Packet Carving with SMB and SMB2

By Chris SandersNovember 2, 20118 commentsPacket Analysiscarving, extracting, file transfer, microsoft, network forensics, packets, server message block, smb, smb2, tcpdump, windows networking, wireshark

One of the more useful network forensic skills is the ability to extract files from packet captures. This process, known as packet data carving, is crucial when you want to analyze malware or other artifacts of compromise that are transferred across the network. That said, packet data carving has varying degrees of difficulty depending on the type of traffic you are attempting to extract data from. Carving files from simple protocols like HTTP and FTP is something that can be done in a matter of minutes and is usually cut and dry enough that it can be done in an automated fashion with tools like Foremost and Network Miner.

There are articles all over the Internet about carving files from simple protocols so I won't rehash those. Instead, I want to take a look at a two more complex protocols that are extremely common in production networks. Server Message Block (SMB) is the application-layer protocol that Microsoft operating systems use for file sharing and communication between networked devices. If you live on a Microsoft network (or a Unix network that utilizes SAMBA) then you are a user of SMB or SMB2, depending on your operating system version. In this article I'm going to discuss the art of carving files from SMB and SMB2 traffic. If you want to follow along you'll need to download a copy of Wireshark (http://www.wireshark.org) and your favorite hex editor. I've used Cygnus Hex Editor (http://www.softcircuits.com/cygnus/fe/) for the purpose of this article since it's simple and a free version exists.

�

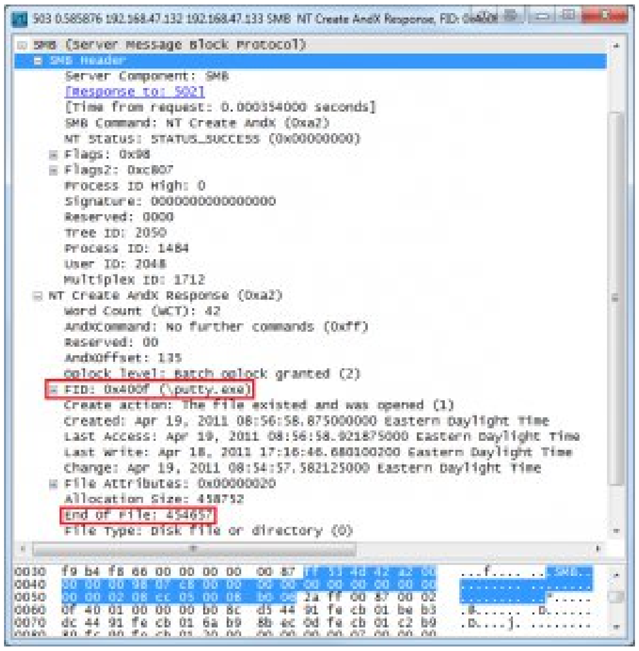
Carving Files from SMB Packets

�

The first version of SMB is in use on all modern Microsoft operating systems prior to Windows Vista. In order to setup a packet capture for this scenario I took two Windows XP SP3 virtual machines running on VMWare Workstation and placed them in the same network. Once they were able to communicate with each other I setup a shared folder on one host (192.168.47.132) that is acting as the server. I then fired up Wireshark and began capturing packets as I copied an executable file from the client (192.168.47.133) to the servers shared folder. The resulting packet capture is called **smb_puttyexe_xfer.pcap**.

If you've never looked at SMB traffic then don't get scared by all the different types of SMB packets in the capture, we will only be looking at a few of them. This article isn't meant to be an exhaustive reference on each and every type of SMB packet (there are over a hundred of them), so if you want the gory details then take a look at the references at this end of this article.

In order to carve the file out of these packets we have to find some basic information about it. Before and after transferring a file to a server the client will attempt to open the file in order to see if it exists. This is done with an SMB NT Create AndX Request packet. The response from the server to this is an SMB NT Create AndX Response, which contains the name, extension, and size of the file being transferred. This is everything we need to get started. You can filter for Create AndX Response packets in Wireshark with the filter *(smb.cmd == 0xa2) && (smb.flags.response == 1)*. If we examine one of those requests that occur after the file has been transferred, we can identify that the file being transferred is putty.exe and its file size is 454,657 bytes. We will use this information later.



```
503 0.585876 192.168.47.132 192.168.47.133 SMB NT Create AndX Response, FID: 0x400f
⊟ SMB (Server Message Block Protocol)
  ⊟ SMB Header
      Server Component: SMB
      [Response to: 502]
      [Time from request: 0.000354000 seconds]
      SMB Command: NT Create AndX (0xa2)
      NT Status: STATUS_SUCCESS (0x00000000)
    ⊞ Flags: 0x98
    ⊞ Flags2: 0xc807
      Process ID High: 0
      Signature: 0000000000000000
      Reserved: 0000
      Tree ID: 2050
      Process ID: 1484
      User ID: 2048
      Multiplex ID: 1712
  ⊟ NT Create AndX Response (0xa2)
      Word Count (WCT): 42
      AndXCommand: No further commands (0xff)
      Reserved: 00
      AndXOffset: 135
      Oplock level: Batch oplock granted (2)
    ⊞ FID: 0x400f (\putty.exe)
      Create action: The file existed and was opened (1)
      Created: Apr 19, 2011 08:56:58.875000000 Eastern Daylight Time
      Last Access: Apr 19, 2011 08:56:58.921875000 Eastern Daylight Time
      Last Write: Apr 18, 2011 17:16:46.680100200 Eastern Daylight Time
      Change: Apr 19, 2011 08:54:57.582125000 Eastern Daylight Time
    ⊞ File Attributes: 0x00000020
      Allocation Size: 458752
      End Of File: 454657
      File Type: Disk file or directory (0)

0030  f9 b4 f8 66 00 00 00 00  00 87 ff 53 4d 42 a2 00   ...f.... ...SMB..
0040  00 00 00 98 07 c8 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 02 08 cc 05 00 08  b0 06 2a ff 00 87 00 02   ........ ..*.....
0060  0f 40 01 00 00 00 b0 8c  d5 44 91 fe cb 01 be b3   .@...... .D......
0070  dc 44 91 fe cb 01 6a b9  8b ec 0d fe cb 01 c2 b9   .D....j. ........
```

Figure 1: Note the file name, extension, and size.

The next step we have to take in order to extract this file is to isolate the appropriate block of traffic. Wireshark makes this pretty easy with its Follow TCP Stream functionality. Start by right-clicking any packet in the capture file and selecting Follow TCP Stream. This will bring up a window that contains all of the data being transferred in this particular communication stream concatenated together without all of the layer 2-4 headers getting in the way. We are only concerned about the traffic transferred from the client to the server so we will need to specify this in the directional drop down box by selecting 192.168.47.133 –> 192.168.47.132 (458592 bytes). Click Save As and save the file using the name putty.raw.

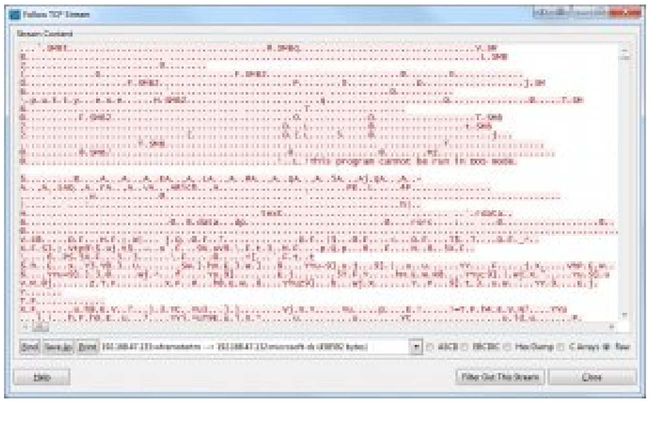
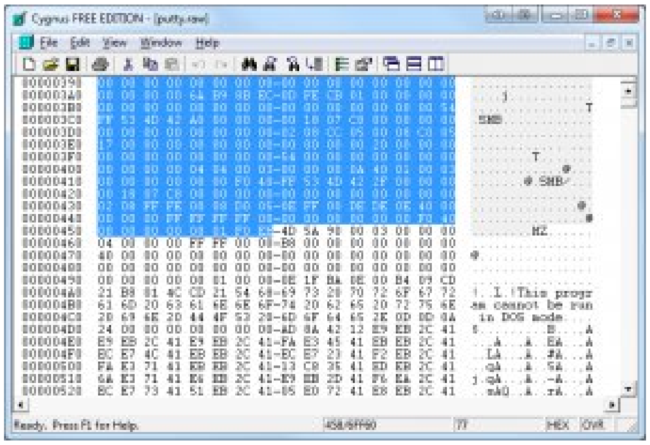


Figure 2: Saving the isolated traffic from Wireshark

If you were to view the properties of the data you just extracted and save you should find that its file size is 458,592 bytes. This is 3,935 bytes more than the size of the actual file that was transferred. This means that our goal is to get this raw files size down to exactly 454,657 bytes. This is where the real carving begins.

First things first, we have to delete all of the extra data that occurs before the executable data actually begins. Since we do know that the transferred file is an executable the quickest way to do this is to look for the executable header and delete everything that occurs before it. The executable header begins with the hex bytes 4D 5A (MZ in ASCII), which occurs approximately 1112 bytes into the putty.raw file. Once deleted, resave the file as putty.stage1. You should now be down to a file size of 457,480 bytes.


Cygnus FREE EDITION - [putty.raw]
File Edit View Window Help
Ready. Press F1 for Help.
HEX
OVR

```
00000390  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
000003A0  00 00 00 00 6A B9 8B EC-0D FE CB 01 00 00 00 00  ....j...........
000003B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 54  ...............T
000003C0  FF 53 4D 42 A0 00 00 00-00 18 07 C8 00 00 00 00  .SMB............
000003D0  00 00 00 00 00 00 00 00-02 08 CC 05 00 08 C0 05  ................
000003E0  17 00 00 00 00 00 00 00-00 00 00 00 20 00 00 00  ................
000003F0  00 00 00 00 00 00 00 00-54 00 00 00 00 00 00 00  ........T.......
00000400  00 00 00 00 04 04 00 03-00 00 00 DA 40 01 00 02  ............@...
00000410  00 00 00 00 00 00 F0 40-FE 53 4D 42 2F 00 00 00  .......@.SMB/...
00000420  00 18 07 C8 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000430  02 08 FF FE 00 08 D0 05-0E FF 00 DE DE 0E 40 00  ..............@.
00000440  00 00 00 FF FF FF FF 00-00 00 00 00 00 00 F0 40  ...............@
00000450  00 00 00 00 00 01 F0 00-4D 5A 90 00 03 00 00 00  ........MZ......
00000460  04 00 00 00 FF FF 00 00-B8 00 00 00 00 00 00 00  ................
00000470  40 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  @...............
00000480  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000490  00 00 00 00 00 01 00 00-0E 1F BA 0E 00 B4 09 CD  ................
000004A0  21 B8 01 4C CD 21 54 68-69 73 20 70 72 6F 67 72  !..L.!This progr
000004B0  61 6D 20 63 61 6E 6E 6F-74 20 62 65 20 72 75 6E  am cannot be run
000004C0  20 69 6E 20 44 4F 53 20-6D 6F 64 65 2E 0D 0D 0A   in DOS mode....
000004D0  24 00 00 00 00 00 00 00-AD 8A 42 12 E9 EB 2C 41  $.......B.....A
000004E0  E9 EB 2C 41 E9 EB 2C 41-FA E3 45 41 EB EB 2C 41  ...A...A..EA...A
000004F0  EC E7 4C 41 EB EB 2C 41-EC E7 23 41 F2 EB 2C 41  ..LA...A..#A...A
00000500  FA E3 71 41 ED EB 2C 41-13 C8 35 41 ED EB 2C 41  ..qA...A..5A...A
00000510  6A E3 71 41 E6 EB 2C 41-E9 EB 2D 41 F6 EA 2C 41  j.qA...A..-A...A
00000520  EC E7 73 41 51 EB 2C 41-05 E0 72 41 E8 EB 2C 41  ..sAQ..A..rA...A
```

Figure 3: Removing added bytes from the beginning of the file

Now things get a bit trickier. SMB transmits data in blocks. This is great for reliability since a lost or damaged block can be retransmitted, but it adds some extra work for us. This is because each block must contain some bytes of SMB header data in order to be interpreted correctly by the host that is receiving it. The good thing is that the size of this data is somewhat predictable, but you have to understand a bit more about SMB in order to put the rubber to the road. The thing to know here is that the data block size in SMB is limited to 64KB, or 65536 bytes. Of this amount, only 60KB is typically used for each block. These 61,440 bytes are combined with an additional 68 bytes of SMB header information. This means that after every 61,440 bytes of data we will have to strip out the next 68 bytes.

There is one thing to add to this that must be taken into consideration before stripping out those bytes. As a part of the normal SMB communication sequence, an additional packet is sent right after the first block. This is an NT Trans Request packet, which is packet 77 in the capture file. The SMB portion of this packet is 88 bytes, which means we will have to remove those 88 bytes in addition to the 68 bytes that make up the normal SMB block header, for a total of 156 bytes.

Now that we have all that sorted out let’s start removing bytes. In your hex editor, skip one byte past the 61,440th byte. This will be offset 0x0F000. You should start with this byte and select a range of 156 bytes and delete them. Save this file as putty.stage2.

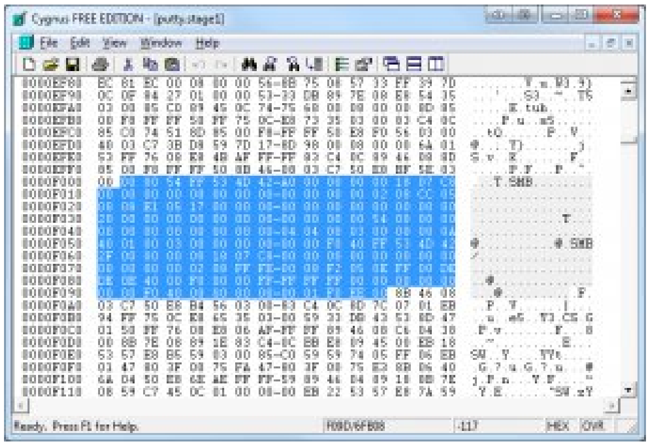

Cygnus FREE EDITION - [putty.stage1]
File Edit View Window Help
Ready. Press F1 for Help.

Figure 4: Removing the initial 156 bytes

Things get a bit easier now as we are just concerned with stripping out the 68 bytes after every block. Skip through the file in 61,440 byte increments deleting 68 bytes each time. This should occur X times in this file at offsets 0x1e000, 0x2d000, and 0x3c000, 0x4b000, 0x5a000, 0x69000. Once finished, save the file as putty.stage3.

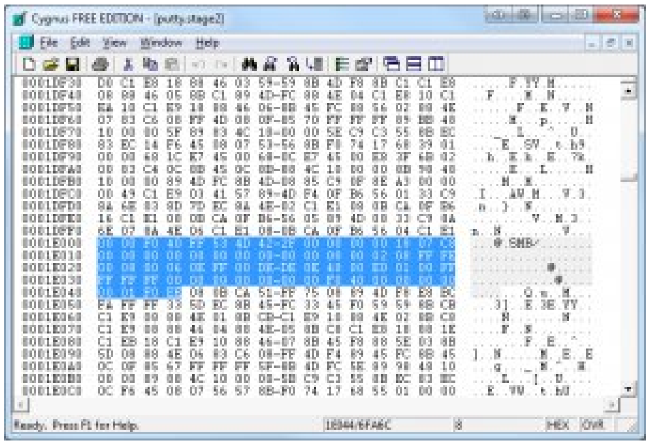
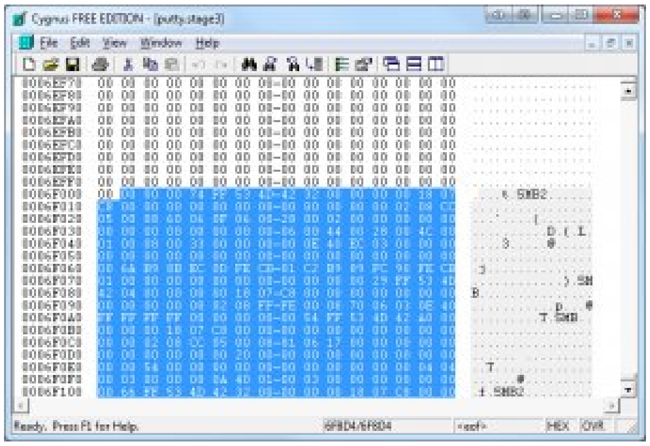

Cygnus FREE EDITION - [putty.stage2]
File Edit View Window Help
Ready. Press F1 for Help.

Figure 5: Removing a 68 byte SMB header block

Go ahead and take a look at the file size of putty.stage3. We are still XXX bytes off from our target, but luckily the last part is the easiest. The data stream is actually just padded by some extra information that needs to be deleted. We know that the file should be 454,657 bytes, so browse to that byte and delete everything that occurs after it.


Cygnus FREE EDITION - [putty.stage3]
File Edit View Window Help

```
0006EF70  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006EF80  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006EF90  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006EFA0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006EFB0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006EFC0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006EFD0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006EFE0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006EFF0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006F000  00 00 00 00 74 FE 53 4D-42 32 00 00 00 00 18 03  ....t.SMB2......
0006F010  CF 00 00 00 00 00 00 00-00 00 00 00 00 02 00 CC  ................
0006F020  05 00 00 60 06 EF 06 00-20 00 02 00 00 00 00 00  ...`.... .......
0006F030  00 00 00 00 00 00 00 00-06 00 44 00 28 00 4C 00  ..........D.(.L.
0006F040  01 00 00 00 33 00 00 00-00 0E 40 EC 03 00 00 00  ....3.....@.....
0006F050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0006F060  00 6A B9 0D EC 0D FE C8-01 C2 B9 09 FC 9E FE C0  .j..............
0006F070  01 00 00 00 00 00 00 00-00 00 00 00 29 FE 53 4D  ............).SM
0006F080  42 04 00 00 00 00 10 07-C8 00 00 00 00 00 00 00  B...............
0006F090  00 00 00 00 00 82 08 FF-FE 00 00 70 06 03 0E 40  ...........p...@
0006F0A0  FF FF FF FF 00 00 00 00-00 54 FE 53 4D 42 40 00  .........T.SMB@.
0006F0B0  00 00 00 10 07 C8 00 00-00 00 00 00 00 00 00 00  ................
0006F0C0  00 00 82 08 CC 85 00 00-81 06 17 00 00 00 00 00  ................
0006F0D0  00 00 00 00 00 00 20 00-00 00 00 00 00 00 00 00  ...... .........
0006F0E0  00 00 54 00 00 00 00 00-00 00 00 00 00 00 04 04  ..T.............
0006F0F0  00 03 00 00 00 0A 40 01-00 03 00 00 00 00 00 00  ......@.........
0006F100  00 6A FE 53 4D 42 32 00-00 00 00 18 07 C8 00 00  .j.SMB2.........
```

```
Ready, Press F1 for Help.    6F804/6F804    <eof>    HEX    OVR
```

Figure 6: Trimming the extra bytes off the end of the file

Save the final product as putty.exe and if you did everything right, you should have a fully functioning executable.

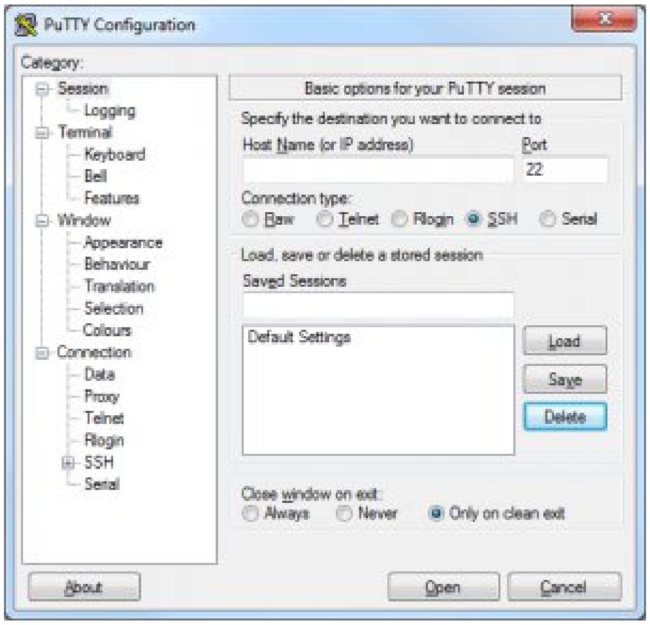


Figure 7: Success! The executable runs!

�

The whole process can be broken down into a series of repeatable steps:

1. Record the file name, extension, and size by examining one of the SMB NT Create AndX Response packets
2. Isolate and extract the appropriate stream data from Wireshark by using the Follow TCP Stream feature and selecting the appropriate direction of traffic
3. Remove all of the bytes occurring before the actual file header using a hex editor
4. Following the first 61,440 byte block, remove 156 bytes
5. Following each successive 61,440 byte block, remove 68 bytes
6. Trim the remaining bytes off of the file so that it matches the file size recorded in step 1

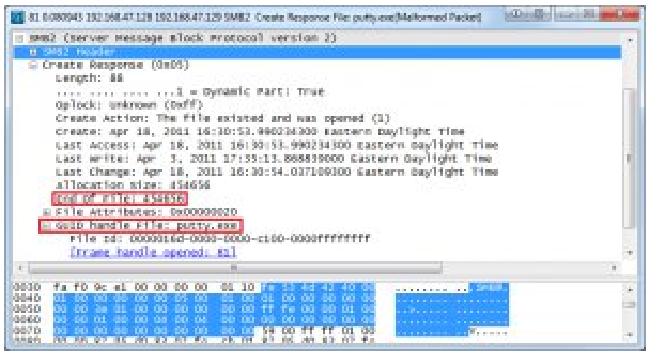
�

Carving Files from SMB2 Packets

�

Microsoft introduced SMB2 with Windows Vista and began using it with its newer operating systems moving forward. In order to setup a packet capture for this scenario I took two Windows 7 (x32) virtual machines running on VMWare Workstation and placed them in the same network. Once they were able to communicate with each other I setup a shared folder on one host (192.168.47.128) that is acting as the server. I then fired up Wireshark and began capturing packets as I copied an executable file from the client (192.168.47.129) to the servers shared folder. The resulting packet capture is called **smb2_puttyexe_xfer.pcap**.

You should notice that this traffic is a little bit cleaner than the SMB traffic we looked at earlier. This is because SMB2 is optimized so that there are a lot less commands. Whereas SMB had over a hundred commands and subcommands, SMB2 only has nineteen. Regardless, we still need to find the filename being transferred and the size of that file. One of the best places to do this is at one of the SMB2 Create Response File packets. This packet type serves a purpose similar to that of the SMB NT Create AndX Response packet. You can filter these out in Wireshark with the filter *(smb2.cmd == 5) && (smb2.flags.response == 1)*. The last one of these in the capture, which is packet 81, is the one we want to look at since it occurs after the file transfer is complete. This identifies the file name as putty.exe and the file size as 454,656 bytes. This is indeed the same file as our earlier example, but it is being reported as being one byte smaller. The missing byte is just padding at the end of the file and has a null value so it's not of any real concern to us.



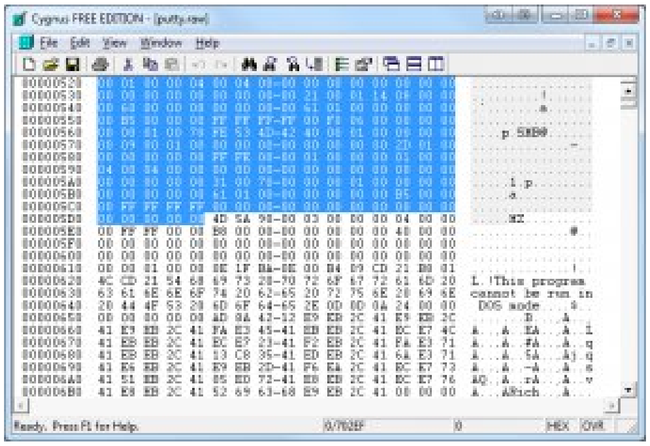
```
81 0.080943 192.168.47.128 192.168.47.129 SMB2 Create Response File: putty.exe[Malformed Packet]
SMB2 (Server Message Block Protocol version 2)
  SMB2 Header
  Create Response (0x05)
    Length: 88
    .... .... .... ...1 = Dynamic Part: True
    Oplock: Unknown (0xff)
    Create Action: The file existed and was opened (1)
    Create: Apr 18, 2011 16:30:53.990234300 Eastern Daylight Time
    Last Access: Apr 18, 2011 16:30:53.990234300 Eastern Daylight Time
    Last Write: Apr  3, 2011 17:55:13.868839000 Eastern Daylight Time
    Last Change: Apr 18, 2011 16:30:54.037109300 Eastern Daylight Time
    Allocation Size: 454656
    End Of File: 454656
    File Attributes: 0x00000020
    GUID handle File: putty.exe
      File Id: 0000016d-0000-0000-c100-0000ffffffff
      [Frame handle opened: 81]

0030  fa f0 9c a1 00 00 00 00  01 10 fe 53 4d 42 40 00   ...........SMB@.
0040  [illegible]
0050  [illegible]
0060  [illegible]
0070  [illegible]
```

Figure 8: Once again we note the file name, extension, and size

At this point you should perform the same steps as we did earlier to isolate and extract the data stream from the capture using Wiresharks Follow TCP Stream option. Doing this should yield a new putty.raw file whose file size is 459,503 bytes. This is 4,847 too big, so it's time to get to carving.

Once again we need to start by stripping out all of the data before the executable header. Fire up your favorite hex editor and remove everything before the bytes 4D 5A. This should account for a deletion of 1,493 bytes.

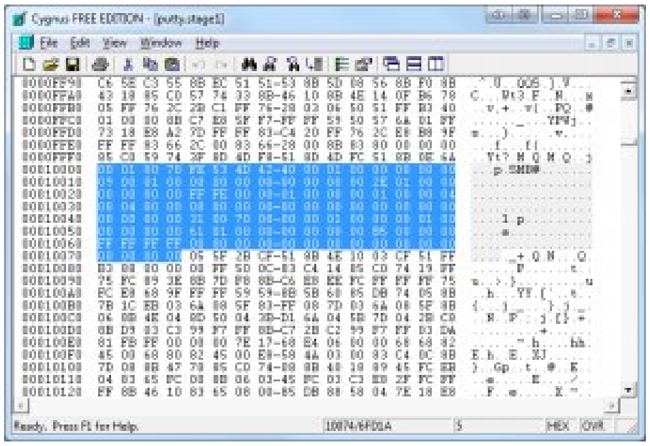

Cygnus FREE EDITION - [putty.raw]
File Edit View Window Help
Ready. Press F1 for Help.
0/702EF
0
HEX
OVR

```
00000520  00 01 00 00 04 00 04 00-00 00 00 00 00 00 00 00
00000530  00 00 00 00 00 00 00 00-00 21 00 01 14 00 00 00
00000540  00 60 00 00 00 00 00 00-00 61 01 00 00 00 00 00
00000550  00 B5 00 00 00 FF FF FF-FF 00 F8 06 00 00 00 00
00000560  00 00 01 00 70 FE 53 4D-42 40 00 01 00 00 00 00    p SMB@
00000570  00 09 00 01 00 00 00 00-00 00 00 00 00 2D 01 00
00000580  00 00 00 00 00 FF FE 00-00 01 00 00 00 01 00 00
00000590  04 00 04 00 00 00 00 00-00 00 00 00 00 00 00 00
000005A0  00 00 00 00 00 31 00 70-00 00 00 01 00 00 00 00    1 p
000005B0  00 00 00 00 00 61 01 00-00 00 00 00 00 B5 00 00    a
000005C0  00 FF FF FF FF 00 00 00-00 00 00 00 00 00 00 00
000005D0  00 00 00 00 00 4D 5A 90-00 03 00 00 00 04 00 00    MZ
000005E0  00 FF FF 00 00 B8 00 00-00 00 00 00 00 40 00 00
000005F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00000600  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00000610  00 00 01 00 00 0E 1F BA-0E 00 B4 09 CD 21 B8 01
00000620  4C CD 21 54 68 69 73 20-70 72 6F 67 72 61 6D 20    L.!This program
00000630  63 61 6E 6E 6F 74 20 62-65 20 72 75 6E 20 69 6E    cannot be run in
00000640  20 44 4F 53 20 6D 6F 64-65 2E 0D 0D 0A 24 00 00     DOS mode....$
00000650  00 00 00 00 00 AD 8A 42-12 E9 EB 2C 41 E9 EB 2C
00000660  41 E9 EB 2C 41 FA E3 45-41 EB EB 2C 41 EC E7 4C
00000670  41 EB EB 2C 41 EC E7 23-41 F2 EB 2C 41 FA E3 71
00000680  41 EB EB 2C 41 13 C8 35-41 ED EB 2C 41 6A E3 71
00000690  41 E6 EB 2C 41 E9 EB 2D-41 F6 EA 2C 41 EC E7 73
000006A0  41 51 EB 2C 41 85 ED 72-41 EB EB 2C 41 EC E7 76
000006B0  41 E8 EB 2C 41 52 69 63-68 E9 EB 2C 41 00 00 00    A...ARich..A
```

Figure 9: Removing the extra bytes found prior to the executable header

Now things change a bit. SMB2 works in a method similar to SMB, but it actually allows for more data to be transferred at once. SMB had a maximum block size of 64K because it has a limit of 16-bit data sizes. SMB2 uses either 32-bit or 64-bit data sizes, which raises the 64KB limit. In the case of the transfer taking place in the sample PCAP file, these were two 32-bit Windows 7 hosts under their default configuration which means that the block size is set at 64KB. Unlike SMB however, the full 64KB is used, so we will see data in chunks of 65,536 bytes being transferred. These 65,536 bytes combine with a 116 byte SMB2 header to form the full block.

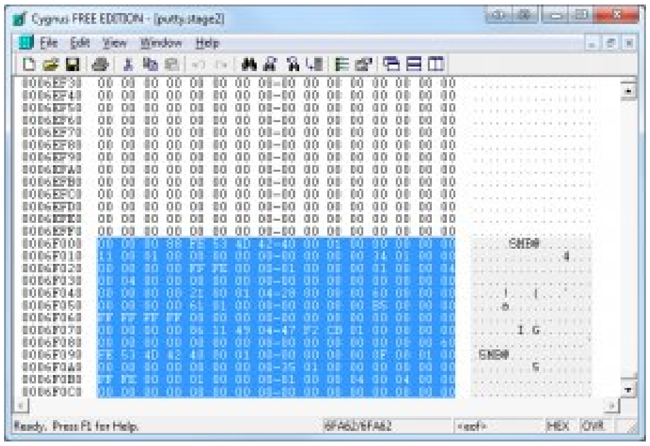
SMB2 doesn't include an additional initial request packet like the SMB Trans Request, so we don't have to worry about stripping out any extra bytes right off the bat. As a matter of fact, some might say that carving data from SMB2 is a bit easier since you only have to strip out 116 bytes after each block of 65,536 bytes. You can do this now on putty.stage1. In doing so you should be deleting 116 bytes of data at offsets 0x10000, 0x20000, 0x30000, 0x40000, 0x50000and 0x60000.


Cygnus FREE EDITION - [putty.stage1]
File Edit View Window Help
Ready. Press F1 for Help.
10074/6FD1A
5
HEX
OVR

```
0000FF90  C6 5E C3 55 8B EC 51 51-53 8B 5D 08 56 8B F0 8B
0000FFA0  43 18 85 C0 57 74 33 8B-46 10 8B 4E 14 0F B6 78
0000FFB0  05 FF 76 2C 2B C1 FF 76-28 03 06 50 51 FF B3 40
0000FFC0  01 00 00 8B C7 E8 5F F7-FF FF 59 50 57 6A 01 FF
0000FFD0  73 18 E8 A2 7D FF FF 83-C4 20 FF 76 2C E8 B8 9F
0000FFE0  FF FF 83 66 2C 00 83 66-28 00 8B 83 80 00 00 00
0000FFF0  85 C0 59 74 3F 8D 4D F8-51 8D 4D FC 51 8B 0E 6A
00010000  00 01 00 70 FE 53 4D 42-40 00 01 00 00 00 00 00    p SMB@
00010010  09 00 01 00 00 00 00 00-00 00 00 00 2E 01 00 00
00010020  00 00 00 00 FF FE 00 00-01 00 00 00 01 00 00 04
00010030  00 04 00 00 00 00 00 00-00 00 00 00 00 00 00 00
00010040  00 00 00 00 31 00 70 00-00 00 01 00 00 00 01 00    1 p
00010050  00 00 00 00 61 01 00 00-00 00 00 00 B5 00 00 00    a
00010060  FF FF FF FF 00 00 00 00-00 00 00 00 00 00 00 00
00010070  00 00 00 00 05 5F 2B CF-51 8B 4E 10 03 CF 51 FF
00010080  B3 80 00 00 00 FF 5D 0C-83 C4 14 85 C0 74 13 FF
00010090  75 FC 89 3E 8B 7D F8 8B-C6 E8 EE FC FF FF FF 75
000100A0  FC E8 68 9F FF FF 59 59-8B 5B 60 85 DB 74 05 8B
000100B0  7B 1C EB 03 6A 08 5F 83-FF 08 7D 03 6A 08 5F 8B
000100C0  06 8B 4E 04 8D 50 04 2B-D1 6A 04 5B 7D 04 2B C8
000100D0  8B D9 83 C3 99 F7 FF 8B-C7 2B C2 99 F7 FF 83 DA
000100E0  81 FB FF 00 00 00 7E 17-68 E4 06 00 00 68 68 82
000100F0  45 00 68 80 82 45 00 E8-58 4A 03 00 83 C4 0C 8B
00010100  7D 08 8B 47 70 85 C0 74-08 8B 40 18 89 45 FC EB
00010110  04 83 65 FC 00 8B 06 03-45 FC 03 C3 8B 2F FC FF
00010120  FF 8B 46 10 83 65 08 00-85 DB 88 58 04 7E 18 E8
```

Figure 10: Removing 116 bytes of data following the first 65,536 chunk

Once you've finished this save the file as putty.stage2. All that is left is to remove the final trailing bytes from the file. In order to do this, browse to by 454,656 and delete every byte that occurs after it.


Cygnus FREE EDITION - [putty.stage2]
File Edit View Window Help

```
0006EF30  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EF40  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EF50  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EF60  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EF70  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EF80  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EF90  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EFA0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EFB0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EFC0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EFD0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EFE0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006EFF0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006F000  00 00 00 88 FE 53 4D 42-40 00 01 00 00 00 00 00      SMB@
0006F010  11 00 01 00 00 00 00 00-00 00 00 00 34 01 00 00             4
0006F020  00 00 00 00 FF FE 00 00-01 00 00 00 01 00 00 04
0006F030  00 04 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0006F040  00 00 00 00 21 00 01 04-28 00 00 00 60 00 00 00      !   (
0006F050  00 00 00 00 61 01 00 00-00 00 00 00 B5 00 00 00      a
0006F060  FF FF FF FF 00 00 00 00-00 00 00 00 00 00 00 00
0006F070  00 00 00 00 05 11 49 04-47 F2 CD 01 00 00 00 00       I G
0006F080  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 50
0006F090  FE 53 4D 42 40 00 01 00-00 00 00 00 0F 00 01 00  SMB@
0006F0A0  00 00 00 00 00 00 00 00-35 01 00 00 00 00 00 00          5
0006F0B0  FF FE 00 00 01 00 00 00-01 00 00 04 00 04 00 00
0006F0C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
Ready. Press F1 for Help.    6FA62/6FA62    <eof>    HEX  OVR
```

Figure 11: Removing the final trailing bytes

Finally, save the file as putty.exe and you will have a fully functioning executable. The process of carving a file from an SMB2 data stream breaks down as follows:

1. Record the file name, extension, and size by examining one of the SMB2 Create Response File packets
2. Isolate and extract the appropriate stream data from Wireshark by using the Follow TCP Stream feature and selecting the appropriate direction of traffic
3. Remove all of the bytes occurring before the actual file header using a hex editor
4. Following each successive 65,536 byte block (assuming a 64K block size), remove 116 bytes
5. Trim the remaining bytes off of the file so that it matches the file size recorded in step 1

�

Conclusion

�

That's all there is to it. I'll be the first to admit that I didn't cover every single aspect of SMB and SMB2 here and there are a few factors that might affect your success in carving files from these streams, but this article shows the overall process. Taking this one step farther, it's pretty reasonable to assume that this process can be automated with a quick Python script, but this is something I've not devoted the time to yet. If you feel like taking up that challenge then be sure to get in touch and I'll be glad to post your code as an addendum to this post. In the mean time, happy carving!

�

References

http://msdn.microsoft.com/en-us/library/cc246231%28v=PROT.10%29.aspx

http://msdn.microsoft.com/en-us/library/cc246482%28v=PROT.10%29.aspx

http://channel9.msdn.com/Blogs/Darryl/Server-Message-Block-SMB21-Drill-down