

We Secure the Internet.

## How to use fw monitor 10-Jul-2003

### Abstract

Inspecting network traffic is an essential part of today's deployment and troubleshooting tasks. With `fw monitor` Check Point provides a powerful built-in tool to simplify this task. `fw monitor` captures network packets at multiple capture points within the FireWall-1 chain. These packets can be inspected using industry-standard tools later on. This documents describes how to use `fw monitor` and use it's features to simplify the capturing tasks and provide the information you need.

Document Title: How to use fw monitor

Creation Date: 26-Feb-2003

Modified Date: 10-Jul-2003

Document Revision: 1.01

Product Class: FireWall-1 / VPN-1, fw monitor, SecuRemote/SecureClient, Ethereal, CPethereal

Product and Version: FireWall-1/VPN-1 NG

Author: Bernd Ochsmann <bochsman@checkpoint.com, Udo Schneider <udos@checkpoint.com>

## Table of Contents

<b>ABSTRACT</b> .....	<b>1</b>
<b>ACKNOWLEDGMENTS</b> .....	<b>4</b>
<b>TYPOGRAPHIC CONVENTIONS USED IN THIS PAPER</b> .....	<b>5</b>
<b>OVERVIEW</b> .....	<b>6</b>
<b>COMMAND SYNTAX</b> .....	<b>7</b>
Break Sequence.....	7
Printing the UUID or the SUUID [-u s] .....	7
Flush standard output [-i].....	7
Debugging fw monitor [-d] / [-D] .....	7
Filter fw monitor packets <{-e expr}+ -f <filter-file ->> .....	7
Limit the packet length [-l len].....	9
Capture masks [-m mask] .....	9
Print packet/payload data [-x offset[,len]] .....	9
Write output to file [-o <file>] .....	10
Insert fw monitor chain module at a specific position <[-pi pos] [-pl pos] [-po pos] [-pO pos]   -p all > .....	10
Use absolute chain positions [-a] .....	10
Capture a specific number of packets [-ci count] [-co count] .....	11
Capture on a specific Virtual Router or Virtual Machine [-vs vsid or vsname] .....	11
<b>STANDARD USAGE</b> .....	<b>12</b>
Using fw monitor .....	12
Reading fw monitor output .....	12
How does fw monitor work? .....	13
<b>ADVANCED USAGE</b> .....	<b>14</b>
Capture masks .....	14
Print packet/payload data.....	16
Limit the packet length .....	17
Using UUIDs and SSIDs .....	17
How to change the position of the fw monitor chain module .....	19
fw monitor filters .....	30
<b>INSPECT FW MONITOR FILES</b> .....	<b>39</b>
Using snoop to inspect fw monitor files .....	39
Using tcpdump to inspect fw monitor files .....	42
Using Ethereal to inspect fw monitor files.....	43
Using CPEThereal to inspect fw monitor files.....	55
<b>SRFW – FW MONITOR ON THE CLIENT SIDE</b> .....	<b>64</b>

<b>FW MONITOR ON FIREWALL-1 VSX .....</b>	<b>65</b>
<b>RESOURCES .....</b>	<b>66</b>
Secure Knowledge Links .....	66
Detecting sniffers on your network .....	67
snoop .....	67
tcpdump .....	67
Ethereal .....	67
CPEthereal .....	68
Miscellaneous .....	68
<b>REFERENCE .....</b>	<b>69</b>
Multicast MAC addresses .....	69
fw monitor file format .....	69
UUID format .....	70

## Acknowledgments

Due to many questions and requests regarding `fw monitor` we developed the idea to write a “short” paper about `fw monitor`. We thought 15-20 pages would be more than enough to cover all important aspects of `fw monitor`...

As we started to collect information about `fw monitor` and related topics we soon discovered that there was much more to write about than we initially thought. We had two choices: Writing a short note much like a man page or choosing the long way and write a comprehensive manual. We decided for the second and this paper is the result.

No way could we have completed this paper without the awesome help of many people who give us invaluable feedback. We would like to thank Shaul Eizikovich for his fabulous CPEThereal; Misha Pak for giving us deep insight in many `fw monitor` functionalities and mechanisms; Lior Cohen, Tal Manor and Mark Wellins from Solutions Center for their great ongoing support; Joe Green for pointing us to missing details we totally overlooked; our colleagues in the german Check Point Office who kept us working on the paper by asking permanently for the final version; Alfred Köbler (ICON Systems GmbH) for initially adding `fw monitor` decoding support to Ethereal, Manuela Menges (BASF IT Services GmbH) for comments which where nearly as long as the whole document and to all others which provided comments and suggestions.

## Typographic Conventions used in this paper

The following table describes the typographic conventions used in this paper.

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output.	Use <code>fw monitor -m iO</code> to see all Pre-In and Post-Out packets.
<b>AaBbCc123</b>	What you type, when contrasted with on-screen computer output.	<pre>[cpmodule]# <b>fw monitor -m iO</b> monitor: getting filter (from command line) monitor: compiling</pre>
<i>AaBbCc123</i>	Book titles or words to be emphasized.	Please refer to the <i>FireWall-1 Getting Started Guide</i> for further information.
<b>AaBbCc123</b>	Text that appears on an object in a window.	Use <b>CheckPoint/Decode as FW-1 Monitor file</b> to enable decoding.

## Overview

In many deployment and support scenarios capturing network packets is an essential functionality. `tcpdump` or `snoop` are tools normally used for this task. `fw monitor` provides an even better functionality but omits many requirements and risks of these tools.

- **No Security Flaws**
  - `tcpdump` and `snoop` are normally used with network interface cards in promiscuous mode. Unfortunately the promiscuous mode allows remote attacks against these tools (see [Snoop vulnerable to a remotely exploitable buffer overflow](#)). `fw monitor` does not use the promiscuous mode to capture packets. In addition most FireWalls' operating systems are hardened. In most cases this hardening includes the removal of tools like `tcpdump` or `snoop` because of their security risk.
- **Available on all FireWall-1 installations**
  - `fw monitor` is a built-in firewall tool which needs no separate installation in case capturing packets is needed. It is a functionality provided with the installation of the FireWall package.
- **Multiple capture positions within the FireWall-1 kernel module chain**
  - `fw monitor` allows you to capture packets at multiple capture positions within the FireWall-1 kernel module chain; both for inbound and outbound packets. This enables you to trace a packet through the different functionalities of the firewall.
- **Same tool and syntax on all platforms**
  - Another important fact is the availability of `fw monitor` on different platforms. Tools like `snoop` or `tcpdump` are often platform dependent or have specific "enhancements" on certain platforms. `fw monitor` and all its' related functionality and syntax is absolutely identical across all platforms. There is no need to learn any new "tricks" on an unknown platform.

Normally the Check Point kernel modules are used to perform several functions on packets (like filtering, en- and decrypting, QoS ...). `fw monitor` adds its own modules to capture packets. Therefore `fw monitor` can capture all packets which are seen and/or forwarded by the FireWall.

## Command syntax

```
fw monitor [-u|s] [-i] [-d] [-D] <{-e expr+|-f <filter-file|->> [-l len] [-m mask] [-x offset[,len]] [-o <file>] <[-pi pos] [-pI pos] [-po pos] [-pO pos] | -p all > [-a] [-ci count] [-co count] [-vs vsid or vsname]
```

Figure 1: fw monitor command line options

## Break Sequence

Use `^C` (that is `Control + C`) to stop `fw monitor` from capturing packets.

## Printing the UUID or the SUUID [-u|s]

The option `-u` or `-s` is used to print UUIDs or SUUIDs for every packet. Please note that it is only possible to print the UUID or the SUUID – not both. Please refer to [Using UUIDs and SSIDs](#) for further information.

## Flush standard output [-i]

Use `-i` to make sure that captured data for each packet is at once written to standard output. This is especially useful if you want to kill a running `fw monitor` process and want to be sure that all data is written to a file.

## Debugging fw monitor [-d] / [-D]

The `-d` option is used to start `fw monitor` in debug mode. This will give you an insight into `fw monitor`'s inner workings although this option is only rarely used outside Check Point. It's also possible to use `-D` to create an even more verbose output.

## Filter fw monitor packets <{-e expr+|-f <filter-file|->>

`fw monitor` has the ability to capture only packets in which you are interested in. It is possible to set the filter expression on the command line (using the `-e` switch), read it from a file (`-f`) or to read it from standard input (`-f -`). Please refer to [fw monitor filters](#) for a detailed description of the filter syntax.

In the following examples we are filtering for the 9<sup>th</sup> byte of the IP Header (`'accept [9:1]=1;'`). The 9<sup>th</sup> byte is the IP protocol and we are only accepting IP protocol 1 which is ICMP.

Bas

**!** When using filter expressions on the command line (using `-e`) you should make sure that they are properly quoted. On Windows and UNIX Operating systems this can be done by surrounding the expression with single quote (`'` – ASCII Value 39) or double quotes (`"` – ASCII Value 34). Please note that depending on your operating system and shell used there might be differences between the two forms – especially when using special characters or (shell) variables in the filter expression.

```
[Expert@cpmodule]# fw monitor -e 'accept [9:1]=1;'
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
eth0:i[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=20919
ICMP: type=8 code=0 echo request id=6506 seq=256
eth0:I[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=20919
ICMP: type=8 code=0 echo request id=6506 seq=256
eth0:o[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=24617
ICMP: type=0 code=0 echo reply id=6506 seq=256
eth0:O[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=24617
ICMP: type=0 code=0 echo reply id=6506 seq=256
^C
monitor: caught sig 2
monitor: unloading
```

*Figure 2: fw monitor – using filter expressions on the command line*

```
[Expert@cpmodule]# echo "accept [9:1]=1;" >myfilter.pf
[Expert@cpmodule]# fw monitor -f myfilter.pf
monitor: getting filter (from myfilter.pf)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
eth0:i[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=21213
ICMP: type=8 code=0 echo request id=7018 seq=256
eth0:I[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=21213
ICMP: type=8 code=0 echo request id=7018 seq=256
eth0:o[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=24620
ICMP: type=0 code=0 echo reply id=7018 seq=256
eth0:O[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=24620
ICMP: type=0 code=0 echo reply id=7018 seq=256
^C
monitor: caught sig 2
monitor: unloading
```

*Figure 3: fw monitor – using filter expressions in a file*

Please use **^D** (that is **Control + D**) as EOF (End Of File) character when reading the filter expression from standard input. `fw monitor` reads the expression just up to this point and processes it.

```
[Expert@cpmodule]# fw monitor -f -
monitor: getting filter (from stdin)
accept [9:1]=1;
^D
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
eth0:i[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=21307
ICMP: type=8 code=0 echo request id=7530 seq=256
eth0:I[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=21307
ICMP: type=8 code=0 echo request id=7530 seq=256
eth0:o[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=24623
ICMP: type=0 code=0 echo reply id=7530 seq=256
eth0:O[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=24623
ICMP: type=0 code=0 echo reply id=7530 seq=256
^C
monitor: caught sig 2
monitor: unloading
```

*Figure 4: fw monitor – reading filter expressions from standard input*

### Limit the packet length [-l len]

`fw monitor` allow you to limit the packet data which will be read from the kernel with `-l`. Refer to [Limit the packet length](#) for further information. This is especially useful if you have to debug high sensitive communication. It allows you to capture only the headers of a packet (e.g. IP and TCP header) while omitting the actual payload. Therefore you can debug the communication without seeing the actual data transmitted. Another possibility is to keep the amount of data low. If you don't need the actual payload for debugging you can decrease the file size by omitting the payload. It's also very useful to reduce packet loss on high-loaded machines. `fw monitor` uses a buffer to transfer the packets from kernel to user space. If you reduce the size of a single packet this buffer won't fill up so fast.

### Capture masks [-m mask]

By default `fw monitor` captures packets before and after the virtual machine in both directions (these positions can be changed. Refer to [How to change the position of the fw monitor chain module](#) for more information). The option `-m` options allows you to specify in which of the four positions you are interested. For further information refer to [Capture masks](#).

### Print packet/payload data [-x offset[,len]]

In addition to the IP and Transport header `fw monitor` can also print the packets' raw data. This can be done using the `-x` option. Optionally it is also possible to limit the data written to the screen. Please refer to [Print packet/payload data](#) for more information.

## Write output to file [-o <file>]

In addition to the ability to print out the packet's information, `fw monitor` is also able to save the raw packets' data to a file. The file format used is the same format used by tools like `snoop` (Refer to [Snoop file format \(RFC 1761\)](#) for further information). This file format can be examined using by tools like `snoop`, `tcpdump` or `Ethereal`.

```
[Expert@cpmodule]# fw monitor -e 'accept ip_p=1;' -o ping.cap
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
12
^C
monitor: caught sig 2
monitor: unloading
```

**# of captured packets**

Figure 5: `fw monitor` – writing raw packet data to a file

**!** The `snoop` file format is normally used to store Layer 2 frames. For “normal” capture files this means that the frame includes data like a source and a destination MAC address. `fw monitor` operates in the firewall kernel and therefore has no access to Layer 2 information like MAC addresses anymore. Instead of writing random MAC addresses, `fw monitor` includes information like interface name, direction and chain position as “MAC addresses”.

## Insert `fw monitor` chain module at a specific position <[-pi pos] [-pl pos] [-po pos] [-pO pos] | -p all >

In addition to capture masks (which give you the ability to specify whether you are interested in packets in a specific position) `fw monitor` has the ability to define where exactly (in the FireWall-1 chain) the packets should be captured. This can be defined using `-p[iIoO] [pos]`. Please refer to [How to change the position of the `fw monitor` chain module](#) for further information.

## Use absolute chain positions [-a]

If you use `fw monitor` to output the capture into a file (option `-o`), one of the fields written down to the capture file is the chain position of the `fw monitor` chain module. Together with an simultaneous execution of `fw ctl chain` you can determine where the packet was captured (see [How to change the position of the `fw monitor` chain module](#) for more information on this). Especially when using `-p all` you will find the same packet captured multiples times at different chain positions.

The option `-a` changes the chain id from an relative value (which only makes sense with the matching `fw ctl chain` output) to an absolute value. These absolute values are known to `CPEthereal` (see [Using `CPEthereal` to inspect `fw monitor` files](#)) and can be displayed by it.

## Capture a specific number of packets [-ci count] [-co count]

`fw monitor` enables you to limit the number of packets being captured. This is especially useful in situations where the firewall is filtering high amounts of traffic. In such situations `fw monitor` may bind so many resources (for writing to the console or to a file) that recognizing the break sequence (Control-C) might take very long.

```
[Expert@cpmodule]# fw monitor -ci 3 -o dump1.cap
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
12
monitor: unloading
Read 3 inbound packets and 3 outbound packets
[Expert@cpmodule]# fw monitor -co 3 -o dump2.cap
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
14
monitor: unloading
Read 4 inbound packets and 3 outbound packets
```

Figure 6: `fw monitor` – capture a specific number of packets

`fw monitor` counts "real" packets. In the example above we decided to capture just 3 packets. But the packet counter was 12 and 14. This can be explained by the multiple capture positions. In the first example we had three inbound and three outbound packets (six in sum). Each packet is counted to times (preInbound/postInbound or preOutbound/postOutbound):

$3 \text{ (inbound)} * 2 \text{ (pre/post)} + 3 \text{ (outbound)} * 2 \text{ (pre/post)} = 12 \text{ packets.}$

The same for the second example:

$4 \text{ (inbound)} * 2 \text{ (pre/post)} + 3 \text{ (outbound)} * 2 \text{ (pre/post)} = 14 \text{ packets.}$



Please note that it is possible to use the `-ci` and the `-co` switches together. `fw monitor` will stop capturing packets if the number of packets for one of the two counters reaches it's value.

## Capture on a specific Virtual Router or Virtual Machine [-vs vsid or vsname]

FireWall-1 VSX enables you to run multiple Virtual Routers and FireWalls on one physical machine. Using the option `-vs` you can specify on which virtual component the packets should be captured. This option is only available on a FireWall-1 VSX module – not on a standard module. Please refer to [fw monitor on FireWall-1 VSX](#) for more information.

## Standard Usage

### Using fw monitor

The easiest way to use `fw monitor` is to invoke it without any parameter. This will output every packet from every interface that passes (or at least reaches) the enforcement module. Please note that the same packet is appearing several times (two times in the example below). This is caused by `fw monitor` capturing the packets at different capture points. Please refer to [Capture masks](#) for a more detailed explanation.

```
[cpmodule]# fw monitor
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
eth0:i[285]: 172.16.1.133 -> 172.16.1.2 (TCP) len=285 id=1075
TCP: 1050 -> 18190 ...PA. seq=bf8bc98e ack=941b05bc
eth0:I[285]: 172.16.1.133 -> 172.16.1.2 (TCP) len=285 id=1075
TCP: 1050 -> 18190 ...PA. seq=bf8bc98e ack=941b05bc
eth0:o[197]: 172.16.1.2 -> 172.16.1.133 (TCP) len=197 id=44599
TCP: 18190 -> 1050 ...PA. seq=941b05bc ack=bf8bca83
eth0:O[197]: 172.16.1.2 -> 172.16.1.133 (TCP) len=197 id=44599
TCP: 18190 -> 1050 ...PA. seq=941b05bc ack=bf8bca83
eth0:o[1500]: 172.16.1.2 -> 172.16.1.133 (TCP) len=1500 id=44600
TCP: 18190 -> 1050 ....A. seq=941b0659 ack=bf8bca83
^C
monitor: caught sig 2
monitor: unloading
```

Figure 7: Invoking fw monitor without parameters

### Reading fw monitor output

```
eth0:i[285]: 172.16.1.133 -> 172.16.1.2 (TCP) len=285 id=1075
```

Figure 8: Reading fw monitor output – first line

This packet was captured on the first network interface (`eth0`) in inbound direction before the virtual machine (lowercase `i`; see [Capture masks](#) for a more detailed explanation). The packet length is 285 bytes (in square parenthesis; repeated at the end of the line. Please note that these two values may be different. Refer to the [Virtual Defragmentation note](#) for further information) and the packets ID is 1075. The packet was sent from 172.16.1.133 to 172.16.1.2 and carries a TCP header/payload.

```
TCP: 1050 -> 18190 ...PA. seq=bf8bc98e ack=941b05bc
```

*Figure 9: Reading fw monitor output – second line*

The second line tells us that this is an `TCP` payload inside the IP packet which was sent from port 1050 to port 18190. The following element displays the TCP flags set (in this case `PUSH` and `ACK`). The last two elements are showing the sequence number (`seq=bf8bc98e`) of the TCP packet and the acknowledged sequence number (`ack=941b05bc`). You will see similar information for UDP packets.

**!** You will only see a second line if the transport protocol used is known to `fw monitor`. Known protocols are for example `TCP`, `UDP` and `ICMP`. If the transport protocol is unknown or can not be analyzed because it is encrypted (e.g. `ESP` or encapsulated (e.g. `GRE`) the second line is missing.

### How does fw monitor work?

In contrast to other capturing tools like `snoop` or `tcpdump`, `fw monitor` does not use the promiscuous mode on network interface cards. Based on the fact that FireWall-1 already receives all packets (due to the FireWall-1 kernel module between NIC driver and IP stack) `fw monitor` uses it's own kernel module to capture packets (compared to filtering/encrypting them).

Unlike `snoop` or `tcpdump`, `fw monitor` has the ability to capture packets at different positions (refer to [Capture position](#) for more information about the four locations) in the FireWall-1 kernel module chain. `snoop` and `tcpdump` are capturing packets when they enter or leave the computer. Especially when NAT with FireWall-1 is involved `fw monitor` offers the possibility to capture packets at multiple locations (e.g. after the FireWall Virtual in inbound direction). This can help you to see how the packets are translated by the firewall and on which IP address the routing decision is made.

## Advanced usage

### Capture masks

`fw monitor` is able to capture packets at four different positions in the FireWall-1 chain:

- on the inbound interface *before* the Virtual Machine (pre-inbound)
- on the inbound interface *after* the Virtual Machine (post-inbound)
- on the outbound interface *before* the Virtual Machine (pre-outbound)
- on the outbound interface *after* the Virtual Machine (post-outbound)

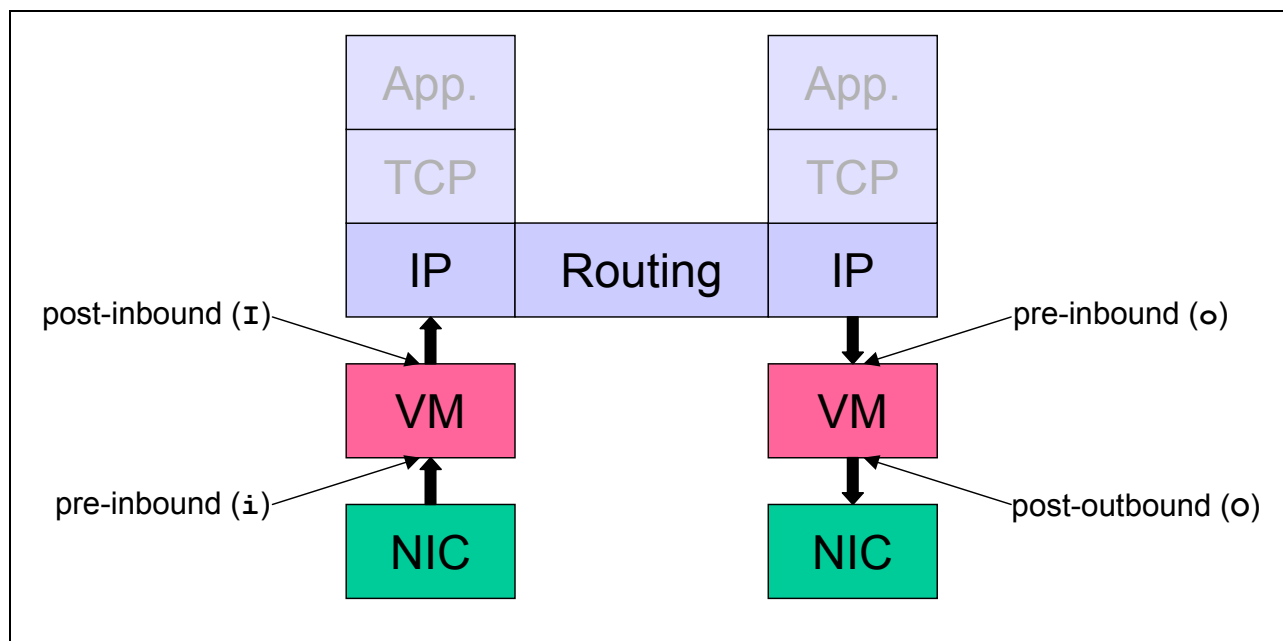


Figure 10: `fw monitor` capture positions



The picture above is a simplified figure of the actual implementation. To find out more please refer to [How to change the position of the fw monitor chain module](#) for more information.

By default `fw monitor` captures packets at all four positions. With `-m` it is possible to capture packets at specific positions. `fw monitor` uses single letters as indicators for the position:

Capture position	fw monitor mask value
pre-inbound	i (lowercase i)
post-inbound	I (uppercase i)
pre-outbound	o (lowercase o)
post-outbound	O (uppercase o)

Figure 11: *fw monitor capture position masks*

Using `fw monitor` masks it's easily possible to capture only packets before they are inspected by the firewall in inbound direction and after they have been inspected by the firewall in outbound direction.

### fw monitor capture mask example

In the example below we are capturing a communication between a client (10.2.4.12) and a web server (172.16.1.1). The client address is translated to 172.16.1.3 and the server address is translated to 10.2.253.2. You can easily see how the non-translated packet enters the firewall and how the translated packet (source and destination) is leaving the firewall:

```
[Expert@cpmodule]# fw monitor -m iO
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
eth1:i[60]: 10.2.4.12 -> 10.2.254.2 (TCP) len=60 id=41817
TCP: 34762 -> 80 .S.... seq=e8527fe7 ack=00000000
eth0:O[60]: 172.16.1.3 -> 172.16.1.1 (TCP) len=60 id=41817
TCP: 34762 -> 80 .S.... seq=e8527fe7 ack=00000000
eth0:i[60]: 172.16.1.1 -> 172.16.1.3 (TCP) len=60 id=41818
TCP: 80 -> 34762 .S..A. seq=e7c90e3e ack=e8527fe8
eth1:O[60]: 10.2.254.2 -> 10.2.4.12 (TCP) len=60 id=41818
TCP: 80 -> 34762 .S..A. seq=e7c90e3e ack=e8527fe8
eth1:i[52]: 10.2.4.12 -> 10.2.254.2 (TCP) len=52 id=41819
TCP: 34762 -> 80 ....A. seq=e8527fe8 ack=e7c90e3f
eth0:O[52]: 172.16.1.3 -> 172.16.1.1 (TCP) len=52 id=41819
TCP: 34762 -> 80 ....A. seq=e8527fe8 ack=e7c90e3f
^C
monitor: caught sig 2
monitor: unloading
```

Figure 12: *Using fw monitor capture masks*

Using the right combination of capture masks it's very easy to find out when the firewall applies which NAT rules (Hide NAT, Static Destination NAT or Static Source NAT). This is especially useful when you need to know which packets the routing of the operating system is using to do the routing decision.

## Print packet/payload data

Using `-x` it's possible to print the packet's raw data. You have to specify a specific offset (e.g. used to jump over the IP/TCP header) from which the packet data is printed. It's also possible to limit the length of the raw data:

In the following example we are skipping the IP and TCP header (offset 52) and are using a length of 96:

```
[Expert@cpmodule]# fw monitor -m i -x 52,96
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
eth1:i[60]: 10.2.4.12 -> 10.2.254.2 (TCP) len=60 id=18687
TCP: 36242 -> 80 .S.... seq=afe21c6a ack=00000000
0000 0000 0103 0300 .....

eth0:i[60]: 172.16.1.1 -> 172.16.1.3 (TCP) len=60 id=18688
TCP: 80 -> 36242 .S..A. seq=b060b1df ack=afe21c6b
0198 23ef 0103 0300 ..#.....

eth1:i[52]: 10.2.4.12 -> 10.2.254.2 (TCP) len=52 id=18689
TCP: 36242 -> 80 ....A. seq=afe21c6b ack=b060b1e0

eth1:i[594]: 10.2.4.12 -> 10.2.254.2 (TCP) len=594 id=18690
TCP: 36242 -> 80 ...PA. seq=afe21c6b ack=b060b1e0
4745 5420 2f43 504c 6f67 6f48 6f72 697a GET /CPLogoHoriz
5075 7270 2e67 6966 2048 5454 502f 312e Purp.gif HTTP/1.
310d 0a48 6f73 743a 2031 302e 1..Host: 10.

eth0:i[52]: 172.16.1.1 -> 172.16.1.3 (TCP) len=52 id=18691
TCP: 80 -> 36242 ....A. seq=b060b1e0 ack=afe21e89

eth0:i[288]: 172.16.1.1 -> 172.16.1.3 (TCP) len=288 id=18692
TCP: 80 -> 36242 ...PA. seq=b060b1e0 ack=afe21e89
4854 5450 2f31 2e31 2033 3034 204e 6f74 HTTP/1.1 304 Not
204d 6f64 6966 6965 640d 0a53 6572 7665 Modified..Serve
723a 2074 6874 7470 642f 322e r: thttpd/2.

^C
monitor: caught sig 2
monitor: unloading
```

Figure 13: fw monitor – printing packet raw data

## Limit the packet length

`fw monitor` can limit the amount of packet data which will be read from the kernel. The option `-l` is used for this purpose. `fw monitor` will only read as many bytes from the kernel as you specified for the `-l` option. Please make sure to capture as least as many bytes so that the IP and transport headers are included.

## Using UUIDs and SSIDs

UUIDs (universal-unique-identifiers) are a new feature in NG. The firewall assigns an UUID to every connection passing the firewall. This UUID is kept through all firewall operations. Therefore you can follow a connection through the firewall even if the packet content is NAT'ed. The UUID is also kept in the connection table entry for the connection.

Additionally there is the concept of an SUUID (Session UUID). For services which are using several connections (e.g. FTP) every connection has a unique UUID but the SUUID is equal for all the connections (it's the same as the first/control connection's UUID).

UUIDs and SUIDs are very helpful for tracking connection through different chain modules of the firewall. Even if a connection is NAT'ed the UUID or SUID remains the same. Therefore filtering for the UUID or SUID helps you to find all packets belonging to a connection or session, even if the packets change.

Please note that the first packet of a connection or session as no UUID or SUID assigned yet (SUID/SUID is all zero). After the first packet has been processed by the firewall a UUID or SUID is assigned and will remain the same for the whole connection/session.

An UUID is built from four 32bit values using a timestamp, a counter, the firewall IP address and a process ID. From this 128bit value a smaller 32bit value is constructed which is printed as well. Please refer to [UUID format](#) for detailed information.

The UUID/SUUID is printed in front of the IP information. The first value is the striped UUID (32bit). The second value is the complete UUID (128bit).

```
[Expert@cpmodule]# fw monitor -u
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
[00000000 - 00000000 00000000 00000000]:eth1:i[60]: 10.2.4.12 -> 10.2.254.2 (TCP) len=60 id=46124
TCP: 34838 -> 80 .S.... seq=5c2282fa ack=00000000
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth1:I[60]: 10.2.4.12 -> 172.16.1.1 (TCP) len=60 id=46124
TCP: 34838 -> 80 .S.... seq=5c2282fa ack=00000000
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth0:o[60]: 10.2.4.12 -> 172.16.1.1 (TCP) len=60 id=46124
TCP: 34838 -> 80 .S.... seq=5c2282fa ack=00000000
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth0:O[60]: 172.16.1.3 -> 172.16.1.1 (TCP) len=60 id=46124
TCP: 34838 -> 80 .S.... seq=5c2282fa ack=00000000
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth0:i[60]: 172.16.1.1 -> 172.16.1.3 (TCP) len=60 id=46125
TCP: 80 -> 34838 .S..A. seq=5c3b9465 ack=5c2282fb
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth0:I[60]: 172.16.1.1 -> 10.2.4.12 (TCP) len=60 id=46125
TCP: 80 -> 34838 .S..A. seq=5c3b9465 ack=5c2282fb
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth1:o[60]: 172.16.1.1 -> 10.2.4.12 (TCP) len=60 id=46125
TCP: 80 -> 34838 .S..A. seq=5c3b9465 ack=5c2282fb
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth1:O[60]: 10.2.254.2 -> 10.2.4.12 (TCP) len=60 id=46125
TCP: 80 -> 34838 .S..A. seq=5c3b9465 ack=5c2282fb
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth1:i[52]: 10.2.4.12 -> 10.2.254.2 (TCP) len=52 id=46126
TCP: 34838 -> 80 ....A. seq=5c2282fb ack=5c3b9466
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth1:I[52]: 10.2.4.12 -> 172.16.1.1 (TCP) len=52 id=46126
TCP: 34838 -> 80 ....A. seq=5c2282fb ack=5c3b9466
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth0:o[52]: 10.2.4.12 -> 172.16.1.1 (TCP) len=52 id=46126
TCP: 34838 -> 80 ....A. seq=5c2282fb ack=5c3b9466
[6b770000 - 3e5c776b 00000000 020110ac 000007b6]:eth0:O[52]: 172.16.1.3 -> 172.16.1.1 (TCP) len=52 id=46126
TCP: 34838 -> 80 ....A. seq=5c2282fb ack=5c3b9466
^C
monitor: caught sig 2
monitor: unloading
```

Figure 14: fw monitor UUID output

**!** Please note that new connections are using a UUID of 0x0000.... Once they have been “seen” by the firewall module a UUID is assigned and maintained.

## How to change the position of the fw monitor chain module

In [Capture masks](#) we described `fw monitor` capture masks. The positions were defined to be *before* the virtual machine and *after* the virtual machine. Although not wrong it is not completely right.

Check Point uses a so called “kernel module chain” for different kernel modules which are working with the packets. The different modules (Firewall, VPN , FloodGate ... ) are passing on a packet to the next module and building up a kind of chain this way.

The example below shows how the packets is processed by different chain modules while entering and leaving the firewall machine:

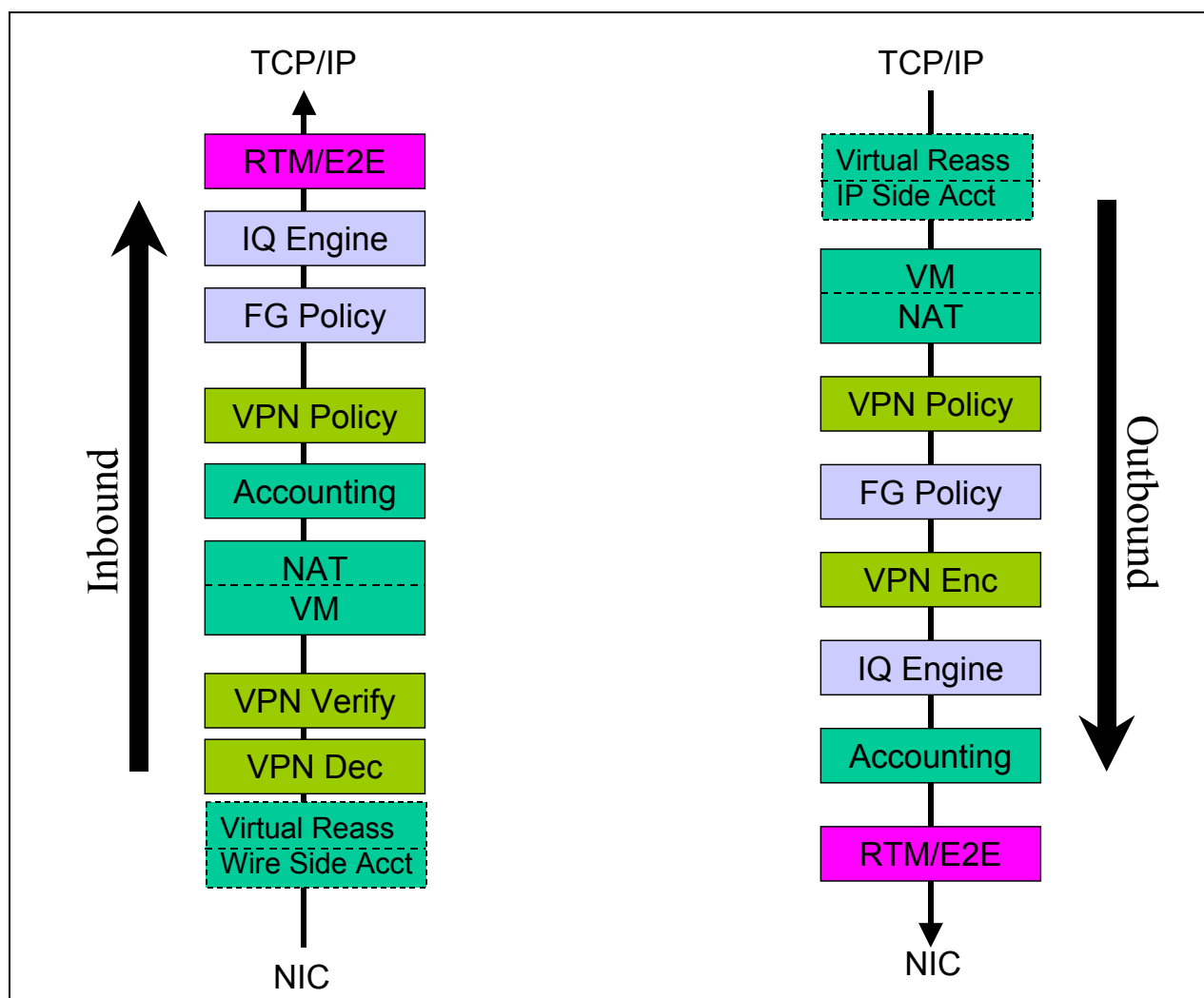


Figure 15: FireWall chain – schematic overview

You can take a look at the actual chain using the `fw ctl chain` command. This will show you the chain modules actually loaded on your machine and their order. Please note that there are more kernel modules in the chain which are not visible by `fw ctl chain` and also cannot be used for `fw monitor` kernel module positioning.

```
[Expert@cpmodule]# fw ctl chain
in chain (9):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 2000000 (cb1c1c64) vpn decrypt (vpn)
  2: - 1ffffff6 (ca8da0f8) Stateless verifications (asm)
  3: - 1ffffff0 (cb1c17f0) vpn decrypt verify (vpn_ver)
  4: - 1000000 (ca8eb688) SecureXL connection syn (secxl_sync)
  5:          0 (ca8aa0c0) fw VM inbound (fw)
  6:  2000000 (cb1c2aa0) vpn policy inbound (vpn_pol)
  7: 10000000 (ca8eb728) SecureXL inbound (secxl)
  8: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (8):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 1fffffff (cb1c16fc) vpn nat outbound (vpn_nat)
  2: - 1f00000 (ca8da0f8) Stateless verifications (asm)
  3:          0 (ca8aa0c0) fw VM outbound (fw)
  4:  2000000 (cb1c26e0) vpn policy outbound (vpn_pol)
  5: 10000000 (ca8eb728) SecureXL outbound (secxl)
  6: 20000000 (cb1c2164) vpn encrypt (vpn)
  7: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
```

Figure 16: fw ctl chain output

**!** The output of `fw ctl chain` is platform, version and product dependent. There is no reason to worry if your `fw ctl chain` output looks different. The number and kind of modules displayed here may vary based on the platform used and products installed. Please note that even the offsets shown here are version dependent and may change.

`fw monitor` inserts its own modules in this module chain and is capturing packets there. By default this is not the first and last position in the chain. Therefore the original meaning of *before* and *after* needs to be redefined:

- Without changing the position of the kernel module everything is quite simple:
  - *Before* can be interpreted as being before any firewall, VPN or NAT action.
  - *After* is defined as being after and NAT or VPN operation has occurred.
- If you change the kernel module position using `-p` (see [How to change the position of the fw monitor chain module](#)) but do not capture at all positions (see [All positions](#)):
  - *Before* (pre-inbound /pre-outbound) describes the first instance of the `fw monitor` kernel module (although it may be *after* the VM!)
  - *After* (post-inbound/post-outbound) describes the second instance of the `fw monitor` kernel module (although in fact, like above, it may be before the VM).
- If you are using `-p all` to capture packets between every kernel module (see [All positions](#)):
  - All packets captured between the kernel module *before* the VM are marked as being pre-inbound/pre-outbound
  - All packets captured after the VM are marked as being post-inbound/post-outbound.

**!** Due to the fact that the `fw monitor` chain module is a “normal” chain module there are some issues one should be aware of. All chain modules are working on already (virtual) defragmented packets. Even if a packet is fragmented `fw monitor` will show the defragmented packet, not the fragments.

The virtual defragmentation may lead to some confusion when working with fragmented packets: `fw monitor` captures defragmented packets but some of the info about the packet is taken from the first IP fragment. This may lead to two “anomalies”:

1. If you are printing `fw monitor` output to standard output you may see two different size values:



```
hme1:i[828] 10.0.0.1 -> 10.0.0.2 (ICMP) len=420 id=224 off=0
```

In this example here the first length (square parenthesis) is 828 Bytes. This is the length of the defragmented packet. The second size (`len=`) is 420 Bytes. This is the size of the first IP Fragment. This may also cause “invalid packets” in [Ethereal](#) because the size in the IP header (430 bytes here) is different from the size of the actual packet.

2. In addition it may be that the “more fragments” bit is set in the IP header, although the packet itself is already defragmented.

If `fw monitor` is active you can see the `fw monitor` chain modules using `fw ctl chain`:

```
[Expert@cpmodule]# fw monitor -o dump.cap
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
```

### Switch to another terminal

```
[cpmodule]# fw ctl chain
in chain (11):
 0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
 1: -70000000 (ca8c6020) fwmonitor (i/f side)
 2: - 2000000 (cb1c1c64) vpn decrypt (vpn)
 3: - 1ffffff6 (ca8da0f8) Stateless verifications (asm)
 4: - 1ffffff0 (cb1c17f0) vpn decrypt verify (vpn_ver)
 5: - 1000000 (ca8eb688) SecureXL connection syn (secxl_sync)
 6: 0 (ca8aa0c0) fw VM inbound (fw)
 7: 2000000 (cb1c2aa0) vpn policy inbound (vpn_pol)
 8: 10000000 (ca8eb728) SecureXL inbound (secxl)
 9: 70000000 (ca8c6020) fwmonitor (IP side)
10: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (10):
 0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
 1: -70000000 (ca8c6020) fwmonitor (IP side)
 2: - 1fffffff (cb1c16fc) vpn nat outbound (vpn_nat)
 3: - 1f00000 (ca8da0f8) Stateless verifications (asm)
 4: 0 (ca8aa0c0) fw VM outbound (fw)
 5: 2000000 (cb1c26e0) vpn policy outbound (vpn_pol)
 6: 10000000 (ca8eb728) SecureXL outbound (secxl)
 7: 20000000 (cb1c2164) vpn encrypt (vpn)
 8: 70000000 (ca8c6020) fwmonitor (i/f side)
 9: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
```

Figure 17: `fw monitor` modules in firewall chain

```
[cpmodule]# fw ctl chain
in chain (11):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: -70000000 (ca8c6020) fwmonitor (i/f side)
  2: - 2000000 (cb1c1c64) vpn decrypt (vpn)
  3: - 1ffffff6 (ca8da0f8) Stateless verifications (asm)
  4: - 1ffffff0 (cb1c17f0) vpn decrypt verify (vpn_ver)
  5: - 1000000 (ca8eb688) SecureXL connection syn (secxl_sync)
  6: 0 (ca8aa0c0) fw VM inbound (fw)
  7: 2000000 (cb1c2aa0) vpn policy inbound (vpn_pol)
  8: 10000000 (ca8eb728) SecureXL inbound (secxl)
  9: 70000000 (ca8c6020) fwmonitor (IP side)
  10: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (10):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: -70000000 (ca8c6020) fwmonitor (IP side)
  2: - 1fffffff (cb1c16fc) vpn nat outbound (vpn_nat)
  3: - 1f00000 (ca8da0f8) Stateless verifications (asm)
  4: 0 (ca8aa0c0) fw VM outbound (fw)
  5: 2000000 (cb1c26e0) vpn policy outbound (vpn_pol)
  6: 10000000 (ca8eb728) SecureXL outbound (secxl)
  7: 20000000 (cb1c2164) vpn encrypt (vpn)
  8: 70000000 (ca8c6020) fwmonitor (i/f side)
  9: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
```

Figure 18: fw monitor modules in firewall chain

In inbound direction all chain positions *before* the firewall are considered to be **preInbound**. All chain modules *after* the firewall VM are **postInbound**.

In outbound direction all chain position *before* the firewall VM are considered to be **preOutbound**. All chain modules *after* the VM are **postOutbound**.

The `-p[iIoO]` switch allows you to insert the `fw monitor` module at different positions in the chain. The letters “iIoO” are used with the same meaning like the [fw monitor capture masks](#). There are four possibilities to define the position of the fw monitor module in the chain:

- relative position using a number
- relative position using an alias
- absolute position
- all positions

## Relative position using a Number

The chain modules are ordered with an ascending number starting with zero: You can use this number to specify the position where the `fw monitor` module should be inserted. The `fw monitor` module does not replace the module with this number. The previous module (and all following modules) are moved by one position:

```
[Expert@cpmodule]# fw ctl chain
in chain (9):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 2000000 (cb1c1c64) vpn decrypt (vpn)
  2: - 1ffffff6 (ca8da0f8) Stateless verifications (asm)
  3: - 1ffffff0 (cb1c17f0) vpn decrypt verify (vpn_ver)
  4: - 1000000 (ca8eb688) SecureXL connection syn (secxl_sync)
  5:      0 (ca8aa0c0) fw VM inbound (fw)
  6:  2000000 (cb1c2aa0) vpn policy inbound (vpn_pol)
  7: 10000000 (ca8eb728) SecureXL inbound (secxl)
  8: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (8):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 1fffffff (cb1c16fc) vpn nat outbound (vpn_nat)
  2: - 1f00000 (ca8da0f8) Stateless verifications (asm)
  3:      0 (ca8aa0c0) fw VM outbound (fw)
  4:  2000000 (cb1c26e0) vpn policy outbound (vpn_pol)
  5: 10000000 (ca8eb728) SecureXL outbound (secxl)
  6: 20000000 (cb1c2164) vpn encrypt (vpn)
  7: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
```

*Figure 19: fw ctl chain – relative module positions*

In the following example we are inserting the `fw monitor` chain module `prelnbound (i)` at position 4:

```
[Expert@cpmodule]# fw monitor -pi 4 -o dump.cap
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
in chain (11):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 2000000 (cb1c1c64) vpn decrypt (vpn)
  2: - 1ffffff6 (ca8da0f8) Stateless verifications (asm)
  3: - 1ffffff0 (cb1c17f0) vpn decrypt verify (vpn_ver)
  4: - 1000001 (ca8c6020) fwmonitor (i/f side)
  5: - 1000000 (ca8eb688) SecureXL connection syn (secxl_sync)
  6:      0 (ca8aa0c0) fw VM inbound (fw)
  7:  2000000 (cb1c2aa0) vpn policy inbound (vpn_pol)
  8: 10000000 (ca8eb728) SecureXL inbound (secxl)
  9: 70000000 (ca8c6020) fwmonitor (IP side)
 10: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (10):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: -70000000 (ca8c6020) fwmonitor (IP side)
  2: - 1fffffff (cb1c16fc) vpn nat outbound (vpn_nat)
  3: - 1f000000 (ca8da0f8) Stateless verifications (asm)
  4:      0 (ca8aa0c0) fw VM outbound (fw)
  5:  2000000 (cb1c26e0) vpn policy outbound (vpn_pol)
  6: 10000000 (ca8eb728) SecureXL outbound (secxl)
  7: 20000000 (cb1c2164) vpn encrypt (vpn)
  8: 70000000 (ca8c6020) fwmonitor (i/f side)
  9: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
monitor: monitoring (control-C to stop)
13
^C
monitor: caught sig 2
monitor: unloading
```

Figure 20: `fw monitor` – relative positioning using a module number

! Please note that the relative positions, the number and the order of the modules is in no way fixed. Every change of the configuration or installed products may change this. If you are using relative number you should use `fw ctl chain` to verify the positions you intended to use. Another possibility is to use [aliases](#) for the modules. Even if the position of the module may change the alias remains the same.

## Relative position using an Alias

Another possibility to specify the position of the `fw monitor` module is to use a modules alias (shown in parenthesis). Compared to the relative positioning by numbers you have the additional possibility to decide whether you want to insert the `fw monitor` module *before* or *after* the module you specified. This can be done using `+` or `-` in front of the module alias:

```
[Expert@cpmodule]# fw ctl chain
in chain (9):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 2000000 (cb1c1c64) vpn decrypt (vpn)
  2: - 1ffffff6 (ca8da0f8) Stateless verifications (asm)
  3: - 1ffffff0 (cb1c17f0) vpn decrypt verify (vpn_ver)
  4: - 1000000 (ca8eb688) SecureXL connection syn (secxl_sync)
  5:          0 (ca8aa0c0) fw VM inbound (fw)
  6:  2000000 (cb1c2aa0) vpn policy inbound (vpn_pol)
  7: 10000000 (ca8eb728) SecureXL inbound (secxl)
  8: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (8):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 1fffffff (cb1c16fc) vpn nat outbound (vpn_nat)
  2: - 1f000000 (ca8da0f8) Stateless verifications (asm)
  3:          0 (ca8aa0c0) fw VM outbound (fw)
  4:  2000000 (cb1c26e0) vpn policy outbound (vpn_pol)
  5: 10000000 (ca8eb728) SecureXL outbound (secxl)
  6: 20000000 (cb1c2164) vpn encrypt (vpn)
  7: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
```

Figure 21: `fw ctl chain` – module aliases

In the following example we are inserting the fw monitor chain module *before* (-) SecureXL connection synchronization (secxl\_sync):

```
[Expert@cpmodule]# fw monitor -pi -secxl_sync -o dump.cap
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
in chain (11):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 2000000 (cb1c1c64) vpn decrypt (vpn)
  2: - 1ffffff6 (ca8da0f8) Stateless verifications (asm)
  3: - 1ffffff0 (cb1c17f0) vpn decrypt verify (vpn_ver)
  4: - 1000001 (ca8c6020) fwmonitor (i/f side)
  5: - 1000000 (ca8eb688) SecureXL connection syn (secxl_sync)
  6:      0 (ca8aa0c0) fw VM inbound (fw)
  7:  2000000 (cb1c2aa0) vpn policy inbound (vpn_pol)
  8: 10000000 (ca8eb728) SecureXL inbound (secxl)
  9: 70000000 (ca8c6020) fwmonitor (IP side)
 10: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (10):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: -70000000 (ca8c6020) fwmonitor (IP side)
  2: - 1fffffff (cb1c16fc) vpn nat outbound (vpn_nat)
  3: - 1f000000 (ca8da0f8) Stateless verifications (asm)
  4:      0 (ca8aa0c0) fw VM outbound (fw)
  5:  2000000 (cb1c26e0) vpn policy outbound (vpn_pol)
  6: 10000000 (ca8eb728) SecureXL outbound (secxl)
  7: 20000000 (cb1c2164) vpn encrypt (vpn)
  8: 70000000 (ca8c6020) fwmonitor (i/f side)
  9: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
monitor: monitoring (control-C to stop)
48
^C
monitor: caught sig 2
monitor: unloading
```

Figure 22: fw monitor – relative positioning using module aliases

## Absolute position

Although in most cases the use of aliases for positioning is recommended it is also possible to use absolute positioning. This allows you to specify the position to insert the fw monitor module using its absolute position. Every chain module as such a position and the kernel sorts them according to this position. The absolute position is printed in hex after the relative position. Please note that chain positions *before* the virtual machine are *negative* values:

```
[Expert@cpmodule]# fw ctl chain
in chain (9):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 2000000 (cb1c1c64) vpn decrypt (vpn)
  2: - 1ffffff6 (ca8da0f8) Stateless verifications (asm)
  3: - 1ffffff0 (cb1c17f0) vpn decrypt verify (vpn_ver)
  4: - 10000000 (ca8eb688) SecureXL connection syn (secxl_sync)
  5:          0 (ca8aa0c0) fw VM inbound (fw)
  6:  2000000 (cb1c2aa0) vpn policy inbound (vpn_pol)
  7: 10000000 (ca8eb728) SecureXL inbound (secxl)
  8: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (8):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 1fffffff (cb1c16fc) vpn nat outbound (vpn_nat)
  2: - 1f000000 (ca8da0f8) Stateless verifications (asm)
  3:          0 (ca8aa0c0) fw VM outbound (fw)
  4:  2000000 (cb1c26e0) vpn policy outbound (vpn_pol)
  5: 10000000 (ca8eb728) SecureXL outbound (secxl)
  6: 20000000 (cb1c2164) vpn encrypt (vpn)
  7: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
```

Figure 23: fw ctl chain – absolute positions



Please note that the absolute position is a property of the kernel module assigned by Check Point R&D: This value may change in future versions.

```
[Expert@cpmodule]# fw monitor -pi -0x1ffffe0 -pO 0x2000001
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
in chain (11):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 2000000 (cb1c1c64) vpn decrypt (vpn)
  2: - 1fffff6 (ca8da0f8) Stateless verifications (asm)
  3: - 1fffff0 (cb1c17f0) vpn decrypt verify (vpn_ver)
  4: - 1ffffe0 (ca8c6020) fwmonitor (i/f side)
  5: - 1000000 (ca8eb688) SecureXL connection syn (secxl_sync)
  6:      0 (ca8aa0c0) fw VM inbound (fw)
  7:  2000000 (cb1c2aa0) vpn policy inbound (vpn_pol)
  8:  10000000 (ca8eb728) SecureXL inbound (secxl)
  9:  70000000 (ca8c6020) fwmonitor (IP side)
 10:  7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (10):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: -70000000 (ca8c6020) fwmonitor (IP side)
  2: - 1fffffff (cb1c16fc) vpn nat outbound (vpn_nat)
  3: - 1f000000 (ca8da0f8) Stateless verifications (asm)
  4:      0 (ca8aa0c0) fw VM outbound (fw)
  5:  2000000 (cb1c26e0) vpn policy outbound (vpn_pol)
  6:  10000000 (ca8eb728) SecureXL outbound (secxl)
  7:  20000000 (cb1c2164) vpn encrypt (vpn)
  8:  20000001 (ca8c6020) fwmonitor (i/f side)
  9:  7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
monitor: monitoring (control-C to stop)
^C
monitor: caught sig 2
monitor: unloading
```

Figure 24: fw monitor – absolute positioning

**!** fw ctl chain does not show the preceding 0x to specify hex numbers. Nevertheless you have to add a preceding 0x in front of the number to use it with fw monitor.

## All positions

A new option in NG with Application Intelligence (FP4) allows you to insert `fw monitor` modules between *all* modules. This gives you the ability to follow a packet through the FireWall-1 kernel module chain. The position where the packet was captured is printed after the direction (module in parenthesis) and also written down to the capture file if the `-o` option is used..

```
[Expert@cpmodule]# fw monitor -p all
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
in chain (9):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 2000000 (cblc1c64) vpn decrypt (vpn)
  2: - 1fffff6 (ca8da0f8) Stateless verifications (asm)
  3: - 1fffff0 (cblc17f0) vpn decrypt verify (vpn_ver)
  4: - 1000000 (ca8eb688) SecureXL connection syn (secxl_sync)
  5: 0 (ca8aa0c0) fw VM inbound (fw)
  6: 2000000 (cblc2aa0) vpn policy inbound (vpn_pol)
  7: 10000000 (ca8eb728) SecureXL inbound (secxl)
  8: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
out chain (8):
  0: -7f800000 (ca8d9698) IP Options Strip (ipopt_strip)
  1: - 1ffffff (cblc16fc) vpn nat outbound (vpn_nat)
  2: - 1f00000 (ca8da0f8) Stateless verifications (asm)
  3: 0 (ca8aa0c0) fw VM outbound (fw)
  4: 2000000 (cblc26e0) vpn policy outbound (vpn_pol)
  5: 10000000 (ca8eb728) SecureXL outbound (secxl)
  6: 20000000 (cblc2164) vpn encrypt (vpn)
  7: 7f800000 (ca8d98e4) IP Options Restore (ipopt_res)
monitor: monitoring (control-C to stop)
eth0:i0 (IP Options Strip)[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:i1 (vpn decrypt)[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:i2 (Stateless verifications)[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:i3 (vpn decrypt verify)[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:i4 (SecureXL connection syn)[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:i5 (fw VM inbound ) [84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:I6 (vpn policy inbound)[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:I7 (SecureXL inbound)[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:I8 (IP Options Restore)[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:I9 (Chain End)[84]: 172.16.1.1 -> 172.16.1.2 (ICMP) len=84 id=11936
ICMP: type=8 code=0 echo request id=16436 seq=256
eth0:o0 (IP Options Strip)[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=49943
ICMP: type=0 code=0 echo reply id=16436 seq=256
eth0:o1 (vpn nat outbound)[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=49943
ICMP: type=0 code=0 echo reply id=16436 seq=256
eth0:o2 (Stateless verifications)[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=49943
ICMP: type=0 code=0 echo reply id=16436 seq=256
eth0:o3 (fw VM outbound)[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=49943
ICMP: type=0 code=0 echo reply id=16436 seq=256
eth0:o4 (vpn policy outbound)[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=49943
ICMP: type=0 code=0 echo reply id=16436 seq=256
eth0:o5 (SecureXL outbound)[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=49943
ICMP: type=0 code=0 echo reply id=16436 seq=256
eth0:o6 (vpn encrypt)[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=49943
ICMP: type=0 code=0 echo reply id=16436 seq=256
eth0:o7 (IP Options Restore)[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=49943
ICMP: type=0 code=0 echo reply id=16436 seq=256
eth0:o8 (Chain End)[84]: 172.16.1.2 -> 172.16.1.1 (ICMP) len=84 id=49943
ICMP: type=0 code=0 echo reply id=16436 seq=256
^C
monitor: caught sig 2
monitor: unloading
```

Figure 25: `fw monitor` – all positions

**!** It is not recommended to use this option on a high-loaded production machine, except that you add specific filters to reduce the output. Without a filter it may output more than 15 captured packets (in this example 8 packets inbound and 9 packets outbound) per packet passing the firewall.

## fw monitor filters

`fw monitor` filters are using a subset of `INSPECT` to specify the packets to be captured. The general syntax is:

```
accept expression;
```

*Figure 26: fw monitor filter expression – general syntax*

**!** “accept” in `fw monitor` filters does not mean that packets are actually accepted by the firewall. `fw monitor` captures all packets which are accepted by the filter and discards the rest. A filter like `accept;` (capturing all packets) will in no way change the behavior of the FireWall and its rulebase.

The complexity of an expression can vary from a simple test (checking for a specific value at a specific offset) to a complex expression using different checks and logical operators.

## Simple Checks

Simple checks are used to check for a value at a specific offset in the packet:

```
[ offset : length , order ] relational-operator value
```

*Figure 27: fw monitor simple checks – general syntax*

`offset` specifies the offset relative to the beginning of the IP packet from where the value should be read.

`length` specifies the number of bytes and can be 1 (byte), 2 (word) or 4 (dword). If `length` is not specified `fw monitor` assumes 4 (dword).

`order` is used to specify the byte order. Possible values are `b` (big endian) or `l` (little endian, or host order). If `order` is not specified little endian byte order is assumed.

`relational-operator` is a [relational operator](#) to express the relation between the packet data and the value.

`value` is one of the [data types](#) known to `INSPECT` (e.g. an IP address or an integer).

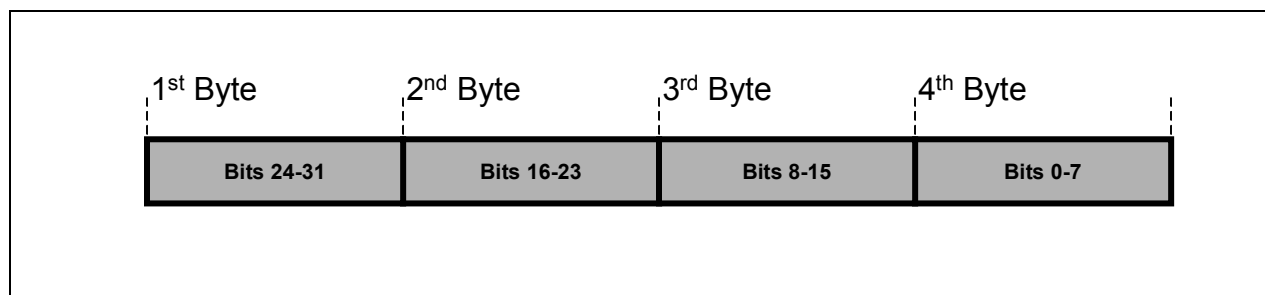


Figure 28: Big Endian byte order

Big Endian order means that the most significant byte as the lowest address (the word is stored 'big-endian-first')

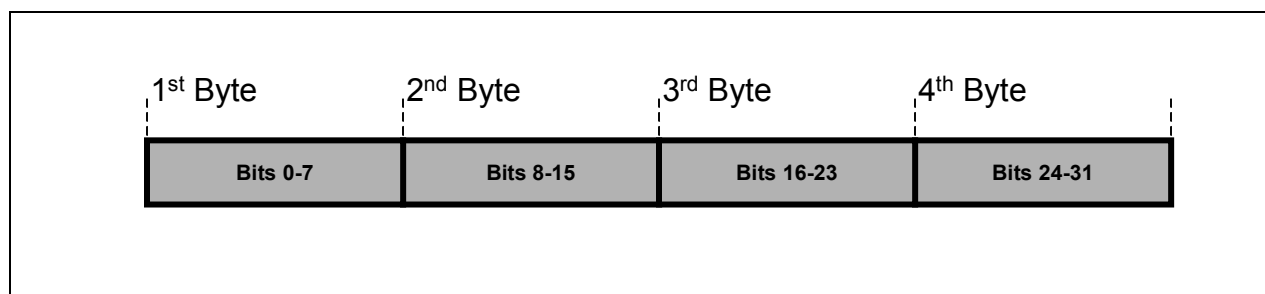


Figure 29: Little Endian byte order

Little Endian order means that bytes at lower addresses have lower significance (the word is stored 'little-endian-first')

Please note that the byte order is processor architecture dependent. On processors like Motorola 68xxx big endian byte order is used. Little endian byte order is used e.g. by Intel 386 and compatible processors. There are also processors which are able to work with both byte orders (e.g. PowerPC). You can find more information about byte orders at [An Essay on Endian Order](#).

To filter for specific values it is essential to know *where* these values are stored. Therefore it is important to know the different protocols and their fields:

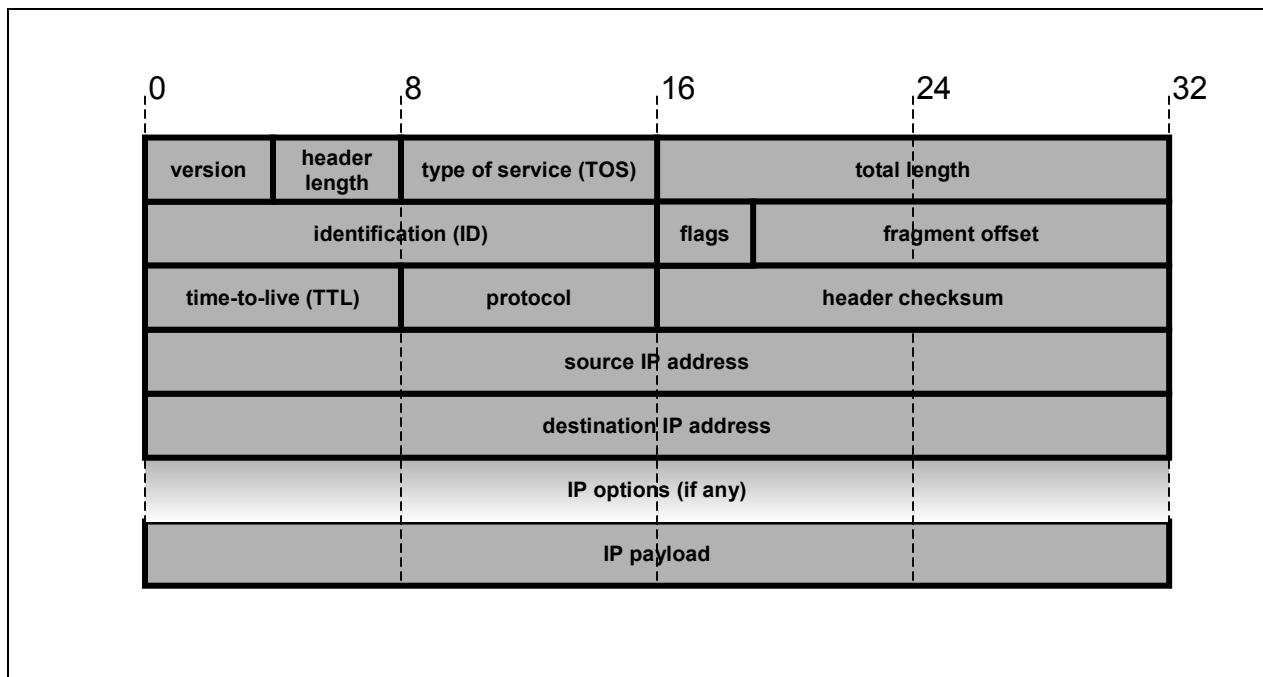


Figure 30: IP protocols – IP header

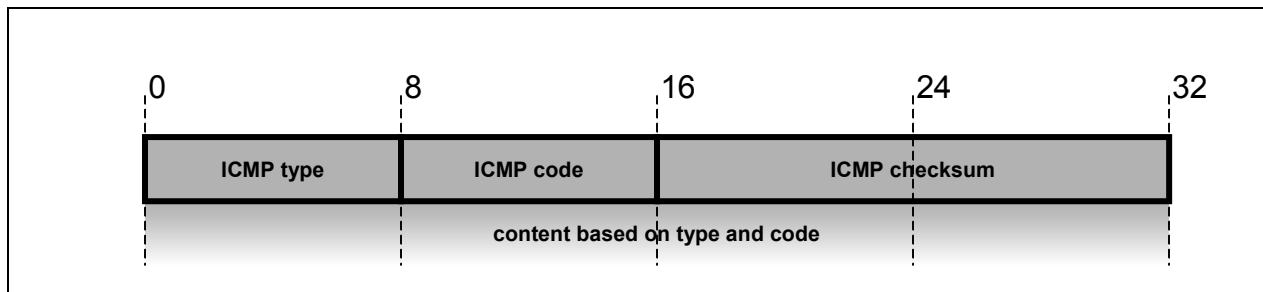


Figure 31: IP protocols – ICMP header

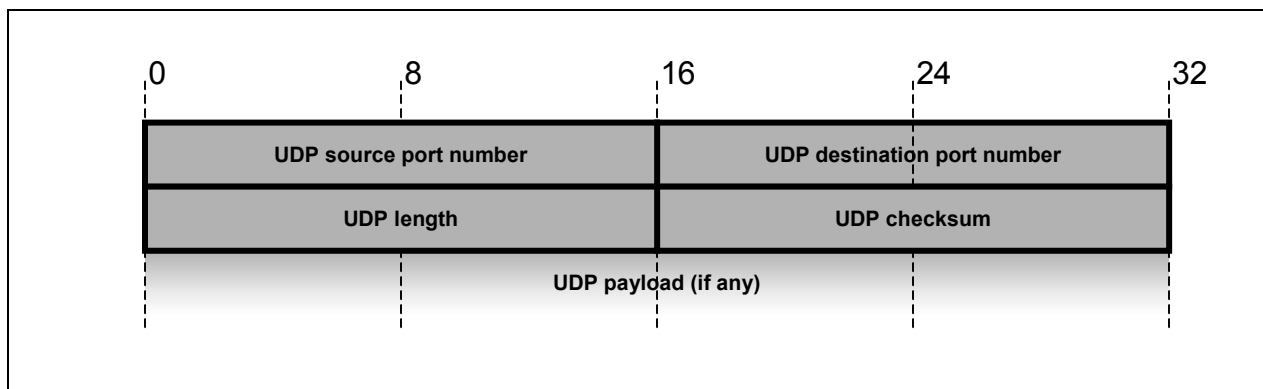


Figure 32: IP protocols – UDP header

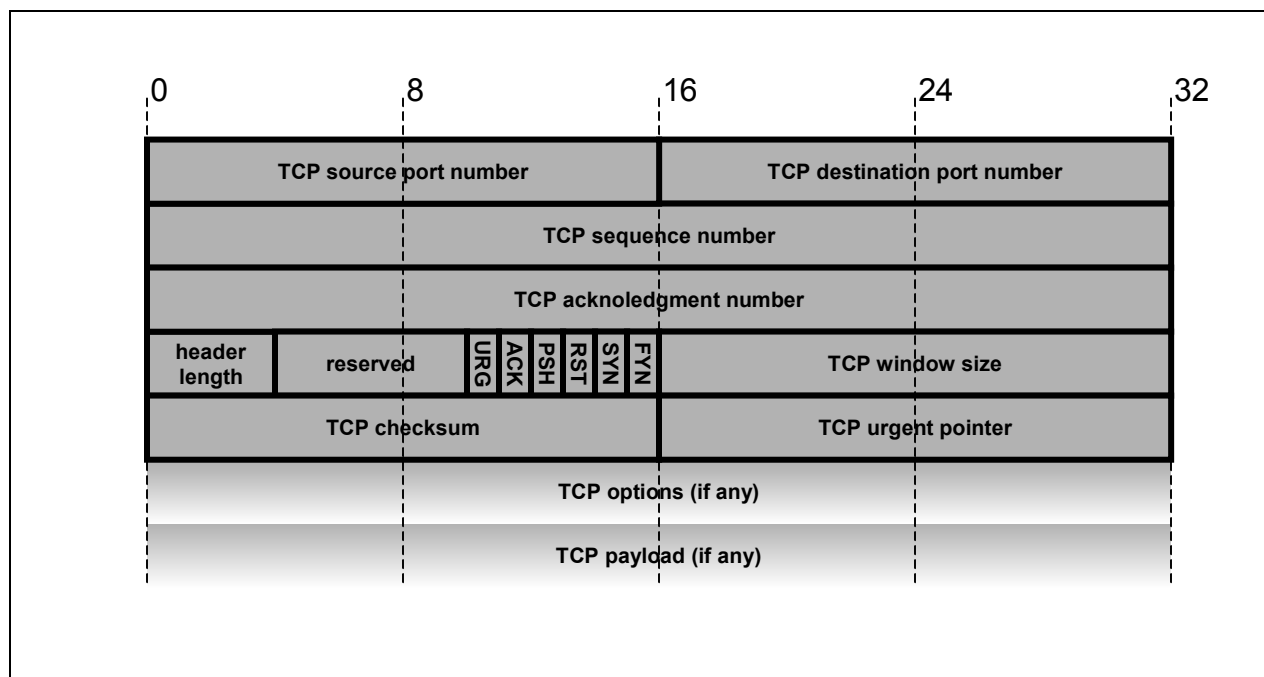


Figure 33: IP protocols – TCP header

Simple Checks can be used for a wide variety of checks. Some examples:

Filter on source or destination IP address. The IP addresses are stored as dwords at offset 12 (source address) and 16 (destination address):

address	filter expression
source	accept [12, b]=172.16.1.2;
destination	accept [16, b]=10.2.4.12;

Figure 34: fw monitor simple checks – IP addresses

**!** Please note the use of IP addresses instead of simple numbers in the example above. INSPECT “knows” IP addresses and converts them automatically to an integer. There is no need to do this manually although this is possible. Please refer to the Check Point Reference Guide for more information.

Filter on the IP protocol. The IP protocol is stored as a byte at offset 9 in the IP packet:

IP protocol	filter expression
ICMP	accept [9:1] = 1;
TCP	accept [9:1] = 6;
UDP	accept [9:1] = 17;
ESP	accept [9:1] = 50;

Figure 35: fw monitor simple checks – IP protocol examples

Filter on ports (when using TCP or UDP). The ports are stored as a word at offset 20 (source port) and 22 (destination port):

IP protocol	filter expression
source port HTTP	accept [20:2,b]=80;
destination port HTTP	accept [22:2,b]=80;
source port FTP (control channel)	accept [20:2,b]=21;
destination port FTP (control channel)	accept [22:2,b]=21;

Figure 36: fw monitor simple checks – TCP/UDP ports examples

## Network checks

INSPECT allows you to check whether a specific IP address belongs to a specified network. There are two possibilities to achieve this:

```
accept netof [IP Address] = [Network Address];
```

Figure 37: simple network checks – expression syntax

```
accept netof src = 172.16.1.0;
```

Figure 38: simple network checks – example

Although this is very easy to use and to remember it has one limitation: It is not possible to define the subnet mask to be used. Instead the subnet mask is automatically determined by the IP address.

The second possibility allows you to specify an IP range – therefore enabling you to filter not only for none-implied subnet masks but even for IP address ranges:

```
[listname] = { [ IP address ranges ] };  
accept [IP address] in [listname];
```

Figure 39: advanced network checks – expression syntax

```
internal = { <172.16.1.0, 172.16.1.255>, <172.16.8.0,172.16.8.255> };  
accept (src in internal);
```

Figure 40: advanced network checks – example

Please note the it is possible to include multiple networks in a list. This allows you e.g. to define all your internal networks and use the resulting list in the filter expression.

## Data types

INSPECT knows several native data types. Just some of them are useful for fw monitor:

Hexadecimal Integers	A number beginning with 0x	e.g. 0x5ab4
Octal Integers	A Number beginning with 0	e.g. 0777
Decimal Integers	Any other number	e.g. 23
IP Address	Four decimal integers separated by three periods	e.g. 172.45.2.4

Figure 41: fw monitor – data types

## Logical and Relational Operators

In addition to the single expressions testing for equality it is possible to combine different expressions using several logical and relational operators

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
= or is	Equal
!= or is not	Not equal

Figure 42: fw monitor – Relational Operators

,	Logical AND
or	Logical Or
xor	Logical XOR
not	Logical NOT

Figure 43: fw monitor – Logical Operators

**!** Please note that INSPECT uses another operator precedence than e.g. C. In INSPECT the expression “a , b or c” is understood as “a , ( b or c)”. That is, or takes precedence over , (and). Parentheses “(“ and “)” – can be used to force operator precedence. There is no penalty for redundant parentheses..

Using relational and logical operators it is easily possible to build complex capture filters:

Everything except http	<code>accept not ( [20:2,b]=80 or [22:2,b]=80);</code>
Every non-root TCP connection	<code>accept [9:1]=9 , th sport &gt; 1024;</code>
Every TCP packet between 10.2.4.12 and 172.16.1.2	<code>accept [9:1]=9 , (([12:4,b]=10.2.4.12 , [16:4,b]=172.16.1.2) or ([12:4,b]=172.16.1.2 , [16:4,b]=10.2.4.12));</code>

Figure 44: fw monitor – example of logical and relational operators

**!** Even if fw monitor filters allow you to specify complex filters it’s normally not advisable. In many cases a too complex filter might not capture packets you are interested in. It’s normally better to just filter out bulk traffic you’re not interested in (e.g. HTTP) and do the granular filtering later on (e.g. using Ethereal on files generated with -o). An exception is using fw monitor on high-loaded gateways. There you might have simply no choice but to reduce the amount of traffic being captured.

## Macros

Because all offsets, lengths and orders are hard to remember fw monitor offers an more intuitive way of specifying the desired field:

Field	Macro	Expression
source address	src	[12:4,b]
destination address	dst	[16:4,b]
source port	sport	[20:2,b]
destination port	dport	[22:2,b]

Figure 45: fw monitor – built-in macros

Using these macros it very easy to define filters (and understanding them again a few weeks later!):

Everything except http	accept not ( sport=80 or dport=80);
All TCP packets sdn between host 10.2.4.12 and 172.16.1.2	accept [9:1]=9 , ((src=10.2.4.12 , dst=172.16.1.2) or (src=172.16.1.2 , dst=10.2.4.12));

Figure 46: fw monitor – example of logical and relational operators using macros

These macros are not a part of INSPECT. INSPECT (and therefore fw monitor as well) uses a C preprocessor to replace named macros with their low-level equivalents. If you are using filters on the command line (using -e) fw monitor creates a new file with the definitions above and appends your filter expression. The file is called \$FWDIR/tmp/monitorfilter.pf:

```
[Expert@cpmodule]# fw monitor -e 'accept src=10.2.4.12 or dst=10.2.4.12;'
monitor: getting filter (from command line)
monitor: compiling
monitorfilter:
Compiled OK.
monitor: loading
monitor: monitoring (control-C to stop)
^C
monitor: caught sig 2
monitor: unloading
[Expert@cpmodule]# cat $FWDIR/tmp/monitorfilter.pf
#define src ip_src
#define dst ip_dst
#define sport th_sport
#define dport th_dport
#include "tcpip.def"
accept src=10.2.4.12 or dst=10.2.4.12;
```

Figure 47: monitorfilter.pf

The last line of monitorfilter.pf is your filter expression (or multiple expressions if you used multiple -e expressions). The first four lines are defining src, dst, sport and dport. These are defined using macros again. This macros are defined in tcpip.def. The fifth line includes something called "tcpip.def".

As mentioned earlier INSPECT uses a C preprocessor. Therefore you can use C preprocessor directives overall in your fw monitor scripts (on the command line as well as in files).

`src` for example is defined as `ip_src`. `ip_src` is defined as `[12, b]` in the included `tcpip.def`. `tcpip.def` can be found in `$FWDIR/lib` and is a very good resource for useful definitions. You can include other files in `$FWDIR/lib` as well if you like.

If you use `fw monitor` you can create your own “library” and include it (e.g. using the `-f` option). This allows you to define your own definitions of commands and expressions you are using on a regular basis. Take a look at [Useful macros in tcpip.def](#) for a collection of useful expressions.

**!** Please note that predefined macros (like `src`, `dport`, `sport` ...) are only automatically defined if you are using expressions on the command line. If you are using files or standard input for providing filter expressions you have to define the macros for yourself or include them using the `#include` directive manually.

## Useful macros in tcpip.def

Macro	Example	Description
<code>ip_p</code>	<code>ip_p = PROTO_icmp</code>	IP protocol
<code>ip_len</code>	<code>ip_len &gt; 128</code>	Length of the IP packet
<code>ip_ttl</code>	<code>ip_ttl &lt; 31</code>	Time to live
<code>ip_src</code>	<code>ip_src = 172.16.4.3</code>	Source IP address
<code>ip_dst</code>	<code>ip_dst = 10.2.4.12</code>	Destination IP address
<code>th_sport</code>	<code>th_sport &gt; 1024</code>	TCP source port
<code>th_dport</code>	<code>th_dport = 80</code>	TCP destination port
<code>th_seq</code>	<code>th_seq &gt; 54245</code>	TCP sequence number
<code>th_ack</code>	<code>th_ack &lt; 349274</code>	TCP acknowledged number
<code>th_flags</code>	<code>th_flags = (TH_SYN &amp; TH_ACK)</code>	TCP flags
<code>uh_sport</code>	<code>uh_sport &gt; 1024</code>	UDP source port
<code>uh_dport</code>	<code>uh_dport = 53</code>	UDP destination port
<code>icmp_type</code>	<code>icmp_type = ICMP_UNREACH</code>	ICMP type
<code>icmp_code</code>	<code>icmp_code = ICMP_UNREACH_PORT</code>	ICMP code

**!** Please note that this is just a small amount of macros defined in `tcpip.def`. Take a look at `tcpip.def` for yourself to find other useful expressions you may want to use.

**!** Do not modify anything in `tcpip.def` or in any other `*.def` file in `$FWDIR/lib` by yourself. Check Point does not support any configuration with changed `*.def` files. An exception are modifications done together with Check Point Support (according to a Service Request) or found on SecureKnowledge.

## More information about INSPECT

Refer to the *Check Point Reference Guide* for a complete overview about INSPECT. Reading the `*.def` files in `$FWDIR/lib` will give you a good overview about the possibilities as well.

## Inspect fw monitor files

The recommended tool for analyzing `fw monitor` capture files is Ethereal ([Using Ethereal to inspect fw monitor files](#)) or CPEThereal ([Using CPEThereal to inspect fw monitor files](#)). Nevertheless `fw monitor` capture files can be inspected with every tool which is able to read the snoop file format ([Snoop file format \(RFC 1761\)](#)).

### Using snoop to inspect fw monitor files

`snoop` is a tool normally found on Sun Solaris machines. `snoop` allows you to capture packets and to examine them. As described in [Write output to file](#) `fw monitor` writes its capture files in the file format used by `snoop`. This allows us to use `snoop` to decode the files later on. This means you can generate the `fw monitor` files on one machine and examine them on another machine using all of `snoop`'s functions including verbose output and filtering.

**!** `snoop` is only available on Sun Solaris. For other platforms refer to [Using tcpdump to inspect fw monitor files](#) or [Using Ethereal to inspect fw monitor files](#).

The following example shows how an `fw monitor` capture file (two ICMP Echo Request and ICMP Echo Replies, PreIn/PostIn and PreOut/PostOut) which was generated on a Linux machine is inspected on a Sun:

```
bash-2.03# snoop -i fwmonitor.cap
 1  0.00000  172.16.1.1 -> 172.16.1.2  ICMP Echo request (ID: 51470 Sequence number: 256)
 2  0.00000  172.16.1.1 -> 172.16.1.2  ICMP Echo request (ID: 51470 Sequence number: 256)
 3  0.00000  172.16.1.2 -> 172.16.1.1  ICMP Echo reply (ID: 51470 Sequence number: 256)
 4  0.00000  172.16.1.2 -> 172.16.1.1  ICMP Echo reply (ID: 51470 Sequence number: 256)
 5  0.00000  172.16.1.1 -> 172.16.1.2  ICMP Echo request (ID: 51470 Sequence number: 512)
 6  0.00000  172.16.1.1 -> 172.16.1.2  ICMP Echo request (ID: 51470 Sequence number: 512)
 7  0.00000  172.16.1.2 -> 172.16.1.1  ICMP Echo reply (ID: 51470 Sequence number: 512)
 8  0.00000  172.16.1.2 -> 172.16.1.1  ICMP Echo reply (ID: 51470 Sequence number: 512)
```

Figure 48: Inspecting fw monitor files with snoop

```

bash-2.03# snoop -V -i fwmonitor.cap
1 0.00000 172.16.1.1 -> 172.16.1.2 ETHER Type=0800 (IP), size = 98 bytes
1 0.00000 172.16.1.1 -> 172.16.1.2 IP D=172.16.1.2 S=172.16.1.1 LEN=84, ID=47628
1 0.00000 172.16.1.1 -> 172.16.1.2 ICMP Echo request (ID: 51470 Sequence number: 256)
-----
2 0.00000 172.16.1.1 -> 172.16.1.2 ETHER Type=0800 (IP), size = 98 bytes
2 0.00000 172.16.1.1 -> 172.16.1.2 IP D=172.16.1.2 S=172.16.1.1 LEN=84, ID=47628
2 0.00000 172.16.1.1 -> 172.16.1.2 ICMP Echo request (ID: 51470 Sequence number: 256)
-----
3 0.00000 172.16.1.2 -> 172.16.1.1 ETHER Type=0800 (IP), size = 98 bytes
3 0.00000 172.16.1.2 -> 172.16.1.1 IP D=172.16.1.1 S=172.16.1.2 LEN=84, ID=4875
3 0.00000 172.16.1.2 -> 172.16.1.1 ICMP Echo reply (ID: 51470 Sequence number: 256)
-----
4 0.00000 172.16.1.2 -> 172.16.1.1 ETHER Type=0800 (IP), size = 98 bytes
4 0.00000 172.16.1.2 -> 172.16.1.1 IP D=172.16.1.1 S=172.16.1.2 LEN=84, ID=4875
4 0.00000 172.16.1.2 -> 172.16.1.1 ICMP Echo reply (ID: 51470 Sequence number: 256)
-----
5 0.00000 172.16.1.1 -> 172.16.1.2 ETHER Type=0800 (IP), size = 98 bytes
5 0.00000 172.16.1.1 -> 172.16.1.2 IP D=172.16.1.2 S=172.16.1.1 LEN=84, ID=47629
5 0.00000 172.16.1.1 -> 172.16.1.2 ICMP Echo request (ID: 51470 Sequence number: 512)
-----
6 0.00000 172.16.1.1 -> 172.16.1.2 ETHER Type=0800 (IP), size = 98 bytes
6 0.00000 172.16.1.1 -> 172.16.1.2 IP D=172.16.1.2 S=172.16.1.1 LEN=84, ID=47629
6 0.00000 172.16.1.1 -> 172.16.1.2 ICMP Echo request (ID: 51470 Sequence number: 512)
-----
7 0.00000 172.16.1.2 -> 172.16.1.1 ETHER Type=0800 (IP), size = 98 bytes
7 0.00000 172.16.1.2 -> 172.16.1.1 IP D=172.16.1.1 S=172.16.1.2 LEN=84, ID=4876
7 0.00000 172.16.1.2 -> 172.16.1.1 ICMP Echo reply (ID: 51470 Sequence number: 512)
-----
8 0.00000 172.16.1.2 -> 172.16.1.1 ETHER Type=0800 (IP), size = 98 bytes
8 0.00000 172.16.1.2 -> 172.16.1.1 IP D=172.16.1.1 S=172.16.1.2 LEN=84, ID=4876
8 0.00000 172.16.1.2 -> 172.16.1.1 ICMP Echo reply (ID: 51470 Sequence number: 512)

```

Figure 49: Inspecting fw monitor files with snoop – summary output

```
bash-2.03# snoop -v -c 1 -i fwmonitor.cap
ETHER: ----- Ether Header -----
ETHER:
ETHER: Packet 1 arrived at 8:26:43.00
ETHER: Packet size = 98 bytes
ETHER: Destination = 69:31:65:74:68:30, (multicast)
ETHER: Source = 0:0:0:0:0:0,
ETHER: Ethertype = 0800 (IP)
ETHER:
IP: ----- IP Header -----
IP:
IP: Version = 4
IP: Header length = 20 bytes
IP: Type of service = 0x00
IP:   xxx. .... = 0 (precedence)
IP:   ...0 .... = normal delay
IP:   .... 0... = normal throughput
IP:   .... .0.. = normal reliability
IP: Total length = 84 bytes
IP: Identification = 47628
IP: Flags = 0x4
IP:   .1.. .... = do not fragment
IP:   ..0. .... = last fragment
IP: Fragment offset = 0 bytes
IP: Time to live = 64 seconds/hops
IP: Protocol = 1 (ICMP)
IP: Header checksum = 2679
IP: Source address = 172.16.1.1, 172.16.1.1
IP: Destination address = 172.16.1.2, 172.16.1.2
IP: No options
IP:
ICMP: ----- ICMP Header -----
ICMP:
ICMP: Type = 8 (Echo request)
ICMP: Code = 0 (ID: 51470 Sequence number: 256)
ICMP: Checksum = 2be5
ICMP:

1 packets captured
bash-2.03#
```

Figure 50: Inspecting fw monitor files with snoop – verbose output

! Especially when working in verbose mode (-v) it is recommended to display only a few packets. Use -c to limit the number of packets or use filter expressions. snoop filter expressions are not discussed in this paper. Refer to the [snoop man page](#) for further information.

! This paper does not cover advanced snoop usage including things like filtering, converting etc. You can find further information at [The Secrets of Snoop](#).

## Using tcpdump to inspect fw monitor files

tcpdump has a similar functionality like snoop. Compared to snoop it runs on many platforms including Linux, IPSO, FreeBSD .... tcpdump uses a slightly different file format than snoop. Therefore it is not possible to open fw monitor files with tcpdump directly:

```
brain:/home/udos # tcpdump -r fwmonitor.cap
tcpdump: bad dump file format
```

Figure 51: Inspecting fw monitor files with tcpdump – bad file format

This means we have to convert the fw monitor capture file to a file format which tcpdump is able to read. One possibility is to use editcap (see [editcap](#) for further information). editcap is a tool from the Ethereal package which is able to convert between different capture file formats. By default editcap converts any input file to an output file in tcpdump format (tcpdump actually uses the libpcap file format. Visit the [tcpdump/libpcap homepage](#) for further information).

```
brain:/home/udos # editcap fwmonitor.cap tcpdump.cap
```

Figure 52: editcap – Converting from snoop file format to tcpdump (libpcap) file format

This will give you a capture file named tcpdump.cap with the same content as fwmonitor.cap which can be read by tcpdump:

```
brain:/home/udos # tcpdump -r tcpdump.cap
08:26:43.000000 172.16.1.1 > 172.16.1.2: icmp: echo request (DF)
08:26:43.000000 172.16.1.1 > 172.16.1.2: icmp: echo request (DF)
08:26:43.000000 172.16.1.2 > 172.16.1.1: icmp: echo reply
08:26:43.000000 172.16.1.2 > 172.16.1.1: icmp: echo reply
08:26:43.000000 172.16.1.1 > 172.16.1.2: icmp: echo request (DF)
08:26:43.000000 172.16.1.1 > 172.16.1.2: icmp: echo request (DF)
08:26:43.000000 172.16.1.2 > 172.16.1.1: icmp: echo reply
08:26:43.000000 172.16.1.2 > 172.16.1.1: icmp: echo reply
```

Figure 53: Inspecting fw monitor files with tcpdump – summary output

Like snoop, tcpdump offers the possibility to output the data in an even more detailed way. This can be achieved by using verbose options. tcpdump offers three verbose options – -v, -vv and -vvv – with different verbose levels:

```
brain:/home/udos # tcpdump -v -r tcpdump.cap
08:26:43.000000 172.16.1.1 > 172.16.1.2: icmp: echo request (DF) (ttl 64, id 47628, len 84)
08:26:43.000000 172.16.1.1 > 172.16.1.2: icmp: echo request (DF) (ttl 64, id 47628, len 84)
08:26:43.000000 172.16.1.2 > 172.16.1.1: icmp: echo reply (ttl 255, id 4875, len 84)
08:26:43.000000 172.16.1.2 > 172.16.1.1: icmp: echo reply (ttl 255, id 4875, len 84)
08:26:43.000000 172.16.1.1 > 172.16.1.2: icmp: echo request (DF) (ttl 64, id 47629, len 84)
08:26:43.000000 172.16.1.1 > 172.16.1.2: icmp: echo request (DF) (ttl 64, id 47629, len 84)
08:26:43.000000 172.16.1.2 > 172.16.1.1: icmp: echo reply (ttl 255, id 4876, len 84)
08:26:43.000000 172.16.1.2 > 172.16.1.1: icmp: echo reply (ttl 255, id 4876, len 84)
```

Figure 54: Inspecting fw monitor files with tcpdump – verbose output



This paper does not cover advanced tcpdump usage including things like filtering, converting etc. You can find further information at [tcpdump man page](#).

## Using Ethereal to inspect fw monitor files

### Basic Ethereal usage

Ethereal is a graphical tool to analyze and capture network traffic. Ethereal is available on a wide range of platforms and operating systems including all major UNIX flavors (Solaris, Linux, \*BSD ...), Windows (Windows 9x, ME, NT 4, 2000 and XP), Mac OS X and many more. The screenshots in this paper were taken on a Linux machine (for Ethereal). Ethereal reads a wide variety of capture formats including the format used by `fw monitor` (which is in fact the same format as `snoop`). This means you can simply open a `fw monitor` file in Ethereal:

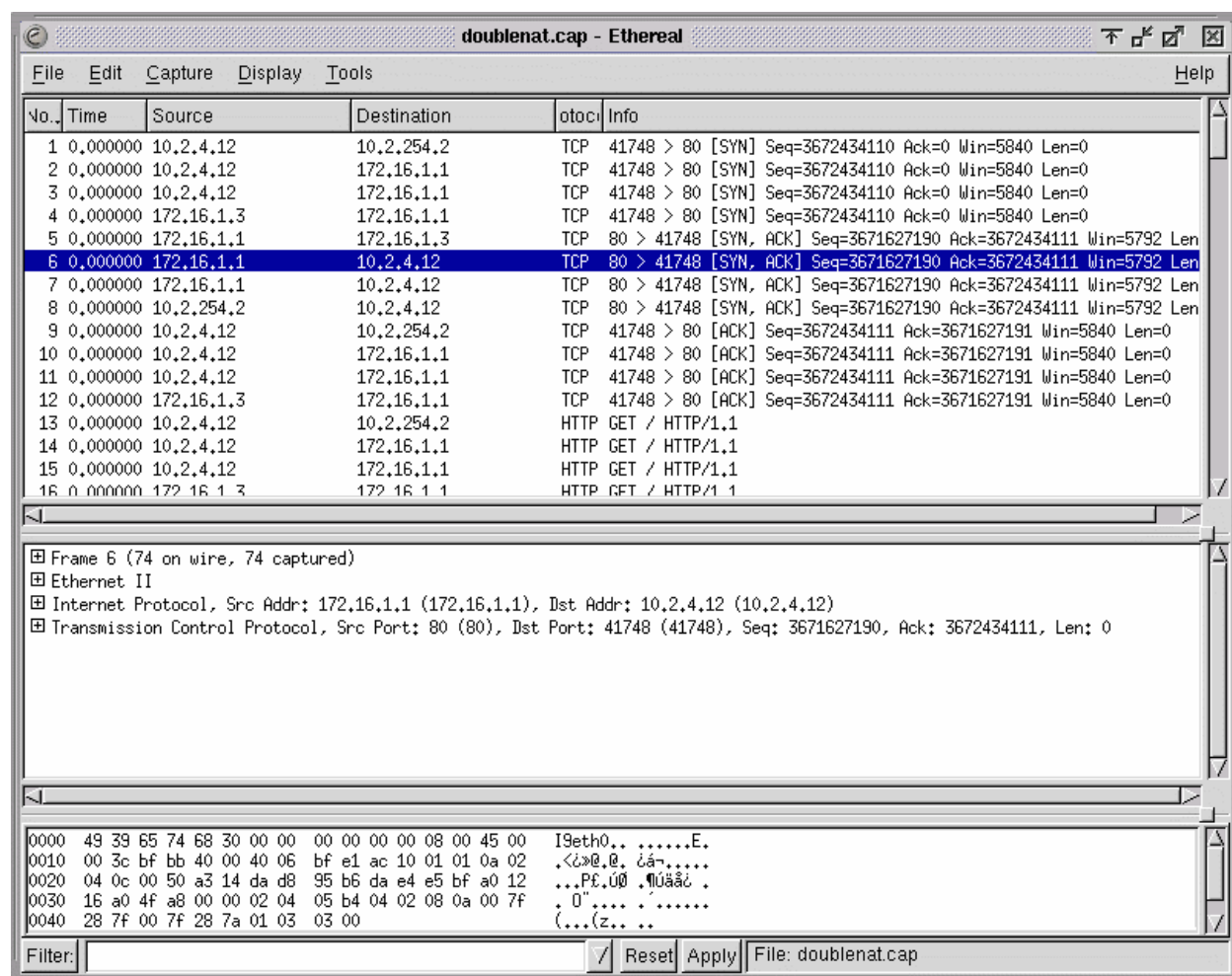


Figure 55: Ethereal – main window

The Ethereal main window consists of three panes. The top pane lists all packets in the opened file. This overview pane lists information like capture time, source- and destination address together with a short (protocol dependent) information:

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.2.4.12	10.2.254.2	TCP	41748 > 80 [SYN] Seq=3672434110 Ack=0 Win=5840 Len=0
2	0.000000	10.2.4.12	172.16.1.1	TCP	41748 > 80 [SYN] Seq=3672434110 Ack=0 Win=5840 Len=0
3	0.000000	10.2.4.12	172.16.1.1	TCP	41748 > 80 [SYN] Seq=3672434110 Ack=0 Win=5840 Len=0
4	0.000000	172.16.1.3	172.16.1.1	TCP	41748 > 80 [SYN] Seq=3672434110 Ack=0 Win=5840 Len=0
5	0.000000	172.16.1.1	172.16.1.3	TCP	80 > 41748 [SYN, ACK] Seq=3671627190 Ack=3672434111 Win=5792 Len=0
6	0.000000	172.16.1.1	10.2.4.12	TCP	80 > 41748 [SYN, ACK] Seq=3671627190 Ack=3672434111 Win=5792 Len=0
7	0.000000	172.16.1.1	10.2.4.12	TCP	80 > 41748 [SYN, ACK] Seq=3671627190 Ack=3672434111 Win=5792 Len=0
8	0.000000	10.2.254.2	10.2.4.12	TCP	80 > 41748 [SYN, ACK] Seq=3671627190 Ack=3672434111 Win=5792 Len=0
9	0.000000	10.2.4.12	10.2.254.2	TCP	41748 > 80 [ACK] Seq=3672434111 Ack=3671627191 Win=5840 Len=0
10	0.000000	10.2.4.12	172.16.1.1	TCP	41748 > 80 [ACK] Seq=3672434111 Ack=3671627191 Win=5840 Len=0
11	0.000000	10.2.4.12	172.16.1.1	TCP	41748 > 80 [ACK] Seq=3672434111 Ack=3671627191 Win=5840 Len=0
12	0.000000	172.16.1.3	172.16.1.1	TCP	41748 > 80 [ACK] Seq=3672434111 Ack=3671627191 Win=5840 Len=0
13	0.000000	10.2.4.12	10.2.254.2	HTTP	GET / HTTP/1.1
14	0.000000	10.2.4.12	172.16.1.1	HTTP	GET / HTTP/1.1
15	0.000000	10.2.4.12	172.16.1.1	HTTP	GET / HTTP/1.1
16	0.000000	172.16.1.3	172.16.1.1	HTTP	GET / HTTP/1.1

Figure 56: Ethereal – overview pane

The pane in the middle shows protocol specific decodes of the different packet layers. This decode pane uses a tree view to display the different protocol values:

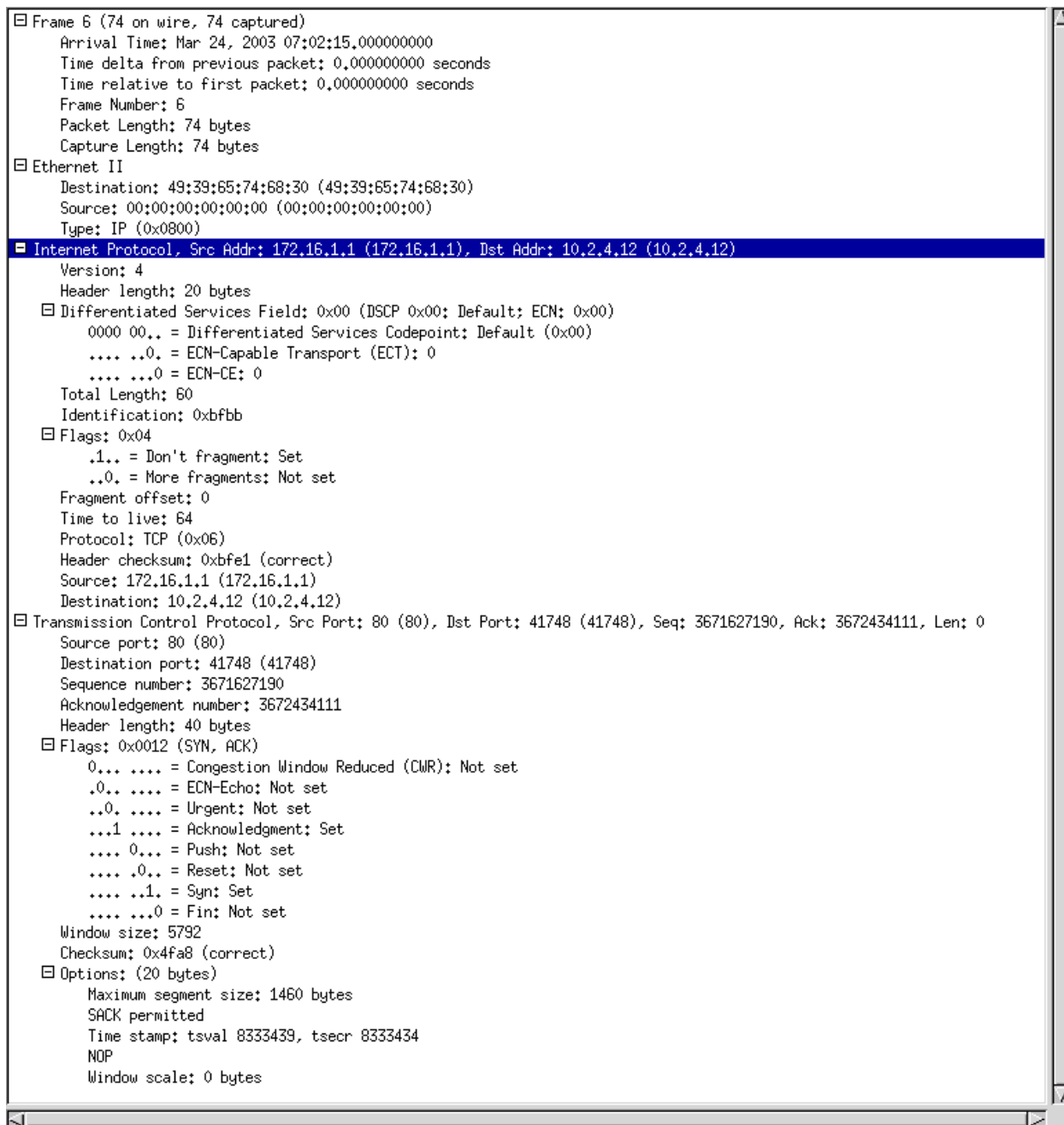


Figure 57: Ethereal – decode pane

The bottom pane displays the raw packets' data. This raw data pane highlights parts according to the selection in the decode pane:

```
0000 49 39 65 74 68 30 00 00 00 00 08 00 45 00  I9eth0.. .....E.
0010 00 3c bf bb 40 00 40 06 bf e1 ac 10 01 01 0a 02  <@>@.@. @á.....
0020 04 0c 00 50 a3 14 da d8 95 b6 da e4 e5 bf a0 12  ...PE.00 .Uáâç.
0030 16 a0 4f a8 00 00 02 04 05 b4 04 02 08 0a 00 7f  .0".....
0040 28 7f 00 7f 28 7a 01 03 03 00                (...{z... ..
```

Figure 58: Ethereal – raw data pane

As you can see, Ethereal displays four “lines” per packet (preIn, postIn, preOut and postOut). Please not that depending on the -m and/or -p switches there might be more or less lines per packet. The information about the direction and the interface is not visible at first. This information is “hidden” in the MAC addresses:

The screenshot shows the Ethereal interface with the following details:

- Packet List:** A table with columns No., Time, Source, Destination, otoc, and Info. Packet 6 is highlighted in blue.
- Packet Details (Frame 6):**
  - Ethernet II: Destination: 49:39:65:74:68:30 (49:39:65:74:68:30), Source: 00:00:00:00:00:00 (00:00:00:00:00:00), Type: IP (0x0800)
  - Internet Protocol: Src Addr: 172.16.1.1 (172.16.1.1), Dst Addr: 10.2.4.12 (10.2.4.12)
  - Transmission Control Protocol: Src Port: 80 (80), Dst Port: 41748 (41748), Seq: 3671627190, Ack: 3672434111, Len: 0
- Raw Packet Data:** Hex and ASCII representation of the packet. Red circles highlight the destination MAC address (49 39 65 74 68 30) and the interface name (I9eth0).
- Filter:** Destination Hardware Address (eth.dst), 6 bytes

Figure 59: Ethereal – direction and interface as MAC address

## Ethereal fw monitor additions

Alfred Köbler ([Alfred.Koebler@icon.de](mailto:Alfred.Koebler@icon.de)) wrote an addition to Ethereal which enables Ethereal to display not MAC addresses but the fw monitor information. This addition is part of the standard Ethereal distribution since version 0.9.9. It can be activated using **Edit/Preferences/Protocols/Ethernet/Interpret as FireWall-1 monitor file**:

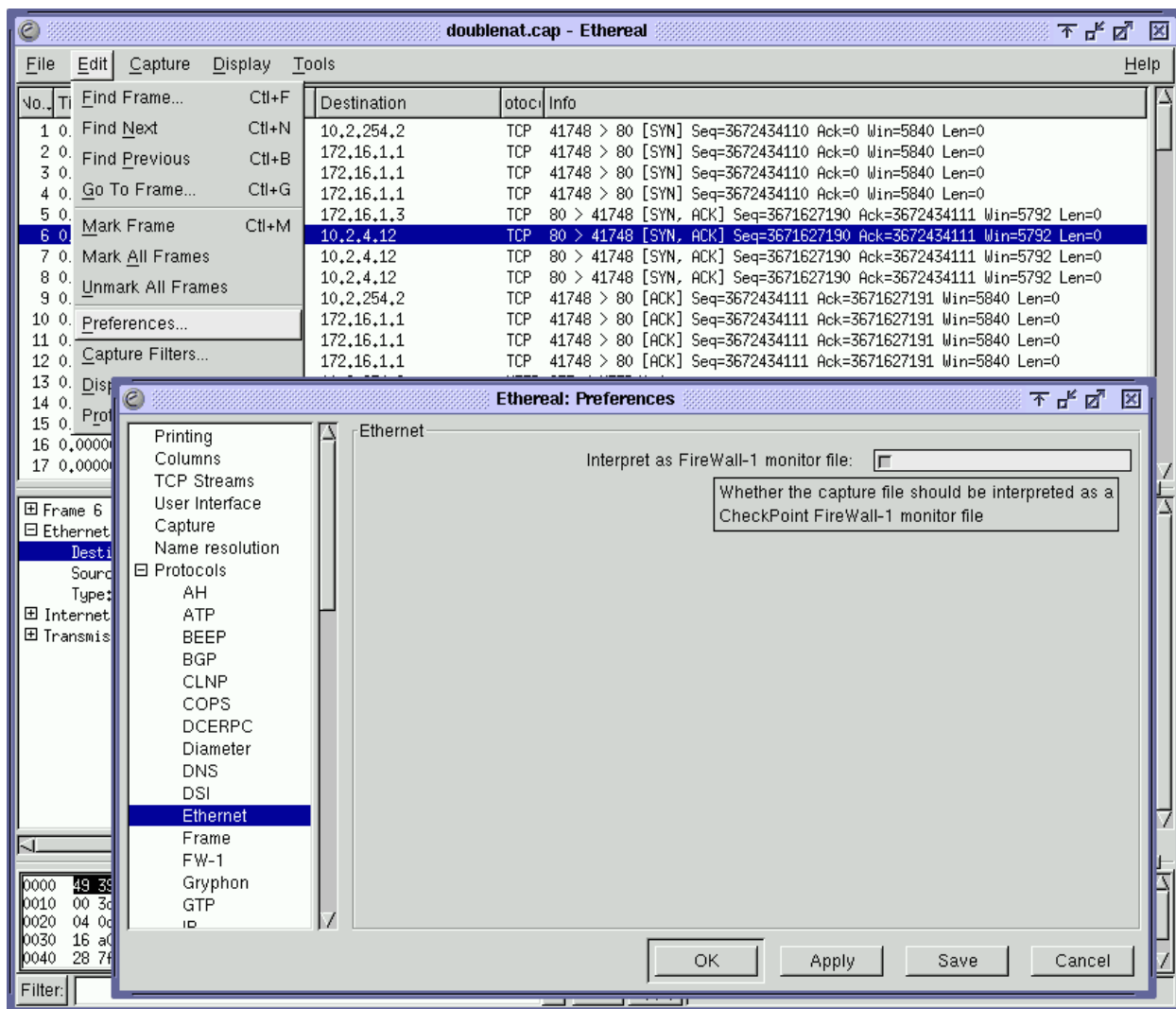


Figure 60: Ethereal – activate fw monitor decoding

If the fw monitor decoding is activated, Ethereal will display the decoded fw monitor information in the MAC addresses instead of the MAC addresses itself. It will show the direction ( i - preIn, I - postIn, o - preOut or O - postOut) and the interface:

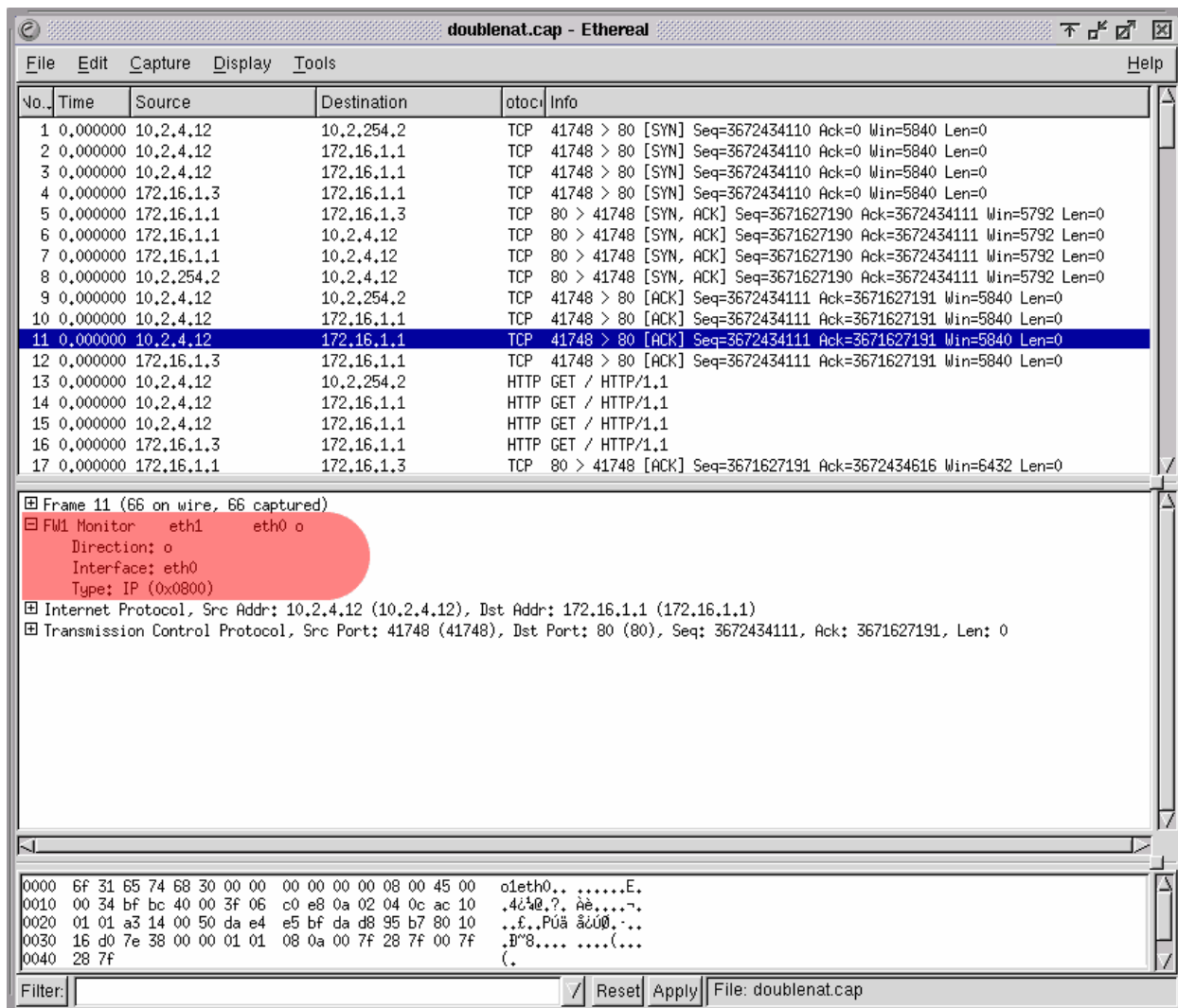


Figure 61: Ethereal – fw monitor decoding

The summary line (which can also be displayed as an additional column in the overview pane) lists all encountered interfaces and the packet's direction. For a packet entering the gateway through eth0 and leaving the gateway through eth1 the summary line will show:

Interface	Direction	Summary line
eth0	i - preIn	i eth1 eth0
eth0	I - postIn	eth1 I eth0
eth1	o - preOut	eth1 eth0 o
eth1	O - postOut	eth1 O eth0

## Activate the FW-1 chain column

The interface and direction information described above can also be displayed as an additional column in the overview pane. To activate the chain column go to **Edit/Preferences/Protocols/Columns** and add a new column like showed below:

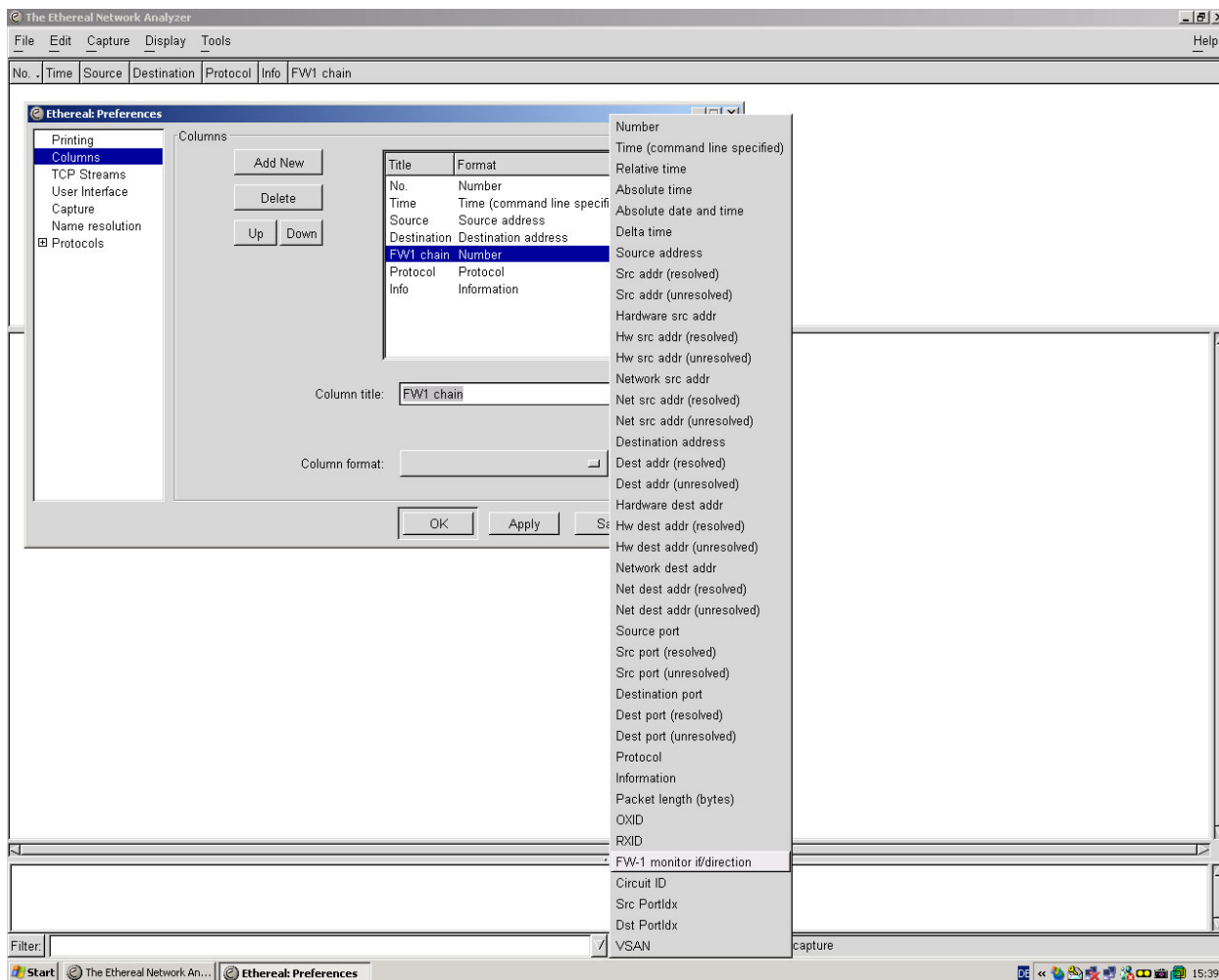


Figure 62: Ethereal – activate FW-1 direction/interface column

This will give you an additional column which displays the interface and direction information:

No.	Time	Source	Destination	FW1 chain	Protocol	Info
1	0.000000	10.2.4.12	10.2.254.2	i eth1	TCP	41748 > http [SYN] Seq=3672434110 Ack=...
2	0.000000	10.2.4.12	172.16.1.1	eth1 I	TCP	41748 > http [SYN] Seq=3672434110 Ack=...
3	0.000000	10.2.4.12	172.16.1.1	eth1 o eth0	TCP	41748 > http [SYN] Seq=3672434110 Ack=...
4	0.000000	172.16.1.3	172.16.1.1	eth1 i eth0	TCP	41748 > http [SYN] Seq=3672434110 Ack=...
5	0.000000	172.16.1.1	172.16.1.3	eth1 o eth0	TCP	http > 41748 [SYN, ACK] Seq=367162719...
6	0.000000	172.16.1.1	10.2.4.12	eth1 i eth0	TCP	http > 41748 [SYN, ACK] Seq=367162719...
7	0.000000	172.16.1.1	10.2.4.12	eth1 o eth0	TCP	http > 41748 [SYN, ACK] Seq=367162719...
8	0.000000	10.2.254.2	10.2.4.12	o eth1 eth0	TCP	http > 41748 [SYN, ACK] Seq=367162719...
9	0.000000	10.2.4.12	10.2.254.2	i eth1 eth0	TCP	41748 > http [ACK] Seq=3672434111 Ack=...
10	0.000000	10.2.4.12	172.16.1.1	eth1 I	TCP	41748 > http [ACK] Seq=3672434111 Ack=...
11	0.000000	10.2.4.12	172.16.1.1	eth1 o eth0	TCP	41748 > http [ACK] Seq=3672434111 Ack=...
12	0.000000	172.16.1.3	172.16.1.1	eth1 i eth0	TCP	41748 > http [ACK] Seq=3672434111 Ack=...
13	0.000000	10.2.4.12	10.2.254.2	i eth1 eth0	HTTP	GET / HTTP/1.1
14	0.000000	10.2.4.12	172.16.1.1	eth1 I	HTTP	GET / HTTP/1.1
15	0.000000	10.2.4.12	172.16.1.1	eth1 o eth0	HTTP	GET / HTTP/1.1
16	0.000000	172.16.1.3	172.16.1.1	eth1 i eth0	HTTP	GET / HTTP/1.1

Frame 1 (74 bytes on wire, 74 bytes captured)  
 FW1 Monitor i eth1 eth0  
 Direction: i  
 Interface: eth1  
 Type: IP (0x0800)  
 Internet Protocol, Src Addr: 10.2.4.12 (10.2.4.12), Dst Addr: 10.2.254.2 (10.2.254.2)  
 Transmission Control Protocol, Src Port: 41748 (41748), Dst Port: http (80), Seq: 3672434110, Ack: 0, Len: 0

```

0000  69 31 65 74 68 31 00 00 00 00 00 08 00 45 00  f1eth1...E.
0010  00 3c bf ba 40 00 40 06 64 ef 0a 02 04 0c 0a 02  .<..@.@. d.....
0020  fe 02 a3 14 00 50 da e4 e5 be 00 00 00 a0 02  ....P.....
0030  16 d0 8e 23 00 00 02 04 05 b4 04 02 08 0a 00 7f  ..#.....
0040  28 7a 00 00 00 00 01 03 03 00  (z.....
  
```

Figure 63: Ethereal – FW-1 direction/interface column

## Using display and color filters on fw monitor parameters

Ethereal offers the possibility to display only specific packets and/or to display them with different colors. The easiest way to display only specific packets is to select a packet in the overview pane and select **Follow TCP Stream** from the context menu. This will automatically set a display filter to only display packets of this specific connection (based on source/destination IP addresses and ports). You can see this filter below the raw data pane. Additionally it displays the data exchanged between client and server in a separate dialog box:

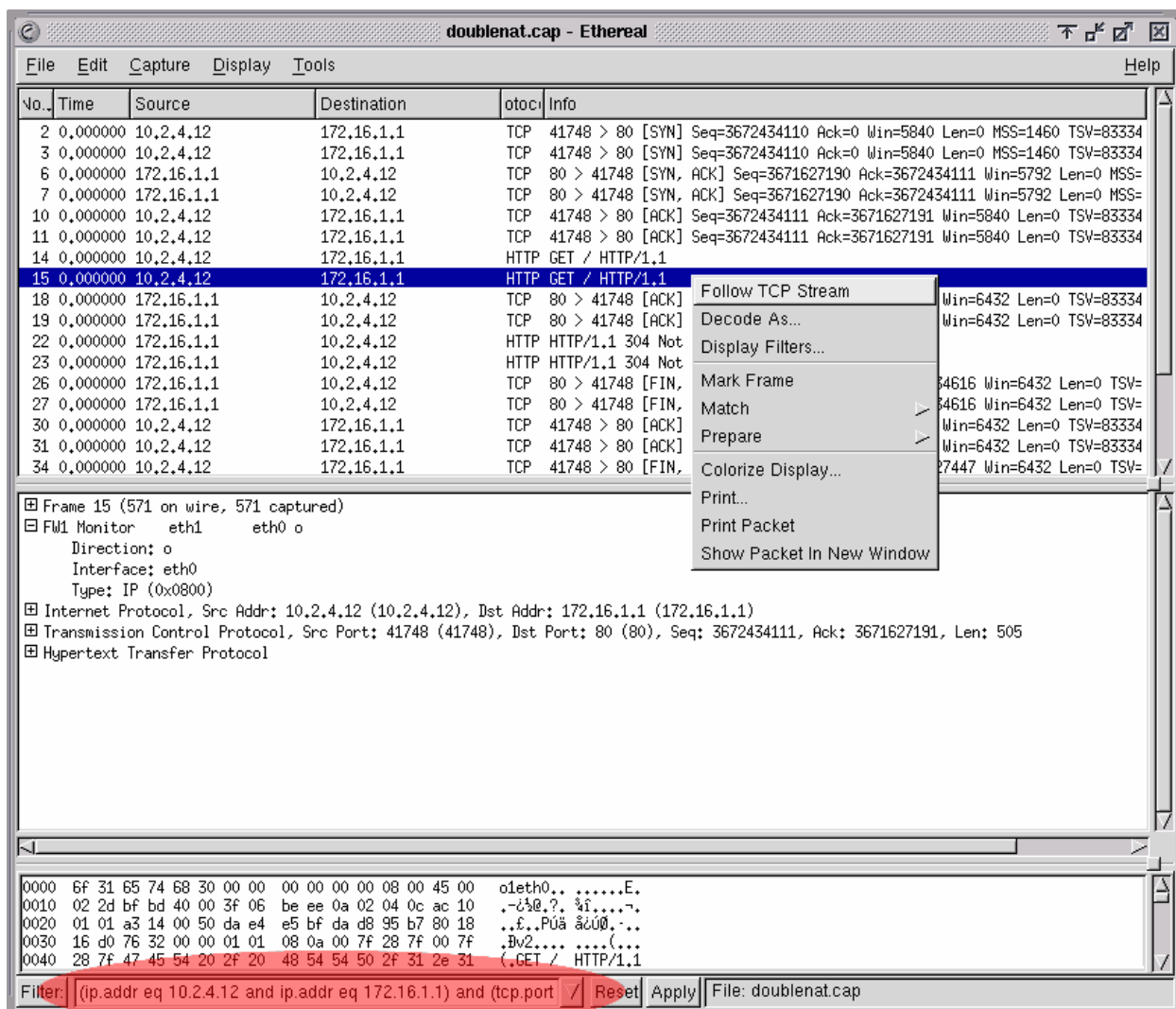


Figure 64: Ethereal – Follow TCP Stream

The display filter in this case is:

```
(ip.addr eq 10.2.4.12 and ip.addr eq 172.16.1.1) and (tcp.port eq 41748 and tcp.port eq 80)
```

Figure 65: Ethereal – TCP Stream display filter example

**!** Please note that this filter only uses IP addresses and ports. Therefore you will still have all four lines per packet in the overview pane. An exception might be if you are using NAT (where the addresses might change inbound and/or outbound) or if you used capture masks ([Capture masks](#)) while creating the capture file.

Another possibility is to select a value in the decode pane and select **Match** or **Prepare** together with an logical operator. This is especially useful to discover how the property is called and which data types it accepts:

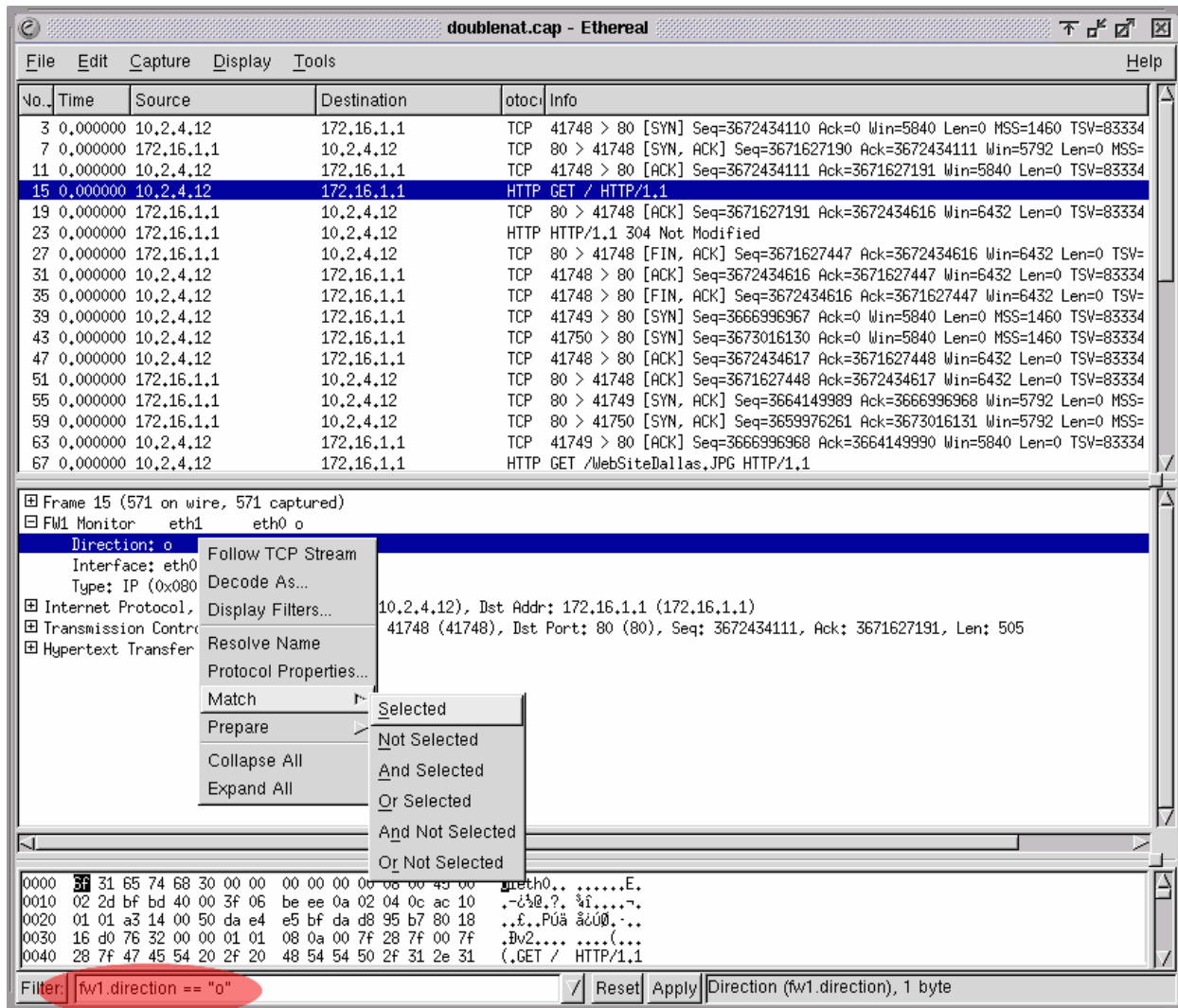


Figure 66: Ethereal – Match selected property

The filter above would only list packets in the overview pane which where captured postOut (outbound interface, after the VM).

Other useful expressions are:

Field	Property	Value
IP address (source or destination)	<code>ip.addr</code>	IP address
Source IP address	<code>ip.src</code>	IP address
Destination IP address	<code>ip.dst</code>	IP Address
TCP port (source or destination)	<code>tcp.port</code>	Port number (0-65535)
TCP source port	<code>tcp.srcport</code>	Port number (0-65535)
TCP destination port	<code>tcp.dstport</code>	Port number (0-65535)
UDP port (source or destination)	<code>udp.port</code>	Port number (0-65535)
UDP source port	<code>udp.srcport</code>	Port number (0-65535)
UDP destination port	<code>udp.dstport</code>	Port number (0-65535)
fw monitor direction	<code>fwl.direction</code>	"i", "I", "o" or "O"
fw monitor interface	<code>fwl.interface</code>	An Interface name (e.g. "eth0")

*Figure 67: Ethereal – Useful filter properties*

**!** Ethereal filters require no special syntax to check whether an IP address belongs to a specific subnet. Instead you can use an IP address with Classless Inter Domain Routing (CIDR) notation (e.g. 192.168.10.26/24) anywhere instead of a normal IP address. To check whether a packet is sent from or sent to a specific network (192.168.10.26/24) you can use the following filter:  
`ip.addr eq 192.168.10.26/24`

You can find a list with all known properties under **Help/Help/Display Filters**.

In addition Ethereal offers the possibility to colorize packets according to filters. The syntax used there is the same like the syntax for the display filters. You can add color filters using **Display/Colorize Display....** A simple color filter is to colorize packets according to their interface direction:

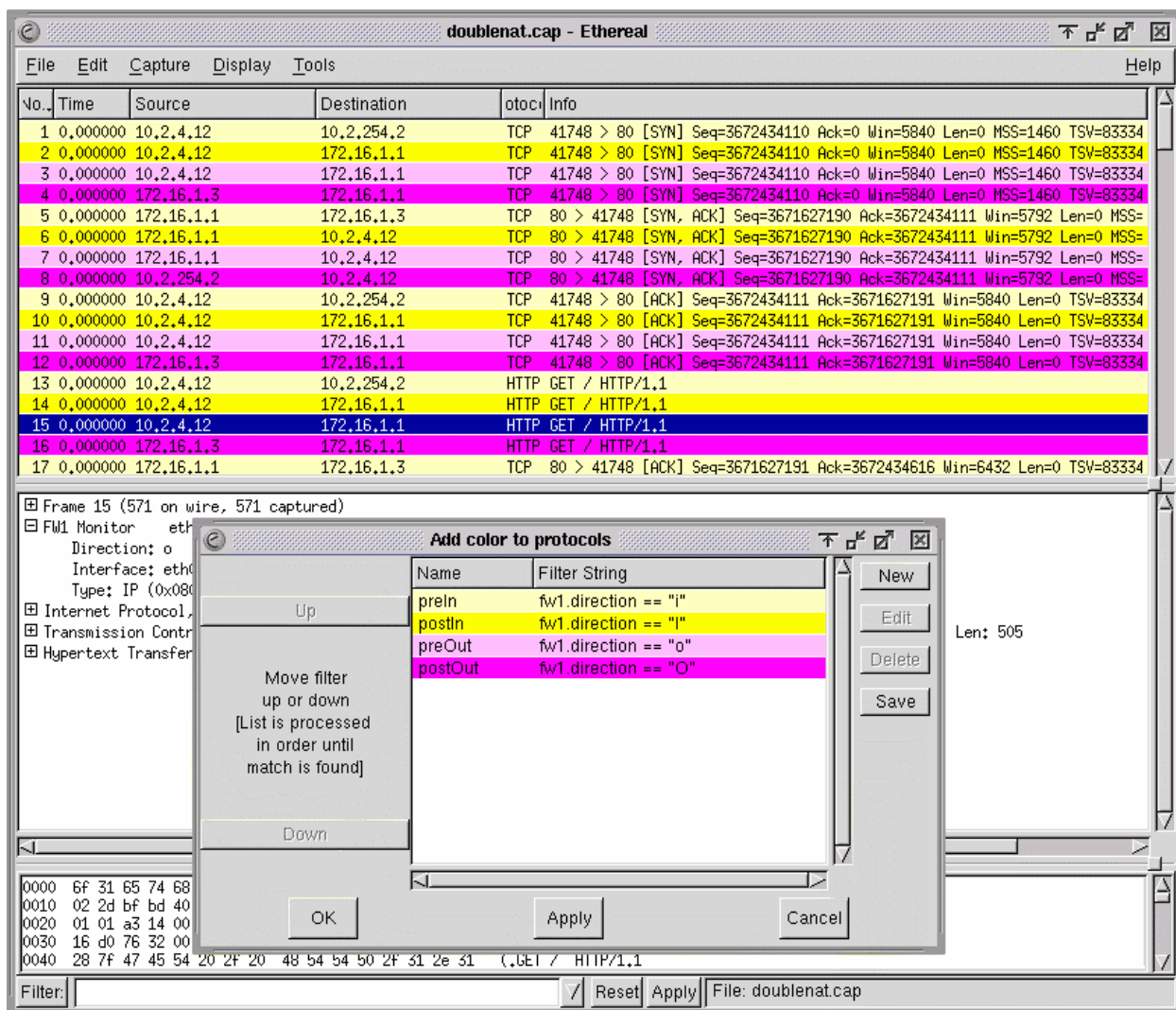


Figure 68: Ethereal – Color Filters

## Using CPETHEREAL to inspect fw monitor files

Based on the standard Ethereal Pedro Paixão and Shaul Eizikovich created an enhanced version of Ethereal. This “Check Point flavor of Ethereal” (reference as CPETHEREAL on the following pages) extends the standard Ethereal in many areas to cover Check Point (an fw monitor) specific needs and functions. CPETHEREAL is available in two versions. A public version with slightly improved fw monitor decoding ([public CPETHEREAL](#)) and an enhanced CSP version with all the features covered below ([CSP Etheral](#)).

### Block coloring

Because fw monitor may capture multiple samples of the same packet passing through the firewall it is sometimes hard to differentiate between the different packets. CPETHEREAL can group samples of the same packets by coloring them. This can be activated using **CheckPoint/Colorize**:

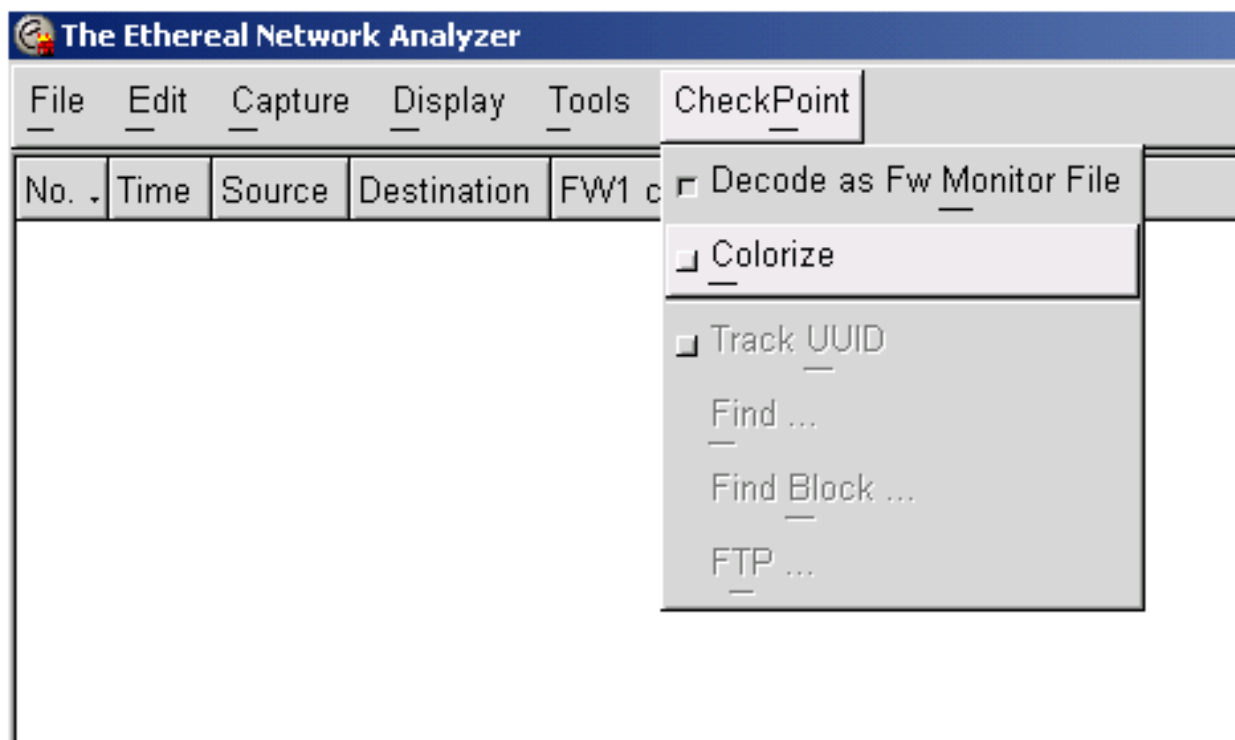


Figure 69: CPETHEREAL – activate Block coloring

Once activated CPEThereal will colorize samples of the same packets in blue and red like the example below:

The screenshot shows the CPEThereal application window titled "fwmonitor.cap - Ethereal". The main pane displays a table of captured packets:

No. .	Time	Source	Destination	FW1 chain	Protocol	Info
1	0.000000	172.16.1.1	172.16.1.2	i eth0	ICMP	Echo (ping) request
2	0.000000	172.16.1.1	172.16.1.2	eth0 I	ICMP	Echo (ping) request
3	0.000000	172.16.1.2	172.16.1.1	eth0 o	ICMP	Echo (ping) reply
4	0.000000	172.16.1.2	172.16.1.1	o eth0	ICMP	Echo (ping) reply
5	0.000000	172.16.1.1	172.16.1.2	i eth0	ICMP	Echo (ping) request
6	0.000000	172.16.1.1	172.16.1.2	eth0 I	ICMP	Echo (ping) request
7	0.000000	172.16.1.2	172.16.1.1	eth0 o	ICMP	Echo (ping) reply
8	0.000000	172.16.1.2	172.16.1.1	o eth0	ICMP	Echo (ping) reply

Below the table, the packet details pane shows:

- Frame 1 (98 bytes on wire, 98 bytes captured)
- FW1 Monitor i eth0
- Internet Protocol, Src Addr: 172.16.1.1 (172.16.1.1), Dst Addr: 172.16.1.2 (172.16.1.2)
- Internet Control Message Protocol

The hex dump at the bottom shows the raw packet data in hexadecimal and ASCII:

```

0000  69 31 65 74 68 30 00 00 00 00 00 08 00 45 00  f1eth0...E.
0010  00 54 ba 0c 40 00 40 01 26 79 ac 10 01 01 ac 10  .T..@.@.&y.....
0020  01 02 08 00 2b e5 c9 0e 01 00 8f 2b 7b 3e 0b 9f  ...+...>...
0030  01 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15  .....!#$%
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25  .....
  
```

The Filter field at the bottom is empty, and the file name is "fwmonitor.cap".

Figure 70: CPEThereal – Active Block coloring

## NAT Highlighting

Following a connection through the firewall can be simplified by using Display Filters (refer to [Using display and color filters on fw monitor parameters](#)). However, once you are using NAT things might get more complicated. To simplify this task CPEThereal recognizes NATted packets and marks them red in the overview pane. Additionally it provides some more information about the NAT type in the decode pane:

The screenshot shows the CPEThereal interface with a packet list and a detailed decode view for a NATted packet.

No.	Time	Source	Destination	FW1 chain	Protocol	Info
79	0.000000	172.16.1.1	10.2.4.12	eth0	eth1 o TCP	http > 41749 [ACK] Seq=3664149990 Ack=3666997439 wi
80	0.000000	10.2.254.2	10.2.4.12	eth0	o eth1 TCP	http > 41749 [ACK] Seq=3664149990 Ack=3666997439 wi
81	0.000000	172.16.1.1	172.16.1.3	i eth0	eth1 HTTP	HTTP/1.1 304 Not Modified
82	0.000000	172.16.1.1	10.2.4.12	eth0 I	eth1 o HTTP	HTTP/1.1 304 Not Modified
83	0.000000	172.16.1.1	10.2.4.12	eth0	eth1 o HTTP	HTTP/1.1 304 Not Modified
84	0.000000	10.2.254.2	10.2.4.12	eth0	o eth1 HTTP	HTTP/1.1 304 Not Modified
85	0.000000	172.16.1.1	172.16.1.3	i eth0	eth1 TCP	http > 41749 [FIN, ACK] Seq=3664150229 Ack=36669974
86	0.000000	172.16.1.1	10.2.4.12	eth0 I	eth1 TCP	http > 41749 [FIN, ACK] Seq=3664150229 Ack=36669974
87	0.000000	172.16.1.1	10.2.4.12	eth0	eth1 o TCP	http > 41749 [FIN, ACK] Seq=3664150229 Ack=36669974
88	0.000000	10.2.254.2	10.2.4.12	eth0	o eth1 TCP	http > 41749 [FIN, ACK] Seq=3664150229 Ack=36669974
89	0.000000	172.16.1.1	172.16.1.3	i eth0	eth1 TCP	http > 41750 [ACK] Seq=3659976262 Ack=3673016604 wi
90	0.000000	172.16.1.1	10.2.4.12	eth0 I	eth1 TCP	http > 41750 [ACK] Seq=3659976262 Ack=3673016604 wi
91	0.000000	172.16.1.1	10.2.4.12	eth0	eth1 o TCP	http > 41750 [ACK] Seq=3659976262 Ack=3673016604 wi
92	0.000000	10.2.254.2	10.2.4.12	eth0	o eth1 TCP	http > 41750 [ACK] Seq=3659976262 Ack=3673016604 wi
93	0.000000	172.16.1.1	172.16.1.3	i eth0	eth1 HTTP	HTTP/1.1 304 Not Modified
94	0.000000	172.16.1.1	10.2.4.12	eth0 I	eth1 HTTP	HTTP/1.1 304 Not Modified

Frame 86 (66 bytes on wire, 66 bytes captured)

- FW1 Monitor eth0 I eth1 FW-1 NAT, Original: 172.16.1.3 (41749), Translated: 10.2.4.12 (41749), NAT: STATIC\_DST
  - Direction: I9
  - UUID: 0x00000000
  - Interface: eth0
  - NAT: STATIC\_DST
  - Type: IP (0x0800)
- Internet Protocol, Src Addr: 172.16.1.1 (172.16.1.1), Dst Addr: 10.2.4.12 (10.2.4.12)
- Transmission Control Protocol, Src Port: http (80), Dst Port: 41749 (41749), Seq: 3664150229, Ack: 3666997439, Len: 0

```

0000 49 39 65 74 68 30 00 00 00 00 00 08 00 45 00  I9eth0... ..E.
0010 00 34 bf cf 40 00 40 06 bf d5 ac 10 01 01 0a 02  .4..@.@. ....
0020 04 0c 00 50 a3 15 da 66 7e d5 da 91 f0 bf 80 11  ...P...f ~.....
0030 19 20 88 6b 00 00 01 01 08 0a 00 7f 28 90 00 7f  . .k.... ..()(.
0040 28 90
  
```

Figure 71: CPEThereal – NAT Highlighting

## Improved FTP decomposing and search mechanism

Many environments have problems with malformed FTP transfers. Although not directly Check Point related, CPEThereal provides enhanced FTP features.

First of all CPEThereal provides a more detailed FTP control connection decomposing than the standard Ethereal. This includes things like an explicit test for an ending <CR><LF> and a decoding and counting of replied lines (banners in most cases):

No. .	Time	Source	Destination	FW1 chain	Protocol	Info
32	0.000000	172.16.1.2	172.16.1.1	secx1 : o eth0 eth1	TCP	auth > 33337 [RST, ACK]
33	0.000000	172.16.1.2	172.16.1.1	ipopt_res : o eth0 eth1	TCP	auth > 33337 [RST, ACK]
34	0.000000	172.16.1.2	172.16.1.1	(Redundant) : o eth0 eth1	TCP	auth > 33337 [RST, ACK]
35	0.000000	172.16.1.1	172.16.1.2	ipopt_strip : i eth0 eth1	FTP	Response: 220 ready, dude
36	0.000000	172.16.1.1	172.16.1.2	asm : i eth0 eth1	FTP	Response: 220 ready, dude
37	0.000000	172.16.1.1	172.16.1.2	secx1_sync : i eth0 eth1	FTP	Response: 220 ready, dude
38	0.000000	172.16.1.1	172.16.1.2	fw : i eth0 eth1	FTP	Response: 220 ready, dude
39	0.000000	172.16.1.1	172.16.1.2	secx1 : eth0 I eth1	FTP	Response: 220 ready, dude
40	0.000000	172.16.1.1	172.16.1.2	scv : eth0 I eth1	FTP	Response: 220 ready, dude
41	0.000000	172.16.1.1	172.16.1.2	ipopt_res : eth0 I eth1	FTP	Response: 220 ready, dude
42	0.000000	172.16.1.1	172.16.1.2	(Redundant) : eth0 I eth1	FTP	Response: 220 ready, dude
43	0.000000	172.16.1.2	172.16.1.1	ipopt_strip : eth0 o eth1	TCP	32840 > ftp [ACK] Seq=1329294903
44	0.000000	172.16.1.2	172.16.1.1	asm : eth0 o eth1	TCP	32840 > ftp [ACK] Seq=1329294903
45	0.000000	172.16.1.2	172.16.1.1	fw : eth0 o eth1	TCP	32840 > ftp [ACK] Seq=1329294903
46	0.000000	172.16.1.2	172.16.1.1	secx1 : o eth0 eth1	TCP	32840 > ftp [ACK] Seq=1329294903
47	0.000000	172.16.1.2	172.16.1.1	ipopt_res : o eth0 eth1	TCP	32840 > ftp [ACK] Seq=1329294903

```

Frame 35 (105 bytes on wire, 105 bytes captured)
  Ethernet II, Src: eth0, Dst: eth1
    Direction: i
      UUID: 0x0000bf66
      Interface: eth0
      Chain Position: ipopt_strip
      Type: IP (0x0800)
    Internet Protocol, Src Addr: 172.16.1.1 (172.16.1.1), Dst Addr: 172.16.1.2 (172.16.1.2)
    Transmission Control Protocol, Src Port: ftp (21), Dst Port: 32840 (32840), Seq: 1329294903, Ack: 1356449990, Len: 51
    File Transfer Protocol (FTP)
      Has <CRLF>: TRUE
      Response code: 220
      Response arg: ready, dude (vsFTPD 1.1.0: beat me, break me)
      Number of Lines: 1
      Complete Response: True
  
```

```

0000  69 0c 65 74 68 30 00 00 00 00 bf 66 08 00 45 00  i.eth0...f..E.
0010  00 5b eb 08 40 00 40 06 f5 70 ac 10 01 01 ac 10  .[.0.0..p.....
0020  01 02 00 15 80 48 4f 3b 6e 37 50 d9 c8 c6 50 18  ....HO; n7P...P.
0030  16 d0 ef 6f 00 00 32 32 30 20 72 65 61 64 79 2c  ...o..22 0 ready,
0040  20 64 75 64 65 20 28 76 73 46 54 50 64 20 31 2e  dude (v sFTPD 1.
  
```

Figure 72: CPEThereal – FTP decomposing

Because some problems (missing <CR><LF> at the end, too long banner) are not uncommon CPetereal also provides a function for searching such problematic packets using **CheckPoint/FTP**:

The screenshot displays the CPetereal interface with a packet capture window titled 'uid\_all.cap - Ethereal'. The main pane shows a list of captured packets. Packet 38 is selected, and a 'Find FTP Fra...' dialog box is open over it. The dialog has three options: 'Banner size exceeds: 1', 'Missing <CRLF>' (checked), and 'Incomplete Response'. The packet details pane shows the following information:

- Frame 38 (105 bytes on wire, 105 bytes captured)
- FW1 Monitor 1 eth0 eth1
- Direction: i
- UUID: 0x0000bf66
- Interface: eth0
- Chain Position: fw
- Type: IP (0x0800)
- Internet Protocol, Src Addr: 172.16.1.1 (172.16.1.1), Dst Addr: 172.16.1.2 (172.16.1.2)
- Transmission Control Protocol, Src Port: ftp (21), Dst Port: 32840 (32840), Seq: 1329294903, Ack: 1356449990, Len: 51
- File Transfer Protocol (FTP)
  - Has <CRLF>: True
  - Response code: 220
  - Response arg: ready, dude (vsFTPd 1.1.0: beat me, break me)
  - Number of Lines: 1
  - Complete Response: True

The hex dump at the bottom shows the raw packet data:

```

0000  69 01 65 74 68 30 00 00  00 00 bf 66 08 00 45 00  i.eth0...f..E.
0010  00 5b eb 08 40 00 40 06  f5 70 ac 10 01 01 ac 10  .[.0.0..p.....
0020  01 02 00 15 80 48 4f 3b  6e 37 50 d9 c8 c6 50 18  ....HO;n7P...P.
0030  16 d0 ef 6f 00 00 32 32  30 20 72 65 61 64 79 2c  ...o..22 0 ready,
0040  20 64 75 64 65 20 28 76  73 46 54 50 64 20 31 2e  dude (vsFTPd 1.
    
```

Figure 73: CPetereal – FTP search

## Check Point enhanced search

Using CheckPoint/Find... it is possible to search packets according to their Check Point specific properties:

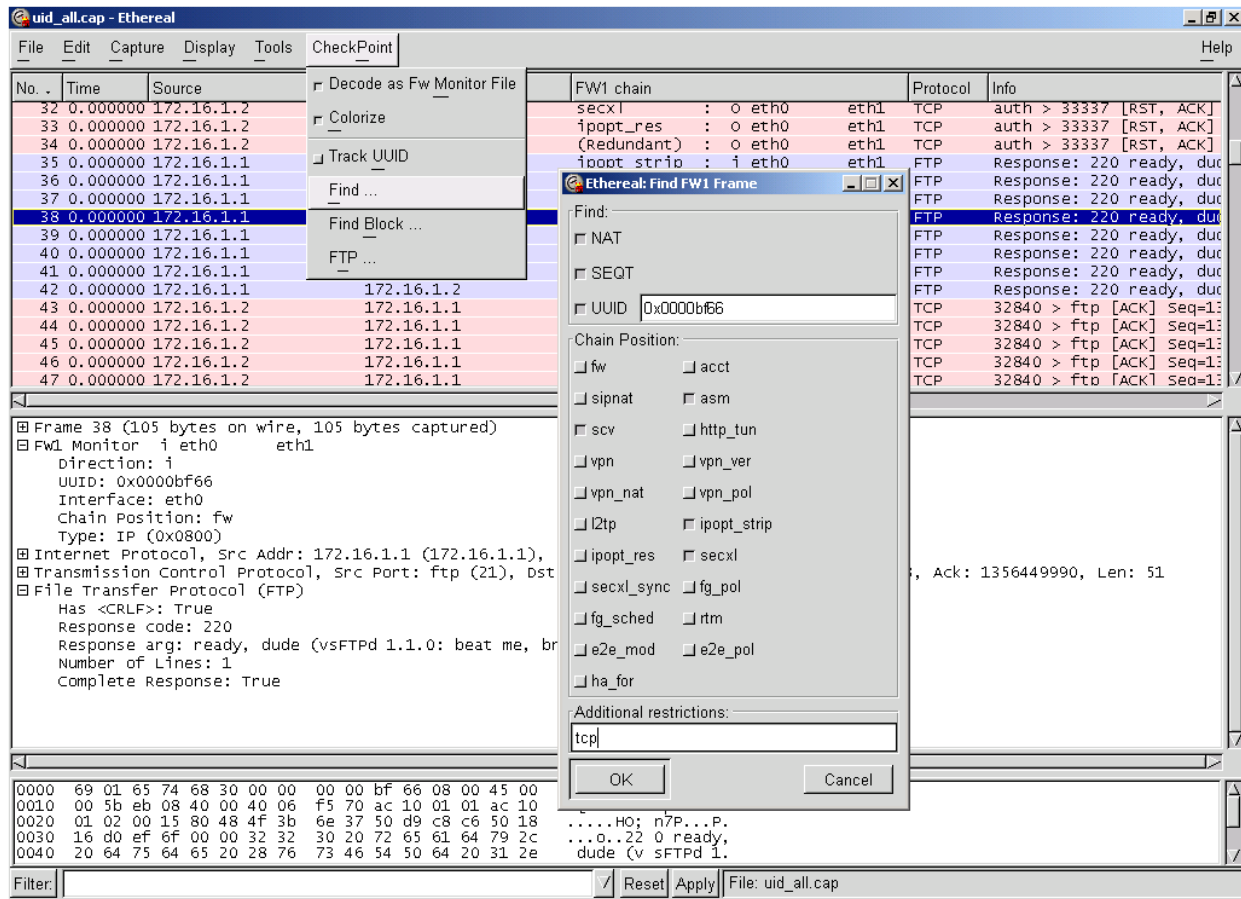


Figure 74: CPEThereal – Check Point enhanced search

The Check Point enhanced search dialog consists of three search areas.

The top area allows you to find packets based on connection properties:

- **NAT**: Find packets which where NATed
- **SEQT** Find packets where the sequence number or the acknowledge number was changes
- **UUID**: Find packets belonging to specific connection based on their UUID

The pane in the middle allow you to filter the packets based on their capture position in the chain.

In addition it's possible to specify additional restrictions using Ethereal filters (refer to [Using display and color filters on fw monitor parameters](#) for an overview about Ethereal filter syntax) in the bottom pane.

**!** Please note that the chain positions in the enhanced search do only make sense for capture files captured with NG with Application Intelligence (FP4) or higher. This feature requires absolute chain positions ([Use absolute chain positions \[-a\]](#)) which are only available since NG with Application Intelligence.

## Block Filters

Block filters allow you to find packet blocks (see [Block coloring](#) for further details) based on specific packet chain positions based or absent in these blocks. It's also possible to additionally specify an Ethereal filter (refer to [Using display and color filters on fw monitor parameters](#) for an overview about Ethereal filter syntax):

The screenshot displays the CP Ethernal interface with a packet capture window and a 'Find Block' dialog box. The packet list shows various FTP and TCP packets. The 'Find Block' dialog box is open, showing a table of chain positions (i0/0 to i11/11) with 'Inbound' and 'Outbound' checkboxes. The 'Outbound' checkbox for i4/04 and i9/09 is checked. The dialog also has an 'Additional restrictions' field containing 'udp'.

Chain Position	Inbound	Outbound
i0/0	Exists	Do not Test
i1/1	Do not Test	Do not Test
i2/2	Do not Test	Do not Test
i3/3	Do not Test	Do not Test
i4/4	Do not Test	Exists
i5/5	Exists	Do not Test
i6/6	Do not Test	Do not Test
i7/7	Do not Test	Do not Test
i8/8	Do not Test	Do not Test
i9/9	Do not Test	Exists
i10/10	Do not Test	Do not Test
i11/11	Do not Test	Do not Test

Additional restrictions:  
udp

Figure 75: CP Ethernal – Block Filter

### Tracking UUIDs and chain positions

Since FP3 fw monitor is able to write the connection UUID to the capture file ([Using UUIDs and SSIDs](#)). First of all CPethereal is able to display the UUID in the decode pane. Additionally it's possible to follow a connection based on the UUID. Select a packet of a connection you're interested in and choose **CheckPoint/Track UUID**. This will show you only packets with the same UUID like the UUID of the selected packet:

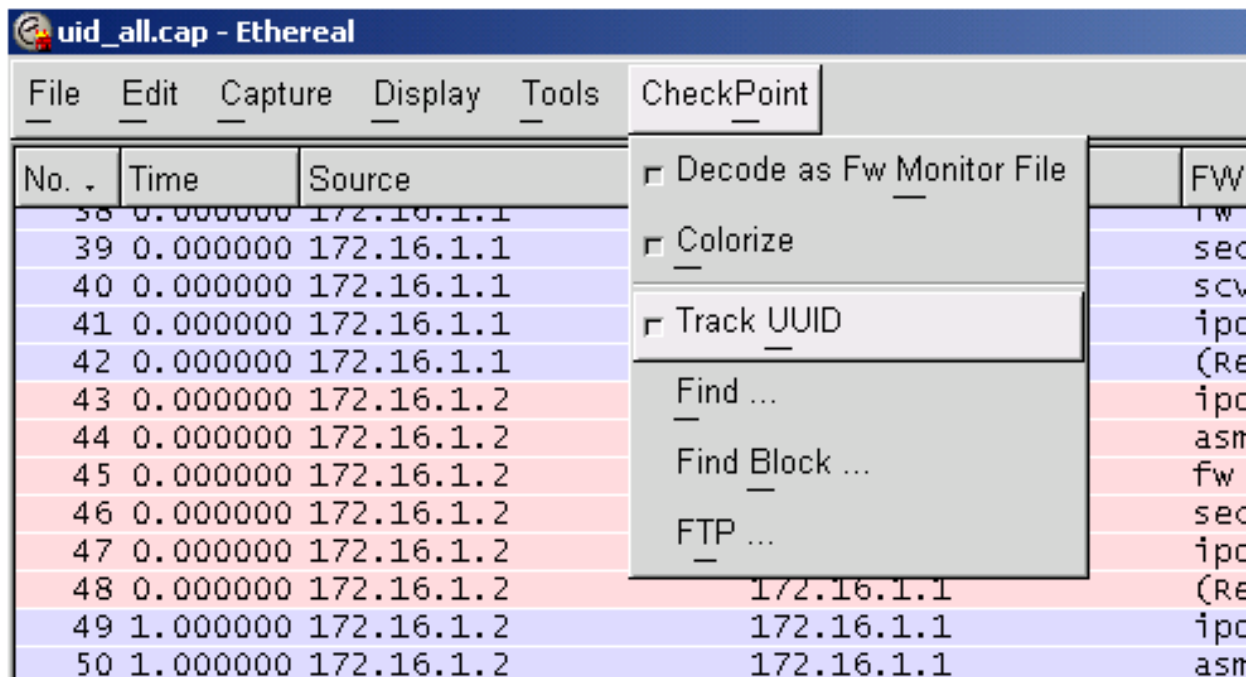


Figure 76: CPethereal – Track UUID

A new feature in NG with Application Intelligence (FP4) is `fw monitor`'s ability to write absolute chain IDs ([Use absolute chain positions \[-a\]](#)) to the capture files rather than relative chain IDs which do only make sense with the corresponding `fw ctl chain` output. CPEThereal knows the absolute chain IDs used by `fw monitor` and is therefore able to display the mnemonic for the chain position as additional information in the FW-1 chain column and in the decode pane:

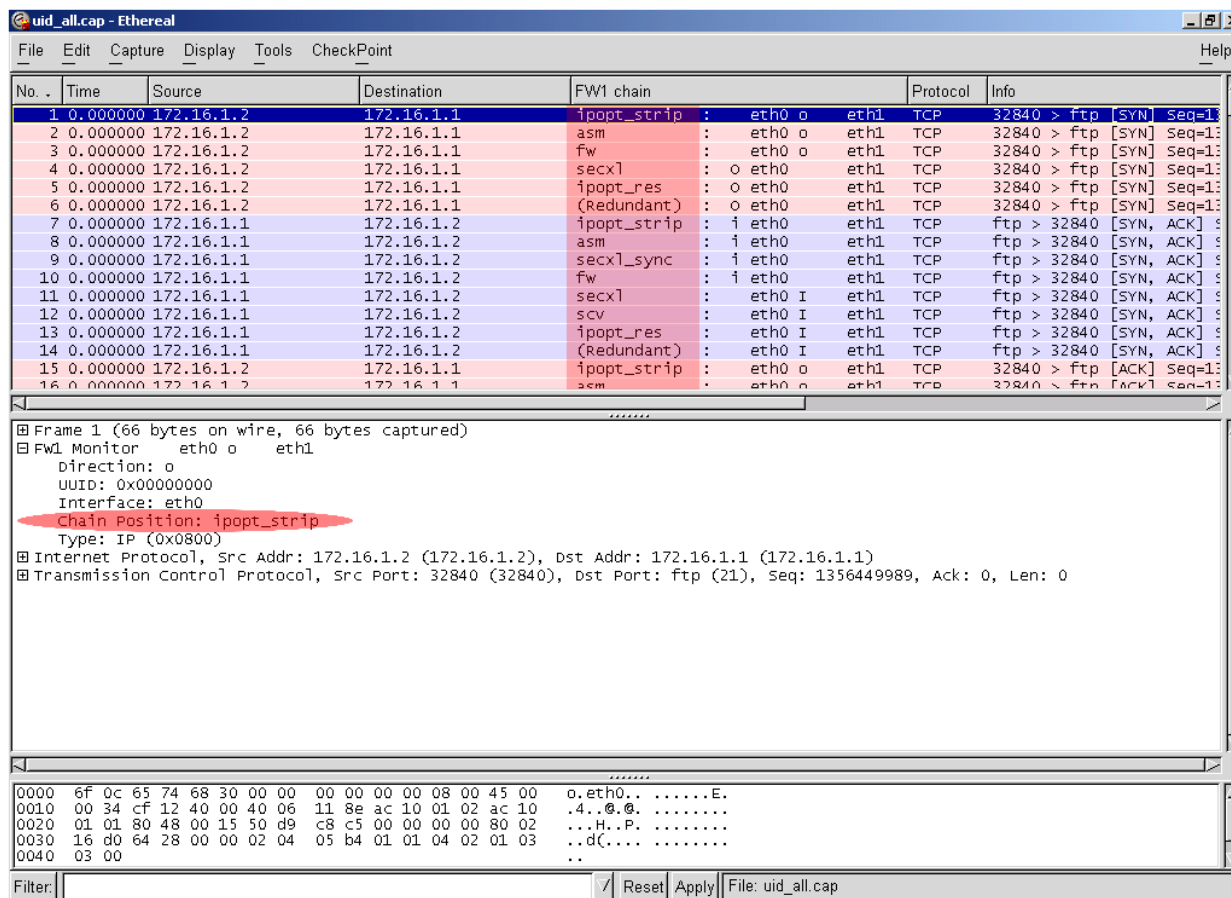


Figure 77: CPEThereal – display absolute FW-1 chain positions

#### Additional fw monitor header properties

CPEThereal includes an improved `fw monitor` decoding. This includes the possibility to use [display or color filters](#) on additional packet properties:

Field	Property	Value
fw monitor direction	fwl.direction	"i", "I", "o" or "O"
fw monitor interface	fwl.interface	An Interface name (e.g. "eth0")
fw monitor connection uuid/suid	fwl.uuid	32bit integer
fw monitor chain module	fwl.chain	Chain module <a href="#">alias</a> name
fw monitor NAT mode	fwl.nat	"HIDE", "STATIC_SRC" or "STATIC_DST"

Figure 78: CPEThereal – Useful filter properties

## srfw – fw monitor on the client side

SecuRemote/SecureClient since Feature Pack 3 includes an utility named “srfw” which provides some functionality of the fw command on the client side. One functionality is to capture packets on the client side with srfw monitor like it is possible on the gateway side with fw monitor. The binary (srfw.exe) is located under %SRDIR%\bin (normally C:\Program Files\CheckPoint\SecuRemote\bin). The general syntax is:

```
srfw monitor [-d] <{-e expr}>+ | -f <filterfile | ->> [-l len] [-m mask]
[-x offset[,length]] [-o file]
```

Figure 79: srfw monitor syntax

The usage of srfw monitor (e.g. the [Break Sequence](#)) and the options are the same as the fw monitor options.

```
C:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Dokumente und Einstellungen\Administrator>cd c:\Programme\CheckPoint\SecuRemote\bin

C:\Programme\CheckPoint\SecuRemote\bin>srfw monitor
[ 2756] srfw: getting filter <from command line>
[ 2756] srfw: loading
[ 2756] srfw: monitoring (control-C to stop)
PCnet10:o[60]: 172.16.1.128 -> 172.16.1.1 <ICMP> len=60 id=20408
ICMP: type=8 code=0 echo request id=1024 seq=3072
PCnet10:O[60]: 172.16.1.128 -> 172.16.1.1 <ICMP> len=60 id=20408
ICMP: type=8 code=0 echo request id=1024 seq=3072
PCnet10:i[60]: 172.16.1.1 -> 172.16.1.128 <ICMP> len=60 id=14866
ICMP: type=0 code=0 echo reply id=1024 seq=3072
PCnet10:I[60]: 172.16.1.1 -> 172.16.1.128 <ICMP> len=60 id=14866
ICMP: type=0 code=0 echo reply id=1024 seq=3072
[ 1664] srfw: caught sig 2
[ 2756] srfw: unloading

C:\Programme\CheckPoint\SecuRemote\bin>
```

Figure 80: srfw monitor example – four ICMP echo requests/replies on a german Windows XP

**!** Please note that although srfw monitor understands most of the fw monitor command line switches not every switch is implemented. You can use some switches (e.g. -e and -f) with srfw monitor (srfw monitor isn't even complaining about it!), but they simply perform no actual function. But this can change in future versions of SecuRemote/SecureClient.

## fw monitor on FireWall-1 VSX

If you are using FireWall-1 VSX you have multiple virtual routers and firewalls on one physical machine. Each router and each firewall has its own IP stack and also its own kernel module chain. On a VSX module each firewall command has the ability to specify on which VS (virtual System) this command should be executed. Each VS has a name and number. You can find out this number using `fw vsx stat`:

```
#fw vsx stat -v
VSX Status Report
=====
Number of Virtual Systems allowed by license:          100
Customer Virtual Systems active / configured:         9 / 9
Virtual Routers active / configured:                 1 / 1
Management Virtual Systems active / configured:       1 / 1
VSID |VRID | Type & Name           | Main IP           | Policy Name       | SIC Stat
-----+-----+-----+-----+-----+-----
  0 |  0 | M noor                         | 194.29.37.185    | Standard          | Trust
  6 |  6 | R noor-vr1                     | 46.46.2.2        | InitialPolicy     | No Trust
 13 | 13 | S noor_vs_7                   | 46.46.11.11     | Standard          | Trust
 22 | 22 | S noor_vs_6                   | 46.46.10.10     | Standard          | Trust
Type: M - Management VS, R - Virtual Router, S - Virtual System.
Total of 11 Virtual Systems
```

Figure 81: fw vsx stat example

`fw monitor`, when used with `-vs` option, monitors Virtual System traffic. It does not show any traffic passing through Virtual Routers.

```
fw monitor -vs <vsid or vs name>
```

Figure 82: fw monitor on FireWall-1 VSX

**!** `fw monitor` on a Virtual Router will only show packets which are inspected by the Virtual Router (which are packet which are targeted to the Virtual Router's virtual IP stack only).

## Resources

### Secure Knowledge Links

#### What is "fw monitor"?

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=10022.0.1862922.2481845>

#### Syntax examples for using the fw monitor command

<https://support.checkpoint.com/csp/idsearch.jsp?id=sk1062>

#### How to run the "fw monitor" command in FireWall-1 4.0 SP3 and above and FireWall-1 4.1

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=10022.0.1862930.2481845>

#### How to view the 'fw monitor' output file!

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=sk3474>

#### How does NG handle TCP connections

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=sk11022>

#### What license feature is needed to run the command "fw monitor" on a VPN-1/FireWall-1 module?

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=10022.0.2594497.2500363>

#### Can the fw monitor utility run during FireWall-1 Policy installation?

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=sk14444>

#### How to prevent the error "/opt/CPfw1-41/tmp/monitorfilter.pf" when running fw monitor?

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=55.0.12289645.2846374>

#### How to avoid the error: "Failed to Load Security Policy: Bad file number"

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=sk336>

#### Error when running 'fw monitor' command: "unknown interface (255): Interrupted system call"

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=55.0.12289624.2846374>

#### What to do if FTP data suddenly stops working.

<https://support.checkpoint.com/csp/idsearch.jsp?resultStart=1&id=sk10494>

## **Detecting sniffers on your network**

[http://www.securiteam.com/unixfocus/Detecting\\_sniffers\\_on\\_your\\_network.html](http://www.securiteam.com/unixfocus/Detecting_sniffers_on_your_network.html)

## **snoop**

### **snoop vulnerable to a remotely exploitable buffer overflow**

<http://www.securiteam.com/exploits/3B5PQRPQAO.html>

### **The Secrets of Snoop**

<http://www.spitzner.net/snoop.html>

### **snoop man page**

Use man snoop to see the snoop manual page. An online copy is available at

<http://www.uwsg.iu.edu/usail/man/solaris/snoop.1.html>

### **Snoop file format (RFC 1761)**

<http://www.ietf.org/rfc/rfc1761.txt?number=1761>

## **tcpdump**

### **tcpdump/libpcap homepage**

<http://www.tcpdump.org/>

### **tcpdump man page**

Use man tcpdump to see the tcpdump manual page. An online copy is available at

[http://www.tcpdump.org/tcpdump\\_man.html](http://www.tcpdump.org/tcpdump_man.html)

## **Ethereal**

### **Ethereal homepage**

<http://www.ethereal.com/>

### **Ethereal user guide**

<http://www.ethereal.com/docs/user-guide/>

### **editcap**

<http://www.ethereal.com/editcap.1.html>

### **Ethereal fw monitor additions**

<http://www.ethereal.com/lists/ethereal-dev/200206/msg00290.html>

## CPEthereal

### Public Version

<http://www.checkpoint.com/techsupport/csp/downloads.html> - cpethereal

### CSP Version

<http://www.checkpoint.com/techsupport/downloadsng/utilities.html> - CPethereal

## Miscellaneous

### An Essay on Endian Order

<http://www.cs.umass.edu/~verts/cs32/endian.html>

## Reference

### Multicast MAC addresses

Some tools are not able to decode `fw monitor` Layer 2 header information properly. `fw monitor` stores its own information in the header fields designed for MAC addresses (Refer to [fw monitor file format](#)). This can be misinterpreted in some cases as Multicast MAC addresses.

### fw monitor file format

Although `fw monitor` capture files are using the snoop file format the content is slightly different. `fw monitor` does not write down MAC addresses (12 bytes; 6 per MAC address) in the Layer 2 Frame header. Instead `fw monitor` writes down information about the interface and chain position where the packet was captured.

If you do not use the `-u` or `-s` option or an older version of `fw monitor` the fields for the MAC addresses are used as follows:

Byte	0	1	2	3	4	5	6	7	8	9	10	11
<b>snoop file</b>	Source MAC address						Destination MAC address					
<b>fw monitor file</b>	Packet direction (i/l/o/o)	chain position	Interface Name									

If you are using `-u` or `-s` the fields are used as follows:

Byte	0	1	2	3	4	5	6	7	8	9	10	11
<b>snoop file</b>	Source MAC address						Destination MAC address					
<b>fw monitor file</b>	Packet direction (i/l/o/o)	chain position	Interface Name						UUID / SUUID			

## UUID format

As described in [Using UUIDs and SSIDs](#) the firewall assigns a UUID to each connection passing through it. This UUID is a 128 bit value built from four 32 bit value where only the first two are relevant.

1. UUID value	Timestamp
2. UUID value	A counter which is used if the first UUID value is not unique
3. UUID value	The IP address of the local firewall (constant)
3. UUID value	A PID (currently a constant, can be ignored).

*Figure 83: UUID format*

When using the `-o` option together with the `-u` or `-s` option, `fw monitor` does not write the full length 128 bit value to the capture file. Instead `fw monitor` writes down a stripped down 32 bit value. This value is composed of the two least significant bytes of the second UUID value (counter) and the two least significant bytes of the first UUID (timestamp).