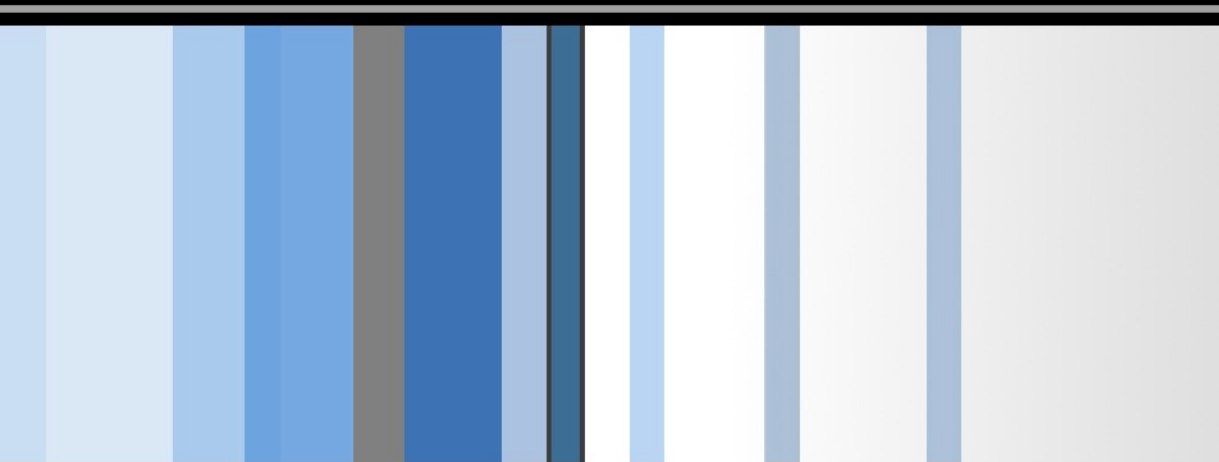


CSC 471 Modern Malware Analysis

Code Injection (3) & PE Structure (2)

Si Chen (schen@wcupa.edu)

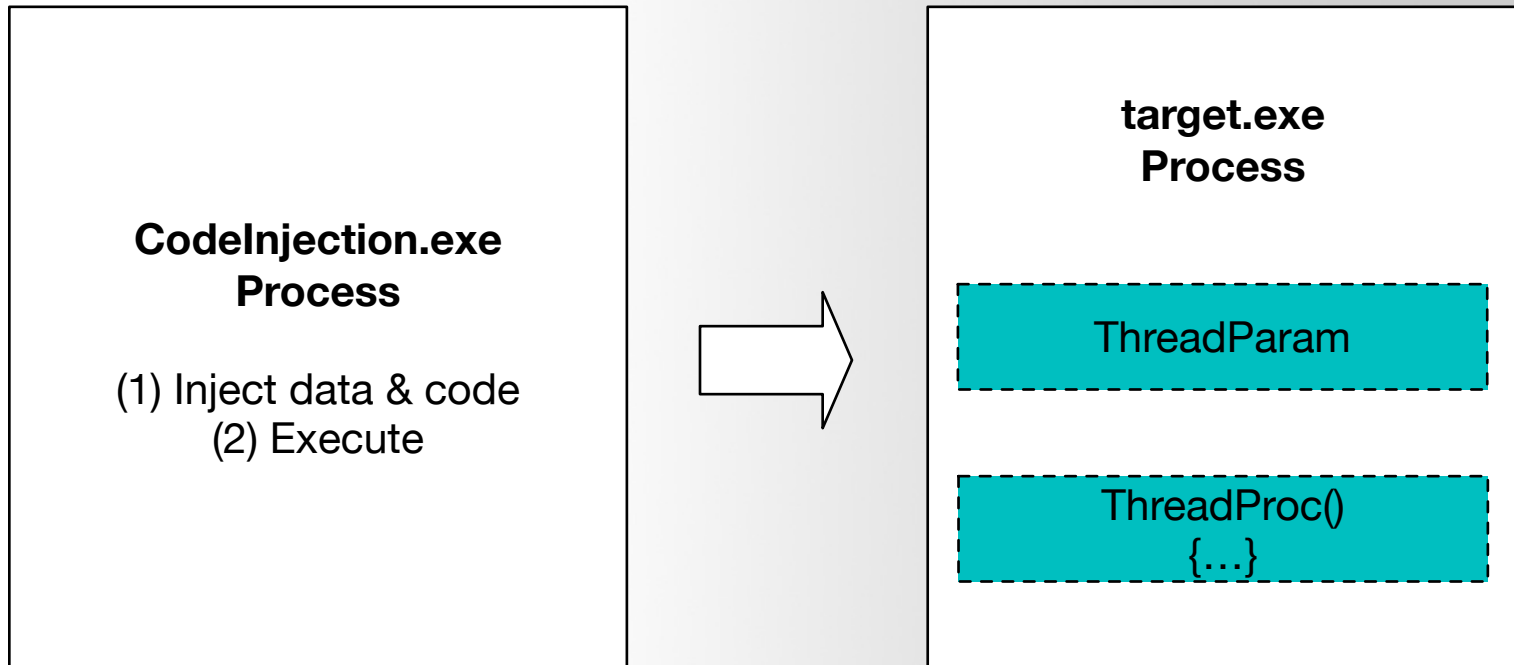




CODE INJECTION

Code injection is the term used to describe attacks that inject code into an application. That injected code is then interpreted by the application.

Code Injection (thread injection)



code → injected by ThreadProc()
data → injected as ThreadParam

CodeInjection.cpp – ThreadProc()

```
32
33 DWORD WINAPI ThreadProc(LPVOID lParam)
34 {
35     PTHREAD_PARAM    pParam        = (PTHREAD_PARAM)lParam;
36     HMODULE           hMod          = NULL;
37     FARPROC           pFunc         = NULL;
38
39     // LoadLibrary()
40     hMod = ((PFLOADLIBRARYA)pParam->pFunc[0])(pParam->szBuf[0]);    // "user32.dll"
41     if( !hMod )
42         return 1;
43
44     // GetProcAddress()
45     pFunc = (FARPROC)((PFGETPROCADDRESS)pParam->pFunc[1])(hMod, pParam->szBuf[1]);    // "MessageBoxA"
46     if( !pFunc )
47         return 1;
48
49     // MessageBoxA()
50     ((PFMESSAGEBOXA)pFunc)(NULL, pParam->szBuf[2], pParam->szBuf[3], MB_OK);
51
52     return 0;
53 }
54
```

hMod = LoadLibraryA("user32.dll");

pFunc = GetProcAddress(hMod, "MessageBoxA");

pFunc(NULL, www.reversecore.com, "ReverseCore", MB_OK);

CodeInjection.cpp – InjectCode()

```
// Main injection function: performs process and thread injection into a target process.
✓ BOOL InjectCode(DWORD dwPID)
{
✓   // Prepare the THREAD_PARAM structure with necessary function pointers and strings.
   // Open the target process with necessary privileges.
   // Allocate memory in the target process for THREAD_PARAM.
   // Write THREAD_PARAM to the allocated memory in the target process.
   // Allocate memory for the ThreadProc function in the target process and set it to executable.
   // Write the ThreadProc function to the allocated memory in the target process.
   // Create a remote thread in the target process that starts at the ThreadProc function.
   // Wait for the thread to complete execution.
   // Close handles and return TRUE on successful injection.

   HMODULE      hMod      = NULL;
   THREAD_PARAM param      = {0,};
   HANDLE        hProcess  = NULL;
   HANDLE        hThread   = NULL;
   LPVOID        pRemoteBuf[2] = {0,};
   DWORD         dwSize    = 0;

   hMod = GetModuleHandleA("kernel32.dll");

   // set THREAD_PARAM
   param.pFunc[0] = GetProcAddress(hMod, "LoadLibraryA");
   param.pFunc[1] = GetProcAddress(hMod, "GetProcAddress");
   strcpy_s(param.szBuf[0], "user32.dll");
   strcpy_s(param.szBuf[1], "MessageBoxA");
   strcpy_s(param.szBuf[2], "cs.wcupa.edu");
   strcpy_s(param.szBuf[3], "Dr. Chen");

   // Open Process
   if ( !(hProcess = OpenProcess(PROCESS_ALL_ACCESS, // dwDesiredAccess
                                FALSE,              // bInheritHandle
                                dwPID)) )           // dwProcessId
   {
      printf("OpenProcess() fail : err_code = %d\n", GetLastError());
      return FALSE;
   }

   // Allocation for THREAD_PARAM
   dwSize = sizeof(THREAD_PARAM);
   if( !(pRemoteBuf[0] = VirtualAllocEx(hProcess, // hProcess
                                       NULL,       // lpAddress
                                       dwSize,     // dwSize
                                       MEM_COMMIT, // flAllocationType
                                       PAGE_READWRITE)) ) // flProtect
   {
      printf("VirtualAllocEx() fail : err_code = %d\n", GetLastError());
      return FALSE;
   }
}
```

CodeInjection.cpp – InjectCode()

```
// Allocation for THREAD_PARAM
dwSize = sizeof(THREAD_PARAM);
if( !(pRemoteBuf[0] = VirtualAllocEx(hProcess,           // hProcess
                                     NULL,               // lpAddress
                                     dwSize,              // dwSize
                                     MEM_COMMIT,          // flAllocationType
                                     PAGE_READWRITE)) )   // flProtect
{
    printf("VirtualAllocEx() fail : err_code = %d\n", GetLastError());
    return FALSE;
}

if( !WriteProcessMemory(hProcess,                       // hProcess
                        pRemoteBuf[0],                  // lpBaseAddress
                        (LPVOID)&param,                  // lpBuffer
                        dwSize,                          // nSize
                        NULL) )                          // [out] lpNumberOfBytesWritten
{
    printf("WriteProcessMemory() fail : err_code = %d\n", GetLastError());
    return FALSE;
}

// Allocation for ThreadProc()
dwSize = (DWORD)InjectCode - (DWORD)ThreadProc;
if( !(pRemoteBuf[1] = VirtualAllocEx(hProcess,          // hProcess
                                     NULL,               // lpAddress
                                     dwSize,              // dwSize
                                     MEM_COMMIT,          // flAllocationType
                                     PAGE_EXECUTE_READWRITE)) ) // flProtect
{
    printf("VirtualAllocEx() fail : err_code = %d\n", GetLastError());
    return FALSE;
}

if( !WriteProcessMemory(hProcess,                       // hProcess
                        pRemoteBuf[1],                  // lpBaseAddress
                        (LPVOID)ThreadProc,              // lpBuffer
                        dwSize,                          // nSize
                        NULL) )                          // [out] lpNumberOfBytesWritten
{
    printf("WriteProcessMemory() fail : err_code = %d\n", GetLastError());
    return FALSE;
}

if( !(hThread = CreateRemoteThread(hProcess,            // hProcess
                                   NULL,                 // lpThreadAttributes
                                   0,                   // dwStackSize
                                   (LPTHREAD_START_ROUTINE)pRemoteBuf[1], // dwStackSize
                                   pRemoteBuf[0],         // lpParameter
                                   0,                   // dwCreationFlags
                                   0))
```

CodeInjection.cpp – InjectCode()

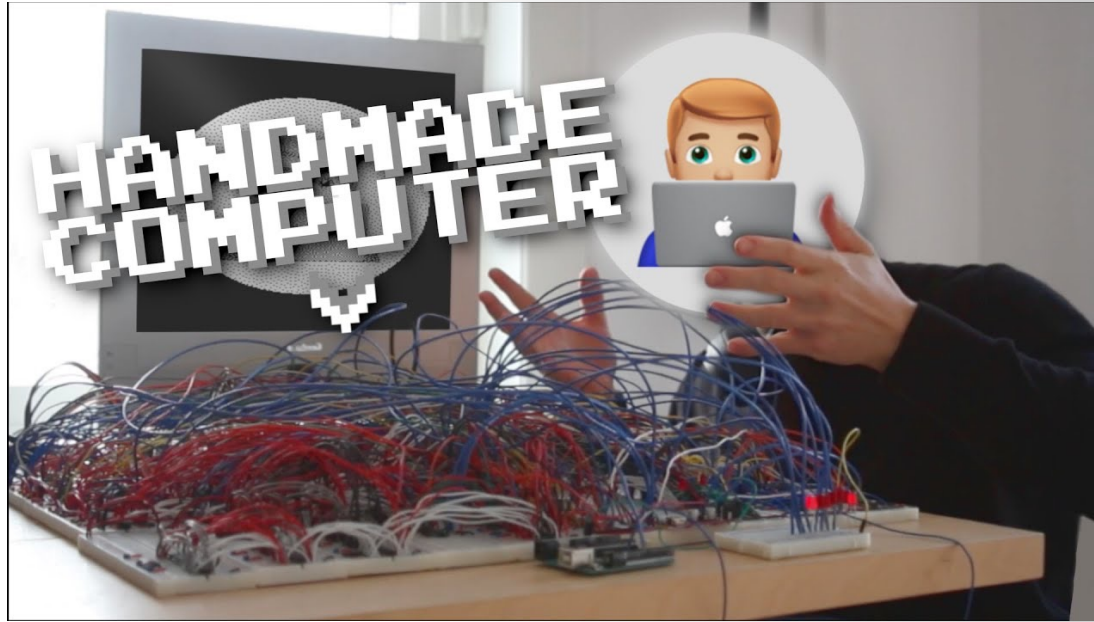
```
// Prepare the THREAD_PARAM structure with necessary function pointers and
strings.
// Open the target process with necessary privileges.
// Allocate memory in the target process for THREAD_PARAM.
// Write THREAD_PARAM to the allocated memory in the target process.
// Allocate memory for the ThreadProc function in the target process and set
it to executable.
// Write the ThreadProc function to the allocated memory in the target
process.
// Create a remote thread in the target process that starts at the ThreadProc
function.
// Wait for the thread to complete execution.
// Close handles and return TRUE on successful injection.
```

- OpenProcess()
- **//data: THREAD_PARAM**
- VirtualAllocEx()
- WriteProcessMemory()
- **//Code: ThreadProc()**
- VirtualAllocEx()
- WriteProcessMemory()
- CreateRemoteThread()

How to Debug Code Injection (OllyDBG)

```
* OllyDbg - NOTEPAD.EXE - [CPU - thread 00000808]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
00980000 55 PUSH EBP
00980001 8BEC MOV EBP,ESP
00980003 56 PUSH ESI
00980004 8B75 08 MOV ESI,DWORD PTR SS:[EBP+8]
00980007 8B0E MOV ECX,DWORD PTR DS:[ESI]
00980009 8D46 08 LEA EAX,DWORD PTR DS:[ESI+8]
0098000C 50 PUSH EAX
0098000D FFD1 CALL ECX
0098000F 85C0 TEST EAX,EAX
00980011 75 0A JNZ SHORT 0098001D
00980013 B8 01000000 MOV EAX,1
00980018 5E POP ESI
00980019 5D POP EBP
0098001A C2 0400 RETN 4
0098001D 8D96 88000000 LEA EDX,DWORD PTR DS:[ESI+88]
00980023 52 PUSH EDX
00980024 50 PUSH EAX
00980025 8B46 04 MOV EAX,DWORD PTR DS:[ESI+4]
00980028 FFD0 CALL EAX
0098002A 85C0 TEST EAX,EAX
0098002C 74 E5 JE SHORT 00980013
0098002E 6A 00 PUSH 0
00980030 8D8E 88010000 LEA ECX,DWORD PTR DS:[ESI+188]
00980036 51 PUSH ECX
00980037 81C6 08010000 ADD ESI,108
0098003D 56 PUSH ESI
0098003E 6A 00 PUSH 0
00980040 FFD0 CALL EAX
00980042 33C0 XOR EAX,EAX
00980044 5E POP ESI
00980045 5D POP EBP
00980046 C2 0400 RETN 4
```


Ancient forbidden technique: manual code injection.

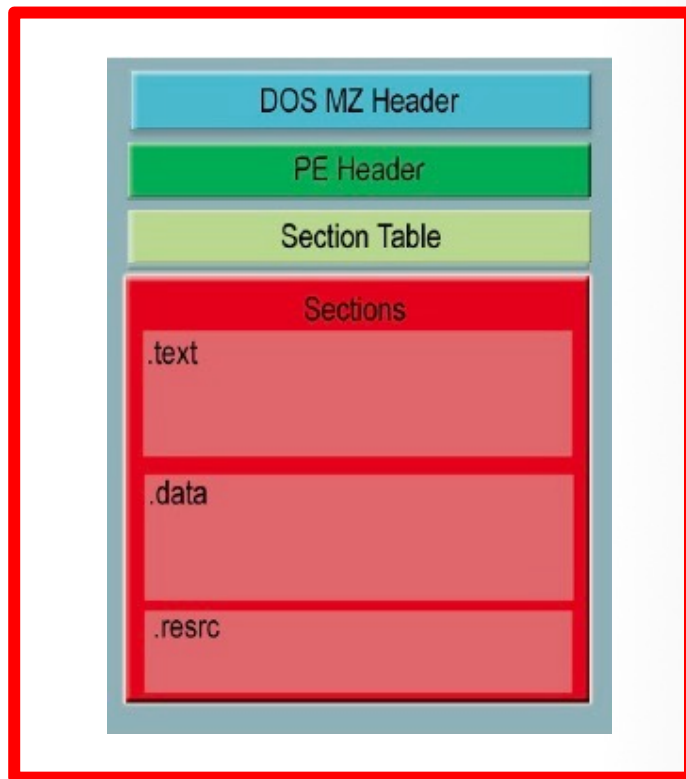


00401000	55	PUSH EBP	
00401001	8BEC	MOV EBP,ESP	
00401003	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]	
00401006	68 6C6C0000	PUSH 6C6C	
00401008	68 33322E64	PUSH 642E3233	
00401010	68 75736572	PUSH 72657375	
00401015	54	PUSH ESP	
00401016	FF16	CALL DWORD PTR DS:[ESI]	
00401018	68 6F784100	PUSH 41786F	
0040101D	68 61676542	PUSH 42656761	
00401022	68 4D657373	PUSH 7373654D	
00401027	54	PUSH ESP	
00401028	50	PUSH EAX	
00401029	FF56 04	CALL DWORD PTR DS:[ESI+4]	asntest.00401029(guessed Arg1)
0040102C	6A 00	PUSH 0	
0040102E	E8 00000000	CALL 0040103B	
00401033	44	INC ESP	
00401034	72 2E	JB SHORT 00401064	
00401036	43	INC EBX	
00401037	68 656E00E8	PUSH E8006E65	
0040103C	1900	SBB DWORD PTR DS:[EAX],EAX	
0040103E	0000	ADD BYTE PTR DS:[EAX],AL	
00401040	6373 2E	ARPL WORD PTR DS:[EBX+2E],SI	
00401043	77 63	JAE SHORT 004010A8	
00401045	75 70	JNE SHORT 004010B7	
00401047	61	POPAD	
00401048	2E	CS:	Two prefixes from the same group
00401049	65	GS:	Two prefixes from the same group
0040104A	64:75 2F	JNE SHORT 0040107C	Superfluous segment override prefix
0040104D	6D	INS DWORD PTR ES:[EDI],DX	I/O command
0040104E	61	POPAD	
0040104F	6C	INS BYTE PTR ES:[EDI],DX	I/O command
00401050	77 61	JAE SHORT 004010B3	
00401052	72 65	JB SHORT 004010B9	
00401054	3230	XOR DH,BYTE PTR DS:[EAX]	
00401056	323400	XOR DH,BYTE PTR DS:[EAX+EAX]	
00401059	6A 00	PUSH 0	
0040105B	FFD0	CALL EAX	
0040105D	89EC	MOV ESP,EBP	
0040105F	5D	POP EBP	
00401060	C3	RETN	

OL-00 (current registers)															
Address	Hex dump														ASCII
00401000	55	8B	EC	8B	75	08	68	6C	6C	00	00	68	33	32	2E 64 U i u h l h 32 . d
00401010	68	75	73	65	72	54	FF	16	68	6F	78	41	00	68	61 67 huserT y hoxA hag
00401020	65	42	68	4D	65	73	73	54	50	FF	56	04	6A	00	E8 08 eBhMessTP y j e
00401030	00	00	00	44	72	2E	43	68	65	6E	00	E8	19	00	00 00 Dr . Chen e
00401040	63	73	2E	77	63	75	70	61	2E	65	64	75	2F	6D	61 6C cs.wcupa.edu/mal
00401050	77	61	72	65	32	30	32	34	00	6A	00	FF	D0	89	EC 5D ware2024 j y i]
00401060	C3	00	00	66	39	05	00	00	40	00	75	38	A1	3C	00 40 A f 9 @ u ; < @

Portable Executable (PE) file

- A Portable Executable (**PE**) **file** is the standard binary **file** format for an **Executable (.exe) or DLL** under Windows NT, Windows 95, and Win32.
- Derived from COFF (Common Object File Format) in UNIX platform, and it is not really “portable”.



Now here is the kicker. Even though this specification is spelled out by Microsoft, compilers/linkers chose to ignore some parts of it.

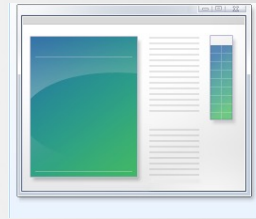
To make things even worse, the Microsoft loader doesn't enforce a good portion of this specification and instead makes assumptions if things start getting weird.

So even though the spec outlined here says a particular field is supposed to hold a certain value, the compiler/linker or **even a malicious actor could put whatever they want in there and the program will likely still run...**

Portable Executable (PE) file

- PE formatted files include:

- .exe, .scr (executable)
- .dll, .ocx, .cpl, drv (library)
- .sys, .vxd (driver files)
- .obj (objective file)



- All PE formatted files can be executed, except obj file.

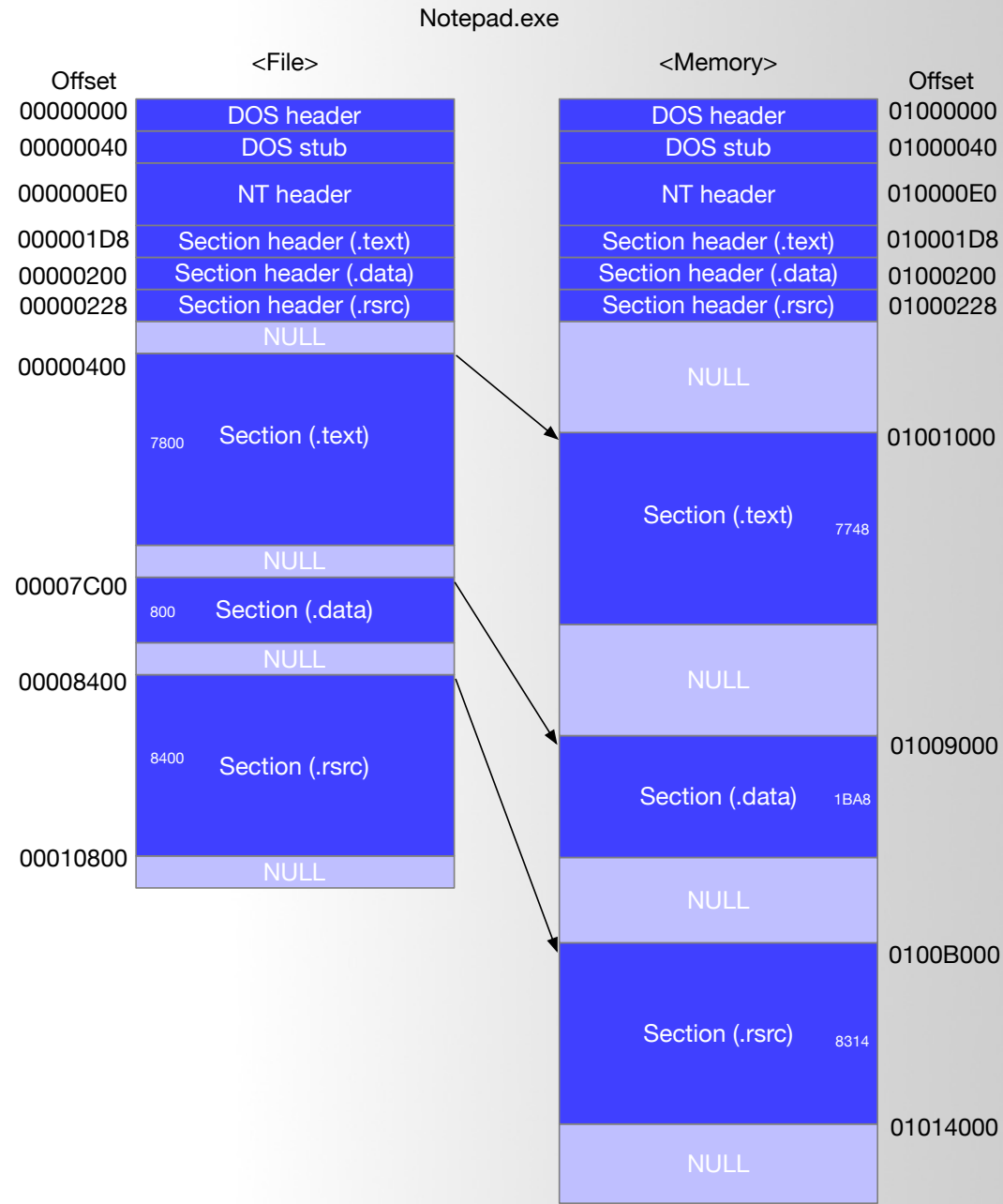
- .exe, .scr can be directly executed inside Shell (explorer.exe)
- others can be executed by other program/service

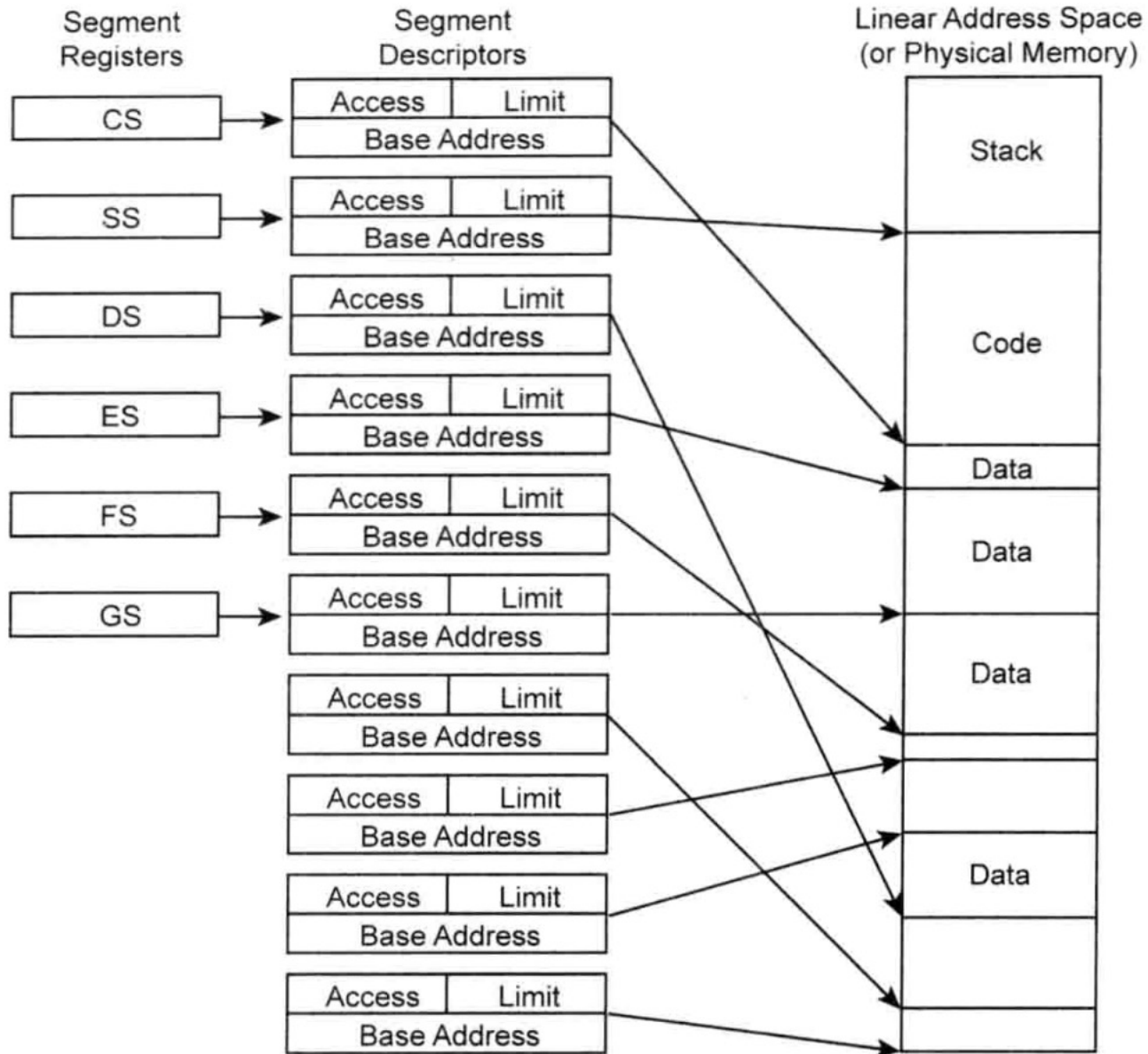
- **PE refers to 32 bit** executable file, or **PE32**. **64 bit** executable file is named as **PE+ or PE32+**. (Note that it is not PE64).

PE Example – Notepad.exe

00000000	4D 5A 90 00 03 00 00 00	04 00 00 00 FF FF 00 00	MZÉ..... ..
00000010	B8 00 00 00 00 00 00 00	40 00 00 00 00 00 00 00	7.....@.....
00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 00 00	00 00 00 00 E8 00 00 00Φ...
00000040	0E 1F BA 0E 00 B4 09 CD	21 B8 01 4C CD 21 54 68!=!7.L=!Th
00000050	69 73 20 70 72 6F 67 72	61 6D 20 63 61 6E 6E 6F	is.program.canno
00000060	74 20 62 65 20 72 75 6E	20 69 6E 20 44 4F 53 20	t.be.run.in.DOS.
00000070	6D 6F 64 65 2E 0D 0D 0A	24 00 00 00 00 00 00 00	mode....\$......
00000080	A5 6D 16 9B E1 0C 78 C8	E1 0C 78 C8 E1 0C 78 C8	Ñm.¢ß.xℒß.xℒ
00000090	1B 2F 38 C8 E0 0C 78 C8	E1 0C 78 C8 E0 0C 78 C8	./8ℒα.xℒß.xℒα.xℒ
000000A0	1B 2F 61 C8 F2 0C 78 C8	E1 0C 79 C8 23 0C 78 C8	./aℒ≥.xℒß.yℒ#.xℒ
000000B0	76 2F 3D C8 E0 0C 78 C8	3B 2F 64 C8 F2 0C 78 C8	v/=ℒα.xℒ;/dℒ≥.xℒ
000000C0	1B 2F 45 C8 E0 0C 78 C8	52 69 63 68 E1 0C 78 C8	./Eℒα.xℒRichß.xℒ
000000D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000000E0	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L...
000000F0	0D 84 7D 3B 00 00 00 00	00 00 00 00 E0 00 0F 01	.ä};.....α...
00000100	0B 01 07 00 00 6E 00 00	00 A6 00 00 00 00 00 00n... ^a
00000110	E0 6A 00 00 00 10 00 00	00 80 00 00 00 00 00 01	αj.....Ç.....
00000120	00 10 00 00 00 02 00 00	05 00 01 00 05 00 01 00
00000130	04 00 00 00 00 00 00 00	00 30 01 00 00 04 00 00θ.....
00000140	55 D8 01 00 02 00 00 80	00 00 04 00 00 10 01 00	U†.....Ç.....
00000150	00 00 10 00 00 10 00 00	00 00 00 00 10 00 00 00
00000160	00 00 00 00 00 00 00 00	20 6D 00 00 C8 00 00 00m..ℒ...
00000170	00 A0 00 00 48 89 00 00	00 00 00 00 00 00 00 00	.á..Hë.....
00000180	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00000190	40 13 00 00 1C 00 00 00	00 00 00 00 00 00 00 00	@.....

Load PE file (Notepad.exe) into Memory





VA & RVA

- VA (Virtual Address): The address is called a “VA” because **Windows creates a distinct VA space for each process, independent of physical memory**. For almost all purposes, a VA should be considered just an address. A VA is not as predictable as an RVA because the loader might not load the image at its preferred location.
- RVA (Relative Virtual Address): The address of an item after it is loaded into memory, with the base address of the image file subtracted from it. The RVA of an item almost always differs from its position within the file on disk (file pointer).

$$\text{RVA} + \text{ImageBase} = \text{VA}$$

In 32bit Windows OS, each process has 4GB virtual memory which means the range of VA is: **00000000 - FFFFFFFF**

DOS Header

```
struct DOS_Header
{
    // short is 2 bytes, long is 4 bytes
    char signature[2] = { 'M', 'Z' };
    short lastsize;
    short nblocks;
    short nreloc;
    short hdrsize;
    short minalloc;
    short maxalloc;
    void *ss; // 2 byte value
    void *sp; // 2 byte value
    short checksum;
    void *ip; // 2 byte value
    void *cs; // 2 byte value
    short relocpos;
    short noverlay;
    short reserved1[4];
    short oem_id;
    short oem_info;
    short reserved2[10];
    long e_lfanew; // Offset to the 'PE\0\0' signature relative to the beginning of the file
}
```

The first 2 letters are **always** the letters "**MZ**", the initials of Mark Zbikowski, who created the first linker for DOS. To some people, the first few bytes in a file that determine the type of file are called the "**magic number**,"

DOS Header

```
long e_lfanew;
```

long → 32 bit → ? Byte

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....yy..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00à...

E0 00 00 00

value for e_lfanew → ?

DOS Header

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....yy..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00à...

e_lfanew → 000000E0

DOS stub

00000040	OE 1F BA OE 00 B4 09 CD 21 B8 01 4C CD 21 54 68	[.°..'.í!_.Lí!Th
00000050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
00000060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
00000070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$.....
00000080	EC 85 5B A1 A8 E4 35 F2 A8 E4 35 F2 A8 E4 35 F2	i...[;`ä5ò`ä5ò`ä5ò
00000090	6B EB 3A F2 A9 E4 35 F2 6B EB 55 F2 A9 E4 35 F2	kë:ò@ä5òkëUò@ä5ò
000000A0	6B EB 68 F2 BB E4 35 F2 A8 E4 34 F2 63 E4 35 F2	këhò»ä5ò`ä4òcä5ò
000000B0	6B EB 6B F2 A9 E4 35 F2 6B EB 6A F2 BF E4 35 F2	këkò@ä5òkëjòçä5ò
000000C0	6B EB 6F F2 A9 E4 35 F2 52 69 63 68 A8 E4 35 F2	këoò@ä5òRich`ä5ò
000000D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

<https://virtualconsoles.com/online-emulators/dos/>

```
C:\>notepad.exe
This program cannot be run in DOS mode.
```

IMAGE_NT_HEADERS32 structure

12/04/2018 • 2 minutes to read

Represents the PE header format.

Syntax

C++

 Copy

```
typedef struct _IMAGE_NT_HEADERS {  
    DWORD      Signature;  
    IMAGE_FILE_HEADER      FileHeader;  
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;  
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

Members

Signature

A 4-byte signature identifying the file as a PE image. The bytes are "PE\0\0".

FileHeader

An [IMAGE_FILE_HEADER](#) structure that specifies the file header.

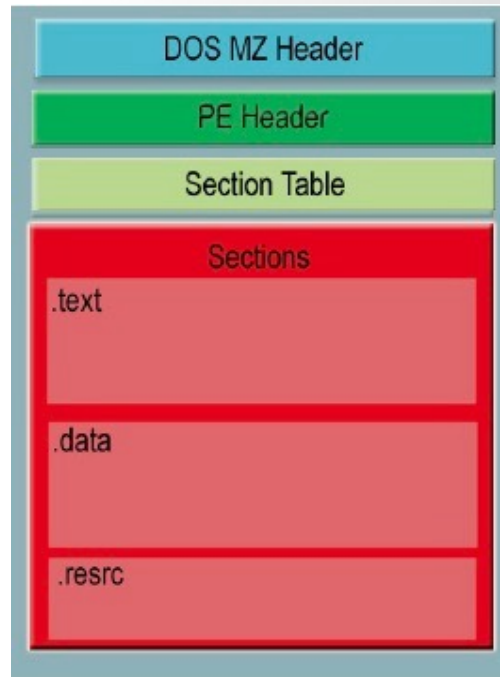
OptionalHeader

An [IMAGE_OPTIONAL_HEADER](#) structure that specifies the optional file header.

NT Header

NOTEPAD.EXE																	
Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000000E0	50	45	00	00	4C	01	03	00	A3	C3	B0	4A	00	00	00	00	PE..L...ËÃ°J....
000000F0	00	00	00	00	E0	00	0F	01	0B	01	07	0A	00	78	00	00à.....x..
00000100	00	A6	00	00	00	00	00	00	9D	73	00	00	00	10	00	00s.....
00000110	00	90	00	00	00	00	00	01	00	10	00	00	00	02	00	00
00000120	05	00	01	00	05	00	01	00	04	00	00	00	00	00	00	00
00000130	00	40	01	00	00	04	00	00	33	30	01	00	02	00	00	80	.@.....30.....€
00000140	00	00	04	00	00	10	01	00	00	00	10	00	00	10	00	00
00000150	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00
00000160	04	76	00	00	C8	00	00	00	00	B0	00	00	58	89	00	00	.v..È....°..X%..
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000180	00	00	00	00	00	00	00	00	50	13	00	00	1C	00	00	00P.....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001A0	00	00	00	00	00	00	00	00	A8	18	00	00	40	00	00	00"....@...
000001B0	00	00	00	00	00	00	00	00	00	10	00	00	48	03	00	00H...
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000001D0	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00text...

Section Header



Name	Privilege
.code	Executable, read
.data	Non-Executable, read/write
.resource	Non-Executable, read

IMAGE_SECTION_HEADER structure

12/04/2018 • 4 minutes to read

Represents the image section header format.

Syntax

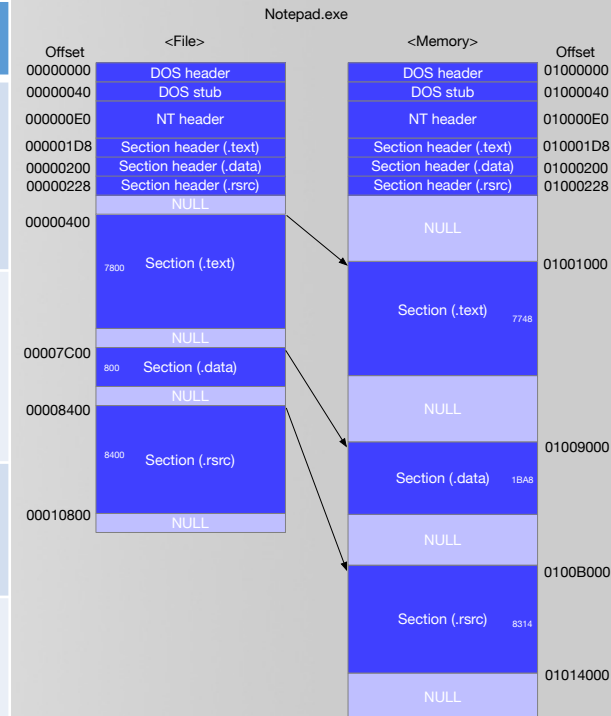
C++

 Copy

```
typedef struct _IMAGE_SECTION_HEADER {  
    BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];  
    union {  
        DWORD PhysicalAddress;  
        DWORD VirtualSize;  
    } Misc;  
    DWORD VirtualAddress;  
    DWORD SizeOfRawData;  
    DWORD PointerToRawData;  
    DWORD PointerToRelocations;  
    DWORD PointerToLinenumbers;  
    WORD  NumberOfRelocations;  
    WORD  NumberOfLinenumbers;  
    DWORD Characteristics;  
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```


Section Header

Members	Meaning
VirtualSize	The total size of the section when loaded into memory, in bytes.
VirtualAddress	The address of the first byte of the section when loaded into memory (RVA)
SizeOfRaw Data	The size of the section data on disk , in bytes.
PointerToRawData	The address of the first byte of the section on disk.
Characteristics	The characteristics of the image.



https://docs.microsoft.com/en-us/windows/desktop/api/winnt/ns-winnt-_image_section_header

Section Header

000001D0	00 00 00 00 00 00 00 00	2E 74 65 78 74 00 00 00text...
000001E0	48 77 00 00 00 10 00 00	00 78 00 00 00 04 00 00	Hw.....x.....
000001F0	00 00 00 00 00 00 00 00	00 00 00 00 20 00 00 60`
00000200	2E 64 61 74 61 00 00 00	A8 1B 00 00 00 90 00 00	.data..."
00000210	00 08 00 00 00 7C 00 00	00 00 00 00 00 00 00 00
00000220	00 00 00 00 40 00 00 C0	2E 72 73 72 63 00 00 00@..À.rsrc...
00000230	58 89 00 00 00 B0 00 00	00 8A 00 00 00 84 00 00	X%...°...Š...//..
00000240	00 00 00 00 00 00 00 00	00 00 00 00 40 00 00 40@..@

Inspecting PE Header Information in Linux

```
1 import pefile
2 import sys
3
4 malware_file = sys.argv[1]
5 pe = pefile.PE(malware_file)
6 for section in pe.sections:
7     print "Name: %s VirtualSize: %s VirtualAddr: %s SizeofRawData: %s PointerToRawData: %s" %
8         (section.Name, hex(section.Misc_VirtualSize), hex(section.VirtualAddress), section.SizeOfRawData, section.PointerToRawData)
```

```
root@localhost ~# python display_sections.py a99c01d5748b1bfd203fc1763e6612e8
```

```
Name: .text VirtualSize: 0x7378 VirtualAddr: 0x1000 SizeofRawData: 29696 PointerToRawData: 1024
Name: .rdata VirtualSize: 0x261c VirtualAddr: 0x9000 SizeofRawData: 10240 PointerToRawData: 30720
Name: .data VirtualSize: 0x2cac VirtualAddr: 0xc000 SizeofRawData: 3584 PointerToRawData: 40960
Name: .rsrc VirtualSize: 0x1b4 VirtualAddr: 0xf000 SizeofRawData: 512 PointerToRawData: 44544
```

Inspecting PE Header Information

PEview - C:\WINDOWS\notepad.exe

File View Go Help

NOTEPAD.EXE

- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
 - Signature
 - IMAGE_FILE_HEADER
 - IMAGE_OPTIONAL_HEADER
 - IMAGE_SECTION_HEADER**
 - IMAGE_SECTION_HEADER
 - IMAGE_SECTION_HEADER
- SECTION .text
 - IMPORT Address Table
 - IMAGE_DEBUG_DIRECTORY
 - IMAGE_LOAD_CONFIG_DIRECTORY
 - IMAGE_DEBUG_TYPES
 - IMPORT Directory Table
 - IMPORT Name Table
 - IMPORT Hints/Name Table
- SECTION .data
- SECTION .rsrc
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY

pFile	Data	Description	Value
000001D8	2E 74 65 78	Name	.text
000001DC	74 00 00 00		
000001E0	00007748	Virtual Size	
000001E4	00001000	RVA	
000001E8	00007800	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	
			IMAGE_SCN_CNT_CODE
			IMAGE_SCN_MEM_EXECUTE
			IMAGE_SCN_MEM_READ

Viewing IMAGE_SECTION_HEADER .text

Inspecting file imports with pefile library

```
1  import pefile
2  import sys
3
4  malware_file = sys.argv[1]
5  pe = pefile.PE(malware_file)
6  if hasattr(pe, 'DIRECTORY_ENTRY_IMPORT'):
7      for entry in pe.DIRECTORY_ENTRY_IMPORT:
8          print "%s" % entry.dll
9          for imp in entry.imports:
10             if imp.name != None:
11                 print "\t%s" % (imp.name)
12             else:
13                 print "\tord(%s)" % (str(imp.ordinal))
14         print "\n"
```

Inspecting file export with pefile library

```
1  import pefile
2  import sys
3
4  malware_file = sys.argv[1]
5  pe = pefile.PE(malware_file)
6  if hasattr(pe, 'DIRECTORY_ENTRY_EXPORT'):
7      for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:
8          print "%s" % exp.name
9
```

Inspecting PE Header Information in Linux

```
1 import pefile
2 import sys
3
4 malware_file = sys.argv[1]
5 pe = pefile.PE(malware_file)
6 for section in pe.sections:
7     print "Name: %s VirtualSize: %s VirtualAddr: %s SizeofRawData: %s PointerToRawData: %s" %
8         (section.Name, hex(section.Misc_VirtualSize), hex(section.VirtualAddress), section.SizeOfRawData, section.PointerToRawData)
```

```
root@localhost ~# python display_sections.py a99c01d5748b1bfd203fc1763e6612e8
```

```
Name: .text VirtualSize: 0x7378 VirtualAddr: 0x1000 SizeofRawData: 29696 PointerToRawData: 1024
Name: .rdata VirtualSize: 0x261c VirtualAddr: 0x9000 SizeofRawData: 10240 PointerToRawData: 30720
Name: .data VirtualSize: 0x2cac VirtualAddr: 0xc000 SizeofRawData: 3584 PointerToRawData: 40960
Name: .rsrc VirtualSize: 0x1b4 VirtualAddr: 0xf000 SizeofRawData: 512 PointerToRawData: 44544
```

Inspecting PE Header Information

PEview - C:\WINDOWS\notepad.exe

File View Go Help

NOTEPAD.EXE

- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
 - Signature
 - IMAGE_FILE_HEADER
 - IMAGE_OPTIONAL_HEADER
 - IMAGE_SECTION_HEADER
 - IMAGE_SECTION_HEADER
 - IMAGE_SECTION_HEADER
- SECTION .text
 - IMPORT Address Table
 - IMAGE_DEBUG_DIRECTORY
 - IMAGE_LOAD_CONFIG_DIRECTORY
 - IMAGE_DEBUG_TYPES
 - IMPORT Directory Table
 - IMPORT Name Table
 - IMPORT Hints/Name Table
- SECTION .data
- SECTION .rsrc
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DATA_ENTRY

pFile	Data	Description	Value
000001D8	2E 74 65 78	Name	.text
000001DC	74 00 00 00		
000001E0	00007748	Virtual Size	
000001E4	00001000	RVA	
000001E8	00007800	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	
			IMAGE_SCN_CNT_CODE
			IMAGE_SCN_MEM_EXECUTE
			IMAGE_SCN_MEM_READ

Viewing IMAGE_SECTION_HEADER .text

Examining PE Section Table and Sections

- <https://hub.docker.com/r/remnux/pescanner/>

IAT (Import Address Table)

IAT (Import Address Table)

- Let's review the concept of DLL (Dynamic Link Library) **again**...

Dynamic Linking

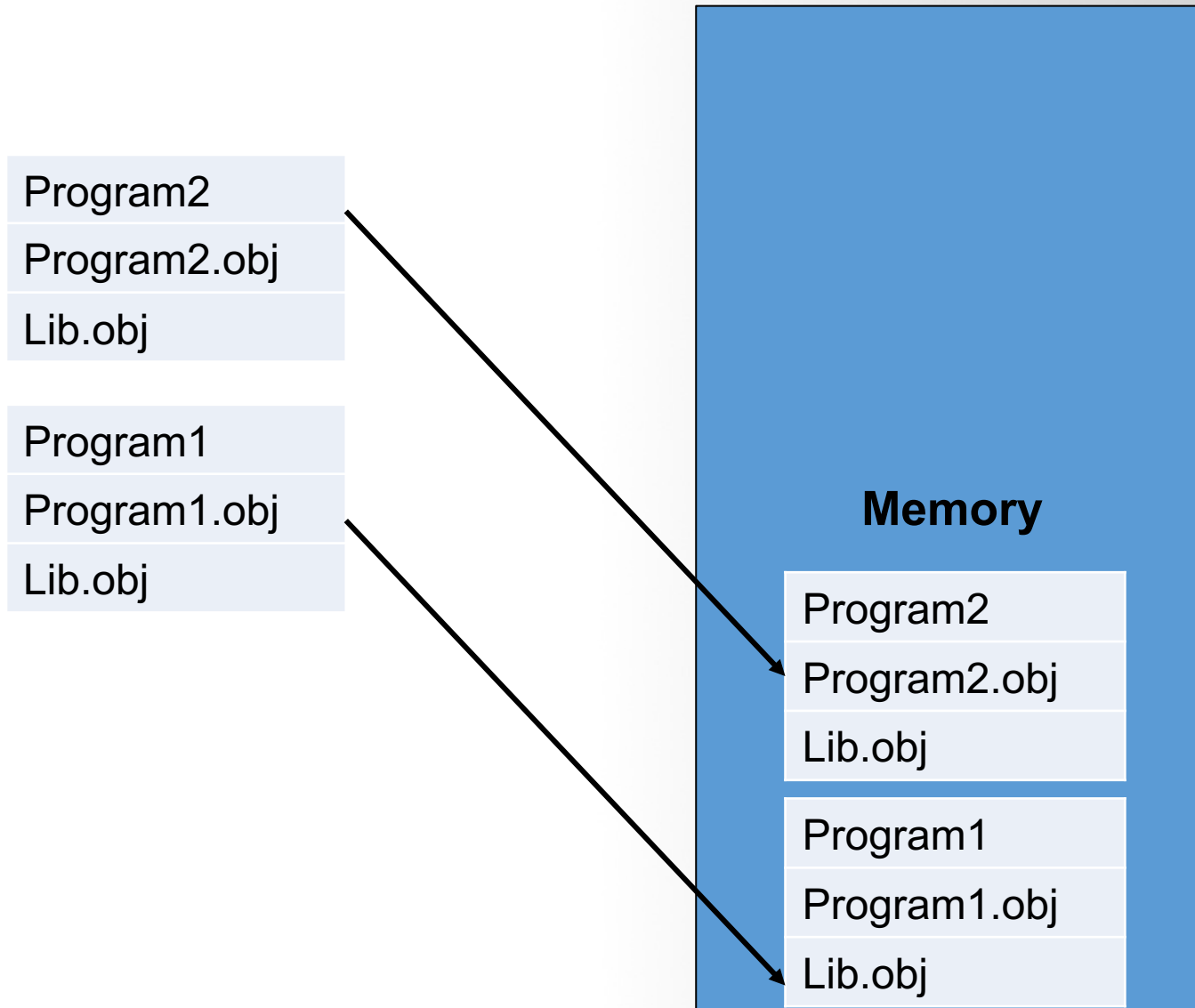
16-Bit DOS System

21. lab1.c (~) - VIM (ssh)

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void hacked()
5 {
6 >---/* change YOURNAME to your name :) */
7 >---puts("Hacked by YOURNAME!!!!");
8 }
9
10 void return_input(void)
11 {
12 >---/* Please set the array size equal to-
13 >--- the last two digits of your student ID
14 >--- e.g. 0861339 --> array size should set to 39 */
15 >---char array[39];--
16 >---gets(array);
17 >---printf("%s\n", array);
18 }
19
20 main()
21 {
22 >---return_input();
23 >---return 0;
24 }
```

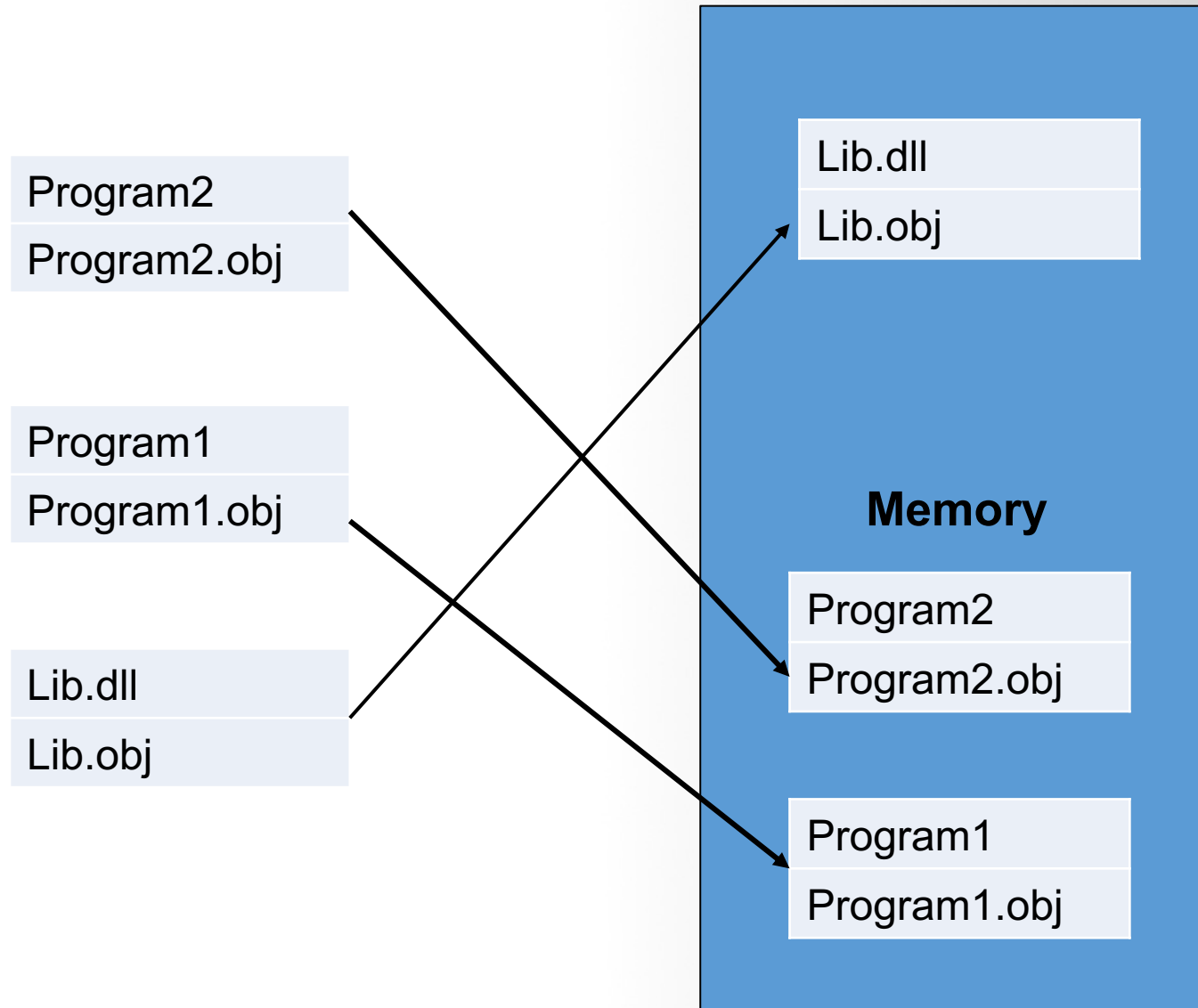
import Library → Put binary code of stdio library into the executable file

Static Linking



- Waste space
- Hard to maintain

Dynamic Linking



Dynamic linking has the following advantages:

- 1.Saves memory
- 2.Saves disk space.
- 3.Upgrades to the DLL are easier.
- 4.Provides after-market support.
- 5.Supports multi language programs.
- 6.Eases the creation of international versions

Two ways to Load DLL

Notepad.exe Process

.text

.data

.rsrc

kernel32.dll

user32.dll

gdi32.dll

shell32.dll

advapi32.dll

ntdll32.dll

Two ways to Load DLL

An executable file links to (or loads) a DLL in one of two ways:

- Explicit Linking (**run-time dynamic linking**)

- the executable using the DLL must make function calls to explicitly load and unload the DLL, and to access the DLL's exported functions.

1. Call LoadLibrary() (or a similar function) to load the DLL and obtain a module handle.
2. Call GetProcAddress() to obtain a function pointer to each exported function that the application wants to call.
3. Call FreeLibrary() when done with the DLL.

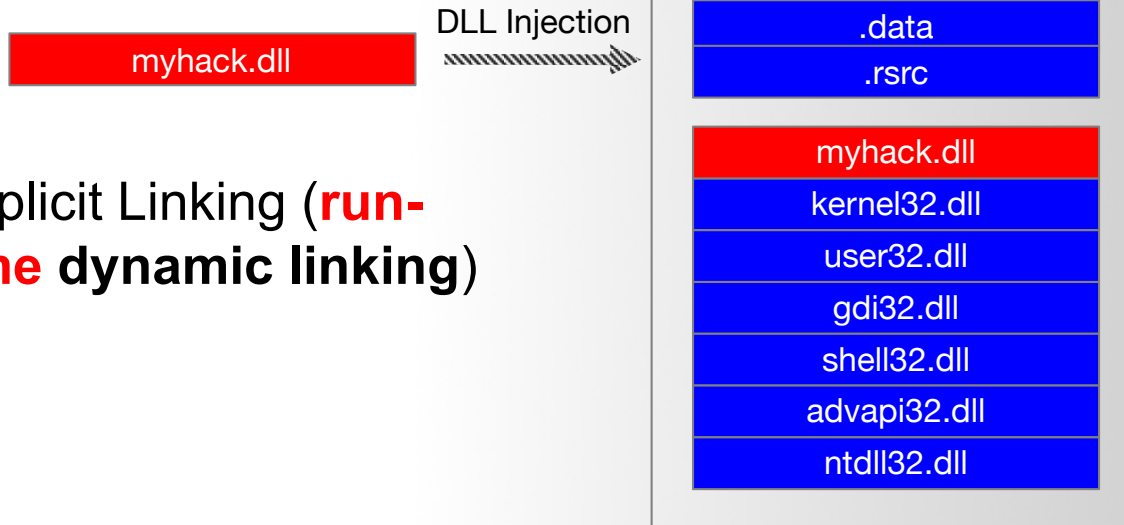
- Implicit Linking (**load-time dynamic linking**)

- The operating system loads the DLL when the executable using it is loaded.

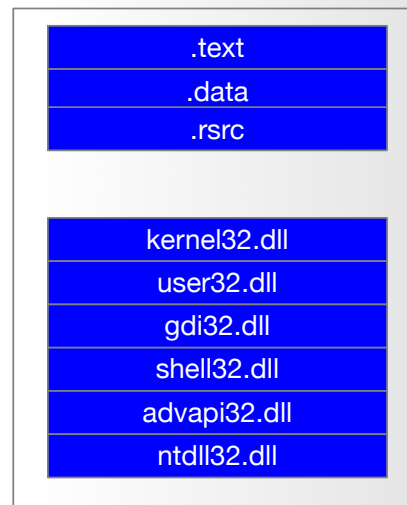
1. A header file (.H file) containing the declarations of the exported functions and/or C++ classes.
2. An import library (.LIB files) to link with. The linker creates the import library when the DLL is built.
3. The actual DLL (.DLL file).

Two ways to Load DLL

Explicit Linking (**run-time** dynamic linking)




Notepad.exe Process




Implicit Linking
(**load-time** dynamic linking)

Two ways to Load DLL

An executable file links to (or loads) a DLL in one of two ways:

- Explicit Linking (**run-time dynamic linking**)  **DLL Injection**
 - the executable using the DLL must make function calls to explicitly load and unload the DLL, and to access the DLL's exported functions.

1. Call LoadLibrary() (or a similar function) to load the DLL and obtain a module handle.
2. Call GetProcAddress() to obtain a function pointer to each exported function that the application wants to call.
3. Call FreeLibrary() when done with the DLL.

- Implicit Linking (**load-time dynamic linking**)  **IAT Table**
 - The operating system loads the DLL when the executable using it is loaded.

1. A header file (.H file) containing the declarations of the exported functions and/or C++ classes.
2. An import library (.LIB files) to link with. The linker creates the import library when the DLL is built.
3. The actual DLL (.DLL file).

Implicit Linking and IAT (Import Address Table)

- Notepad.exe Call CreateFileW() → Call 0x01001104 → Call 0x7C810CD9

OlllyDbg - NOTEPAD.EXE - [CPU - main thread, module NOTEPAD]

File View Debug Plugins Options Window Help

String2 = "*.txt"
String1
lstrcpyW
pOpenFileName = NOTEPAD.0100A680
UNICODE "txt"
UNICODE "NpEncodingDialog"
GetOpenFileNameW
hTemplateFile
Attributes = NORMAL
Mode = OPEN_EXISTING
pSecurity
ShareMode = FILE_SHARE_READ;FILE_SHARE_WRITE
Access = GENERIC_READ
FileName
CreateFileW
DS:[01001104]=7C810CD9 (kernel32.CreateFileW)

Registers (FPU)
EAX 00000000
ECX 0007FFB0
EDX 7C90E514 ntdll
EBX 7FFD7000
ESP 0007FFC4
EBP 0007FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll
EIP 01007390 NOTEPAD
C 0 ES 0023 32bit
P 1 CS 001B 32bit
A 0 SS 0023 32bit
Z 1 DS 0023 32bit
S 0 FS 003B 32bit
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR
EFL 00000246 (NO,NO)
ST0 empty -UNORM B
ST1 empty +UNORM B
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.000000
ST7 empty 1.000000
FST 4020 Cond 1 0
FCW 027F Prec NEAR

Address	Hex dump	ASCII
01001104	D9 0C 81 7C 36 AA 80 7C C0 99 80 7C 40 AE 80 7C	...U16-C!40C!0<C!
01001114	E3 5F 81 7C AA 14 81 7C 9C EE 80 7C 7D EE 80 7C	...U!-U!40C!0<C!
01001124	EC B7 80 7C 6C AA 80 7C 66 98 80 7C 8F BA 80 7C	...U!-U!40C!0<C!
01001134	5C 26 83 7C 21 FE 90 7C FF 12 81 7C 30 FE 90 7C	...U!-U!40C!0<C!
01001144	74 A1 80 7C 9C 09 83 7C 2F 4C 83 7C CC F6 82 7C	...U!-U!40C!0<C!
01001154	E6 20 83 7C D3 1F 83 7C B5 99 80 7C 14 BA 80 7C	...U!-U!40C!0<C!
01001164	98 9C 80 7C A5 1F 80 7C F2 4C 86 7C 00 00 00 00	...U!-U!40C!0<C!
01001174	38 7F A7 7C 06 29 A1 7C 23 B3 A2 7C A7 31 A6 7C	...U!-U!40C!0<C!
01001184	00 00 00 00 00 00 42 7E 30 99 42 7E 90 86 41 7E	...U!-U!40C!0<C!
01001194	C7 86 41 7E AB 47 42 7E 22 78 42 7E F6 98 42 7E	...U!-U!40C!0<C!
010011A4	20 8D 42 7E 9C B1 42 7E 7B 1F 43 7E 56 AF 42 7E	...U!-U!40C!0<C!
010011B4	23 98 42 7E FF 97 42 7E C7 03 43 7E D2 90 41 7E	...U!-U!40C!0<C!
010011C4	36 9E 41 7E 7F EE 42 7E 22 B2 42 7E 7F AF 41 7E	...U!-U!40C!0<C!
010011D4	97 7B 42 7E 69 9D 41 7E 46 DE 41 7E A3 D0 42 7E	...U!-U!40C!0<C!
010011E4	D2 01 42 7E C8 98 42 7E BC E8 42 7E 0E 96 42 7E	...U!-U!40C!0<C!
010011F4	5A CA 42 7E 34 AF 41 7E AB AE 42 7E 50 F7 42 7E	...U!-U!40C!0<C!
01001204	4C B2 42 7E 9B 92 41 7E 49 98 42 7E 15 B4 42 7E	...U!-U!40C!0<C!
01001214	38 5F 41 7E 0F 43 7E 9C 8F 41 7E 82 82 42 7E	...U!-U!40C!0<C!

Analysing NOTEPAD: 77 heuristical procedures, 516 calls to known, 151 calls to guessed functions

Paused

Implicit Linking and IAT (Import Address Table)

- Notepad.exe Call CreateFileW() → Call **0x01001104** → Call **0x7C810CD9**

Call **0x01001104**

↓ Look up

IAT Table

Function Name	IAT Address	Real Address
...		
CreateFileW()	0x01001104	0x7C810CD9
...		

When the application was first compiled, it was designed so that all API calls will **NOT use direct hardcoded addresses** but rather work through a function pointer.

This was accomplished through the use of **an import address table**. This is a **table of function pointers** filled in by the windows loader as the dlls are loaded.

Why IAT?

IAT (Import Address Table)

- Support different Windows Version (9X, 2K, XP, Vista, 7, 8, 10)

Call CreateFileW() --> Call **0x01001104**

↓ Look up

XP

IAT Table

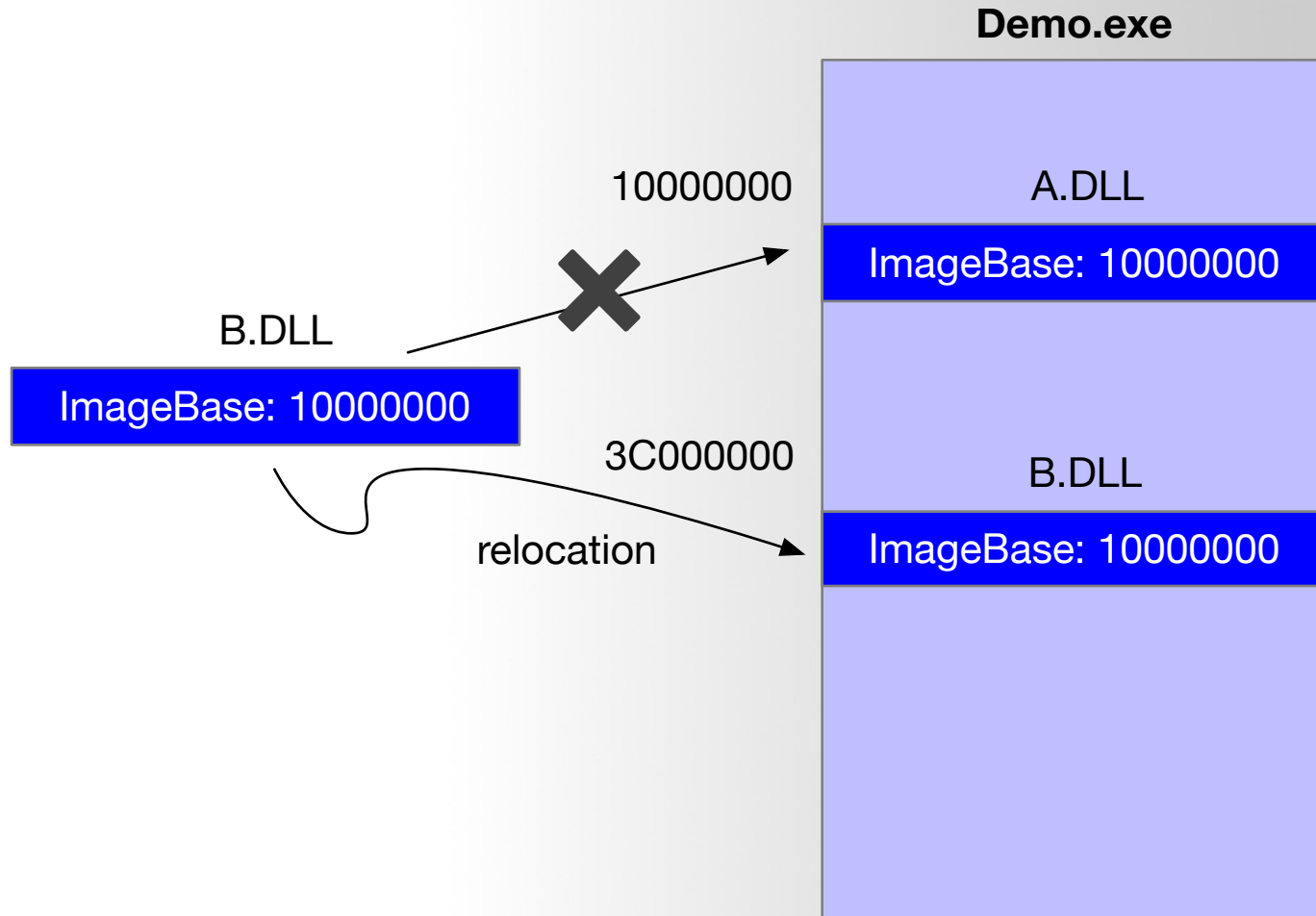
Function Name	IAT Address	Real Address
...		
CreateFileW()	0x01001104	0x7C810CD9
...		

Windows 7

Function Name	IAT Address	Real Address
...		
CreateFileW()	0x01001104	0x7C81FFFF
...		

IAT (Import Address Table)

▪ Support DLL Relocation



Look up IAT Table with PView

PView - C:\WINDOWS\NOTEPAD.EXE

File View Go Help

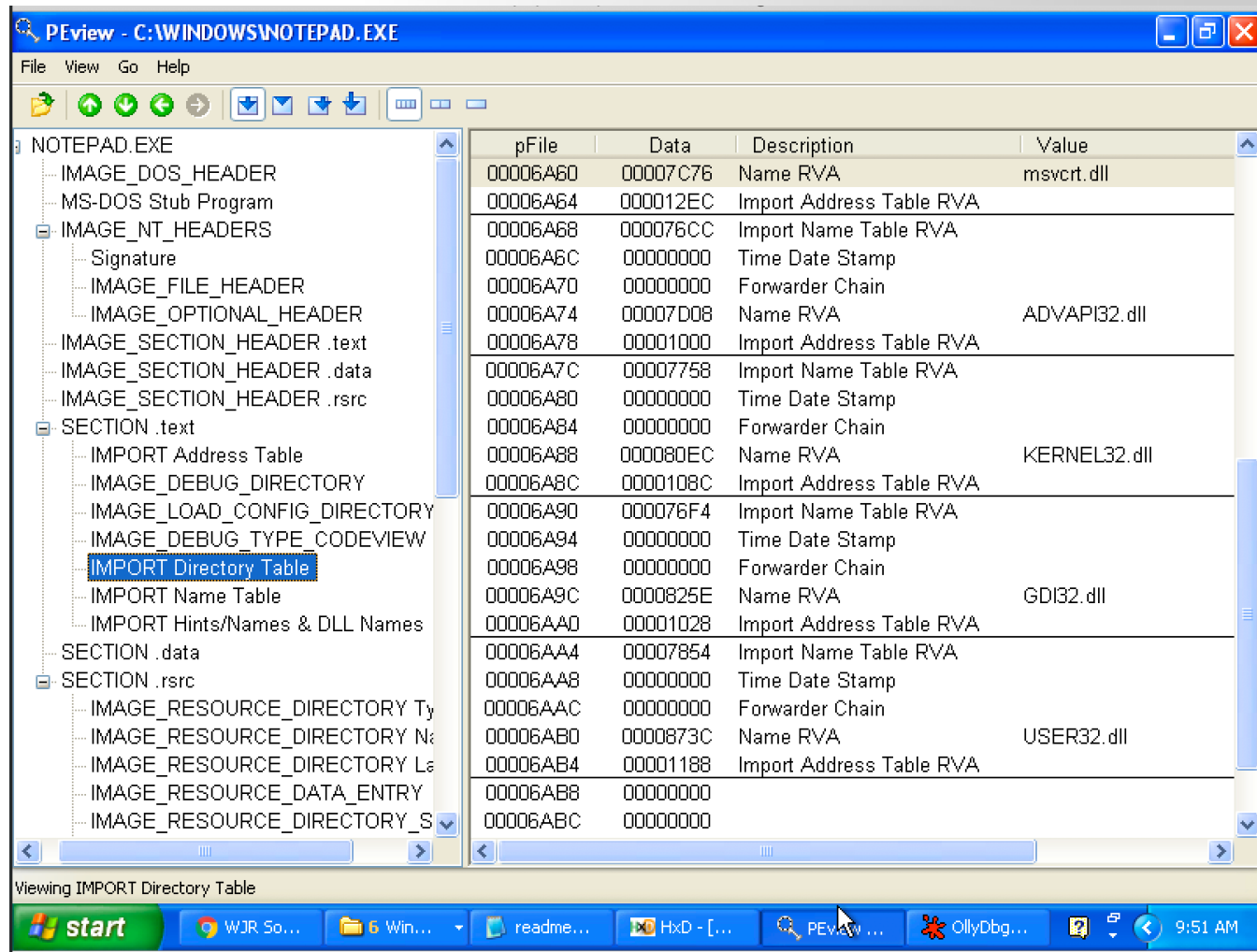
NOTEPAD.EXE

- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
 - Signature
 - IMAGE_FILE_HEADER
 - IMAGE_OPTIONAL_HEADER
 - IMAGE_SECTION_HEADER .text
 - IMAGE_SECTION_HEADER .data
 - IMAGE_SECTION_HEADER .rsrc
- SECTION .text
 - IMPORT Address Table
 - IMAGE_DEBUG_DIRECTORY
 - IMAGE_LOAD_CONFIG_DIRECTORY
 - IMAGE_DEBUG_TYPE_CODEVIEW
 - IMPORT Directory Table
 - IMPORT Name Table
 - IMPORT Hints/Names & DLL Names
- SECTION .data
- SECTION .rsrc
 - IMAGE_RESOURCE_DIRECTORY Ty
 - IMAGE_RESOURCE_DIRECTORY Na
 - IMAGE_RESOURCE_DIRECTORY La
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DIRECTORY_S

pFile	Data	Description	Value
000004E8	00007D5C	Hint/Name RVA	0254 LocalUnlock
000004EC	00007D6A	Hint/Name RVA	0038 CompareStringW
000004F0	00007D7C	Hint/Name RVA	0250 LocalLock
000004F4	00007D88	Hint/Name RVA	00EA FoldStringW
000004F8	00007D96	Hint/Name RVA	0031 CloseHandle
000004FC	00007DA4	Hint/Name RVA	03B3 lstrcpvW
00000500	00007DB0	Hint/Name RVA	02A6 ReadFile
00000504	00007DBC	Hint/Name RVA	0052 CreateFileW
00000508	00007DCA	Hint/Name RVA	03B0 lstrcmplW
0000050C	00007DD6	Hint/Name RVA	013C GetCurrentProcessId
00000510	00007DEC	Hint/Name RVA	0198 GetProcAddress
00000514	00007DFE	Hint/Name RVA	010A GetCommandLineW
00000518	00007E10	Hint/Name RVA	03AA lstrcatW
0000051C	00007E1C	Hint/Name RVA	00CC FindClose
00000520	00007E28	Hint/Name RVA	00D3 FindFirstFileW
00000524	00007E3A	Hint/Name RVA	0159 GetFileAttributesW
00000528	00007E50	Hint/Name RVA	03AD lstrcmpW
0000052C	00007E5C	Hint/Name RVA	0266 MulDiv
00000530	00007E66	Hint/Name RVA	03B6 lstrcpyW
00000534	00007E72	Hint/Name RVA	0253 LocalSize
00000538	00007E7E	Hint/Name RVA	0168 GetLastError
0000053C	00007E8E	Hint/Name RVA	0390 WriteFile
00000540	00007E9A	Hint/Name RVA	0316 SetLastError
00000544	00007EAA	Hint/Name RVA	0383 WideCharToMultiByte

Import Directory Table

- The Import Directory Table contains entries for every DLL which is loaded by the executable. Each entry contains, among other, Import Lookup Table (ILT) and **Import Address Table (IAT)**



PEView - C:\WINDOWS\notepad.exe

File View Go Help

NOTEPAD.EXE

- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
 - Signature
 - IMAGE_FILE_HEADER
 - IMAGE_OPTIONAL_HEADER
 - IMAGE_SECTION_HEADER .text
 - IMAGE_SECTION_HEADER .data
 - IMAGE_SECTION_HEADER .rsrc
- SECTION .text
 - IMPORT Address Table
 - IMAGE_DEBUG_DIRECTORY
 - IMAGE_LOAD_CONFIG_DIRECTORY
 - IMAGE_DEBUG_TYPE_CODEVIEW
 - IMPORT Directory Table**
 - IMPORT Name Table
 - IMPORT Hints/Names & DLL Names
- SECTION .data
- SECTION .rsrc
 - IMAGE_RESOURCE_DIRECTORY Ty
 - IMAGE_RESOURCE_DIRECTORY N
 - IMAGE_RESOURCE_DIRECTORY L
 - IMAGE_RESOURCE_DATA_ENTRY
 - IMAGE_RESOURCE_DIRECTORY_S

pFile	Data	Description	Value
00006A60	00007C76	Name RVA	msvcrt.dll
00006A64	000012EC	Import Address Table RVA	
00006A68	000076CC	Import Name Table RVA	
00006A6C	00000000	Time Date Stamp	
00006A70	00000000	Forwarder Chain	
00006A74	00007D08	Name RVA	ADVAPI32.dll
00006A78	00001000	Import Address Table RVA	
00006A7C	00007758	Import Name Table RVA	
00006A80	00000000	Time Date Stamp	
00006A84	00000000	Forwarder Chain	
00006A88	000080EC	Name RVA	KERNEL32.dll
00006A8C	0000108C	Import Address Table RVA	
00006A90	000076F4	Import Name Table RVA	
00006A94	00000000	Time Date Stamp	
00006A98	00000000	Forwarder Chain	
00006A9C	0000825E	Name RVA	GDI32.dll
00006AA0	00001028	Import Address Table RVA	
00006AA4	00007854	Import Name Table RVA	
00006AA8	00000000	Time Date Stamp	
00006AAC	00000000	Forwarder Chain	
00006AB0	0000873C	Name RVA	USER32.dll
00006AB4	00001188	Import Address Table RVA	
00006AB8	00000000		
00006ABC	00000000		

Viewing IMPORT Directory Table

start WJR So... 6 Win... readme... HxD - [...] PEView... OllyDbg... 9:51 AM

Inspecting file imports with pefile library

```
import pefile
import sys

malware_file = sys.argv[1]
pe = pefile.PE(malware_file)
if hasattr(pe, 'DIRECTORY_ENTRY_IMPORT'):
    for entry in pe.DIRECTORY_ENTRY_IMPORT:
        print "%s" % entry.dll
        for imp in entry.imports:
            if imp.name != None:
                print "\t%s \t%s" % (hex(imp.address), imp.name)
            else:
                print "\tord(%s)" % (str(imp.ordinal))
    print "\n"
```

0x1000c000	GetModuleFileNameW
0x1000c004	OutputDebugStringW
0x1000c008	CloseHandle
0x1000c00c	CreateThread
0x1000c010	WriteConsoleW
0x1000c014	CreateFileW
0x1000c018	UnhandledExceptionFilter
0x1000c01c	SetUnhandledExceptionFilter
0x1000c020	GetCurrentProcess
0x1000c024	TerminateProcess
0x1000c028	IsProcessorFeaturePresent
0x1000c02c	QueryPerformanceCounter
0x1000c030	GetCurrentProcessId
0x1000c034	GetCurrentThreadId
0x1000c038	GetSystemTimeAsFileTime
0x1000c03c	InitializeSListHead
0x1000c040	IsDebuggerPresent
0x1000c044	GetStartupInfoW
0x1000c048	GetModuleHandleW
0x1000c04c	InterlockedFlushSList
0x1000c050	RtlUnwind
0x1000c054	GetLastError
0x1000c058	SetLastError
0x1000c05c	EnterCriticalSection
0x1000c060	LeaveCriticalSection
0x1000c064	DeleteCriticalSection
0x1000c068	InitializeCriticalSectionAndSpinCount
0x1000c06c	TlsAlloc
0x1000c070	TlsGetValue
0x1000c074	TlsSetValue
0x1000c078	TlsFree
0x1000c07c	FreeLibrary
0x1000c080	GetProcAddress
0x1000c084	LoadLibraryExW
0x1000c088	RaiseException
0x1000c08c	ExitProcess
0x1000c090	GetModuleHandleExW
0x1000c094	HeapAlloc
0x1000c098	HeapFree
0x1000c09c	FindClose
0x1000c0a0	FindFirstFileExW
0x1000c0a4	FindNextFileW
0x1000c0a8	IsValidCodePage
0x1000c0ac	GetACP
0x1000c0b0	GetOEMCP
0x1000c0b4	GetCPInfo
0x1000c0b8	GetCommandLineA
0x1000c0bc	GetCommandLineW
0x1000c0c0	MultiByteToWideChar
0x1000c0c4	WideCharToMultiByte
0x1000c0c8	GetEnvironmentStringsW
0x1000c0cc	FreeEnvironmentStringsW
0x1000c0d0	GetStdHandle
0x1000c0d4	GetFileType
0x1000c0d8	LCMapStringW
0x1000c0dc	GetProcessHeap
0x1000c0e0	GetStringTypeW
0x1000c0e4	HeapSize
0x1000c0e8	HeapReAlloc
0x1000c0ec	SetStdHandle
0x1000c0f0	FlushFileBuffers
0x1000c0f4	WriteFile
0x1000c0f8	GetConsoleCP
0x1000c0fc	GetConsoleMode
0x1000c100	SetFilePointerEx
0x1000c104	DecodePointer

urlmon.dll
 0x1000c10c URLDownloadToFileW

Real-world Case Study

16d6b0e2c77da2776a88dd88c7cfc672

(Trojan.Win32.Dllhijack.a)

16d6b0e2c77da2776a88dd88c7cfc672

```
root@li254-249 ~ python enum_exports.py 16d6b0e2c77da2776a88dd88c7cfc672
0x100011e0 CreateDatabaseQueryObject 1
0x100011e0 DataImporterMain 2
0x100011e0 FlashboxMain 3
0x100010d0 KugouMain 4
```

KuGou

From Wikipedia, the free encyclopedia

KuGou (**Chinese:** 酷狗音乐) is a Chinese [music streaming](#) and [download](#) service established in 2004 and owned by [Tencent Music](#).^{[1][2]} It is the largest music streaming service in the world, with more than 450 million [monthly active users](#).^[2] KuGou is the largest online music service in China, with a market share of 28%.^[1] It has more than 800 million users.^[1] A merger between China Music Corporation and [Tencent's QQ Music](#) was announced on July 15, 2016.^{[1][3]} The services are expected to continue being offered separately.^[1] Together with Kuwo, another online music service also owned by Tencent Music and the third largest one in China,^[1] KuGou holds a music award ceremony, the [KU Music Asian Music Awards](#),^[4] also known as Cool Music Asia Festival Award.^[5]

References [\[edit \]](#)

1. ^ ^a ^b ^c ^d ^e ^f Zen Soo (July 15, 2016). "Tencent to merge QQ Music service with China Music Corp to create streaming giant" ^g. *South China Morning Post*. Retrieved August 20, 2016.
2. ^ ^a ^b Jubb, Nathan (October 19, 2016). "The Future of Music Streaming Lies in China's Small Cities" ^g. *Sixth Tone*. Retrieved October 27, 2016.
3. ^ Millward, Steven (July 15, 2016). "In China, 'Spotify' is free" ^g. *Tech In Asia*. Retrieved August 20, 2016.
4. ^ Kim Dong-Joo (March 31, 2016). "Kang Ta & SHINee garner awards at 'KU MUSIC ASIAN MUSIC AWARDS'" ^g. *sg.style.yahoo.com*. Retrieved August 20, 2016.
5. ^ "FTISLAND Wins "Asia's Popular Band" Award at Cool Music Asia Festival Award" ^g. *Soompi*. April 23, 2015. Retrieved August 20, 2016.

External links [[edit](#)]

- [Official website](#) (in Chinese)

KuGou



Developer(s)	Tencent Music
Initial release	2004; 15 years ago
Operating system	Android, iOS, Web, Windows
Type	Music streaming
Website	www.kugou.com

```
root@li254-249 ~$ python enum_exports.py 16d6b0e2c77da2776a88dd88c7cfc672
0x100011e0 CreateDatabaseQueryObject 1
0x100011e0 DataImporterMain 2
0x100011e0 FlashboxMain 3
0x100010d0 KugouMain 4
```

Dump of assembler code for function kugou!FlashboxMain:

```
0x100011e0 <+0>: xor    eax,eax
0x100011e2 <+2>: ret
0x100011e3 <+3>: nop
0x100011e4 <+4>: nop
0x100011e5 <+5>: nop
0x100011e6 <+6>: nop
0x100011e7 <+7>: nop
0x100011e8 <+8>: nop
0x100011e9 <+9>: nop
0x100011ea <+10>: nop
0x100011eb <+11>: nop
0x100011ec <+12>: nop
0x100011ed <+13>: nop
0x100011ee <+14>: nop
0x100011ef <+15>: nop
```


Incident Response

Risk Assessment

Remote Access Uses network protocols on unusual ports

Network Behavior Contacts 2 domains and 2 hosts. View the [network section](#) for more details.

- <https://www.hybrid-analysis.com/sample/037203d274cb66bad34559c0f426e9e1bf91a048155240581f4aa554be17925c?environmentId=100>

0fd6e3fb1cd5ec397ff3cdbaac39d80c

```

root@li254-249:~# python enum_exports.py 0fd6e3fb1cd5ec397ff3cdbaac39d80c
0x10002628 AheadLib_LpkPresent 36
0x10002634 AheadLib_ScriptApplyDigitSubstitution 37
0x10002640 AheadLib_ScriptApplyLogicalWidth 38
0x1000264c AheadLib_ScriptBreak 39
0x10002658 AheadLib_ScriptCPtoX 40
0x10002664 AheadLib_ScriptCacheGetHeight 41
0x10002670 AheadLib_ScriptFreeCache 42
0x1000267c AheadLib_ScriptGetCMap 43
0x10002688 AheadLib_ScriptGetFontProperties 44
0x10002694 AheadLib_ScriptGetGlyphABCWidth 45
0x1000271f AheadLib_ScriptGetLogicalWidths 46
0x1000272b AheadLib_ScriptGetProperties 47
0x10002737 AheadLib_ScriptIsComplex 48
0x10002743 AheadLib_ScriptItemize 49
0x10003091 AheadLib_ScriptJustify 50
0x1000309d AheadLib_ScriptLayout 51
0x100030a9 AheadLib_ScriptPlace 52
0x100030b5 AheadLib_ScriptRecordDigitSubstitution 53
0x100030c1 AheadLib_ScriptShape 54
0x100030cd AheadLib_ScriptStringAnalyse 55
0x100030d9 AheadLib_ScriptStringCPtoX 56
0x100030e5 AheadLib_ScriptStringFree 57
0x100030f1 AheadLib_ScriptStringGetLogicalWidths 58
0x100030fd AheadLib_ScriptStringGetOrder 59
0x10003109 AheadLib_ScriptStringOut 60
0x10003115 AheadLib_ScriptStringValidate 61
0x10003121 AheadLib_ScriptStringXtoCP 62
0x1000312d AheadLib_ScriptString_pLogAttr 63
0x10003139 AheadLib_ScriptString_pSize 64
0x10003145 AheadLib_ScriptString_pcOutChars 65
0x10003151 AheadLib_ScriptTextOut 66
0x1000315d AheadLib_ScriptXtoCP 67
0x10003169 AheadLib_UspAllocCache 68
0x10003175 AheadLib_UspAllocTemp 69
0x10003181 AheadLib_UspFreeMem 70
0x100023cf LpkDllInitialize 311
0x100023db LpkDrawTextEx 411
0x1001d040 LpkEditControl 71
0x100023f3 LpkExtTextOut 611
0x100023ff LpkGetCharacterPlacement 711
0x100025f3 LpkGetTextExtentExPoint 811
0x100023b7 LpkInitialize 111
0x10002604 LpkPSMTextOut 911
0x10002628 LpkPresent 1
0x100023c3 LpkTabbedTextOut 211
0x10002610 LpkUseGDIWidthCache 1011
0x100023cf MemCode_LpkDllInitialize 72
0x100023db MemCode_LpkDrawTextEx 73
0x100023e7 MemCode_LpkEditControl 74
0x100023f3 MemCode_LpkExtTextOut 75
0x100023ff MemCode_LpkGetCharacterPlacement 76
0x100025f3 MemCode_LpkGetTextExtentExPoint 77
0x100023b7 MemCode_LpkInitialize 78
0x10002604 MemCode_LpkPSMTextOut 79
0x100023c3 MemCode_LpkTabbedTextOut 80
0x10002610 MemCode_LpkUseGDIWidthCache 81
0x1000261c MemCode_ftsWordBreak 82
0x10002634 ScriptApplyDigitSubstitution 2
0x10002640 ScriptApplyLogicalWidth 3

```

```

gdb-peda$ disas ScriptBreak
Dump of assembler code for function drc!ScriptBreak:
0x1000264c <+0>:    push    0x1001d804
0x10002651 <+5>:    call   0x1000233d
0x10002656 <+10>:   jmp     eax
End of assembler dump.

```

```

gdb-peda$ disas LpkPresent
Dump of assembler code for function drc!LpkPresent:
0x10002628 <+0>:    push    0x1001d7c0
0x1000262d <+5>:    call   0x1000233d
0x10002632 <+10>:   jmp     eax
End of assembler dump.

```

6a764e4e6db461781d080034aab85aff
&
cc3c6c77e118a83ca0513c25c208832c

root@li254-249	python enum_exports.py 6a764e4e6db461781d80034aab85aff	root@li254-249	python enum_exports.py cc3c6c77e118a83ca0513c25c208832c
0x10004f00	AheadLib_ScriptApplyDigitSubstitution 36	0x10001100	LpkPresent 1
0x10004f10	AheadLib_ScriptApplyLogicalWidth 37	0x10001120	ScriptApplyDigitSubstitution 2
0x10004f20	AheadLib_ScriptBreak 38	0x10001140	ScriptApplyLogicalWidth 3
0x10004f30	AheadLib_ScriptCPToX 39	0x10001160	ScriptBreak 4
0x10004f40	AheadLib_ScriptCacheGetHeight 40	0x10001180	ScriptCPToX 5
0x10004f50	AheadLib_ScriptFreeCache 41	0x100011a0	ScriptCacheGetHeight 6
0x10004f60	AheadLib_ScriptGetCMap 42	0x100011c0	ScriptFreeCache 7
0x10004f70	AheadLib_ScriptGetFontProperties 43	0x100011e0	ScriptGetCMap 8
0x10004f80	AheadLib_ScriptGetGlyphABCWidth 44	0x10001200	ScriptGetFontProperties 9
0x10004f90	AheadLib_ScriptGetLogicalWidths 45	0x10001220	ScriptGetGlyphABCWidth 10
0x10004fa0	AheadLib_ScriptGetProperties 46	0x10001240	ScriptGetLogicalWidths 11
0x10004fb0	AheadLib_ScriptIsComplex 47	0x10001260	ScriptGetProperties 12
0x10004fc0	AheadLib_ScriptItemize 48	0x10001280	ScriptIsComplex 13
0x10004fd0	AheadLib_ScriptJustify 49	0x100012a0	ScriptItemize 14
0x10004fe0	AheadLib_ScriptLayout 50	0x100012c0	ScriptJustify 15
0x10004ff0	AheadLib_ScriptPlace 51	0x100012e0	ScriptLayout 16
0x10005000	AheadLib_ScriptRecordDigitSubstitution 52	0x10001300	ScriptPlace 17
0x10005010	AheadLib_ScriptShape 53	0x10001320	ScriptRecordDigitSubstitution 18
0x10005020	AheadLib_ScriptStringAnalyse 54	0x10001340	ScriptShape 19
0x10005030	AheadLib_ScriptStringCPToX 55	0x10001360	ScriptStringAnalyse 20
0x10005040	AheadLib_ScriptStringFree 56	0x10001380	ScriptStringCPToX 21
0x10005050	AheadLib_ScriptStringGetLogicalWidths 57	0x100013a0	ScriptStringFree 22
0x10005060	AheadLib_ScriptStringGetOrder 58	0x100013c0	ScriptStringGetLogicalWidths 23
0x10005070	AheadLib_ScriptStringOut 59	0x100013e0	ScriptStringGetOrder 24
0x10005080	AheadLib_ScriptStringValidate 60	0x10001400	ScriptStringOut 25
0x10005090	AheadLib_ScriptStringXtoCP 61	0x10001420	ScriptStringValidate 26
0x100050a0	AheadLib_ScriptString_pLogAttr 62	0x10001440	ScriptStringXtoCP 27
0x100050b0	AheadLib_ScriptString_pSize 63	0x10001460	ScriptString_pLogAttr 28
0x100050c0	AheadLib_ScriptString_pcOutChars 64	0x10001480	ScriptString_pSize 29
0x100050d0	AheadLib_ScriptTextOut 65	0x100014a0	ScriptString_pcOutChars 30
0x100050e0	AheadLib_ScriptXtoCP 66	0x100014c0	ScriptTextOut 31
0x100050f0	AheadLib_UspAllocCache 67	0x100014e0	ScriptXtoCP 32
0x10005100	AheadLib_UspAllocTemp 68	0x10001890	ServiceMain 36
0x10005110	AheadLib_UspFreeMem 69	0x10001500	UspAllocCache 33
0x10004ef0	AheadLib_mmLpkPresent 70	0x10001520	UspAllocTemp 34
0x10004e50	LpkDllInitialize 311	0x10001540	UspFreeMem 35
0x10004e60	LpkDrawTextEx 411		
0x1000e92c	LpkEditControl 71		
0x10004e80	LpkExtTextOut 611		
0x10004e90	LpkGetCharacterPlacement 711		
0x10004ea0	LpkGetTextExtentExPoint 811		
0x10004e30	LpkInitialize 111		
0x10004ec0	LpkPSMTextOut 911		
0x10004ef0	LpkPresent 1		
0x10004e40	LpkTabbedTextOut 211		
0x10004ed0	LpkUseGDIWidthCache 1011		
0x10004e50	MemCode_LpkDllInitialize 72		
0x10004e60	MemCode_LpkDrawTextEx 73		
0x10004e70	MemCode_LpkEditControl 74		
0x10004e80	MemCode_LpkExtTextOut 75		
0x10004e90	MemCode_LpkGetCharacterPlacement 76		
0x10004ea0	MemCode_LpkGetTextExtentExPoint 77		
0x10004e30	MemCode_LpkInitialize 78		
0x10004ec0	MemCode_LpkPSMTextOut 79		
0x10004e40	MemCode_LpkTabbedTextOut 80		
0x10004ed0	MemCode_LpkUseGDIWidthCache 81		
0x10004ee0	MemCode_ftsWordBreak 82		
0x10004f00	ScriptApplyDigitSubstitution 2		
0x10004f10	ScriptApplyLogicalWidth 3		

e0bed0b33e7b6183f654f0944b607618

```
root@li254-249 ~$ python enum_exports.py e0bed0b33e7b6183f654f0944b607618
0x100165f0      LsaApCallPackage          1
0x10016610      LsaApCallPackagePassthrough 2
0x10016600      LsaApCallPackageUntrusted 3
0x100165e0      LsaApInitializePackage    4
0x10016620      LsaApLogonTerminated      5
0x10016470      LsaApLogonUserEx2         6
0x10016560      SpInitialize              7
0x100165d0      SpInstanceInit            8
0x10016570      SpLsaModeInitialize       9
0x100165c0      SpUserModeInitialize     10
```

db8199eeb2d75e789df72cd8852a9fbb

(Rootkit.Win32.blackken.b)


```
root@li254-249 ~$ python enum_exports.py db8199eeb2d75e789df72cd8852a9fbb
0x10006707 ?Start@@YGKPAX@Z 1
0x10006707 MakeCache 2
```

Is this claim correct?

If two export functions share the same address, it's a malware.

1c1131112db91382b9d8b46115045097

1c1131112db91382b9d8b46115045097

```
root@li254-249 ~ ➤ python enum_exports.py 1c1131112db91382b9d8b46115045097
0x100014a0 AfxGetHttpRequestMgr 3
0x100014b0 AfxGetHttpRequestMgr 4
0x10001490 InitInstance 2
0x10001490 MessageLoop 1
```

EAT (Export Address Table)

- Similar to IAT, EAT data is stored in IMAGE_EXPORT_DIRECTORY
- EAT contains an RVA that points to an array of pointers to **(RVAs of) the functions in the module.**

- Create your own anti-malware system based on heuristic analysis.
- Check course website