

# Module 1

A journey from high level languages, through assembly, to the running process

[https://github.com/hasherezade/malware\\_training\\_voll](https://github.com/hasherezade/malware_training_voll)

# Basics of PE (Portable Executable)





# Basics of a PE file

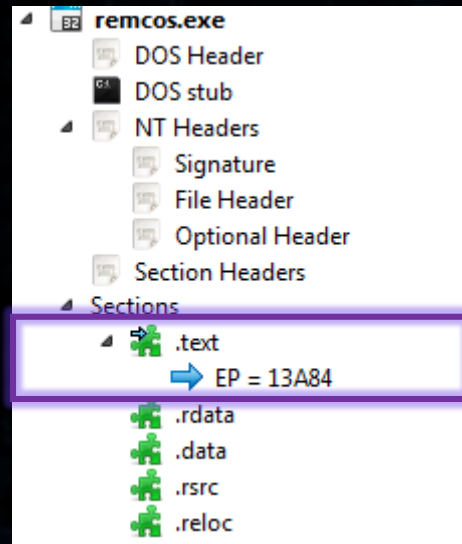
- PE (Portable Executable) is a native executable format on Windows
- PE files:
  - user mode: EXE, DLL
  - kernel mode: driver (.sys), kernel image (ntoskrnl.exe)
  - UEFI (run in SMM – System Management Mode)
  - Also OBJ files have structures similar to PE



# Basics of a PE file

- PE (Portable Executable) contains information:
  - What to execute: the **compiled code**
  - How to execute: **headers** with data necessary for loading it

remcos.exe



	Hex	Disasm
413A84	55	PUSH EBP
413A85	8BEC	MOV EBP, ESP
413A87	6AFF	PUSH -1
413A89	68805F4100	PUSH 0X415F08
413A8E	68103C4100	PUSH 0X413C10
413A93	64A100000000	MOV EAX, DWORD PTR FS:[0]
413A99	50	PUSH EAX
413A9A	64892500000000	MOV DWORD PTR FS:[0], ESP
413AA1	83EC68	SUB ESP, 0X68
413AA4	53	PUSH EBX

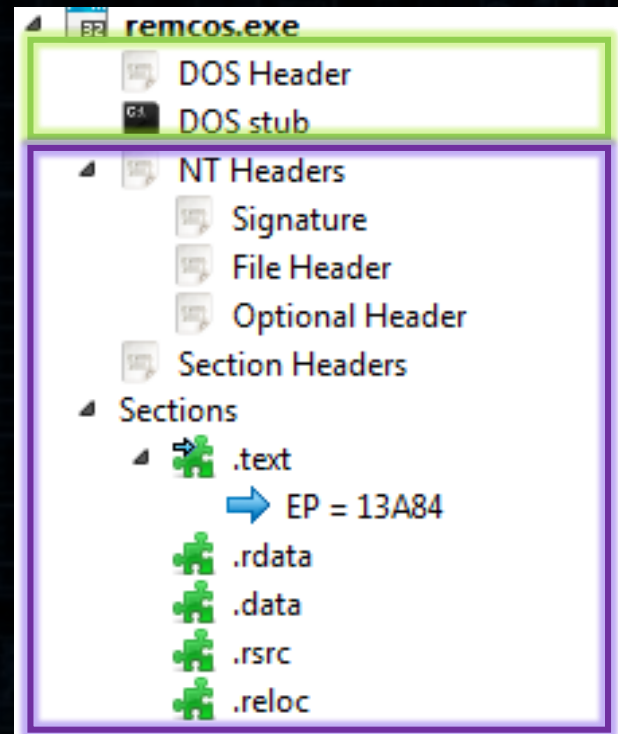


# Basics of a PE file

- PE format is based on a Unix format COFF – that was used in VAX/VMS
- It was introduced as a part of specification Win32
- Throughout many years, the core of the format didn't change, only some new fields of some structures have been added
- Since introduction of 64 bit environment, PE needed to be adjusted to it: 64 bit PE was introduced
- Also, new variants have been introduced, like .NET PE – containing additional structures with intermediate code and metadata

# Basics of a PE file

- PE file structure: the DOS part (legacy) and the Windows Part





# Basics of a PE file

- DOS Header: only e\_magic, and e\_lfanew must be filled:

```
typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE header
    WORD   e_magic;                      // Magic number -----> „MZ”
    WORD   e_cblp;                       // Bytes on last page of file
    WORD   e_cp;                          // Pages in file
    WORD   e_crlc;                       // Relocations
    WORD   e_cparhdr;                    // Size of header in paragraphs
    WORD   e_minalloc;                   // Minimum extra paragraphs needed
    WORD   e_maxalloc;                   // Maximum extra paragraphs needed
    WORD   e_ss;                         // Initial (relative) SS value
    WORD   e_sp;                         // Initial SP value
    WORD   e_csum;                       // Checksum
    WORD   e_ip;                         // Initial IP value
    WORD   e_cs;                         // Initial (relative) CS value
    WORD   e_lfarlc;                     // File address of relocation table
    WORD   e_ovno;                       // Overlay number
    WORD   e_res[4];                     // Reserved words
    WORD   e_oemid;                      // OEM identifier (for e_oeminfo)
    WORD   e_oeminfo;                   // OEM information; e_oemid specific
    WORD   e_res2[10];                  // Reserved words
    LONG   e_lfanew;                    // File address of new exe header -----> Points to the NT header
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

# Basics of a PE file

- PE sections

- PE is divided into sections with different permissions
- Sections introduce a logical layout of the binary, that compilers/linkers can follow
- Dividing PE on section improves security: the code is isolated from the data
- HOWEVER:
  - if DEP is disabled, page without execution permission can still be executed
  - The section containing the Entry Point will always be treated as executable



# Basics of a PE file

- PE sections are defined by sections header

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
▣ .text	1000	13000	1000	12D26	60000020	0	0	0
>	14000	^	13D26	^	r-x			
▣ .rdata	14000	6000	14000	5490	40000040	0	0	0
>	1A000	^	19490	^	r--			
▣ .data	1A000	1000	1A000	114C	C0000040	0	0	0
>	1B000	^	1B14C	^	rw-			
▣ .rsrc	1B000	1000	1C000	B80	40000040	0	0	0
>	1C000	^	1CB80	^	r--			
▣ .reloc	1C000	3000	1D000	268A	42000040	0	0	0
>	1F000	^	1F68A	^	r--			

# Basics of a PE file

- PE sections

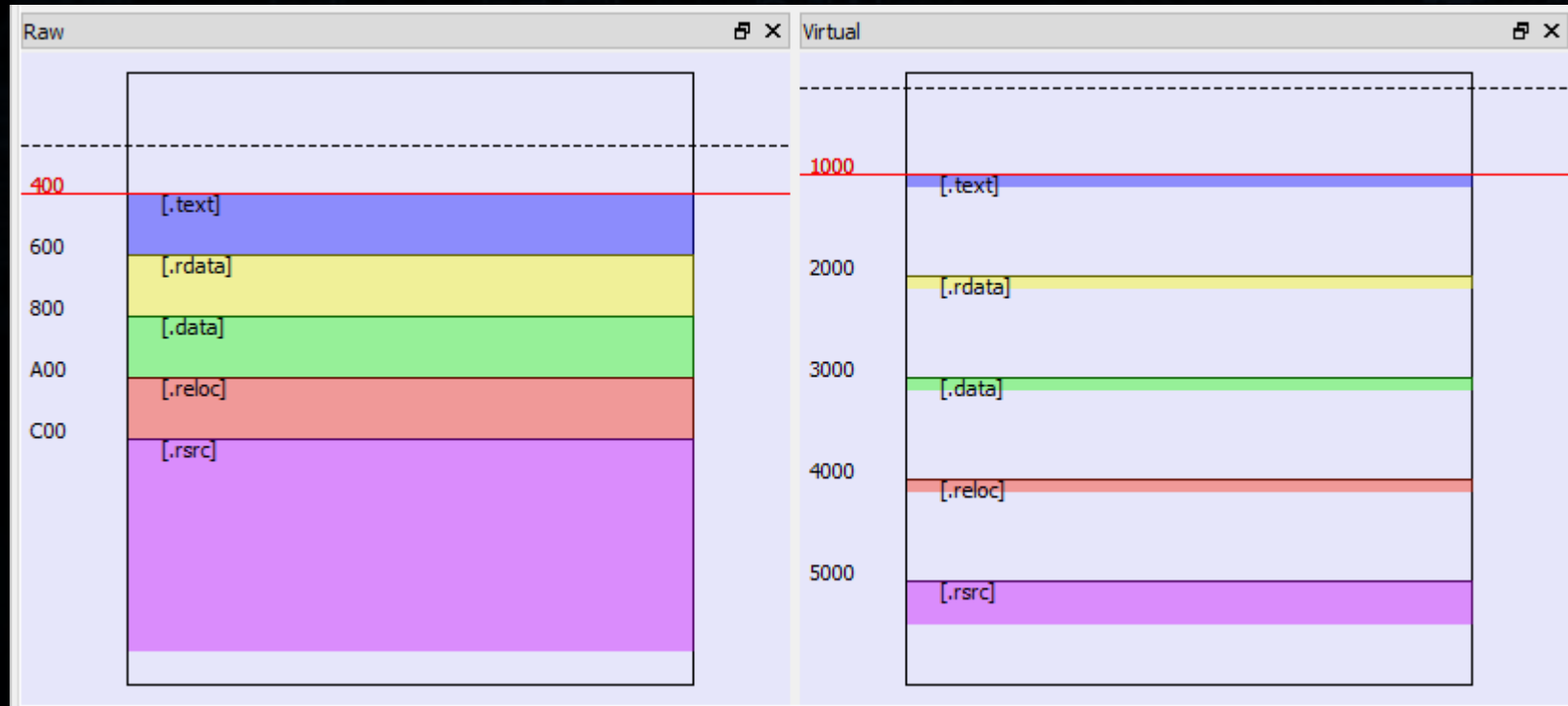
- on the disk PE is stored in a raw format (the unit is defined by File Alignment)
- In memory PE is mapped to its virtual format (the unit is defined by Section Alignment) – usually of the granularity of one page (0x1000)

Disasm: .rdata	General	DOS Hdr	Rich Hdr	File Hdr	Optional Hdr
Offset	Name	Value		Value	
110	Entry Point	47A3			
114	Base of Code	1000			
118	Base of Data	1D000			
11C	Image Base	400000			
120	Section Alignment	1000			
124	File Alignment	200			



# Basics of a PE file

- Raw (file on the disk), Virtual (mapped in the memory)



# Basics of a PE file

- The most information lies in data directories

	Data Directory	Address	Size
F8	Export Directory	294000	1E62
100	Import Directory	296000	3600
108	Resource Directory	29C000	4B134
110	Exception Directory	0	0
118	Security Directory	0	0
120	Base Relocation Table	2E8000	16ED0
128	Debug Directory	0	0
130	Architecture Specific Data	0	0
138	RVA of GlobalPtr	0	0
140	TLS Directory	29B000	18
148	Load Configuration Directory	0	0
150	Bound Import Directory in headers	0	0
158	Import Address Table	2968D4	7A8
160	Delay Load Import Descriptors	0	0
168	.NET header	0	0



# Basics of a PE file: Relocation

- Relocation Table

Disasm: .rdata

General

DOS Hdr

Rich Hdr

File Hdr

Optional Hdr

Section Hdrs

Imports

Resources

BaseReloc.

Offset	Page RVA	Block Size	Entries Count
24A00	1000	94	46
24A94	2000	74	36
24B08	3000	70	34
24B78	4000	11C	8A
24C94	5000	A0	4C
24D24	6000	74	26

Relocation Block [ 70 entries ]

Offset	Value	Type	Offset from Page	Reloc RVA
24A08	300A	32 bit field	A	100A
24A0A	3043	32 bit field	43	1043
24A0C	3055	32 bit field	55	1055
24A0E	305B	32 bit field	5B	105B
24A10	30C7	32 bit field	C7	10C7
24A12	30F6	32 bit field	F6	10F6
24A14	3119	32 bit field	119	1119
24A16	3153	32 bit field	153	1153
24A18	318A	32 bit field	18A	118A
24A1A	31D6	32 bit field	1D6	11D6

# Basics of a PE file: Relocation

1. PE comes with some default base address in the header
2. All the absolute addresses inside the PE code assume that the PE was loaded at this base

The screenshot displays a disassembler interface with two main panes. The left pane shows the PE header fields, and the right pane shows the disassembled code. Annotations highlight the base address and a specific relocation entry.

**PE Header Fields (Left Pane):**

Offset	Name	Value
100	Magic	10B
102	Linker Ver. (Major)	A
103	Linker Ver. (Minor)	0
104	Size of Code	6C800
108	Size of Initialized Data	39200
10C	Size of Uninitialized Data	0
110	Entry Point	52D0
114	Base of Code	10B
118	Base of Data	5E000
11C	Image Base	400000

**Assembly Code (Right Pane):**

Address	Hex	Disasm
45E977	E8E84AFFFF	CALL 0X453464
45E97C	8BF0	MOV ESI, EAX
45E97E	FF1540E04600	CALL DWORD PTR [0X46E040] [KERNEL32.dll].GetLastError
45E984	50	PUSH EAX
45E985	E8984AFFFF	CALL 0X453422

**Annotations:**

- A green box points to the **Image Base** field (400000) in the header, with the text: **Base Address = 400000**.
- A green box points to the relocation entry `[0X46E040]` in the assembly code, with the text: **46E040 = 400000 + 6E040**.



# Basics of a PE file: Relocation

- In the past EXEs were usually loaded at their default base (only DLLs didn't have to)
- Nowadays most PEs load at a dynamic base too (due to ASLR)
- A flag in the header determines if a dynamic base will be used

Disasm: .text	General	DOS Hdr	Rich Hdr	File Hdr	Optional Hdr	Section H
Offset	Name	Value	Value			
140	Checksum	AF742				
144	Subsystem	3	Windows cons			
146	DLL Characteristics	8140				
		40	DLL can move			
		100	Image is NX compatible			
		8000	TerminalServer aware			

**DLL Characteristics: DLL can move**

# Basics of a PE file: Relocation

- If the PE was loaded at the **different base** than the one defined in the header, all its fields that were using **absolute addresses** must be recalculated (**rebased**)

	Hex	Disasm
45E977	E8E84AFFFF	CALL 0X453464
	8BF0	MOV ESI, EAX
	FF15 40E04600	CALL DWORD PTR [0X46E040] [KERNEL32.dll].GetLastError
45E984	50	PUSH EAX
45E985	E8984AFFFF	CALL 0X453422

**46E040 = 400000 + 6E040**

Address	Offset	Section Name
002B0000	00001000	
002C0000	00001000	pe-sieve32.exe
002C1000	00006000	".text"
0032E000	00010000	".rdata"
0033E000	00006000	".data"
00344000	00022000	".rsrc"
00366000	00006000	".reloc"

**Load base = 2C0000**

Address	Offset	Disasm
0031E977	E8 E84AFFFF	call pe-sieve32.313464
	8BF0	mov esi,eax
	FF15 40E03200	call dword ptr ds:[<&GetLastError>]
	50	push eax
0031E985	E8 984AFFFF	call pe-sieve32.313422

**32E040 = 2C0000 + 6E040**



# Basics of a PE file: Relocation

- How does PE know **where** are the fields that needs to be **rebased**?

# Basics of a PE file: Relocation

- How does PE know **where** are the fields that needs to be **rebased**?
- They are listed in the **Relocation Table!**



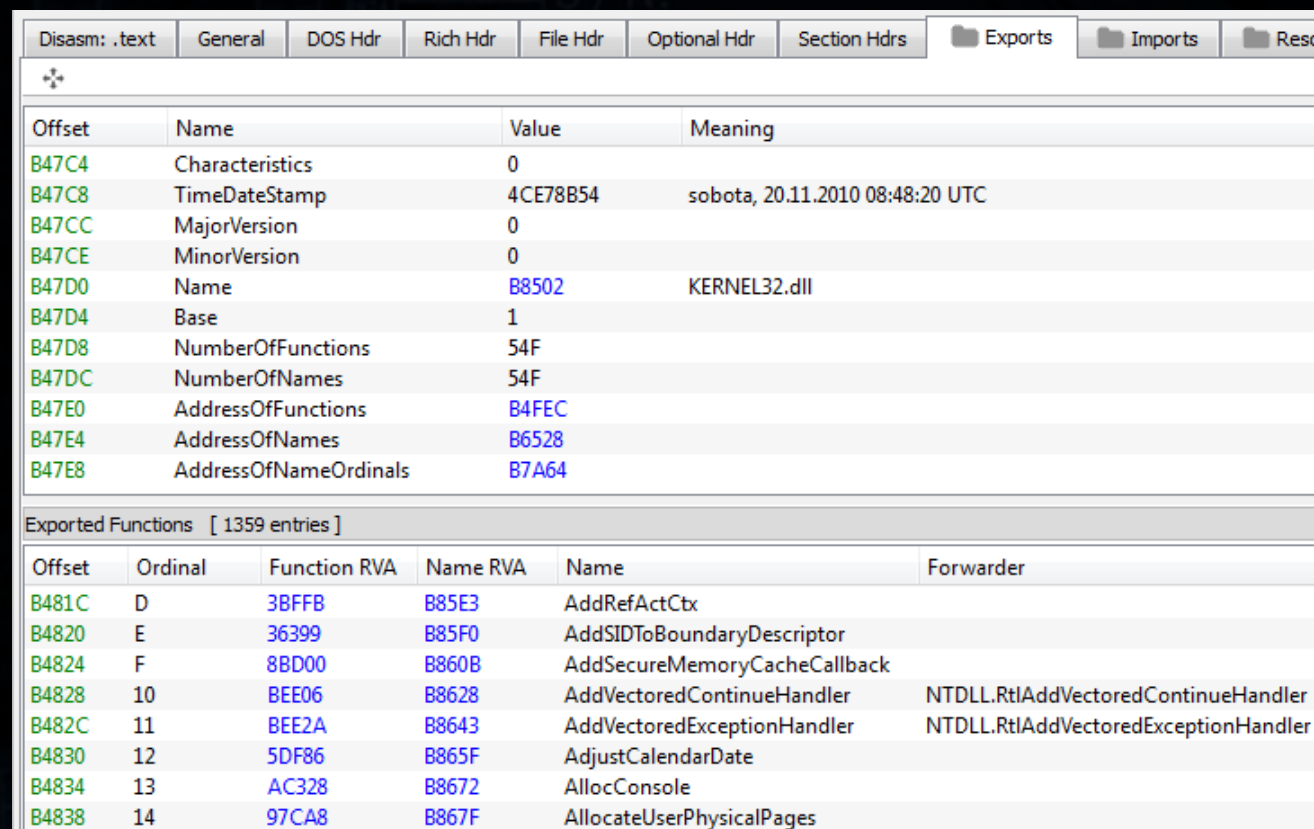
# Basics of a PE file: Relocation

- Let's open one of our sample PEs in PE-bear and see the relocation table
- Check the code snippet to see how the relocation table is processed

Exercise time...

# Basics of a PE file: Exports

- Export Table



The screenshot shows the 'Exports' tab in a PE file viewer. The top table lists the export table entries with their offsets, names, values, and meanings. The bottom table lists the exported functions with their offsets, ordinals, function RVA, name RVA, names, and forwarders.

Offset	Name	Value	Meaning
B47C4	Characteristics	0	
B47C8	TimeDateStamp	4CE78B54	sobota, 20.11.2010 08:48:20 UTC
B47CC	MajorVersion	0	
B47CE	MinorVersion	0	
B47D0	Name	B8502	KERNEL32.dll
B47D4	Base	1	
B47D8	NumberOfFunctions	54F	
B47DC	NumberOfNames	54F	
B47E0	AddressOfFunctions	B4FEC	
B47E4	AddressOfNames	B6528	
B47E8	AddressOfNameOrdinals	B7A64	

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
B481C	D	3BFFB	B85E3	AddRefActCtx	
B4820	E	36399	B85F0	AddSIDToBoundaryDescriptor	
B4824	F	8BD00	B860B	AddSecureMemoryCacheCallback	
B4828	10	BEE06	B8628	AddVectoredContinueHandler	NTDLL.RtlAddVectoredContinueHandler
B482C	11	BEE2A	B8643	AddVectoredExceptionHandler	NTDLL.RtlAddVectoredExceptionHandler
B4830	12	5DF86	B865F	AdjustCalendarDate	
B4834	13	AC328	B8672	AllocConsole	
B4838	14	97CA8	B867F	AllocateUserPhysicalPages	

[https://github.com/hasherezade/malware\\_training\\_vol1/blob/main/exercises/module1/lesson2\\_pe/pe\\_snippets/export\\_lookup.h](https://github.com/hasherezade/malware_training_vol1/blob/main/exercises/module1/lesson2_pe/pe_snippets/export_lookup.h)



# Basics of a PE file: Exports

1. DLLs are libraries of functions for other PEs to use
2. An Export Table is a catalogue allowing to find and use a particular function

Exported Functions [ 1359 entries ]				
Offset	Ordinal	Function RVA	Name RVA	Name
B56E0	3BE	60E73	BCF4F	ReadConsoleW
B56E4	3BF	42C62	BCF5C	ReadDirectoryChangesW
B56E8	3C0	496FB	BCF72	ReadFile
B56EC	3C1	63D99	BCF7B	
B56F0	3C2	31B14	BCF86	
B56F4	3C3	3C1CE	BCF96	
B56F8	3C4	9851F	BCFA8	
B56FC	3C5	4CB4F	BCFC0	
B5700	3C6	42D7C	BCFCC	
B5704	3C7	40D25	BCFDC	
B5708	3C8	A8CD5	BCFEC	
B570C	3C9	36644	BCFFC	

	Hex	Disasm
77E296FB	6A0C	PUSH 0XC
77E296FD	685897E277	PUSH 0X77E29758
77E29702	E869260000	CALL 0X77E2BD70
77E29707	8B7D14	MOV EDI, DWORD PTR [EBP + 0X14]
77E2970A	85FF	TEST EDI, EDI
77E2970C	7403	JE SHORT 0X77E29711
77E2970E	832700	AND DWORD PTR [EDI], 0
77E29711	8B7508	MOV ESI, DWORD PTR [EBP + 8]

# Basics of a PE file: Exports

```
00311F4F . 6A 00          PUSH 0
00311F51 . 8D95 ECEFFFFF LEA EDX,DWORD PTR SS:[EBP-1014]
00311F57 . 52            PUSH EDX
00311F58 . 68 00100000    PUSH 1000
00311F5D . 8D95 FCEFFFFF LEA EDX,DWORD PTR SS:[EBP-1004]
00311F63 . 52            PUSH EDX
00311F64 . FF3401        PUSH DWORD PTR DS:[ECX+EAX]
00311F67 . FF15 00F03100 CALL DWORD PTR DS:[<&KERNEL32.ReadFile>]
00311F6D . 85C0          TEST EAX,EAX
00311F6F . 0F84 6D010000 JE pe_unmap.003120E2
00311F7F . 58            POP EAX
DS:[0031F008]=768E96FB kernel32.ReadFile
```

pOverlapped = NULL  
pBytesRead  
BytesToRead = 1000 (4096.)  
Buffer  
hFile  
ReadFile

Address	Hex dump	ASCII
0031F008	FB 96 8E 76 FF 98 8F 76 8E 3E 8E 76 95 A2 03 77	GPAv sCvE>AvLdAw
0031F018	10 CD 03 77 B5 76 8F 76 8E BF 8E 76 D0 BB 8E 76	AwAwAwAwAwAwAwAwAwAw

	Hex	Disasm	
77E296FB	★ 6A0C	PUSH 0xC	ReadFile
77E296FD	685897E277	PUSH 0X77E29758	
77E29702	E869260000	CALL 0X77E2BD70	
77E29707	8B7D14	MOV EDI, DWORD PTR [EBP + 0X14]	
77E2970A	85FF	TEST EDI, EDI	
77E2970C	7403	JE SHORT 0X77E29711	
77E2970E	832700	AND DWORD PTR [EDI], 0	
77E29711	8B7508	MOV ESI, DWORD PTR [EBP + 8]	



# Basics of a PE file: Exports

1. Functions can be exported by a name or by ordinal (a number)
2. Some exports can be forwarded (pointing to other functions, in other DLLs)



# Basics of a PE file: Exports

- Forwarded functions

kernel32.dll

Exported Functions [ 1359 entries ]					
Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
B4B28	D0	2028C	B9588	DelayLoadFailureHook	
B4B2C	D1	4266A	B959D	DeleteAtom	
B4B30	D2	BEF89	B95A8	DeleteBoundaryDescriptor	NTDLL.RtlDeleteBoundaryDescriptor
B4B34	D3	BEFAB	B95C1	DeleteCriticalSection	NTDLL.RtlDeleteCriticalSection
B4B38	D4	34F66	B95D7	DeleteFiber	
B4B3C	D5	447CB	B95E3	DeleteFileA	
B4B40	D6	89993	B95EF	DeleteFileTransactedA	

ntdll.dll

Exported Functions [ 1990 entries ]				
Offset	Ordinal	Function RVA	Name RVA	Name
3614C	2E6	556D1	3ECD1	RtlDeleteBoundaryDescriptor
36150	2E7	59AC5	3ECED	RtlDeleteCriticalSection
36154	2E8	5DD50	3ED06	RtlDeleteElementGenericTable
36158	2E9	DC18	3ED23	RtlDeleteElementGenericTableAvl
3615C	2EA	76E87	3ED43	RtlDeleteHashTable
36160	2EB	B6994	3ED56	RtlDeleteNoSplay

	Disasm	Hint
59AC5	★ MOV EDI, EDI	RtlDeleteCriticalSection
59AC7	PUSH EBP	
59AC8	MOV EBP, ESP	
59ACA	PUSH -2	
59ACC	PUSH 0X77F10DE8	
59AD1	PUSH 0X77EDE0ED	



# Basics of a PE file: Imports

- Import Table

Disasm: .rdata	General	DOS Hdr	Rich Hdr	File Hdr	Optional Hdr	Section Hdrs	Imports	Resources	BaseRel
✕ + 📁									
Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk	
22474	KERNEL32.dll	93	FALSE	2309C	0	0	23328	1D000	
KERNEL32.dll [ 93 entries ]									
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint			
1D000	CreateDirectoryA	-	23214	23214	-	C1			
1D004	CloseHandle	-	23228	23228	-	8E			
1D008	GetLastError	-	23236	23236	-	26A			
1D00C	OpenProcess	-	23246	23246	-	408			
1D010	VirtualFree	-	23254	23254	-	5AE			
1D014	CreateToolhelp...	-	23262	23262	-	10A			
1D018	Module32First	-	2327E	2327E	-	3DF			
1D01C	Module32Next	-	2328E	2328E	-	3E1			
1D020	CreateFileA	-	2329E	2329E	-	CE			
1D024	GetFileSize	-	232AC	232AC	-	254			
1D028	MapViewOfFile	-	232BA	232BA	-	3DB			
1D02C	UnmapViewOff...	-	232CA	232CA	-	593			
1D030	CreateFileMapp...	-	232DC	232DC	-	CF			

[https://github.com/hasherezade/malware\\_training\\_vol1/blob/main/exercises/module1/lesson2\\_pe/pe\\_snippets/imports\\_load.h](https://github.com/hasherezade/malware_training_vol1/blob/main/exercises/module1/lesson2_pe/pe_snippets/imports_load.h)

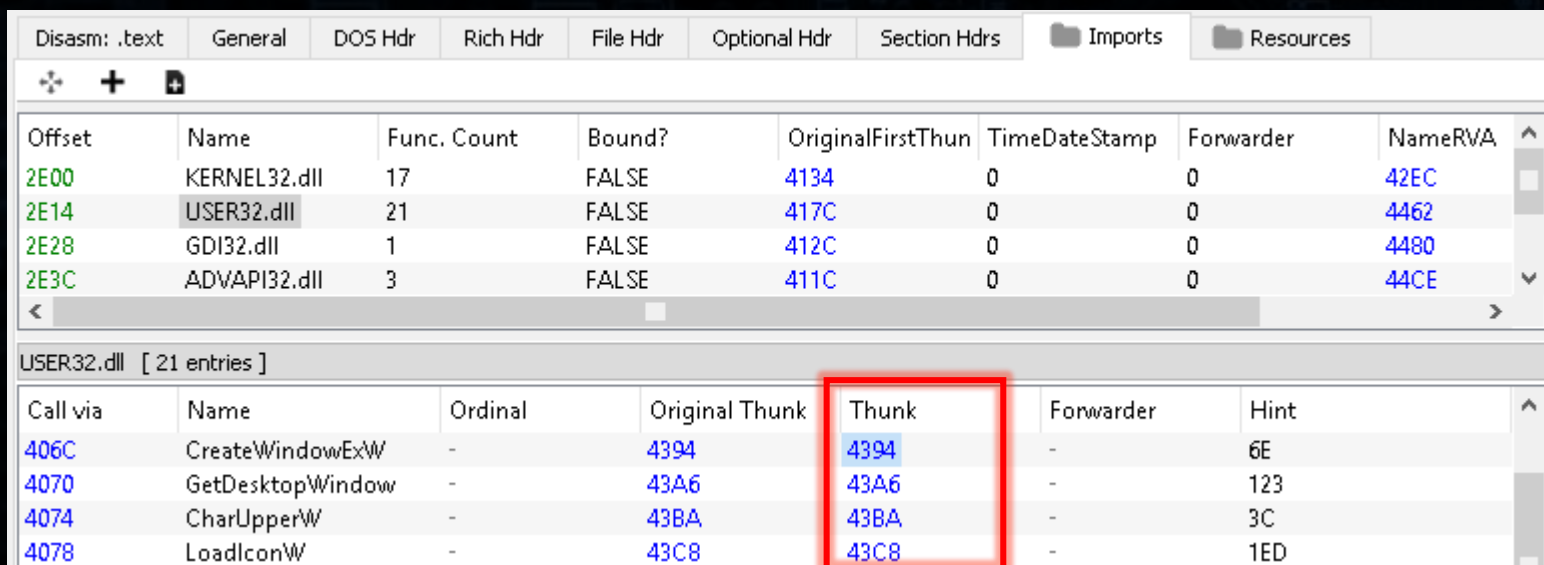
- Import Table: structure





# Basics of a PE file: Imports

- Raw: before filling imports



The screenshot shows the 'Imports' tab in a PE file viewer. The top table lists imported DLLs: KERNEL32.dll, USER32.dll, GDI32.dll, and ADVAPI32.dll. The 'USER32.dll' entry is selected, and a second table below it shows the raw import entries for USER32.dll. The 'Thunk' column in this second table is highlighted with a red box, showing values like 4394, 43A6, 43BA, and 43C8.

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA
2E00	KERNEL32.dll	17	FALSE	4134	0	0	42EC
2E14	USER32.dll	21	FALSE	417C	0	0	4462
2E28	GDI32.dll	1	FALSE	412C	0	0	4480
2E3C	ADVAPI32.dll	3	FALSE	411C	0	0	44CE

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
406C	CreateWindowExW	-	4394	4394	-	6E
4070	GetDesktopWindow	-	43A6	43A6	-	123
4074	CharUpperW	-	43BA	43BA	-	3C
4078	LoadIconW	-	43C8	43C8	-	1ED

# Basics of a PE file

- Loaded: after filling imports – thunks are filled with addresses of exported functions

Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeStamp	Forwarder	NameRVA
2E00	KERNEL32.dll	17	FALSE	4134	0	0	42EC
2E14	USER32.dll	21	FALSE	417C	0	0	4462
2E28	GDI32.dll	1	FALSE	412C	0	0	4480
2E3C	ADVAPI32.dll	3	FALSE	411C	0	0	44CE

Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
406C	CreateWindowExW	-	4394	768A2480	-	6E
4070	GetDesktopWindow	-	43A6	768AA100	-	123
4074	CharUpperW	-	43BA	768B53B0	-	3C
4078	LoadIconW	-	43C8	768B5890	-	1ED



# Basics of a PE file: Imports

The screenshot shows a disassembler window with the 'Imports' tab selected. The 'USER32.dll' import table is displayed, showing the 'Name' and 'Thunk' columns. The 'Thunk' column contains the absolute addresses of the imported functions. A callout box points to the 'Thunk' value for 'CreateWindowExW', which is '768A2480'. The callout box contains the following text:

**DLL Base + Function RVA**  
**Example:**  
**76B70000 + 32480 =**  
**768A2480**

The 'Exported Functions' table is also visible, showing the 'Function RVA' and 'Name RVA' columns. The 'Function RVA' for 'CreateWindowExW' is '32480', which is highlighted by a red box. A red arrow points from this 'Function RVA' to the 'Thunk' value in the 'Imports' table.

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	Thunk
2E00	KERNEL32.dll	17	FALSE	4134	
2E14	USER32.dll	21	FALSE	417C	
2E28	GDI32.dll	1	FALSE	412C	
2E3C	ADVAPI32.dll	3	FALSE	411C	

Call via	Name	Ordinal	Original Thunk	Thunk
406C	CreateWindowExW	-	4394	768A2480
4070	GetDesktopWindow	-	43A6	768AA100
4074	CharUpperW	-	43BA	768B53B0
4078	LoadIconW	-		768B5890

Offset	Name	Value	Meaning
9BCF0	Characteristics	0	
9BCF4	TimeDateStamp	7BD785F8	Saturday, 03.11.2035 17:01:44 UTC
9BCF8	MajorVersion	0	
9BCFA	MinorVersion	0	
9BCFC	Name	9F384	USER32.dll
9BD00	Base	5DE	
9BD04	NumberOfFunc...	4BF	
9BD08	NumberOfNames	3E8	
9BD0C	AddressOfFunc...	9C918	
9BD10	AddressOfNames	9DC14	

Offset	Ordinal	Function RVA	Name RVA	Name
9BF04	659	2C340	9FBC9	CreateWindowExA
9BF08	65A	32480	9FBD9	CreateWindowExW
9BF0C	65B	88D90	9FBE9	CreateWindowInBand
9BF10	65C	88DE0	9BFBC	CreateWindowInBandEx
9BF14	65D	88E30	9FC11	CreateWindowIndirect
9BF18	65E	907F0	9FC26	CreateWindowStationA

# Basics of a PE file: Imports

```
00311F4F . 6A 00          PUSH 0
00311F51 . 8D95 ECEFFFFF LEA EDX,DWORD PTR SS:[EBP-1014]
00311F57 . 52            PUSH EDX
00311F58 . 68 00100000    PUSH 1000
00311F5D . 8D95 FCEFFFFF LEA EDX,DWORD PTR SS:[EBP-1004]
00311F63 . 52            PUSH EDX
00311F64 . FF3401        PUSH DWORD PTR DS:[ECX+EAX*1]
00311F67 . FF15 00F03100 CALL DWORD PTR DS:[<&KERNEL32.ReadFile>]
00311F6D . 85C0          TEST EAX,EAX
00311F6F . 0F84 6D010000 JE pe_unmap.003120E2
00311F70 . 58            POP EAX
DS:[0031F008]=768E96FB (kernel32.ReadFile)
```

pOverlapped = NULL  
pBytesRead  
BytesToRead = 1000 (4096.)  
Buffer  
hFile  
ReadFile

Address	Hex dump
0031F008	FB 96 8E 76 FF 98 8F 76 A8 3E 8E 76 95
0031F018	10 CD 03 77 EF 76 8F 76 00 BF 8E 76 D0

Disasm: .text   General   DOS Hdr   Rich Hdr   File Hdr   Optional Hdr   Section Hdrs   Imports						
✕ + D						
Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forward
1013C	KERNEL32.dll	77	FALSE	11364	0	0
KERNEL32.dll [ 77 entries ]						
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
F000	VirtualAlloc	-	1149C	768F2FB6	-	5AB
F004	VirtualFree	-	114AC	768F1DA4	-	5AE
F008	ReadFile	-	114C8	768E96FB	-	458
F00C	GetCommandLi...	-	114D4	768F98FF	-	1E2



# Basics of a PE file: Imports

- Let's open one of our sample PEs in PE-bear and see the import table. Find the corresponding DLLs and their exports.
- Check the code snippets to see how the import and export tables are processed

Exercise time...

# Exercise

- Compile the given code of a custom PE loader and get familiar with it
  - [https://github.com/hasherezade/malware\\_training\\_voll/tree/main/exercises/module1/lesson2\\_pe](https://github.com/hasherezade/malware_training_voll/tree/main/exercises/module1/lesson2_pe)