Advanced Packet Filtering

By Laura Chappell, Protocol Analysis Institute, LLC <www.packet-level.com>

Note: This is the second article in a two-part set. 'Basic Packet Filtering' can be found online at www.packet-level.com.

Now we'll do some really cool stuff (wringing hands….).

Advanced packet filtering requires a very strong knowledge of packet structures and protocols – it's not for the faint of heart. In this article, I'll show you the steps required to build an advanced filter using the Sniffer Pro 3.5.

Most quality analyzers have some way of building advanced filters with the following characteristics:

- Bit or byte value patterns that enable you to capture traffic that contains a specific value at a specific location within a packet.
- Boolean operations that enable you to combine these patterns with AND/OR operands.

Bit and Byte Value Patterns

The practice of specifying values at exact bit and/or byte locations in a packet can turn any analyzer into an extremely powerful tool. Here are a few examples of some patterns that you may want to match:

- TCP headers with the SYN bit set to 1 (indicating someone attempting to make a TCP connection on the network).
- TCP packets that contain the value RETR above the TCP header (indicating a device is transferring a file using FTP).
- ICMP packets with the type 0 (indicating that a device is sending destination unreachable packets onto the network).

Let's take a look at how some of these advanced filters are built.

TCP Handshakes

Connection-oriented services, such as FTP and HTTP, require an initial TCP handshake to establish the connection and exchange a starting sequence number. This sequence number increases in accordance with the amount of data received, thereby offering reliable, guaranteed service for TCP data.

In the first packet of the TCP handshake, the SYN (SYNchronize sequence number) bit in the TCP header is set to 1.

Why would you care about this packet? Well… let's say you've configured a very secure network. You decided that no one should be able to connect to your inside computers from the Internet. That means no SYN packets should make it through the firewall, right? You might want to check that out by building a filter on all SYN traffic crossing the wire on the inside of the firewall.

The bit sequence in TCP headers is structured as follows:

r	r	U	A	P	R	S	F	
0	0	0	0	0	0	1	0	= 0x02

where,

r = reserved bits – set to 0
U = Urgent bit (value 0x20)
A = Acknowledgment bit (value 0x10)
P = Push bit (value 0x08)
R = Reset bit (value 0x04)
S = Synchronize bit (value 0x02)
F = Finish bit (value 0x01)

Figure 1 shows the filter I built to capture all packets that have the SYN bit set.

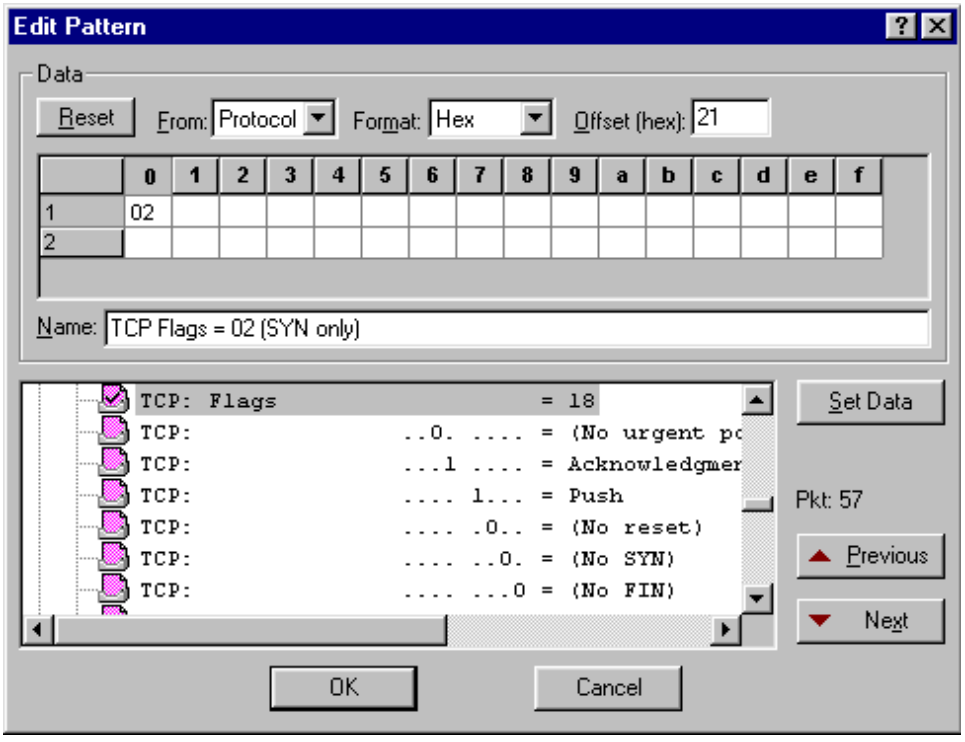


Figure 1: The TCP flag fields are at offset 21 (hex) from the protocol layer (after the MAC header).

Note: If someone is really trying to be sneaky, they may try to go through your firewall using a packet with both the SYN flag and another flag set. Ugh. In this case, you might have to build a more advanced filter – a boolean filter that ORs together various flag setting patterns.

Hidden FTP File Transfers

Although the specification defines port 21 for FTP commands, there are numerous FTP server packages that allow you to select your own port number to run FTP services from. This is certainly a security hazard. Consider what would happen when a disgruntled employee sets up his office desktop with FTP services on port 80. He goes home and zips right through the firewall (which naturally would support port 80 which is used for HTTP operations). Yipes! Life is ugly!

Figure 2 shows a simple filter set up to capture all RETR commands regardless of the port number that is being used.

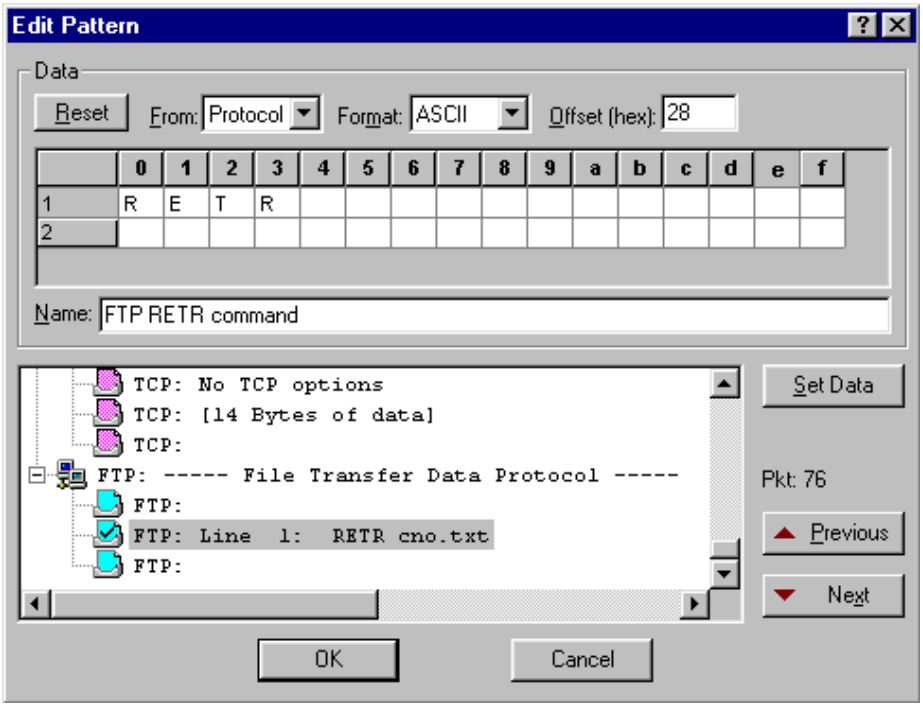


Figure 2: By switching over to ASCII format, you can enter the commands as they are listed above.

Note: *Ephemeral* ports are ports that are temporary. They are only used for a short-period of time. For example, when an FTP client issues an "ls" command to view the contents of a directory, the FTP client software translates this to the NLST command. First, however, the client will set up a temporary port number to be used for the transfer of the directory contents. The use of these ephemeral ports really irritates most protocol analysts. Just when you think you've set up a nice 'FTP' filter using the checkbox method (see "Basic Protocol Filtering" at www.packet-level.com), you find out that the filter won't capture even half of the actual FTP communications. Sigh.

Naturally, if you really want to create an awesome filter, you would build a filter that combines all the different commands using an OR, as shown later in this article.

Destination Unreachables

When a device cannot find a service, network or host, it can respond to a request with a Destination Unreachable ICMP packet.

Any decent analyzer should have a predefined ICMP filter. These filters, however, don't look for specific types of ICMP traffic. The value 0x03 in the ICMP header's type field identifies the packet as a Destination Unreachable packet.

In Figure 3, I've set up a filter that only captures traffic with the value 0x03 in the 'Type' field.

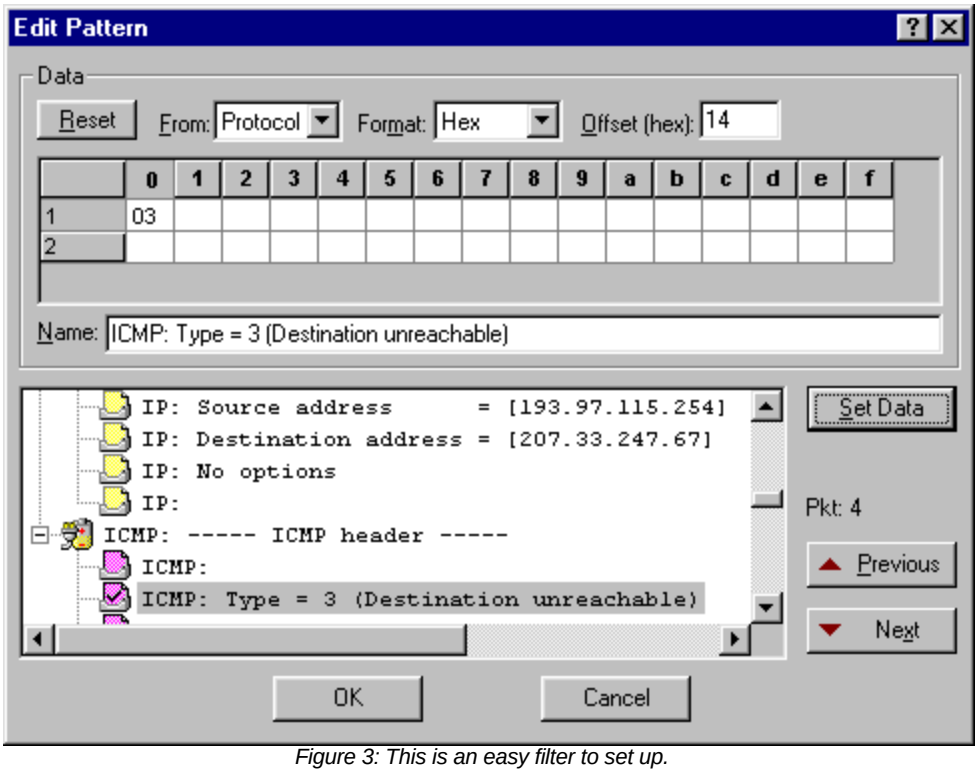


Figure 3: This is an easy filter to set up.

I highly recommend that you learn ICMP from the inside out -- today! Read RFC 792, get the "Packet-Level ICMP" course from www.podbooks.com and look at your own ICMP traffic. You'll be amazed at how you can optimize your network!

Note: If you are planning on going to BrainShare 2001, check out Laura's session listing for information on her "Troubleshooting with ICMP" seminar.

Boolean Operations

To effectively use some patterns, you have to mix them together into a single filter. For example, if you are trying to capture all fragments on the network and you filter on all packets that have the 'more fragments' bit set to 1, you'd miss the last fragment of the fragment set.

Likewise, if you filtered on the RETR FTP command to catch anyone who might be running a non-standard implementation of FTP, you'd miss anyone putting files onto your network using the STOR command.

These are perfect examples of why you need to using boolean operands to mix up these patterns.

The standard operands include:

AND
OR
AND NOT

An example of each of these follows:

AND (Catching Port Unreachables)

Packets with the ICMP type field value of 3 (Destination Unreachable)
AND
Packets with the ICMP Code value 3 (Port Unreachable)

OR (Catching Non-Standard FTP Operations)

Packets with STOR following the TCP header (for FTP put commands)
OR
Packets with RETR following the TCP header (for FTP get commands)
OR
Packets with NLST following the TCP header (for FTP file listings)
(Consider adding AND NOT port 21 to this one.)

AND NOT (Catching All Fragmented Packets)

Packets with the 'more fragments' bit set to 1 (part of a fragment set)
AND NOT
The packets with an IP fragment offset of 0

As you can see, a solid understanding of the protocols is required to really take full advantage of the advanced boolean filtering. Once you put in the time and effort, however, you will find that your analyzer has become ten times as useful as before.

Let's go through each of the boolean examples above to show you how these boolean filters are defined on the Sniffer 3.5.

AND (Catching Port Unreachables)

In this example, you want to capture a specific type of ICMP Destination Unreachable packet – you want to look for the Port Unreachable type packets that indicate someone is looking for a service that just doesn't exist at that location.

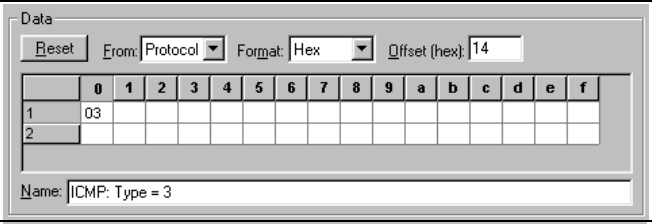
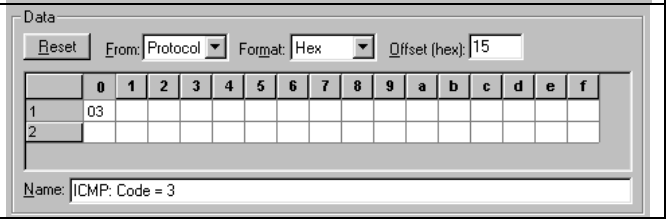
Destination Unreachable code numbers are listed below.

Codes

0 Net Unreachable
1 Host Unreachable
2 Protocol Unreachable
3 Port Unreachable
4 Fragmentation Needed and Don't Fragment was Set
5 Source Route Failed
6 Destination Network Unknown
7 Destination Host Unknown
8 Source Host Isolated
9 Communication with Destination Net is Administratively Prohibited
10 Communication with Destination Host is Administratively Prohibited
11 Destination Network Unreachable for Type of Service
12 Destination Host Unreachable for Type of Service
13 Communication Administratively Prohibited [see RFC1812]
14 Host Precedence Violation [see RFC1812]
15 Precedence cutoff in effect [see RFC1812]

Note: This list is maintained by IANA (Internet Assigned Numbers Authority). As a protocol analyst, you should consult them at www.iana.org to stay up-to-date on the latest assigned numbers for various protocols.

Here are details on the two patterns we are interested in:

Patterns	Sniffer Pattern Window
Pattern 1: Packets with the ICMP type field value of 3 (Destination Unreachable)	
Pattern 2: Packets with the ICMP Code value 3 (Port Unreachable)	

We want to "AND" these two operations because you are looking for packets that have both the type 3 value **and** the code 3 value at the correct offset. The final data pattern filter should look like the one shown in Figure 4.

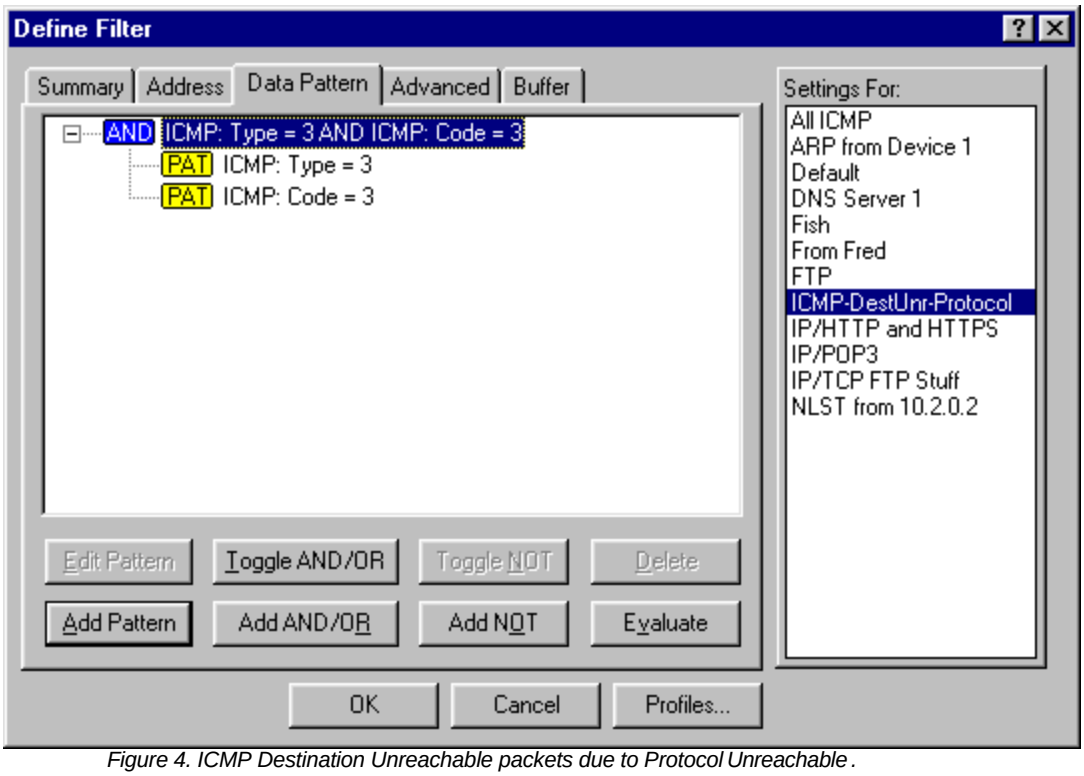


Figure 4. ICMP Destination Unreachable packets due to Protocol Unreachable.

OR (Catching Non-Standard FTP Operations)

Earlier in this article, we made a filter that would capture all the packets that contain the value RETR (used when someone retrieves a file using FTP).

FTP, however, uses a series of commands directly after the TCP header. The commands are:

USER	Login with this username
PASS	Use this password for login
NLST	List files on remote system
CWD	Change working directory (on remote system)
PORT	Use the following ephemeral port number
RETR	Retrieve a file
STOR	Put a file on the remote system
QUIT	Logout

What if you are interested all the traffic that is used to put files onto local systems, retrieve files from local systems, or view file lists? In this case we want to build a filter that identifies RETR, STOR and NLST packets.

Here are details on the three patterns we are interested in:

Patterns	Sniffer Pattern Window
Pattern 1: Packets with the RETR pattern (used to get files using FTP)	Data Reset From: Protocol Format: ASCII Offset (hex): 28 0 1 2 3 4 5 6 7 8 9 a b c d e f 1 R E T R 2 Name: FTP RETR Command
Pattern 2: Packets with the STOR pattern (used to put files using FTP).	Data Reset From: Protocol Format: ASCII Offset (hex): 28 0 1 2 3 4 5 6 7 8 9 a b c d e f 1 S T O R 2 Name: FTP STOR Command
Pattern 3: Packets with the NLST pattern (used to list files using FTP).	Data Reset From: Protocol Format: ASCII Offset (hex): 28 0 1 2 3 4 5 6 7 8 9 a b c d e f 1 N L S T 2 Name: FTP NLST Command

We want to “OR” these three operations because you are looking for packets that have either the value RETR, STOR and NLST commands. The final data pattern filter should look like the one shown in Figure 5.

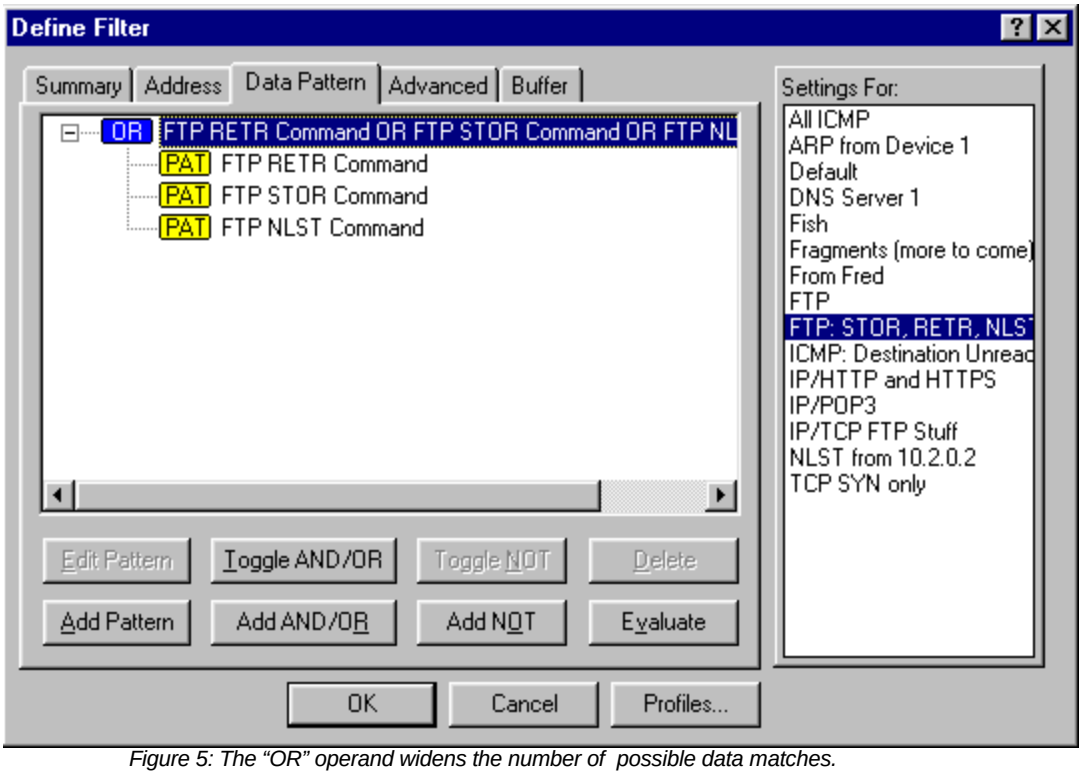


Figure 5: The “OR” operand widens the number of possible data matches.

AND NOT (Catching All Fragmented Packets)

IP has the ability to fragment packets when packets must cross a network that supports a smaller MTU (maximum transmission unit) size.

Figure 6 shows the format of an IP header.

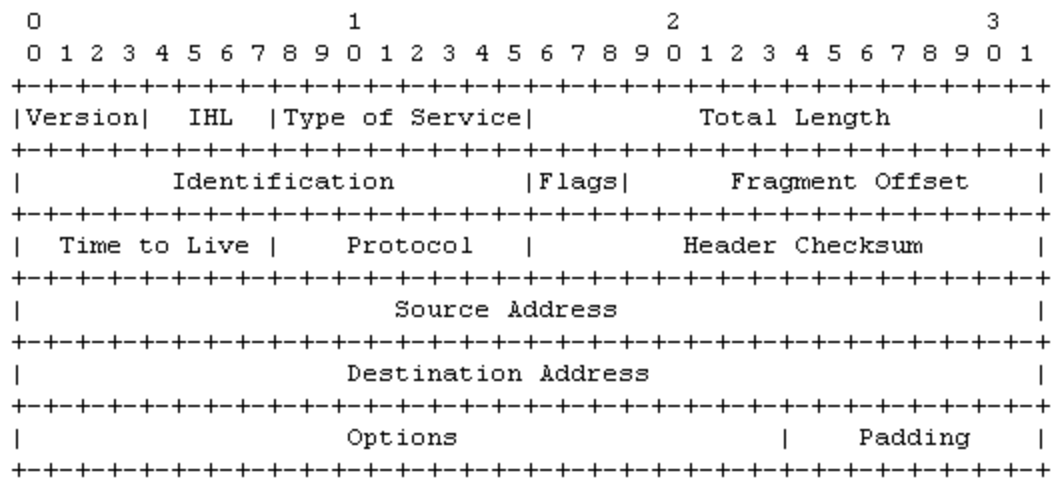
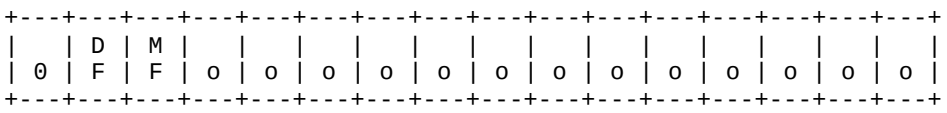


Figure 6: The lovely IP header structure – straight out of RFC 791.

The breakdown of the Flags and Fragment Offset fields shown below in binary format:



Where ‘0’ is always set to zero, and

‘DF’ identifies the ‘Don’t Fragment bit’ which means
‘Don’t Fragment’ when set to 1
‘OK to Fragment’ when set to 0
and

‘MF’ identifies the ‘More Fragments’ bit which means
‘More Fragments to Come’ when set to 1
‘Last Fragment’ when set to 0
and

“o” identifies the offset field which defines the location of this packet in the entire data stream.

A simple filter based on “1 in the more to come bit” captures all fragments *except* the last fragment in the set because the last fragment in the set contains the value 0 in the ‘more to come’ bit field. This requires some extra thinking…

Here are the characteristics of the various packets seen on the network:

First Fragment: More to come bit = 1; Fragment Offset = 0

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Middle Fragments: More to come bit = 1; Fragment Offset 0

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0 | 0 | 1 |     ------- not equal to 0 --------               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

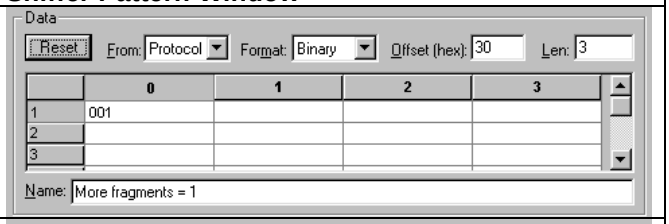
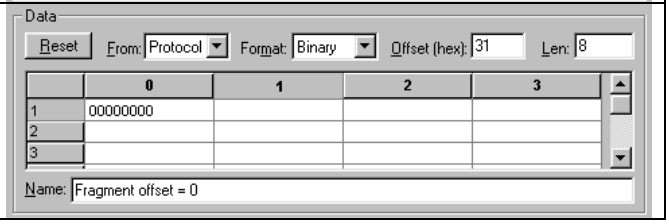
Last Fragment: More to come bit = 0; Fragment Offset 0

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0 | 0 | 0 |     ------- not equal to 0 --------               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Unfragmented Packets: More to come bit = 0; Fragment Offset = 0

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Can you find the defining patterns that would enable you to capture all fragments?

Patterns	Sniffer Pattern Window
Pattern 1: Packets with more to come bit set to 1	
Pattern 2: Packets that do not contain offset value 0.	

Yes! That's it! All packets in a fragment set have either '1' in the 'More Fragments' bit location or they have some value other than 0 in the offset field!

And we can even use Sniffer's binary format to make it a bit more clear instead of jumping to hex translations and all that garbage! Check out Figure 7 that contains the summary of the final filter that uses the AND NOT operand.

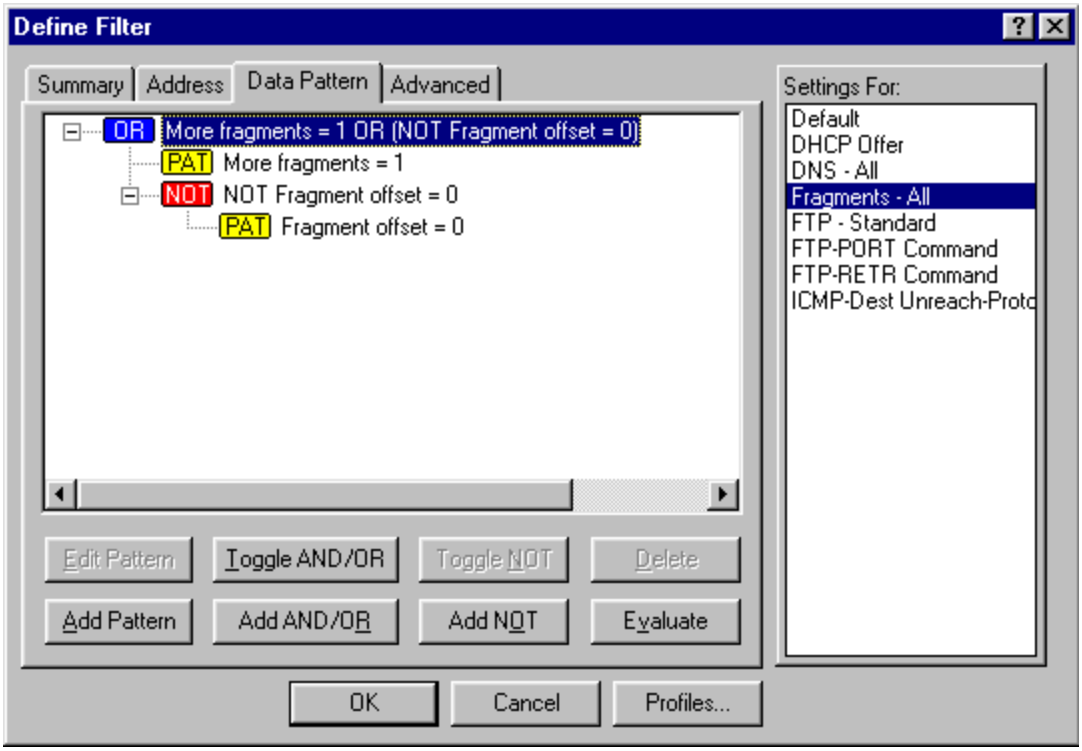


Figure 7: Wow! What a slick little filter!

If you've ever heard me lecture on filtering, then you know that I consider it a true art form. You need to know your protocols – know where to get the offset and field value information – know how to figure out what the possible variations are – and test your filters.

Archived Laura Chappell articles (www.packet-level.com/archives.htm):

- Looking at the Sniffer Dashboard
- Sniffer: Using the Capture Panel
- TrenchTime: Ports to Watch
- Did Your Know: Wireless Networks are Not Immune to Sniffing?
- The 10 Truths of Network Troubleshooting
- Carnivore?