TALOS VULNERABILITY REPORT

TALOS-2015-0007

MICROSOFT WINDOWS CDD FONT PARSING KERNEL MEMORY CORRUPTION

SEP 8, 2015

DESCRIPTION

An exploitable kernel memory corruption vulnerability exists in Microsoft Windows. A specially crafted font file can cause the Microsoft Windows kernel to corrupt internal memory structures, leading to denial of service on the system or possible code execution and elevation of privilege.

TESTED VERSIONS

Windows 7 x64 SP1
Windows 8.1 X64

PRODUCT URLS

http://microsoft.com

DETAILS

The faulting code is located inside the CDD driver. The DrvTextOut routine acquires and locks the associated device and acts differently based on the surface type. If the type is a bitmap and the Windows DWM is on, the routine reads/writes directly to the video frame buffer and call EngTextOut then exits. Otherwise, a new background rect is generated mixing the "OpaqueRect" rectangle located in the sixth parameter and the rectangle located in the "pStringTextObj" object.

If the ClipObject describes a non-trivial clip, the "rclBounds" of the clip object is merged to the background rectangle. The font object is parsed and finally the routine decides if it should clip the background rect or not.

The final decision is based on the following check:

```
f (mix != 0 && prclOpaque == NULL && (pSurfObj->dhSurf->CBCSMask & 1 == 1)

   && (pBrushFore->iSolidColor != WHITE) && mix == 0xD0D) {

   // CDD Bitmap GDI Descriptor object
   CDDBITMAP_GDIDESC CddBmpGdiDesc = {0};
   // Clip Rects Data structure
   CLIP_RECTS_DATA clipRectData = {0};
   CddBitmapHw * bmpHw = pSurfObj->dhSurf;

   // Get the CDD Bitmap GDI Descriptor object
   bmpHw->StartGdiAccessNoUpdate(&bckRect, 1, &cddBmpGdiDesc);
   // Build here the Clip descriptor object
   clipRectData.pClipObj = pClipObj;

   clipRectData.pRect1 = &bckRect;   // The 3 rectangles are set to
   clipRectData.pRect2 = &bckRect;   // the calculated background rectangle
   clipRectData.pRect3 = &bckRect;
   // These 2 lines are VERY IMPORTANT because they describe the WHAT and
   // WHERE part of the bug:
   clipRectData.lpVideoMemory = cddBpmGdiDesc.lpVideoMemory;
   clipRectData.dwValueToWrite = 0;

   // Do here the actual clip and trigger the vulnerability
   ClipDstRects(&clipRectData, FillColorKeyCallback);   // BANG!

   // Call the actual GDI text drawing routine
   EngTextOut( ... )
   // ... other code
}
```

The ClipDstRect routine performs the actual clipping based on the complexity of the Clip object. If the complexity is DC_TRIVIAL, the CPU cache is flushed and the callback routine is called.

As we see below, the FillColorKeyCallback routine implementation fails to properly check the rectangle width and height:

```
NTSTATUS FillColorKeyCallback(CLIP_RECTS_DATA * pClipRectsData, BOOL bClip, const RECT * p
Rect) {
if (!bClip) return STATUS_SUCCESS;

DWORD dwRectWidth = 0, dwRectHeight = 0;
DWORD dwRatioX = 0, dwRatioY = 0;
LPBYTE lpMemPtr = 0;        // Target Video memory pointer
DWORD dwValToWrite = pClipRectsData-> dwValueToWrite;

// BANG BANG - Serious error here:
dwRectWidth = pRect->right - pRect->left;
dwRectHeight = pRect->bottom - pRect->top;

// Calculate ratios:
dwRatioX = pRect->left * 4;
dwRatioY = pRect->top * pClipRectsData-> dwClipStep;
lpMemPtr = (LPBYTE)pClipRectsData-> lpVideoMemory + (dwRatioY - dwRatioX);

while (dwRectHeight--) {
DWORD dwMemPtrBits = (DWORD)-((QWORD)lpMemPtr / 4) & 3;
DWORD dwNumOfBytesX = (dwRectWidth - dwMemPtrBits) & 0xFFFFFFFD;
if (dwMemPtrBits > 0) {
// Trigger the overflow here
LPBYTE lpMemPtrCopy = lpMemPtr;
do {
*((DWORD*)lpMemPtrCopy) = dwValToWrite;
lpMemPtrCopy += sizeof(DWORD);
} while (--dwMemPtrBits);
}

// ... other code ...
lpMemPtr += pClipRectsData-> dwClipStep;
}
}
```

The background rectangle dimensions are checked in the DrvTextOut routine code:

```
if (bckRect->top >= bckRect->bottom ||
bckRect->left >= bckRect->right) {
LPVOID lpAssertion = WdLogNewEntry5_WdAssertion()
WdLogEvent5_WdAssertion(lpAssertion)
}
```

As you can see, the check appears to log the bug but fails to prevent the condition that leads to kernel memory corruption.

CRASH INFORMATION

```
4: kd> !analyze -v
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

PAGE_FAULT_IN_NONPAGED_AREA (50)
Invalid system memory was referenced.  This cannot be protected by try-except,
it must be protected by a Probe.  Typically the address is just plain bad or it
is pointing at freed memory.
Arguments:
Arg1: fffff80185bea3c8, memory referenced.
Arg2: 0000000000000001, value 0 = read operation, 1 = write operation.
Arg3: fffff96000869ccd, If non-zero, the instruction address which referenced the bad memo
ry
    address.
Arg4: 0000000000000000, (reserved)


FAULTING_IP:
cdd!FillColorKeyCallback+bd
fffff960`00869ccd 0fc328          movnti  dword ptr [rax],ebp

MM_INTERNAL_CODE:  0

IMAGE_NAME:  cdd.dll

DEBUG_FLR_IMAGE_TIMESTAMP:  53186c99

MODULE_NAME: cdd

FAULTING_MODULE: fffff9600085b000 cdd

DEFAULT_BUCKET_ID:  WIN8_DRIVER_FAULT

BUGCHECK_STR:  AV

URRENT_IRQL:  0

ANALYSIS_VERSION: 6.3.9600.17237 (debuggers(dbg).140716-0327) amd64fre

TRAP_FRAME:  ffffd000225d7440 -- (.trap 0xffffd000225d7440)
NOTE: The trap frame does not contain all registers.
Some register values may be zeroed or incorrect.
rax=fffff80185bea3c8 rbx=0000000000000000 rcx=0000000000000002
rdx=0000000000000002 rsi=0000000000000000 rdi=0000000000000000
rip=fffff96000869ccd rsp=ffffd000225d75d0 rbp=0000000000000000
 r8=0000000000000002  r9=00000000ffffe90 r10=ffffe001da2c9ab0
r11=0000000000000010 r12=0000000000000000 r13=0000000000000000
r14=0000000000000000 r15=0000000000000000
iopl=0         nv up ei pl nz na pe nc
cdd!FillColorKeyCallback+0xbd:
fffff960`00869ccd 0fc328          movnti  dword ptr [rax],ebp ds:fffff801`85bea3c8
=????????
Resetting default scope

LAST_CONTROL_TRANSFER:  from fffff80263c60f1e to fffff80263bd4d90

STACK_TEXT:
ffffd000`225d6a38 fffff802`63c60f1e : 00000000`00000000 00000000`00000000 ffffd000`225d6ba
0 fffff802`63b52de8 : nt!DbgBreakPointWithStatus
ffffd000`225d6a40 fffff802`63c6082f : 00000000`00000003 ffffd000`225d6ba0 fffff802`63bdc1d
0 ffffd000`225d70f0 : nt!KiBugCheckDebugBreak+0x12
ffffd000`225d6aa0 fffff802`63bce2a4 : 00000000`00000000 ffffe001`d7256360 00000000`0000000
0 00000000`00000000 : nt!KeBugCheck2+0x8ab
ffffd000`225d71b0 fffff802`63c07ab8 : 00000000`00000050 fffff801`85bea3c8 00000000`0000000
1 ffffd000`225d7440 : nt!KeBugCheckEx+0x104
ffffd000`225d71f0 fffff802`63ae3e78 : 00000000`00000001 ffffe001`dac40900 ffffd000`225d744
0 fffff960`008625ce : nt! ?? #####NODOBFM#####string'+0x29408
ffffd000`225d7290 fffff802`63bd842f : ffffd000`225d7b10 fffff960`00868680 fffff901`400c400
0 ffffd000`225d7440 : nt!MmAccessFault+0x758
ffffd000`225d7440 fffff960`00869ccd : 00000000`00000000 fffff960`00869c10 ffffd000`225d84d
0 ffffd000`225d84d0 : nt!KiPageFault+0x12f
ffffd000`225d75d0 fffff960`00867f9f : 00000000`00000000 ffffd000`00000001 ffffd000`225d7b1
0 fffff960`0015f7f2 : cdd!FillColorKeyCallback+0xbd
ffffd000`225d7610 fffff960`00865e66 : fffff901`407e1940 ffffd000`00000001 fffff901`407e194
0 ffffd000`225d7c00 : cdd!ClipDstRects+0x4cf
ffffd000`225d7870 fffff960`0028e1bb : fffff901`400bc010 00000000`00000000 00000000`0000000
0 00000000`00000000 : cdd!DrvTextOut+0x7a6
ffffd000`225d7de0 fffff960`001661e5 : fffff901`446b65d0 ffffd000`225d86d0 ffffd000`225d83c
0 00000000`00000000 : win32k!SpTextOut+0x20c
ffffd000`225d82b0 fffff960`0020da7c : 00000000`00000000 00000000`00000000 ffffd000`225d8a1
0 00000000`00000000 : win32k!GreExtTextOutWLocked+0x925
ffffd000`225d8740 fffff960`0016974e : 00000000`00004a01 fffff901`40224100 00000000`0099e7d
9 fffff901`00ffffff : win32k!GreBatchTextOut+0x1f4
ffffd000`225d87e0 fffff960`001b5671 : 00000000`00000000 00000000`00000007 ffffe001`da2aa9d
0 00000000`7f1cc000 : win32k!NtGdiFlushUserBatch+0x5ce
ffffd000`225d8a70 fffff802`63e6240a : 00000000`00000001 00000000`00000000 00000000`0000000
7 00000000`7f1cc000 : win32k!W32CalloutDispatch+0x1d1
ffffd000`225d8ac0 fffff802`63bd98c7 : 00000000`00cefa04 fffff960`00201d40 00000000`0000002
0 00000000`7f1cc000 : nt!PsInvokeWin32Callout+0x42
ffffd000`225d8b00 00000000`77572772 : 00000000`77572738 00000023`75e7d472 00000000`0000002
3 00000000`00001004 : nt!KiSystemServiceGdiTebAccess+0x33
00000000`00b8e7e8 00000000`77572738 : 00000023`75e7d472 00000000`00000023 00000000`0000100
4 00000000`00cefca8 : wow64cpu!CpupSyscallStub+0x2
00000000`00b8e7f0 00000000`7760323a : 00000000`00000000 00000000`77571503 00000000`0000000
0 00000000`77603420 : wow64cpu!Thunk0Arg+0x5
00000000`00b8e8a0 00000000`7760317e : 00000000`00000000 00000000`00000000 00000000`00b8fd3
0 00000000`00b8f210 : wow64!RunCpuSimulation+0xa
00000000`00b8e8f0 00007ffa`9f86533b : 00000000`00ba00f8 00000000`00000000 00000000`0000001
0 00000000`7f1c4000 : wow64!Wow64LdrpInitialize+0x172
00000000`00b8ee30 00007ffa`9f85826a : 00007ffa`9f7a0000 00000000`00000000 00000000`0000000
0 00000000`7f1c4000 : ntdll!LdrpInitializeProcess+0x157b
00000000`00b8f150 00007ffa`9f809c6a : 00000000`00b8f210 00000000`00000000 00000000`0000000
0 00000000`7f1c4000 : ntdll!_LdrpInitialize+0x4e5ae
00000000`00b8f1c0 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`0000000
0 00000000`00000000 : ntdll!LdrInitializeThunk+0xe


STACK_COMMAND:  kb

FOLLOWUP_IP:
cdd!FillColorKeyCallback+bd
fffff960`00869ccd 0fc328          movnti  dword ptr [rax],ebp
```

CREDIT

Discovered by Piotr Bania and Andrea Allievi of Cisco Talos.