Wireshark & TCPDump Study Guide

Contents

Overview: Capture filters versus display filter

A Capture filter in Wireshark specifies which packets you would like to capture and keep. A filter of this type needs to be set before you start capturing anything, and it can not be edited during a capture session. Any packets that do not make it through this filter are essentially lost.

Display filters on the other hand, enable filtering of packets shown in the Wireshark UI. A packet that matches a capture filter, but filtered away by a display filter will still be kept; it just won't be visible so long as the display filter is active.

Note: The display filters syntax can be used to apply Wireshark coloring rules.

Display filters

As mentioned, display filters are used to reduce the number of packets visible in GUI, as well as for applying coloring rules. Every field under Packet Details can be used for filtering, and the comparison operators listed below can be used. The filter itself is applied at the top of the Wireshark window:

Operators

These are logical and comparison-operators for use in display filters. Either the operators in the first column, or the English word in the second column can be used in filter expressions:

Operator	English	Meaning
==	eq	Test for equality
!=	ne	Test for "not equal to"
<	gt	Greater than
>	lt	Less than
>=	ge	Greater than or equal to
<=	le	Less than or equal to
Contains	contains	Search for contained value
~	matches	Match using Perl regular expressions
&	bitwise_and	Bit field value comparisons
&&	And	Both expressions must be true
||	or	At least one expression must be true
^^	xor	Logial Xor
!	not	Logical Not
[...]		The *slice*-operator; Enables selection of a sub-sequence / a part of a sequence.

In {}		*Membership; t*est if a value is represented in a set of values.

Types

Each field has a specific type, which has implications for how you can use it. You can find a list of available fields for different protocols, and their types in the Display Filter Reference (see "more information" below).

Available types are:

- Text strings.
- Integers (signed or unsigned). These can be compared to a value declared in decimal, octal, or hexadecimal.
- Boolean: If the code for a flag is used as a filter, only packets which contain that flag will be visible (whether or not the flag is actually set).
- Ethernet addresses
- IPv4 addresses (including a subnet-part)

Functions

These allow for manipulation of field values before a comparison is performed

Function	Effect
Upper	Converts all characters in a string field to uppercase
Lower	Converts all characters in a string field to lowercase
Len	Returns the byte length of a field
Count	Returns the number of occurrences of a field in a frame
String	Converts a field into a string, allowing for string comparisons.

Examples

Filter for a specific IP, for both source and destination:

```
ip.addr == 10.43.54.65
```

Filter for a specific IP, but only for source:

```
ip.src == 10.43.54.65
```

Filter for a specific IP, but only for destination:

```
ip.dst == 10.43.54.65
```

Filter away any traffic to or from a specific IP (i.e. packets with either *src* or *dst* IP matching):

```
 ! ( ip.addr == 10.43.54.65 )
```

Show only SMTP (port 25) and ICMP traffic:

```
tcp.port eq 25 or icmp
```

Show only traffic in the LAN (192.168.x.x), between workstations and servers -- no Internet::

```
ip.src==192.168.0.0/16 and ip.dst==192.168.0.0/16
```

Match packets containing the (arbitrary) 3-byte sequence 0x81, 0x60, 0x03 at the beginning of the UDP payload, skipping the 8-byte UDP header. :

```
udp[8:3]==81:60:03
```

Using slice to filter on the vendor identifier part (OUI) of the MAC address to only packets from a specific device manufacturer:

```
  eth.addr[0:3]==00:06:5B
```

Match packets that contains the 3-byte sequence 0x81, 0x60, 0x03 anywhere in the UDP header or payload:

```
  udp contains 81:60:03
```

Match packets where SIP To-header contains the string "a1762" anywhere in the header:

```
sip.To contains "a1762"
```

Filter for packets with a destination IP matching one of three in a set:

```
ip.dst in {224.0.0.251 224.0.0.2 224.0.0.251}
```

Use RegEx to filter for HTTP requests where the last characters in the uri are "gl=se":

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

More information

The examples above show some of what can be achieved using display filters. There are is a lot more available at wireshark.org, including a detailed Display Filter Reference found at the following url:

- https://www.wireshark.org/docs/dfref/

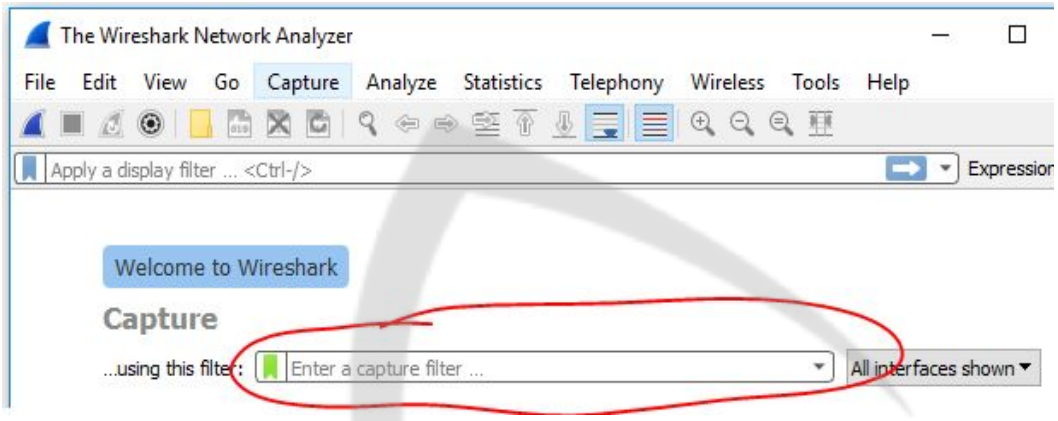
This includes a list of all the available fields for the different protocols, as well as information about their datatypes and a description. For more general information, see please refer to the documentation:

- https://wiki.wireshark.org/DisplayFilters
- https://www.wireshark.org/docs/man-pages/wireshark-filter.html

Capture Filters

While display filters are used to hide captured packets, Capture Filters actually limit the packets that are captured in the first place. A capture filter must be applied before a capture session is started, and can not be changed during the capture.

A capture filter can be entered right at the start, after opening Wireshark:



BPF Syntax overview

A Wireshark capture filter is declared using the *Berkeley Packet Filter* syntax (BPF), and will typically consists of one or more *Primitives*.

A *primitive* consists of one or more *qualifiers* followed by an *id.* The *id* can be a name or number, such as a hostname, an IP address, or a range of ports, for instance. The *qualifier* specifies which properties of the *id* should be taken into account by the filter.

Several primitives can be combined using ***and*** (or "&&"), ***or*** (or "||"), or ***not*** (or "!").

Note: "***not***" or "***!***" has a higher precedence than ***and*** and ***or.***

A primitive with two qualifiers, and an IP address for an id:

```
src host 192.168.10.152
```

Two primitives combined using *and*:

```
dst host 192.168.10.152 and port 51020
```

Qualifiers

There are three different kinds of *qualifiers*:

Kind	Available qualifiers	Description
type	host, net , port, portrange	These declare what the *id* is pointing to; e.g. an ip address, a hostname, or a range of ports. If no type qualifier is used, *host* is assumed.
dir	src, dst, src or dst, src and dst, ra, ta, addr1, addr2, addr3, addr4	These declare the direction to or from the *id* (source or destination). **Note:** *ra, ta,* and *addr** are only for IEEE 802.11 Wireless LAN link layers.
proto	ether, fddi, tr, wlan, ip, ip6, arp, rarp, decnet, tcp, udp	If used, limits the filter to the declared protocol(s). If no protocol is specified, all will be accepted. **Note:** *ether, tr, wlan* and *fddi* are effectively aliases here: *ether* refers to the data link layer of a network; *fddi* is the same, only for fiber-optic lines, while *wlan* and *tr* refer to Wireless LAN's and Token Rings, respectively. The differences between these do not affect filtering here.

Special Keywords

In addition to the qualifiers above, there are a few special keywords:

Gateway: Checks if a package uses a specified host as gateway

```
gateway hostname
```

Broadcast: Checks if a package is a broadcast packet. This can be either an *ethernet* broadcast packet or an *IP* broadcast packet. If neither *ethernet* or *IP* is specified, *ethernet* is assumed as a default:

```
ether broadcast
ip broadcast
```

Less: Check if the packet has a length less than or equal to a specified length

```
Less 128
```

Greater: Check if the packet has a length greater than or equal to a specified length:

```
greater 128
```

Examples

True if either source or destination is hostid, which can be a hostname or an address:

```
host hostid
```

True if source is hostid:

```
src host hostid
```

True if destination is hostid:

```
dst host hostid
```

True if destination is hostid, and the protocol of a packet is IP:

```
ip dst host hostid
```

True if the ethernet destination address for a packet matches hostid, which can be a MAC address given in hexadecimal:

```
Ether dst hostid
```

True if a packet has an IP destination address that matches the network number net. net can be a name or a network number. For IP version 4, it can be written as part of an ip address, such as 192.168.1.0, 192.168.1, or 192.168, for instance:

```
dst net net
```

To select all IP packets between host-one and any host except host-two:

```
ip host host-one and not host-two
```

True if a packet has a destination matching the network net with the specified network mask:

```
dst net net mask netmask
```

Select all FTP-traffic passing through *gateway*:

```
gateway gateway and (port ftp or ftp-data)
```

True if a packet has the given port as its destination

```
dst port portnr
```

True if the destination port for a package lies within the specified range:

```
dst portrange port1-port2
```

True if a packet is an IP v4 packet of protocol type protocol. Valid options for protocol *are icmp, icmp6, igmp, igrp, pim, ah, esp, vrrp, udp*, or *tcp*.
Note: *tcp*, *upd* and *icmp* are keywords which need to be escaped using a backslash (\), as in the next example. These are essentially abbreviations for "proto protocol".

```
ip proto protocol
```

True for an IPv4 udp packet:

```
ip proto \udp
```

Equivalent to the line above; note that *udp* is a keyword here:

```
udp and ip
```

Like the above, but also for IP v6:

```
udp
```

Again; the equivalent of the previous example:

```
udp and (ip || ip6)
```

True for an IPv6 packet with a matching protocol:

```
ip6 proto protocol
```

True if a packet is of ether type protocol. Valid options for protocol are ip, ip6, arp, rarp, atalk, aarp, decnet, sca, lat, mopdl, moprc, iso, stp, ipx, or netbeui.
Note: All of these are also keywords, and need to be escaped using a backslash.

```
ether proto protocol
```

True for arp-packets:

```
ether proto \arp
```

Equivalent to the previous line (now using the keyword instead of escaping it):

```
arp
```

Capture all packets to/from 10.10.10.10 that are not to/from 192.168.0.0:

```
host 10.10.10.10 && !net 192.168
```

Capture all packets going to or from the address 10.10.10.10 and to or from port 80:

```
host 10.10.10.10 && port 80
```

Go to byte 8 of the ip header and check one byte (TTL field)

```
ip[8]
```

Go to the start of the tcp header and check 2 bytes (source port)

```
tcp[0:2]
```

Select the start and end packets (SYN and FIN) of each TCP conversation that involves a non-local host.

```
tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net localnet
```

select IP packets longer than 576 bytes sent through gateway:

```
gateway snup and ip[2:2] > 576
```

All ICMP packets that are not echo requests/replies (i.e., not ping packets):

```
icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply
```

tcpdump

tcpdump is a command line packet analyzer which can capture and display various network traffic on the computer on which it runs. Like Wireshark, it uses BPF filters to limit the packets captured.

Syntax

The basic syntax for running tcpdump with a BPF filter is as follows, where *options* are the command line parameters for tcpdump, and *filter* is a filter in the format described previously:

```
root@kali:~# tcpdump options filter
```

To see a list of possible *options*, use "*-help"*, or ror a fuller description, refer to the tcpdump man pages:

```
root@kali:~# tcpdump –help
root@kali:~# man tcpdump
```

Examples

Capture up to five packets destined for the host cybrary.it:

```
root@kali:~# tcpdump -c 5 dst host cybrary.it
```

List available interfaces to listen to / capture from:

```
root@kali:~# tcpdump -D
```

Specify which interface to listen to (in this case *lo*, which is the Loopback; i.e.

```
root@kali:~# tcpdump -i lo
```

Capture without translating addresses to names (i.e. show IP addresses instead of hostnames)

```
root@kali:~# tcpdump -n
```

Capture all packets and write them to a file named *capture-log*:

```
root@kali:~# tcpdump -w capture-log
```

Capture the first 25 packets of TCP data destined for either port 80 or 8080, and write the contents of those packets to the file *capture-log*:

```
root@kali:~# tcpdump -c 25 -w capture-log tcp dst port 80 or 8080
```