CHAPPELL UNIVERSITY

Wireshark® Network Analysis

The Official Wireshark Certified Network Analyst™ Study Guide

1st Edition (Version 1.0e)

To arrange bulk purchase discounts for sales promotions, events, training courses, or other purposes, please contact Chappell University at the address listed on the next page.

Book URL: *www.wiresharkbook.com*
13-digit ISBN: 978-1-893939-99-8
10-digit ISBN: 1-893939-99-5

Distributed worldwide for Chappell University through Protocol Analysis Institute, LLC.

For general information on Chappell University or Protocol Analysis Institute, LLC, including information on corporate licenses, updates, future titles or courses, contact the Protocol Analysis Institute, LLC at 408/378-7841 or send email to *info@chappellU.com*.

For authorization to photocopy items for corporate, personal or educational use, contact Protocol Analysis Institute, LLC at email to *info@chappellU.com*.

Chapter 9: Create and Apply Display Filters

Wireshark Certified Network Analyst Exam Objectives covered:

- Understanding the Purpose of Display Filters
- Create Display Filters Using Auto-Complete
- Apply Saved Display Filters
- Use Expressions for Filter Assistance
- Make Display Filters Quickly Using Right-Click Filtering
- Understand Display Filter Syntax
- Combine Display Filters with Comparison Operators
- Alter Display Filter Meaning with Parentheses
- Filter on Specific Bytes in a Packet
- Use Display Filter Macros for Complex Filtering
- Avoid Common Display Filter Mistakes
- Manually Edit the *dfilters* File

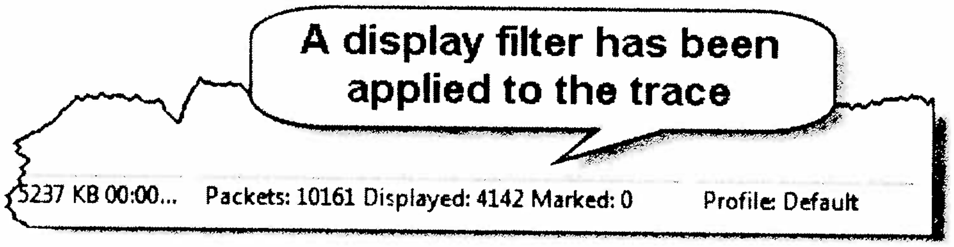
Understanding the Purpose of Display Filters

Display filters enable you focus on specific packets based on a criteria you define. You can filter on traffic that you want to see (inclusion filtering) or filter undesired traffic out of your view (exclusion filtering).

Display filters can be created using several techniques:

- Type in the display filter (possibly using auto-complete)
- Apply saved display filters
- Use expressions
- Right-click filter
- Apply conversation or endpoint filters

When you apply a display filter, the Status Bar indicates the total number of packets and the packets displayed, as shown in Figure 125. In this example, the trace file contains 10,161 packets, but only 4,142 are displayed because they match our filter.



```
5237 KB 00:00...   Packets: 10161 Displayed: 4142 Marked: 0        Profile: Default
```

Figure 125. The Status Bar shows the displayed packet count

Wireshark's display filters use a specialized Wireshark filter format while capture filters use the Berkeley Packet Filtering (BPF) format—they are not interchangeable. The BPF filter format is also used by tcpdump. In rare instances, there just happen to be some capture and display filters that look the same because both filter mechanisms support identical filter syntax. For example, the capture and display filter syntax for TCP traffic is the same: `tcp`.

Wireshark includes a default set of display filters that are saved in a file called *dfilters* in the Global Configurations directory. When you edit the default display filters, a new *dfilters* file is saved in the Personal Configurations directory or in the active profile directory.

Display filters can be relatively simple. The filter field or protocol must be defined in lower case in most situations[56]. You can use uppercase characters for the value portion of the filter as we will cover later in this chapter.

[56] All field field or protocol text was in lowercase until VoIP filters were added. Some VoIP-related filters use uppercase and lowercase definitions. Use the auto complete feature to help with VoIP filters.

The followi

Display filt
example, th
packets that

Using opera
create displ
based on fie

Display filte
following ar

[57] Note that W
recognizes bo

The following are examples of very basic display filters.

```
tcp
ip
udp
icmp
bootp[57]
arp
dns
nbns
```

Display filters can be created based on a packet characteristic (not an actual field) if desired. For example, the following filters display packets that contain one of the TCP analysis flags packets and packets that have an invalid IP header checksum. These are not actual fields in a TCP packet.

```
tcp.analysis.flags
ip.checksum_bad
```

Using operators (see *Combine Display Filters with Comparison Operators* on Page 220) you can create display filters based on the contents of a field. The following list provides examples of filters based on field values.

```
http.request.method == "GET"
tcp.flags == 0x20
tcp.window_size < 1460
tcp.stream eq 1
icmp.type == 8
dns.qry.name == "www.wireshark.org"
```

Display filters can be quite complex and include numerous criteria that must be matched. The following are some examples of display filters using multiple criteria:

- The following filter displays ARP requests except ARP requests from the MAC address 00:01:5c:22:a5:82.
  ```
  (arp.opcode == 0x0001) &&
  !(arp.src.hw_mac == 00:01:5c:22:a5:82)
  ```
- The following display filter shows any BOOTP/DHCP packets to or from 74.31.51.150 that lists 73.68.136.1 as the relay agent.
  ```
  (bootp.ip.relay == 73.68.136.1) &&
  (bootp.ip.your == 74.31.51.150)
  ```

[57] Note that Wireshark does not recognize `dhcp` as a display filter. DHCP is based on BOOTP and Wireshark recognizes `bootp` as the filter to display all DHCP traffic.

- The following filter displays packets that have the TCP ACK bit set but not packets that have the TCP SYN bit set.
  ```
  (tcp.flags.ack == 1) &&
  !(tcp.flags.syn == 1)
  ```
- The following filter displays ICMP Destination Unreachable packets that indicate the host is unreachable or the protocol is unreachable.
  ```
  (icmp.type == 3) && ((icmp.code == 0x01)
  || (icmp.code == 0x02))
  ```

There is another form of display filter—one that uses the offset and a value calculated from a specific point in a packet. These types of display filter use the same format as offset capture filters *proto[expr:size]*. These filters may not be used often, but knowing how to create one when you need it can save you loads of time.

```
eth.src[4:2] == 22:1b
ip[14:2] == 96:2c
```

These offset filters are discussed in *Filter on Specific Bytes in a Packet* on page 222.

Use Your Display Filters in Command Line Capture

If you know how to build display filters efficiently, those filters can be used with the `-R` parameter with Tshark for command-line capture. You can even use Tshark to read an existing trace file, apply a display filter and output to a new trace file using the `-r`, `-R` and `-w` parameters together. Using display filters with Tshark during a live capture does not limit the packets you are capturing; it only limits the packets you see. Using these display filters with Tshark on previously saved captures can allow you to create a subset of the original trace file however. For examples of using display filters with Tshark, refer to *Tshark Examples* on page 686.

Create Display Filters Using Auto-Complete

If you know the display filter syntax you want to use, you can type it directly into the display filter area. Wireshark has an auto-complete feature that helps you create your filters. For example, if you type in `tcp.` (be sure to include the period after `tcp`) as shown in Figure 126, Wireshark's auto-complete feature lists possible display filter values that could be created beginning with `tcp.`

Note that `tcp` without the period is a valid display filter (as noted by the green background), but `tcp` followed by a period is not a valid display filter—you must either complete the filter by removing the period or add remaining text to the display filter as shown in the drop down list. For more information on Wireshark's validity checks, refer to *Let Wireshark Catch Display Filter Mistakes* on page 223.

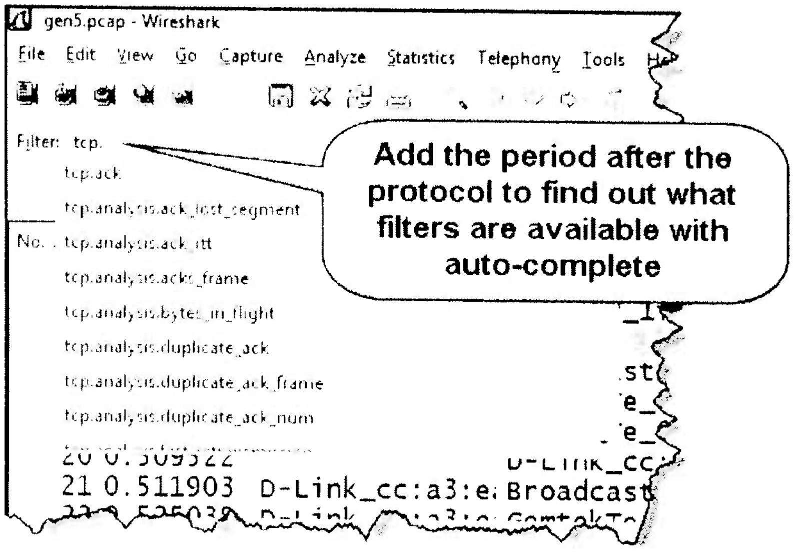

gen5.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: tcp.
tcp.ack
tcp.analysis.ack_lost_segment
tcp.analysis.ack_rtt
tcp.analysis.acks_frame
tcp.analysis.bytes_in_flight
tcp.analysis.duplicate_ack
tcp.analysis.duplicate_ack_frame
tcp.analysis.duplicate_ack_num
21 0.511903 D-Link_cc:a3:e: Broadcast
Add the period after the protocol to find out what filters are available with auto-complete

Figure 126. Wireshark's auto-complete feature helps you create valid filters

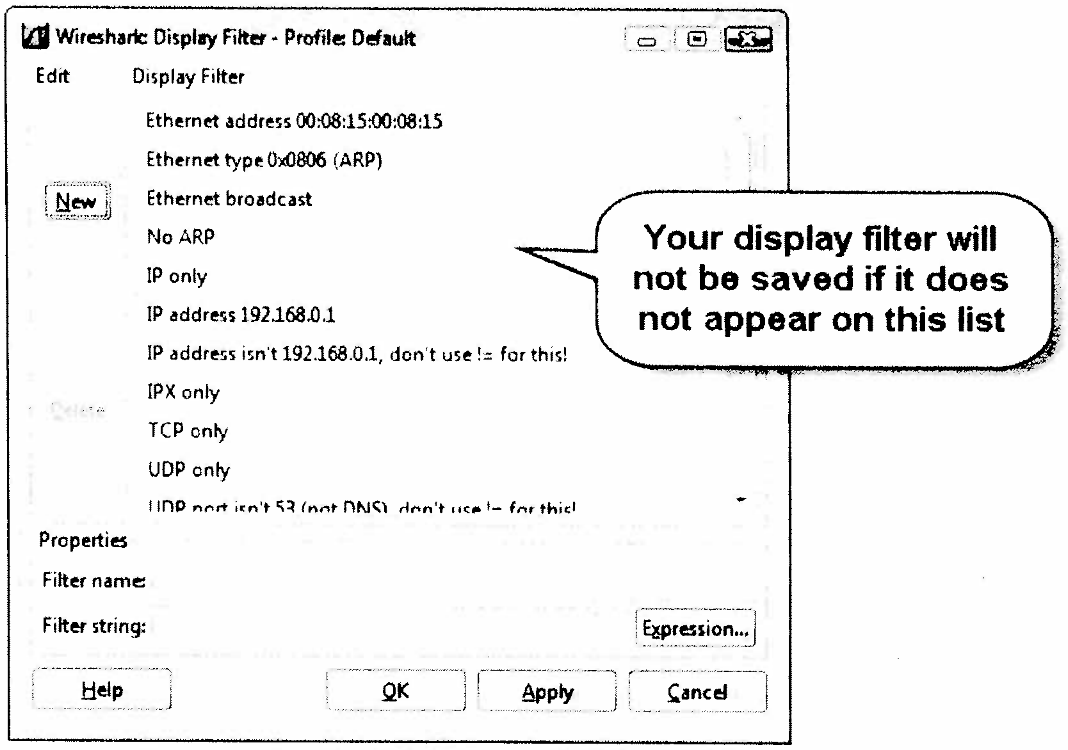
Apply Saved Display Filters

Click on the **Filter** button to the left of the display filter area to open the display filter window, shown in Figure 127. When you create filters that you want to use again, save them using the display filters window.

To create and save a new display filter, click **New** then enter the filter name and filter string. Wireshark supports error checking and auto-complete in the display filter box.

How to Ensure Your Display Filter is Saved

If you do not see your filter listed in the display filters list, your filter cannot be saved. You must click ***New*** *to create a new the filter. This is a common mistake people make when creating new display filters.*



Wireshark: Display Filter - Profile: Default

Edit

New

Delete

Display Filter

Ethernet address 00:08:15:00:08:15
Ethernet type 0x0806 (ARP)
Ethernet broadcast
No ARP
IP only
IP address 192.168.0.1
IP address isn't 192.168.0.1, don't use != for this!
IPX only
TCP only
UDP only
UDP port isn't 53 (not DNS), don't use != for this!

Properties

Filter name:

Filter string:

Expression...

Help | OK | Apply | Cancel

Your display filter will not be saved if it does not appear on this list

Figure 127. The display filter window

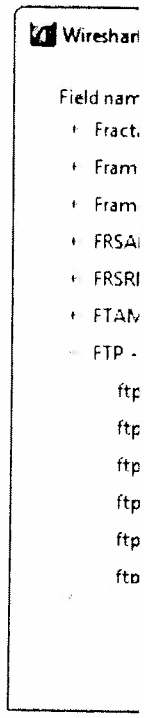
Use Expressions for Filter Assistance

In some cases you may want to make more complex filters, but you might not know the syntax. In addition, you might not be aware of the fields available for a specific type of communication. The Expression button is located to the right of the Display Filter field.

Expressions "walk you through" the filter creation process.

Some of the protocols and applications listed in the Filter Expression window include predefined values for individual fields. The FTP expression detail for `ftp.response.code` provides an example of a fully-defined expression as shown in Figure 128.

Expressions consist of field names, relations, values, pre-defined values (if available) and range. Selecting the "is present" relation simply builds a filter for the existence of the protocol, application or field. For example, selecting Mobile IP as the field name and "is present" in the relation area creates a `mip` filter that just looks for all Mobile IP traffic.



Wireshar
Field nam
Fract
Fram
Fram
FRSAI
FRSRI
FTAM
FTP -
ftp
ftp
ftp
ftp
ftp
ftp

Figure 128. Some

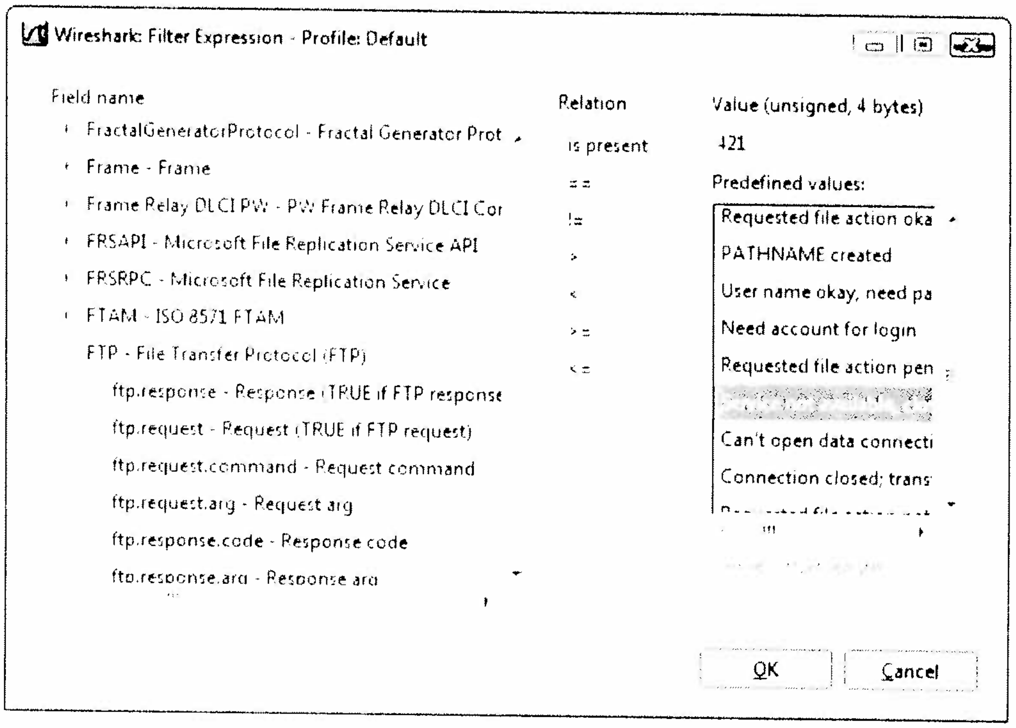
The following t

Display Filter
expert.seve
expert.mess
bootp.type
dns.flags.o

Make Di Filtering

You can use rig this technique in

You can right cl row that you rig Details pane. Ra **Apply as Filter**



Wireshark: Filter Expression - Profile: Default

Field name
- FractalGeneratorProtocol - Fractal Generator Prot
- Frame - Frame
- Frame Relay DLCI PW - PW Frame Relay DLCI Cor
- FRSAPI - Microsoft File Replication Service API
- FRSRPC - Microsoft File Replication Service
- FTAM - ISO 8571 FTAM
- FTP - File Transfer Protocol (FTP)
  - ftp.response - Response (TRUE if FTP response
  - ftp.request - Request (TRUE if FTP request)
  - ftp.request.command - Request command
  - ftp.request.arg - Request arg
  - ftp.response.code - Response code
  - ftp.response.arg - Response arg

Relation
- is present
- ==
- !=
- >
- <
- >=
- <=

Value (unsigned, 4 bytes)

421

Predefined values:
- Requested file action oka
- PATHNAME created
- User name okay, need pa
- Need account for login
- Requested file action pen
- Can't open data connecti
- Connection closed; trans

Figure 128. Some expression fields have pre-defined values

The following table provides examples of display filters created with Expressions:

Display Filter	Expression Path
`expert.severity == 1536`	Expert | Expert Severity | == | Warn
`expert.message`	Expert | Expert Message | is present
`bootp.type == 1`	BOOTP or DHCP | bootp.type | == | Boot Request
`dns.flags.opcode == 1`	DNS | dns.flags.opcode | == | Inverse Query

Make Display Filters Quickly Using Right-Click Filtering

You can use right-click filtering in the Packet List pane and the Packet Details pane. You cannot use this technique in the Packet Bytes pane.

You can right click on the Packet List pane and prepare or apply a filter based on the column and row that you right clicked on. You can also right click on a field or summary line in the Packet Details pane. Rather than type out a field value, right click on the field of interest and select either **Apply as Filter | Selected** or **Prepare a Filter | Selected**.

Apply as Filter

Use Apply as Filter to apply the filter immediately. You can edit the filter after it has been applied or expand the filter by using this technique and specifying one of the other filter options such as:

• Not Selected	(create an exclusion filter based on the selection)
• ... and Selected	(must match existing filter AND the selection)
• ... or Selected	(must match either existing filter OR the selection)
• ... and Not Selected	(must match existing filter AND NOT selection)
• ... or Not Selected	(must match either existing filter OR NOT selection)

Be careful—if you choose Apply as | Selected again or choose Apply as | Not Selected you will replace your original filter with the current field name and value. These two options replace anything already shown in your display filter window.

If you choose another option, an operator (&& or || or != or !) is placed after the existing filter portion and the field and value selected will be appended to the existing filter. For more information on display filter operators, refer to *Combine Display Filters with Comparison Operators* on Page 220.

For example, if you already have `arp` in your display filter window when you click on a source MAC address and select ...and Not Selected, your filter would display all ARP packets except those with the MAC address selected.

Prepare a Filter

Right click on a field and select **Prepare a Filter** to create a filter, but not apply it immediately. This process is useful for creating longer, more complex filters with numerous operators. For example, if you wanted to build a filter on ICMP Destination Unreachable/Port Unreachable packets, you could select the ICMP type value of 3 first and then select the ICMP code value of 3 using the **...and Selected** operation. You can edit your filter before applying it if necessary.

Copy | As Filter

One of the newer additions to Wireshark's display filter creation process is the right click ability to **Copy** | **As Filter**. Using this method, you can right click a field in either the Packet List pane or a field in the Packet Details pane and buffer a display filter based on that field using this copy feature. This technique is very useful for creating coloring rules, building more complex display filters or copying filters between Wireshark instances. (Thanks, Sake Blok, for this feature!)

Filter on Conversations and Endpoints

You can create a filter based on the conversations or endpoints window contents. Right click on a conversation of interest and select either **Prepare a Filter** or **Apply as Filter**. As shown in Figure 129, when creating filters based on a conversation, you are prompted for the direction of travel in addition to the basic filter type. The directions are based on the Address A and Address B column titles in the conversation window.

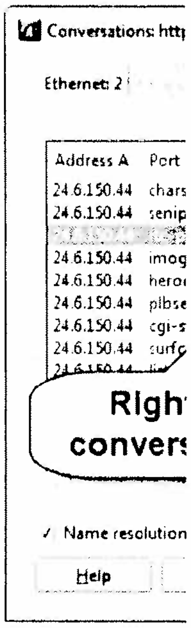


Figure 129. Create

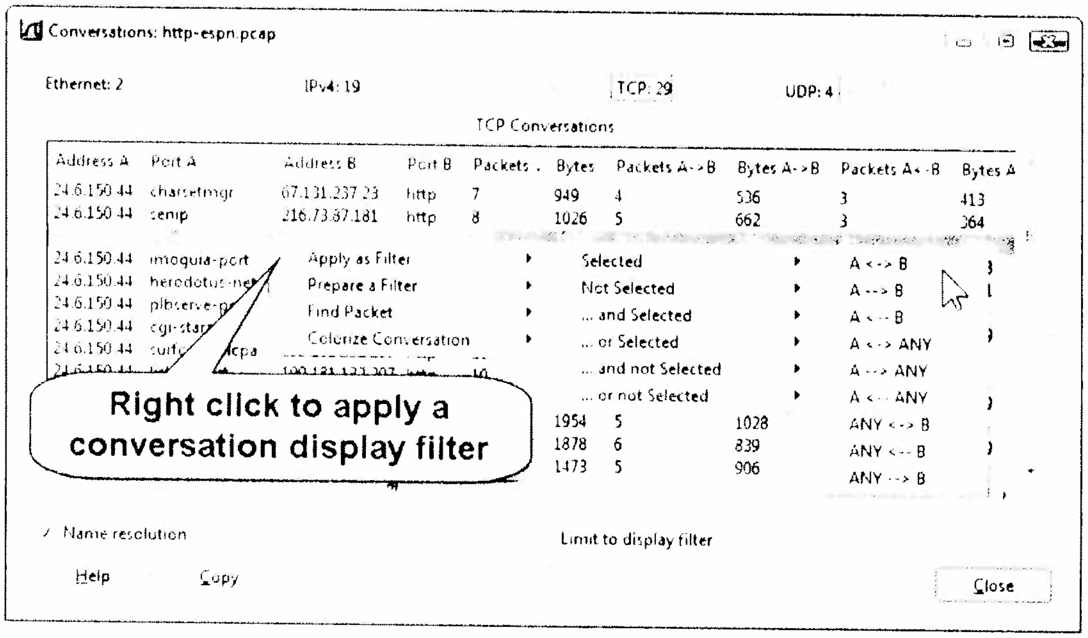
You can use the
exception—the e

Understa

Display filters sy

Every field show
simply a packet c
a field in the Pac
Figure 130 we se
that we know the

Figure 130. The fie


Conversations: http-espn.pcap

Ethernet: 2 IPv4: 19 TCP: 29 UDP: 4

TCP Conversations

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A
24.6.150.44	charsetmgr	67.131.237.23	http	7	949	4	536	3	413
24.6.150.44	cenip	216.73.87.181	http	8	1026	5	662	3	364
24.6.150.44	imoquia-port								
24.6.150.44	herodotus-net								
24.6.150.44	plbserve-p								
24.6.150.44	cgi-star								
24.6.150.44	suit								
					1954	5	1028		
					1878	6	839		
					1473	5	906		

Figure 129. Create a bidirectional display filter based on a conversation

You can use the same steps to create display filters based on the endpoints window with one exception—the endpoints window does not offer an option to define the direction of the traffic.

Understand Display Filter Syntax

Display filters syntax is used to create display filters and coloring rules.

Every field shown in the Packet Details pane (whether that field actually exists in a packet or is simply a packet characteristic, such as a retransmission) can be used to create these filters. Highlight a field in the Packet Details pane and the related display filter value is shown in the status area. In Figure 130 we selected the TCP window size field. The field name is `tcp.window_size`. Now that we know the field name, we can create a `tcp.window_size == 65535` filter.

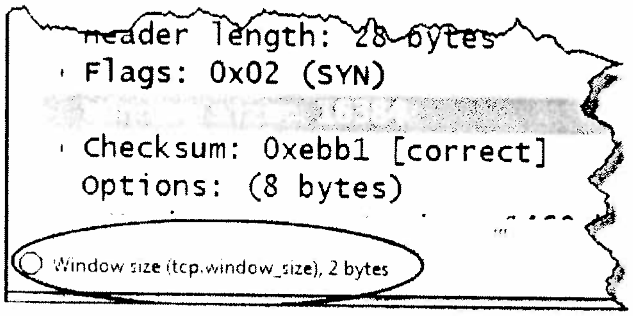
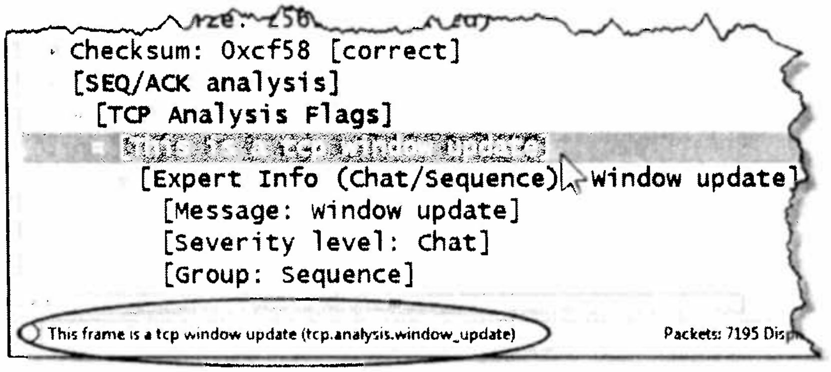

```
header length: 28 bytes
Flags: 0x02 (SYN)
Checksum: 0xebb1 [correct]
Options: (8 bytes)
```
Window size (tcp.window_size), 2 bytes

Figure 130. The field name selected in a packet is shown in the Status bar

As mentioned earlier, you can create display filters on packet characteristics as opposed to actual fields. In Figure 131 we selected the TCP analysis line stating *"[This is a tcp window update]"*. The display syntax for all TCP window update packets is `tcp.analysis.window_update`. You can right click and apply a filter for TCP window update packets even though this field does not exist.



```
  Checksum: 0xcf58 [correct]
  [SEQ/ACK analysis]
    [TCP Analysis Flags]

      [Expert Info (Chat/Sequence): Window update]
        [Message: Window update]
        [Severity level: Chat]
        [Group: Sequence]

This frame is a tcp window update (tcp.analysis.window_update)          Packets: 7195 Dis
```

Figure 131. Not all fields displayed actually exist in a packet

Combine Display Filters with Comparison Operators

Comparison and logical operators enable you to combine multiple filters to further define the traffic of interest and offer a negative operand to filter out undesired traffic (exclusion filtering).

Description	Symbol	Text
equal to	==	eq
or	||	or
and	&&	and
greater than	>	gt
less than	<	lt
greater than or equal to	>=	ge
less than or equal to	<=	le
not	!	not
not equal to	!=	ne
contains		contains
matches		matches

Understand Wireshark Warnings on Using !=

*Wireshark colorizes the display filter area in yellow whenever you use the != operator. It doesn't mean your filter won't work—it's just a warning that it **may not** work. See Let Wireshark Catch Display Filter Mistakes on page 223 for more details.*

You can crea
typing in the

Display Filte

ip.addr =

!arp && !

bootp ||

tcp conta

dns.count

tcp match

You can use c
show) or excl

The majority
however. The
field.

Check **Help |**
"matches" ope

The following

This filter exa
the string "gl=

In the Practice
using "matche

You can create display filters with operators using the right-click method, expressions or just by typing in the filter. The following provides examples of various display filters using operators:

Display Filter	Description
`ip.addr == 10.2.3.4 && http`	Only display HTTP traffic to or from 10.2.3.4
`!arp && !icmp`	Display all traffic *except* ARP and ICMP traffic
`bootp || dns`	Only display BOOTP/DHCP or DNS traffic
`tcp contains "PASS"`	Only display packets that have the ASCII string "PASS" in the TCP segment
`dns.count.answers > 2`	Only display DNS responses that contain more than two answers
`tcp matches "zip"`	The TCP stream includes the text value "zip." This is a great filter if you are looking for HTTP downloads of compressed files. Consider using "exe" as the target content (note that "zip" may be included in the Accept-Encoding HTTP modifier)

You can use operators to make more complex inclusion display filters (indicating traffic you want to show) or exclusion display filters (traffic you want to hide).

The majority of the comparison operators are relatively intuitive—the **matches** operator is not, however. The *matches* operator is used with Perl regular expressions to search for a string within a field.

Check **Help | About Wireshark.** If the About box says "NOTE: this build doesn't support the "matches" operator for Wireshark filter syntax" then "matches" isn't supported.

The following is an example out of the wiki page (*wiki.wireshark.org/DisplayFilters*):

```
http.request.uri matches "gl=se$"
```

This filter examines the end (as denoted by the "$") of the URI Request line in an HTTP packet for the string "gl=se".

In the Practice What You've Learned section at the end of this chapter you will have a chance to test using "matches" on a trace file.

Alter Display Filter Meaning with Parentheses

Use parentheses to have the conditions evaluated in a specific order. For example, the two filters shown in the next table have different interpretations based on the parentheses set:

Filter with Parentheses	Interpretation
`(ip.src==192.168.0.105 and udp.port==53) or tcp.port==80`	DNS/port 53 traffic from 192.168.0.105 **plus** all HTTP/port 80 traffic on the network
`ip.src==192.168.0.105 and (udp.port==53 or tcp.port==80)`	DNS/port 53 **or** HTTP/port 80 traffic from 192.168.0.105

Filter on Specific Bytes in a Packet

Offset filters are also referred to as Subset Operators. These filters define a frame element, an offset, length (optional), operator and value. You can use these filters when no simpler filter method is available. For example, if you want to filter on Ethernet source addresses that end with a specific two-byte value, use an offset filter.

An example of an offset display filter is shown below.

```
eth.src[4:2] == 22:1b
```

The display filter shown above begins looking at the Ethernet Source Address field in a frame then counts over 5 bytes (we begin counting with zero) and looks for the two-byte value 0x221b. This means we are looking at the last two bytes in the Ethernet source field for the value 0x221b.

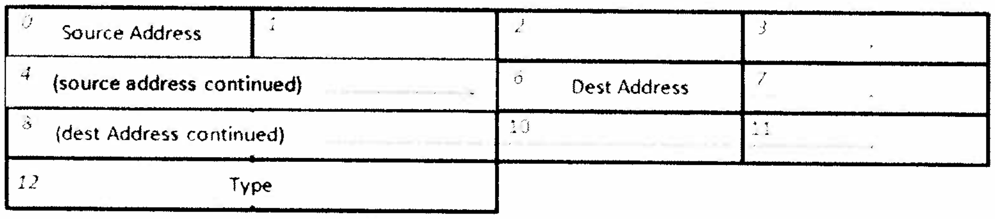


Figure 132. The filter `eth.src[4:2]` *looks at the last two bytes of the Ethernet Source Address field*

Another example of an offset filter is shown below.

```
ip[14:2] == 96:2c
```

This filter looks at the 15th and 16th bytes of the IP header (the last two bytes of the source IP address) for the value 0x962c (this would equate to a source IP address ending in 150.44). Figure 133 shows the breakdown of an IP header. Remember that the value [14:2] means we count over 15 bytes (start counting at 0) and look for a two-byte value.

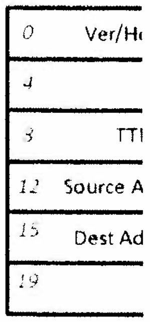


Figure 133. The fil address field

The need to creat Wireshark. There fields for a partia

Let Wire

Wireshark contai filter mistakes su *Display Filter M*

•

•

•

Not all display fi consider the filte It can't.

Use Disp

Display filters m **Display Filter m**

Display filter ma create display filt saved in the asso "name", "filt

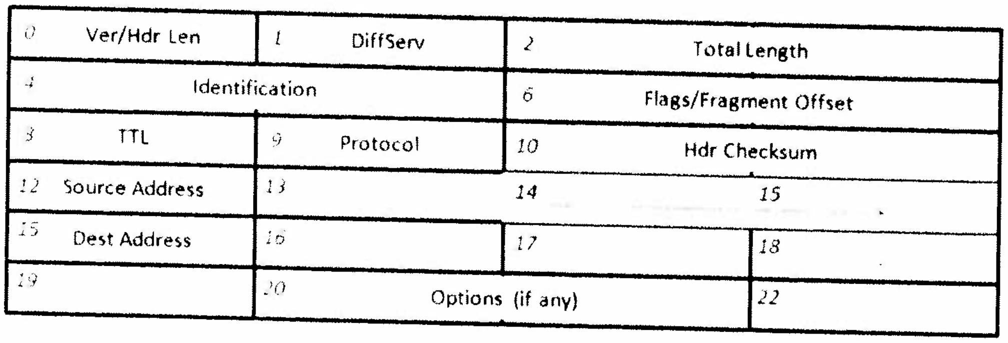


Figure 133. The filter `ip[14:2]` *looks at the 15th and 16th bytes(start counting at 0) in IP header source address field*

The need to create offset filters has been reduced because of the number of filters built into Wireshark. There are still times, however, when you need to use these offset filters to look inside fields for a partial value match.

Let Wireshark Catch Display Filter Mistakes

Wireshark contains error checking to help you avoid syntax problems and even practical display filter mistakes such as `ip.addr != 10.2.4.1`. This mistake is defined in *Avoid Common Display Filter Mistakes* on page 224.

- A green background indicates the filter syntax is correct and logical.
- A yellow background indicates the syntax is correct, but it may not be logically correct. For an example of a filter that would be colored yellow, see *Avoid Common Display Filter Mistakes* on page 225.
- A red background indicates a syntax error. Filters marked with a red background will not process correctly.

Not all display filter mistakes are caught by Wireshark's error checking mechanism. For example, consider the filter `http && arp`. How can a packet be both an HTTP packet *and* an ARP packet? It can't.

Use Display Filter Macros for Complex Filtering

Display filters macros are used to create shortcuts for more complex display filters. Select **Analyze | Display Filter macros | New** to create a new macro.

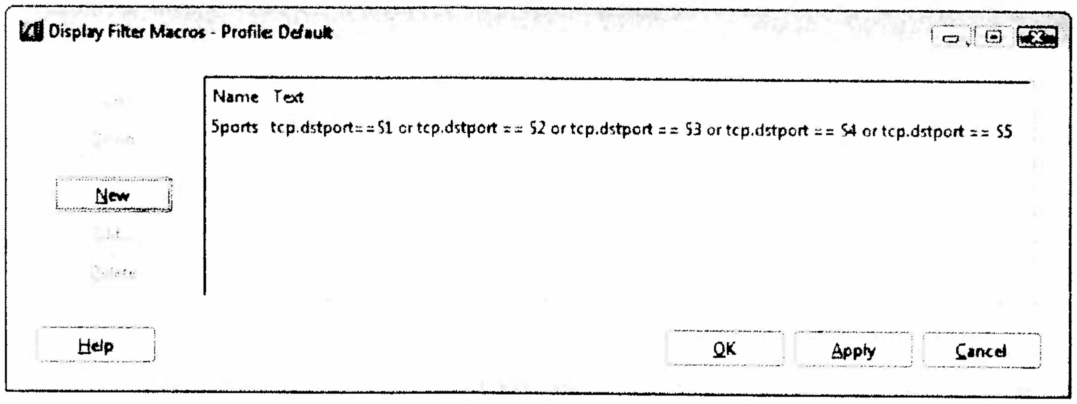
Display filter macros are saved in *dfilters_macros* in your Personal Configuration folder. If you create display filter macros under a profile other than the default profile, the *dfilters_macros* file is saved in the associated profile's directory. The syntax of this file is `"name","filter_string"`.

To create a display filter macro, first you must name the macro. You must use this name to call the macro in the display filter window. Figure 134 shows a display filter macro used to view traffic destined to five ports. Without using a display filter macro, this display filter syntax would be `tcp.dstport == 5600 || tcp.dstport == 5603 || tcp.dstport == 6400 || tcp.dstport == 6500 || tcp.dstport == 6700`.

The macro shown in Figure 134 is named "5ports". Dollar signs precede the variable numbers. The syntax to use this macro would be `${5ports:5600;5603;6400;6500;6700}`.

When we run this display filter macro, the five port variables will be substituted as follows:

```
$1    tcp.dstport == 5600
$2    tcp.dstport == 5603
$3    tcp.dstport == 6400
$4    tcp.dstport == 6500
$5    tcp.dstport == 6700
```



Name	Text
5ports	tcp.dstport==$1 or tcp.dstport == $2 or tcp.dstport == $3 or tcp.dstport == $4 or tcp.dstport == $5

Figure 134. Display filter macros provide shortcuts for more complex filters

Another example of a time saving display filter macro would be one that focuses on a specific conversation—we'll call this macro "tcp_conv". The display filter macro syntax would be:

```
(ip.src == $1 and ip.dst == $2 and tcp.srcport == $3 and
tcp.dstport == $4) or (ip.src == $2 and ip.dst == $1 and
tcp.srcport == $4 and tcp.dstport == $3)
```

In the example provided above, the display filter macro focuses on a specific conversation based on IP addresses and TCP port numbers. To run the macro, we would use the following command in the display filter window:

```
${tcp_conv:192.168.1.1;192.168.1.99;1201;2401}
```

You can share display filter macros by simply copying the *dfilters_macros* file from your Personal Configuration folder or profile folder to another Wireshark system.

Avoid C

One of the mos
mostly seen wh

Many people a
contain the IP a
enter ip.add
10.2.4.1. This f

The filter ip.a
ip.addr fiel
packet, howeve
two fields. Firs
looks at the des

The table belov
one of the IP ac

Source IP
10.2.4.1 (nc
10.99.99.99
10.2.4.1 (nc
10.2.2.2 (

The correct filt

Manuall

You can add fil
dfilters file is lc
and a copy is pl

The syntax of tl

"fil

The *dfilters* file
filter entry or it

The advantage
indenting and ti

Avoid Common Display Filter Mistakes

One of the most common filter mistakes involves the use of the `!` or `not` operand. This problem is mostly seen when filtering out traffic to or from an IP address or port number.

Many people are familiar with the `ip.addr==10.2.4.1` syntax for displaying packets that contain the IP address 10.2.4.1 in either the source or destination IP address field. Naturally, they enter `ip.addr != 10.2.4.1` to try to view all packets *except* ones that contain the address 10.2.4.1. This filter structure does not work, however.

The filter `ip.addr != 10.2.4.1` actually means you are looking for a packet that has an `ip.addr` field that contains a value other than 10.2.4.1. There are two IP address fields in the packet, however and this filter will allow a packet to be displayed if it has 10.2.4.1 in *either* of those two fields. First Wireshark looks at the source IP address field to see if the filter matches. Next it looks at the destination IP address field.

The table below shows how packets are examined. Using the filter `ip.addr != 10.2.4.1`, if one of the IP addresses matches the filter then the packet will be displayed.

Source IP Address	Destination IP Address	Show Packet?
10.2.4.1 (no match)	255.255.255.255 (match)	Yes
10.99.99.99 (match)	10.2.4.1 (no match)	Yes
10.2.4.1 (no match)	10.99.99.99 (match)	Yes
10.2.2.2 (match)	10.1.1.1 (match)	Yes

The correct filter syntax is `!ip.addr==10.2.4.1`. Place the `!` or not before `ip.addr`.

Manually Edit the *dfilters* File

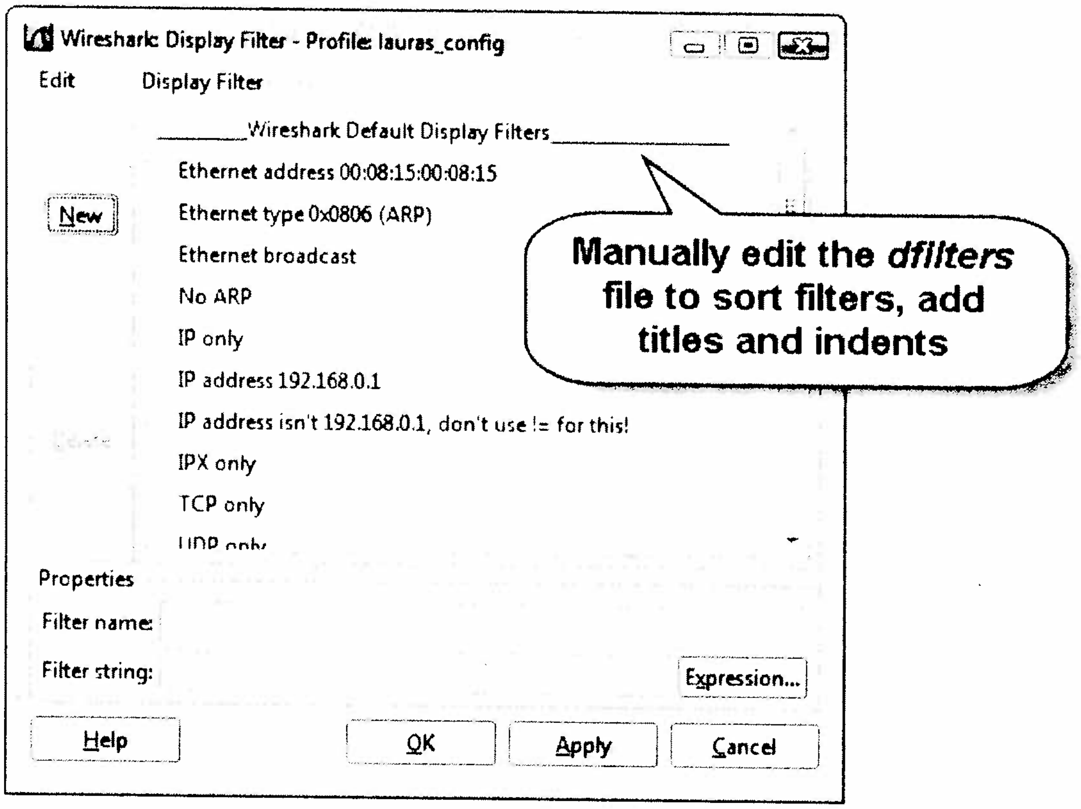
You can add filters through the Wireshark GUI interface or edit the *dfilters* file directly. The default *dfilters* file is located in the Global Configurations directory. New filters are added to the *dfilters* file and a copy is placed in the Personal Configurations folder or in the current profile directory.

The syntax of the *dfilters* file is:

```
    "filter name" filter string
```

The *dfilters* file does not have an extension and you must include a new line after the last display filter entry or it will not show up in the Wireshark display filter list.

The advantage of manually editing the *dfilters* file is the ability to reorder the display filters and add indenting and titles to your display filters list as shown in Figure 135.

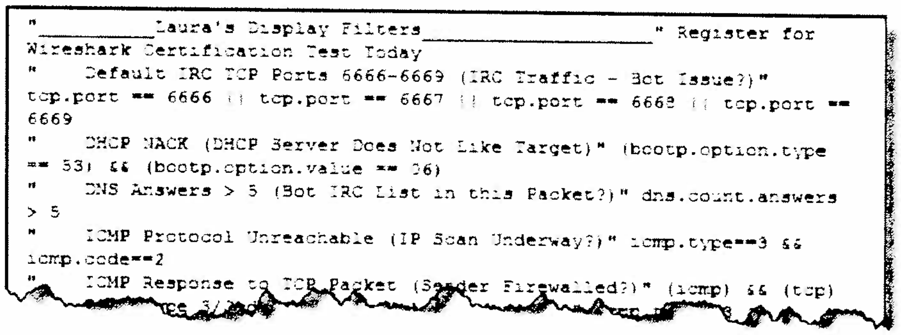

Wireshark: Display Filter - Profile: lauras_config
Edit
Display Filter
New
Properties
Filter name:
Filter string:
Expression...
Help
OK
Apply
Cancel
Manually edit the *dfilters* file to sort filters, add titles and indents

```
________Wireshark Default Display Filters________________
    Ethernet address 00:08:15:00:08:15
    Ethernet type 0x0806 (ARP)
    Ethernet broadcast
    No ARP
    IP only
    IP address 192.168.0.1
    IP address isn't 192.168.0.1, don't use != for this!
    IPX only
    TCP only
    UDP only
```

Figure 135. Edit the dfilters file to organize your personal display filters

To add the title and indents to your display filters list, manually edit the *dfilters* file and put underscores and spaces inside the quotes used around the name of the display filter, as shown in Figure 136.

The Download section of the book website, *www.wiresharkbook.com*, contains the *dfilters* file shown in Figure 136.



```
"__________Laura's Display Filters____________________" Register for
Wireshark Certification Test Today
"    Default IRC TCP Ports 6666-6669 (IRC Traffic - Bot Issue?)"
tcp.port == 6666 || tcp.port == 6667 || tcp.port == 6668 || tcp.port ==
6669
"    DHCP NACK (DHCP Server Does Not Like Target)" (bootp.option.type
== 53) && (bootp.option.value == 06)
"    DNS Answers > 5 (Bot IRC List in this Packet?)" dns.count.answers
> 5
"    ICMP Protocol Unreachable (IP Scan Underway?)" icmp.type==3 &&
icmp.code==2
"    ICMP Response to TCP Packet (Sender Firewalled?)" (icmp) && (tcp)
```

Figure 136. An edited dfilters file

Using di
finding t
on the "r

The case
concerns

Using display filters helps avoid the "needle in the haystack issue" and speeds up the process of finding the cause of network problems and identifying unusual traffic patterns. For more information on the "needle in the haystack issue," see *Overcome the "Needle in the Haystack Issue"* on page 9.

The case studies at the end of this chapter provide examples of using display filters to solve network concerns and perform application analysis.

Case Study: Using Filters and Graphs to Solve Database Issues

***Submitted by:* Coleen D.**
Network Analyst

There appeared to be way too many connections to our documentation server at specific times during the day. The server administrators thought someone was attacking the server and they wanted to know how many active connections had been established to the server throughout the day and by whom.

I ended up using a display filter for the third packet of the TCP handshake to catch all successful connections and plotting this on an IO Graph. My filter is shown below.

```
(tcp.flags == 0x10) && (tcp.seq == 1) &&
(tcp.ack == 1)
```

The first part of my filter looked for packets that had just the ACK bit set in the TCP header. The second part looked for the TCP Sequence Number field set to 1 and the third part looked for the TCP Acknowledgment Number field set to 1.

We always have "relative sequence numbering" enabled in Wireshark's TCP preferences (otherwise this wouldn't work) and these field values are always seen in the third packet of the TCP handshakes.

To see these connections, I put this display filter in the red graph line in the IO Graph and used the Fbar format so it really showed up.

Sure enough, we did find that the connections spiked around 2pm each day.

Interestingly, it was one of the documentation server administrator machines that made over 1,000 connections to their own server around that time. It turned out someone in their group was testing out a new document management package that flooded the documentation server with connections every time they ran it.

We could easily show the source of the connections and recommend against the lousy program they were about to buy!

We saved the company a ton of money and headache using Wireshark!

Case Study: The Chatty Browser

To analyze Twitter traffic, I created a filter for all traffic to/from my IP address (ip.addr==192.168.0.106) and then filtered out any of my unrelated traffic—the idle traffic and the background traffic sent when my browser connected to Web of Trust or other sites that had nothing to do with the Twitter communications.

I was working backwards and separating out my Firefox traffic and any other noise that my host generates without my interaction. I created a number exclusions to my display filter as I identified my background traffic to my printer, my router's management port, DHCP noise, ARP noise, traffic from my iPhone (which was being bridged onto the wired network), Google Analytics and Google Malware updates from Firefox, World News and BBC background feeds from Firefox and anything else not related to my Twitter communications.

When my convoluted display filter was completed, I could see no background traffic from superfluous processes.

My final display filter was extremely long:

```
ip.addr==192.168.0.106  && !srvloc && !dns &&
!ip.addr==74.6.114.56 && !ip.addr==239.255.255.250 &&
!ip.addr==96.17.0.0/16 && !ip.addr==192.168.0.102 && !smb
&& !nbns && !ip.addr== 192.168.0.103 &&
!ip.addr==64.74.80.187 && ! ip.addr==83.150.67.33 &&
!ip.addr==67.217.0.0/16 && !ip.addr==66.102.7.101 &&
!ip.addr==216.115.0.0/16 && !ip.addr==216.219.0.0/16 &&
!ip.addr==69.90.30.72
```

Although I started out analyzing Twitter traffic, I ended up finding out that all the plugins we added to Firefox made our browsers way too chatty—they were talking all the time.

We temporarily turned on network name resolution in Wireshark to make it easier to find out who the plugins were talking to. It made Wireshark really slow when we opened the Conversation and Endpoint statistics, but we could easily spot the plugin traffic by the targets.

We ended up uninstalling some of the plugins that were talking all day long in the background. We didn't need them and they just added too much garbage to the network.

Case Study: Catching Viruses and Worms

Submitted by: Todd Lerdal

Computer viruses and worms were a great learning time for me with packet analysis. I was very new at packet analysis and would just fire off traces on a VLAN to get a "feel" of what was running on my network. "Unofficial base-lining" is probably a better description—never documented anything other than getting an idea in my head of what was normal.

I knew the sorts of applications that I should expect to see, NCP, Web, Telnet, Citrix, etc. If there was something out there I didn't recognize, I'd filter down to it just to get a better understanding of "should this be running?"

Then, the worms hit.

I spent many hours/days with a monitor session on our server VLAN just watching how worms would spread to help identify, isolate, and inoculate infected workstations.

It doesn't take long once you start watching to see the unusual traffic on your LAN. What I would see is what appeared to be ping sweeps or port scans coming from multiple hosts.

Once I'd captured enough packets I was able to then build better display filters to identify just these sweeps so that I could then isolate the infected workstations and help the desktop and server teams to go clean these devices before allowing them back on the network.

With practice, it didn't take me long to generate lists of IP addresses or device names to provide the desktop folks so they could start cleaning.

Summar

Display filter
Display filter
which use th

Wireshark pr
red = incorre
Expressions

One of the fa
Filter or Prep
be careful of
meaning.

Display filter
file. You can

The
boo
in F
Wireshark ar
Personal Cor
more inform

Practice

The followin
chapter.

http-aol.pca

app-norton-

Summary

Display filters are used to focus on specific packets, protocols, conversations or endpoints of interest. Display filters use the special Wireshark syntax—they are not interchangeable with capture filters which use the BPF filter syntax (also used by tcpdump).

Wireshark provides automatic error checking of your display filter syntax (green = correct syntax, red = incorrect syntax, yellow = may yield unexpected results). You can also use Wireshark's Expressions to create filters using predefined fields and field values.

One of the fastest ways to create a display filter is to right click on a field and select either Apply as Filter or Prepare a Filter. You can use comparison operators to combine multiple display filters, but be careful of the parentheses in your filters. The location of parentheses can alter a display filter's meaning.

Display filters are saved in the *dfilters* file and can be edited through the GUI or directly in the text file. You can share your display filters by simply sending someone a copy of your *dfilters* file.

There are numerous Wireshark display filters contained in the Downloads section of the book website, *www.wiresharkbook.com*. One of the *dfilters* files available online is shown in Figure 136. This *dfilters* file includes the default set of display filters released with Wireshark and 15 additional display filters. To use this *dfilters* file, simply copy the file into your Personal Configuration folder or create a new profile and copy this file into the profile's folder. For more information on creating a new profile, refer to *Chapter 11: Customize Wireshark Profiles*.

Practice What You've Learned

The following table lists several trace files to use when practicing what you've learned in this chapter.

http-aol.pcap	It takes 18 different TCP connections to load the *www.aol.com* site. Have you analyzed the connection to your corporate website lately? Use `http.request.uri matches "laptop$"` as the display filter. Another example of using the matches operator is `http.request.uri matches "&+"`. This filter examines the URI field for the value "&" and display the packet if the value is found 1 or more times as denoted by the +.
app-norton-update2.pcap	This Symantec update process doesn't seem to work very well. Consider building a filter for `http.response.code == 404` and note the number of "File Not Found" responses. Now compare these two display filters and examine the difference in your results: (a) `!http.response.code == 404` (b) `http.response.code !=404`

ftp-crack.pcap

Apply the following display filter to the traffic:

```
ftp.request.command == "USER" ||
ftp.request.command == "PASS"
```

This reveals that the password cracking attempt is only focused on the admin account and the passwords are coming from a dictionary that includes names. Looks like they are cycling through the password list—we caught them on the letter M, but they start at the beginning later (packet 4739).

Review

Q9.1 W

Q9.2 W

Q9.3 W

Q9.4 Wl

(ip.

ip.s