Vista&Win7 下 CreateRemoteThread 应用的若干问题和解决方案

在 xp、03 下面，用 CreateRemoteThread 往别的进程注入 shellcode 或者是 dll 是一件非常容易的事情，甚至是往系统进程里面注入（如 svchost、winlogon 等），但在 vista 下，你会发现事情没有那么容易了，就算是在 xp 下一模一样的代码，在 vista 下面都有可能给你一个 ERROR_NOT_ENOUGH_MEMORY ,error Id 是 8，中文解释是“存储空间不足，无法处理此命令。”通常发生的情况是在 2 个进程的 sessionid 不一样的时候。搜索网上的资料，原来早有前辈解决过这个问题。

链接一：http://www.cnasm.com/view.asp?classid=51&newsid=292 从无花果前辈的分析来看，他分析的是 vista 版本的。我们暂且以此为例来进行说明。

```
.text:7C8105B0                  cmp     [ebp-3ACh], ebx// 这里发生了跳转 ,引起 CreateRemoteThread 失败。
.text:7C8105B6                  jl      loc_7C83ABD3
```

这里进行的比较和跳转要处理的方式有很多中，MJ0011 大牛也给了一种办法，原文链接在这里 http://bbs.pediy.com/showthread.php?t=88454 从文件 MircoCode.rar-》xxload.exe 里面可以找到。

在

```
.text:00401685          lea     edx, [ebp+Buffer]
.text:0040168B          push    ecx
.text:0040168C          push    edx
.text:0040168D          call    sub_4011B0
```

sub_4011B0 函数里面有对上述比较地方进行处理的代码，我们跟进看看

在

```
.text:004013D4          mov     eax, ds:CreateRemoteThread
.text:004013D9          cmp     word ptr [eax+edi], 1D38h
.text:004013DF          jnz     short loc_4013F2
.text:004013E1          mov     edx, [eax+edi+2]
.text:004013E5          push    1                       ; ucb
.text:004013E7          push    edx                     ; lp
.text:004013E8          call    ds:IsBadWritePtr
.text:004013EE          test    eax, eax
.text:004013F0          jz      short loc_401407
.text:004013F2
.text:004013F2 loc_4013F2:                              ; CODE XREF: sub_4011B0+22Fj
.text:004013F2          inc     edi
.text:004013F3          cmp     edi, 300h
.text:004013F9          jb      short loc_4013D4
.text:004013FB          mov     edx, [esp+44h+dwSize]
.text:004013FF          push    4000h
.text:00401404          push    edx
```

```
.text:00401405                          jmp         short loc_40147F
.text:00401407 ; ---------------------------------------------------------------------
.text:00401407
.text:00401407  loc_401407:                                                    ; CODE  XREF:
sub_4011B0+240 j
.text:00401407                          mov         eax, ds:CreateRemoteThread
.text:0040140C                          mov         edi, [eax+edi+2]
.text:00401410                          test        edi, edi
.text:00401412                          mov         cl, [edi]
.text:00401414                          mov         byte ptr [edi], 1
.text:00401417                          mov         byte ptr [esp+44h+arg_4], cl
```

这段是进行处理的过程，如果你觉得这样不好看，下面我们简单把它翻译成 C 语言的。

```
BOOL  ModifyCreateRemoteThreadByMJ0011()
{
    //for vista
    //win7 disable
    DWORD index = 0;
    PCHAR StartAddress = (PCHAR)CreateRemoteThread;
    for(index = 0;index<0x300;index++)
    {
        if(*(PUSHORT)(StartAddress+index)==0x1D38
            &&IsBadWritePtr((PVOID)(StartAddress+index+2),1))
        {
            printf("OK:%d!\n",index);
            *((PCHAR)(*(PDWORD)(StartAddress+index+2))) = 1;
            return TRUE;
        }
    }
    printf("没有找到!\n");
    return FALSE;
};
```

里面的 printf 是我加进去的。意思很简单，从 CreateRemoteThread 开始，最多找 0x300 个字节，找到 0x1D38 后，将某个值赋值为 1，现在我们回头再看 CreateRemoteThread 里面的 0x1D38 是个什么语句。

```
76454826      8981 600F0000             mov       dword ptr [ecx+F60], eax
7645482C      381D 40EF4D76              cmp       byte ptr [764DEF40], bl
76454832    ^ 0F85 28E3FBFF             jnz       76412B60
76454838      8B45 B4                   mov       eax, dword ptr [ebp-4C]
7645483B      8985 C0FEFFFF             mov       dword ptr [ebp-140], eax
76454841      8B45 94                   mov       eax, dword ptr [ebp-6C]
76454844      8985 C4FEFFFF             mov       dword ptr [ebp-13C], eax
7645484A      8B45 98                   mov       eax, dword ptr [ebp-68]
```

```
7645484D    8985 C8FEFFFF        mov     dword ptr [ebp-138], eax
76454853    6A 0C                push    0C
76454855    68 01000100          push    10001
7645485A    53                   push    ebx
7645485B    8D85 98FEFFFF        lea     eax, dword ptr [ebp-168]
76454861    50                   push    eax
76454862    FF15 4C104176        call    dword ptr [<&ntdll.CsrClientCall>;
ntdll.CsrClientCallServer
```

如果你仔细看的话，在第 2 句代码哪里就是 381D ，MJ0011 的代码就是将里面的数据 0x764DEF40 取出来，作为地址，赋值为 1.从而实现远程线程的顺利注入。但是这段代码在 win7 下面却又行不通了，因为根本找不到 0x1D38，就不能进行 patch。那么 win7 下面的 CreateRemoteThread 又是个什么样子呢？
看下面。

```
.text:77E6F403    mov     edi, edi
.text:77E6F405    push    ebp
.text:77E6F406    mov     ebp, esp
.text:77E6F408    push    [ebp+lpThreadId]
.text:77E6F40B    push    0
.text:77E6F40D    push    [ebp+dwCreationFlags]
.text:77E6F410    push    [ebp+lpParameter]
.text:77E6F413    push    [ebp+lpStartAddress]
.text:77E6F416    push    [ebp+dwStackSize]
.text:77E6F419    push    [ebp+lpThreadAttributes]
.text:77E6F41C    push    [ebp+hProcess]
.text:77E6F41F    call    CreateRemoteThreadEx_0
.text:77E6F424    pop     ebp
.text:77E6F425    retn    1Ch
```

就这么一段，中间调用了 CreateRemoteThreadEx，然后返回了。
而 CreateRemoteThreadEx 仅仅是从 API-MS-WIN-CORE-PROCESSTHREADS-L1-1-0.DLL（你的系统可能和我的不一样）引入的一个函数。所以 MJ0011 的那段代码在 win7 下是不能用了。

我们现在再回头看看无花果前辈给出的解决办法，自己用代码实现 CreateRemoteThread 的功能。测试一下。
在 vista、win7 下都能工作吧。不过就是一运行，目标程序就挂了，检查我们的代码并没有发现使你异常。那就来比较一下调用系统的 CreateRemoteThread 和调用 OsCreateRemoteThread(DWORD dwpid,StartAddress:pointer) 2 个有什么不一样的地方，很快我们发现（以 vista 为例），系统的调用结束后是用 ntdll.RtlExitUserThread 来结束当前线程的，而我们用 OsCreateRemoteThread 创建的并没有调用这个函数，而是返回到了一个莫名其妙的地方。OK，那我们在代码的最后自己来调用 ntdll.RtlExitUserThread 结束，OK，正常了。
但是我们要用的地方非常多啊，如果都去修改的话，不但容易错，将来维护也麻烦啊。那我

们就自己来实现这个桩函数吧。

```
const CHAR myBaseThreadInitThunk[] =
{
    //  00830000    8BFF            mov     edi, edi
    '\x8B','\xFF',
    //  00830002    55              push    ebp
    '\x55',
    //  00830003    8BEC            mov     ebp, esp
    '\x8B','\xEC',
    //  00830005    51              push    ecx     //ntdll.RtlExitUserThread
    '\x51',
    //  00830006    53              push    ebx     //参数
    '\x53',
    //  00830007    FFD0            call    eax     //函数地址
    '\xFF','\xD0',
    //  00830009    59              pop     ecx     //恢复结束函数地址
    '\x59',
    //  0083000A    50              push    eax     //将刚才的结果压栈
    '\x50',
    //  0083000B    FFD1            call    ecx     //调用 RtlExitUserThread 结
束
    '\xFF','\xD1',
    //  0083000D    90              nop
    '\x90'
};
```

根据以上的内容和参考无花果前辈的代码，我们可以实现这样一个函数。

```
HANDLE WINAPI OsCreateRemoteThread2(
                        HANDLE hProcess,
                        LPSECURITY_ATTRIBUTES lpThreadAttributes,
                        SIZE_T dwStackSize,
                        LPTHREAD_START_ROUTINE lpStartAddress,
                        LPVOID lpParameter,
                        DWORD dwCreationFlags,
                        LPDWORD lpThreadId)
{
    //by 80695073(QQ)
    //email kiss2008ufo@yahoo.com.cn
    CONTEXT         context = {CONTEXT_FULL};
    CLIENT_ID   cid;
    DWORD       ret;
    HANDLE          hThread = NULL;
    DWORD       StackReserve;
```

```
DWORD        StackCommit = 0x1000;
ULONG_PTR Stack = 0;
INITIAL_TEB InitialTeb;
ULONG        x;
const CHAR myBaseThreadInitThunk[] =
{
    //    00830000    8BFF                mov        edi, edi
    '\x8B','\xFF',
    //    00830002    55                  push       ebp
    '\x55',
    //    00830003    8BEC                mov        ebp, esp
    '\x8B','\xEC',
    //    00830005    51                  push       ecx     //ntdll.RtlExitUserThread
    '\x51',
    //    00830006    53                  push       ebx     //参数
    '\x53',
    //    00830007    FFD0                call       eax     //函数地址
    '\xFF','\xD0',
    //    00830009    59                  pop        ecx     //恢复结束函数地址
    '\x59',
    //    0083000A    50                  push       eax     //将刚才的结果压栈
    '\x50',
    //    0083000B    FFD1                call       ecx     //调用 RtlExitUserThread 结
束
    '\xFF','\xD1',
    //   0083000D    90                  nop
    '\x90'
};
PVOID   pBaseThreadThunk = NULL; //不能释放

//0、分配非 OS 的加载函数
StackReserve = 0x1000;
ret = ZwAllocateVirtualMemory(hProcess,
    /*&stack.ExpandableStackBottom*/(PVOID*)&pBaseThreadThunk,
    0,
    &StackReserve,
    MEM_COMMIT,
    PAGE_EXECUTE_READWRITE);
if (ret >= 0x80000000)
{
    //失败
    printf("Error IN OsCreateRemoteThread2 ZwAllocateVirtualMemory0 !\n");
    goto OsCreateRemoteThread2Ret;
    //end
```

```
}
ret = ZwWriteVirtualMemory(hProcess,
        pBaseThreadThunk,
        (LPVOID)myBaseThreadInitThunk,
        sizeof(myBaseThreadInitThunk),&x);
if (ret >= 0x80000000)
{
        //失败
        printf("Error IN OsCreateRemoteThread2 Zw AllocateVirtualMemory0 !\n");
        goto OsCreateRemoteThread2Ret;
        //end
}
cid.UniqueProcess = hProcess;

//1、准备堆栈
StackReserve = 0x10000;
ret = Zw AllocateVirtualMemory(hProcess,
        /*&stack.ExpandableStackBottom*/(PVOID*)&Stack,
        0,
        &StackReserve,
        MEM_RESERVE,
        PAGE_READWRITE);
if (ret >= 0x80000000)
{
        //失败
        printf("Error IN OsCreateRemoteThread2 Zw AllocateVirtualMemory1!\n");
        goto OsCreateRemoteThread2Ret;
        //end
}
printf("OK OsCreateRemoteThread2:Zw AllocateVirtualMemory 0x%08x\n",Stack);

InitialTeb.AllocatedStackBase = (PVOID)Stack;
InitialTeb.StackBase = (PVOID)(Stack + StackReserve);

/* Update the Stack Position */
Stack += StackReserve - StackCommit;

Stack -= 0x1000;
StackCommit += 0x1000;

/* Allocate memory for the stack */
ret = Zw AllocateVirtualMemory(hProcess,
        (PVOID*)&Stack,
        0,
```

```
            &StackCommit,
            MEM_COMMIT,
            PAGE_READWRITE);
        if (ret >= 0x80000000)
        {
            //失败
            printf("Error IN OsCreateRemoteThread2 Zw AllocateVirtualMemory2!\n");
            goto OsCreateRemoteThread2Ret;
            //end
        }
        printf("OK OsCreateRemoteThread2:Zw AllocateVirtualMemory 2 0x%08x\n",Stack);
        InitialTeb.StackLimit = (PVOID)Stack;


        StackReserve = 0x1000;
        ret = ZwProtectVirtualMemory(hProcess, (PVOID*)&Stack, &StackReserve,
PAGE_READWRITE | PAGE_GUARD, &x);
        if (ret >= 0x80000000)
        {
            //失败
            printf("Error IN OsCreateRemoteThread2 ZwProtectVirtualMemory!\n");
            goto OsCreateRemoteThread2Ret;
            //end
        }
        /* Update the Stack Limit keeping in mind the Guard Page */
        InitialTeb.StackLimit = (PVOID)((ULONG_PTR)InitialTeb.StackLimit - 0x1000);
        //2、准备 CONTEXT
//      CONTEXT context = {CONTEXT_FULL};
        ret = ZwGetContextThread(GetCurrentThread(),&context);
        if (ret >= 0x80000000)
        {
            //失败
            printf("Error IN OsCreateRemoteThread2 ZwGetContextThread!\n");
            goto OsCreateRemoteThread2Ret;
            //end
        }
        context.Esp = (ULONG)InitialTeb.StackBase;
        context.Eip = (ULONG)pBaseThreadThunk; //这里填写需要加载的地址，不过需要自己终
结自己
        context.Ebx = (ULONG)lpParameter;
        //other init
        //must
        context.Eax = (ULONG)lpStartAddress;
        context.Ecx = 0x778B0859;/*win7*///0x77AEEC01;/*vista*/ //ntdll.RtlExitUserThread
```

```
    context.Edx = 0x00000000; //nouse


    ret = ZwCreateThread(&hThread, THREAD_ALL_ACCESS, 0, hProcess, &cid, &context,
&InitialTeb, TRUE);
    if (ret >= 0x80000000)
    {
        //失败
        printf("Error IN OsCreateRemoteThread2 ZwCreateThread!\n");
        goto OsCreateRemoteThread2Ret;
        //end
    }
    if(lpThreadId)
    {
        *lpThreadId = (DWORD)cid.UniqueThread;
    }
    if (!(dwCreationFlags & CREATE_SUSPENDED))
    {
        ZwResumeThread(hThread, NULL);
    }
OsCreateRemoteThread2Ret:
    return hThread;
}
```

里面红色的部分是需要根据实际需要修改的，当然，你也可以这样

```
context.Ecx =(ULONG)GetProcAddress(GetModuleHandle("ntdll.dll"),"RtlExitUserThread");
```

简单解释一哈上面的代码。

//0 部分是准备桩函数用的。不详细解释

//1 部分是准备线程运行时候用的堆栈，其初始化的大小和方式来自 ReactOS 的 CreateRemoteThread-》BasepCreateStack。

//2 部分是准备线程运行的上下文的。可以查看 ReactOS 的 CreateRemoteThread-》BasepInitializeContext。这里为简便起见，获取了当前线程的 CONTEXT，然后根据实际情况设置了 ESP\EIP\EBP，EAX\ECX\EDX 为其他初始化情况。在这里可以不用修改。

从 myBaseThreadInitThunk 的代码来看，其要求 shellcode 具有如下形式：

DWORD WINAPI FuncName(PVOID lpParam){.....}形式。

到这里关于 CreateRemoteThread 的话题就说完了。

再说说在实际测试的时候发现的一些现象（问题？）吧。

1、我们调用 GetDesktopWindow 获取窗口句柄，再 GetWindowThreadProcessId 获取进程 ID 得到的将是 csrss.exe 的 PID，在 VISTA（WIN7）下获取的就是当前会话的 csrss.exe 的 PID，服务里面的就是 sessionID 为 0 的 csrss.exe 的 PID。但上述调用过程就是不能在 csrss.exe 自己内部调用，如果调用会导致 csrss.exe 异常退出。机器蓝屏。

2、OS 名称: Microsoft Windows 7 旗舰版

OS 版本: 6.1.7600 暂缺 Build 7600

以下代码失效,常用于 ShellCode 中查找 KERNEL32.DLL 的基址

```
HMODULE hKernel32;
__asm
{
    mov eax, fs:[0x30];
    mov eax, [eax + 0x0C];
    mov eax, [eax + 0x1C];
    mov eax, [eax];
    mov eax, [eax + 0x08];
    mov hKernel32, eax;
}
```

将找到 KernelBase.dll
有效系统 2K\XP\03\VISTA。

BY 飞心
2009.11.18