登录

Microsoft Windows远程内核崩溃漏洞分析

mss**** / 2018-06-22 16:14:13 / 浏览数 5663 技术文章 技术文章 顶(0) 踩(0)

原文：https://www.fortinet.com/blog/threat-research/microsoft-windows-remote-kernel-crash-vulnerability.html?utm_source=social

概述

在2018年1月底，我们在Microsoft
Windows系统中发现了一个远程内核崩溃漏洞，并根据认真负责的披露流程向微软报告了这个安全问题。6月12日，微软发布了一份包含该漏洞修复程序的公告，并将该漏

实际上，这个内核崩溃漏洞位于Microsoft
Windows代码完整性内核模块"ci.dll"中。并且，目前所有流行的Windows版本都受该漏洞影响，这些版本包括Windows 10、Windows 7、Windows
8.1、Windows Server 2008、Windows Server 2012和Windows Server 2016。

这个漏洞可以通过从网站或SMB共享远程下载精心制作的.dll或.lib文件来触发。使用IE或Edge浏览器下载并保存漏洞触发文件时，会执行一个让Windows内核指针取消引用
Bugcheck（内核崩溃）。对于Windows 10来说，系统重新启动后，用户登录时又会发生内核崩溃，从而进入一个死循环。

在本文中，我们将同读者一道对这个漏洞进行详细的分析。

漏洞分析

为了重现该远程内核崩溃漏洞，可以在Windows
10上打开IE或Edge浏览器，并在地址栏输入http://192.168.0.111/poc.dll（可以是任何托管PoC文件的URL），然后在弹出的窗口中选择"保存"。当保存文件poc.dll时，就
10蓝屏死现象（内核崩溃）。对于Windows 10中来说，如果内核发生崩溃，即使重新启动，内核还会继续崩溃，从而导致Windows
10机器无法正常工作。对于用户来说，如果遇到这种情况，那只好重装系统了。

以下是发生崩溃时的调用堆栈情况。

```
83157e5c 81d8e8ab 00000003 789e5eef 00000065 nt!RtlpBreakWithStatusInstruction
83157eb0 81d8e2fa 8337e340 831582cc 83158340 nt!KiBugCheckDebugBreak+0x1f
831582a0 81d1508a 00000050 835e1000 00000000 nt!KeBugCheck2+0x739
831582c4 81d14fc1 00000050 835e1000 00000000 nt!KiBugCheck2+0xc6
831582e4 81cbd03c 00000050 835e1000 00000000 nt!KeBugCheckEx+0x19
83158340 81c5b212 83158474 835e1000 831583b8 nt!MiSystemFault+0xc72
831583d8 81d2d9a4 00000000 835e1000 00000000 nt!MmAccessFault+0x100
831583d8 85d01391 00000000 835e1000 00000000 nt!KiTrap0E+0x288
831585bc 85d0375c fffff03a 831585e4 835e00b0 CI!SymCryptSha1AppendBlocks+0x331
831585f0 85d00fa7 835e0150 fffffe7a 83158610 CI!SymCryptHashAppendInternal+0xd0
83158600 85d1c7d3 fffffeb6 85b37248 83158688 CI!SymCryptSha1Append+0x15
83158610 85d15743 fffffeb6 908c86c4 00008004 CI!HashpHashBytes+0x55
83158688 85d169be 835e0000 85b37248 00000210 CI!CipImageGetImageHash+0x12b
831586b4 85d15107 85b37008 00008004 835e0000 CI!CipCalculateImageHash+0x32
8315872c 85d14f1c 85b37008 a3566880 838334c0 CI!CipValidateFileHash+0x147
83158768 85d13d97 a3566880 838334c0 835e0000 CI!CipValidateImageHash+0x48
83158830 81eca6f5 a3566880 835e0000 00001000 CI!CiValidateImageHeader+0x399
83158878 81eca923 00001000 838334c0 b000f600 nt!SeValidateImageHeader+0x47
83158958 81ecdc68 00000001 b000f600 00000000 nt!MiValidateSectionCreate+0x1ed
83158a20 81e7ec37 00000000 00000000 b000f600 nt!MiCreateNewSection+0x3c4
83158aa8 81e7e501 00000000 11000000 b000f600 nt!MiCreateImageOrDataSection+0x201
83158b44 81e7da17 00000000 83158bbc 00000002 nt!MiCreateSection+0x6d
83158b84 81e7d8d2 83158c0c 00000005 00000000 nt!MmCreateSection+0x6f
83158bf0 81d279ae 0403eba0 00000005 00000000 nt!NtCreateSection+0x104
83158bf0 77604350 0403eba0 00000005 00000000 nt!KiSystemServicePostCall
0403eb28 77602f8a 746a6cc7 0403eba0 00000005 ntdll!KiFastSystemCallRet
0403eb2c 746a6cc7 0403eba0 00000005 00000000 ntdll!NtCreateSection+0xa
0403ecc4 746a965f 00000000 00000022 0403ecec KERNELBASE!BasepLoadLibraryAsDataFileInternal+0x257
0403ed00 746a394b 07d0c940 00000000 00000022 KERNELBASE!LoadLibraryExW+0xcf
0403ed50 56e869e4 00000001 07d0c940 0403ed70 KERNELBASE!GetFileVersionInfoSizeExW+0x2b
0403ed78 56e88088 07dd46f8 0403edd8 07dd4690 IEFRAME!CDownloadSecurity::_DetermineFileVersion+0x26
0403eda0 56e8753b 07dd4690 07dd4690 0403edc4 IEFRAME!CDownloadSecurity::_StartPreAppRepCheck+0x79
0403edb0 56e86f03 06827320 56e86030 07c75858 IEFRAME!CDownloadSecurity::_NextSecurityState+0x7a
0403edc4 56e86040 0403ede4 56f8f8a4 07dd469c IEFRAME!CDownloadSecurity::_FinishMalwareCheck+0xeb
0403edcc 56f8f8a4 07dd469c 0403edf4 54d01884 IEFRAME!CDownloadSecurity::OnURSResultAvailable+0x10
0403ede4 56f8f759 04d73d64 80072ee2 0403ee30 IEFRAME!CURSState::ProcessResults+0x88
0403ee04 56f8f661 06827328 00000000 0290dc44 IEFRAME!CURSState::Invoke+0x79
0403ee48 772e6d8b 07c7f738 00000000 0290dc44 IEFRAME!CURSProcessor::Invoke+0x101
0403ee88 772ff42b 0290dc44 00000400 000c0001 OLEAUT32!IDispatch_Invoke_Stub+0x66
0403eebc 756e6905 0403ef94 8a91250b 06826718 OLEAUT32!IDispatch_RemoteInvoke_Thunk+0x5b
0403f36c 752732a9 06826718 0682c070 0290ebb4 RPCRT4!NdrStubCall2+0xfb5
0403f3b8 772e0253 06826718 0290ebb4 0682c070 combase!CStdStubBuffer_Invoke+0x99 [onecore\com\combase\ndr\ndrole\stub.cxx @ 1530]
0403f3dc 752ca958 0682b1d0 0290ebb4 0682c070 OLEAUT32!CStubWrapper::Invoke+0x43
0403f430 752ca4a1 0682b1d0 0290ebb4 0682c070 combase!ObjectMethodExceptionHandlingAction<<lambda_1ba7c1521bf8e7d0ebd8f0b3c0295667> >+0xa8 [
0403f554 752c8fb0 0682c070 0682b1d0 0403f750 combase!DefaultStubInvoke+0x221 [onecore\com\combase\dcomrem\channelb.cxx @ 1891]
0403f6d4 752cb32b 0290ebb4 0682b1d0 0682c070 combase!ServerCall::ContextInvoke+0x440 [onecore\com\combase\dcomrem\ctxchnl.cxx @ 1541]
0403f774 752cc3c9 0682b1d0 07c7f738 0290dc40 combase!AppInvoke+0x8bb [onecore\com\combase\dcomrem\channelb.cxx @ 1604]
0403f904 752ff7d5 0403f977 07d0f818 0290eb60 combase!ComInvokeWithLockAndIPID+0x599 [onecore\com\combase\dcomrem\channelb.cxx @ 2722]
0403f960 752ff42b 00010350 00000400 00000001 combase!ComInvoke+0x1c5 [onecore\com\combase\dcomrem\channelb.cxx @ 2242]
0403f9a0 75896bb7 00010350 00000400 0000babe combase!ThreadWndProc+0x21b [onecore\com\combase\dcomrem\chancont.cxx @ 741]
```

图1. 发生崩溃时的调用堆栈

从上面的调用堆栈输出可以看到，内核崩溃出现在调用函数“KERNELBASE！GetFileVersionInfoSizeExW”的过程中，然后它会调用“KERNELBASE！LoadLibraryExW”函数

当IE/Edge下载.dll或.lib文件并将其保存到磁盘上时，将调用函数“KERNELBASE！GetFileVersionInfoSizeExW”来获取.dll/.lib文件的版本信息。为了获取.dll/.lib文件的版本
0x22是“LOAD_LIBRARY_AS_DATAFILE（0x00000002）”和“LOAD_LIBRARY_AS_IMAGE_RESOURCE（0x00000020）”的组合。因此，IE/Edge会将下载的.dll/.lib文件作
10发生内核崩溃后，即使重新启动也无法恢复系统，这是因为用户登录Windows时，会扫描IE/Edge临时目录中的.dll/.lib文件。

函数LoadLibraryExW会加载我们精心构造的PoC文件，即poc.dll。当它处理SizeOfHeaders时，得到的大小为0x06，这是我们精心构造的一个值，实际上正确的大小是0x2

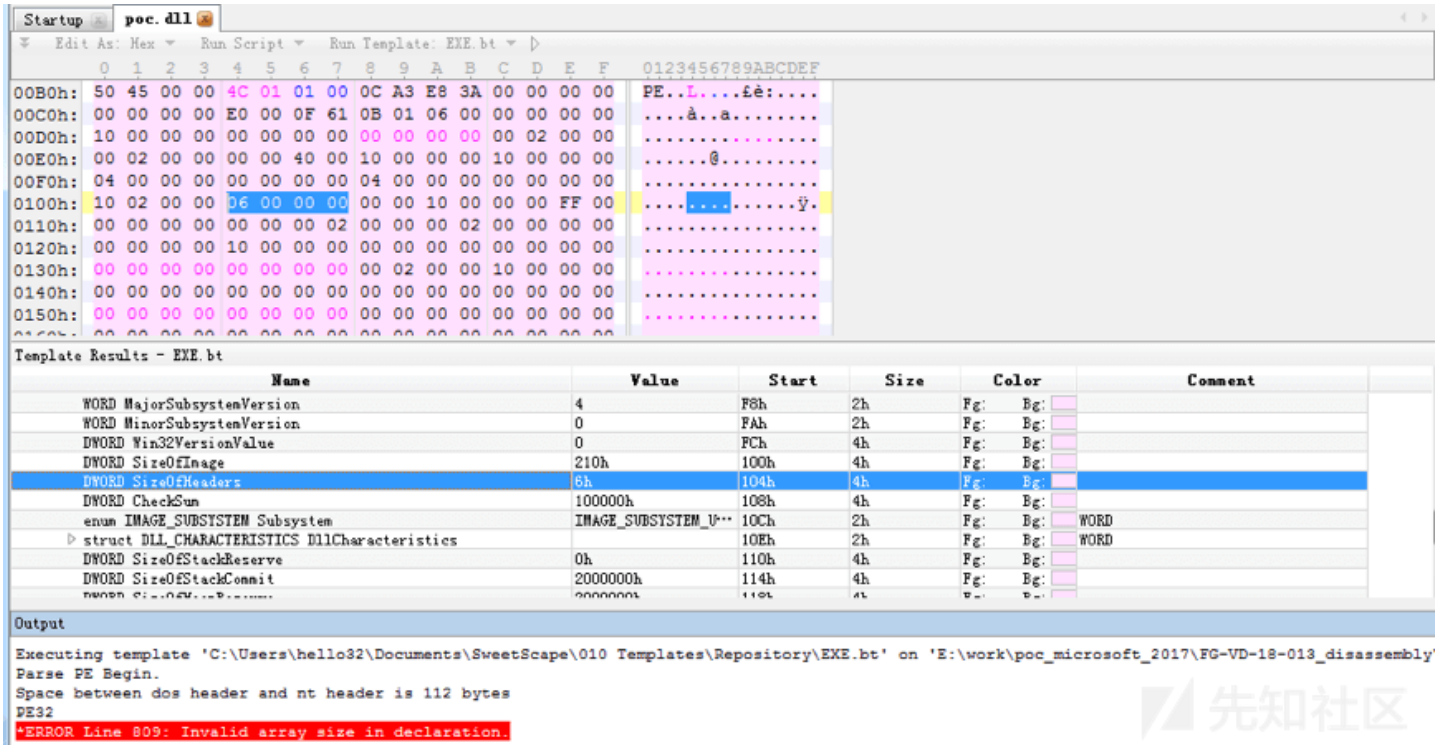

图2.包含精心制作的SizeOfHeaders的poc.dll

通过逆向工程和跟踪，我们可以看到，对函数_CipImageGetImageHash的调用导致sha1块大小整数溢出。

```
PAGE:85D15618 _CipImageGetImageHash@36 proc near      ; CODE XREF:
......
PAGE:85D1571F                 mov     edx, edi
PAGE:85D15721                 mov     ecx, [ebp+arg_4]
PAGE:85D15724                 call    _HashpHashBytes@12 ; HashpHashBytes(x,x,x)
PAGE:85D15729                 lea     edx, [esi+0A0h]
PAGE:85D1572F
PAGE:85D1572F loc_85D1572F:                           ; CODE XREF: CipImageGetImageHash(x,x,x,x,x,x,x,x,x)+CF↑j
PAGE:85D1572F                 mov     edi, [ebp+arg_10]
PAGE:85D15732                 mov     eax, [edi+54h]  ; ----->  here [edi+54h] is obtained from poc.dll at offset 0x104, its v
PAGE:85D15735                 sub     eax, edx        ; ----->  here edx=83560150
PAGE:85D15737                 add     eax, [ebp+BaseAddress]  ----> here [ebp+BaseAddress]=83560000
PAGE:85D1573A                 push    eax             ;  ---------> So, after the above calculation, eax occurs integer subtra
PAGE:85D1573B                 mov     ecx, [ebp+arg_4]
PAGE:85D1573E                 call    _HashpHashBytes@12 ------> the function call chain finally results in a kernel crash
PAGE:85D15743                 mov     esi, [edi+54h]  ;
PAGE:85D15746                 mov     [ebp+var_30], esi
```

在以下函数中，执行的边界检查不够充分:

```
.text:85D0368C @SymCryptHashAppendInternal@16 proc near
.text:85D0368C                                         ; CODE XREF: SymCryptSha1Append(x,x,x)+10↑p
.text:85D0368C                                         ; SymCryptMd5Append(x,x,x)+10↑p
.text:85D0368C
.text:85D0368C var_18          = dword ptr -18h
.text:85D0368C var_14          = dword ptr -14h
.text:85D0368C var_10          = dword ptr -10h
.text:85D0368C var_C           = dword ptr -0Ch
.text:85D0368C var_8           = dword ptr -8
.text:85D0368C var_4           = dword ptr -4
.text:85D0368C Src             = dword ptr  8
.text:85D0368C MaxCount        = dword ptr  0Ch
.text:85D0368C
.text:85D0368C                 mov     edi, edi
.text:85D0368E                 push    ebp
.text:85D0368F                 mov     ebp, esp     .
......
85D0372D                mov     ecx, [ebp+var_8]
.text:85D03730                 mov     edx, [ebp+var_18]
.text:85D03733                 jmp     short loc_85D0373B
```

```
.text:85D03735 ; ---------------------------------------------------------------------------
.text:85D03735
.text:85D03735 loc_85D03735:                           ; CODE XREF: SymCryptHashAppendInternal(x,x,x,x)+46↑j
.text:85D03735                                         ; SymCryptHashAppendInternal(x,x,x,x)+52↑j
.text:85D03735                 mov     ecx, [ebp+Src]
.text:85D03738                 mov     [ebp+var_8], ecx
.text:85D0373B
.text:85D0373B loc_85D0373B:                           ; CODE XREF: SymCryptHashAppendInternal(x,x,x,x)+A7↑j
.text:85D0373B                 cmp     esi, [edx+18h]  ; ----> here [edx+18h] equals 40h, esi equals fffffe7a, due to unsigned
.text:85D0373E                 jb      short loc_85D03769
.text:85D03740                 mov     edi, [edx+1Ch]
.text:85D03743                 lea     eax, [ebp+var_C]
.text:85D03746                 push    eax
.text:85D03747                 push    esi
.text:85D03748                 mov     esi, [edx+0Ch]
.text:85D0374B                 add     edi, ebx
.text:85D0374D                 mov     ecx, esi
.text:85D0374F                 call    ds:___guard_check_icall_fptr ; _guard_check_icall_nop(x)
.text:85D03755                 mov     edx, [ebp+var_8]
.text:85D03758                 mov     ecx, edi
.text:85D0375A                 call    esi
```

使用溢出的sha1块大小后，最终会调用以下函数：

```
.text:85D01060 @SymCryptSha1AppendBlocks@16 proc near  ; CODE XREF: SymCryptSha1Result(x,x)+40↑p
......
.text:85D010A4                 mov     eax, [ebp+arg_0] -----> here eax gets the overflowed sha1 block size= 0xfffffe7a
.text:85D010A7                 mov     [esp+0D0h+var_B4], edi
.text:85D010AB                 mov     [esp+0D0h+var_C4], ecx
.text:85D010AF                 cmp     eax, 40h
.text:85D010B2                 jb      loc_85D02507
.text:85D010B8                 mov     [esp+0D0h+var_58], ecx
.text:85D010BC                 mov     ecx, [esp+0D0h+var_C0]
.text:85D010C0                 mov     [esp+0D0h+var_54], ecx
.text:85D010C4                 lea     ecx, [edx+8]    ;
.text:85D010C7                 shr     eax, 6    -------> the overflowed block size is used as the following loop function cou
.text:85D010CA                 mov     [esp+0D0h+var_60], esi
.text:85D010CE                 mov     [esp+0D0h+var_5C], edi
.text:85D010D2                 mov     [esp+0D0h+var_68], ecx ;
.text:85D010D6                 mov     [esp+0D0h+var_50], eax -----> here is the loop counter
......
.text:85D01359                 ror     edx, 2
.text:85D0135C                 mov     ecx, [ecx+28h]
.text:85D0135F                 bswap   ecx
.text:85D01361                 mov     [esp+0D0h+var_6C], ecx
.text:85D01365                 mov     ecx, eax
.text:85D01367                 rol     ecx, 5
.text:85D0136A                 mov     eax, edi
.text:85D0136C                 add     ecx, [esp+0D0h+var_6C]
.text:85D01370                 xor     eax, edx
.text:85D01372                 and     eax, [esp+0D0h+var_C0]
.text:85D01376                 xor     eax, edi
.text:85D01378                 add     edi, 5A827999h
.text:85D0137E                 add     eax, ecx
.text:85D01380                 mov     ecx, [esp+0D0h+var_68]
.text:85D01384                 add     eax, esi
.text:85D01386                 mov     esi, [esp+0D0h+var_C0]
.text:85D0138A                 mov     [esp+0D0h+var_84], eax
.text:85D0138E                 ror     esi, 2
.text:85D01391                 mov     ecx, [ecx+2Ch]  ----> after a large loop call, here it results in a read access violati
.text:85D01394                 bswap   ecx
.text:85D01396                 mov     [esp+0D0h+var_9C], ecx
.......
.text:85D024DD                 mov     ecx, [esp+0D0h+var_68]
.text:85D024E1                 mov     [esp+0D0h+var_54], eax
.text:85D024E5                 add     ecx, 40h  ----> memory access pointer increases 0x40 in each loop
.text:85D024E8                 mov     [esp+0D0h+var_C0], eax
.text:85D024EC                 mov     eax, [ebp+arg_0]
.text:85D024EF                 sub     eax, 40h
```

```
.text:85D024F2                mov     [esp+0D0h+var_68], ecx
.text:85D024F6                sub     [esp+0D0h+var_50], 1  ------> here the loop counter decreases by 1, not equaling 0, to
.text:85D024FE                mov     [ebp+arg_0], eax
.text:85D02501                jnz     loc_85D010DD
.text:85D02507
```

通过上面的分析我们可以看到，远程内核崩溃的根本原因是LoadLibraryEx函数无法将精心构造的.dll/.lib文件正确解析为资源和数据文件。如果poc.dll包含精心设计的SizeO

我们精心设计的大小值，会导致系统算出错误的sha1块大小（一个负值）。由于边界检查不够充分，所以sha1计算函数将进入一个非常大的循环中，导致与内存读取访问发

解决方案

建议所有受该漏洞影响的Microsoft Windows用户升级到最新的Windows版本或应用最新的修补程序。