



## SMB2 Vulnerability Filter Project

Upon receipt of the notification of a major (personal ranking) vulnerability to Microsoft products supporting SMBv2, I created a Wireshark filter to catch offending packets and tested that filter against good packets and malformed packets.

### The SMB2 Vulnerability

SMBv2 was released with Vista and does provide much faster data transfer rates than the pathetic SMBv1 (it's still no rocket ship though). SMBv2 is enabled by default in Vista, Server 2008 and Windows 7. The alert is defined at <http://seclists.org/fulldisclosure/2009/Sep/0039.html>.

The vulnerability defines the offending character “&” placed in the Process ID High field in an SMB Negotiate Protocol Request packet. This is a 2-byte field and the value should be 0x0000. Please see the above reference for more information on this reported vulnerability. Note that I did not assess the validity of the vulnerability – I focused on finding malformed SMB Negotiate Protocol Request Packets with Wireshark.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.88	192.168.0.99	SMB	Negotiate Protocol Request
2	0.000465	192.168.0.99	192.168.0.88	SMB	Negotiate Protocol Response

  

```
Server Component: SMB
SMB Command: Negotiate Protocol (0x72)
NT Status: STATUS_SUCCESS (0x00000000)
  Flags: 0x18
    0... .... = Request/Response: Message is a request to the server
    .0.. .... = Notify: Notify client only on open
    ..0. .... = Oplocks: OpLock not requested/granted
    ...1 .... = Canonicalized Pathnames: Pathnames are canonicalized
    .... 1... = Case Sensitivity: Path names are caseless
    .... ..0. = Receive Buffer Posted: Receive buffer has not been posted
    .... ...0 = Lock and Read: Lock&Read, Write&Unlock are not supported
  Flags2: 0xc853
  Process ID High: 0
  Signature: 0000000000000000
  Reserved: 0000
  Tree ID: 0
  Process ID: 65279
  User ID: 0
```

Figure 1: The Process ID High field in an SMB Negotiate Protocol Request packet.



---

## Building the Filter

Opening my smb\_protocol-request-reply.pcap trace file (available at <http://www.screencast.com/t/STY3DwqdwMAI>), I began using the right-click method to prepare a filter. I focused on the following fields:

- smb.cmd (0x72 is an SMB Negotiate Protocol command)
- smb.flags.response (a bit value of 0 indicates this is a request packet)
- smb.pid.high (a value other than 0x0000 would be considered 'abnormal')

The final filter is shown below.

```
((smb.cmd == 0x72) && (smb.flags.response == 0)) && !(smb.pid.high == 0)
```

---

## Testing the Filter

I only had legal SMB Negotiate Protocol request packets in my trace file set and had to create illegal packets. I ran the filter against smb\_protocol-request-reply.pcap and no packets were displayed. This is a good result.

Next, I opened up another trace file (that I'd saved with a .cap extension) with more SMB packets using Colasoft's Packet Builder (available for free at [http://www.colasoft.com/download/products/packet\\_builder.php](http://www.colasoft.com/download/products/packet_builder.php)). I edited the Process ID High field values in the requests and sent the packets onto the network while capturing them with Wireshark. The three malformed SMB Negotiate Protocol Request Packets successfully matched my filter.

---

## Resources

Trace File (good SMB packets): smb\_protocol-request-reply.pcap  
<http://www.screencast.com/t/STY3DwqdwMAI>

Trace File (bad SMB packets): smb\_protocol-request-reply4filtertest.pcap  
<http://www.screencast.com/t/iX5CdprK6OAg>

---

## Note:

Although the vulnerability disclosure mentions the value "&" specifically as the issue, I wanted a more generic filter in case this vulnerability turned out (a) to be valid and (b) would be valid with other byte values in the Process ID High field.