

THE SH3LLC0D3R'S BLOG




HOME CONTACT CTF WALKTHROUGHS EXPLOIT DEVELOPMENT MOBILE SECURITY NETWORK

SECURITYTUBE - LINUX ASSEMBLY EXPERT 32-BIT SECURITYTUBE - OFFENSIVE IOT EXPLOITATION SECURITYTUBE EXAMS

CISCO EMBEDDED

Home / Windows Reverse Shell / Windows Reverse Shell Shellcode II.

Windows Reverse Shell Shellcode II.

 February 27, 2016  elcapitan  Windows Reverse Shell

In my previous [post](#) I created a windows reverse shell shellcode. The shellcode was dependent on the windows version as it contained hardcoded function addresses. In order to create a version independent shellcode, we have to get the base address of kernel32.dll first, then we have to get the address of the **LoadLibraryA** and **GetProcAddress** functions. With this information we can get the address of any function and we can create a version independent shellcode.

In this post I will show you how the base address of the kernel32.dll can be determined. The best source of information I found was [this pdf](#). The pdf describes

This blog is dedicated to my research and experimentation on ethical hacking. The methods and techniques published on this site should not be used to do illegal things. I do not take responsibility for acts of other people.

three way to get kernel32 base address. Before we start, we have to understand a few things, like the TEB and PEB, and the SEH chain, etc.

TEB and PEB

Every thread stores information in a data structure, called TEB (Thread Environment Block). In case of x86, the TEB of the current thread can be accessed as an offset of FS segment register. More information on TEB can be found [here](#). TEB contains a pointer to the PEB (Process Environment Block) and can be accessed as FS: [0x30]. The description of the PEB structure can be found [here](#).

OllyDbg 1.1 and Immunity are outdated, but OllyDbg 2.01 has a feature, that it displays these structures in a more understandable way. The address of TEB is 0x7FFDE000, as it can be seen in the FS register.

```
Registers (FPU)
EAX 00341EA0
ECX 77C418BF msvcrt.77C418BF
EDX 77C61B78 msvcrt.77C61B78
EBX 00000001
ESP 0022FEC8
EBP 0022FEF8
ESI 00341F58 ASCII " 4"
EDI 00000039
EIP 00403013 test.00403013
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 8866 NULL
D 0
O 0 LastErr 000000CB ERROR_ENVVAR_NOT_FOUND
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty +UNORM 0002 00000038 00732D98
ST1 empty 0.0000000078275248720e-4933
```

The address of PEB is 0x7FFDF000.

RECENT POSTS

Androguard usage

How to debug an iOS application with Appmon and LLDB

OWASP Uncrackable – Android Level3

OWASP Uncrackable – Android Level2

How to install Appmon and Frida on a Mac

CATEGORIES

Android (5)

Fusion (2)

IoT (13)

Main (3)

Mobile (6)

Protostar (24)

SLAE32 (8)

VulnServer (6)

Windows Reverse Shell (2)

Memory map								
Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
7FFB0000	00024000			Code pages	Map	R	R	
7FFDE000	00001000			Data block of main thread	Priv	RW	RW	
7FFDF000	00001000			Process Environment Block	Priv	RW	RW	
7FFE0000	00001000				Priv	R	R	
80000000	7FFF0000			Kernel memory	Kern			

The structures in memory:

Dump - 7FFDE000..7FFDEFFF			
Address	Hex dump	Decoded data	Comments
7FFDE000	• E0FF2200 DD 0022FFE0		SEH chain = 22FFE0 -> {Next=FFFFFFFF,Handler=kernel32.7C8399F3}
7FFDE004	• 00002300 DD 00230000		Thread's stack base = ASCII "Actx "
7FFDE008	• 00E02200 DD 0022E000		Thread's stack limit = 22E000
7FFDE00C	• 00000000 DD 00000000		TIB of OS/2 Subsystem = NULL
7FFDE010	• 001E0000 DD 00001E00		Fiber data = 00001E00
7FFDE014	• 00000000 DD 00000000		Arbitrary user data = 0
7FFDE018	• 00E0F071 DD 7FFDE000		TIB linear address = 7FFDE000
7FFDE01C	• 00000000 DD 00000000		Environment pointer = NULL
7FFDE020	• 2C070000 DD 0000072C		Process ID = 0000072C
7FFDE024	• 98070000 DD 00000798		Thread ID = 00000798
7FFDE028	• 00000000 DD 00000000		RPC handle = 00000000
7FFDE02C	• 182A2400 DD 00242A18		TLS array = 00242A18
7FFDE030	• 00F0F071 DD 7FFDF000		Process database = 7FFDF000
7FFDE034	• CB000000 DD 000000CB		Thread's last error = ERROR_ENVVAR_NOT_FOUND
7FFDE038	• 00000000 DD 00000000		Number of critical sections = 0
7FFDE03C	• 00000000 DD 00000000		CSR client thread = NULL
7FFDE040	• 00000000 DD 00000000		Thread information = 0
7FFDE044	• 00000000 DD 00000000		Client information[31.] = 0
7FFDE048	• 00000000 DD 00000000		
7FFDE04C	• 00000000 DD 00000000		

The value at 7FFDE030 is the address of the PEB, which is 7FFDF000.

D Dump - 7FFDF000..7FFDFFFF			
Address	Hex dump	Decoded data	Comments
7FFDF000	• 00	00 00	InheritedAddressSpace = 0
7FFDF001	• 00	00 00	ReadImageFileExecOptions = 0
7FFDF002	• 01	00 01	BeingDebugged = TRUE
7FFDF003	• 00	00 00	SpareBool = FALSE
7FFDF004	• FFFFFFFF	00 FFFFFFFF	Mutant = INVALID_HANDLE_VALUE
7FFDF008	• 00004000	00 00FF test.<STRUCT	ImageBaseAddress = 00400000
7FFDF00C	• A01E3400	00 00341EA0	LoaderData = 341EA0
7FFDF010	• 00000200	00 00020000	ProcessParameters = 20000
7FFDF014	• 00000000	00 00000000	SubSystemData = NULL
7FFDF018	• 00002400	00 00240000	ProcessHeap = 00240000
7FFDF01C	• C0E4977C	00 00FFSET ntdll.7C97E4C0	FastPebLock = ntdll.7C97E4C0
7FFDF020	• 0510907C	00 ntdll.RtlEnterCritical	FastPebLockRoutine = ntdll.RtlEnterCriticalSection
7FFDF024	• ED10907C	00 ntdll.RtlLeaveCritical	FastPebUnlockRoutine = ntdll.RtlLeaveCriticalSection
7FFDF028	• 01000000	00 00000001	EnvironmentUpdateCount = 1
7FFDF02C	• 00000000	00 00000000	KernelCallbackTable = NULL
7FFDF030	• 00000000	00 00000000	Reserved = 0
7FFDF034	• 00000000	00 00000000	ThunksOrOptions = 0
7FFDF038	• 00000000	00 00000000	FreeList = 0
7FFDF03C	• 00000000	00 00000000	TlsExpansionCounter = 0
7FFDF040	• 80E4977C	00 00FFSET ntdll.7C97E480	TlsBitmap = ntdll.7C97E480
7FFDF044	• 03000000	00 00000003	TlsBitmapBits[2] = 3
7FFDF048	• 00000000	00 00000000	
7FFDF04C	• 00006F7F	00 7F6F0000	ReadOnlySharedMemoryBase = 7F6F0000
7FFDF050	• 00006F7F	00 7F6F0000	ReadOnlySharedMemoryHeap = 7F6F0000
7FFDF054	• 88066F7F	00 7F6F0688	ReadOnlyStaticServerData = 7F6F0688
7FFDF058	• 0000FB7F	00 7FFB0000	AnsiCodePageData = 7FFB0000
7FFDF05C	• 0010FC7F	00 7FFC1000	OemCodePageData = 7FFC1000
7FFDF060	• 0020FD7F	00 7FFD2000	UnicodeCaseTableData = 7FFD2000
7FFDF064	• 01000000	00 00000001	NumberOfProcessors = 1
7FFDF068	• 70000000	00 00000070	NtGlobalFlag = 112.
7FFDF06C	• 00000000	00 00000000	Reserved = 0
7FFDF070	• 00809B07	00 079B8000	CriticalSectionTimeout.Lo = 79B8000
7FFDF074	• 6DE8FFFF	00 FFFF8E6D	CriticalSectionTimeout.Hi = -1793
7FFDF078	• 00001000	00 00100000	HeapSegmentReserve = 1048576.
7FFDF07C	• 00200000	00 00002000	HeapSegmentCommit = 8192.
7FFDF080	• 00000100	00 00010000	HeapDeCommitTotalFreeThreshold = 65536.
7FFDF084	• 00100000	00 00001000	HeapDeCommitFreeBlockThreshold = 4096.
7FFDF088	• 04000000	00 00000004	NumberOfHeaps = 4
7FFDF08C	• 10000000	00 00000010	MaximumNumberOfHeaps = 16.
7FFDF090	• 80DE977C	00 00FFSET ntdll.7C97DE80	ProcessHeaps = 7C97DE80
7FFDF094	• 00000000	00 00000000	GdiSharedHandleTable = NULL
7FFDF098	• 00000000	00 00000000	ProcessStarterHelper = NULL
7FFDF09C	• 00000000	00 00000000	GdiDCAttributeList = 0
7FFDF0A0	• D8C0977C	00 00FFSET ntdll.7C97C0D0	LoaderLock = 7C97C0D0
7FFDF0A4	• 05000000	00 00000005	OSMajorVersion = 5
7FFDF0A8	• 01000000	00 00000001	OSMinorVersion = 1
7FFDF0AC	• 280A	00 0A28	OSBuildNumber = 2600.
7FFDF0AE	• 0002	00 0020	OSCSDVersion = 512.
7FFDF0B0	• 02000000	00 00000002	OSPlatformId = 2
7FFDF0B4	• 03000000	00 00000003	ImageSubsystem = 3
7FFDF0B8	• 04000000	00 00000004	ImageSubsystemMajorVersion = 4
7FFDF0BC	• 00000000	00 00000000	ImageSubsystemMinorVersion = 0
7FFDF0C0	• 00000000	00 00000000	ImageProcessAffinityMask = 0
7FFDF0C4	• 00000000	00 00000000	GdiHandleBuffer[34.] = 0
7FFDF0C8	• 00000000	00 00000000	

SEH chain

The first element of the TEB is an address that points to the bottom of the SEH chain. SEH chain can be viewed in OllyDbg and Immunity.

SEH chain of main thread	
Address	SE handler
0022FFE0	kernel32.7C8399F3

Each element in the SEH chain consists of two addresses. The first address points to the next address, if the element is not the last one, or 0xFFFFFFFF if the element is the last one in the chain. The second address is the exception handler function. The structure in the memory and in the stack:

Address	Hex dump	ASCII
0022FFE0	FF FF FF FF F3 99 83 7C	soa!
0022FFE8	55 6D 81 7C 00 00 00 00	Xmü!....
0022FFF0	00 00 00 00 00 00 00 00
0022FFF8	E0 14 40 00 00 00 00 00	10@....
0022FFD4	805430FD	==TV
0022FFD8	0022FFC8	is "
0022FFDC	85D5B1F0	0%P
0022FFE0	FFFFFFFF	End of SEH chain
0022FFE4	7C8399F3	soa! SE handler
0022FFE8	7C816D58	Xmü! kernel32.7C816D58
0022FFEC	00000000
0022FFF0	00000000
0022FFF4	00000000
0022FFF8	004014E0	10. test.<ModuleEntryPoint>
0022FFFC	00000000

As it can be seen, the address of the bottom of the SEH chain is 0x0022FFE0. The first address is 0xFFFFFFFF. This means this is the end of the chain and there is no more element. The second address is 0x7C8399F3. This is an address in the kernel32 DLL to a default exception handler. The default exception handler can be overwritten with a custom one. In this case the bottom does not point to an address in the kernel32 DLL.

If an exception is raised, the operating system calls each exception handler function until the exception is handled or the bottom is reached. The return value of the exception handler tells if the exception is handled.

Stack

The second element of the TEB is an address, which is the bottom of the stack. This means, if we push a value onto the stack, the stack value is decremented with the size of the pushed value. As the stack grows towards zero, the stack can be found in front of this address.

Address	Hex dump	ASCII		
7FFD0000	E0 FF 22 00 00 00 23 00	α" . . #.	0022FFA0	00000006
7FFD0008	00 E0 22 00 00 00 00 00	α"	0022FFA4	EEB5D004
7FFD0010	00 1E 00 00 00 00 00 00	α"	0022FFA8	EEB5D004
7FFD0018	00 00 FD 7F 00 00 00 00	α"	0022FFAC	80615CDB
7FFD0020	A4 0A 00 00 B4 08 00 00	α"	0022FFB0	7C90E64E
7FFD0028	00 00 00 00 18 2A 24 00	α"	0022FFB4	7C816D4C
7FFD0030	00 E0 FD 7F CB 00 00 00	α"	0022FFB8	FFFFFFFF
7FFD0038	00 00 00 00 00 00 00 00	α"	0022FFBC	00000009
7FFD0040	00 00 00 00 00 00 00 00	α"	0022FFC0	0022FFF8
7FFD0048	00 00 00 00 00 00 00 00	α"	0022FFC4	7C816D4F
7FFD0050	00 00 00 00 00 00 00 00	α"	0022FFC8	0069006E
7FFD0058	00 00 00 00 00 00 00 00	α"	0022FFCC	00790074
7FFD0060	00 00 00 00 00 00 00 00	α"	0022FFD0	7FFDE000
7FFD0068	00 00 00 00 00 00 00 00	α"	0022FFD4	805430FD
7FFD0070	00 00 00 00 00 00 00 00	α"	0022FFD8	0022FFC8
7FFD0078	00 00 00 00 00 00 00 00	α"	0022FFDC	8E8F4F0
7FFD0080	00 00 00 00 00 00 00 00	α"	0022FFE0	FFFFFFFF
7FFD0088	00 00 00 00 00 00 00 00	α"	0022FFE4	7C8399F2
7FFD0090	00 00 00 00 00 00 00 00	α"	0022FFE8	7C816D58
7FFD0098	00 00 00 00 00 00 00 00	α"	0022FFEC	00000000
7FFD00A0	00 00 00 00 00 00 00 00	α"	0022FFF0	00000000
7FFD00A8	00 00 00 00 00 00 00 00	α"	0022FFF4	00000000
7FFD00B0	00 00 00 00 00 00 00 00	α"	0022FFF8	004014E0
7FFD00B8	00 00 00 00 00 00 00 00	α"	0022FFFC	00000000

The address of the stack is x00230000. The first element of the stack is at 0x0022FFFC, the second element is at 0x0022FFF8, etc.

LoaderData

PEB stores an address at 0x0C, after the ImageBaseAddress, which is usually 0x00400000. This address is the beginning of the LoaderData structure. This structure stores three lists of loaded modules.

Getting kernel32 base address

The pdf I mentioned explains three possible ways of getting the kernel32 base address:

1. PEB
2. SEH
3. TOPSTACK

PEB method

The PEB method uses the fact, that the second initialized module is the kernel32.dll. We get the address of PEB through FS segment register, then we get the address of LoaderData, finally we get the base address of kernel32 module, which is the second in the InitializationOrderModuleList list.

The assembly code:

```
global _start

section .text

_start:
    int 0x03

find_kernel32_peb:
    push esi
    xor eax, eax
    mov eax, [fs:eax+0x30] ; Get the address of PEB
```

```

mov eax, [eax + 0x0c] ; Get the address of LoaderData
mov esi, [eax + 0x1c] ; Get the InitializationOrderModule
lods
mov eax, [eax + 0x8] ; Get the address of second module

pop esi
ret

```

The INT3 is a breakpoint, so that the execution will stop after this point.

According to the paper, this method is the most reliable and works most of the case. I tested it and it works on Windows XP SP2, however it does not work on Windows 7. On Windows 7, the kernel32.dll is the third initialized module. The assembly code for Windows 7:

```

global _start

section .text

_start:
    int 0x03

find_kernel32_peb:
    push esi
    xor eax, eax
    mov eax, [fs:eax+0x30] ; Get the address of PEB

    mov eax, [eax + 0x0c] ; Get the address of LoaderData
    mov esi, [eax + 0x1c] ; Get the InitializationOrderModule
    lods ; Get the second module

```



```
mov esi, eax
lods      ; Get the third module
mov eax, [eax + 0x8] ; Get the address of second module

pop esi
ret
```

The shellcode is also different for Windows 95 (see in the pdf). I did not test it.

SEH method

The SEH method is based on the fact, that the bottom of the SEH chain contains a default exception handler, which can be found in kernel32 module. If we get the address, then we only have to find the PE header of the module. The PE header starts with the MZ signature (0x5A4D):

Memory map								
Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
00022000	00001000				Priv	??? GUA	RW	
00022E000	00002000			stack of ma	Priv	RW	GUA	RW
000230000	00001000							
000240000	00001000							
000340000	00001000							
000350000	00001000							
000360000	00001000							
000380000	00001000							
0003C0000	00001000							
0003D0000	00001000							
0003E0000	00001000							
000400000	00001000							
000401000	00001000							
000403000	00001000							
000404000	00001000							
000405000	00001000							
000406000	00001000							
000407000	00001000							
000408000	00001000							
000409000	00001000							
00040A000	00001000							
00040B000	00001000							
00040C000	00001000							
00040D000	00001000							
00040E000	00001000							
00040F000	00001000							
000410000	00001000							
000411000	00001000							
000412000	00001000							
000413000	00001000							
000414000	00001000							
000415000	00001000							
000416000	00001000							
000417000	00001000							
000418000	00001000							
000419000	00001000							
00041A000	00001000							
00041B000	00001000	test	/92		Inag	R	RWE	
000420000	00001000				Map	R	R	
77C10000	00001000	msvcrt		PE header	Inag	R	RWE	
77C11000	00001000	msvcrt	.text	code,import	Inag	R E	RWE	
77C5D000	00007000	msvcrt	.data	data	Inag	RW	Copy	
77C64000	00001000	msvcrt	.rsrc	resources	Inag	R	RWE	
77C65000	00003000	msvcrt	.reloc	relocations	Inag	R	RWE	
7C800000	00001000	kernel32		PE header	Inag	R	RWE	
7C801000	00002000	kernel32	.text	code,import	Inag	R E	RWE	
7C803000	00005000	kernel32	.data	data	Inag	RW	RWE	
7C808000	00006000	kernel32	.rsrc	resources	Inag	R	RWE	
7C8EE000	00006000	kernel32	.reloc	relocations	Inag	R	RWE	
7C900000	00001000	ntdll		PE header	Inag	R	RWE	
7C901000	00007000	ntdll	.text	code,export	Inag	R E	RWE	
7C97C000	00005000	ntdll	.data	data	Inag	RW	RWE	
7C981000	00002000	ntdll	.rsrc	resources	Inag	R	RWE	
7C9AD000	00003000	ntdll	.reloc	relocations	Inag	R	RWE	
7F6F0000	00007000				Map	R E	R E	
7FFB0000	000024000				Map	R	R	
7FFDE000	00001000			data block	Priv	RW	RW	
7FFDF000	00001000				Priv	RW	RW	
7FFE0000	00001000				Priv	R	R	

The SEH chain can be get from the TEB. It is the first entry in TEB.

The shellcode:

TOPSTACK method

The TOPSTACK method exploits the fact, that the stack contains an address, which points to the kernel32 module. The address is at a constant offset, however it is not

guaranteed to work. This is the least reliable way of getting the base address of kernel32 module.

If we have an address, which points into the kernel32 module, we still have to find the PE header. We do the same thing as presented in the SEH method: we search for the PE header, which is an MZ signature.

The assembly code:

```
global _start

section .text

_start:
    int 0x03

find_kernel32_topstack:
    push esi
    xor esi, esi
    mov esi, [fs:esi + 0x18]
    lodsd
    lodsd
    mov eax, [eax - 0x1c]
find_kernel32_topstack_base:
find_kernel32_topstack_base_loop:
    dec eax
    xor ax, ax
    cmp word [eax], 0x5a4d
    jne find_kernel32_topstack_base_loop
find_kernel32_topstack_base_finished:
```

```
pop esi  
ret
```

[« PREVIOUS POST](#)[NEXT POST »](#)

Copyright © 2019, The sh3llc0d3r's blog. Proudly powered by
[WordPress](#). Blackoot design by [Iceable Themes](#).

[Home](#) [Contact](#) [CTF walkthroughs](#) [Exploit development](#)
[Mobile Security](#) [Network](#)
[SecurityTube – Linux Assembly Expert 32-bit](#)
[SecurityTube – Offensive IoT Exploitation](#) [SecurityTube exams](#)
[CISCO](#) [Embedded](#)