微软 SMBv3 客户端/服务端远程代码执行漏洞（CVE-2020-0796）技术分析

微软安全中心在北京时间 3 月 12 日 23 时发布了影响 Windows 10 等系统用户的 SMBv3 远程代码执行漏洞补丁。我们建议受影响的用户尽快按微软更新信息指南安装该补丁：https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-0796。 同时，360Vulcan Team 对该漏洞进行了快速分析，该漏洞属于高危的零接触远程代码执行漏洞，技术分析如下。

SMB 漏洞分析

根本原因

漏洞发生在 srv2.sys 中,由于 SMB 没有正确处理压缩的数据包,在解压数据包的时候使用客户端传过来的长度进行解压时,并没有检查长度是否合法.最终导致整数溢出。

详细分析

SMB v3 中支持数据压缩,如果 SMB Header 中的 ProtocolId 为 0x424D53FC 也就是 0xFC, 'S', 'M', 'B'.那么就说明数据是压缩的,这时 smb 会调用压缩解压处理的函数.

首先 SMB 会调用 srv2!Srv2ReceiveHandler 函数接收 smb 数据包，并根据 ProtocolId 设置对应的处理函数。

```
1.  __int64 __fastcall Srv2ReceiveHandler(__int64 a1, void *Src, __int64 a3, unsig
    ned int a4, unsigned int *a5, char *Srca, struct _SLIST_ENTRY *a7, _QWORD *a8)
2.  {
3.      ...
4.      //
5.      // 这里判断头部 ProtocolId
6.      //
7.       if ( **((_DWORD **)&v20[15].Next[1].Next + 1) == 'BMS\xFC' )
8.        {
9.          if ( KeGetCurrentIrql() > 1u )
10.         {
11.           v20[14].Next = (_SLIST_ENTRY *)v11;
12.           v20[2].Next = (_SLIST_ENTRY *)Srv2DecompressMessageAsync;
13.           v43 = HIDWORD(v20->Next) == 5;
14.           *((_DWORD *)&v20[3].Next + 2) = 0;
15.           if ( v43 )
16.           {
17.             LOBYTE(v71) = 1;
18.             LOBYTE(v35) = 1;
19.             SRV2_PERF_ENTER_EX(&v20[32].Next + 1, v35, 307i64, "Srv2PostToThrea
    dPool", (_DWORD)v71);
20.           }
21.           v44 = *((_QWORD *)&v20[3].Next[8].Next + 1);
22.           v45 = *(_QWORD *)(v44 + 8i64 * KeGetCurrentNodeNumber() + 8);
23.           if ( !ExpInterlockedPushEntrySList((PSLIST_HEADER)(v45 + 16), v20 +
    1) && *(_WORD *)(v45 + 66) )
24.             RfspThreadPoolNodeWakeIdleWorker(v45);
25.           goto LABEL_168;
26.             }
27.         }
```

```
28. }
```

从代码中我们可以看出如果是压缩的数据则调用 Srv2DecompressMessageAsync 函数进行解压缩. srv2!Srv2DecompressMessageAsync 会调用 Srv2DecompressData 函数.

结构如下:

```
1.  typedef struct _COMPRESSION_TRANSFORM_HEADER
2.  {
3.    ULONG ProtocolId;
4.    ULONG OriginalCompressedSegmentSize;
5.    USHORT CompressionAlgorithm;
6.    USHORT Flags;
7.    ULONG Length;
8.  }COMPRESSION_TRANSFORM_HEADER, *PCOMPRESSION_TRANSFORM_HEADER;
```

产生整数溢出漏洞的代码如下：

```
1.  __int64 __fastcall Srv2DecompressData(__int64 pData)
2.  {
3.    __int64 v2; // rax
4.    COMPRESSION_TRANSFORM_HEADER Header; // xmm0 MAPDST
5.    __m128i v4; // xmm0
6.    unsigned int CompressionAlgorithm; // ebp
7.    __int64 UnComparessBuffer; // rax MAPDST
8.    int v9; // eax
9.    int v11; // [rsp+60h] [rbp+8h]
10.   v11 = 0;
11.   v2 = *(_QWORD *)(pData + 0xF0);
12.   if ( *(_DWORD *)(v2 + 0x24) < 0x10u )          // 这里判断数据包长度的最小值
13.     return 0xC000090Bi64;
```

```
14.  Header = *(COMPRESSION_TRANSFORM_HEADER *)*(_QWORD *)(v2 + 0x18);// [v2+0x1
     8]中为客户端传进来的 Buffer
15.                                                 // [v2+0x24]为数据包长度
16.  v4 = _mm_srli_si128((__m128i)Header, 8);
17.  CompressionAlgorithm = *(_DWORD *)(*(_QWORD *)(*(_QWORD *)(pData + 0x50) + 0
     x1F0i64) + 0x8Ci64);
18.  if ( CompressionAlgorithm != v4.m128i_u16[0] )
19.    return 0xC00000BBi64;
20.  UnCompressBuffer = SrvNetAllocateBuffer((unsigned int)(Header.OriginalCompre
     ssedSegmentSize + v4.m128i_i32[1]), 0i64);// OriginalCompressedSegmentSize + C
     ompressedSegmentSize，这里没有检查相加的值，导致整数溢出，分配一个较小的 UnCompres
     sBuffer
21.  if ( !UnComparessBuffer )
22.    return 0xC000009Ai64;
23.  if ( (int)SmbCompressionDecompress(
24.              CompressionAlgorithm,             // CompressionAlgorithm
25.              *(_QWORD *)(*(_QWORD *)(pData + 0xF0) + 0x18i64) + (unsigned int)
     Header.Length + 0x10i64,// CompressedBuffer
26.              (unsigned int)(*(_DWORD *)(*(_QWORD *)(pData + 0xF0) + 0x24i64) -
      Header.Length - 0x10),// CompressedBufferSize
27.              (unsigned int)Header.Length + *(_QWORD *)(UnComparessBuffer + 0x1
     8),// UncompressedBuffer，会传入 SmbCompressionDecompress 函数进行 Decompress 处
     理。
28.              Header.OriginalCompressedSegmentSize,
29.              &v11) < 0
30.    || (v9 = v11, v11 != Header.OriginalCompressedSegmentSize) )
31.  {
32.    SrvNetFreeBuffer(UnComparessBuffer);
33.    return 0xC000090Bi64;
34.  }
```

```
35.  if ( Header.Length )
36.  {
37.    memmove(
38.      *(void **)(UnComparessBuffer + 24),
39.      (const void *)(*(_QWORD *)(*(_QWORD *)(pData + 240) + 24i64) + 16i64),
40.      (unsigned int)Header.Length);
41.    v9 = v11;
42.  }
43.  *(_DWORD *)(UnComparessBuffer + 36) = Header.Length + v9;
44.  Srv2ReplaceReceiveBuffer(pData, UnComparessBuffer);
45.  return 0i64;
46. }
```

通过上述代码我们可以看到,这个 Header 中的两个长度都没有进行检查. 下面是 MS 文档中对这两个长度的描述:

1. OriginalCompressedSegmentSize (4 bytes) The size, in bytes, of the uncompressed data segment.
2. Offset/Length (4 bytes) If SMB2_COMPRESSION_FLAG_CHAINED is set in Flags field, this field MUST be interpreted as Length. The length, in bytes, of the compressed payload. Otherwise, this field MUST be interpreted as Offset. The offset, in bytes, from the end of this structure to the start of compressed data segment.

随后在 srv2!Srv2DecompressData 函数中，会调用 SmbCompressionDecompress，进而最终调用 nt!RtlDecompressBufferXpressLz 进行数据解压，调用路径如下：

```
1.  ffff9480`20e2ad98  nt!RtlDecompressBufferXpressLz+0x2d0
2.  ffff9480`20e2adb0  nt!RtlDecompressBufferEx2+0x66
3.  ffff9480`20e2ae00  srvnet!SmbCompressionDecompress+0xd8
4.  ffff9480`20e2ae70  srv2!Srv2DecompressData+0xdc
```

nt!RtlDecompressBufferXpressLz 会在 smb compress 协议数据包里解析字段，其中包含需要 Decompress buffer 的大小，并和之前通过 SrvNetAllocateBuffer 分配的 buffer 的 OriginalCompressedSegmentSize 进行比较，确认其大小不大于 OriginalCompressedSegmentSize，随后进行 memcpy。

```
1.  signed __int64 __fastcall RtlDecompressBufferXpressLz(_BYTE *a1, unsigned int
    a2, _BYTE *a3, unsigned int a4, __int64 a5, _DWORD *a6)
2.  {
3.          v9 = &a1[a2];
4.          ....
5.          if ( &a1[v21] > v9 )
6.            return 0xC0000242i64;
7.          ...
8.          v33 = a1;
9.          a1 += v21;
10.         qmemcpy(v33, v23, v21);
11. }
```

如上伪代码所示，a1 指向 SrvNetAllocateBuffer 分配的 UncompressBuffer，a2 的值为 OriginalCompressedSegmentSize，v21 的值为从 smb 数据包中解析的解压缩数据的大小，该值可由攻击者控制，若该大小大于 OriginalCompressedSegmentSize，则会返回 0xC0000242 错误，由于之前对长度没有检查，如果我们传入一个很大的 OriginalCompressedSegmentSize 触发整数溢出，同时 v21 就可以设置一个极大值，而依然可以通过对 decompress size 的判断，最终调用 qmemcpy 拷贝一个极大的 size 导致缓冲区溢出。

补丁

微软的补丁就是在 Srv2DecompressData 里面对上述两个 Length 做了检查.

```
1.  __int64 __fastcall Srv2DecompressData(__int64 a1)
2.  {
3.        //
4.        //添加了对 Header 中 Length 的检查
5.        //
6.        if ( RtlULongAdd(Header.OriginalCompressedSegmentSize, Header2.Length, &
    pulResult) < 0 )
7.        {
8.            ...
9.        }
10.       if ( pulResult > (unsigned __int64)(unsigned int)(*(_DWORD *)(v9 + 0x24)
    + 0x100) + 0x34 )
11.     {
12.         ...
13.     }
14.       if ( RtlULongSub(pulResult, Header.Length, &pulResult) < 0 )
15.     {
16.         ...
17.     }
18. }
```

协议文件见: SMB2 COMPRESSION_TRANSFORM_HEADER