

CVE-2012-0179 分析报告

启明星辰 2012/5

漏洞概要

最近在某国外网站上放出了 CVE-2012-0179 (MS12-032) 的利用 POC，该漏洞为一个本地漏洞，当在 Windows7，Windows2008 等系统上绑定一个由 ipv4 扩展的 ipv6 地址的时候，如果绑定的地址未在本机设置（地址无效），则会出现一个内存的 double-free 错误，导致蓝屏。下面是该漏洞的详细分析。

漏洞分析

漏洞利用代码（部分）如下图所示：

```
bool bindSuccess = false;

while(!bindSuccess)
{
    SOCKET sock = WSASocket(AF_INET6,
        SOCK_DGRAM,
        IPPROTO_UDP,
        NULL,
        0,
        WSA_FLAG_OVERLAPPED);
    if(sock == INVALID_SOCKET)
    {
        printf("WSASocket failed\n");
        exit(-1);
    }

    DWORD val = 0;
    if (setsockopt(sock,
        IPPROTO_IPV6,
        IPV6_V6ONLY,
        (const char*)&val,
        sizeof(val)) != 0)
    {
        printf("setsockopt failed\n");
        closesocket(sock);
        exit(-1);
    }

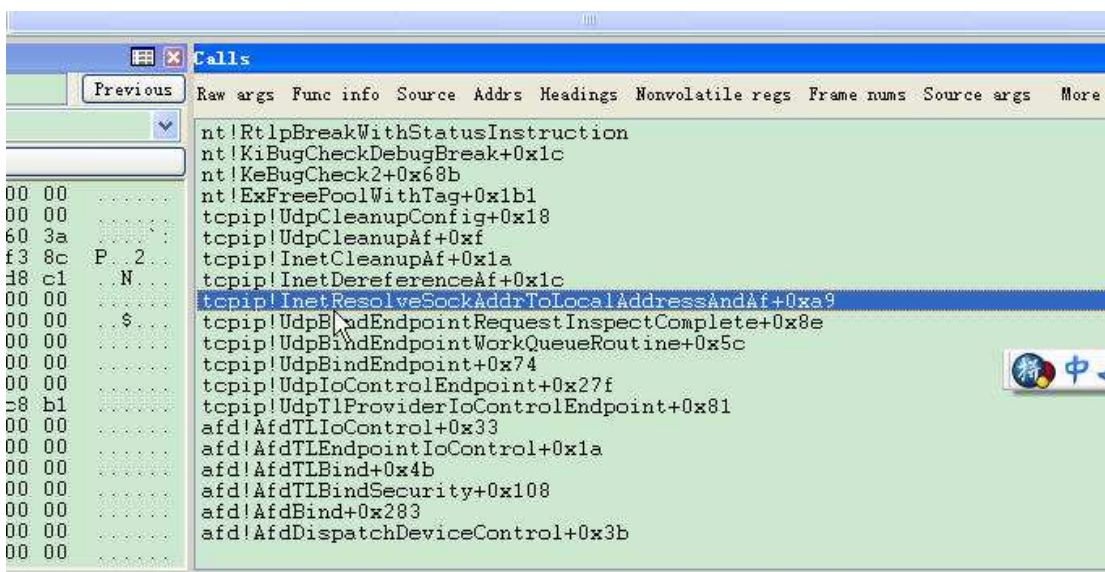
    sockaddr_in6 sockAddr;
    memset(&sockAddr, 0, sizeof(sockAddr));
    sockAddr.sin6_family = AF_INET6;
    sockAddr.sin6_port = htons(5060);

    // set address to IPV6-mapped 169.13.13.13 (not configured on the local machine)
    // that is [::FFFF:169.13.13.13]
    sockAddr.sin6_addr.u.Byte[15] = 1;
    sockAddr.sin6_addr.u.Byte[14] = 0;
    sockAddr.sin6_addr.u.Byte[13] = 0;
    sockAddr.sin6_addr.u.Byte[12] = 127;
    sockAddr.sin6_addr.u.Byte[11] = 0xFF;
    sockAddr.sin6_addr.u.Byte[10] = 0xFF;

    int size = 28; // 28 is sizeof(sockaddr_in6)
```

这里循环对于一个不存在的 ipv4 地址 169.13.13.13 扩展的 ipv6 地址进行绑定。在循环了几次后，便会出现蓝屏。

崩溃时候截图如下



对此漏洞，微软修补了 tcpip.sys 这个内核驱动，我们通过对比找到了线索

Subroutine 01	Subroutine 02	Unidentified 01	Unidentified 02	Identified Members	Match Rate
_FpAlpNotificationWorkQueueRoutine04	_FpAlpNotificationWorkQueueRoutine04	1	5	1	0.0822580645163
_FpSetIpAddress04	_FpSetIpAddress04	6	55	19	0.317883597884
_InetResolveSockAddrToLocalAddressAndAf036	_InetResolveSockAddrToLocalAddressAndAf036	4	7	7	0.446512204716
_TcpGetNextRssProcessingTick04	_TcpGetNextRssProcessingTick04	2	3	6	0.488039019804
_TcpTimerInitializeTimer08	_TcpTimerInitializeTimer08	0	0	1	0.518348623853
_WfpAlpInitialize00	_WfpAlpInitialize00	2	1	3	0.666666666667
_FlUnbindAdapter08	_FlUnbindAdapter08	0	12	20	0.72133658197
_IdpIsFlLuidPublic012	_IdpIsFlLuidPublic012	1	1	7	0.875
_OlsStartTimer0cb012	_OlsStartTimer0cb012	0	0	9	0.876661918329
_IppProcessRawData012	_IppProcessRawData012	3	7	60	0.890256604106
_TcpRepartitionTimerWheels04	_TcpRepartitionTimerWheels04	3	3	68	0.903045840482
_IppProcessSessionState012	_IppProcessSessionState012	2	2	24	0.907692307692
_IppSendDatagramsCommon020	_IppSendDatagramsCommon020	1	10	92	0.94358974359
_TcpPartitionGetNextExpirationTick08	_TcpPartitionGetNextExpirationTick08	0	0	22	0.944273985187
_loc_93994	_loc_93454	2	5	84	0.94435872134
_IppStartUfeTimer04	_IppStartUfeTimer04	0	0	5	0.946666666667
_IFSecSetOutboundContextNoAcquireInitiateTun...	_IFSecSetOutboundContextNoAcquireInitiateTunnel04	0	0	1	0.952941176471
_TcpCreateTimeWaitTcb04	_TcpCreateTimeWaitTcb04	0	0	28	0.95686189669
_RawBindEndpointInspectComplete08	_RawBindEndpointInspectComplete08	0	1	19	0.957535014006
_IFSecSetOutboundContextNoAcquireInitiateTun...	_IFSecSetOutboundContextNoAcquireInitiateTunnel04	0	0	1	0.958974358974
_UdpCreateEndpointWorkQueueRoutine04	_UdpCreateEndpointWorkQueueRoutine04	1	1	25	0.961538461538
_TcpTimeWaitTcbReceive016	_TcpTimeWaitTcbReceive016	0	0	32	0.961849090157
_EqsInitializeDeviceID012	_EqsInitializeDeviceID012	0	0	11	0.963636363636
_EqsUpdateTcpNSIPProvider04	_EqsUpdateTcpNSIPProvider04	0	0	11	0.963636363636
_TcpReplaceTimeWaitTcbWithSynTcb08	_TcpReplaceTimeWaitTcbWithSynTcb08	0	0	6	0.965079365079
_TcpShutdownTimeWaitTcb04	_TcpShutdownTimeWaitTcb04	0	0	6	0.965079365079
_TcpQueueNotifyBacklogChangeSend08	_TcpQueueNotifyBacklogChangeSend08	0	0	1	0.965217391304
_EqsStartPolicyModule00	_EqsStartPolicyModule00	0	0	25	0.968
_UdpBindEndpointInspectComplete08	_UdpBindEndpointInspectComplete08	0	1	19	0.968047337278
_OlsCleanupTimer0cb04	_OlsCleanupTimer0cb04	0	0	6	0.968115942029
_TcpBindEndpointInspectComplete08	_TcpBindEndpointInspectComplete08	0	1	27	0.970181818182
_EqsLogOnePolicyFailure020	_EqsLogOnePolicyFailure020	0	0	27	0.97037037037
_EqsDispatchControl08	_EqsDispatchControl08	0	0	7	0.971428571429
_EqsLogPolicyFarsingSucceeded08	_EqsLogPolicyFarsingSucceeded08	0	0	7	0.971428571429
_TcpReplaceTimeWaitTcbWithTcb08	_TcpReplaceTimeWaitTcbWithTcb08	0	0	8	0.97380952381
_TcpStartPartitionModule00	_TcpStartPartitionModule00	1	1	38	0.974152815359
_TcnTcbEcnCodepoint012	_TcnTcbEcnCodepoint012	0	0	8	0.975

Subroutine 01	Subroutine 02	Unidentified 01	Unidentified 02	Identified Members	Match Rate
_TcpReplaceTimeWaitTcbWithTcb08	_TcpReplaceTimeWaitTcbWithTcb08	0	0	8	0.97380952381
_TcpStartPartitionModule00	_TcpStartPartitionModule00	1	1	38	0.974152815359
_TcpTcbEcnCodepoint012	_TcpTcbEcnCodepoint012	0	0	8	0.975
_OlsGetNextInterface016	_OlsGetNextInterface016	0	0	23	0.980156075808
_OlsQuerySendCallInProgress08	_OlsQuerySendCallInProgress08	0	0	6	0.980952380952
_InetResolveSockAddrToAf036	_InetResolveSockAddrToAf036	0	0	18	0.983333333333
_EqsReadAndEstablishMachinePolicyTable08	_EqsReadAndEstablishMachinePolicyTable08	0	0	13	0.984615384615
_IppInitializePathSet04	_IppInitializePathSet04	0	0	9	0.987794463602
_EqsProcessMachinePolicyChange012	_EqsProcessMachinePolicyChange012	0	0	17	0.988235294118
_OlsSetTcpInterfaceParametersByLuid012	_OlsSetTcpInterfaceParametersByLuid012	0	0	9	0.988846960168
_OlsFindAndReferenceInterfaceByLuid04	_OlsFindAndReferenceInterfaceByLuid04	0	0	7	0.989115646259
_IFSecValidateIFSecS0012	_IFSecValidateIFSecS0012	0	0	16	0.99

从上图我们找出了和崩溃时堆栈回溯相关的几个函数。经过逆向，我们断在 **UdpEndpointRequestInspectComplete** 这个函数上分析这个漏洞。该函数是在上层执行 **bind** 的时候跑到的，因为源代码中绑定的是 UDP 端口，因此自然会跑到 Udp 相关的函数来。

这里有几个重要的结构

StructA（该结构为 Windows 初始化的时候就确定的）

偏移 4 指向 StructB

偏移 0x14 StructIpv4/Ipv6

偏移 0x18 flag of Ipv4/Ipv6
 偏移 0x1a Ipv4 to Ipv6 flag
 偏移 0x1c user buffer
 StructB
 偏移 0x14 StructIpv4/Ipv6

tcpip!UdpBindEndpointRequestInspectComplete:

```

8f458a88 8bff    mov     edi,edi
8f458a8a 55      push    ebp
8f458a8b 8bec    mov     ebp,esp
8f458a8d 83ec0c  sub     esp,0Ch
8f458a90 837d0c00 cmp     dword ptr [ebp+0Ch],0
8f458a94 53      push    ebx
8f458a95 56      push    esi
8f458a96 8b7508  mov     esi,dword ptr [ebp+8] ss:0010:9254b9b8=8dfe1020 ;这里有一个结构，该
结构指针是从 UdpEndpoint 的一个工作队列中取得的，经过逆向发现是开机初始化的时候就
会存下来的一个指针。因为没有具体文档参考，我们将其命名为 StructA
8f458a99 8b5e04  mov     ebx,dword ptr [esi+4] ;在 structA 偏移 4 的位置存放一个指针
8f458aa2 807d1000 cmp     byte ptr [ebp+10h],0
8f458aa6 8b5514  mov     edx,dword ptr [ebp+14h] ss:0010:9254b9c4=8dfe103c(这里传进来
一个参数，该参数为 StructA 偏移 0x1c 处的指针
8f458aa9 57      push    edi
8f458aaa 7431    je      tcpip!UdpBindEndpointRequestInspectComplete+0x55 (8f458add) ;这
里跳走

```

Memory	
Virtual: 8dfe1020	Display format: Byte
8dfe1020	00 00 00 00 18 3e 4d 8e 58 68 78 8f 88 b1 81 8c 00 00 00 00 e8 4d 37 8d 17 00 00 00 17 00 13 c4 00
8dfe1041	00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff a9 0d 0d 0d 00 00 00 00 00 00 72 00 76 00 69 00 00 00
8dfe1062	00 00 00 00 00 00 00 00 00 00 00 00 f8 ba 4a 8e a3 64 78 8f e0 e1 05 8e 14 10 75 8f d0 15 de 8d 02 00 00

之后会将 StructA 不同位置的指针传入 InetResolveSockAddrToLocalAddressAndAf 函数中

```

.text:00052AE7      lea     edi,[esi+10h]; StructA 偏移 0x10
.text:00052AEA      push    edi
.text:00052AEB      lea     eax,[esi+14h] ; StructA 偏移 0x14
.text:00052AEE      push    eax
.text:00052AEF      moveax,[ebx+0Ch] ; StructB 偏移 0xC
.text:00052AF2      lea     ecx,[esi+1Ah];StructA 偏移 0x1A
.text:00052AF5      push    ecx
.text:00052AF6      shreax,4
.text:00052AF9      lea     edx,[esi+18h];StructA 偏移 0x18
.text:00052AFC      push    edx
.text:00052AFD      push    [ebp+struct1c] ;StructA 偏移 0x1C
.text:00052B00      and     al,1
.text:00052B02      movzx   eax,al
.text:00052B05      push    eax
.text:00052B06      push    dword ptr [ebx+28h] ;StructB 偏移 0x28

```

```

.text:00052B09          push    dwordptr [ebx+14h] ;StructB 偏移 0x14
.text:00052B0C          push    offset _UdpInetTransport;
.text:00052B11          call    InetResolveSockAddrToLocalAddressAndAf(x,x,x,x,x,x,x,x)

```

之后 InetResolveSockAddrToLocalAddressAndAf 又将传进来的参数传给了 InetResolveSockAddrToAf 函数

```

.text:00067457          movedi, edi
.text:00067459          push    ebp
.text:0006745A          movebp, esp
.text:0006745C          push    ebx
.text:0006745D          movebx, [ebp+structA14]
.text:00067460          push    esi
.text:00067461          movesi, [ebp+structB28]
.text:00067464          push    edi
.text:00067465          movedi, [ebp+structA1c]
.text:00067468          push    ebx
.text:00067469          push    [ebp+structA18]
.text:0006746C          lea     eax, [ebp+structA18+3]
.text:0006746F          push    [ebp+structA1a]
.text:00067472          push    eax
.text:00067473          push    edi
.text:00067474          push    [ebp+flag]
.text:00067477          push    esi
.text:00067478          push    [ebp+structB14]
.text:0006747B          push    [ebp+arg_0]
.text:0006747E          call    InetResolveSockAddrToAf(x,x,x,x,x,x,x,x)

```

进入 InetResolveSockAddrToAf 函数:

tcpip!InetResolveSockAddrToAf:

```

8f46d50e 8bff      mov     edi,edi
8f46d510 55        push    ebp
8f46d511 8bec      mov     ebp,esp
8f46d513 53        push    ebx
8f46d514 8b5d0c    mov     ebx,dword ptr [ebp+0Ch] ss:0010:9254b940=8d374de8 ; 取出
StructB 偏移 0x14 处的值
8f46d517 66837b0c17 cmp     word ptr [ebx+0Ch],17h ;这里是一个标志，用于比较是否是
Ipv6 地址格式

```



这里为 0x17，显示是一个 IPv6 地址格式；另外通过逆向得知 0x2 表示 ipv4 地址格式

.text:00067520 cmp [ebp+flag], 0 ; 比较刚才传进来的标志位，猜测是表示该 IPv6 地址是否为 ipv4 地址扩展而来的标志

.text:00067524 jnz short loc_6758E; 如果是 ipv4 扩展而来的 ipv6 地址，则不跳转

.text:00067526 mov esi, [ebp+structA1c] ; StructA 偏移 0x1C 位置传进来的是用户态在执行 bind 函数时传进来的 UserBuffer

具体原因:

如图: AfdDispatchDeviceControl 函数通过从三环传进来的 IoControlCode 中计算处理函数的 index 值。这里调用的为 AfdBind 函数。

```
PAGEAFD:000314C9      mov     edi, edi
PAGEAFD:000314CB      push   ebp
PAGEAFD:000314CC      mov     ebp, esp
PAGEAFD:000314CE      mov     ecx, [ebp+arg_4]
PAGEAFD:000314D1      mov     edx, [ecx+60h]
PAGEAFD:000314D4      push   esi
PAGEAFD:000314D5      push   edi
PAGEAFD:000314D6      mov     edi, [edx+IO_STACK_LOCATION.Parameters.DeviceIoControl.IoControlCode]
PAGEAFD:000314D9      mov     eax, edi
PAGEAFD:000314DB      shr     eax, 2
PAGEAFD:000314DE      and     eax, 3FFh
PAGEAFD:000314E3      cmp     eax, 46h
PAGEAFD:000314E6      jnb     short loc_31506
PAGEAFD:000314E6      mov     esi, eax
PAGEAFD:000314E8      shl     esi, 2
PAGEAFD:000314EA      cmp     ds:AfdIoctlTable[esi], edi
PAGEAFD:000314ED      jnz     short loc_31506
PAGEAFD:000314F3      mov     [edx+1], al
PAGEAFD:000314F5      mov     esi, ds:AfdIrpCallDispatch[esi]
PAGEAFD:000314F8      test    esi, esi
PAGEAFD:000314FE      jz      short loc_31506
PAGEAFD:00031500      call    esi
PAGEAFD:00031502      jmp     short loc_3151C
PAGEAFD:00031504
PAGEAFD:00031506      ; -----
PAGEAFD:00031506      ; 000F XREF: AfdIoctlTable[esi], edi
PAGEAFD:00031506      ; 000F XREF: AfdIoctlTable[esi], edi
```

AfdBind 函数会首先检查输入的缓冲区长度和输出缓冲区长度是否合法

```
mov     edi, edx          ; Io_StackLocation
mov     [ebp+var_40], edi
mov     [ebp+Irp], ecx
xor     ebx, ebx
mov     [ebp+var_2C], ebx
mov     eax, [edi+IO_STACK_LOCATION.FileObject]
mov     esi, [eax+0Ch]
mov     [ebp+var_38], esi
mov     [ecx+1Ch], ebx
test    dword ptr [esi+8], 200h
jnz     short loc_24C98

mov     ecx, [edi+IO_STACK_LOCATION.Parameters.DeviceIoControl.InputBufferLength]
cmp     ecx, 8
jb      short failed

mov     eax, [edi+IO_STACK_LOCATION.Parameters.DeviceIoControl.OutputBufferLength]
cmp     eax, 0Ch
jb      short failed

add     ecx, 0FFFFFFCh
```

之后从用户缓冲区偏移 4 的位置拷贝了一个结构给刚申请的一个 buf, 该 buf 正是 StructA 偏移 0x1c 处的结构。

```
PAGE:00024D49      ; AfdBind(x,x)+10F1j
PAGE:00024D49      push   0EC646641h          ; Tag
PAGE:00024D4E      push   [ebp+Number0fBytes] ; Number0fBytes
PAGE:00024D51      push   10h                ; PoolType
PAGE:00024D53      call   ds:ExAllocatePoolWithQuotaTag(x,x,x)
PAGE:00024D59      mov     [ebp+P], eax
PAGE:00024D5C      push   [ebp+Number0fBytes] ; size_t
PAGE:00024D5F      test    dword ptr [esi+8], 200h
PAGE:00024D66      jnz     short loc_24DAA
PAGE:00024D68      mov     eax, [edi+IO_STACK_LOCATION.Parameters.DeviceIoControl.Type3InputBuffer]
PAGE:00024D6B      mov     ecx, [eax]
PAGE:00024D6D      mov     [ebp+var_30], ecx
PAGE:00024D70      add     eax, 4              ; 用户传进来的Buffer偏移4的位置为该结构
PAGE:00024D73      push   eax                  ; void *
PAGE:00024D74      push   [ebp+P]              ; void *
PAGE:00024D77      call   ds:_imp__memmove
PAGE:00024D7D      add     esp, 0Ch
PAGE:00024D80      mov     ecx, [ebp+P]
```

如下图, 该结构开头的 2 字节大小 0x17 标识了 ip 地址格式, 该结构偏移 8 字节开始是 Ip 地址。

Virtual: 8dfe103c	
8dfe103c	17 00 13 c4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff a9 0d 0d 0d 00
8dfe105d	00 69 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f8 ba 4a 8e a3 64 78 8f e0 e1
8dfe107e	de 8d 02 00 00 00 02 00 00 8a c0 a8 ac 17 00 00 00 00 00 00 00 00 00 00 00
8dfe109f	00 00

继续 InetResolveSockAddrToAf 函数:

```
.text:00067529      lea     eax, [esi+8]; 取得 IP 地址指针
.text:0006752C      push   eax
.text:0006752D      call   IN6_IS_ADDR_V4MAPPED(x); 检验 IP 地址是否
```

为 ipv4 地址映射而来。由于 IPv4 地址是 32 位, IPv6 地址为 128 位, 因此在将 IPv4 地址扩展为 IPv6 地址的时候前面被填充了 0, 后面的前 16 位填充为 1, 最后 32 位为 IPv4 地址。

```
.text:00067532      test    al, al
.text:00067534      jz      short loc_67569; 如果是 IPv4 映射而来, 则
```

返回 1, 不跳走

```
.text:00067536      moveax, [ebp+structA1a]
.text:00067539      movedi, [ebp+structA18]
.text:0006753C      push    2
.text:0006753E      mov     byte ptr [eax], 1; StructA1a 也是一个标志位
.text:00067541      pop     eax
.text:00067542      push    eax
.text:00067543      push    [ebp+arg_0]
.text:00067546      mov     [edi], ax ;因为是 ipv4 扩展而来, 则将 structA 偏移
```

0x18 位置的标志改为 0x2

```
.text:00067549      call    InetFindAndReferenceAf(x,x) ;这里查找 ipv4 相关的结构
(StructIpv4/Ipv6), 该结构记录了绑定的相关协议(ipv4)的引用计数, 这里会查找到相应的
结构 (以之前的 UdpInetTransport 为基地址), 并将其中的引用计数增 1, 并返回相应的结构
指针, 我们暂且称该结构为 StructIPV4, 如下图 (圈住的为引用计数)
```

8f498c3f	56	push	esi
8f498c40	8b7508	mov	esi, dword ptr [ebp+8]
8f498c43	57	push	edi
8f498c44	8d4e78	lea	ecx, [esi+78h]
8f498c47	8d550b	lea	edx, [ebp+0Bh]
8f498c4a	e839000000	call	tcpip!RtlAcquireReadLock (8f498c88)
8f498c4f	0fb7450c	movzx	eax, word ptr [ebp+0Ch]
8f498c53	8bbc86ec000000	mov	edi, dword ptr [esi+eax*4+0ECh]
8f498c5a	85ff	test	edi, edi
8f498c5c	740a	jbe	tcpip!InetFindAndReferenceAf+0x2e (8f498c68)
8f498c5e	33c9	xor	ecx, ecx
8f498c60	8d4708	lea	eax, [edi+8]
8f498c63	41	inc	ecx
8f498c64	f00fc108	lock xadd	dword ptr [eax], ecx ds:0023:8cf521f8=00000016
8f498c68	83c67c	add	esi, 7Ch
8f498c6b	83c8ff	or	eax, 0FFFFFFFh
8f498c6e	f00fc106	lock xadd	dword ptr [esi], eax
8f498c72	8a4d0b	mov	cl, byte ptr [ebp+0Bh]
8f498c75	ff15c4474f8f	call	dword ptr [tcpip!_imp_KfLowerIrql (8f4f47c4)]
8f498c7b	8bc7	mov	eax, edi
8f498c7d	5f	pop	edi
8f498c7e	5e	pop	esi
8f498c7f	5d	pop	ebp
8f498c80	c20800	ret	8
8f498c83	90	nop	
8f498c84	90	nop	

Virtual: 8cf521f0	Display format: Byte
8cf521f0	00 00 00 00 00 00 00 00 00 16 00 00 00 02 00 00 00 c0 d7 50 8f f8 f1 5c 8d a8 95 4f 8f d
8cf5220f	8c 00 00 00 00 04 00 24 00 00 00 00 d8 e1 f1 8c 00 00 00 00 00 00 00 00 00 00 00 0
8cf5222e	00 00 f8 84 37 8d 03 00 00 00 0a 00 00 00 ff ff ff ff ff ff ff ff 00 00 00 00 01 00 0
8cf5224d	00 00 05 00 dd 6d 00 e8 03 00 00 78 00 00 00 88 13 00 00 78 00 00 00 00 00 00 00 03 0

```
.text:0006754E      movesi, [ebp+structA14]
```

```

.text:00067551                push    [ebp+structA1c]
.text:00067554                mov     [esi], eax        ; 将返回的 ipv4 相关结构
StructIPv4 赋给 StructA 偏移 0x14 位置
.text:00067556                call   _INETADDR_ADDRESS(x); 返回指向 IP 地址的指
针

```

.text:0006755B cmpdwordptr [eax+0Ch], 0 比较是否得到了 IP 地址, 因为该 IP 地址为 IPV4 地址映射而来。而 IPV6 地址为 128 位, IPv4 地址为 32 位, 为了表示由 IPV4 地址映射而来, 在将 IPv4 转成 IPv6 地址之后, 会在其前面添加标志 0xffff。

Virtual:	8dfcc044	Display
8dfcc044	00 00 00 00 00 00 00 00 00 00 00 ff ff a9 0d 0d 0d	
8dfcc063	00 00 00 00 00 00 00 00 00 00 b8 a4 e3 8d 36 1d 4b	
8dfcc082	00 00 02 00 00 00 8a c0 a8 ac 17 00 00 00 00 00 00	
8dfcc0a1	01 01 00 0c 01 fc 7c 00 00 00 00 00 00 00 00 00	

这里返回的 IP 地址指针存在了 eax 里面, 由于是 ipv4 转换而来, 因此前面 10 个字节均为 0。因此比较 eax+0C 位置的值是否为 0, 从而验证了 IP 地址是否已经设置。

```

.text:0006755F                mov ecx, [ebp+setaddrflag];
.text:00067562                setz    al
.text:00067565                mov     [ecx], al        ; 设置相应标志为 0, 表明 IP 地址已经
设置
.text:00067567                jmp     short loc_675B0
检查 StructA 偏移 0x14 处是否已经被赋值
.text:000675B0                mov esi, [esi]
.text:000675B2                test    esi, esi
.text:000675B4                jnz     short loc_675BD
该函数成功则返回 0。

```

继续 InetResolveSockAddrToLocalAddressAndAf 函数

```

.text:00067483                test    eax, eax
.text:00067485                jl      short failed; 查看 InetResolveSockAddrToAf 的返回
值
.text:00067487                cmp     byte ptr [ebp+structA18+3], 0 ; 比较刚才设置的
标志位, 该标志位表明是否设置了 IPaddress, 0 表示已经设置了 IP 地址
.text:0006748B                jz      short loc_67497

.text:00067497                mov ebx, [ebx]; 取出 StructA 偏移 0x14 位置存储的值
.text:00067499                push    edi
.text:0006749A                cmpebx, [ebp+structB14]; 比较和 StructB 偏移 0x14 存储的
值是否相同, 如果不同则为 ipv4 转换为 ipv6 地址的情况
.text:0006749D                jnz     short ipv4

```

8f47e483	85c0	test	eax, eax		
8f47e485	7c7b	jnl	tcpip!InetResolveSockAddrToLocalAddressAndAf+0xab (8f47e483)	Reg	Value
8f47e487	807d1f00	cmp	byte ptr [ebp+1Fh], 0	gs	0
8f47e48b	740a	je	tcpip!InetResolveSockAddrToLocalAddressAndAf+0x40 (8f47e487)	fs	30
8f47e48d	8b452d	mov	eax, dword ptr [ebp+28h]	es	23
8f47e490	832000	and	dword ptr [eax], 0	ds	23
8f47e493	33f6	xor	esi, esi	edi	8dfe103c
8f47e495	eb69	jmp	tcpip!InetResolveSockAddrToLocalAddressAndAf+0xa9 (8f47e493)	esi	8d5e27b8
8f47e497	8b1b	mov	ebx, dword ptr [ebx]	ebx	8d468148
8f47e499	57	push	edi	edx	0
8f47e49a	3b5d0c	cmp	ebx, dword ptr [ebp+0Ch] ss:0010.9b7db978=8d374de8	ecx	9b7db98b
8f47e49d	7521	jne	tcpip!InetResolveSockAddrToLocalAddressAndAf+0x69 (8f47e49a)	eax	0
8f47e49f	e85dca0100	call	tcpip!INETADDR_SCOPE_ID (8f49af01)	ebp	9b7db96c
8f47e4a4	50	push	eax	eip	8f47e49a
8f47e4a5	57	push	edi	cs	8
8f47e4a6	e837ca0100	call	tcpip!INETADDR_ADDRESS (8f49aee2)	efl	246
8f47e4ab	50	push	eax	esp	9b7db95c
8f47e4ac	6a00	push	0	ss	10
8f47e4ae	56	push	esi	dr0	8dfe1034
8f47e4af	53	push	ebx		

如上图，这里明显不同

Memory	
Virtual:	8d468148
8d468148	00 00 00 00 00 00 00 00 07 00 00 00 02 00 00 00 60 ea 51 8f d8 71 f9 8c a8 a5 50 8:
8d468169	00 00 00 04 00 24 00 0:
8d46818a	99 8d 00 0:
8d4681ab	00 0:

Ipv4 Struct

Memory	
Virtual:	8d374de8
8d374de8	00 00 00 00 00 00 00 00 06 00 00 00 17 00 00 00 60 ea 51 8f b8 4d 37 8d a8 a5 50 8:
8d374e09	00 00 00 10 00 38 00 0:
8d374e2a	7f 8d 00 0:
8d374e4b	00 0:
8d374e6c	41 63 70 53 41 43 50 49 5c 50 4e 50 30 43 38 30 00 00 50 00 30 00 43 00 38 00 30 0:
8d374e8d	00 00 00 05 00 04 04 41 63 70 53 34 39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0:

Ipv6 Struct

```
.text:000674C0      movesi, [esi]
.text:000674C2      call    INETADDR_ADDRESS(x); 取出 IP 地址结构的指针
.text:000674C7      moveax, [eax+0Ch]; 指向 IP 地址
.text:000674CA      push    edi
.text:000674CB      mov     [ebp+structA18], eax; 将其填充到 StructA 偏移
0x18 的位置
.text:000674CE      call    INETADDR_SCOPE_ID(x)
.text:000674CE
.text:000674D3      push    eax
.text:000674D4      lea     eax, [ebp+structA18]
.text:000674D7      push    eax
.text:000674D8      push    esi
.text:000674D9      xoresi, esi
.text:000674DB      push    esi
.text:000674DC      push    ebx
.text:000674DD      call    InetFindAndReferenceLocalAddressAf(x,x,x,x,x); 此
函数检测需要绑定的本地地址是否有效，如果有效则将其对应的结构引用计数增加 1，由于
传入的是一个无效地址，因此此函数返回为 0，表示失败；如果该函数成功，则返回
StructLocalAddr，从函数名上来看应该是记录了绑定在本地地址上的一个结构
```


Offset: 8f46d543				Previous	Next	Customize...
8f46d478	ff750c	push	dword ptr [ebp+0Ch]		Reg	Value
8f46d47b	ff7508	push	dword ptr [ebp+8]		gs	0
8f46d47e	e88b000000	call	tcpip!InetResolveSockAddrToAf (8f46d50e)		fs	30
8f46d483	85c0	test	eax, eax		es	23
8f46d485	7c7b	jl	tcpip!InetResolveSockAddrToLocalAddressAndAf+0xab (8f46d502)		ds	23
8f46d487	807d1f00	cmp	byte ptr [ebp+1Fh], 0		edi	8dfcc03c
8f46d48b	740a	je	tcpip!InetResolveSockAddrToLocalAddressAndAf+0x40 (8f46d497)		esi	8d5e4bc0
8f46d48d	8b4528	mov	eax, dword ptr [ebp+28h]		ebx	8d378470
8f46d490	832000	and	dword ptr [eax], 0		edx	0
8f46d493	33f6	xor	esi, esi		ecx	9254b98b
8f46d495	ebb9	jmp	tcpip!InetResolveSockAddrToLocalAddressAndAf+0xa9 (8f46d500)		eax	0
8f46d497	8b1b	mov	ebx, dword ptr [ebx]		ebp	9254b96c
8f46d499	57	push	edi		esp	9254b95c
8f46d49a	3b5d0c	cmp	ebx, dword ptr [ebp+0Ch] ss:0010:9254b978=8d3c16b8		ss	10
8f46d49d	7521	jne	tcpip!InetResolveSockAddrToLocalAddressAndAf+0x69 (8f46d4c0)		dr0	0
8f46d49f	e85dca0100	call	tcpip!INETADDR_SCOPE_ID (8f489f01)		dr1	0
8f46d4a4	50	push	eax		dr2	0
8f46d4a5	57	push	edi		dr3	0
8f46d4a6	e837ca0100	call	tcpip!INETADDR_ADDRESS (8f489ee2)			
8f46d4ab	50	push	eax			
8f46d4ac	6a00	push	0			
8f46d4ae	56	push	esi			
8f46d4af	53	push	ebx			
8f46d4b0	e806afffff	call	tcpip!InetFindAndReferenceLocalAddressAf (8f4683bb)			

```
.text:000674E2      movcx, [ebp+structA10]
.text:000674E5      mov     [ecx], eax; 将返回的 StructLocalAddr 的结构指针
                  赋给 StructA 偏移 0x10 的位置, 但此时为 0
.text:000674E7      cmpeax, esi; 检测是否为 0
.text:000674E9      jnz     short loc_67500; 返回 0 表示失败, 所以这里不跳
                  转
.text:000674EB      moveax, [ebp+structA14]; 将刚存储的 StructIPV4 指针取出
                  来 (因为刚才已经将其引用计数增 1, 此时需要递减引用计数)
.text:000674EE      moveax, [eax]
.text:000674F0      movesi, STATUS_INVALID_ADDRESS_COMPONENT; 设置返回
                  值
.text:000674F5      cmpeax, [ebp+structB14]
.text:000674F8      jz      short loc_67500; 前文述 StructA 偏移 0x14 处的的
                  值和 StructB 偏移 0x14 处的值不同, 因此此处不跳转
.text:000674FA      push    eax
.text:000674FB      call    InetDereferenceAf(x); 递减 StructIPV4 的引用计数,
                  第一次递减。
```

InetDereferenceAf 函数如下:

```
.text:0009466C      movedi, edi
.text:0009466E      push    ebp
.text:0009466F      movebp, esp
.text:00094671      moveax, [ebp+arg_0]
.text:00094674      push    esi
.text:00094675      lea     ecx, [eax+8]
.text:00094678      or      esi, 0FFFFFFFh
.text:0009467B      lock xadd [ecx], esi
.text:0009467F      decesi          ; 引用计数减 1
.text:00094680      jnz     short loc_94688; 如果引用计数为 0 了, 则将相应
                  的内存释放
.text:00094682      push    eax
.text:00094683      call    InetCleanupAf(x)
之后便从 InetResolveSockAddrToLocalAddressAndAf 返回, 继续
UdpBindEndpointRequestInspectComplete 函数:
.text:00052B16      mov     [ebp+retvalue], eax; 获得返回值, 返回值小于
```

0 的情况则表示失败

```
.text:00052B19      test     eax, eax
.text:00052B1B      jge      short success; 这里不跳转

.text:00052B55      movesi, [ebp+structA] ;StructA 指针赋值给 esi
.text:00052B58      lea      edx, [ebp+var_C]
.text:00052B5B      movecx, ebx
.text:00052B5D      call     ds:KeAcquireInStackQueuedSpinLock(x,x)
最后跳转至此
.text:00052E7B      push     [ebp+retntvalue]
.text:00052E7E      push     esi; StructA
.text:00052E7F      call     UdpBindEndpointInspectComplete(x,x)
```

在 UdpBindEndpointInspectComplete 函数中同样有这么一段代码

```
.text:00052E9B      movedi, [ebp+StructA]
.text:00052E9E      movesi, [edi+4] ;StructB
...
.text:00052F9A      moveax, [edi+14h]
.text:00052F9D      cmpeax, [esi+14h]
.text:00052FA0      jz       short loc_52FA8
.text:00052FA2      push     eax
.text:00052FA3      call     InetDereferenceAf(x); 递减 StructIPV4 的引用计数，
第二次递减。
```

同样是比较 StructA 和 StructB 偏移 0x14 处的值，如果不同（也就是 Ipv4 扩展为 ipv6 的情况）则再一次减小引用计数。这里就出问题了，前面只有一次增加引用计数，却有两次减少引用计数。所以导致程序每执行一次 bind 函数，对应的 Ipv4 Struct 的引用计数则减 1，当减少到 0 的时候，根据 InetDereferenceAf 代码，其对应的一块内存将被释放。

具体地，当引用计数减少到 1 的时候，下一次再执行 bind 操作的时候则会由于第二次引用计数（UdpBindEndpointInspectComplete）调用 InetDereferenceAf 减少引用计数而最终释放掉相应的内存。如图

```

8f469f91 50      push    eax
8f469f92 ff7714   push    dword ptr [edi+14h]
8f469f95 e8a1f80000 call    tcpip!InetDereferenceLocalAddressAf (8f47983b)
8f469f9a 8b4714   mov     eax,dword ptr [edi+14h]
8f469f9d 3b4614   cmp     eax,dword ptr [esi+14h]
8f469fa0 7406     je      tcpip!UdpBindEndpointInspectComplete+0x118 (8f469fa8)
8f469fa2 50      push    eax
8f469fa3 e8c4160400 call    tcpip!InetDereferenceAf (8f4ab66c)
8f469fa8 8b4704   mov     eax,dword ptr [edi+4]
8f469fab 8d4808   lea     ecx,[eax+8]
8f469fae 83caff   or      edx,0FFFFFFFh
8f469fb1 f00fc111 lock xadd dword ptr [ecx],edx
8f469fb5 7506     jne     tcpip!UdpBindEndpointInspectComplete+0x12d (8f469fbd)
8f469fb7 50      push    eax
8f469fb8 e81d60ffff call    tcpip!UdpCleanupEndpoint (8f45ffda)
8f469fbd 57      push    edi
8f469fbe e8f6ae0300 call    tcpip!FsbFree (8f4a4eb9)
8f469fc3 5f      pop     edi

```

Memory

Virtual	8d468148
8d468148	00 00 00 00 00 00 00 00 01 00 00 00 02 00 00 00 60 ea 51 8f d8 71 f9 8c a8 a5 5
8d468169	00 00 00 00 04 00 24 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
8d46818a	99 8d 00 0
8d4681ab	00 0
8d4681cc	44 65 76 e9 01 00 00 00 00 00 00 00 00 00 00 00 19 00 00 02 01 00 00 00 00 00 0
8d4681ed	00 00 00 d8 54 5b 8c 00 83 46 8d 28 19 48 8d 00 00 00 00 00 00 00 00 00 00 00 00 0

再一次执行则会由于引用计数归 0 而释放掉一块内存

```

8f4d5945 8b4040   mov     eax,dword ptr [eax+40h]
8f4d5948 85c0     test    eax,eax
8f4d594a 7409     je      tcpip!TcpCleanupNsiObject+0x18 (8f4d5955)
8f4d594c 6a00     push    0
8f4d594e 50      push    eax
8f4d594f ff154c53508f call    dword ptr [tcpip!_imp_ExFreePoolWithTag (8f50534c)] d:
8f4d5955 5d      pop     ebp
8f4d5956 c20400   ret     4
8f4d5959 90      nop
8f4d595a 90      nop
8f4d595b 90      nop
8f4d595c 90      nop
8f4d595d 90      nop
8f4d595e 90      nop

```

内存释放

而再一次执行 bind 函数的时候，先增加一次引用计数，在其后 InetResolveSockAddrToLocalAddressAndAf 调用的 InetDereferenceAf 函数中会递减引用计数为 0，再一次释放内存。导致 Double-free 错误。

calls

Raw	Args	Func info	Source	Address	Headings	Hex	Disassembly	Registers
8f4d5921	686ae5108	push	offset tcpip!UdpInetTransport (8f51ea60)	8f4d5921	686ae5108	push	offset tcpip!UdpInetTransport (8f51ea60)	reg
8f4d5922	e817ef0200	call	tcpip!InetStopNsiObject (8f504042)	8f4d5922	e817ef0200	call	tcpip!InetStopNsiObject (8f504042)	rs
8f4d5923	6a00	push	0	8f4d5923	6a00	push	0	rs
8f4d5924	58	pop	eax	8f4d5924	58	pop	eax	rs
8f4d5925	b958ae5108	mov	ecx,offset tcpip!UdpStartedModules (8f51ea58)	8f4d5925	b958ae5108	mov	ecx,offset tcpip!UdpStartedModules (8f51ea58)	ds
8f4d5926	f00fc101	lock xadd	dword ptr [ecx],eax	8f4d5926	f00fc101	lock xadd	dword ptr [ecx],eax	edi
8f4d5927	c3	ret		8f4d5927	c3	ret		edi
8f4d5928	90	nop		8f4d5928	90	nop		edi
8f4d5929	90	nop		8f4d5929	90	nop		edi
8f4d592a	90	nop		8f4d592a	90	nop		edi
8f4d592b	90	nop		8f4d592b	90	nop		edi
8f4d592c	90	nop		8f4d592c	90	nop		edi
8f4d592d	90	nop		8f4d592d	90	nop		edi
8f4d592e	90	nop		8f4d592e	90	nop		edi
8f4d592f	90	nop		8f4d592f	90	nop		edi
8f4d5930	90	nop		8f4d5930	90	nop		edi
8f4d5931	55	mov	edi,edi	8f4d5931	55	mov	edi,edi	cs
8f4d5932	8bec	mov	ebp,esp	8f4d5932	8bec	mov	ebp,esp	esp
8f4d5933	8b4508	mov	eax,dword ptr [ebp+8]	8f4d5933	8b4508	mov	eax,dword ptr [ebp+8]	esp
8f4d5934	8b4040	mov	eax,dword ptr [eax+40h]	8f4d5934	8b4040	mov	eax,dword ptr [eax+40h]	cs
8f4d5935	85c0	test	eax,eax	8f4d5935	85c0	test	eax,eax	cs
8f4d5936	7409	je	tcpip!TcpCleanupNsiObject+0x18 (8f4d5955)	8f4d5936	7409	je	tcpip!TcpCleanupNsiObject+0x18 (8f4d5955)	cs
8f4d5937	6a00	push	0	8f4d5937	6a00	push	0	cs
8f4d5938	50	push	eax	8f4d5938	50	push	eax	cs
8f4d5939	ff154c53508f	call	dword ptr [tcpip!_imp_ExFreePoolWithTag (8f50534c)] d:	8f4d5939	ff154c53508f	call	dword ptr [tcpip!_imp_ExFreePoolWithTag (8f50534c)] d:	cs
8f4d593a	5d	pop	ebp	8f4d593a	5d	pop	ebp	cs
8f4d593b	c20400	ret	4	8f4d593b	c20400	ret	4	cs
8f4d593c	90	nop		8f4d593c	90	nop		cs
8f4d593d	90	nop		8f4d593d	90	nop		cs
8f4d593e	90	nop		8f4d593e	90	nop		cs
8f4d593f	90	nop		8f4d593f	90	nop		cs
8f4d5940	90	nop		8f4d5940	90	nop		cs
8f4d5941	90	nop		8f4d5941	90	nop		cs
8f4d5942	90	nop		8f4d5942	90	nop		cs
8f4d5943	90	nop		8f4d5943	90	nop		cs
8f4d5944	90	nop		8f4d5944	90	nop		cs
8f4d5945	90	nop		8f4d5945	90	nop		cs
8f4d5946	90	nop		8f4d5946	90	nop		cs
8f4d5947	90	nop		8f4d5947	90	nop		cs
8f4d5948	90	nop		8f4d5948	90	nop		cs
8f4d5949	90	nop		8f4d5949	90	nop		cs
8f4d594a	90	nop		8f4d594a	90	nop		cs
8f4d594b	90	nop		8f4d594b	90	nop		cs
8f4d594c	90	nop		8f4d594c	90	nop		cs
8f4d594d	90	nop		8f4d594d	90	nop		cs
8f4d594e	90	nop		8f4d594e	90	nop		cs
8f4d594f	90	nop		8f4d594f	90	nop		cs
8f4d5950	90	nop		8f4d5950	90	nop		cs
8f4d5951	90	nop		8f4d5951	90	nop		cs
8f4d5952	90	nop		8f4d5952	90	nop		cs
8f4d5953	90	nop		8f4d5953	90	nop		cs
8f4d5954	90	nop		8f4d5954	90	nop		cs
8f4d5955	90	nop		8f4d5955	90	nop		cs
8f4d5956	90	nop		8f4d5956	90	nop		cs
8f4d5957	90	nop		8f4d5957	90	nop		cs
8f4d5958	90	nop		8f4d5958	90	nop		cs
8f4d5959	90	nop		8f4d5959	90	nop		cs
8f4d595a	90	nop		8f4d595a	90	nop		cs
8f4d595b	90	nop		8f4d595b	90	nop		cs
8f4d595c	90	nop		8f4d595c	90	nop		cs
8f4d595d	90	nop		8f4d595d	90	nop		cs
8f4d595e	90	nop		8f4d595e	90	nop		cs
8f4d595f	90	nop		8f4d595f	90	nop		cs
8f4d5960	90	nop		8f4d5960	90	nop		cs
8f4d5961	90	nop		8f4d5961	90	nop		cs
8f4d5962	90	nop		8f4d5962	90	nop		cs
8f4d5963	90	nop		8f4d5963	90	nop		cs
8f4d5964	90	nop		8f4d5964	90	nop		cs
8f4d5965	90	nop		8f4d5965	90	nop		cs
8f4d5966	90	nop		8f4d5966	90	nop		cs
8f4d5967	90	nop		8f4d5967	90	nop		cs
8f4d5968	90	nop		8f4d5968	90	nop		cs
8f4d5969	90	nop		8f4d5969	90	nop		cs
8f4d596a	90	nop		8f4d596a	90	nop		cs
8f4d596b	90	nop		8f4d596b	90	nop		cs
8f4d596c	90	nop		8f4d596c	90	nop		cs
8f4d596d	90	nop		8f4d596d	90	nop		cs
8f4d596e	90	nop		8f4d596e	90	nop		cs
8f4d596f	90	nop		8f4d596f	90	nop		cs
8f4d5970	90	nop		8f4d5970	90	nop		cs
8f4d5971	90	nop		8f4d5971	90	nop		cs
8f4d5972	90	nop		8f4d5972	90	nop		cs
8f4d5973	90	nop		8f4d5973	90	nop		cs
8f4d5974	90	nop		8f4d5974	90	nop		cs
8f4d5975	90	nop		8f4d5975	90	nop		cs
8f4d5976	90	nop		8f4d5976	90	nop		cs