Secunia

Stay Secure

SA37347 / CVE-2009-3676

Released by Secunia

17 November, 2009

8 pages

Table of Contents

Terms and Conditions:
=====================

This Binary Analysis, including any PoC, pcap, exploit, or other support files (hereinafter referred to as BA), are intended for defensive purposes only.

The BA may not be used to orchestrate attack tools. It may, however, be utilised to verify signatures and rules created. To a certain extent, the information may also be used to verify that patches are properly applied if you are the legal owner of the system or have been legally authorised to test the patch level.

The BA may NOT be redistributed outside the legal entity for which it was purchased, nor may it be republished in any way.

Any employee working with this BA must be aware of the legal aspects related to BA.

Secunia is not in any way liable for any damages, business impact, or legal issues caused or related directly or indirectly to the purchase, use, interpretation, misappropriation, misinterpretation, or derivative work of the BA.

All other legal issues regarding the Secunia Binary Analysis Service are governed by the most recent version of the Secunia Terms and Conditions. A copy of the T&C may be obtained from the following URL:

https://ca.secunia.com/terms_and_conditions/

Introduction:
==========

An error in Microsoft Windows can be exploited to hang an affected system via a specially crafted SMB response.

Technical Details:
==========

The establishment of an SMB session is implemented in kernel space in Windows. An SMB packet is received in a TCP connection to port 445 and is sent inside a network packet starting with four bytes defining the SMB packet's size.

An error exists in the SMB client-side implementation when processing SMB packets having a size larger than the number of following bytes. This can be exploited to trigger the execution of an infinite loop and hang an affected system by tricking a user into connecting to a malicious SMB server.

SmbWskReceiveEvent() in mrxsmb.sys is called from afd.sys when an SMB packet is received inside an established outgoing SMB connection. The function starts by extracting the 32-bit number placed before the received SMB packet in big-endian order. The last 17 bits of the extracted value are considered the SMB packet size.

```
.text:00015175 ; __stdcall SmbWskReceiveEvent(x, x, x, x, x)
.text:00015175 _SmbWskReceiveEvent@20 proc near        ; DATA XREF: .data:_SmbWskConnectionDispatcho
.text:00015175
.text:00015175 var_424         = dword ptr -424h
.text:00015175 var_420         = dword ptr -420h
.text:00015175 packet_size     = dword ptr -41Ch
.text:00015175 var_418         = dword ptr -418h
.text:00015175 var_414         = dword ptr -414h
.text:00015175 bytesout        = dword ptr -410h
.text:00015175 first4bytes     = dword ptr -40Ch
.text:00015175 processedbytes  = dword ptr -408h
.text:00015175 packet_data     = byte ptr -404h
.text:00015175 var_4           = dword ptr -4
.text:00015175 arg_0           = dword ptr  8
.text:00015175 arg_8           = dword ptr  10h
.text:00015175 availablebytes  = dword ptr  14h
.text:00015175 arg_10          = dword ptr  18h
.text:00015175
...
.text:0001527C                 cmp     edx, 4          ; number of available bytes
.text:0001527F                 jb      loc_15562
.text:00015285                 push    ebx
.text:00015286                 push    4
.text:00015288                 push    [ebp+processedbytes]
.text:0001528E                 lea     eax, [ebp+first4bytes]
.text:00015294                 push    eax
.text:00015295                 push    [ebp+var_414]
.text:0001529B                 call    _SmbWskCopyIndicationToBufferEx@20 ; SmbWskCopyIndicationToBufferEx(x,x,x,x,x)
.text:000152A0                 test    eax, eax
.text:000152A2                 jl      @error1
.text:000152A8                 add     [ebp+processedbytes], 4
...
.text:00015324                 cmp     byte ptr [esi+89h], 0
.text:0001532B                 mov     eax, [ebp+first4bytes]
.text:00015331                 bswap   eax             ; from network order to LE
.text:00015333                 mov     [ebp+first4bytes], eax
.text:00015339                 jnz     short loc_15345
.text:0001533B                 and     [ebp+first4bytes], 1FFFFh ; last 17 bits
```

If the number of remaining bytes in the received TCP packet is below the SMB packet size, all available bytes are extracted into a local stack buffer.

```
.text:000153A4                mov     eax, [ebp+availablebytes]
.text:000153A7                sub     eax, [ebp+processedbytes]
.text:000153AD                cmp     [ebp+first4bytes], eax
.text:000153B3                ja      loc_15475
...
.text:00015475 loc_15475:                             ; CODE XREF: SmbWskReceiveEvent(x,x,x,x,x)+229j
.text:00015475                                        ; SmbWskReceiveEvent(x,x,x,x,x)+23Ej
.text:00015475                mov     eax, [ebp+availablebytes]
.text:00015478                sub     eax, [ebp+processedbytes]
.text:0001547E                cmp     eax, edi        ; 400h
.text:00015480                ja      short loc_15484
.text:00015482                mov     edi, eax
.text:00015484
.text:00015484 loc_15484:                             ; CODE XREF: SmbWskReceiveEvent(x,x,x,x,x)+30Bj
.text:00015484                push    0
.text:00015486                push    edi
.text:00015487                push    [ebp+processedbytes]
.text:0001548D                lea     eax, [ebp+packet_data]
.text:00015493                push    eax
.text:00015494                push    [ebp+var_414]
.text:0001549A                call    _SmbWskCopyIndicationToBufferEx@20 ; copy available bytes
.text:0001549F                test    eax, eax
.text:000154A1                jl      @error1
```

VctIndReceive() is next called in order to communicate the receive event and copy the network data. If the number of network bytes is below the SMB size, an error indicating that more processing is required (STATUS_MORE_PROCESSING_REQUIRED) is returned (error return path not shown in disassembly).

```
.text:000154A7                mov     eax, [esi+98h]
.text:000154AD                lea     ecx, [ebp+packet_size]
.text:000154B3                push    ecx
.text:000154B4                lea     ecx, [ebp+var_418]
.text:000154BA                push    ecx
.text:000154BB                lea     ecx, [ebp+var_420]
.text:000154C1                push    ecx
.text:000154C2                lea     ecx, [ebp+packet_data]
.text:000154C8                push    ecx
.text:000154C9                lea     ecx, [ebp+bytesout]
.text:000154CF                push    ecx
.text:000154D0                push    [ebp+first4bytes]
.text:000154D6                push    edi
.text:000154D7                push    0
.text:000154D9                push    esi
.text:000154DA                push    dword ptr [esi+94h]
.text:000154E0                call    dword ptr [eax+8] ; call _VctIndReceive@40
.text:000154E3                mov     ebx, eax        ; STATUS_MORE_PROCESSING_REQUIRED returned
.text:000154E3                                        ; if not enough data was received
```

If STATUS_MORE_PROCESSING_REQUIRED is returned, SmbWskReceive() is called in order to receive the remaining data.

```
.text:00015540                cmp     ebx, STATUS_MORE_PROCESSING_REQUIRED
.text:00015546                jz      loc_155D5
...
.text:000155D5                push    [ebp+var_418]
.text:000155DB                push    [ebp+packet_size]
.text:000155E1                push    [ebp+var_420]
.text:000155E7                push    esi
.text:000155E8                call    _SmbWskReceive@16 ; SmbWskReceive(x,x,x,x)
```

SmbWskReceive() calls afd.sys!WskProAPIReceive() in order to perform an asynchronous receive operation. SmbWskReceiveComplete() is registered as a callback function.

```
.text:0001B634 ; __stdcall SmbWskReceive(x, x, x, x)
.text:0001B634 _SmbWskReceive@16 proc near             ; CODE XREF: SmbWskReceiveEvent(x,x,x,x,x)+473p
.text:0001B634                                         ; RxCeReceive(x,x,x,x,x)+42p
.text:0001B634
.text:0001B634 var_C           = dword ptr -0Ch
.text:0001B634 var_8           = dword ptr -8
.text:0001B634 var_4           = dword ptr -4
.text:0001B634 arg_0           = dword ptr  8
.text:0001B634 arg_4           = dword ptr  0Ch
.text:0001B634 packet_size     = dword ptr  10h
.text:0001B634 arg_C           = dword ptr  14h
.text:0001B634
...
.text:0001B69B                 mov     ecx, [ebp+arg_C]
.text:0001B69E                 mov     [eax+4], ecx
.text:0001B6A1                 mov     [eax], esi
.text:0001B6A3                 mov     ecx, [edi+60h]
.text:0001B6A6                 sub     ecx, 24h
.text:0001B6A9                 push    edi
.text:0001B6AA                 mov     [ecx+20h], eax
.text:0001B6AD                 push    2
.text:0001B6AF                 lea     eax, [ebp+var_C]
.text:0001B6B2                 push    eax
.text:0001B6B3                 mov     dword ptr [ecx+1Ch], offset _SmbWskReceiveComplete@12 ; SmbWskReceiveComplete(x,x,x)
.text:0001B6BA                 mov     byte ptr [ecx+3], 0E0h
.text:0001B6BE                 push    dword ptr [esi+34h]
.text:0001B6C1                 call    dword ptr [ebx+1Ch] ; call afd!WskProAPIReceive
```

Execution continues in SmbWskReceiveEvent() with an immediate function return, ending the mrxsmb.sys processing of the current packet (not shown in disassembly).

SmbWskReceiveComplete() is later called when the TCP connection is terminated by the remote SMB server and in turn calls VctIndDataReady() in order to copy potentially present data.

```
.text:0001B701 ; int __stdcall SmbWskReceiveComplete(int, int, PVOID P)
.text:0001B701 _SmbWskReceiveComplete@12 proc near     ; DATA XREF: SmbWskReceive(x,x,x,x)+7Fo
.text:0001B701
.text:0001B701 arg_4           = dword ptr  0Ch
.text:0001B701 P               = dword ptr  10h
.text:0001B701
...
.text:0001B706                 mov     eax, [ebp+arg_4]
.text:0001B709                 mov     ecx, [eax+1Ch]  ; number of available bytes
...
.text:0001B717                 mov     [ebp+P], ecx    ; save
...
.text:0001B751                 mov     eax, [esi+98h]
.text:0001B757                 push    edi
.text:0001B758                 push    [ebp+P]         ; number of available bytes
.text:0001B75B                 push    dword ptr [ebx+4]
.text:0001B75E                 push    esi
.text:0001B75F                 push    dword ptr [esi+94h]
.text:0001B765                 call    dword ptr [eax+18h] ; call _VctIndDataReady@20
```

If the number of additional bytes sent during the connection (total bytes - 4) is smaller than the number of required bytes, VctPerformReceive() is called.

```
.text:0001B068 ; int __stdcall VctIndDataReady(int, int, size_t, int numrecv, int)
.text:0001B068 _VctIndDataReady@20 proc near           ; CODE XREF: VctPerformReceive(x)+35p
.text:0001B068
...
.text:0001B276                 mov     eax, [esi+0FCh]
.text:0001B27C                 mov     ecx, [edi+4]
.text:0001B27F                 mov     ebx, [edi+8]
.text:0001B282                 add     ebx, [ebp+numrecv] ; total number of bytes
.text:0001B285                 mov     [ebp+var_8], eax
.text:0001B288                 mov     eax, [edi]      ; required bytes
.text:0001B28A                 mov     [ebp+numreq], eax
.text:0001B28D                 mov     [ebp+var_14], ecx ; zero
...
.text:0001B2BC                 mov     eax, [ebp+var_14]
.text:0001B2BF                 cmp     eax, ebx        ; 0
.text:0001B2C1                 jb      short loc_1B2D0 ; bytes received
...
.text:0001B2DC                 mov     eax, 80h        ; test
.text:0001B2E1                 mov     [ebp+availbytes], ebx
.text:0001B2E4                 cmp     [ebp+numreq], eax
.text:0001B2E7                 ja      short loc_1B2F4
.text:0001B2E9                 mov     eax, [ebp+numreq]
.text:0001B2EC                 test    eax, eax
.text:0001B2EE                 jbe     loc_1B223
.text:0001B2F4
.text:0001B2F4 loc_1B2F4:                              ; CODE XREF: VctIndDataReady(x,x,x,x,x)+27Fj
.text:0001B2F4                                         ; VctIndDataReady(x,x,x,x,x)+349j
.text:0001B2F4                 cmp     ebx, eax        ; avail bytes < min(80h, reqbytes);
.text:0001B2F6                 jb      loc_1B450
...
.text:0001B1E8                 mov     [ebp+morebytesrequired], 1
...
.text:0001B234                 cmp     [ebp+morebytesrequired], 0
.text:0001B238                 jz      @return
...
.text:0001B258                 call    _VctPerformReceive@4 ; VctPerformReceive(x)
.text:0001B25D                 jmp     @return
```

VctPerformReceive() calls RxCeReceive(), which in turns calls SmbWskReceive() if the connection was terminated (not shown in disassembly).

SmbWskReceive() calls afd.sys!WskProAPIReceive() and returns, again registering SmbWskReceiveComplete() as a callback function (included in previous disassembly).

Due the current connection being already terminated, SmbWskReceiveComplete() is immediately called via TcpCompleteClientReceiveRequest() from tcpip.sys, leading to another SmbWskReceive() call. As each SmbWskReceive() call requests more data, TcpCompleteClientReceiveRequest() is called an infinite number of times.

```
.text:00061614 ; __stdcall TcpFlushTcbDelivery(x, x, x)
.text:00061614 _TcpFlushTcbDelivery@12 proc near       ; CODE XREF: TcpFlushRequestReceive(x,x)+17p
.text:00061614                                         ; TcpFlushRequestReceive(x,x)+2Ep
.text:00061614
...
.text:000616B5 @deliver:                               ; CODE XREF: TcpFlushTcbDelivery(x,x,x)+237j
.text:000616B5                 mov     eax, [ebx+4]
.text:000616B8                 mov     [ebp+var_34], eax
.text:000616BB                 xor     ecx, ecx
...
.text:00061800                 push    esi
.text:00061801                 push    [ebp+var_38]
.text:00061804                 push    edi
.text:00061805                 call    _TcpCompleteClientReceiveRequest@12 ; call multiple other functions,
.text:00061805                                         ; ending in SmbWskREceiveComplete()
...
.text:00061847                 cmp     dword ptr [ebx+4], 0
.text:0006184B                 jnz     @deliver  ; infinite loop
```

Exploitation:
==========

The vulnerability can be exploited to hang an affected system due to the execution of the infinite loop occuring in kernel space. A user has to be tricked into connecting to a malicious SMB server manually or via an HTML page including e.g. an <iframe> redirecting the browser to an SMB resource.

Secunia has developed a PoC, which is available to customers via the BA customer web interface.

Characteristics:
==========

Detection:
Look for packets received in outgoing SMB connections (to port 445), specifying a size larger than the total number of received bytes for the current SMB packet - 4 in the NetBIOS header. Note that multiple SMB packets can be sent within a single TCP response, each packet being separately interpreted. In such a sequence, the next packet starts after the size specified in the NetBIOS header of the current SMB packet.

Verification:
Listen for TCP packets on port 445, responding with "\x00\x00\x00\x01" to an incoming request and closing the connection. An affected system hangs if an SMB connection to the server is attempted (e.g. open \\SERVER_IP\).

Identification:
mrxsmb.sys version 6.1.7600.16385 is confirmed to be vulnerable, but other versions may also be affected. The default installation location is "%WinDir%\System32\drivers\".

Tested Versions:
==========

The vulnerability was analysed on Windows 7 with mrxsmb.sys version 6.1.7600.16385.

Fixed Versions:
==========

The vulnerability is currently unpatched.

References:
==========

SA37347:
http://secunia.com/advisories/37347/

CVE-2009-3676:
http://secunia.com/advisories/cve_reference/CVE-2009-3676/

Microsoft:
http://www.microsoft.com/technet/security/advisory/977544.mspx

Laurent Gaffié:
http://g-laurent.blogspot.com/2009/11/windows-7-server-2008r2-remote-kernel.html