**微软本地提权漏洞cve\_2019\_1458分析**

# 漏洞简介

（漏洞性质；漏洞时间；影响范围）

该漏洞是windows平台EoP(Escalation of Privilege)本地权限提升漏洞，该漏洞是[Kaspersky Lab](https://www.kaspersky.com/)最早于2019年11月份捕获到的在野0day攻击。该漏洞首次公开披露于2019年12月10日。

受该漏洞影响的系统版本有：

Windows Server 2012 R2 (Server Core installation)、Windows Server 2012 R2、Windows Server 2012 (Server Core installation)、Windows Server 2012、Windows Server 2008 R2 for x64-based Systems Service Pack 1 (Server Core installation)、Windows Server 2008 R2 for x64-based Systems Service Pack 1、Windows Server 2008 R2 for Itanium-Based Systems Service Pack 1、Windows Server 2008 for x64-based Systems Service Pack 2 (Server Core installation)、Windows Server 2008 for x64-based Systems Service Pack 2、Windows Server 2008 for Itanium-Based Systems Service Pack 2、Windows Server 2008 for 32-bit Systems Service Pack 2 (Server Core installation)、Windows Server 2008 for 32-bit Systems Service Pack 2、Windows RT 8.1、Windows 8.1 for x64-based systems、Windows 8.1 for 32-bit systems、Windows 7 for x64-based Systems Service Pack 1、Windows 7 for 32-bit Systems Service Pack 1、Windows Server 2016 (Server Core installation)、Windows Server 2016、Windows 10 Version 1607 for x64-based Systems、Windows 10 Version 1607 for 32-bit Systems、Windows 10 for x64-based Systems、Windows 10 for 32-bit Systems。

修复补丁：<https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2019-1458> #Security Updates

# 漏洞成因

（漏洞发生点的位置；造成漏洞的成因综述）

该漏洞发生在windows的多用户管理驱动文件win32k.sys(路径：C:\Windows\System32\win32k.sys)中。当窗口对象被赋予某些特定值时，使得win32k.sys无法正确处理这些对象，从而可触发执行任意内存写操作，进而将其拓展为任意地址读写操作后就可通过替换token的方法来将已登录的普通权限用户提升至system权限。该漏洞的成因不同于通常的溢出漏洞，倒更像逻辑漏洞。

Win32k.sys文件介绍：该组件主要为应用层提供窗口管理和图形设备接口，win32k.sys向内核注册一组调用函数，介入到内核的进程线程运行，是user32.dll、GDI32.dll等用户态组件的内核实现。

# 漏洞分析

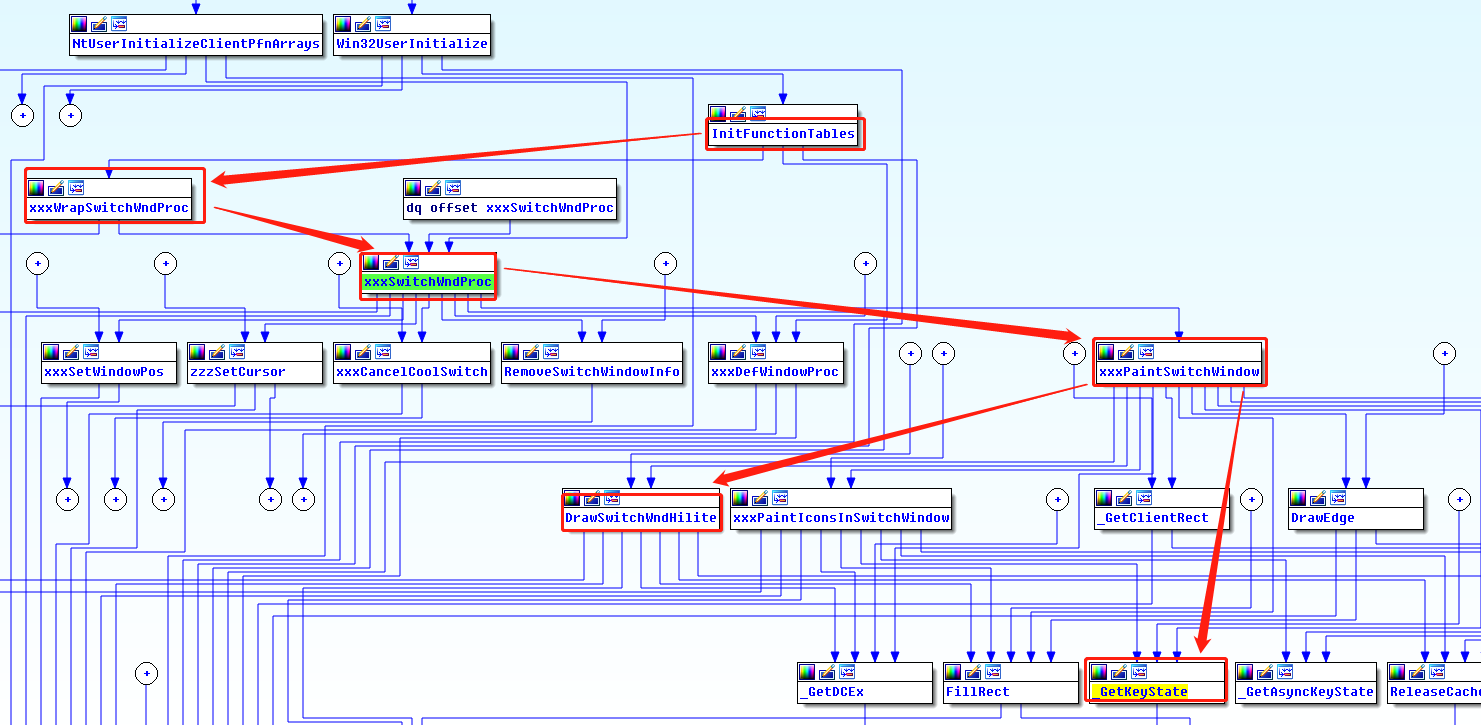
（漏洞成因分析；补丁对比分析；函数调用关系）

以下分析基于windows server 2008 R2环境进行，win32k.sys hash:CBEF2EB83438ED9FC39411CC8378B0E7。

该漏洞根因是位于win32k.sys中的InitFunctionTables()中的\*(gpsi+0x154)未进行初始化定义，这样使得我们有机会调用未公开（即非导出函数）系统函数win32K!NtUserMessageCall()并通过设置其Msg参数及其他来相关条件（哪些条件见下文分析）绕过相关函数中的一些判断分支，最终到达目标函数xxxPaintSwitchWindow，而该函数中的某些内存写操作语句的地址指针可被我们控制（使用SetWindowLongPtr函数），进而造成任意内存读写操作。

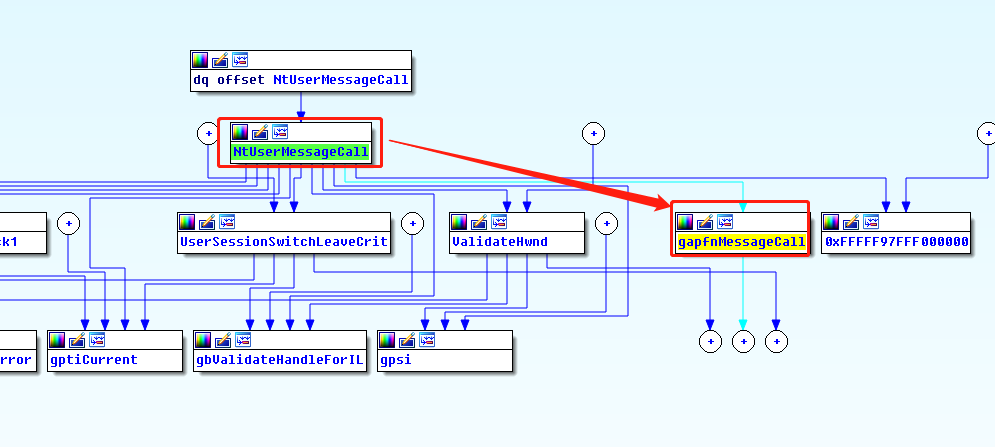
## ·首先我们看下xxxPaintSwitchWindow的函数调用关系：

既然最终漏洞利用落脚在xxxPaintSwitchWindow，首先让我们看下xxxPaintSwitchWindow的被调函数关系：



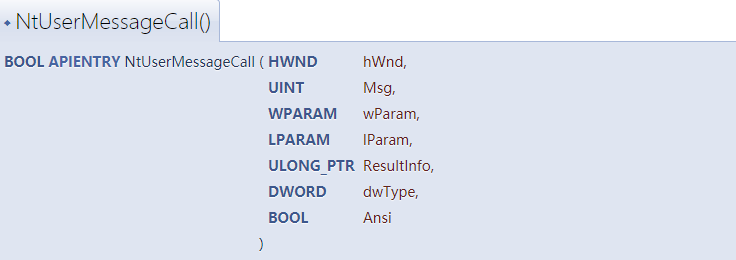
可以看到xxxWrapSwitchWndProc()函数可作为最终达到xxxPaintSwitchWindow的入口。

NtUserMessageCall函数调用：



## ·然后我们顺着win32K!NtUserMessageCall()往下看：

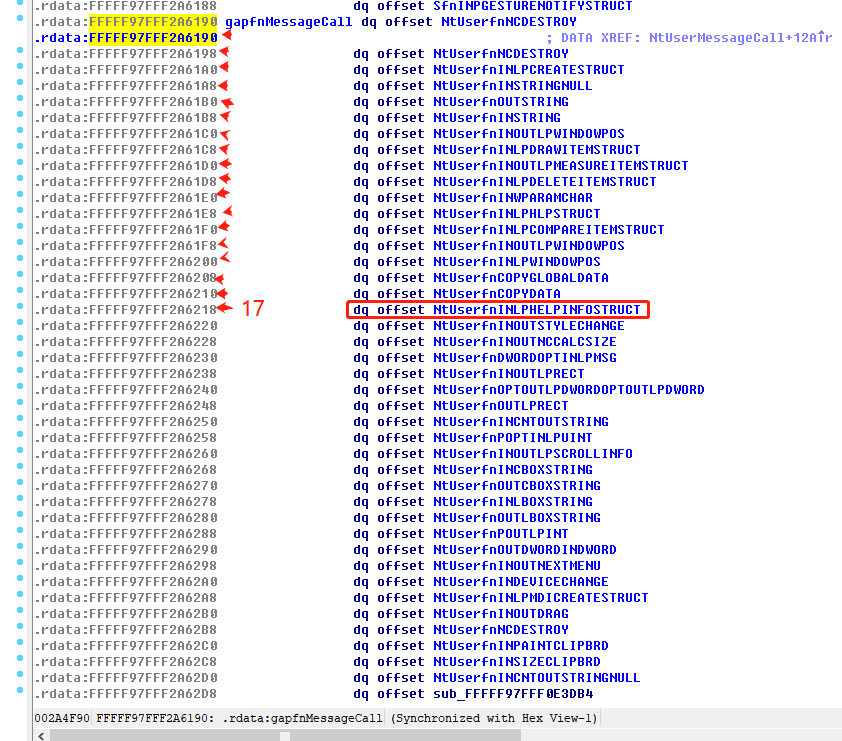
win32K!NtUserMessageCall()虽然是未公开函数，我们可以reactos.org上找到该函数的相关信息以供参考（尽管该站点上的某些函数不一定跟windows系统的完全一致，但有参考总比没有好吧）：



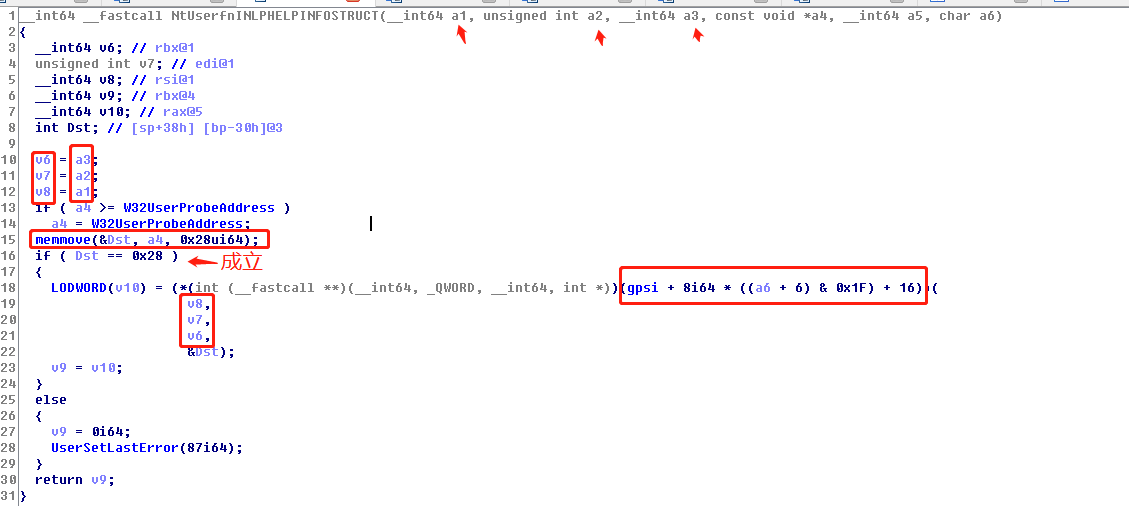
再回过头来看win32k.sys里的伪代码：



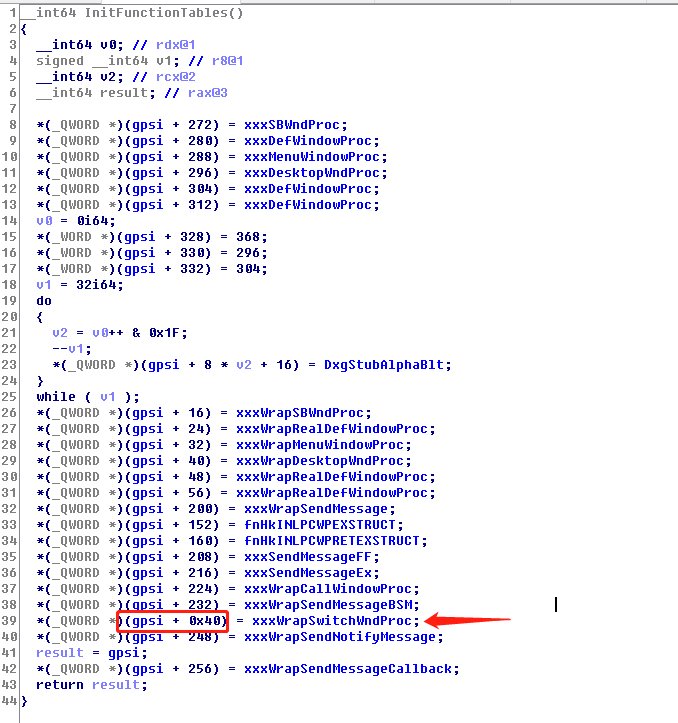
从中我们可以看到NtUserMessageCall()的第二个参数Msg可控制gapfnMessageCall[]中具体执行的函数，当Msg==1时，(v9-0x68001000000i64+0x2A6390)&0x3F)== 0x11,即十进制17，对应的函数为NtUserfnINLPHELPINFOSTRUCT()：



我们从NtUserMessageCall()中可以看到而NtUserfnINLPHELPINFOSTRUCT()参数是从NtUserMessageCall()的参数传过来的(注意：我们可以看到部分参数没有显示在被调函数的参数列表里，这可能是IDA的伪代码转化没正确转化（x64参数传递部分保存在栈区），可以在反汇编代码里进行确认)，而当我们观察NtUserfnINLPHELPINFOSTRUCT()发现它会执行到(gpsi+8i64\*((a6+6)&0x1F)+16))：

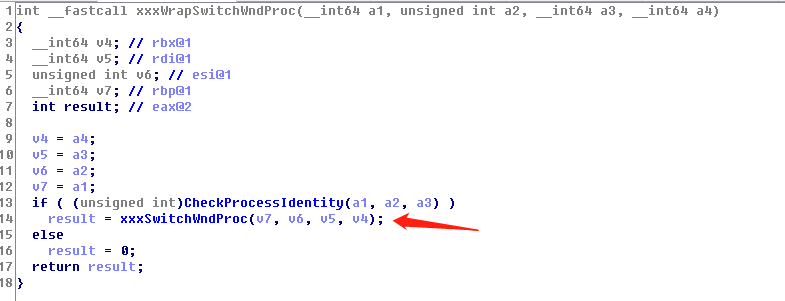


而这个(gpsi+8i64\*((a6+6)&0x1F)+16))中的a6参数是从NtUserMessageCall()的第6个参数dwType来的，即可控的。我们看下gpsi函数数组（在InitFunctionTables()中）：

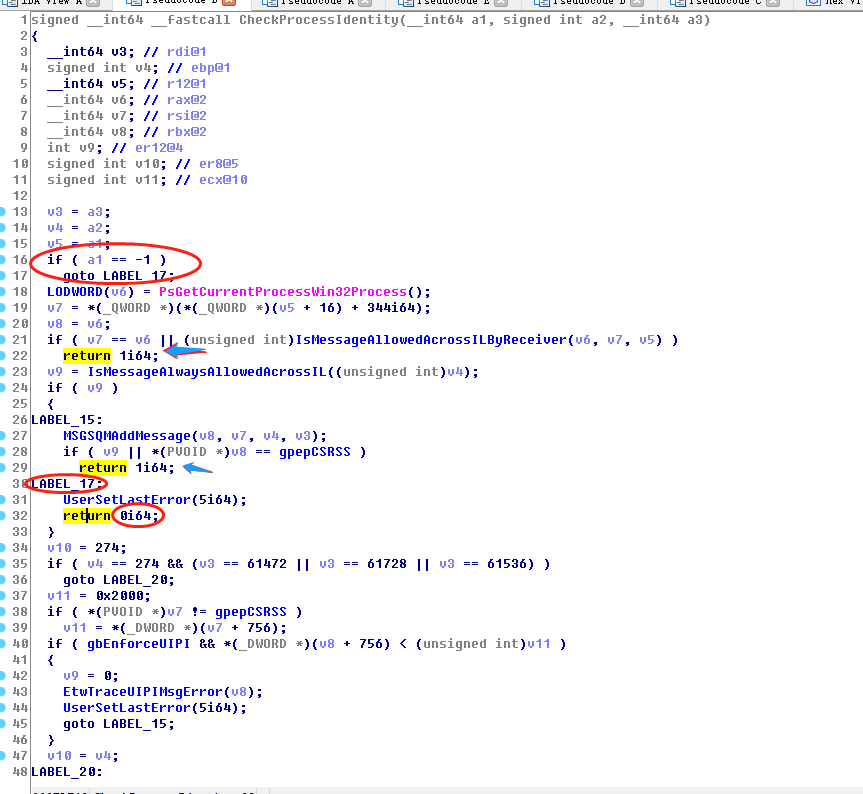


这里我们看到了xxxWrapSwitchWndProc()，即漏洞利用函数调用链的开头。那么要让程序运行到该处，我们就需要让8i64 \* ((a6 + 6) & 0x1F) + 16==0x40，a6为从NtUserMessageCall()的第6个参数dwType，是可控的，计算得：a6==0或a6==0xE6，即NtUserMessageCall()的dwType要设置为0或0xE6。

现在我们控制程序运行到xxxWrapSwitchWndProc()了，我们看下这个函数：



我们看到要想执行ROP的下一个函数需要if条件为真，那么就需要我们进去CheckProcessIdentity()看下：



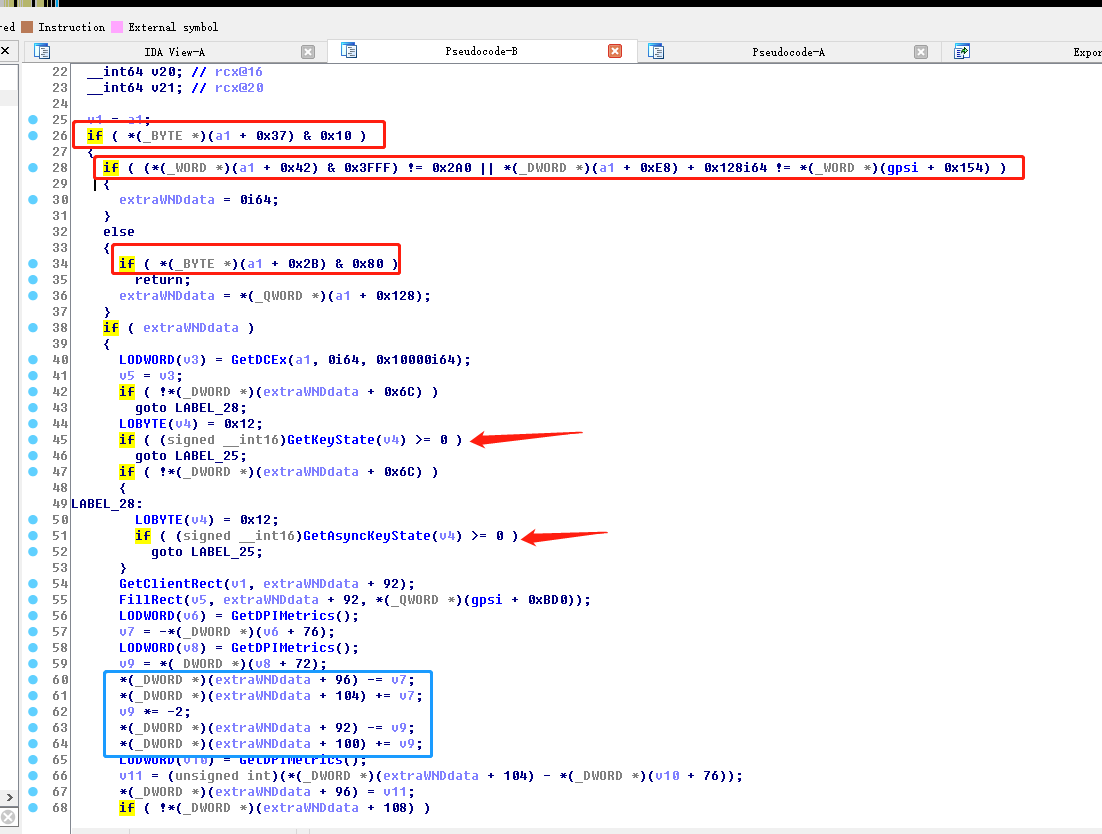
可以看到我们只要不让a1，即NtUserMessageCall()的第一个参数等于-1，事实上我们在使用NtUserMessageCall()时第一个参数为窗口实例句柄，是不会为-1的。所以CheckProcessIdentity()就会恒返回1，即xxxWrapSwitchWndProc()会执行到xxxSwitchWndProc().

OK，现在我们该进去xxxSwitchWndProc()看看了，看到如何能进一步去执行接下来的ROP链函数（接下来剩余的部分也是最难绕过的）：



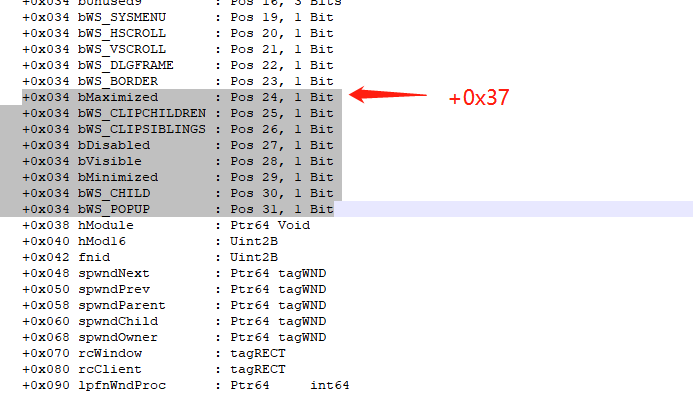
如上，当我们通过设置NtUserMessageCall()的第二个参数Msg==1来使得v6==1时，即可是程序顺利进入到switch语句；紧接着，我们再次调用NtUserMessageCall()并设置Msg==0x14（或者0x3A），就可以顺利进入我们的目标函数xxxPaintSwitchWindow()中了。

那好，我们接下来进入xxxPaintSwitchWindow()中看看：

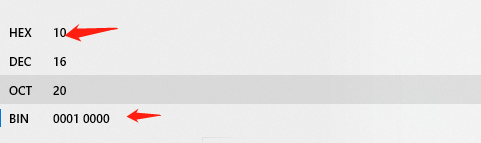


从中我们可以看到第36行函数参数a1（a1即窗口对象）来可控制extraWNDdata的值，通过控制第60、61、63、64行可以实现任意地址写操作，但前提是我们要先跳过红框内的三个检查。a1是窗口对象句柄，其数据结构即tagWND.

先看第一个判断： \*(\_BYTE \*)(a1 + 0x37) & 0x10，我们先看下win32k!tagWND+0x37处：



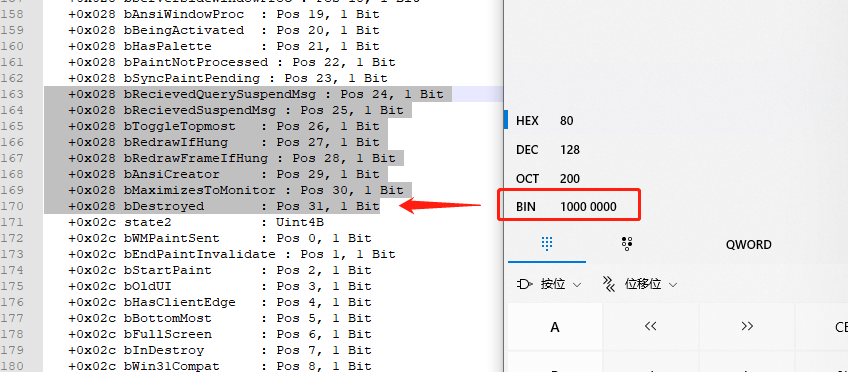
而0x10为：



所以要想让该if语句为真，则需要第5个bit位为1，即bVisible为真，而这个可以通过设置CreateWindowEx第4个参数为WS\_VISIBLE来实现。所以这个判断可以pass.

咱看第二个判断：我们需要让|“或”的两侧都为假。首先\*(\_WORD \*)(a1 + 0x42)已经在我们bypass xxxSwitchWndProc()函数的判断语句时使其等于0x2A0了，0x3FFF0&x2A0依旧等于0x2A0；再看\*(\_DWORD \*)(a1 + 0xE8) + 0x128i64 != \*(\_WORD \*)(gpsi + 0x154)，由于(gpsi + 0x154)未被赋值，所以该语句恒为假。这个判断也pass了。

最后看第三个判断：if ( \*(\_BYTE \*)(a1 + 0x2B) & 0x80 )，该语句为检查bDestroyed是否为真，即窗口是否已被销毁。窗口还没被销毁，当然可以pass该判断。如下是对应的tagWND结构：



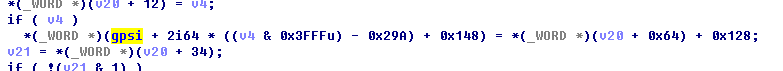
而extraWNDdata = \*(\_QWORD \*)(a1 + 0x128)我们可以通过函数SetWindowLongPtr来控制。

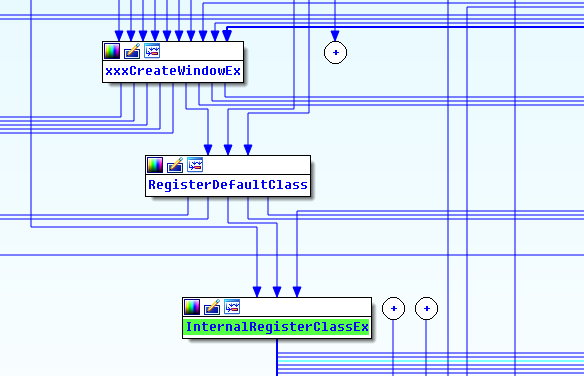
从PaintSwitchWindow中可以看到45、51行有对键盘状态检测的判断你语句，其中GetKeyState()检取指定虚拟键的状态。该状态指定此键是UP状态，DOWN状态，还是被触发的,结果> 0 表示没按下，结果< 0表示被按下。参数0x12为虚拟键码，表示”ALT”按键。同样地，GetAsyncKeyState也是一个用来判断[函数调用](https://baike.baidu.com/item/%E5%87%BD%E6%95%B0%E8%B0%83%E7%94%A8/4127405" \t "https://baike.baidu.com/item/_blank)时指定虚拟键的状态，用于确定用户当前是否按下了键盘上的一个键的函数。因此我们需要模拟ALT键按下来pass它。

OK，现在程序可以顺利到达我们目的地了。

此外需要说明的情况：

1. 我们通过CreateWindowEx()设置（第二个参数为0x8003）切换窗口时，会引起InternalRegisterClassEx()函数执行，而其中该语句会使得\*(gpsi+0x154) 被赋值为 0x130，这就造成了如果我们再次执行exp就无法跳过上边xxxSwitchWndProc()函数的if语句了，因此改exp我们只有一次运行机会，除非系统重启：





2.SetWindowLongPtr()必须在第一次调用NtUserMessageCall() 之后且在调用CreateWindowEx()创建切换窗口之前调用。这是因为,首先第一次调用NtUserMessageCall()时需要extraWNDdata为0来跳过相关判断，所以在其之前我们不能调用SetWindowLongPtr()来设置extraWNDdata。其次我们深入SetWindowLongPtr()内部可以看到，因为在CreateWindowEx()创建切换窗口之后\*(gpsi+0x154) 被赋值为 0x130，所以SetWindowLongPtr()无法正确执行下图中我们希望它执行到的分支，所以要在CreateWindowEx()创建切换窗口之前执行SetWindowLongPtr()。



# POC ro EXP分析

（POC or EXP调试分析；漏洞利用思路分析；POC or EXP效果展示）

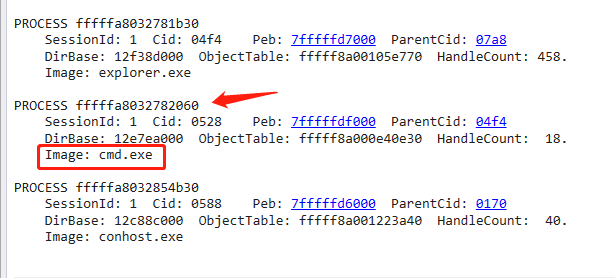
调试EXP路径：https://github.com/unamer/CVE-2019-1458

首先我们用windbg以内核模式进行双机调试，并进行符号路径设置及相关符号的下载。

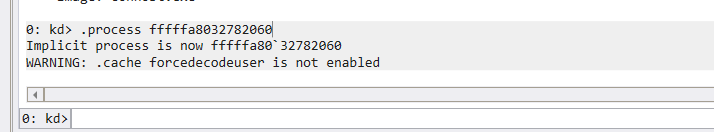
调试环境是windows server 2008 R2环境，使用cmd.exe加载exp:  
cmd.exe c:\users\guest\desktop\6source.exe whoami

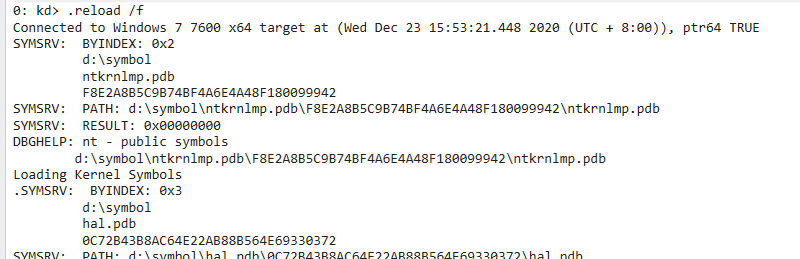


首先我们使用命令!process 0 0找到cmd.exe进程：

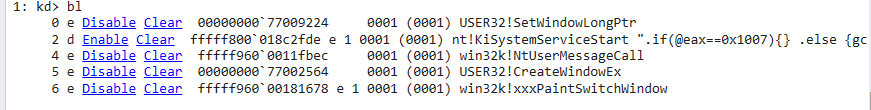


然后使用命令.process fffffa8032782060切换至沉浸式cmd进程

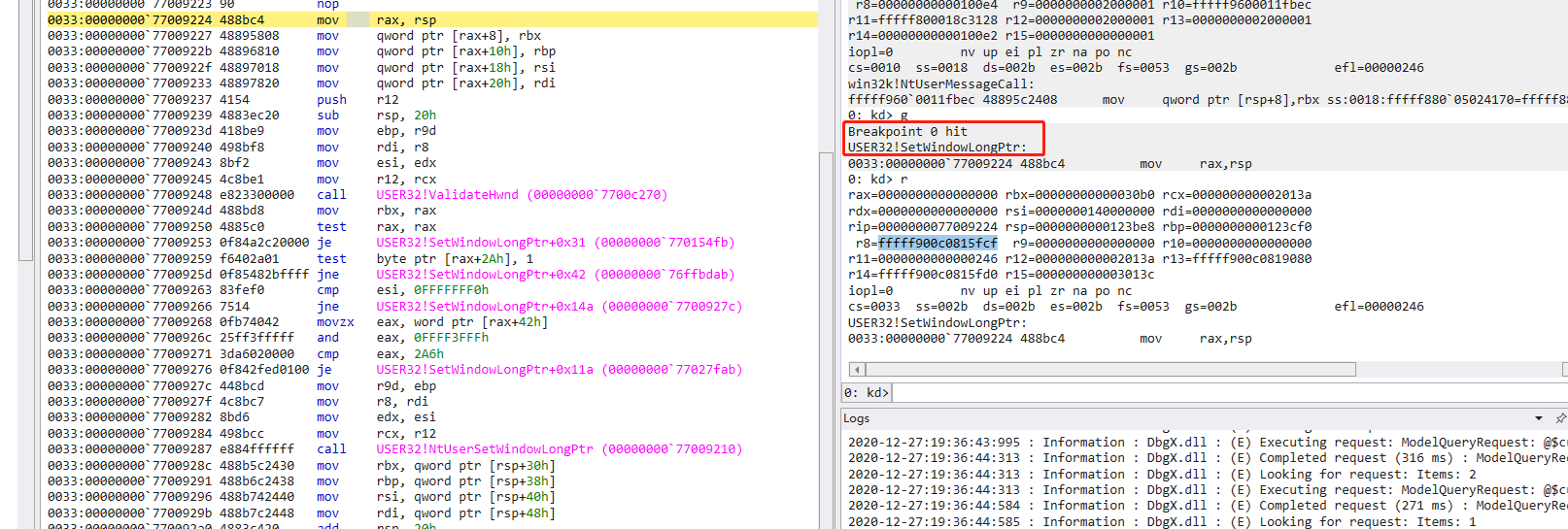
加载符号：



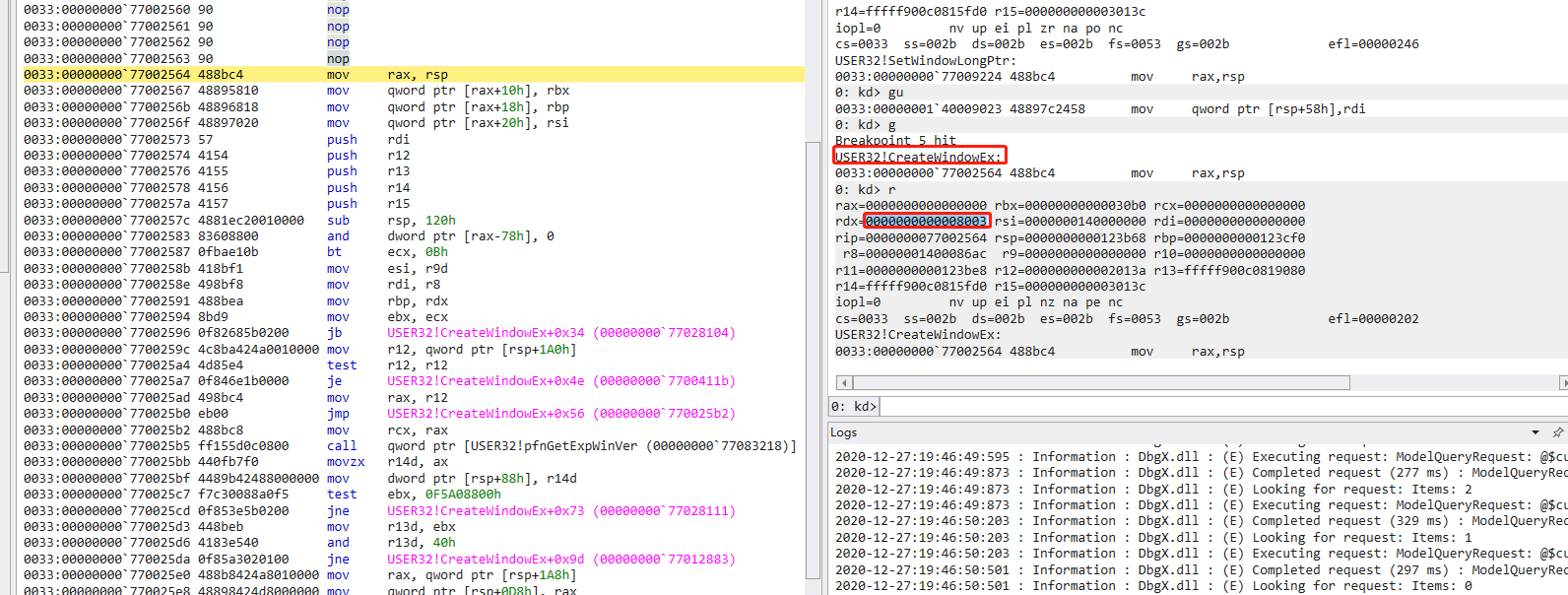
然后我们就可以设置一些用户态断点了（这里我设置了条件断点，更快地定位到目的地址），使windbg在6source.exe中关键点断下：



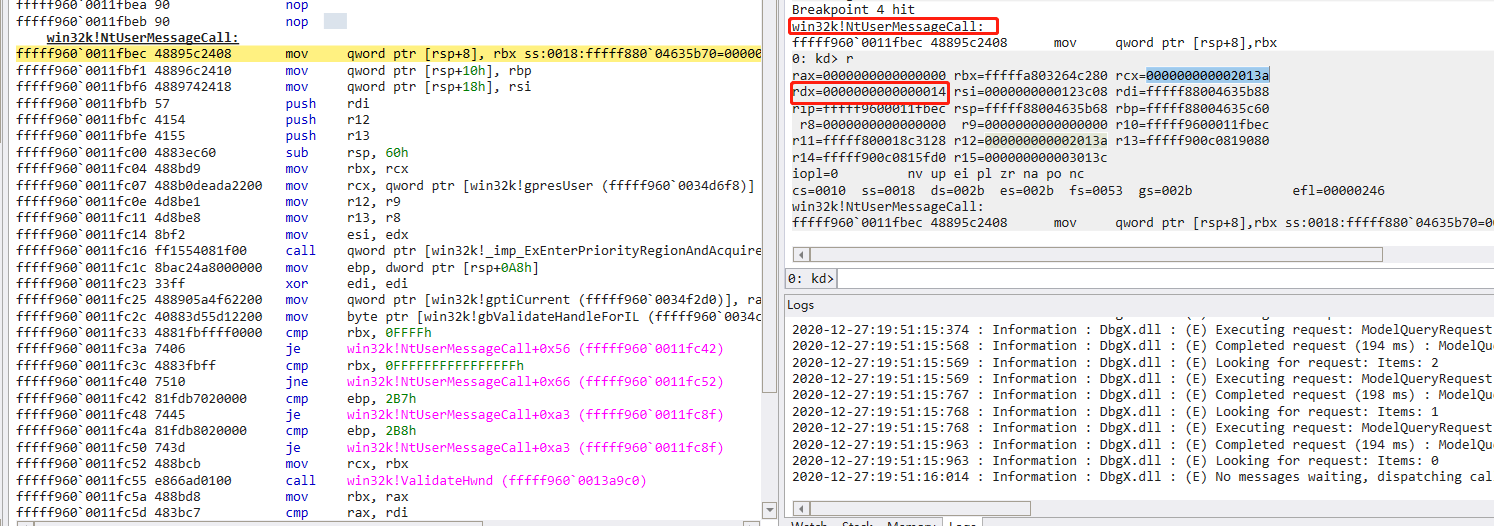
执行命令g运行后，我们在windows server 2008 R2中按回车键运行程序，进而触发断点，EXP编写跟我们的分析思路一致，前边调试过程我们略过。我们进入SetWindowLongPtr()并断下，可以看到窗口对象附加数据extraWNDdata被设置过程：



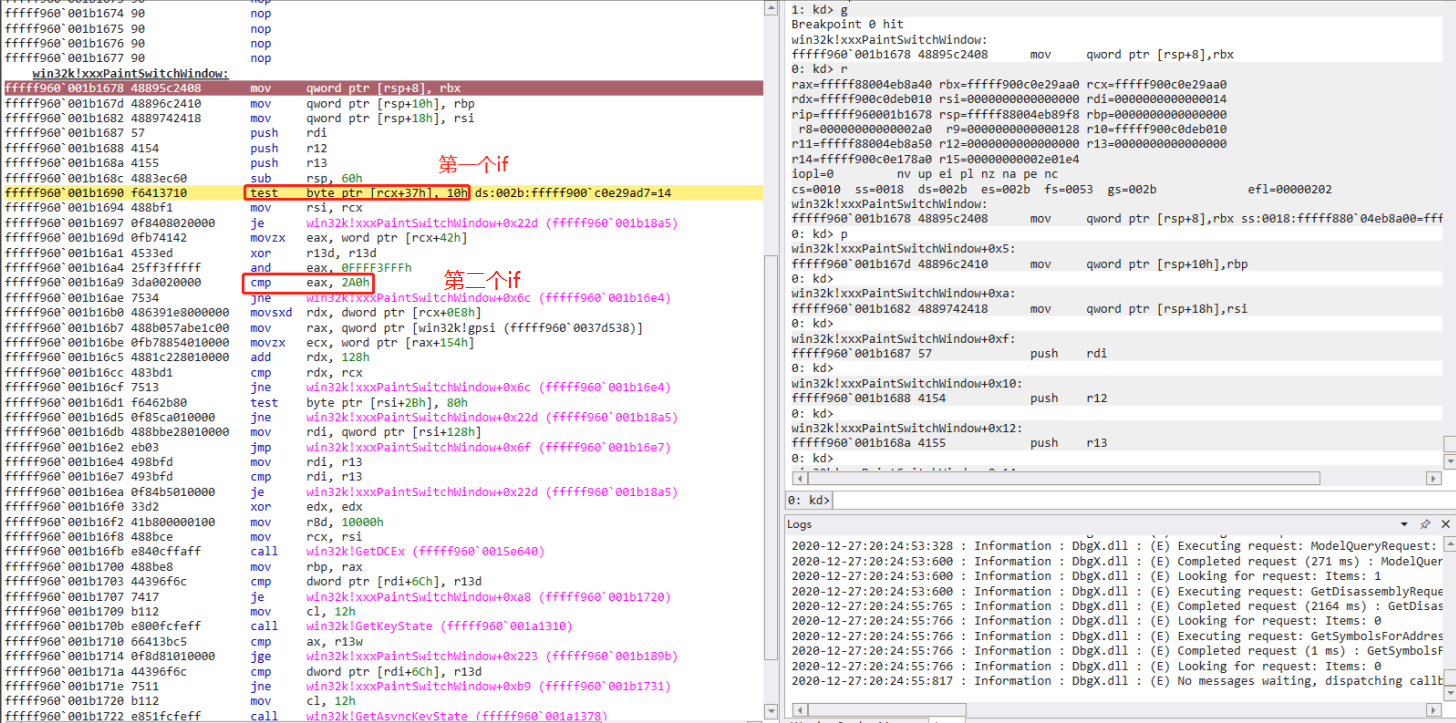
其后看到切换窗口创建：



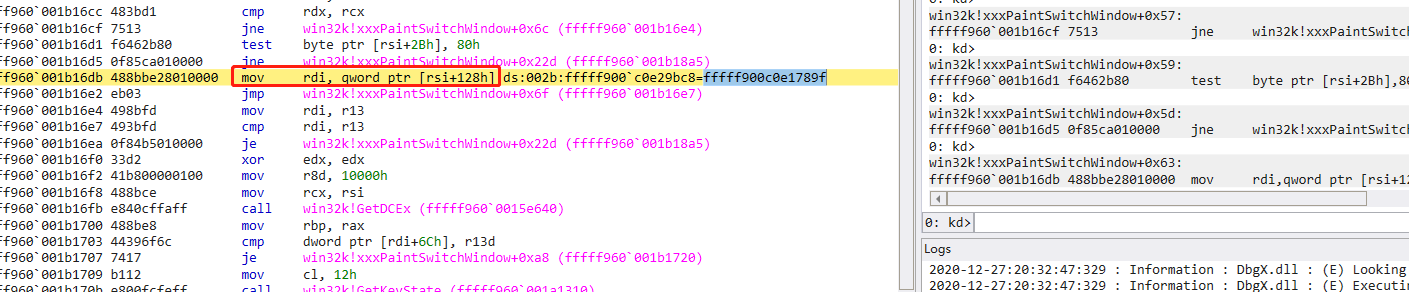
第二次调用NtUserMessageCall()来触发对extraWNDdata的引用：

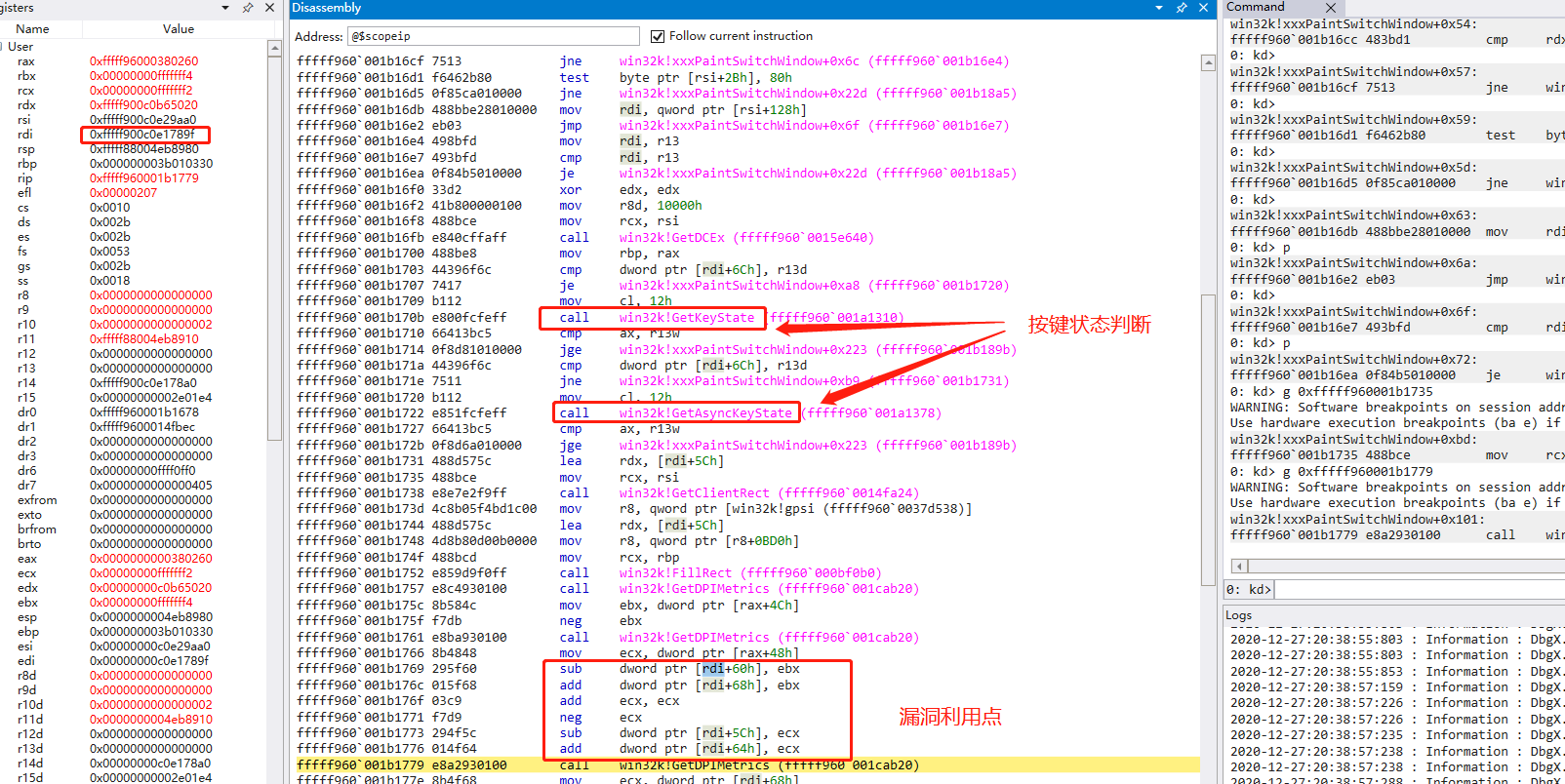


我们设置跟进去，在xxxPaintSwitchWindow()停下：



看到extraWNDdata被引用：





再往后的Token替换过程是已有较为成熟的手段，这里不在赘述。

# 写在最后

（总结分析；相关知识点；IOC；相关参考引用）

漏洞分析和漏洞利用还是不同的层阶的，个人认为漏洞分析更容易一些，而漏洞利用EXP编写，需要掌握一些公开和非公开函数调用及一定开发能力，尤其通用、健壮的EXP编写是更高层阶一些的，是在漏洞分析的基础上再加上漏洞利用与EXP开发能力，当然这里边也已经有很多比较成熟的技巧和方法可供参考。兴趣是最好的老师，加之不懈努力，You will get there.

参考：

EXP：https://github.com/unamer/CVE-2019-1458

POC：https://github.com/piotrflorczyk/cve-2019-1458\_POC

Microsoft:https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2019-1458

TagWND任意读写：http://www.sinkland.cn/?p=222