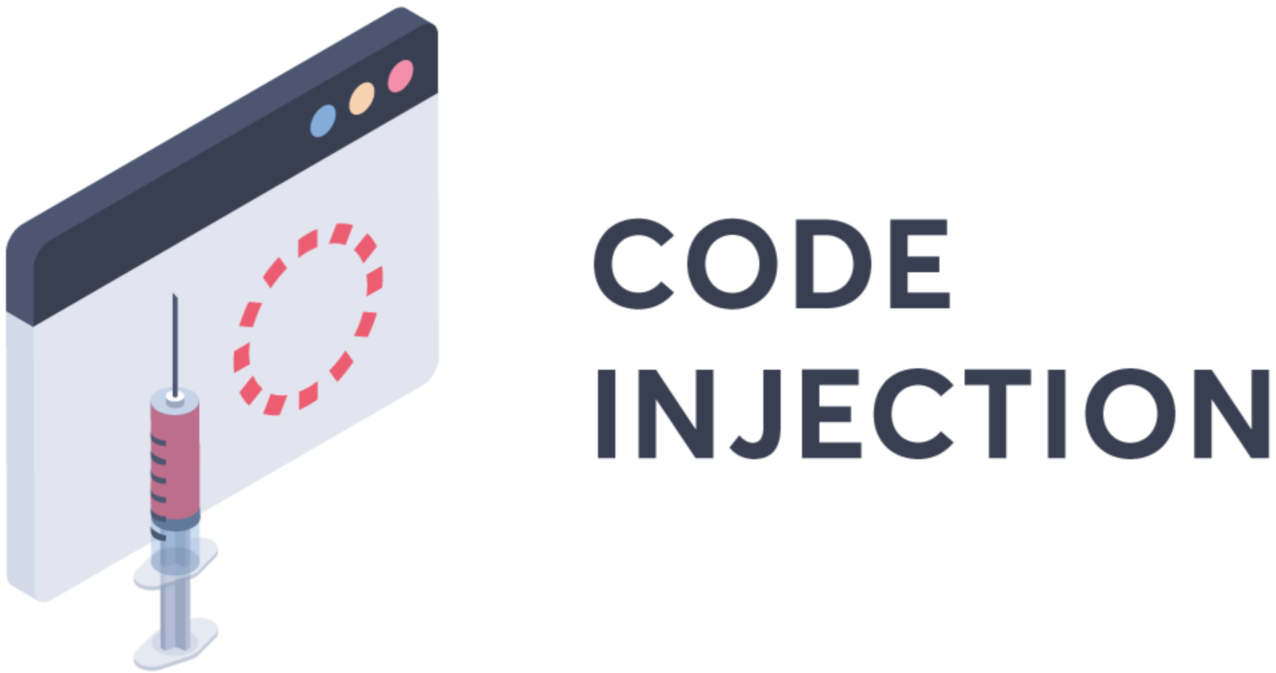
Class 13

CSC 471 Modern Malware Analysis
Code Injection (2)

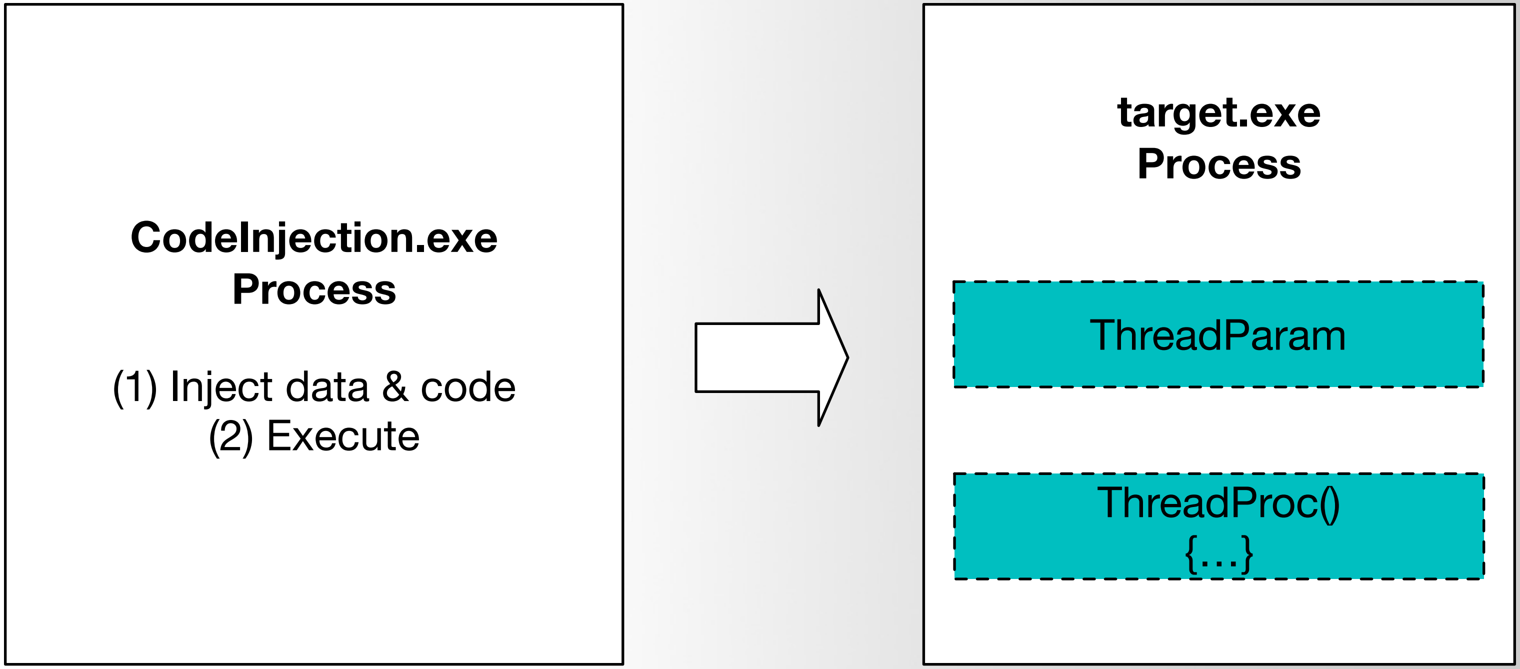
Si Chen (schen@wcupa.edu)

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)



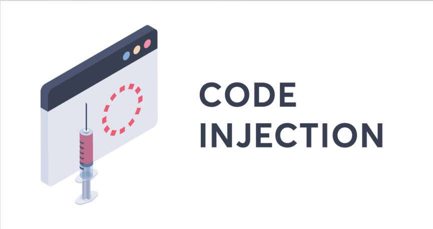
code → injected by ThreadProc()

data → injected as ThreadParam

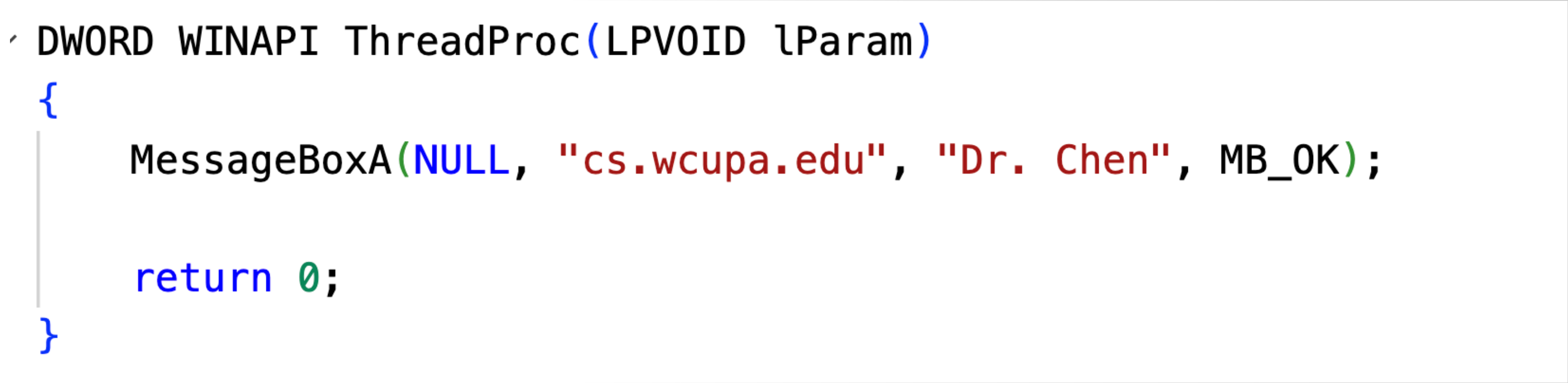
Why Code Injection

- 1. **Use less memory** → you don't need to compile it as DLL
- 2. **Hard to detect** → DLL injection can easily be spotted, code injection is very sneaky.

- In short:
  - **DLL injection** is for huge code base and complex logic.
  - **Code injection** is for small code base with simple logic.



DLL Injection V.S. Code Injection

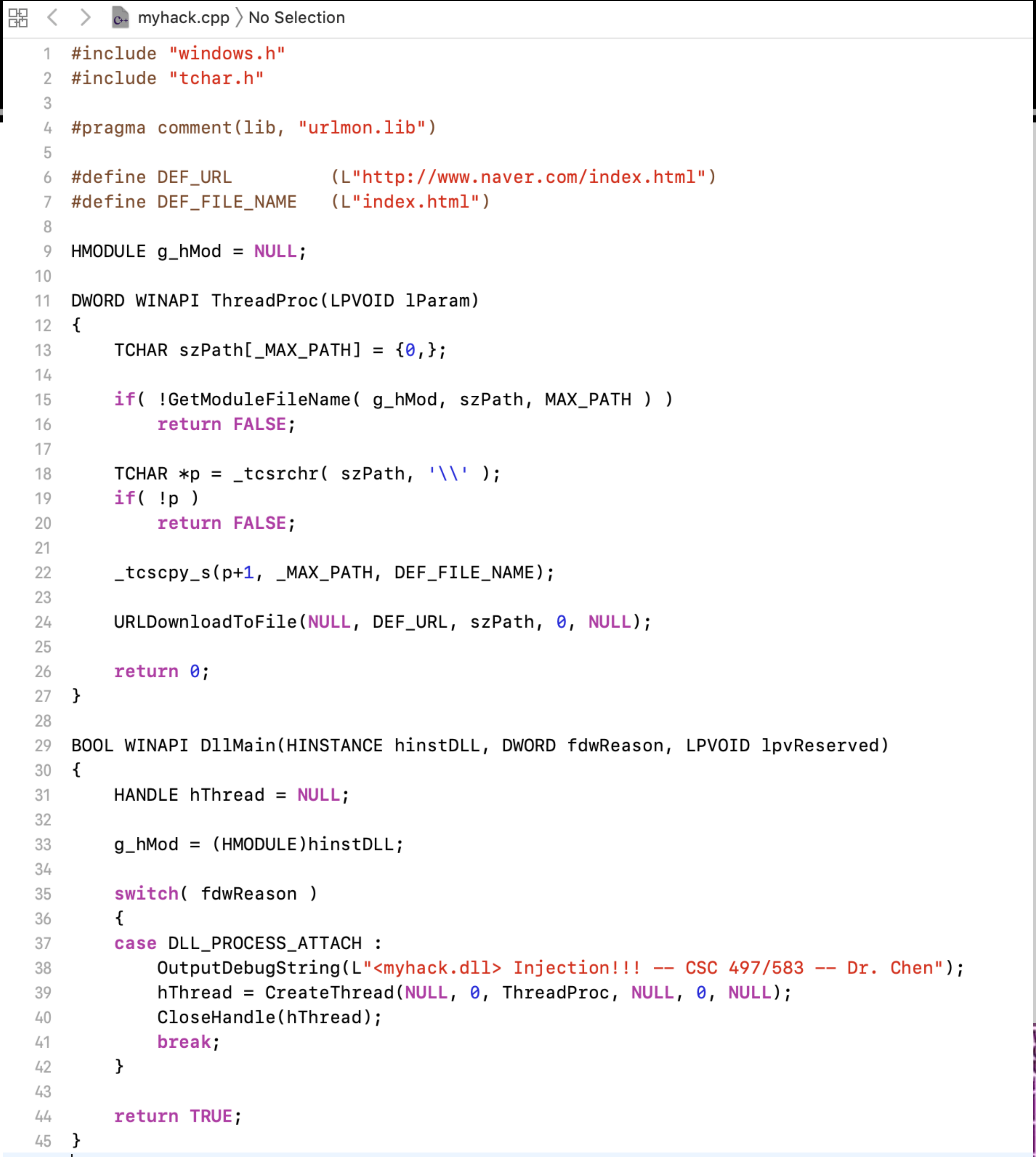


```
DWORD WINAPI ThreadProc(LPVOID lParam)
{
    MessageBoxA(NULL, "cs.wcupa.edu", "Dr. Chen", MB_OK);

    return 0;
}
```

Pop up a Windows message box

How to use DLL Injection to injection the code?

myhack.cpp

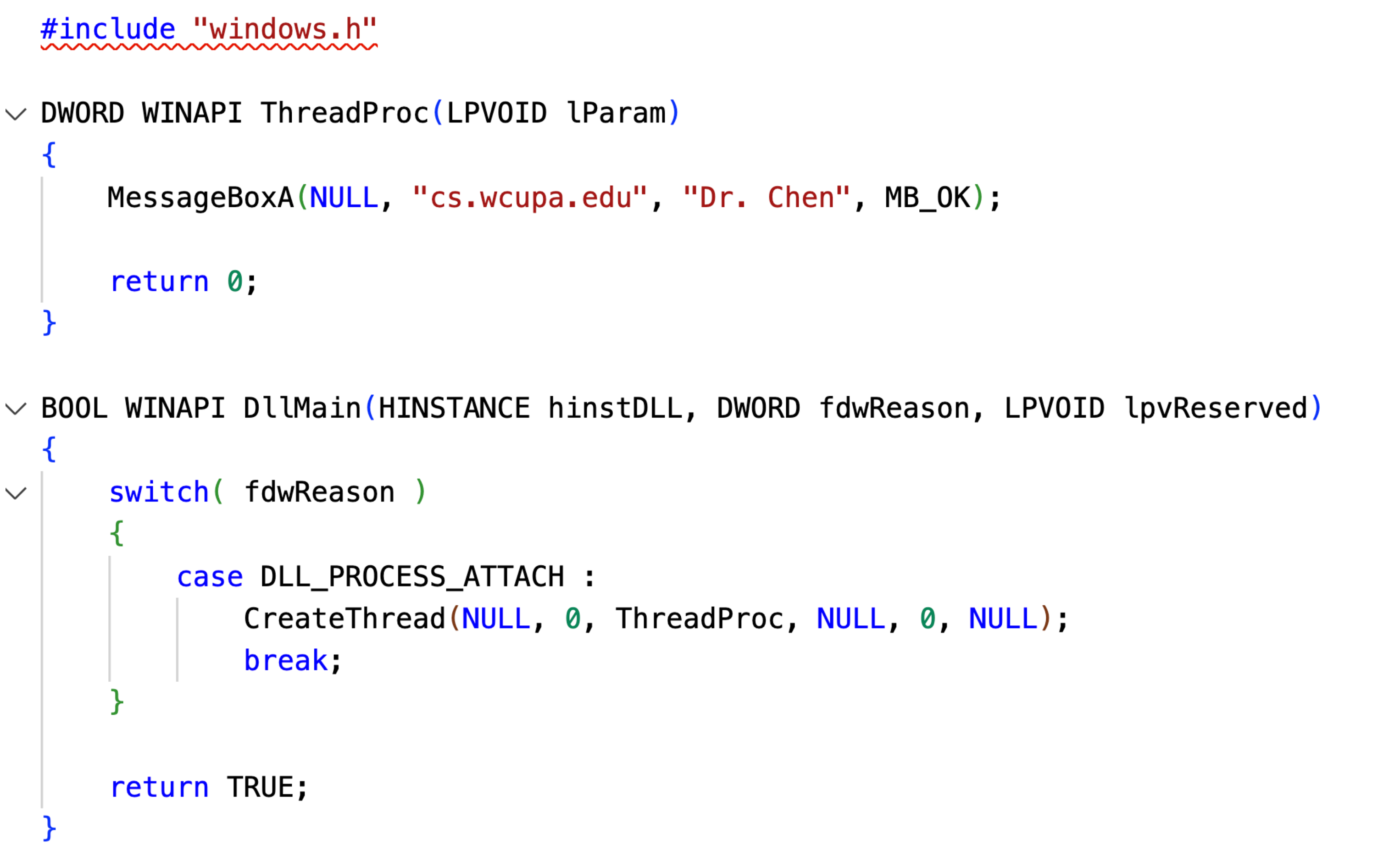


```cpp
#include "windows.h"
#include "tchar.h"

#pragma comment(lib, "urlmon.lib")

#define DEF_URL         (L"http://www.naver.com/index.html")
#define DEF_FILE_NAME   (L"index.html")

HMODULE g_hMod = NULL;

DWORD WINAPI ThreadProc(LPVOID lParam)
{
    TCHAR szPath[_MAX_PATH] = {0,};

    if( !GetModuleFileName( g_hMod, szPath, MAX_PATH ) )
        return FALSE;

    TCHAR *p = _tcsrchr( szPath, '\\' );
    if( !p )
        return FALSE;

    _tcscpy_s(p+1, _MAX_PATH, DEF_FILE_NAME);

    URLDownloadToFile(NULL, DEF_URL, szPath, 0, NULL);

    return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    HANDLE hThread = NULL;

    g_hMod = (HMODULE)hinstDLL;

    switch( fdwReason )
    {
    case DLL_PROCESS_ATTACH :
        OutputDebugString(L"<myhack.dll> Injection!!! -- CSC 497/583 -- Dr. Chen");
        hThread = CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
        CloseHandle(hThread);
        break;
    }

    return TRUE;
}
```

DLL Injection V.S. Code Injection

https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-messagebox

How to use DLL Injection to injection the code?

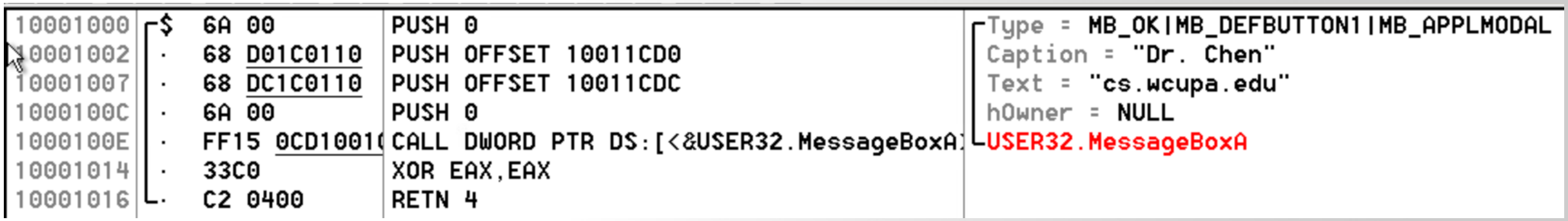


```
#include "windows.h"

DWORD WINAPI ThreadProc(LPVOID lParam)
{
    MessageBoxA(NULL, "cs.wcupa.edu", "Dr. Chen", MB_OK);

    return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    switch( fdwReason )
    {
        case DLL_PROCESS_ATTACH :
            CreateThread(NULL, 0, ThreadProc, NULL, 0, NULL);
            break;
    }

    return TRUE;
}
```

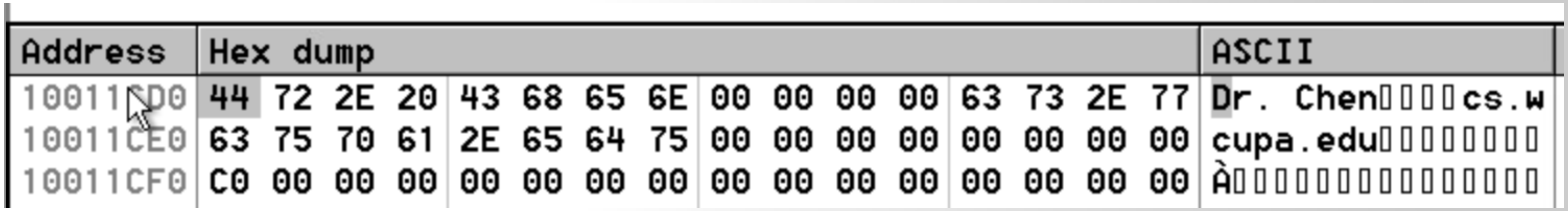
Compile it as MsgBox.dll and inject it to the target process

same as DLL injection lab!

DLL Injection (MsgBox.dll)

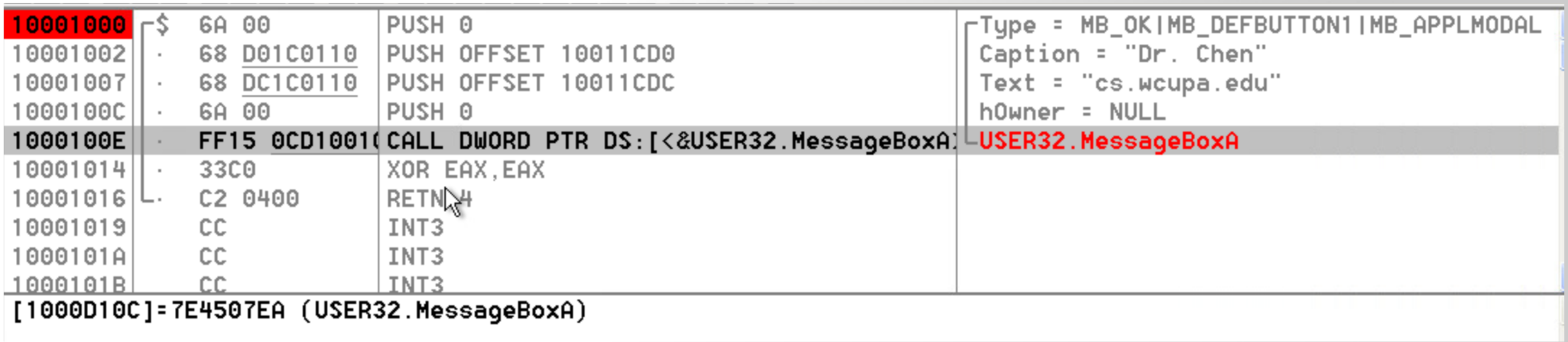


```
10001000 ┌$  6A 00          PUSH 0                                    ┌Type = MB_OK|MB_DEFBUTTON1|MB_APPLMODAL
10001002 │.  68 D01C0110    PUSH OFFSET 10011CD0                      │Caption = "Dr. Chen"
10001007 │.  68 DC1C0110    PUSH OFFSET 10011CDC                      │Text = "cs.wcupa.edu"
1000100C │.  6A 00          PUSH 0                                    │hOwner = NULL
1000100E │.  FF15 0CD1001(  CALL DWORD PTR DS:[<&USER32.MessageBoxA] └USER32.MessageBoxA
10001014 │.  33C0           XOR EAX,EAX
10001016 └.  C2 0400        RETN 4
```

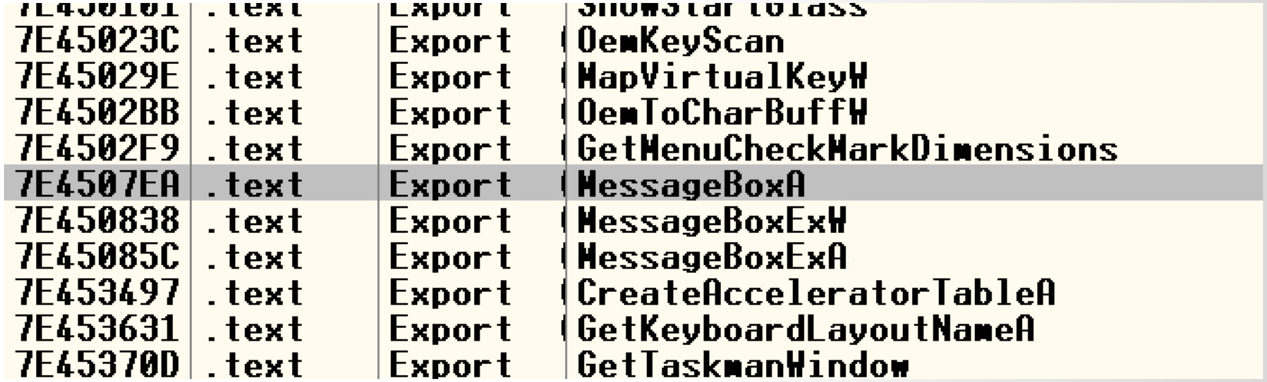


```
Address   Hex dump                                          ASCII
10011CD0  44 72 2E 20 43 68 65 6E 00 00 00 00 63 73 2E 77  Dr. Chen□□□□cs.w
10011CE0  63 75 70 61 2E 65 64 75 00 00 00 00 00 00 00 00  cupa.edu□□□□□□□□
10011CF0  C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  À□□□□□□□□□□□□□□□
```

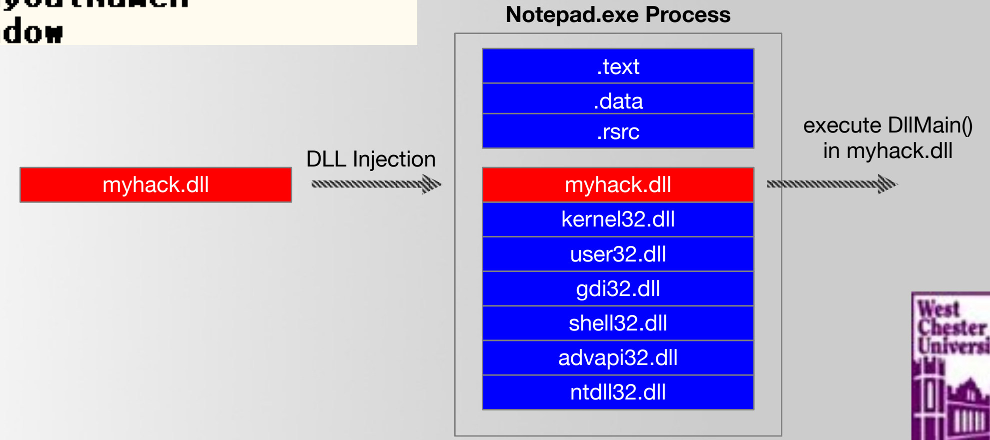
DLL Injection (MsgBox.dll)



```
10001000  $  6A 00           PUSH 0                                  Type = MB_OK|MB_DEFBUTTON1|MB_APPLMODAL
10001002  .  68 D01C0110     PUSH OFFSET 10011CD0                    Caption = "Dr. Chen"
10001007  .  68 DC1C0110     PUSH OFFSET 10011CDC                    Text = "cs.wcupa.edu"
1000100C  .  6A 00           PUSH 0                                  hOwner = NULL
1000100E  .  FF15 0CD1001(   CALL DWORD PTR DS:[<&USER32.MessageBoxA]  USER32.MessageBoxA
10001014  .  33C0            XOR EAX,EAX
10001016  .  C2 0400         RETN 4
10001019     CC              INT3
1000101A     CC              INT3
1000101B     CC              INT3
[1000D10C]=7E4507EA (USER32.MessageBoxA)
```

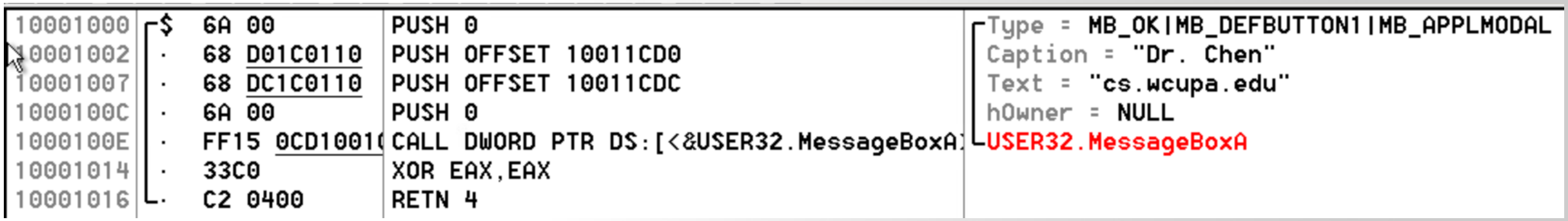


```
7E450161  .text  Export  ShowStartGlass
7E45023C  .text  Export  OemKeyScan
7E45029E  .text  Export  MapVirtualKeyW
7E4502BB  .text  Export  OemToCharBuffW
7E4502F9  .text  Export  GetMenuCheckMarkDimensions
7E4507EA  .text  Export  MessageBoxA
7E450838  .text  Export  MessageBoxExW
7E45085C  .text  Export  MessageBoxExA
7E453497  .text  Export  CreateAcceleratorTableA
7E453631  .text  Export  GetKeyboardLayoutNameA
7E45370D  .text  Export  GetTaskmanWindow
```



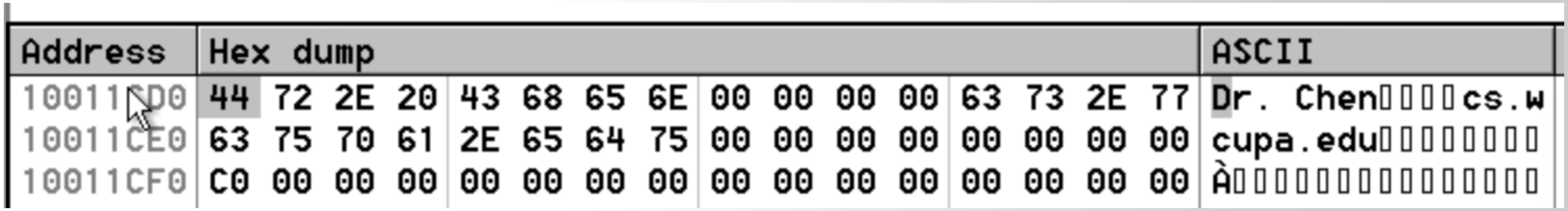
Code Injection

You need to inject the code

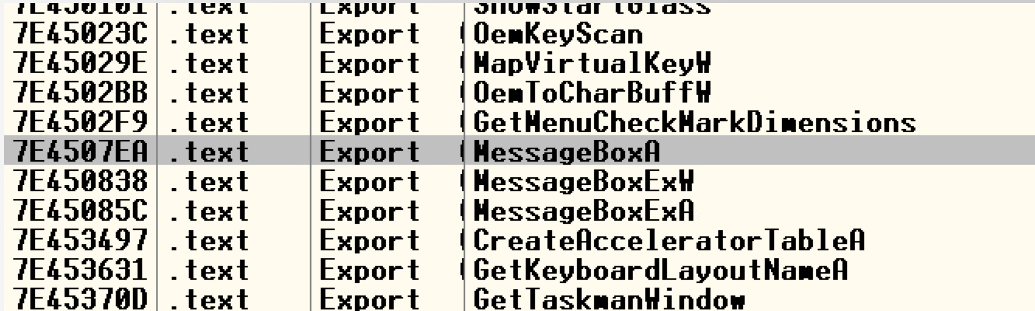


```
10001000  ┌$  6A 00          PUSH 0                                     ┌Type = MB_OK|MB_DEFBUTTON1|MB_APPLMODAL
10001002  │.  68 D01C0110    PUSH OFFSET 10011CD0                       │Caption = "Dr. Chen"
10001007  │.  68 DC1C0110    PUSH OFFSET 10011CDC                       │Text = "cs.wcupa.edu"
1000100C  │.  6A 00          PUSH 0                                     │hOwner = NULL
1000100E  │.  FF15 0CD1001(  CALL DWORD PTR DS:[<&USER32.MessageBoxA)   └USER32.MessageBoxA
10001014  │.  33C0           XOR EAX,EAX
10001016  └.  C2 0400        RETN 4
```

And the data:

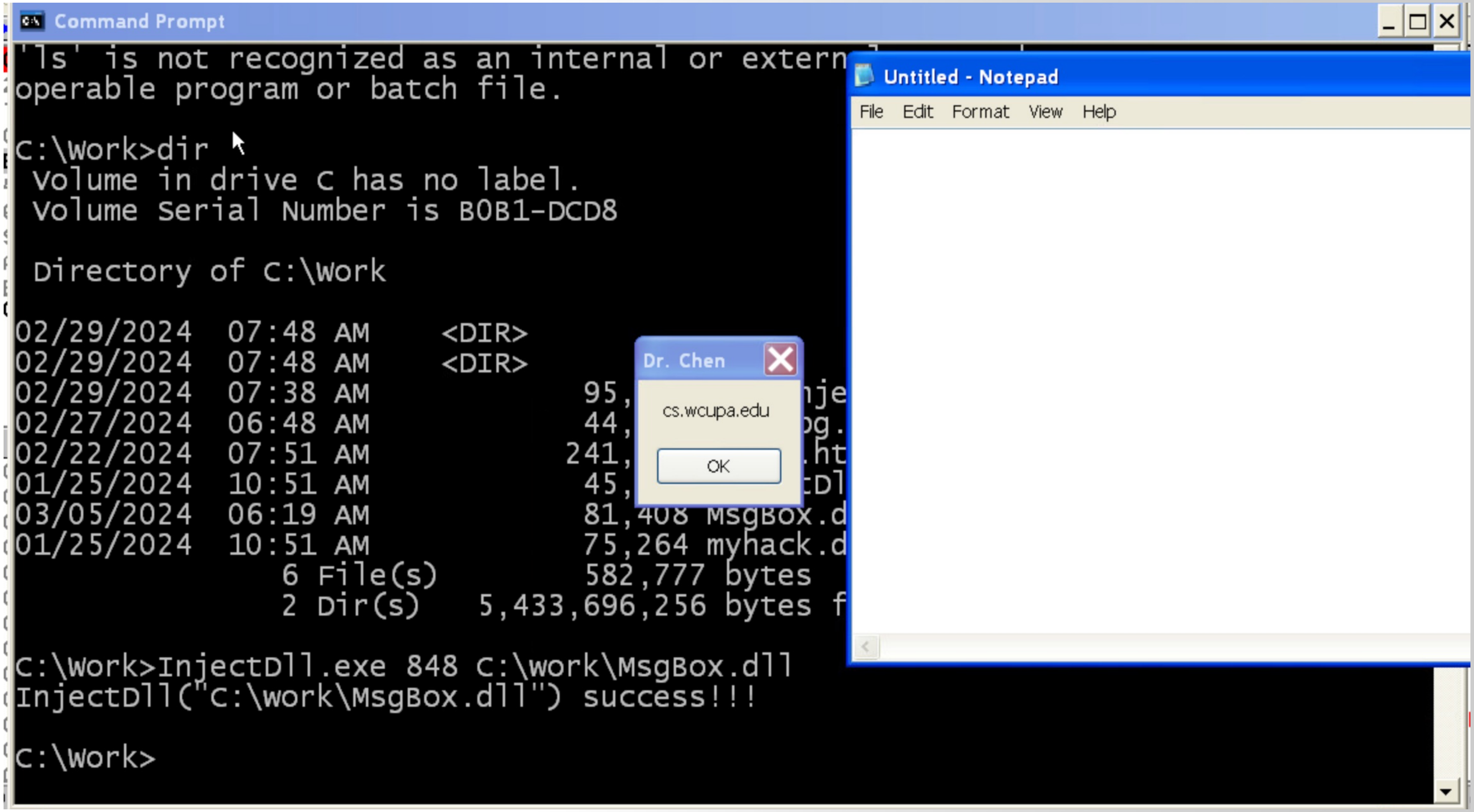


```
Address   Hex dump                                          ASCII
10011CD0  44 72 2E 20 43 68 65 6E 00 00 00 00 63 73 2E 77   Dr. Chen□□□□cs.w
10011CE0  63 75 70 61 2E 65 64 75 00 00 00 00 00 00 00 00   cupa.edu□□□□□□□□
10011CF0  C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   À□□□□□□□□□□□□□□□
```



```
7E45023C  .text  Export  OemKeyScan
7E45029E  .text  Export  MapVirtualKeyW
7E4502BB  .text  Export  OemToCharBuffW
7E4502F9  .text  Export  GetMenuCheckMarkDimensions
7E4507EA  .text  Export  MessageBoxA
7E450838  .text  Export  MessageBoxExW
7E45085C  .text  Export  MessageBoxExA
7E453497  .text  Export  CreateAcceleratorTableA
7E453631  .text  Export  GetKeyboardLayoutNameA
7E45370D  .text  Export  GetTaskmanWindow
```

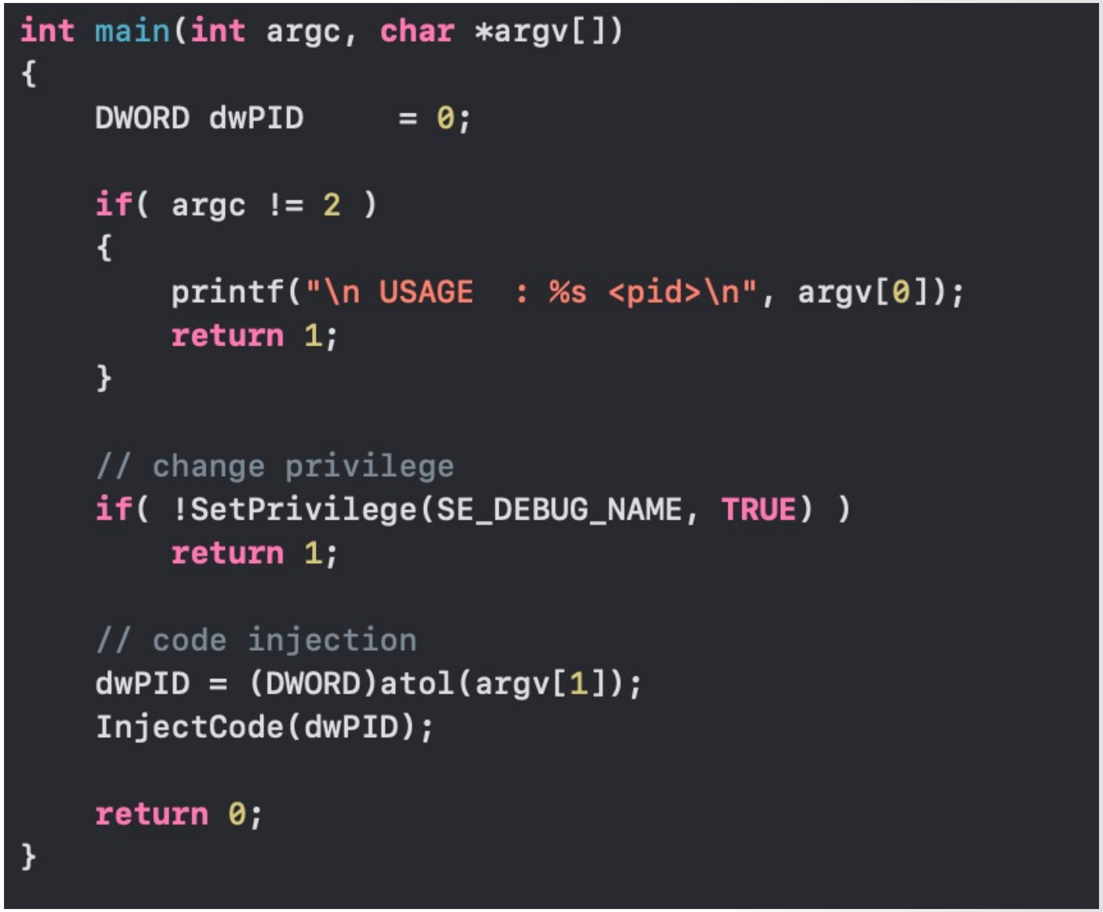
Code Injection Example (CodeInjection.exe)



```
'ls' is not recognized as an internal or extern
operable program or batch file.

C:\Work>dir
 Volume in drive C has no label.
 Volume Serial Number is B0B1-DCD8

 Directory of C:\Work

02/29/2024  07:48 AM    <DIR>
02/29/2024  07:48 AM    <DIR>
02/29/2024  07:38 AM            95,        nje
02/27/2024  06:48 AM            44,        og.
02/22/2024  07:51 AM           241,        .ht
01/25/2024  10:51 AM            45,        :Dl
03/05/2024  06:19 AM            81,408 MsgBox.d
01/25/2024  10:51 AM            75,264 myhack.d
               6 File(s)        582,777 bytes
               2 Dir(s)   5,433,696,256 bytes f

C:\Work>InjectDll.exe 848 C:\work\MsgBox.dll
InjectDll("C:\work\MsgBox.dll") success!!!

C:\Work>
```

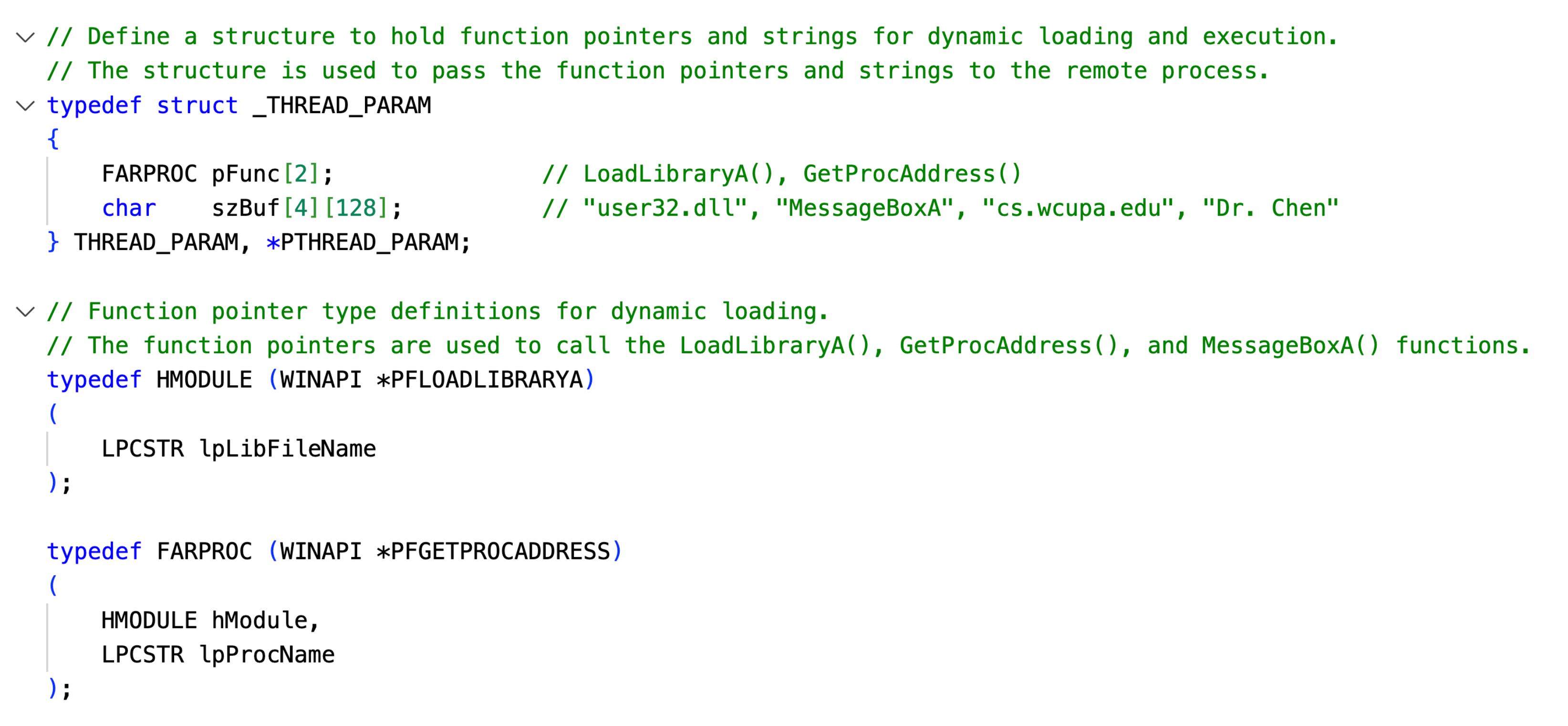
cs.wcupa.edu

CodeInjection.cpp – main()



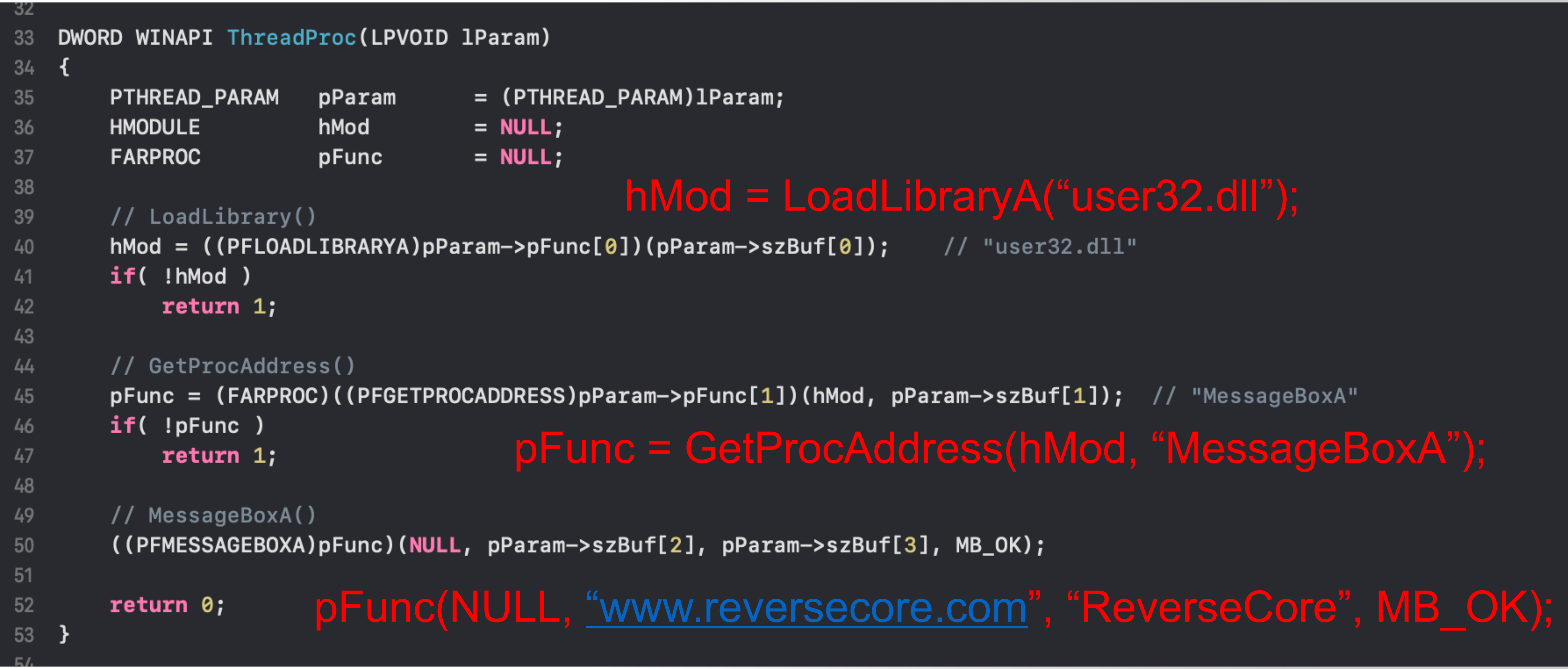
```
int main(int argc, char *argv[])
{
    DWORD dwPID     = 0;

    if( argc != 2 )
    {
        printf("\n USAGE  : %s <pid>\n", argv[0]);
        return 1;
    }

    // change privilege
    if( !SetPrivilege(SE_DEBUG_NAME, TRUE) )
        return 1;

    // code injection
    dwPID = (DWORD)atol(argv[1]);
    InjectCode(dwPID);

    return 0;
}
```

CodeInjection.cpp – ThreadProc()



```
// Define a structure to hold function pointers and strings for dynamic loading and execution.
// The structure is used to pass the function pointers and strings to the remote process.
typedef struct _THREAD_PARAM
{
    FARPROC pFunc[2];               // LoadLibraryA(), GetProcAddress()
    char    szBuf[4][128];          // "user32.dll", "MessageBoxA", "cs.wcupa.edu", "Dr. Chen"
} THREAD_PARAM, *PTHREAD_PARAM;

// Function pointer type definitions for dynamic loading.
// The function pointers are used to call the LoadLibraryA(), GetProcAddress(), and MessageBoxA() functions.
typedef HMODULE (WINAPI *PFLOADLIBRARYA)
(
    LPCSTR lpLibFileName
);

typedef FARPROC (WINAPI *PFGETPROCADDRESS)
(
    HMODULE hModule,
    LPCSTR lpProcName
);
```

CodeInjection.cpp – ThreadProc()



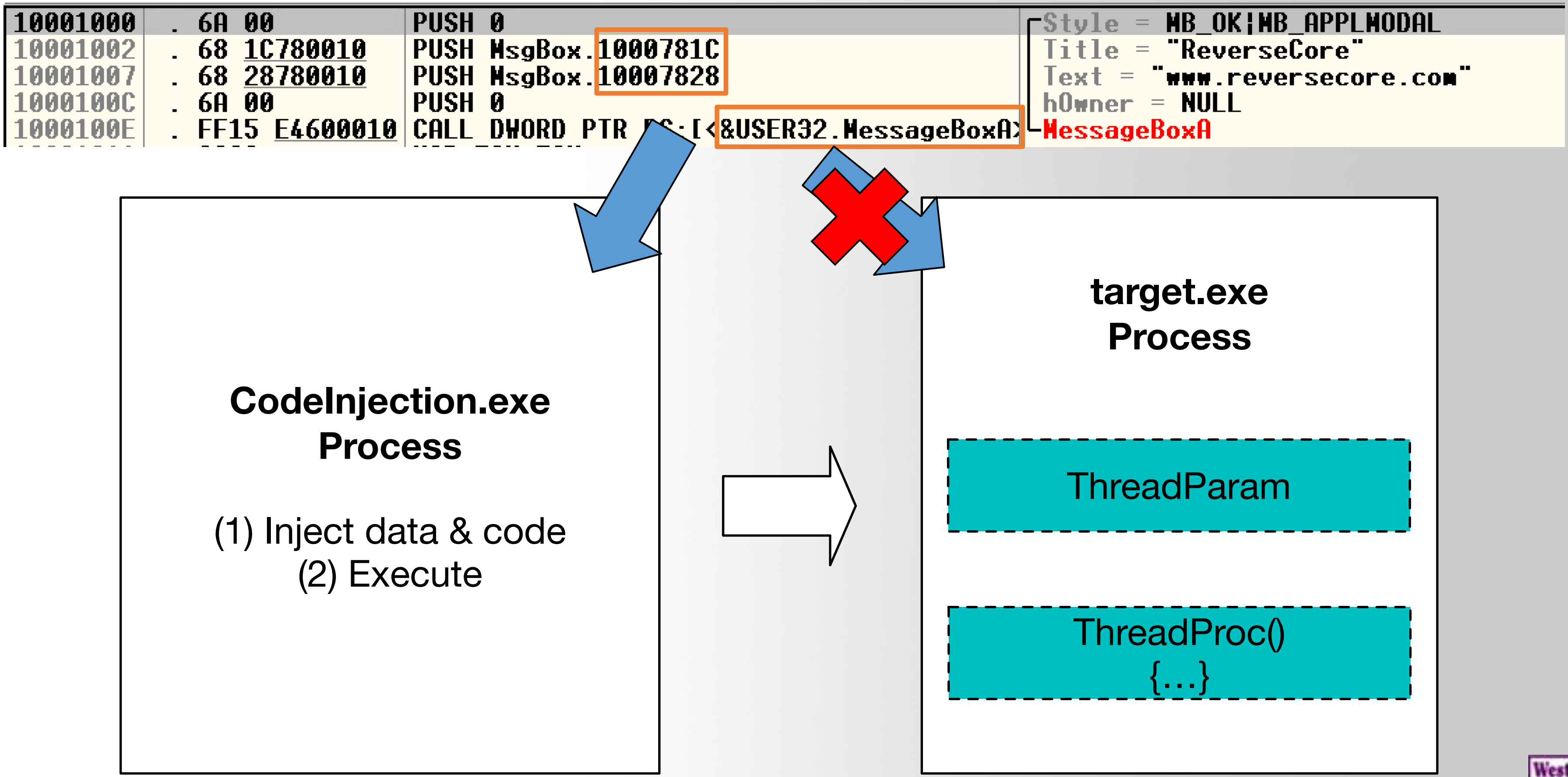
```
DWORD WINAPI ThreadProc(LPVOID lParam)
{
    PTHREAD_PARAM   pParam      = (PTHREAD_PARAM)lParam;
    HMODULE         hMod        = NULL;
    FARPROC         pFunc       = NULL;

    // LoadLibrary()
    hMod = ((PFLOADLIBRARYA)pParam->pFunc[0])(pParam->szBuf[0]);    // "user32.dll"
    if( !hMod )
        return 1;

    // GetProcAddress()
    pFunc = (FARPROC)((PFGETPROCADDRESS)pParam->pFunc[1])(hMod, pParam->szBuf[1]);  // "MessageBoxA"
    if( !pFunc )
        return 1;

    // MessageBoxA()
    ((PFMESSAGEBOXA)pFunc)(NULL, pParam->szBuf[2], pParam->szBuf[3], MB_OK);

    return 0;
}
```

hMod = LoadLibraryA(“user32.dll”);

pFunc = GetProcAddress(hMod, “MessageBoxA”);

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

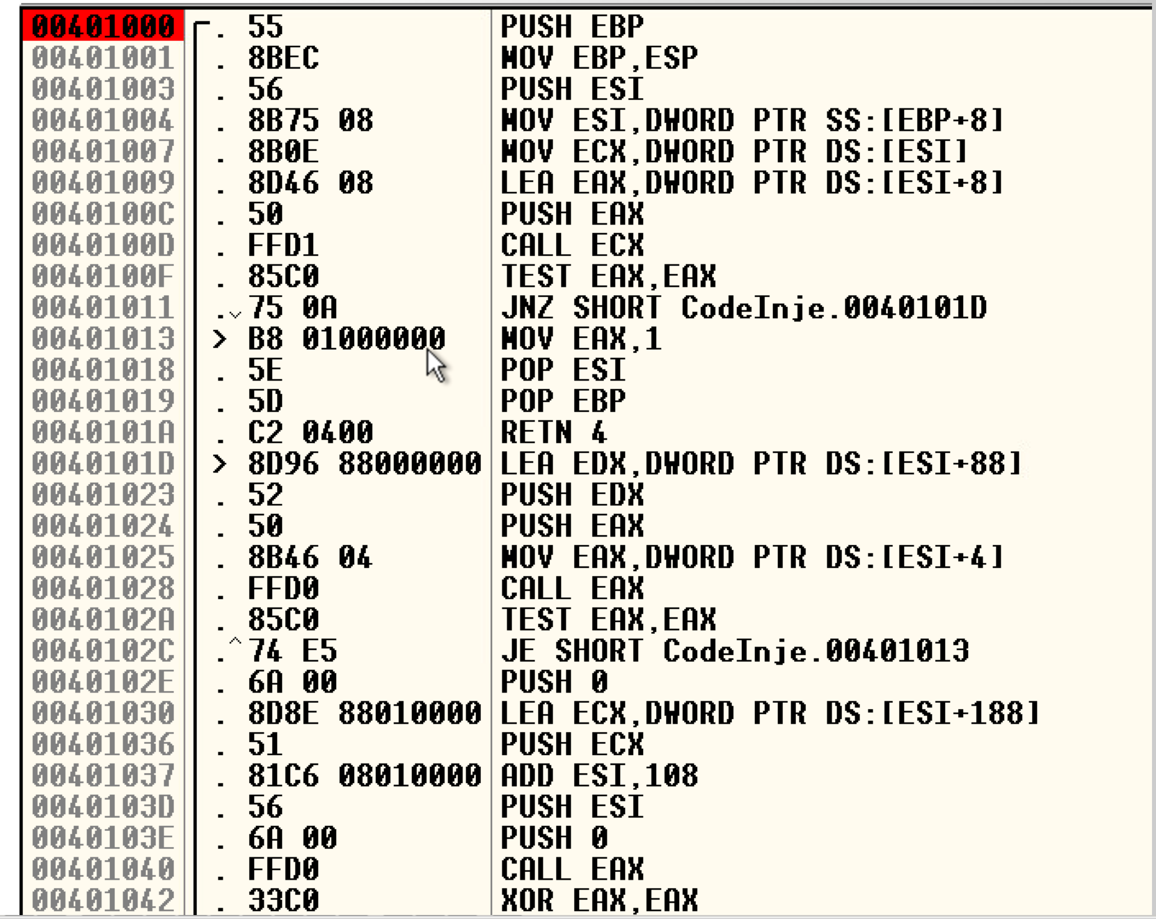
Cannot use the following address for Code Injection



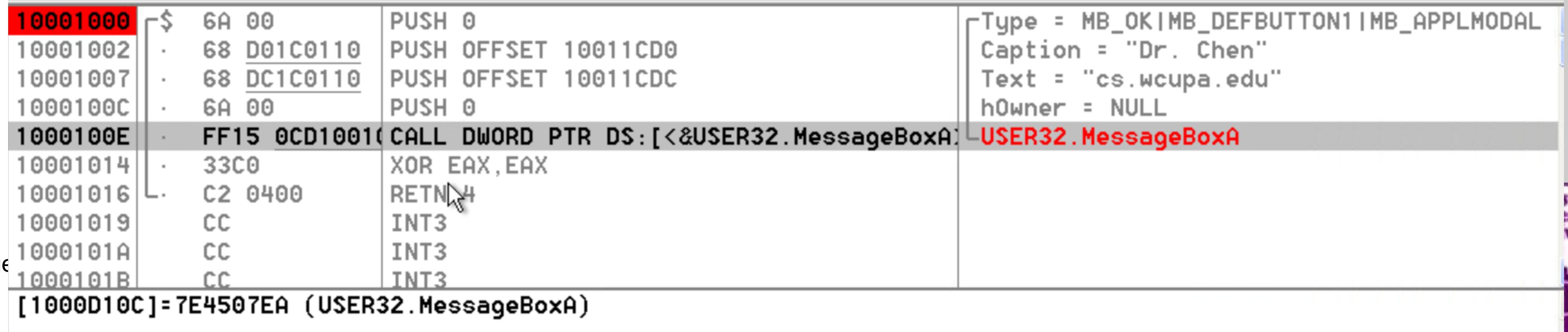
```
10001000  . 6A 00         PUSH 0                                   ┌Style = MB_OK|MB_APPLMODAL
10001002  . 68 1C780010   PUSH MsgBox.1000781C                     │Title = "ReverseCore"
10001007  . 68 28780010   PUSH MsgBox.10007828                     │Text = "www.reversecore.com"
1000100C  . 6A 00         PUSH 0                                   │hOwner = NULL
1000100E  . FF15 E4600010 CALL DWORD PTR DS:[<&USER32.MessageBoxA> └MessageBoxA
```

You cannot use the address provided for code injection.

Because MessageBoxA and Caption and “Text” are not loaded, these addresses cannot be used for code injection.

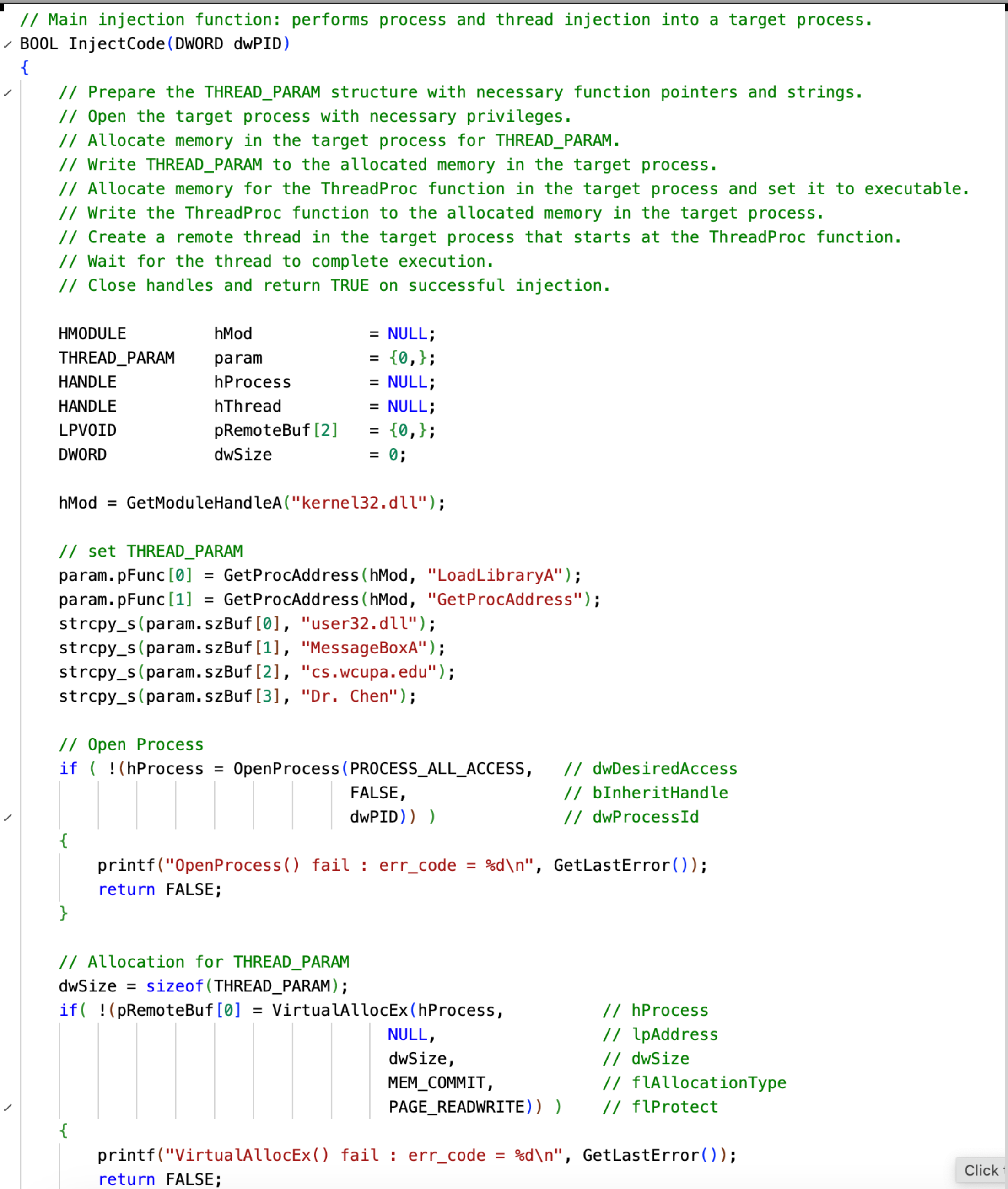


```
00401000 ┌. 55             PUSH EBP
00401001 │. 8BEC           MOV EBP,ESP
00401003 │. 56             PUSH ESI
00401004 │. 8B75 08        MOV ESI,DWORD PTR SS:[EBP+8]
00401007 │. 8B0E           MOV ECX,DWORD PTR DS:[ESI]
00401009 │. 8D46 08        LEA EAX,DWORD PTR DS:[ESI+8]
0040100C │. 50             PUSH EAX
0040100D │. FFD1           CALL ECX
0040100F │. 85C0           TEST EAX,EAX
00401011 │.v 75 0A         JNZ SHORT CodeInje.0040101D
00401013 │> B8 01000000    MOV EAX,1
00401018 │. 5E             POP ESI
00401019 │. 5D             POP EBP
0040101A │. C2 0400        RETN 4
0040101D │> 8D96 88000000  LEA EDX,DWORD PTR DS:[ESI+88]
00401023 │. 52             PUSH EDX
00401024 │. 50             PUSH EAX
00401025 │. 8B46 04        MOV EAX,DWORD PTR DS:[ESI+4]
00401028 │. FFD0           CALL EAX
0040102A │. 85C0           TEST EAX,EAX
0040102C │.^74 E5          JE SHORT CodeInje.00401013
0040102E │. 6A 00          PUSH 0
00401030 │. 8D8E 88010000  LEA ECX,DWORD PTR DS:[ESI+188]
00401036 │. 51             PUSH ECX
00401037 │. 81C6 08010000  ADD ESI,108
0040103D │. 56             PUSH ESI
0040103E │. 6A 00          PUSH 0
00401040 │. FFD0           CALL EAX
00401042 │. 33C0           XOR EAX,EAX
```



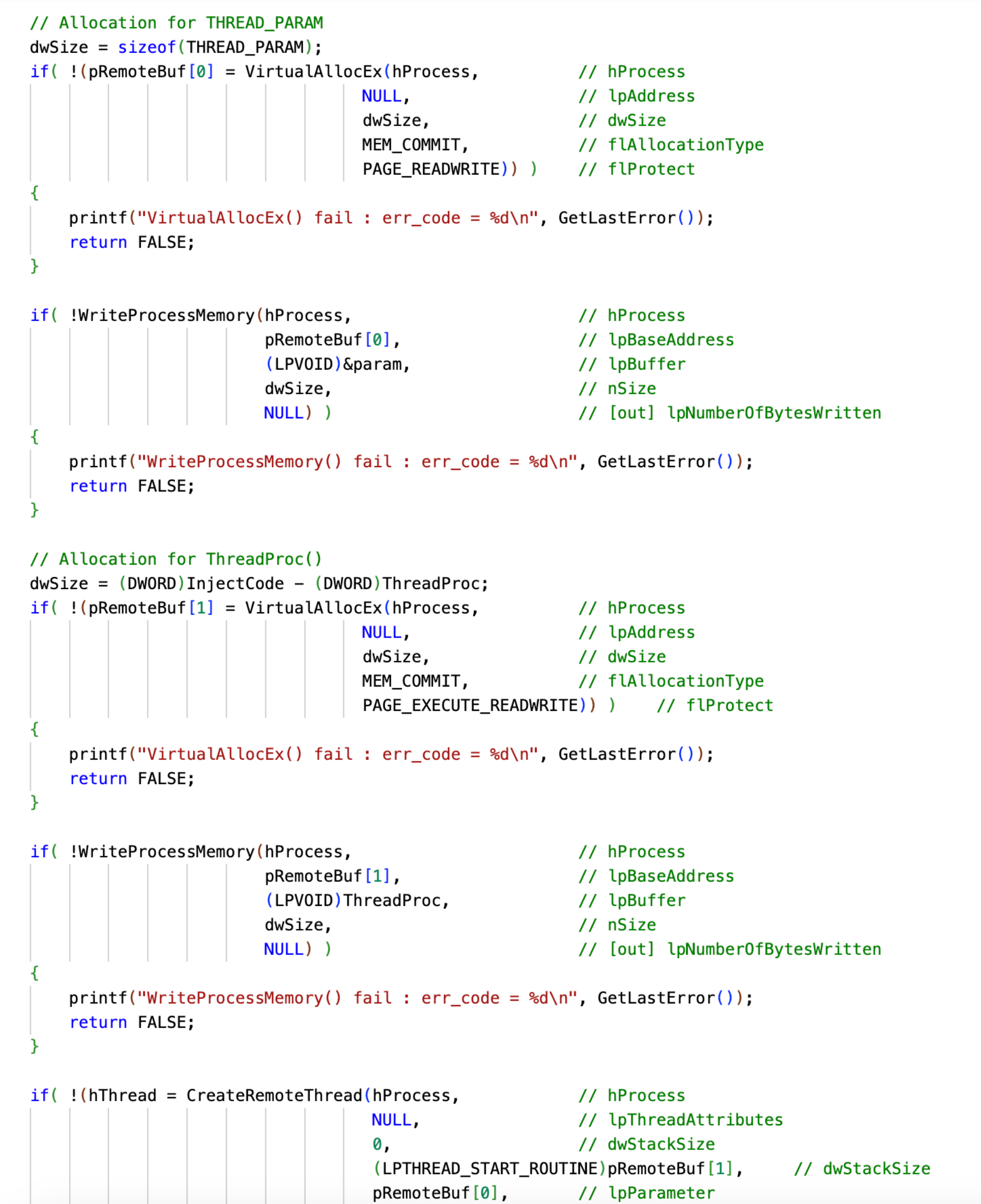
```
10001000 ┌$ 6A 00            PUSH 0                                      ┌Type = MB_OK|MB_DEFBUTTON1|MB_APPLMODAL
10001002 │. 68 D01C0110      PUSH OFFSET 10011CD0                        │Caption = "Dr. Chen"
10001007 │. 68 DC1C0110      PUSH OFFSET 10011CDC                        │Text = "cs.wcupa.edu"
1000100C │. 6A 00            PUSH 0                                      │hOwner = NULL
1000100E │. FF15 0CD1001(    CALL DWORD PTR DS:[<&USER32.MessageBoxA]    └USER32.MessageBoxA
10001014 │. 33C0             XOR EAX,EAX
10001016 └. C2 0400          RETN 4
10001019    CC               INT3
1000101A    CC               INT3
1000101B    CC               INT3
[1000D10C]=7E4507EA (USER32.MessageBoxA)
```

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
```

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.
```

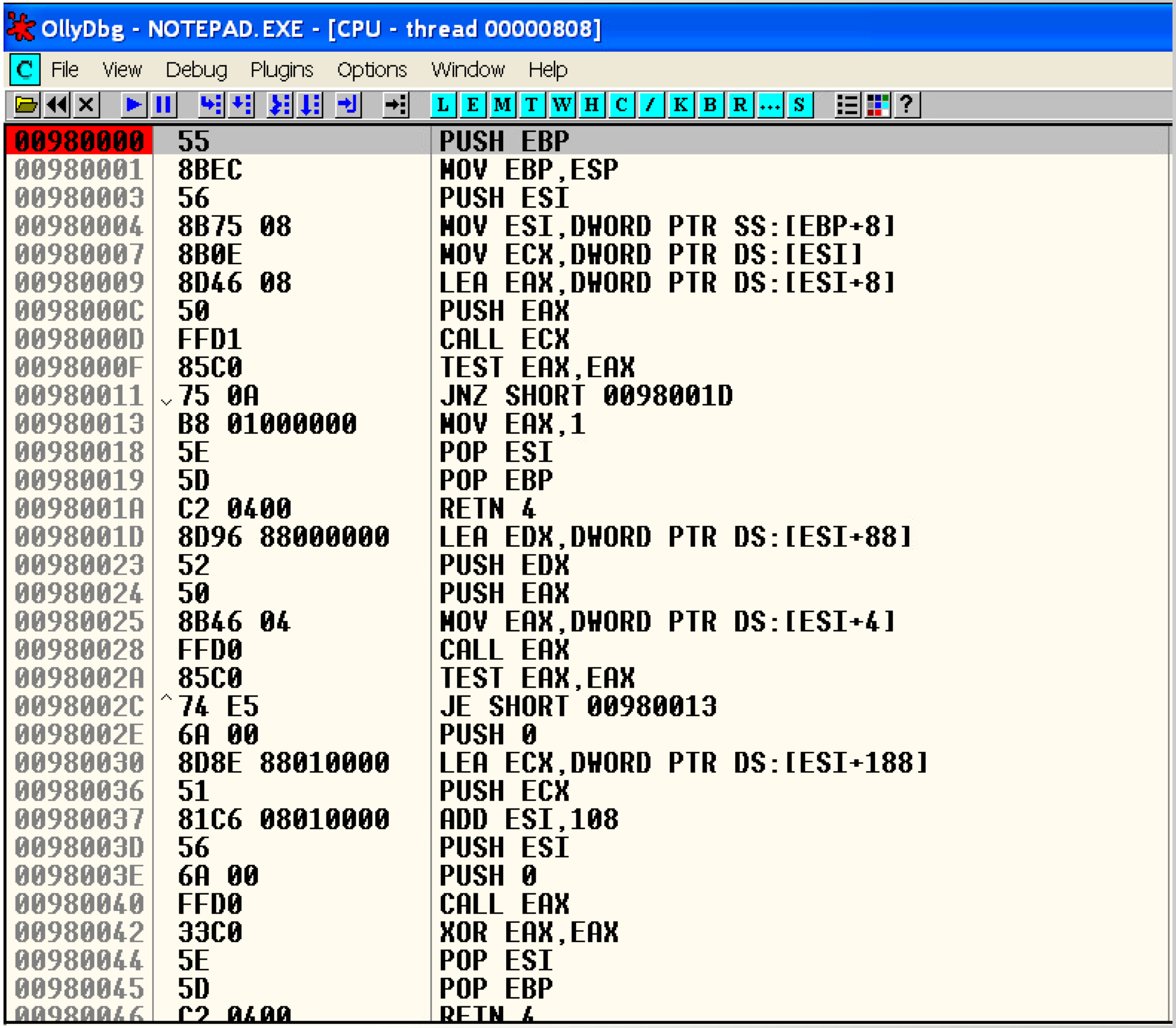
- OpenProcess()

- **//data: THREAD_PARAM**
- VirtualAllocEx()
- WriteProcessMemory()

- **//Code: ThreadProc()**
- VirtualAllocEx()
- WriteProcessMemory()

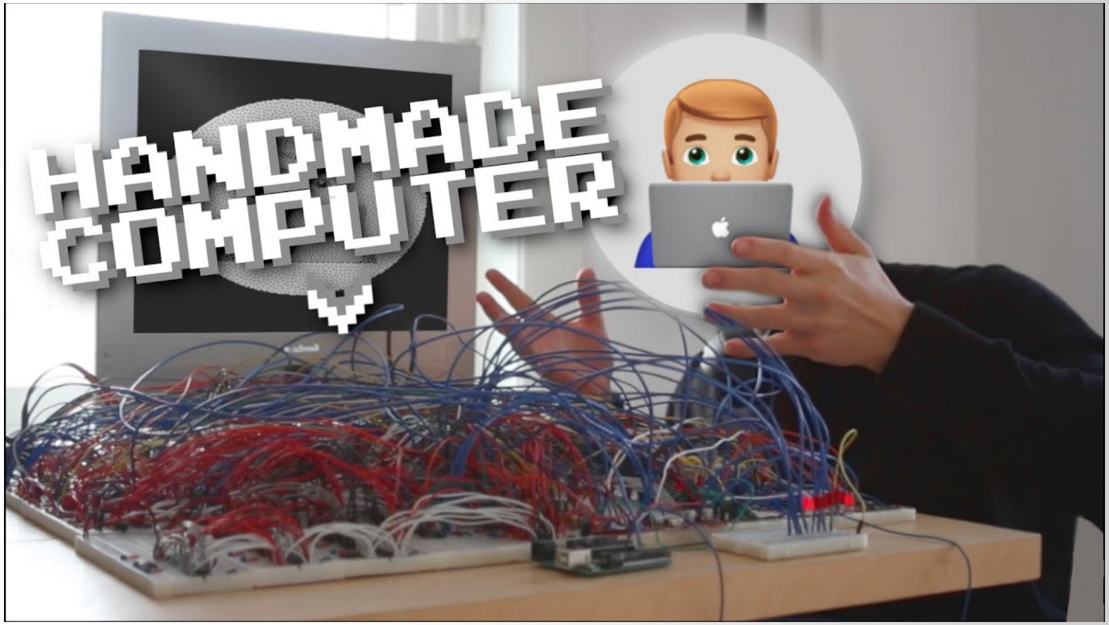
- CreateRemoteThread()

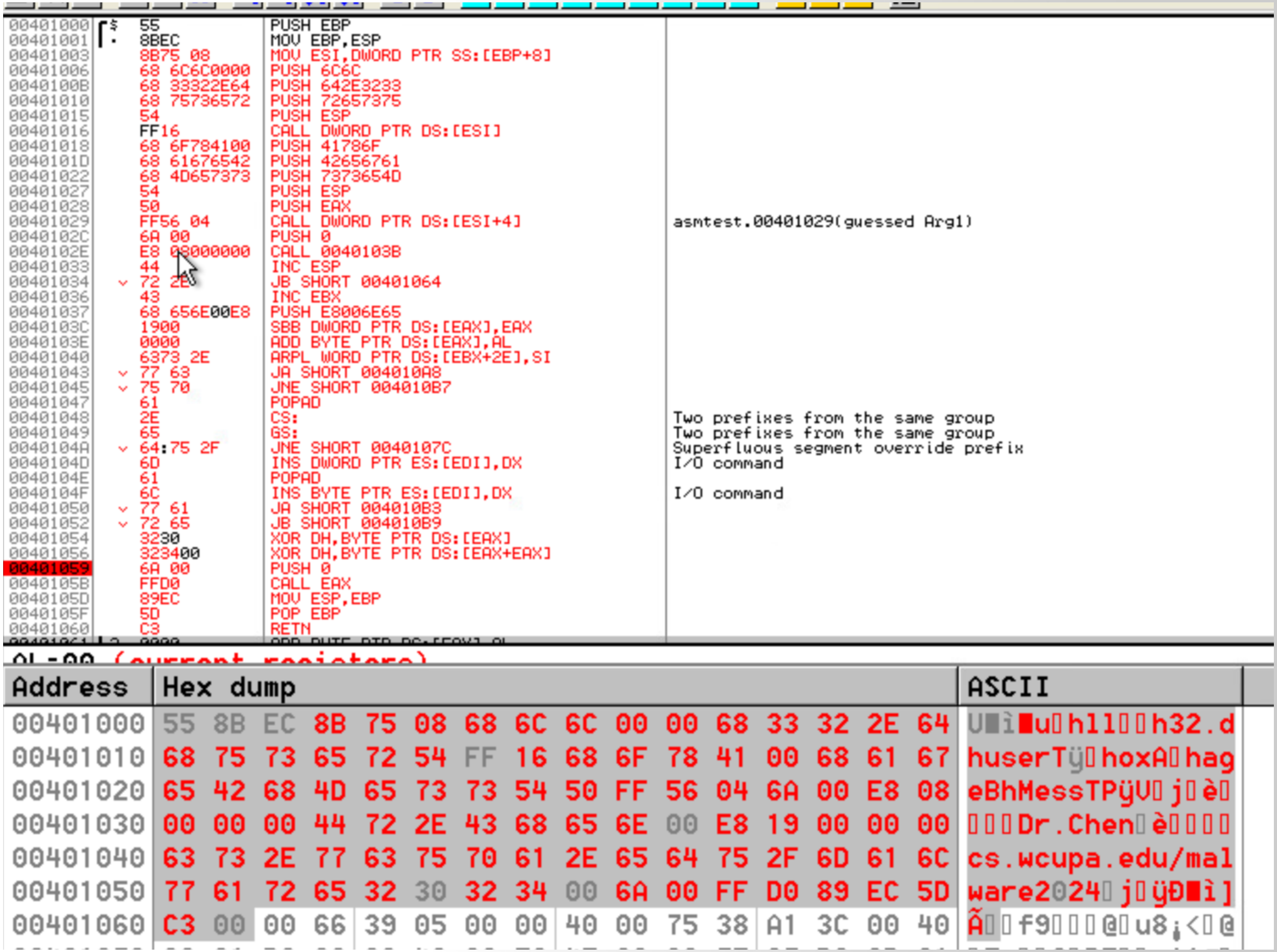
How to Debug Code Injection (OllyDBG)



```
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.





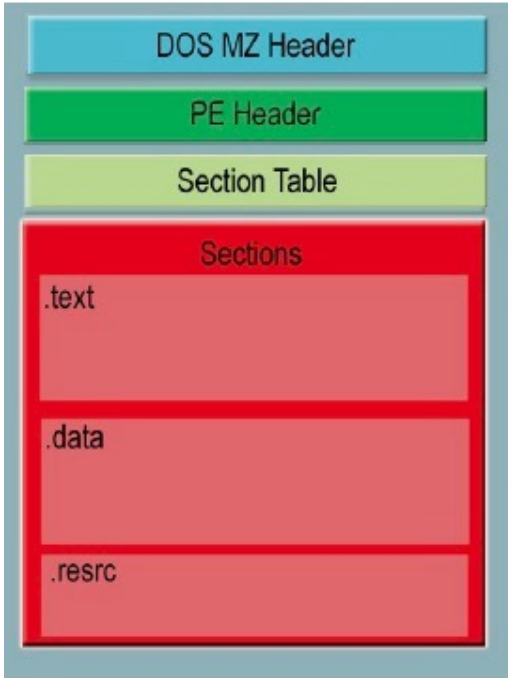
```
00401000 ┌$  55            PUSH EBP
00401001 │.  8BEC          MOV EBP,ESP
00401003     8B75 08       MOV ESI,DWORD PTR SS:[EBP+8]
00401006     68 6C6C0000   PUSH 6C6C
0040100B     68 33322E64   PUSH 642E3233
00401010     68 75736572   PUSH 72657375
00401015     54            PUSH ESP
00401016     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]           asmtest.00401029(guessed Arg1)
0040102C     6A 00         PUSH 0
0040102E     E8 08000000   CALL 0040103B
00401033     44            INC ESP
00401034   v 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   v 77 63         JA SHORT 004010A8
00401045   v 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   v 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   v 77 61         JA SHORT 004010B3
00401052   v 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
00401061  |. 0000          ADD BYTE PTR DS:[EAX],AL
AL=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■ì■u▯hll▯▯h32.d
00401010  68 75 73 65 72 54 FF 16 68 6F 78 41 00 68 61 67  huserTÿ▯hoxA▯hag
00401020  65 42 68 4D 65 73 73 54 50 FF 56 04 6A 00 E8 08  eBhMessTPÿV▯j▯è▯
00401030  00 00 00 44 72 2E 43 68 65 6E 00 E8 19 00 00 00  ▯▯▯Dr.Chen▯è▯▯▯▯
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▯ÿÐ■ì]
00401060  C3 00 00 66 39 05 00 00 40 00 75 38 A1 3C 00 40  Ã▯▯f9▯▯▯@▯u8¡<▯@
```

Q & A

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”.



😑

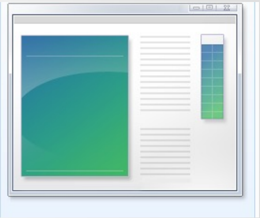
Now here is the kicker. Even though this specification is spelled out by Microsoft, compilers/linkers chose to ignore some parts of it.

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...**

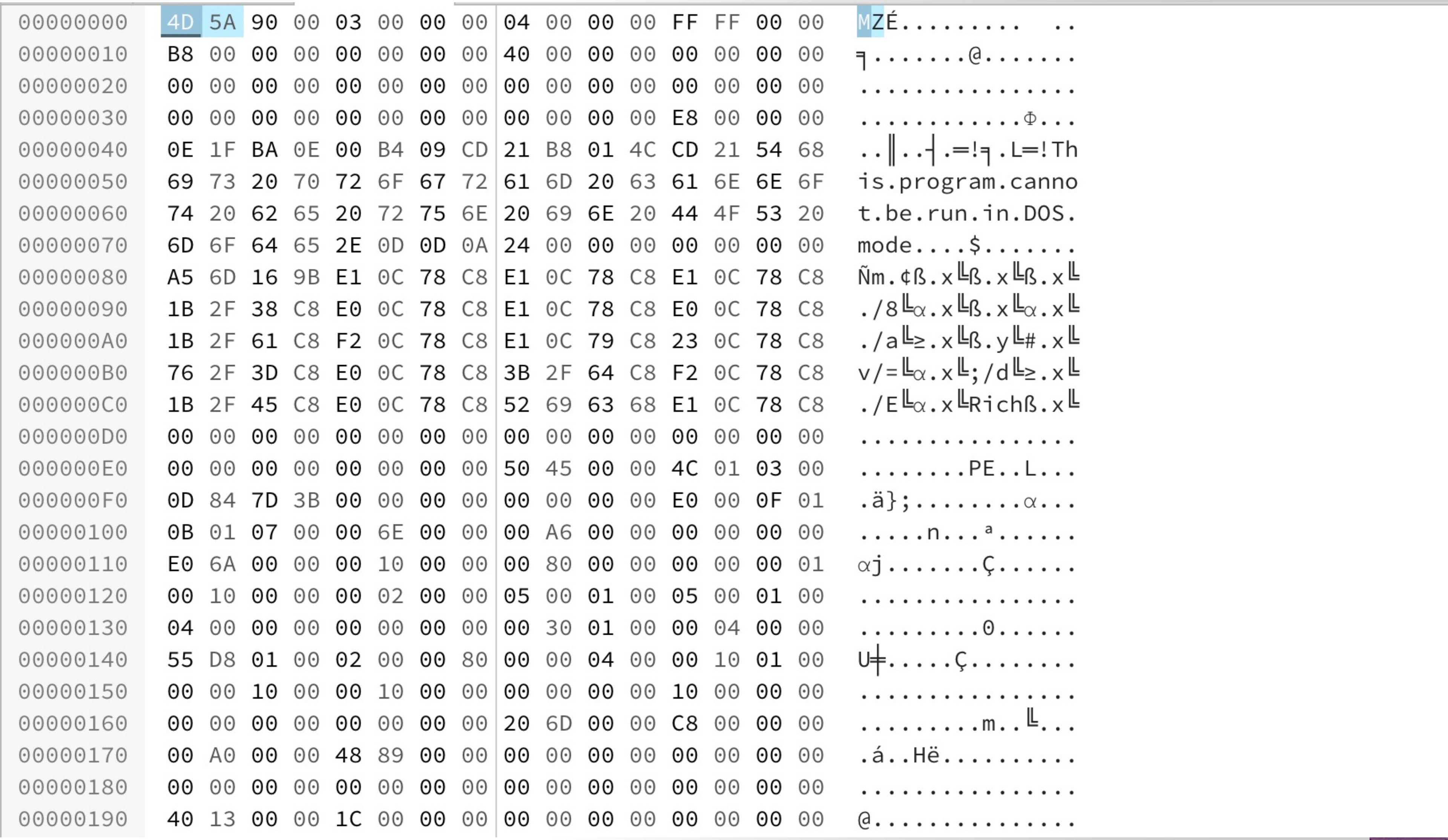
Portable Executable (PE) file

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



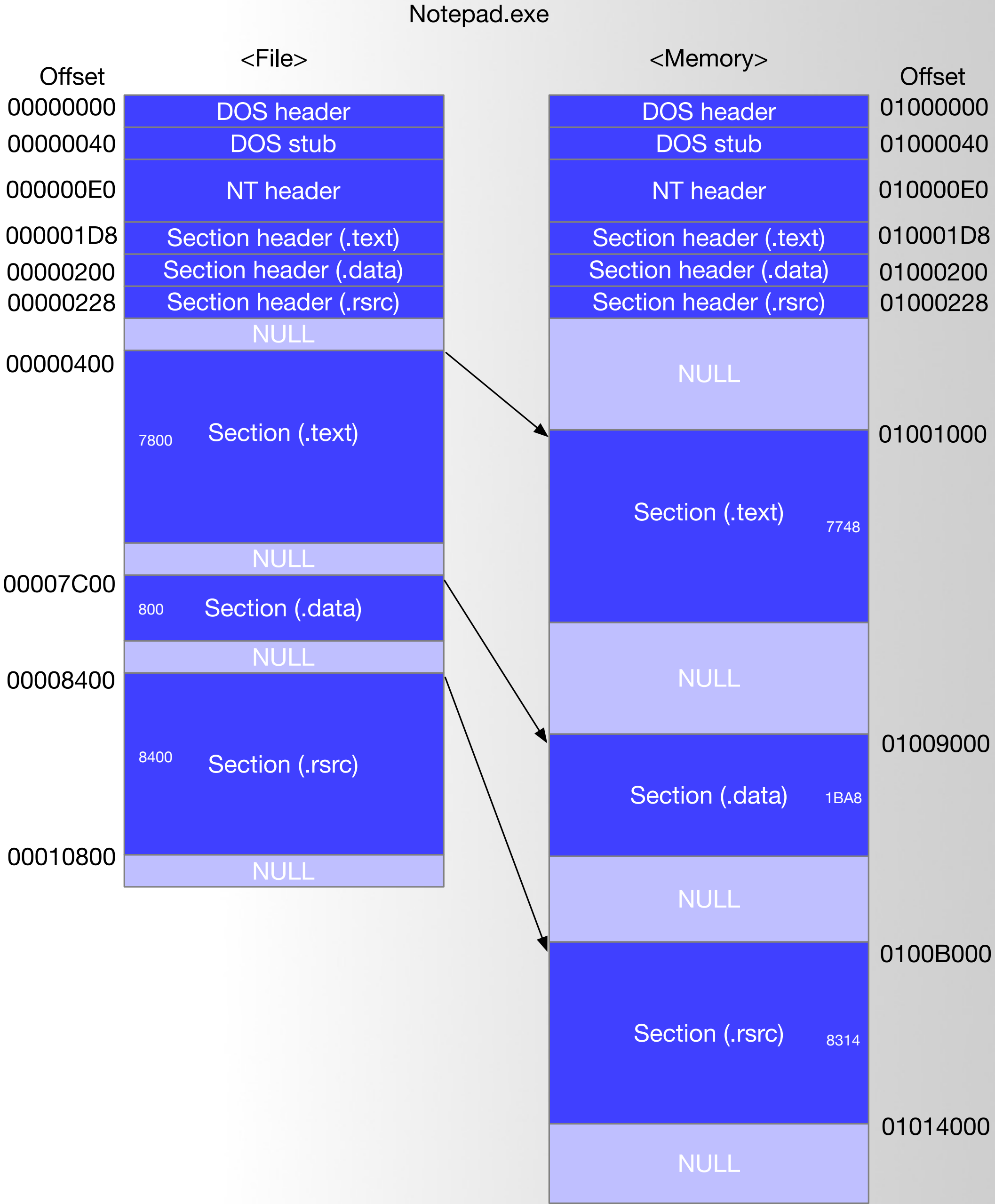
- All PE formatted files can be executed, except obj file.
  - .exe, .scr can be directly executed inside Shell (explorer.exe)
  - others can be executed by other program/service
- **PE refers to 32 bit** executable file, or **PE32**. **64 bit** executable file is named as **PE+ or PE32+.** (Note that it is not PE64).

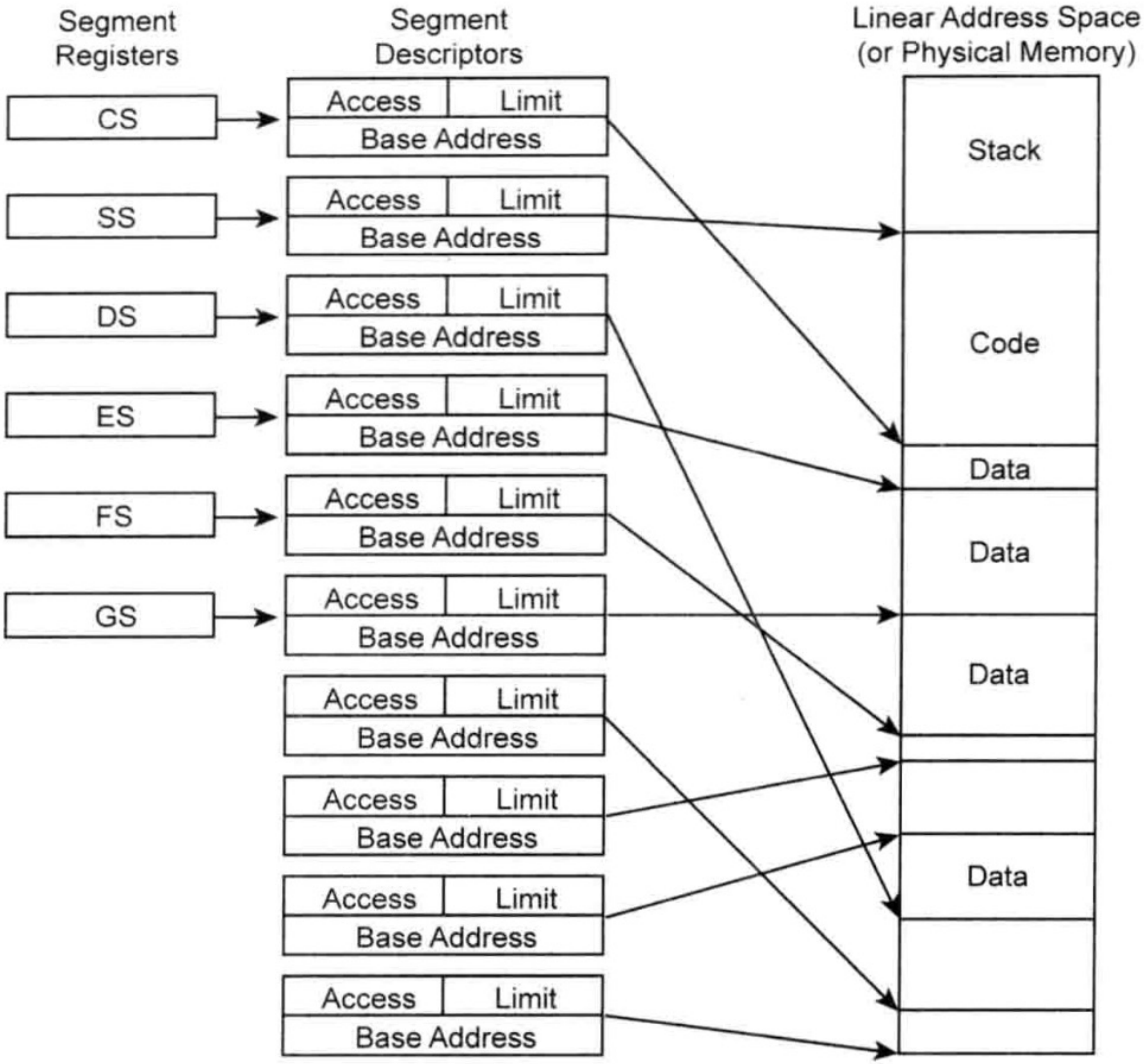
PE Example – Notepad.exe



```
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   ╕.......@.......
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 00   ............Φ...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ..║..┤.═!╕.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.x╚ß.x╚ß.x╚
00000090  1B 2F 38 C8 E0 0C 78 C8 E1 0C 78 C8 E0 0C 78 C8   ./8╚α.x╚ß.x╚α.x╚
000000A0  1B 2F 61 C8 F2 0C 78 C8 E1 0C 79 C8 23 0C 78 C8   ./a╚≥.x╚ß.y╚#.x╚
000000B0  76 2F 3D C8 E0 0C 78 C8 3B 2F 64 C8 F2 0C 78 C8   v/=╚α.x╚;/d╚≥.x╚
000000C0  1B 2F 45 C8 E0 0C 78 C8 52 69 63 68 E1 0C 78 C8   ./E╚α.x╚Richß.x╚
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 00   ........PE..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 00   .....n...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 00   .........0......
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 00   .........m..╚...
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




Segment Registers
Segment Descriptors
Linear Address Space (or Physical Memory)
CS
SS
DS
ES
FS
GS
Access
Limit
Base Address
Stack
Code
Data
Data
Data
Data

VA & RVA

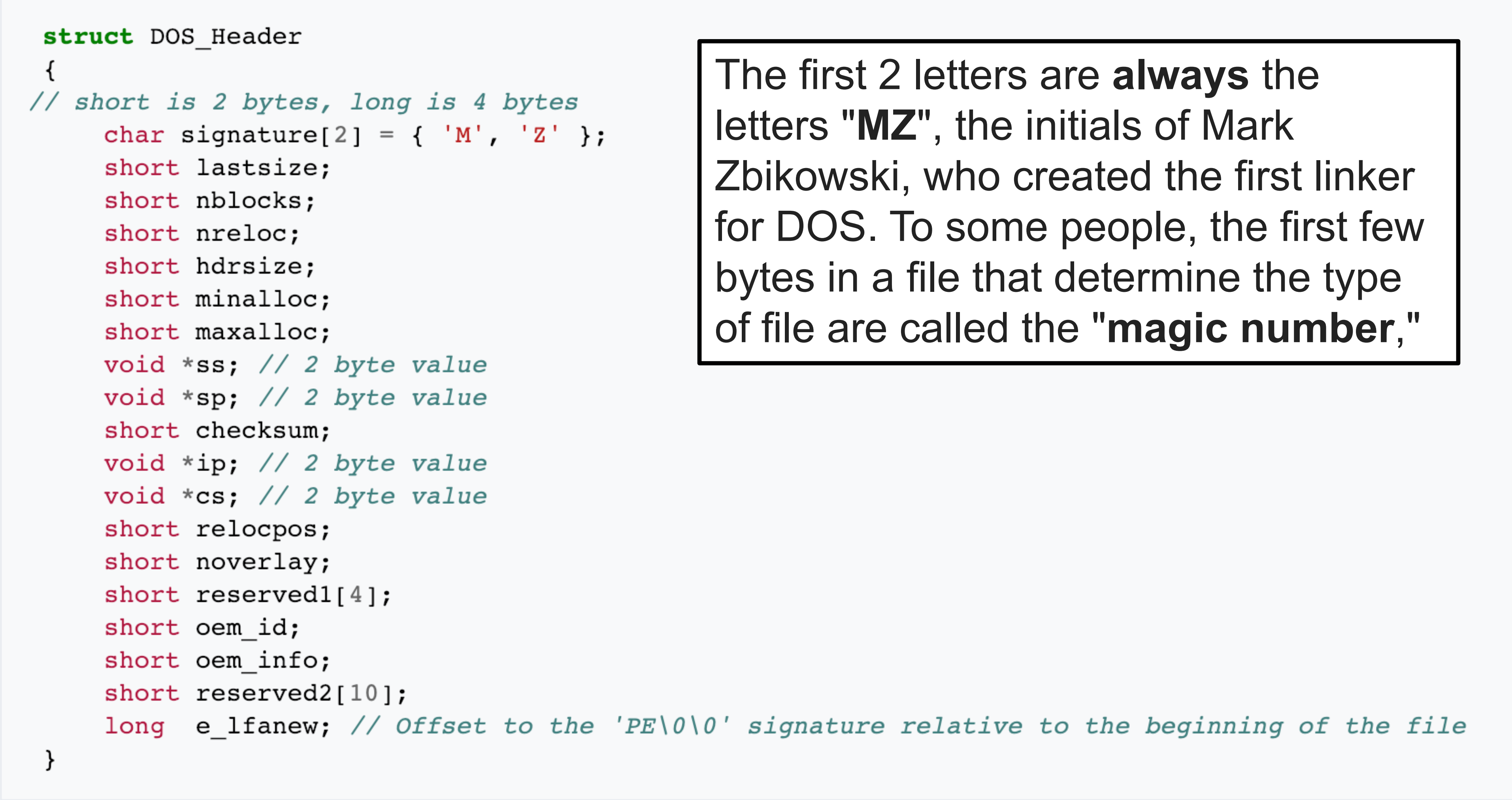
- VA (Virtual Address): The address is called a “VA” because **Windows creates a distinct** VA space for each process, **independent** of physical memory. For almost all purposes, a VA should be considered just an address. A VA is not as predictable as an RVA because the loader might not load the image 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).

RVA + ImageBase = VA

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

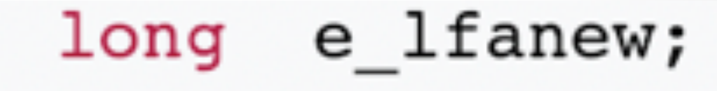
DOS Header



```
 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
 }
```

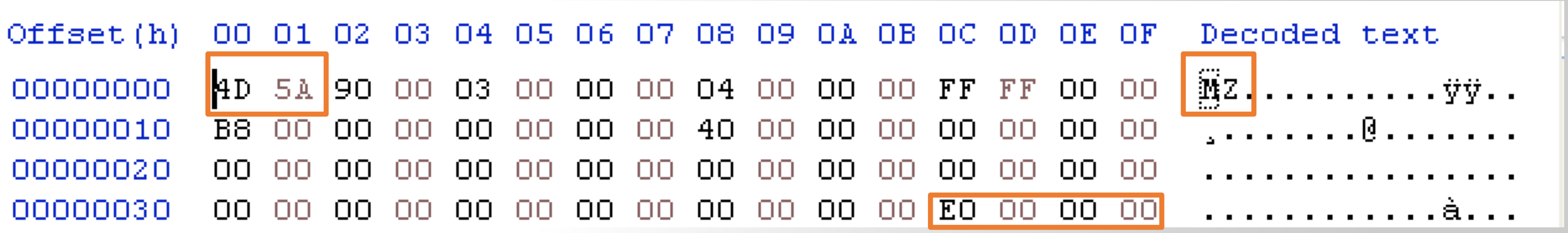
The first 2 letters are **always** the letters "**MZ**", the initials of Mark Zbikowski, who created the first linker for DOS. To some people, the first few bytes in a file that determine the type of file are called the "**magic number**,"

DOS Header



```
long   e_lfanew;
```

long → 32 bit → ? Byte

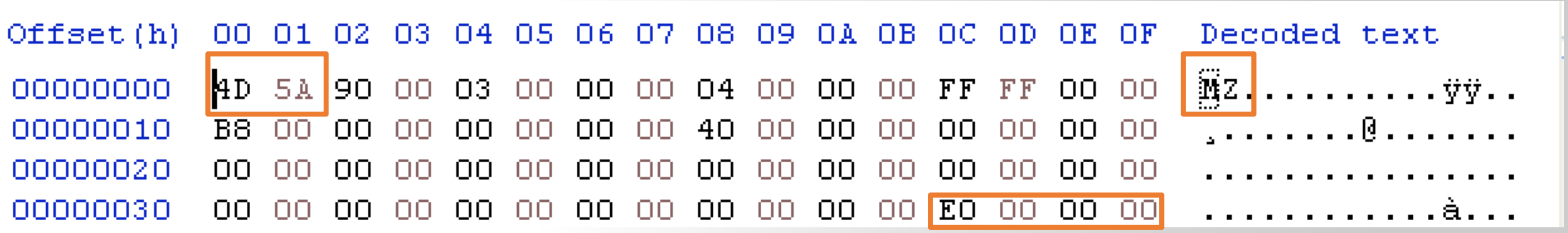


```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010   B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................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

E0 00 00 00

value for e_lfanew → ?

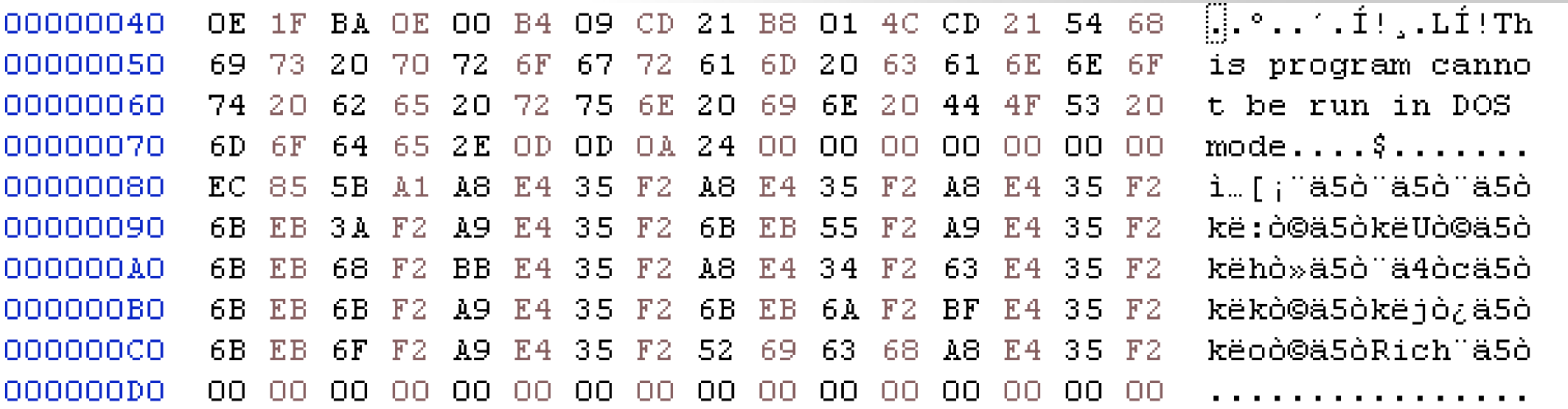
DOS Header



```
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..........ÿÿ..
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 00  ............à...
```

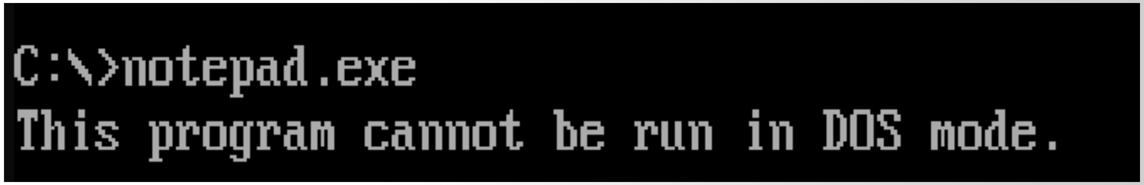
e_lfanew → 000000E0

DOS stub



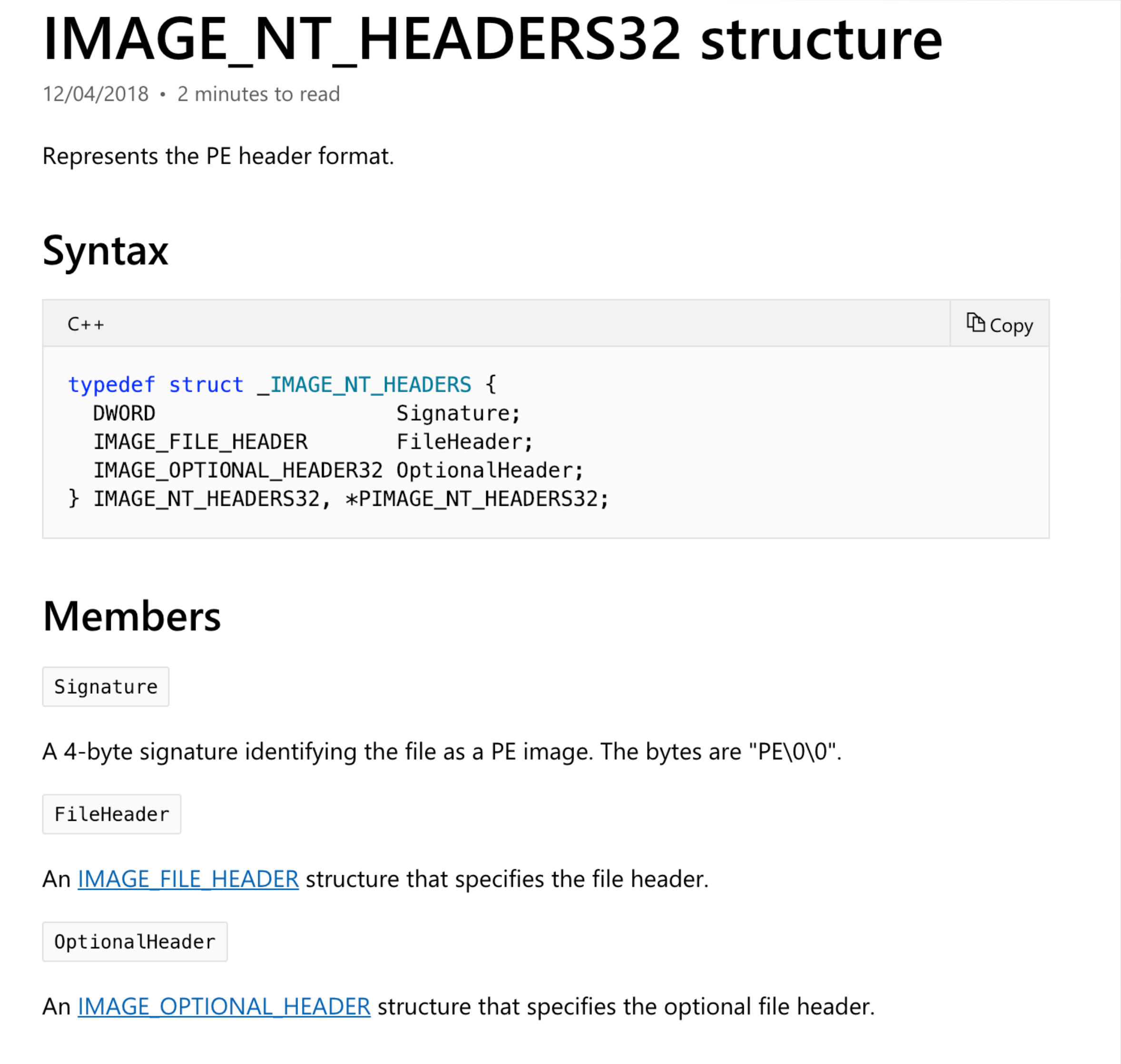
```
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  EC 85 5B A1 A8 E4 35 F2 A8 E4 35 F2 A8 E4 35 F2  ì…[¡¨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©ä5òkëjò¿ä5ò
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.
```

NT Header



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`

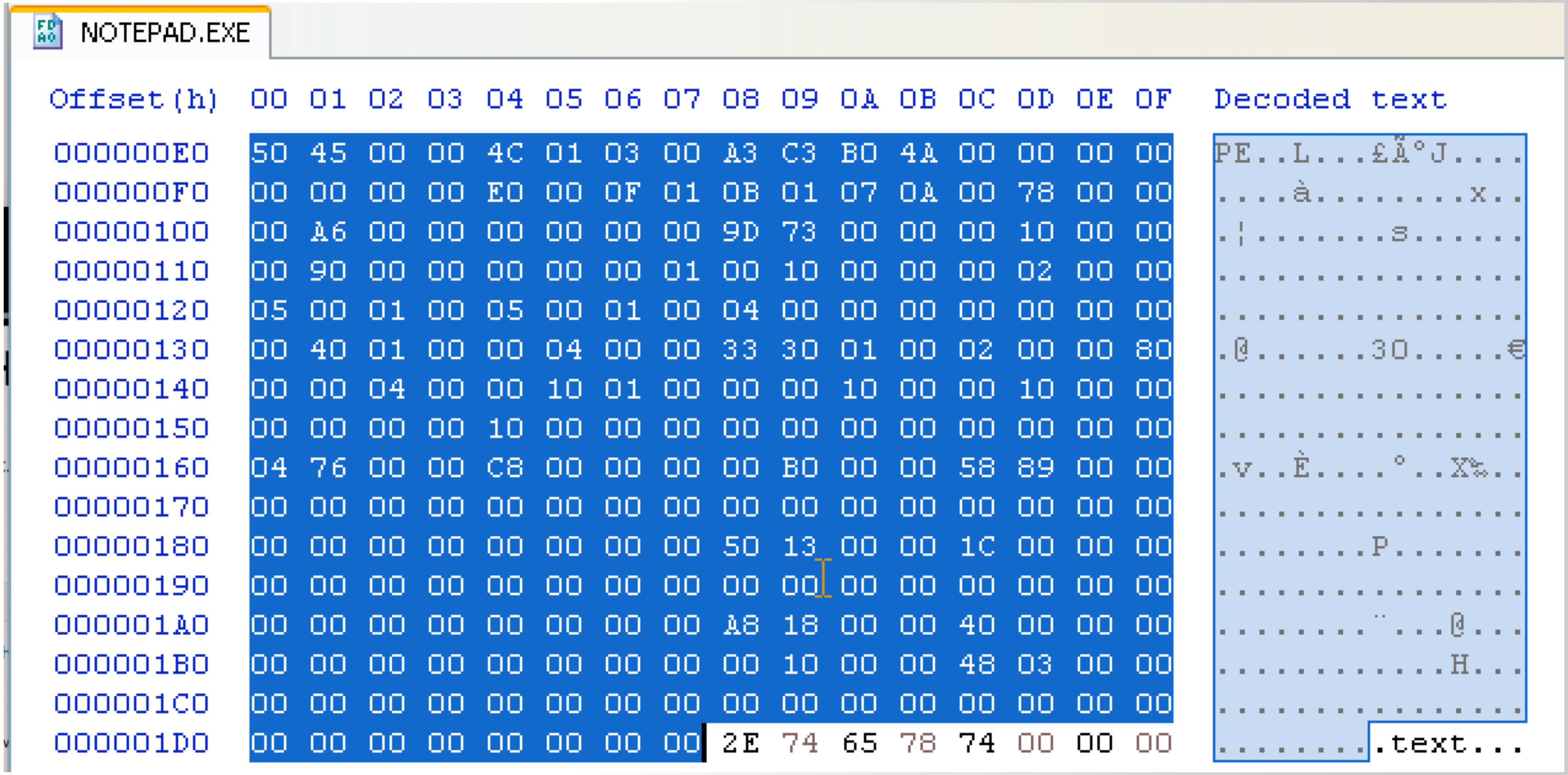
An IMAGE_FILE_HEADER structure that specifies the file header.

`OptionalHeader`

An IMAGE_OPTIONAL_HEADER structure that specifies the optional file header.

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

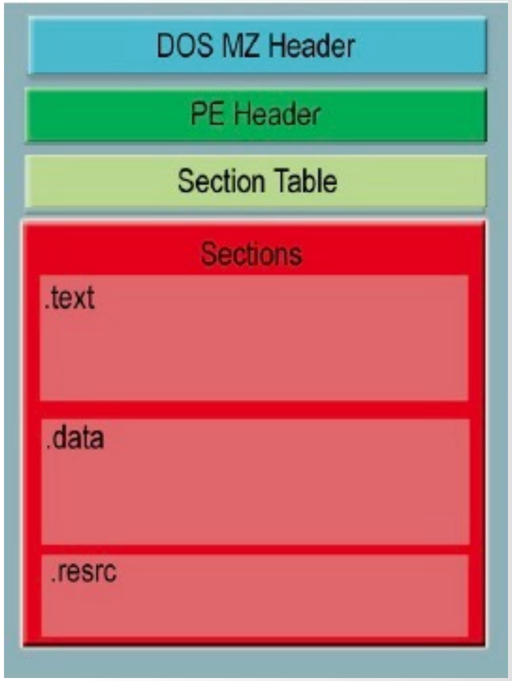
NT Header



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...£Ã°J....
000000F0  00 00 00 00 E0 00 0F 01 0B 01 07 0A 00 78 00 00  ....à........x..
00000100  00 A6 00 00 00 00 00 00 9D 73 00 00 00 10 00 00  .¦.......s......
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 00  ........P.......
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 00  ............H...
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 00  ..........text...
```

Section Header



DOS MZ Header

PE Header

Section Table

Sections

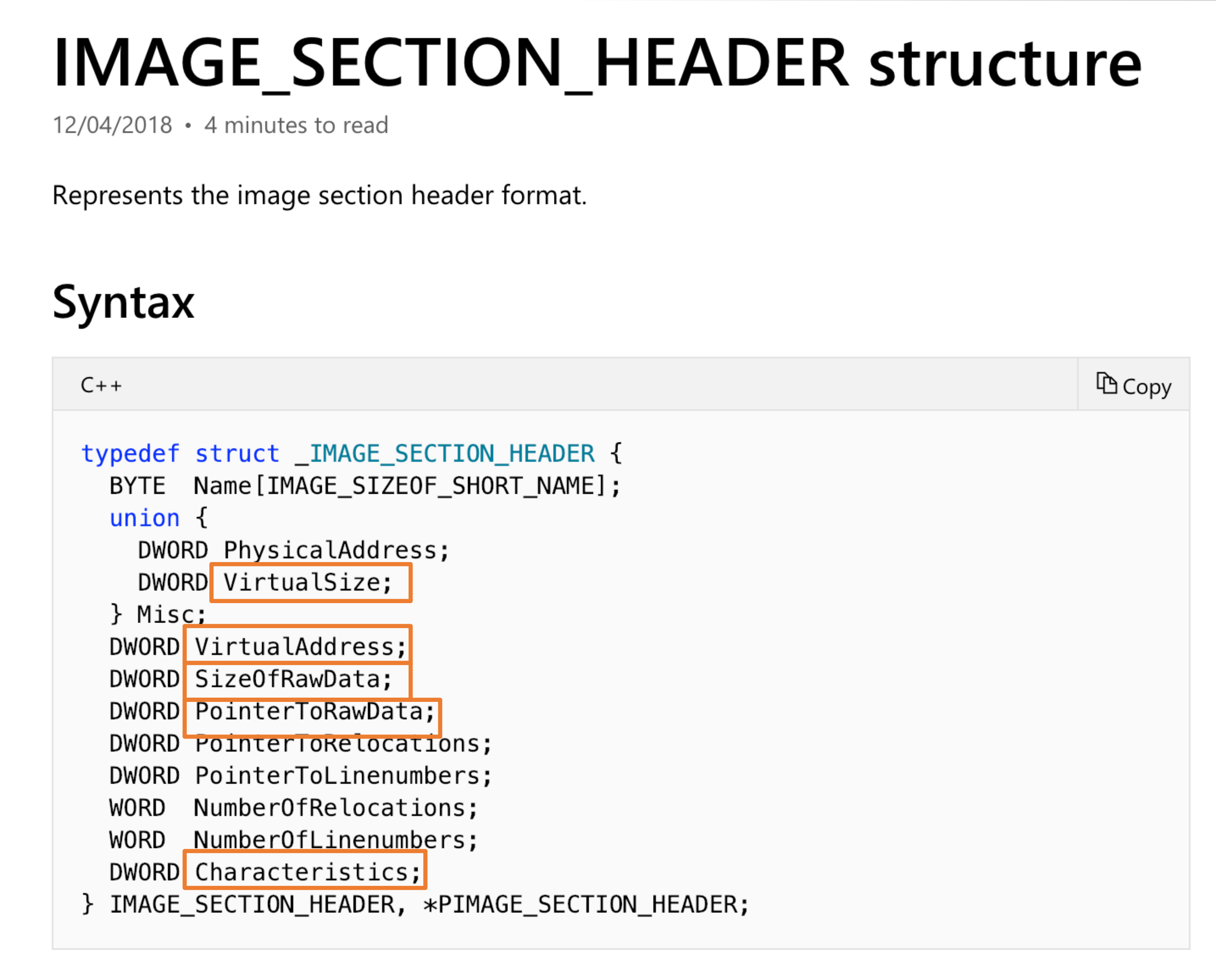
.text

.data

.resrc

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

Section Header



IMAGE_SECTION_HEADER structure

12/04/2018 • 4 minutes to read

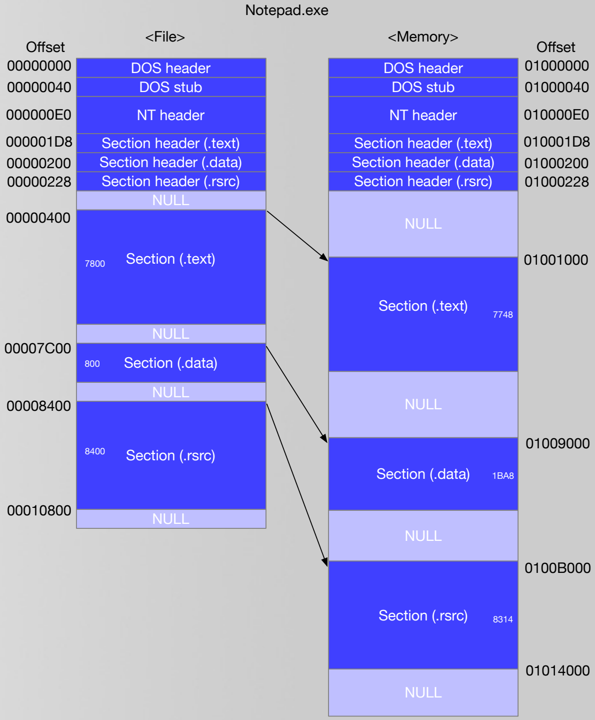
Represents the image section header format.

Syntax

```cpp
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;
```

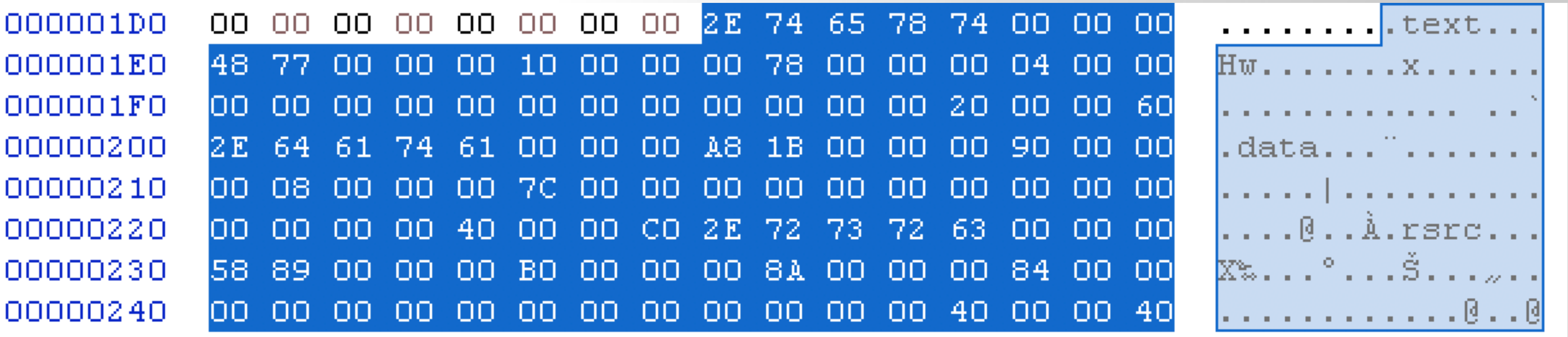
Section Header

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.



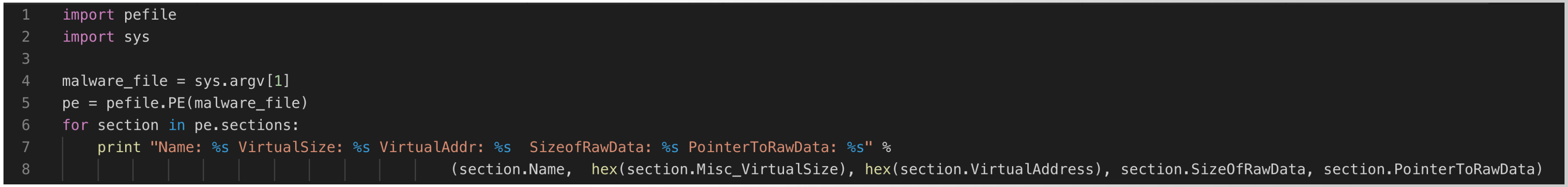
https://docs.microsoft.com/en-us/windows/desktop/api/winnt/ns-winnt-_image_section_header

Section Header

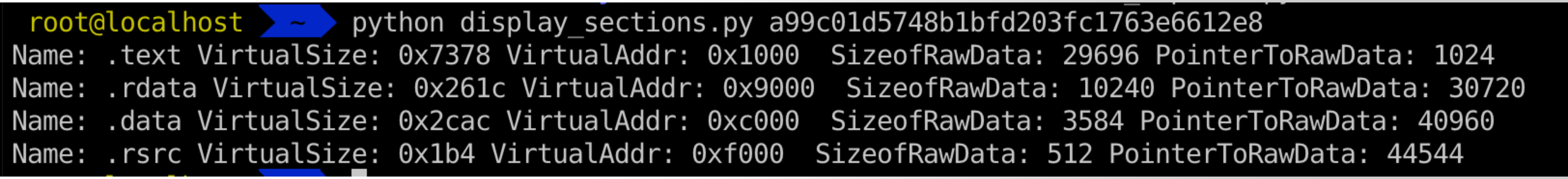


```
000001D0  00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00  .........text...
000001E0  48 77 00 00 00 10 00 00 00 78 00 00 00 04 00 00  Hw.......x......
000001F0  00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60  ............ ..`
00000200  2E 64 61 74 61 00 00 00 A8 1B 00 00 00 90 00 00  .data...¨.......
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

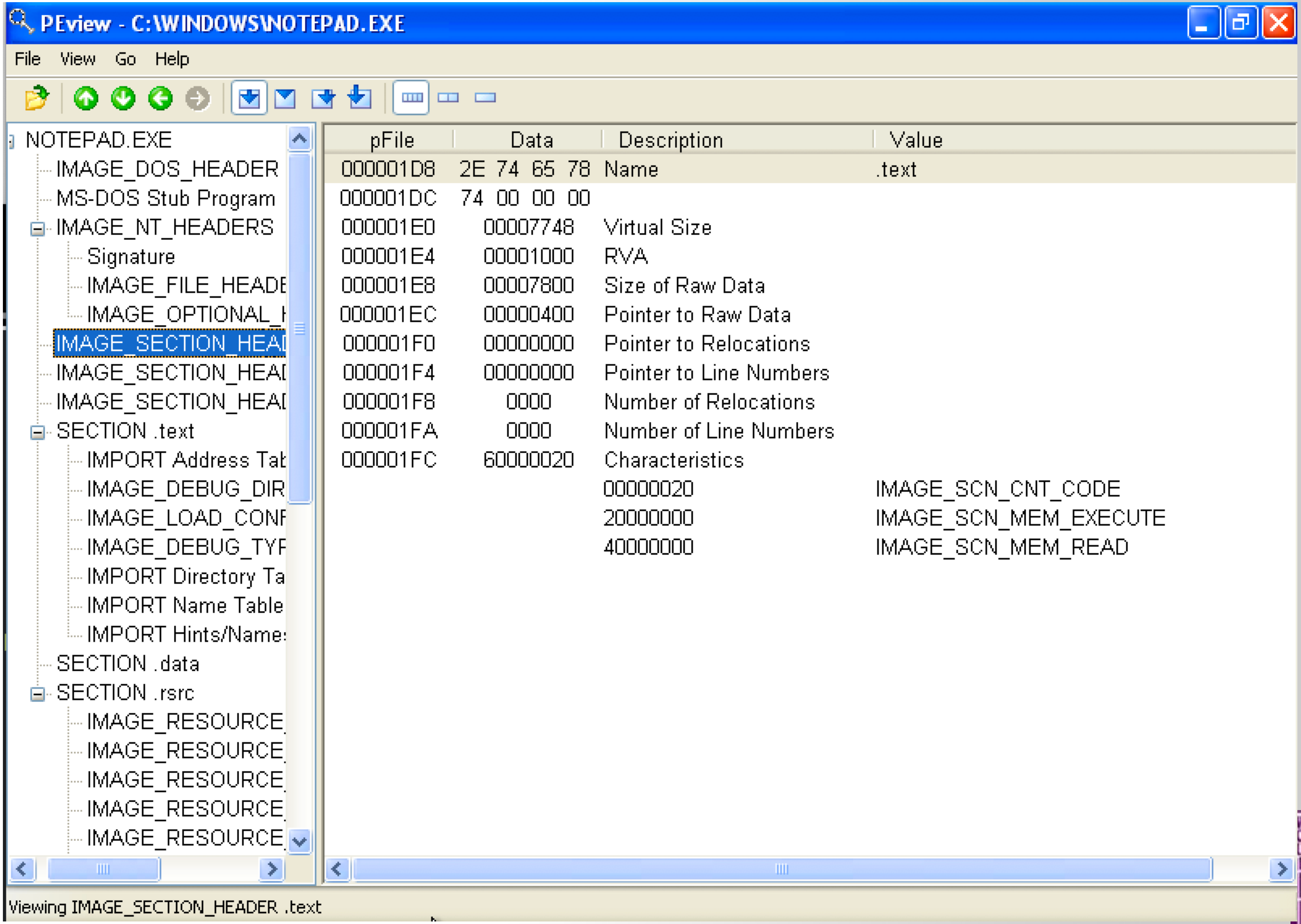


```
import pefile
import sys

malware_file = sys.argv[1]
pe = pefile.PE(malware_file)
for section in pe.sections:
    print "Name: %s VirtualSize: %s VirtualAddr: %s  SizeofRawData: %s PointerToRawData: %s" %
                                        (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
```

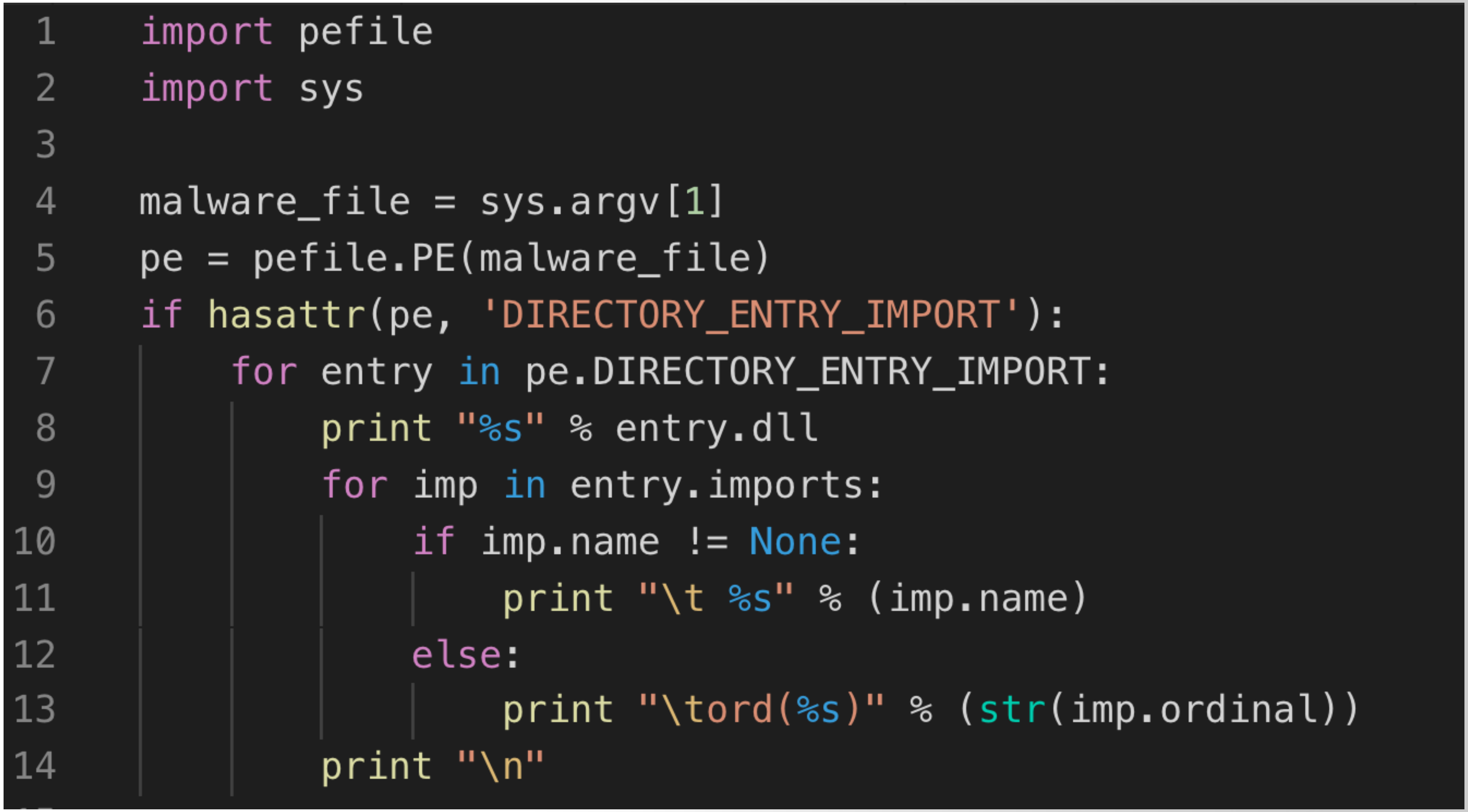
Inspecting PE Header Information



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	
		00000020	IMAGE_SCN_CNT_CODE
		20000000	IMAGE_SCN_MEM_EXECUTE
		40000000	IMAGE_SCN_MEM_READ

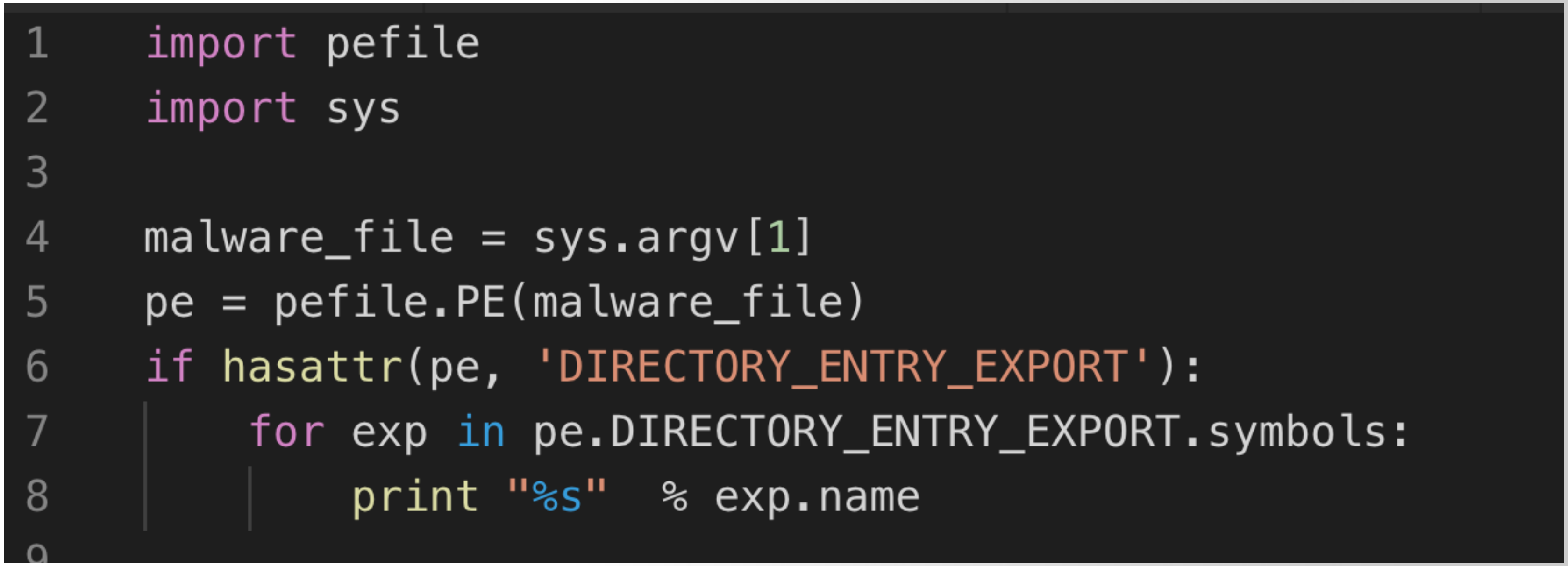
Viewing IMAGE_SECTION_HEADER .text

Inspecting file imports with pefile library



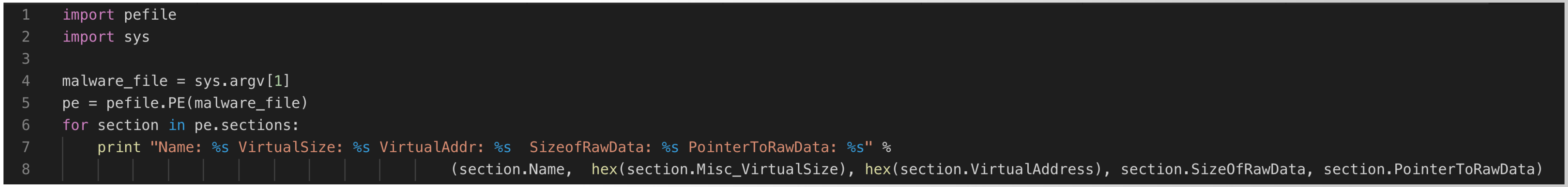
```python
import pefile
import sys

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" % (imp.name)
            else:
                print "\tord(%s)" % (str(imp.ordinal))
        print "\n"
```

Inspecting file export with pefile library

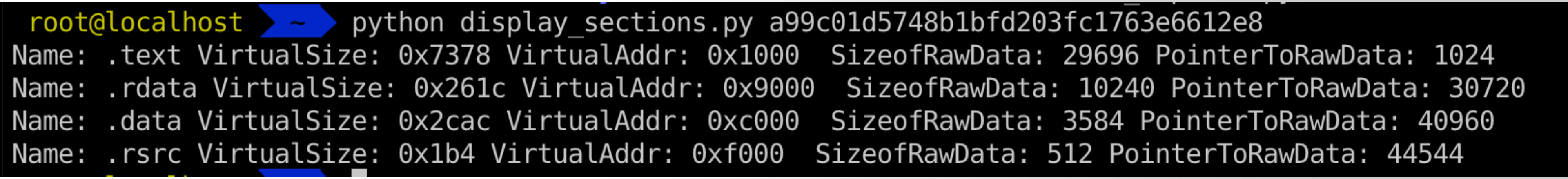


```
import pefile
import sys

malware_file = sys.argv[1]
pe = pefile.PE(malware_file)
if hasattr(pe, 'DIRECTORY_ENTRY_EXPORT'):
    for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:
        print "%s"  % exp.name
```

Inspecting PE Header Information in Linux

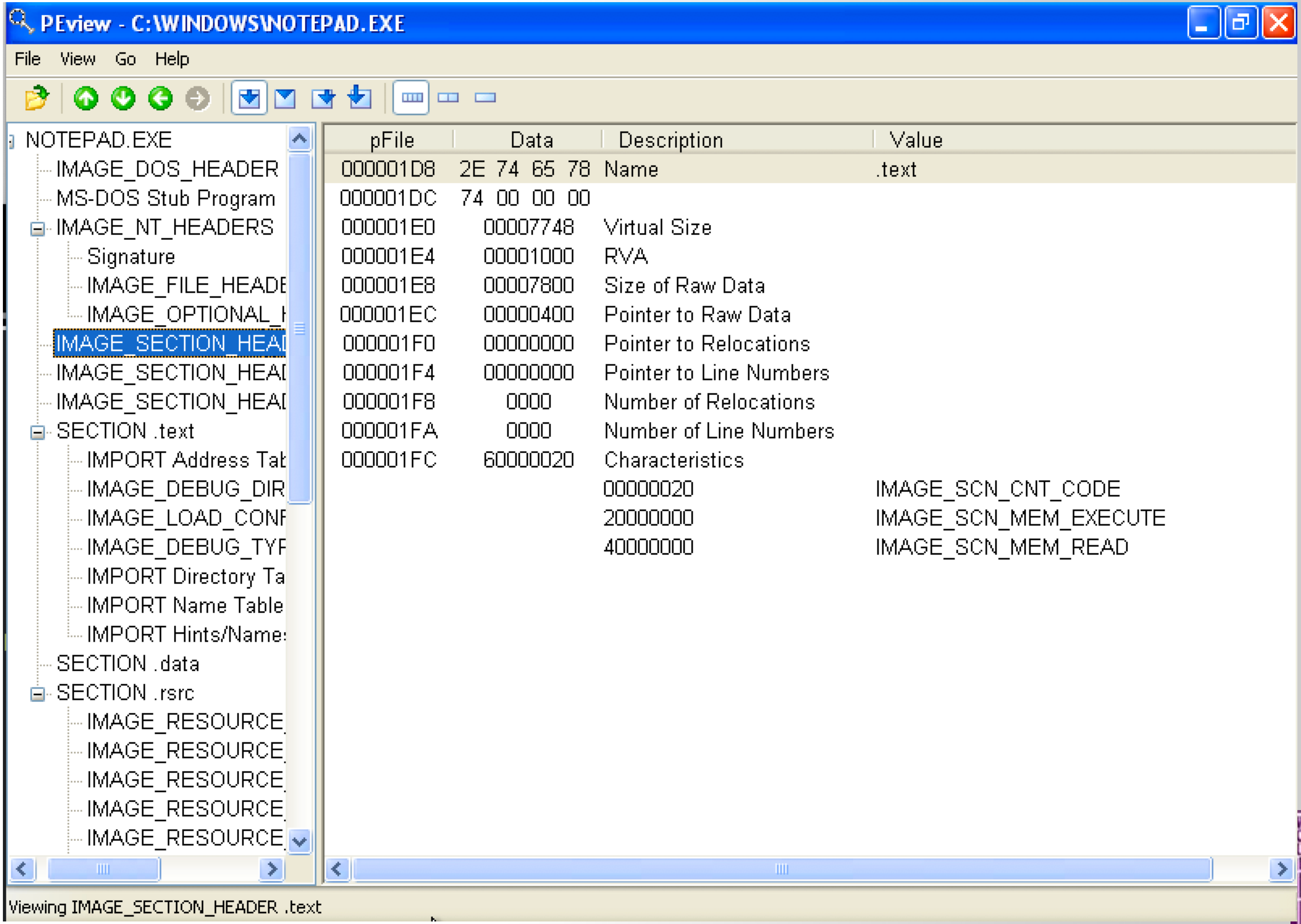


```python
import pefile
import sys

malware_file = sys.argv[1]
pe = pefile.PE(malware_file)
for section in pe.sections:
    print "Name: %s VirtualSize: %s VirtualAddr: %s  SizeofRawData: %s PointerToRawData: %s" %
                                        (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



```
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
                          00000020                    IMAGE_SCN_CNT_CODE
                          20000000                    IMAGE_SCN_MEM_EXECUTE
                          40000000                    IMAGE_SCN_MEM_READ
```

Viewing IMAGE_SECTION_HEADER .text