Q

THE SH3LLCOD3R'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.



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OllyDbg 1.1 and Immunity are outdated, but OllyDbg 2.01 has a feature, that it displays these structures in a more understandable way.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.

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)

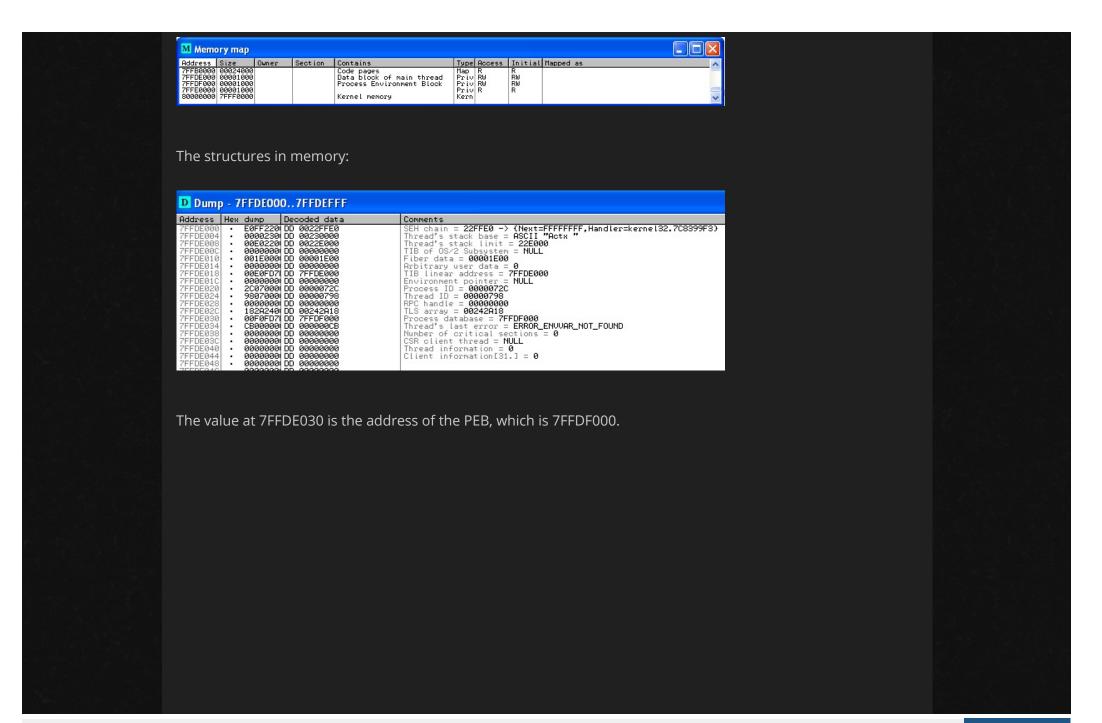
Mobile (6)

Protostar (24)

SLAE32 (8)

VulnServer (6)

Windows Reverse Shell (2)



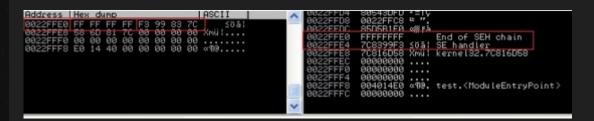


SEH chain

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.



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:

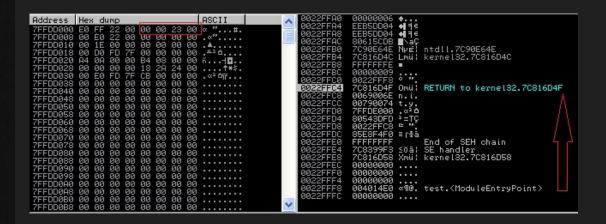


As it can be seen, the address of the bottom of the SEH chain is 0x0022FFE0. The first address is 0xFFFFFFF. This means this is the end of the chain and there is no more element. The second address is 0x7C8399F3. This is an address in the kernel32 DLL to a default exception handler. The default exception 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.



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 LoaderDat;
mov esi, [eax + 0x1c] ; Get the InitializationOrderMolodsd ; Get the second module
mov eax, [eax + 0x8] ; Get the address of second mod

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 InitializationOrderModula
    lodsd ; Get the second module
```

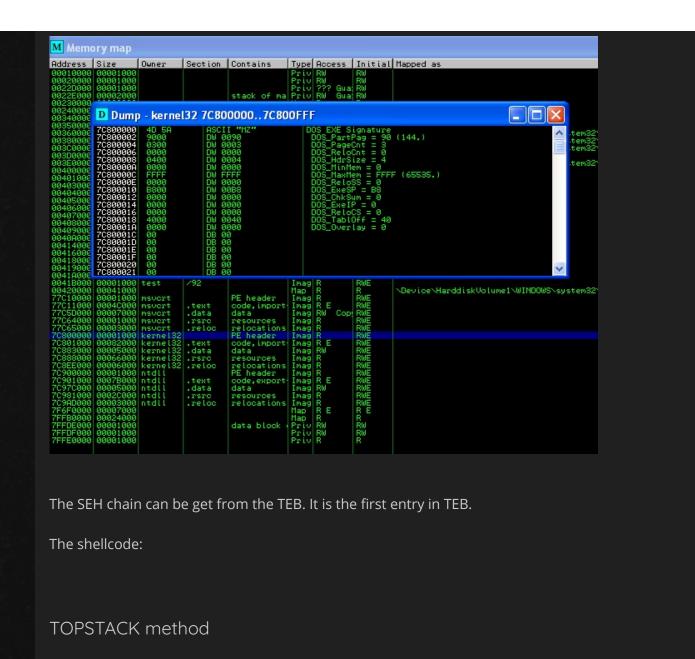
```
mov esi, eax
lodsd ; Get the third module
mov eax, [eax + 0x8] ; Get the address of second module

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):



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 moduel, 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:
```

