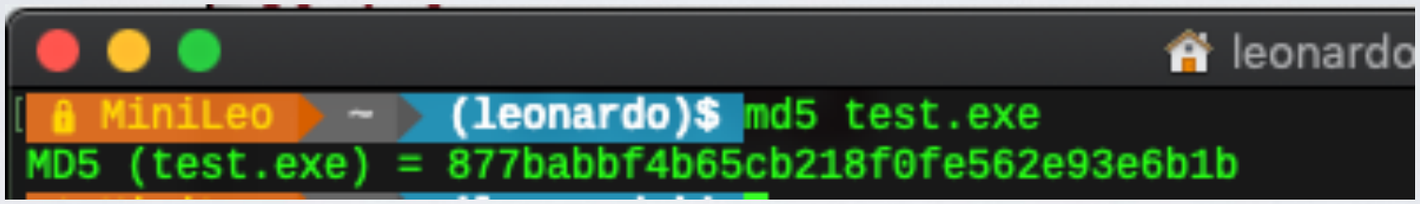
- There are MD5 collisions but they are not common
- Collision: two different files with the same hash

De-facto standard way of uniquely identifying samples

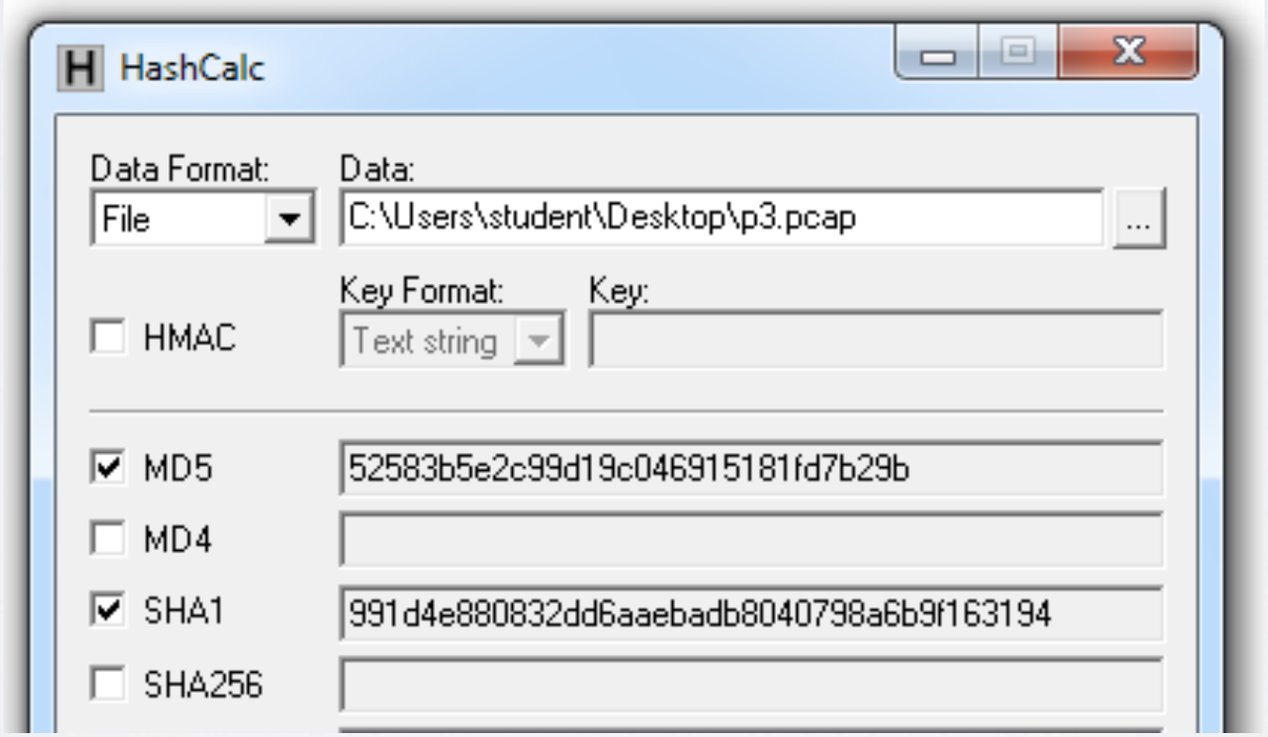
Note: **sample vs. malware**

HASHING STUFF

Most operating systems provide md5 as a standard utility



If you use Windows you may prefer a GUI-wrapped version



HASH USES

- Label a malware sample
- Share the hash with other analysts to identify malware
- Search the hash online to see if someone else has already identified the file

- Interesting alternative: import-hash
  - Calculate hash on the set of imported functions from libraries
  - Useful to cluster similar samples in a single family
  - The idea is that similar versions of a same malware will probably use the same APIs

STRINGS

- Any sequence of printable characters is a **string**
- Strings are terminated by a **null** value (0x00)
- ASCII characters are 8 bits long

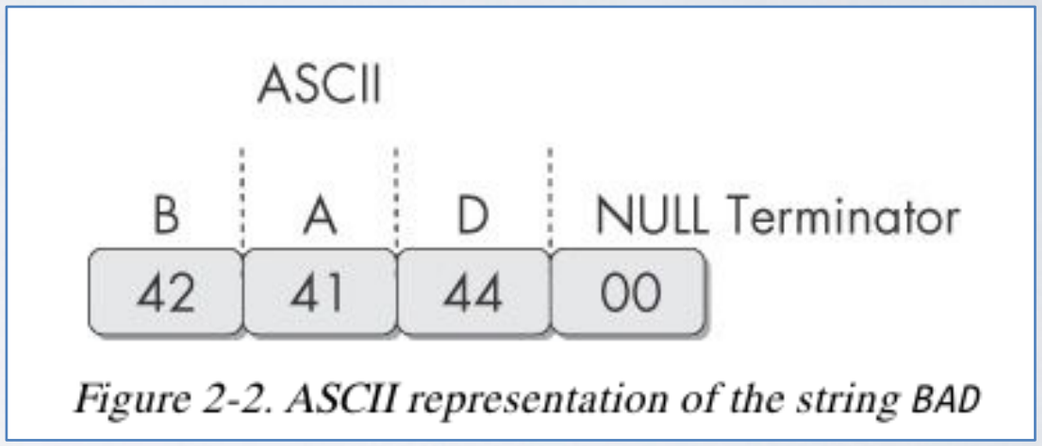


Figure 2-2. ASCII representation of the string BAD

- Unicode characters are 16 bits long
  - Microsoft calls them "wide characters"

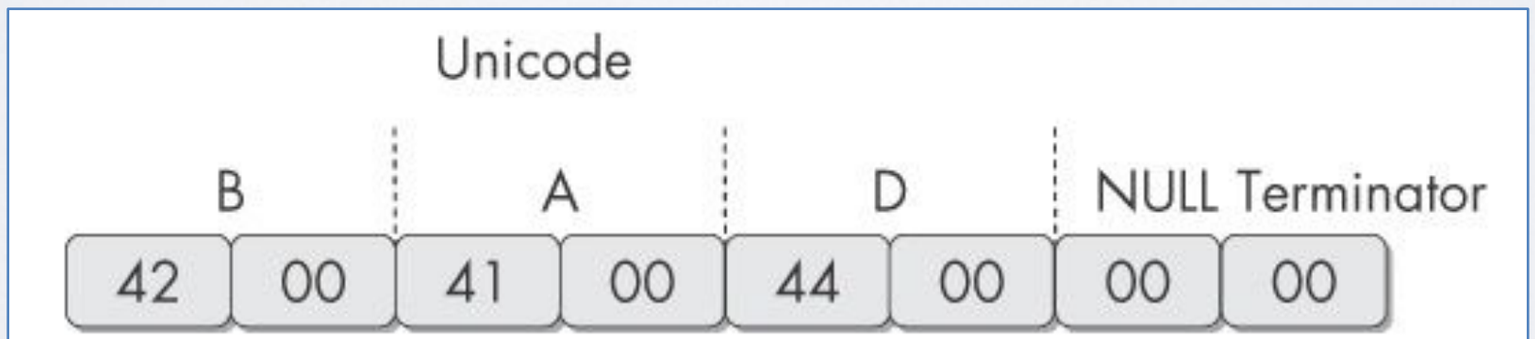


Figure 2-3. Unicode representation of the string BAD

THE STRINGS COMMAND

Native in Linux/OSX, it is also available for Windows

Finds all strings in a file 3 or more characters long

- Bold items can be ignored
- (1,2) GetLayout and SetLayout are Windows functions
- (3) GDI32.DLL is a Dynamic Link Library
- (4) looks like an IP address
- (5) looks like an error msg

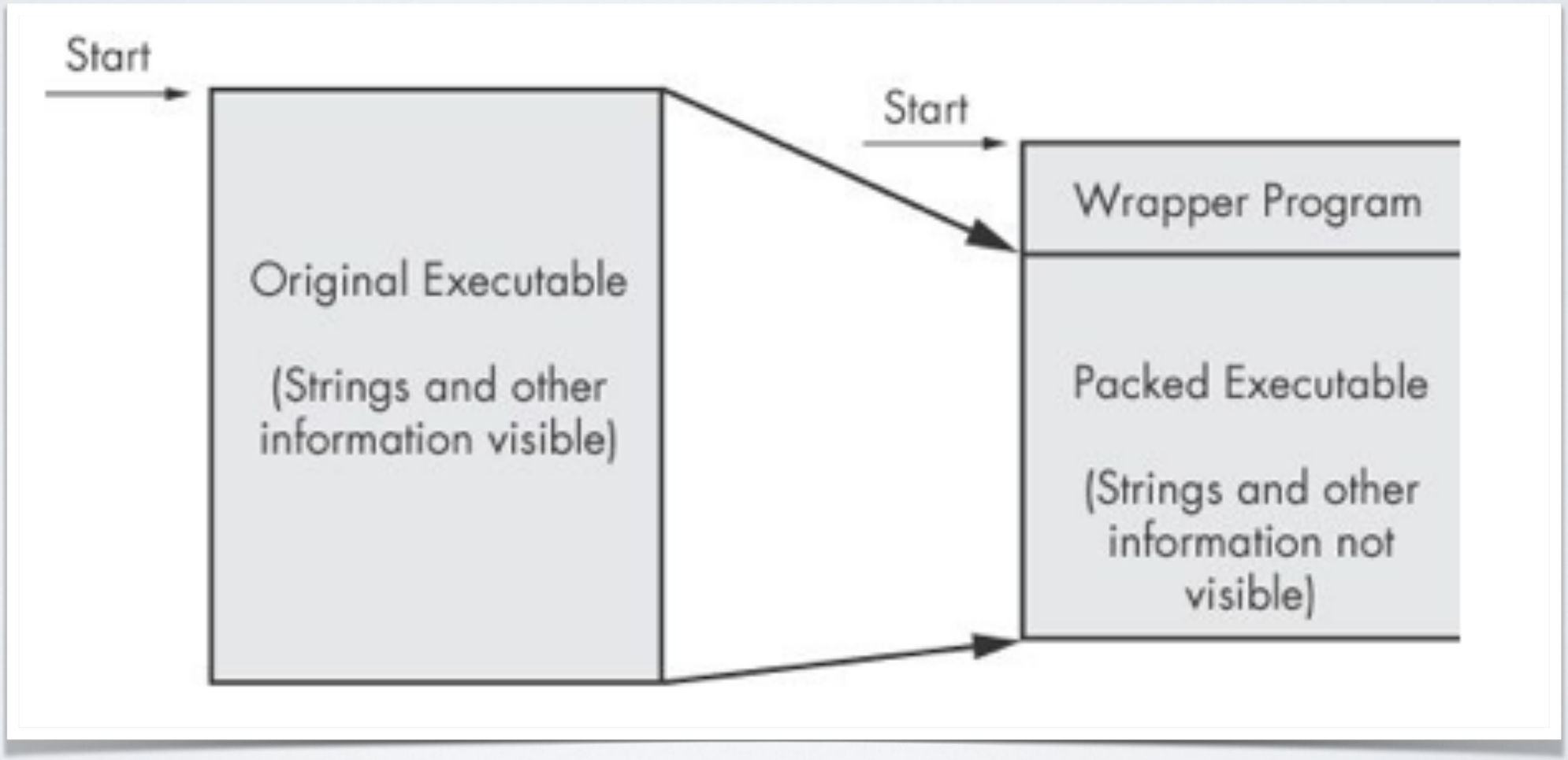
```
C:>strings bp6.ex_
VP3
VW3
t$@
D$4
99.124.22.1 4
e-@
GetLayout 1
GDI32.DLL 3
SetLayout 2
M}C
Mail system DLL is invalid.!Send Mail failed to
send message. 5
```

OBFUSCATION THROUGH PACKING

The code is compressed, like a Zip file

This makes most 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

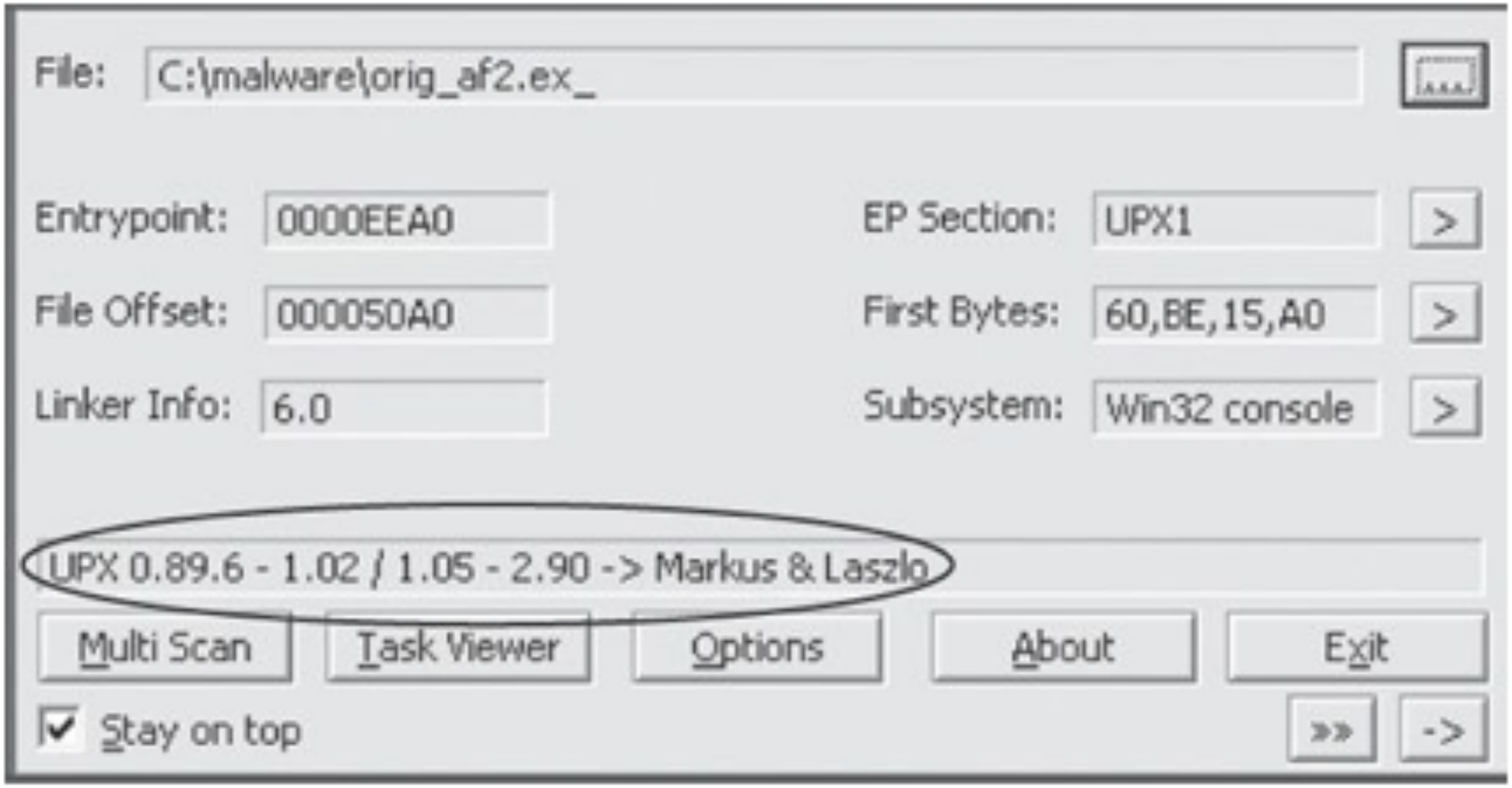


Figure 2-5. The PEiD program

DETECTING PACKERS WITH PEID

Beware!

- Some PEiD plugins may silently run the malware to identify the packer!
- Take care of isolating your testing VM.

FILE EXECUTION IN WINDOWS

What happen when you double click an executable file in Windows?

The OS (actually the loader) looks in the file to understand what to do.

What's in the file?

- Code
- Data
- Other resources
- Metadata

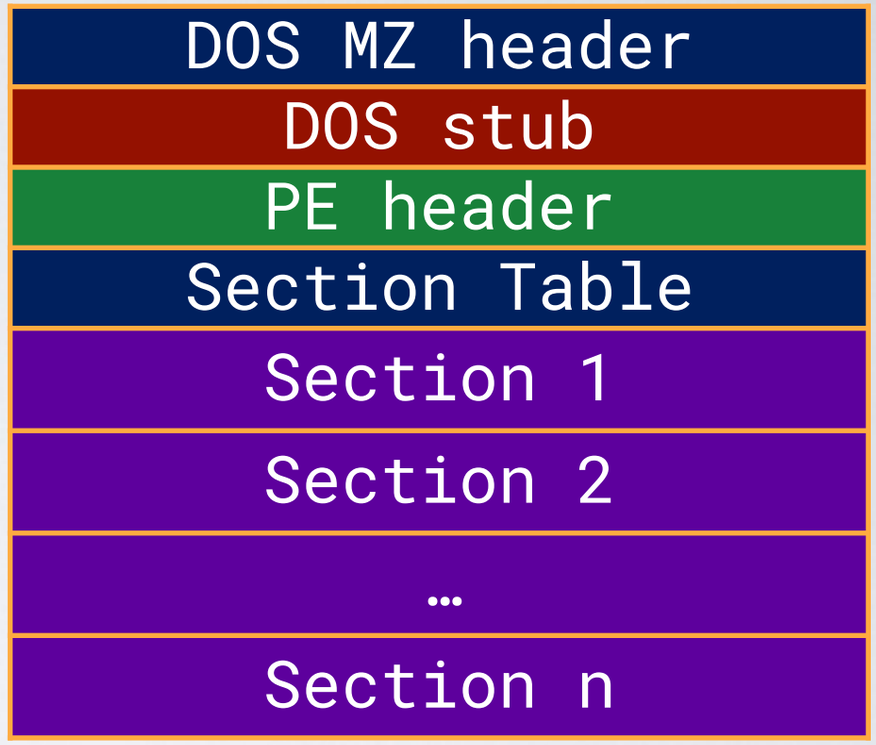
All these information need to be organized to be correctly interpreted by the loader

THE PORTABLE EXECUTION FILE FORMAT

- Used by Windows executable files, object code, and DLLs
  - Very convenient! A single file format for several different usage scenarios
  - Means that you treat EXEs and DLLs in a very similar way
- A data structure that contains the information necessary to Windows for file loading
- Almost every file executed on Windows is in PE format

THE PORTABLE EXECUTION FILE FORMAT

- Starts with MS-DOS header+stub
  - just for compatibility!
- The PE headers contain metadata used to correctly load and execute the file
- File content is structured in “sections”
  - Each section contains a specific kind of data
  - Section table describes the sections
- Usually at least two sections are present



THE PORTABLE EXECUTION FILE FORMAT

The sections that are most commonly present in an executable are:

- Executable Code Section, named **.text** (Microsoft) or CODE (Borland)
- Data Sections, named .data, **.rdata**, or .bss (Microsoft) or DATA (Borland)
- Resources Section, named **.rsrc**
- Export Data Section, named .edata
- Import Data Section, named .idata
- Debug Information Section, named .debug

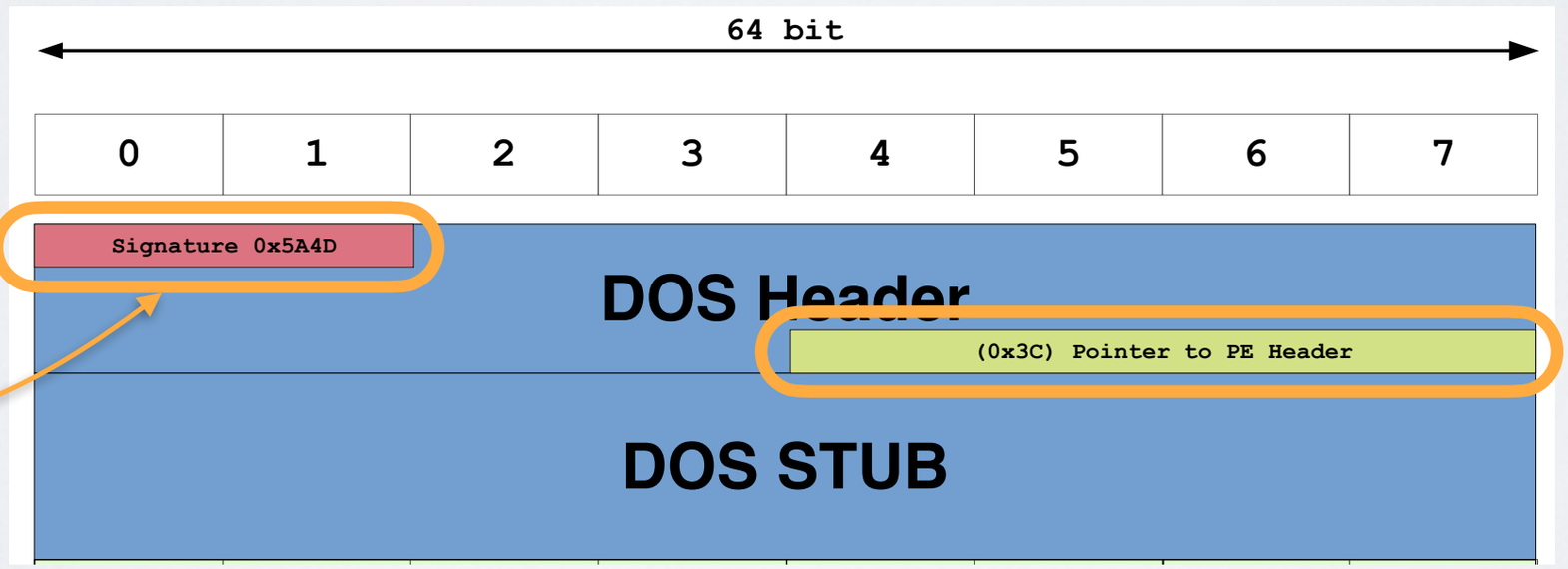
The names are actually irrelevant as they are ignored by the OS and are present only for the convenience of the programmer.

- Means you can find very different names (e.g. packers use their names)

DOS HEADER + STUB

- The DOS header occupies the first 64 bytes of the file.
- It's there in case the program is run from DOS, so DOS can recognise it as a valid executable and run the DOS stub
- The DOS stub usually just prints a string like "This program must be run under Microsoft Windows"

DOS "magic" signature 5A4Dh="MZ"



e_lfanew points to the PE header

DOS HEADER + STUB

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00 ; MZP.........ÿÿ..
00000010h: B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00 ; ¸.......@.......
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 ; ................
00000040h: BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90 ; º....´.Í!¸.LÍ!□□
00000050h: 54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73 ; This program mus
00000060h: 74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57 ; t be run under W
00000070h: 69 6E 33 32 0D 0A 24 37 00 00 00 00 00 00 00 00 ; in32..$7........
00000080h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000090h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000000a0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000000b0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000000c0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000000d0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000000e0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
000000f0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000100h: 50 45 00 00 4C 01 08 00 19 5E 42 2A 00 00 00 00 ; PE..L....^B*....
00000110h: 00 00 00 00 E0 00 8E 81 0B 01 02 19 00 A0 02 00 ; ....à.Ž□..... ..
```

DOS header signature in e_magic

e_lfanew contains offset of PE header (NB bytes in reverse order: 00 00 01 00)

DOS stub

PE signature at start of PE header

PE HEADER

Signature whose values is always 50450000h= “PE”, 4 bytes

COFF FileHeader, 20 bytes

OptionalHeader, 224 bytes
- Headers, 96 bytes
- DataDirectory, 128 bytes

64 bit

0	1	2	3	4	5	6	7
Signature 0x50450000				Machine		#NumberOfSections	
TimeDateStamp				PointerToSymbolTable (deprecated)			
# NumberOfSymbolTable				SizeOfOptionalHeader		Characteristics	
Magic		MajorLinker Version	MinorLinker Version	SizeOfCode (sum of all sections)			
SizeOfInitializedData				SizeOfUninitializedData			
AddressOfEntryPoint (RVA)				BaseOfCode (RVA)			
BaseOfData (RVA)				ImageBase			
SectionAlignment				FileAlignment			
MajorOperating SystemVersion		MinorOperating SystemVersion		MajorImage Version		MinorImage Version	
MajorSubsystem Version		MinorSubsystem Version		Win32VersionValue (zeros filled)			
SizeOfImage				SizeOfHeaders			
CheckSum (images not checked)				Subsystem		DllCharacteristics	
SizeOfStackReserve				SizeOfStackCommit			
SizeOfHeapReserve				SizeOfHeapCommit			
LoaderFlags (zeros filled)				# NumberOfRvaAndSizes			
ExportTable (RVA)				SizeOfExportTable			
ImportTable							

PE HEADER - SIGNATURE + FILE HEADER

64 bit

0	1	2	3	4	5	6	7
Signature 0x50450000				Machine		#NumberOfSections	
TimeDateStamp				PointerToSymbolTable (deprecated)			
# NumberOfSymbolTable (deprecated)				SizeOfOptionalHeader		Characteristics	

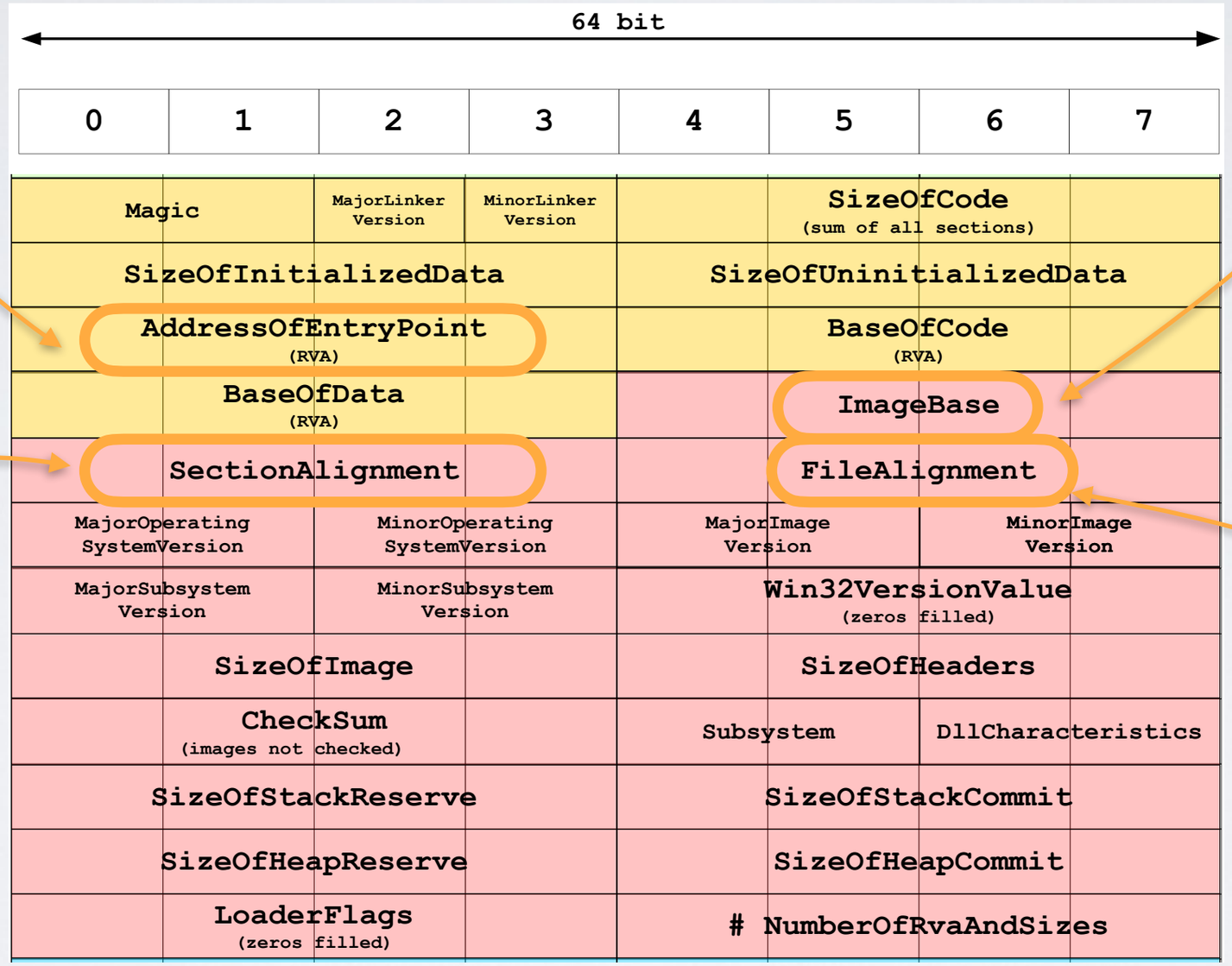
How many sections does this file contain?

Flags which dictate for instance whether this PE file is an executable or a DLL

PE HEADER - OPTIONAL HEADERS

RVA of the first instruction to be executed

Preferred load address for the file in memory. 99% of cases it is 400000h



Granularity of sections alignment in memory. E.g. if value is 4096 (1000h), each section must start at multiples of 4096 bytes

Granularity of sections alignment in file. E.g. if value is 512 (200h), each section must start at multiples of 512 bytes

PE HEADER - OPTIONAL HEADERS

Overall size of the PE image in memory. It's the sum of all headers and sections aligned to SectionAlignment.

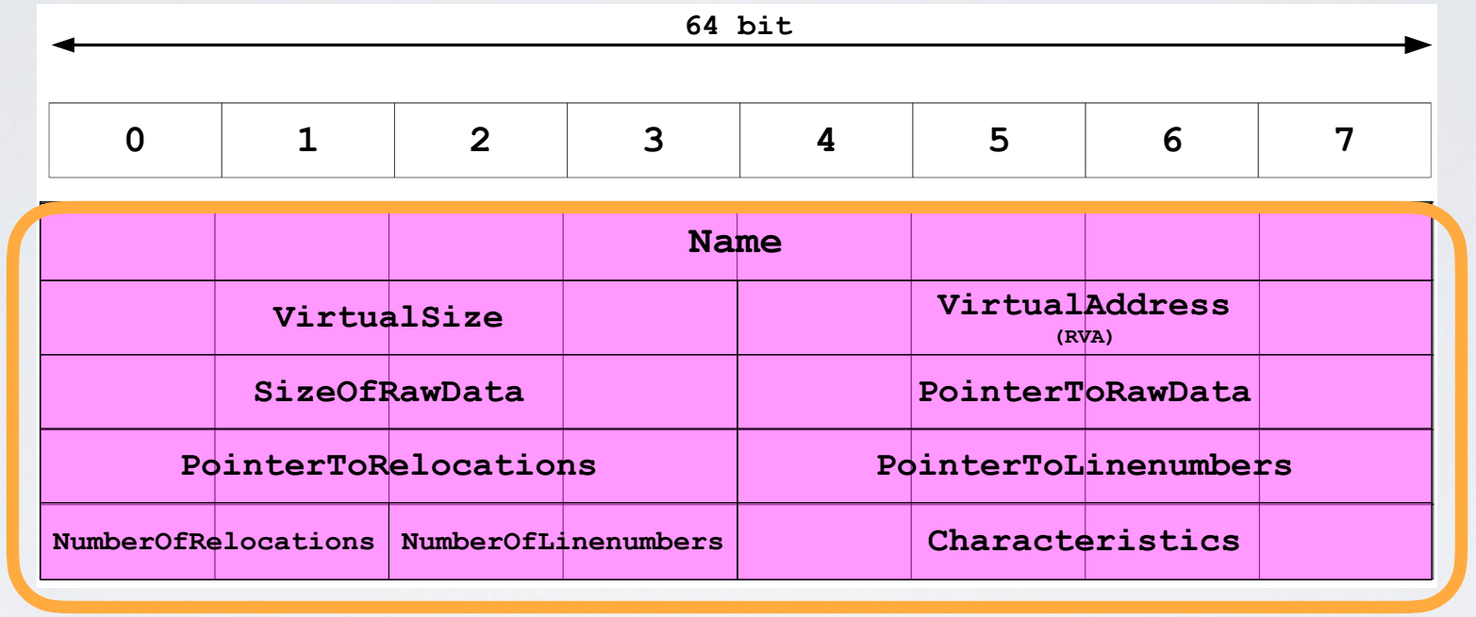
64 bit

0	1	2	3	4	5	6	7
Magic		MajorLinker Version	MinorLinker Version		SizeOfCode (sum of all sections)		
SizeOfInitializedData				SizeOfUninitializedData			
AddressOfEntryPoint (RVA)					BaseOfCode (RVA)		
BaseOfData (RVA)					ImageBase		
SectionAlignment					FileAlignment		
MajorOperating SystemVersion		MinorOperating SystemVersion		MajorImage Version		MinorImage Version	
MajorSubsystem Version		MinorSubsystem Version			Win32VersionValue (zeros filled)		
SizeOfImage					SizeOfHeaders		
CheckSum (images not checked)					Subsystem		DllCharacteristics
SizeOfStackReserve					SizeOfStackCommit		
SizeOfHeapReserve					SizeOfHeapCommit		
LoaderFlags (zeros filled)					# NumberOfRvaAndSizes		

The size of all headers + section table. You can also use this value as the file offset of the first section in the PE file.

PE HEADER - SECTION TABLE

One of these (40 bytes) for each section defined in the file

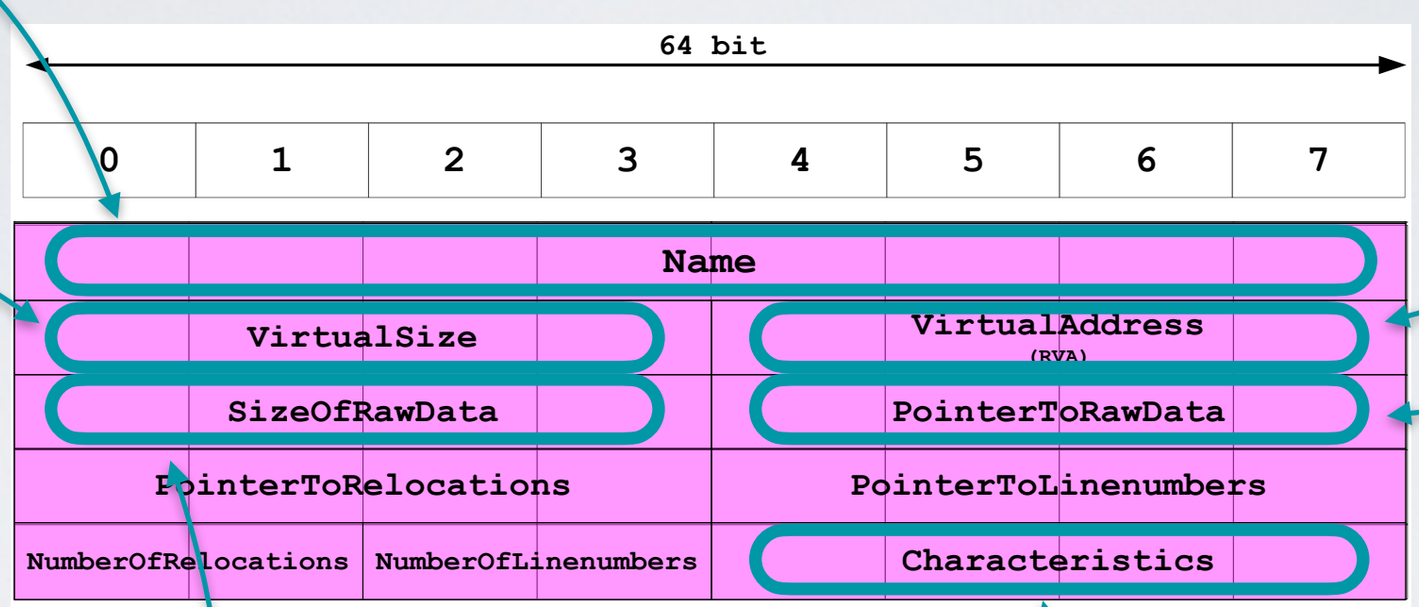


PE HEADER - SECTION TABLE

Section name without terminating char

Size of the section once loaded in memory (how much space the loader will allocate for it)

The size of the section's data in the file on disk



RVA of the section

Offset from the file's beginning to the section's data

Contains flags such as whether this section contains executable code, initialized data, uninitialized data, can it be written to or read from.

PE HEADER - SECTIONS

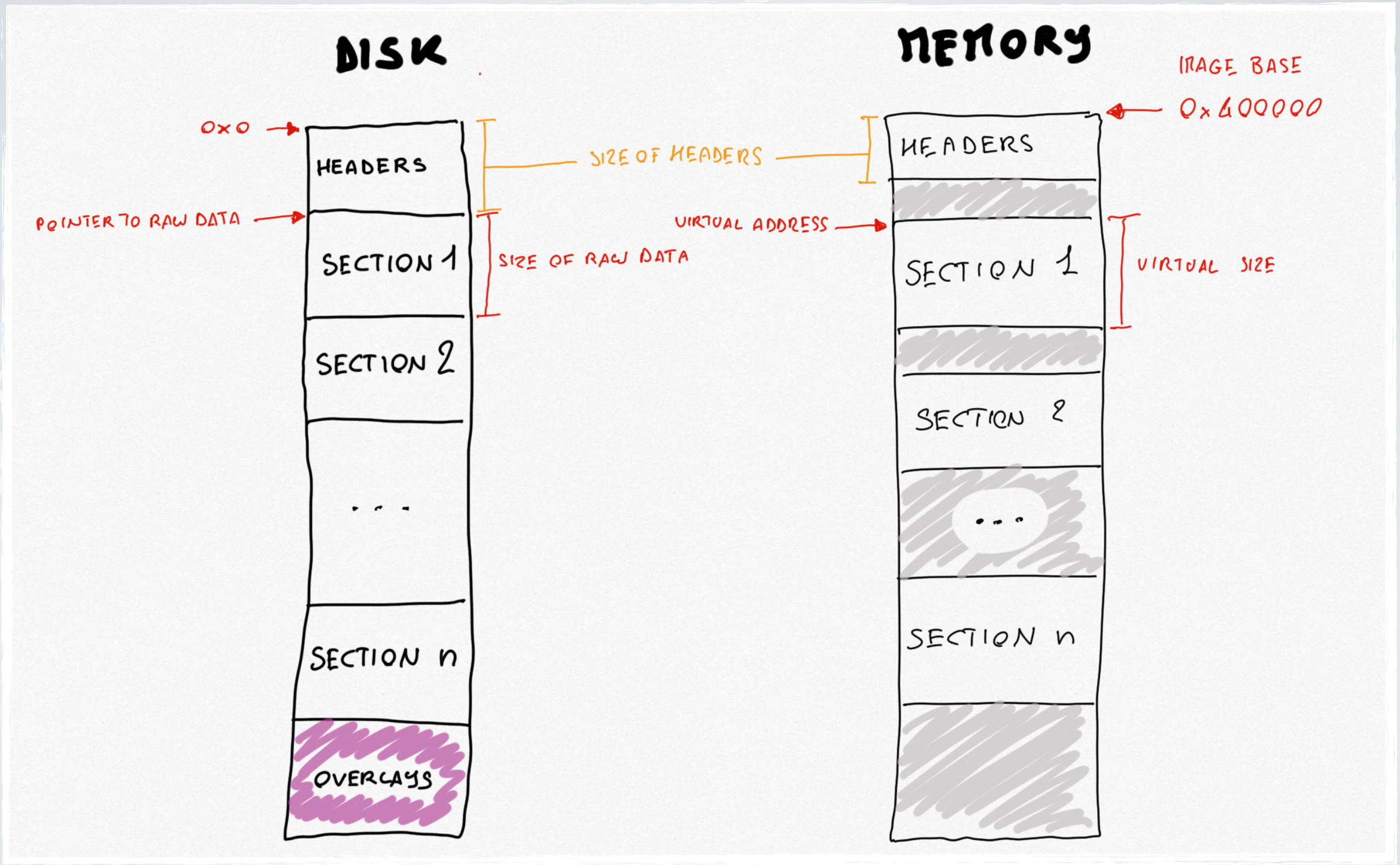
After the section headers we find the sections themselves.

- In the file on disk, each section starts at an offset that is some multiple of the FileAlignment value found in OptionalHeader.
- Between each section's data there will be 00 byte padding.

When loaded into RAM, the sections always start on a page boundary so that the first byte of each section corresponds to a memory page.

- On x86 CPUs pages are 4kB aligned, whilst on IA-64, they are 8kB aligned.
- This alignment value is stored in SectionAlignment also in OptionalHeader.

LOADING IN MEMORY



CLUES IN LIBRARIES

- The PE header lists libraries and functions that will be loaded
- Their names can reveal what the program does
- e.g. URLDownloadToFile indicates that the program downloads something

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

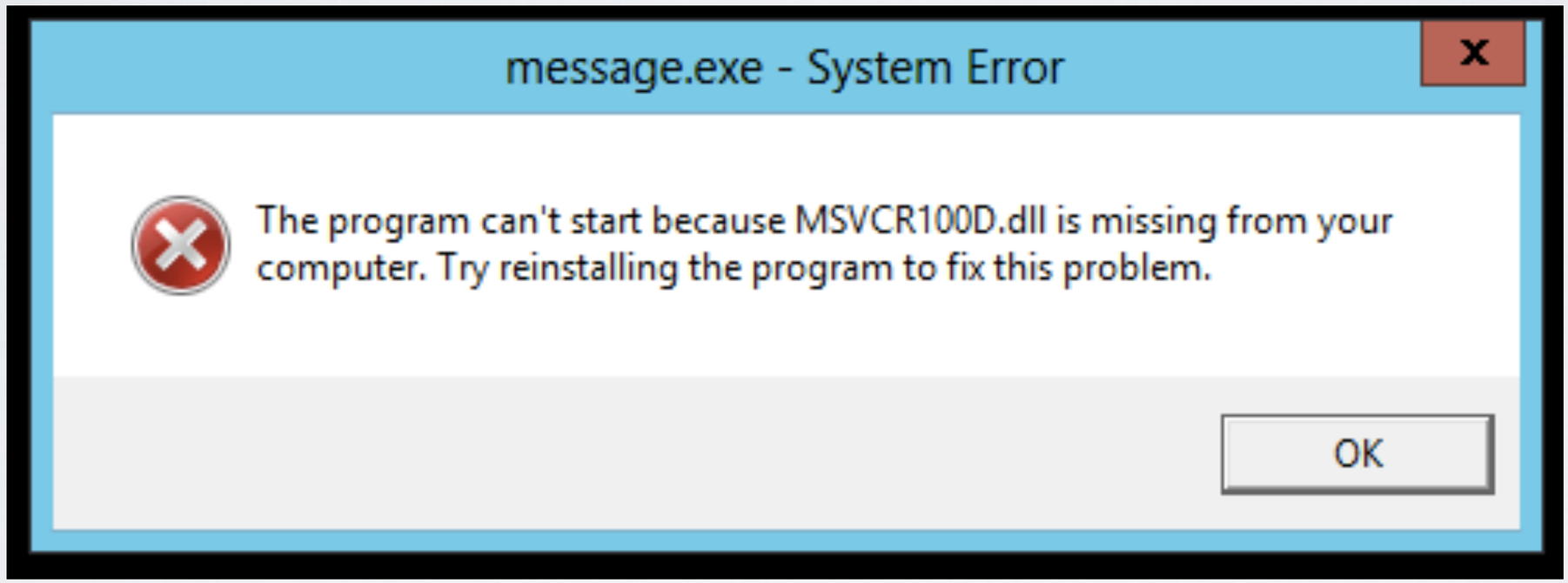
- Statically
- Dynamically
- At Runtime

STATIC LINKING

- Rarely used for Windows executables
- More common in Unix/Linux
- All code from the library is copied into the executable
- Makes executable large in size
- The main way to go if you want to have self-contained code

DYNAMIC LINKING

- Most common method
- Host OSs search for necessary libraries when the program is loaded



RUNTIME LINKING

- Unpopular in friendly programs
- Common in malware, especially packed or obfuscated malware
- But also used by software that allows dynamic loading of plugins
- Connect to libraries only when needed, not when the program starts
- Most commonly done with the **LoadLibrary** and **GetProcAddress** functions

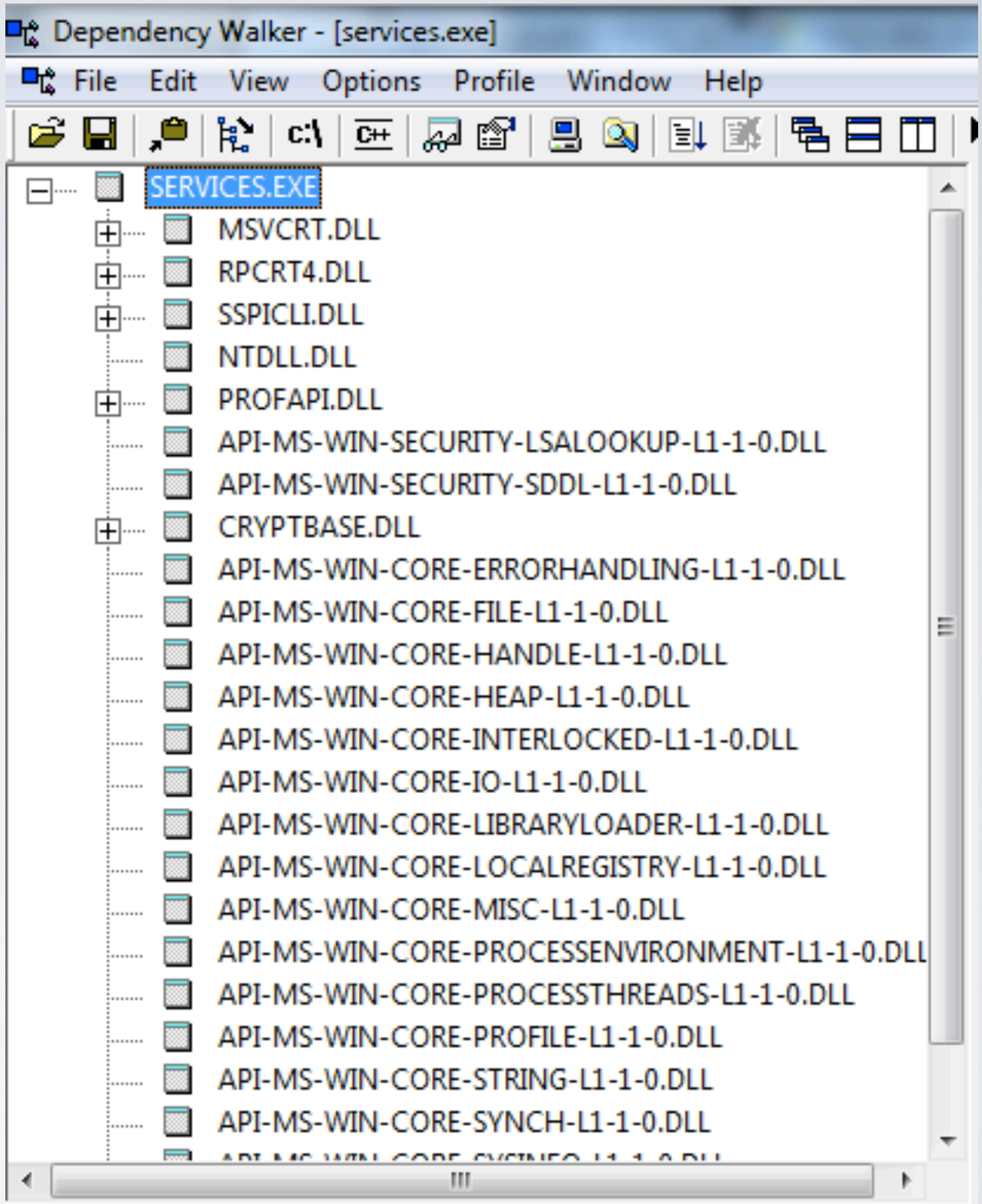
DEPENDENCY WALKER

Shows Dynamically Linked Functions

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

SERVICES.EXE

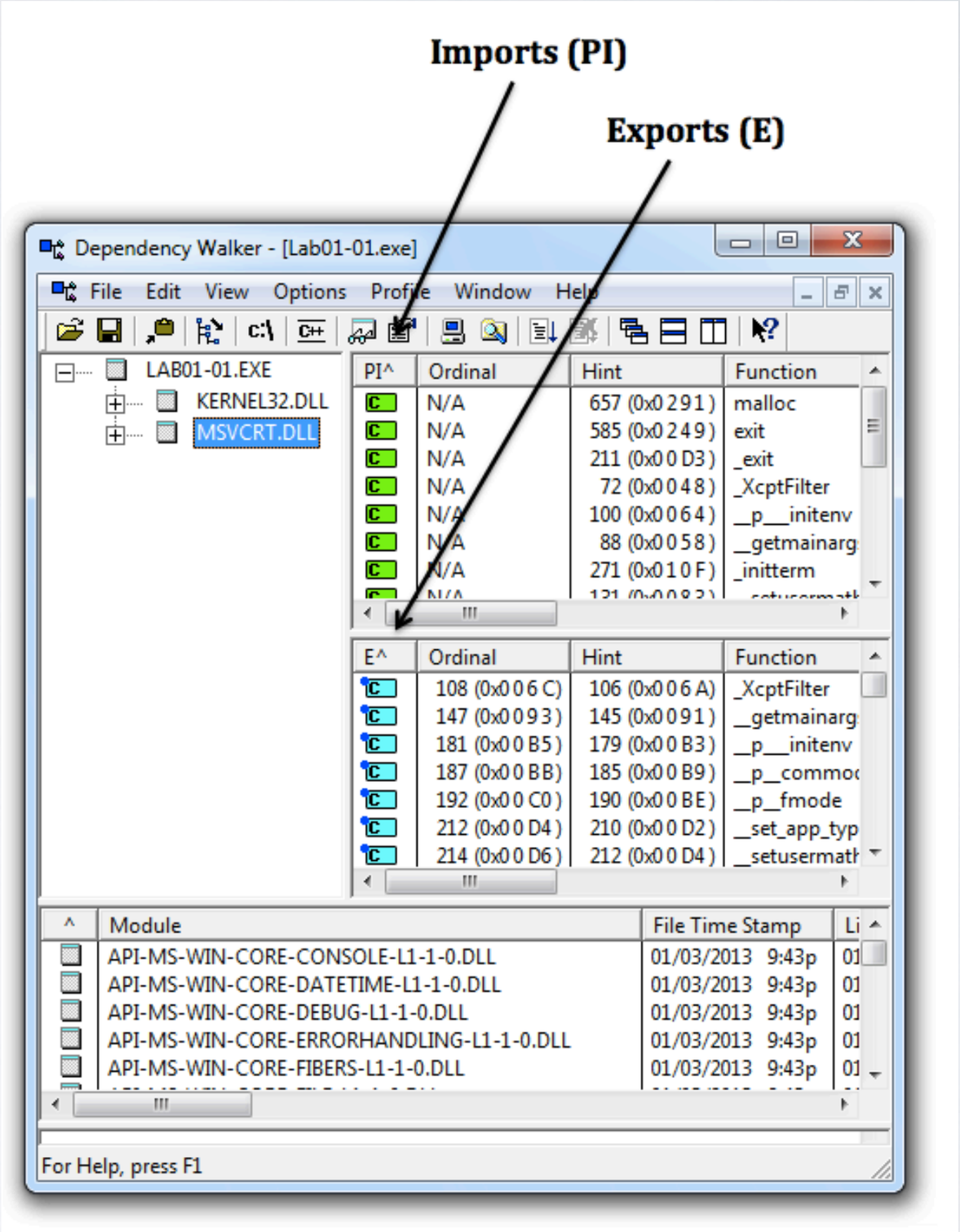
- Example



SERVICES.EX_ (MALWARE)

DEPENDENCY WALKER

- Imports & Exports



COMMON DLLS

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.

COMMON DLLS

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.

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	

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

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

EXPORTS

- DLLs **export** functions
- EXEs **import** functions
- Both exports and imports are listed in the PE header
- Exports should be rare in EXEs (but reality is sometimes different)

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

RESOURCE HACKER

- Lets you browse the **.rsrc** section
- Strings, icons, and menus