# Introduction

Packet capturing and analyzing means to grab a copy of packets off of the wire before they are processed by the OS. The capture can be used in network security tools to analyze raw traffic and detect malicious behaviors, networks scans or attacks, for sniffing, fingerprinting and other purposes.

Common internet protocols (IPs):

* HTTP/HTTPS
* ICMP
* TCP
* UDP
* SSH
* …

# Tools

## Tcpdump

* CLI network packet analyzer
* Open source
* Support Linux (included with several Linux distributions), MacOS, Windows (via WinDUMP).

## Wireshark

* UI network packet analyzer
* Open source
* Support Windows, Linux, MacOS
* The most commonly used network management application
* Compared to Tcpdump, Wireshark provides more advanced features (complex filters, advanced decoder, more protocols supported, etc.)

## Tshark

* A CLI tool from Wireshark
* A more powerful version of Tcpdump
* Open source
* Support Windows, Linux, MacOS

# Tcpdump

## Understand Output Format

Tcpdump is capable of capturing and decoding many different protocols, such as TCP, UDP, ICMP, and many more. We can't cover all of them here, let's explore some:

### TCP

The general format of a TCP packet captured by tcpdump is:

timestamp network-layer-protocol src > dst: Flags [tcpflags], seq data-seqno, ack ackno, win window, urg urgent, options [opts], length len

Details:

*Src* and *dst* are the source and destination IP addresses and ports. *Tcpflags* are some combination of S (SYN), F (FIN), P (PUSH), R (RST), U (URG), W (ECN CWR), E (ECN-Echo) or `.' (ACK), or `none' if no flags are set. *Data-seqno* describes the portion of sequence space covered by the data in this packet (see example below). *Ackno* is sequence number of the next data expected the other direction on this connection. *Window* is the number of bytes of receive buffer space available the other direction on this connection. *Urg* indicates there is `urgent' data in the packet. *Opts* are TCP options (e.g., mss 1024). *Len* is the length of payload data.

*Iptype*, *Src*, *dst*, and *flags* are always present. The other fields depend on the contents of the packet's TCP protocol header and are output only if appropriate.

For example:

08:41:13.729687 IP 192.168.64.28.22 > 192.168.64.1.41916: Flags [P.], seq 196:568, ack 1, win 309, options [nop,nop,TS val 117964079 ecr 816509256], length 372

Details:

1. **08:41:13.729687**: The timestamp of the received packet as per the local clock.
2. **IP**: The network layer protocol – in this case, **IPv4**. For **IPv6** packets, the value is **IP6**.
3. **192.168.64.28.22**: The source IPv4 address and port.
4. **192.168.64.1.41916**: The destination IPv4 address and port.
5. **Flags [P.]**: TCP Flag. Typical values are S (SYN) connection start, F (FIN) connection finish, P (PUSH) data push, R (RST) connection reset, and . (ACK) acknowledgment.

It can also be a combination of these values, such as – in this case, **[P.]** for a **PUSH-ACK** packet.

1. **seq 196:568**: The sequence number of the data contained in the packet. For the first packet captured, this is an absolute number. Subsequent packets use a relative number to make it easier to follow. In this example, the sequence is **seq 196:568,** which means this packet contains bytes 196 to 568 of this flow.
2. **ack 1:** The Ack Number. In this case, it is **1** because this is the side sending data. For the side receiving data, this field represents the next expected byte (data) on this flow. For example, the Ack number for the next packet in this flow would be **568**.
3. **win 309**: The window size which represents the number of bytes available in the receiving buffer.
4. **options [nop,nop,TS val 117964079 ecr 816509256]**: TCP options such as the MSS (Maximum Segment Size) or Window Scale.
5. **length 372**: The packet length which represents the length, in bytes, of the payload data. The length is the difference between the last and first bytes in the sequence number.

### UDP

An example of a typical UDP packet captured by tcpdump looks like this:

21:28:09.469568 IP 172.17.1.96.137 > 172.17.127.255.137: UDP, length 50

Details:

1. **21:28:09.469568**: The timestamp of the received packet as per the local clock.
2. **IP**: The network layer protocol – in this case, **IPv4**. For **IPv6** packets, the value is **IP6**.
3. **172.17.1.96.137**: The source IPv4 address and port.
4. **172.17.127.255.137**: The destination IPv4 address and port.
5. **UDP**: The protocol is UDP
6. **length 50**: The packet length is 50 bytes.

## Commands

Note:

- Tcpdump requires elevated permissions, so its commands must be prefixed with "**sudo**".

- Tcpdump continues to capture packets until it receives an interrupt signal. You can **interrupt capturing by pressing Ctrl + C**.

List of common options:

|  |  |
| --- | --- |
| **Option** | **Description** |
| -D | List all network interfaces in your machine. |
| -i <interface-name> | Choose an interface to capture packages. |
| -c <number> | Limit the number of packets captured to <number> packages. |
| -nn | Disable name resolution. |
| -A | Print the content in ASCII. |
| -X | Print content in hex. |
| -XX | Same as -X, but also shows the ethernet header. |
| -w <filename.pcap> | Save captures to file <filename.pcap>. |
| -r <filename.pcap> | Open captures from file <filename.pcap>. |
| -v | Verbose output. |
| -q | Be less verbose (more quiet) with your output. |
| -vv | Verbose output (more v’s gives more output). |
| -e | Get the ethernet header as well. |
| -S | Print absolute sequence numbers. |
| -tttt | Give maximally human-readable timestamp output.   * Default: 12:51:42.934175 * With this option: 2021-07-11 12:51:40.014567 |
| -s <number> | Snaplen flag which specifies the size of each packet to <number> of bytes (from 0 to 65535 bytes).  By default, tcpdump captures 68 or 96 bytes of data from each packet, depending on the platform.  Common values:  -s 0: sets the maximum packet size to 65535 bytes |

### List all network interfaces (devices) in your machine

**$ sudo tcpdump -D       # '-D' = '--list-interfaces'**

1.eth0 [Up, Running]

2.any (Pseudo-device that captures on all interfaces) [Up, Running]

3.lo [Up, Running, Loopback]

4.nflog (Linux netfilter log (NFLOG) interface)

Only used with IP 127.xxx.xxx.xxx

5.nfqueue (Linux netfilter queue (NFQUEUE) interface)

6.usbmon1 (USB bus number 1)

7.usbmon2 (USB bus number 2)

### Choose interfaces (devices)

To capture package in any interface, use the -ioption with the **any** value:

**$ sudo tcpdump -i any**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

09:56:18.293641 IP rhel75.localdomain.ssh > 192.168.64.1.56322: Flags [P.], seq 3770820720:3770820916, ack 3503648727, win 309, options [nop,nop,TS val 76577898 ecr 510770929], length 196

09:56:18.293794 IP 192.168.64.1.56322 > rhel75.localdomain.ssh: Flags [.], ack 196, win 391, options [nop,nop,TS val 510771017 ecr 76577898], length 0

09:56:18.295058 IP rhel75.59883 > gateway.domain: 2486+ PTR? 1.64.168.192.in-addr.arpa. (43)

09:56:18.310225 IP gateway.domain > rhel75.59883: 2486 NXDomain\* 0/1/0 (102)

09:56:18.323563 IP 192.168.64.1.56322 > rhel75.localdomain.ssh: Flags [.], ack 584, win 411, options [nop,nop,TS val 510771047 ecr 76577928], length 0

---- SKIPPING LONG OUTPUT -----

09:56:18.337177 IP rhel75.localdomain.ssh > 192.168.64.1.56322: Flags [P.], seq 584:1644, ack 1, win 309, options [nop,nop,TS val 76577942 ecr 510771047], length 1060

^C

9003 packets captured

9010 packets received by filter

7 packets dropped by kernel

To capture package in a particular interface, use the -ioption with the value of the interface name.

For example, capture packages in **eth0**:

**$ sudo tcpdump -i eth0**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

09:56:18.293641 IP rhel75.localdomain.ssh > 192.168.64.1.56322: Flags [P.], seq 3770820720:3770820916, ack 3503648727, win 309, options [nop,nop,TS val 76577898 ecr 510770929], length 196

09:56:18.293794 IP 192.168.64.1.56322 > rhel75.localdomain.ssh: Flags [.], ack 196, win 391, options [nop,nop,TS val 510771017 ecr 76577898], length 0

09:56:18.295058 IP rhel75.59883 > gateway.domain: 2486+ PTR? 1.64.168.192.in-addr.arpa. (43)

09:56:18.310225 IP gateway.domain > rhel75.59883: 2486 NXDomain\* 0/1/0 (102)

09:56:18.323563 IP 192.168.64.1.56322 > rhel75.localdomain.ssh: Flags [.], ack 584, win 411, options [nop,nop,TS val 510771047 ecr 76577928], length 0

---- SKIPPING LONG OUTPUT -----

09:56:18.337177 IP rhel75.localdomain.ssh > 192.168.64.1.56322: Flags [P.], seq 584:1644, ack 1, win 309, options [nop,nop,TS val 76577942 ecr 510771047], length 1060

^C

9003 packets captured

9010 packets received by filter

7 packets dropped by kernel

### Limit the number of packets captured

**$ sudo tcpdump -i any -c5**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

11:21:30.242740 IP rhel75.localdomain.ssh > 192.168.64.1.56322: Flags [P.], seq 3772575680:3772575876, ack 3503651743, win 309, options [nop,nop,TS val 81689848 ecr 515883153], length 196

11:21:30.242906 IP 192.168.64.1.56322 > rhel75.localdomain.ssh: Flags [.], ack 196, win 1443, options [nop,nop,TS val 515883235 ecr 81689848], length 0

11:21:30.244442 IP rhel75.43634 > gateway.domain: 57680+ PTR? 1.64.168.192.in-addr.arpa. (43)

11:21:30.244829 IP gateway.domain > rhel75.43634: 57680 NXDomain 0/0/0 (43)

11:21:30.247048 IP rhel75.33696 > gateway.domain: 37429+ PTR? 28.64.168.192.in-addr.arpa. (44)

5 packets captured

12 packets received by filter

0 packets dropped by kernel

### Disable name resolution

By default, tcpdump resolves IP addresses and ports into **names**, as shown in the previous example. But it is often easier to use the IP addresses and port **numbers**.

**$ sudo tcpdump -i any -c5 -nn**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

23:56:24.292206 IP 192.168.64.28.22 > 192.168.64.1.35110: Flags [P.], seq 166198580:166198776, ack 2414541257, win 309, options [nop,nop,TS val 615664 ecr 540031155], length 196

23:56:24.292357 IP 192.168.64.1.35110 > 192.168.64.28.22: Flags [.], ack 196, win 1377, options [nop,nop,TS val 540031229 ecr 615664], length 0

23:56:24.292570 IP 192.168.64.28.22 > 192.168.64.1.35110: Flags [P.], seq 196:568, ack 1, win 309, options [nop,nop,TS val 615664 ecr 540031229], length 372

23:56:24.292655 IP 192.168.64.1.35110 > 192.168.64.28.22: Flags [.], ack 568, win 1400, options [nop,nop,TS val 540031229 ecr 615664], length 0

23:56:24.292752 IP 192.168.64.28.22 > 192.168.64.1.35110: Flags [P.], seq 568:908, ack 1, win 309, options [nop,nop,TS val 615664 ecr 540031229], length 340

5 packets captured

6 packets received by filter

0 packets dropped by kernel

### Choose verbose extent

Less verbose:

**$ sudo tcpdump -i eth0 -c1 -q**

12:53:51.512700 IP 192.168.159.128.56772 > 192.168.159.2.domain: UDP, length 43

More verbose:

**$ sudo tcpdump -i eth0 -c1 -v**

12:53:28.941238 IP (tos 0x0, ttl 64, id 17805, offset 0, flags [DF], proto UDP (17), length 71)

192.168.159.128.54535 > 192.168.159.2.domain: 23559+ PTR? 25.38.17.172.in-addr.arpa. (43)

Much more verbose:

**$ sudo tcpdump -i eth0 -c1 -vv**

12:59:44.713164 IP (tos 0x0, ttl 64, id 41825, offset 0, flags [DF], proto UDP (17), length 71)

192.168.159.128.46027 > 192.168.159.2.domain: [bad udp cksum 0xc018 -> 0xcb1d!] 27817+ PTR? 25.38.17.172.in-addr.arpa. (43)

### Filter packets

Tcpdump can capture too many packets, and many of which are not even related to the issue you're troubleshooting. For example, if you're troubleshooting a connectivity issue with a web server, you're not interested in the SSH traffic, so removing the SSH packets from the output makes it easier to work on the real issue.

Tcpdump allows filtering captured packets using a variety of parameters, such as source and destination IP addresses, ports, protocols, etc.

#### Protocol

For example, capture ICMP packets only by using this command:

**$ sudo tcpdump -i any -c3 icmp**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

In a different terminal, try to ping another machine:

$ ping opensource.com

PING opensource.com (54.204.39.132) 56(84) bytes of data.

64 bytes from ec2-54-204-39-132.compute-1.amazonaws.com (54.204.39.132): icmp\_seq=1 ttl=47 time=39.6 ms

Back in the tcpdump capture, notice that tcpdump captures and displays only the ICMP-related packets. In this case, tcpdump is not displaying name resolution packets that were generated when resolving the name opensource.com:

09:34:20.136766 IP rhel75 > ec2-54-204-39-132.compute-1.amazonaws.com: ICMP echo request, id 20361, seq 1, length 64

09:34:20.176402 IP ec2-54-204-39-132.compute-1.amazonaws.com > rhel75: ICMP echo reply, id 20361, seq 1, length 64

09:34:21.140230 IP rhel75 > ec2-54-204-39-132.compute-1.amazonaws.com: ICMP echo request, id 20361, seq 2, length 64

3 packets captured

3 packets received by filter

0 packets dropped by kernel

#### Host

For example, capture only packets to/from host **54.204.39.132**

**$ sudo tcpdump -i any -c3 -nn host 54.204.39.132**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

09:54:20.042023 IP 192.168.122.98.39326 > 54.204.39.132.80: Flags [S], seq 1375157070, win 29200, options [mss 1460,sackOK,TS val 122350391 ecr 0,nop,wscale 7], length 0

09:54:20.088127 IP 54.204.39.132.80 > 192.168.122.98.39326: Flags [S.], seq 1935542841, ack 1375157071, win 28960, options [mss 1460,sackOK,TS val 522713542 ecr 122350391,nop,wscale 9], length 0

09:54:20.088204 IP 192.168.122.98.39326 > 54.204.39.132.80: Flags [.], ack 1, win 229, options [nop,nop,TS val 122350437 ecr 522713542], length 0

3 packets captured

3 packets received by filter

0 packets dropped by kernel

#### Source IP/hostname

For example, capture packets from host **192.168.122.98**:

**$ sudo tcpdump -i any -c3 -nn src 192.168.122.98**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

10:02:15.220824 IP 192.168.122.98.39436 > 192.168.122.1.53: 59332+ A? opensource.com. (32)

10:02:15.220862 IP 192.168.122.98.39436 > 192.168.122.1.53: 20749+ AAAA? opensource.com. (32)

10:02:15.364062 IP 192.168.122.98.39334 > 54.204.39.132.80: Flags [S], seq 1108640533, win 29200, options [mss 1460,sackOK,TS val 122825713 ecr 0,nop,wscale 7], length 0

3 packets captured

3 packets received by filter

0 packets dropped by kernel

For example, capture packets sent to **192.168.122.98**:

**$ sudo tcpdump -i any -c3 -nn dst 192.168.122.98**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

10:05:03.572931 IP 192.168.122.1.53 > 192.168.122.98.47049: 2248 1/0/0 A 54.204.39.132 (48)

10:05:03.572944 IP 192.168.122.1.53 > 192.168.122.98.47049: 33770 0/0/0 (32)

10:05:03.621833 IP 54.204.39.132.80 > 192.168.122.98.39338: Flags [S.], seq 3474204576, ack 3256851264, win 28960, options [mss 1460,sackOK,TS val 522874425 ecr 122993922,nop,wscale 9], length 0

3 packets captured

3 packets received by filter

0 packets dropped by kernel

#### Port

For example, capture packets related to a web (HTTP) service by using this command:

**$ sudo tcpdump -i any -c3 -nn port 80**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

09:58:28.790548 IP 192.168.122.98.39330 > 54.204.39.132.80: Flags [S], seq 1745665159, win 29200, options [mss 1460,sackOK,TS val 122599140 ecr 0,nop,wscale 7], length 0

09:58:28.834026 IP 54.204.39.132.80 > 192.168.122.98.39330: Flags [S.], seq 4063583040, ack 1745665160, win 28960, options [mss 1460,sackOK,TS val 522775728 ecr 122599140,nop,wscale 9], length 0

09:58:28.834093 IP 192.168.122.98.39330 > 54.204.39.132.80: Flags [.], ack 1, win 229, options [nop,nop,TS val 122599183 ecr 522775728], length 0

3 packets captured

3 packets received by filter

0 packets dropped by kernel

#### Complex Expressions

You can combine filters by using the logical operators and (or &&), or (or ||), not (or !) to create more complex expressions.

For example, filter packets from source IP address **192.168.122.98** and service **HTTP** only, run:

**$ sudo tcpdump -i any -c3 -nn src 192.168.122.98 and port 80**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

10:08:00.472696 IP 192.168.122.98.39342 > 54.204.39.132.80: Flags [S], seq 2712685325, win 29200, options [mss 1460,sackOK,TS val 123170822 ecr 0,nop,wscale 7], length 0

10:08:00.516118 IP 192.168.122.98.39342 > 54.204.39.132.80: Flags [.], ack 268723504, win 229, options [nop,nop,TS val 123170865 ecr 522918648], length 0

10:08:00.516583 IP 192.168.122.98.39342 > 54.204.39.132.80: Flags [P.], seq 0:112, ack 1, win 229, options [nop,nop,TS val 123170866 ecr 522918648], length 112: HTTP: GET / HTTP/1.1

3 packets captured

3 packets received by filter

0 packets dropped by kernel

You can create more complex expressions by grouping filter with **parentheses**. In this case, enclose the entire filter expression with **quotation marks** to prevent the shell from confusing them with shell expressions.

For example, filter packets for HTTP service only (port 80) and source IP addresses **192.168.122.98** or **54.204.39.132**:

**$ sudo tcpdump -i any -c3 -nn "port 80 and (src 192.168.122.98 or src 54.204.39.132)"**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

10:10:37.602214 IP 192.168.122.98.39346 > 54.204.39.132.80: Flags [S], seq 871108679, win 29200, options [mss 1460,sackOK,TS val 123327951 ecr 0,nop,wscale 7], length 0

10:10:37.650651 IP 54.204.39.132.80 > 192.168.122.98.39346: Flags [S.], seq 854753193, ack 871108680, win 28960, options [mss 1460,sackOK,TS val 522957932 ecr 123327951,nop,wscale 9], length 0

10:10:37.650708 IP 192.168.122.98.39346 > 54.204.39.132.80: Flags [.], ack 1, win 229, options [nop,nop,TS val 123328000 ecr 522957932], length 0

3 packets captured

3 packets received by filter

0 packets dropped by kernel

For example, isolate TCP RST flags.

$ sudo tcpdump 'tcp[13] & 4!=0'

$ sudo tcpdump 'tcp[tcpflags] == tcp-rst'

Explanation: The filters above find various packets because tcp[13] looks at offset 13 in the TCP header, the number represents the location within the byte, and the !=0 means that the flag in question is set to 1, i.e. it’s on.

For example, isolate TCP SYN flags:

$ sudo tcpdump 'tcp[13] & 2!=0'  
$ sudo tcpdump 'tcp[tcpflags] == tcp-syn'

For example, isolate TCP URG flags:

$ sudo tcpdump 'tcp[13] & 32!=0'  
$ sudo tcpdump 'tcp[tcpflags] == tcp-urg'

For example, isolate TCP ACK flags:

$ sudo tcpdump 'tcp[13] & 16!=0'  
$ sudo tcpdump 'tcp[tcpflags] == tcp-ack'

For example, isolate TCP PSH flags:

$ sudo tcpdump 'tcp[13] & 8!=0'  
$ sudo tcpdump 'tcp[tcpflags] == tcp-push'

For example, isolate TCP FIN flags:

$ sudo tcpdump 'tcp[13] & 1!=0'  
$ sudo tcpdump 'tcp[tcpflags] == tcp-fin'

For example, isolate packets that have both the SYN and ACK flags set:

$ sudo tcpdump 'tcp[13]=18'

For example, find http user agents:

$ sudo tcpdump -vvAls0 | grep 'User-Agent:'

Explanation: The -l switch lets you see the traffic as you’re capturing it, and helps when sending to commands like grep.

For example, find http cookies

**$ sudo tcpdump -vvAls0 | grep 'Set-Cookie|Host:|Cookie:'**

### Checking packet content

In most case, we only need to check the packets' headers to know the source, destinations, ports, etc. to troubleshoot network connectivity issues. Sometimes, however, we need to inspect the content of the packet to ensure that the message we're sending contains what we need or that we received the expected response.

To see the packet content, tcpdump provides two additional flags: -X to print content in hex and -A to print the content in ASCII.

For example, inspect the HTTP content of a web request and display in **hex** format:

$ sudo tcpdump -i any -c5 -nn -A port 80

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

13:02:14.871803 IP 192.168.122.98.39366 > 54.204.39.132.80: Flags [S], seq 2546602048, win 29200, options [mss 1460,sackOK,TS val 133625221 ecr 0,nop,wscale 7], length 0

E..<..@.@.....zb6.'....P...@......r............

............................

13:02:14.910734 IP 54.204.39.132.80 > 192.168.122.98.39366: Flags [S.], seq 1877348646, ack 2546602049, win 28960, options [mss 1460,sackOK,TS val 525532247 ecr 133625221,nop,wscale 9], length 0

E..<..@./..a6.'...zb.P..o..&...A..q a..........

.R.W.......     ................

13:02:14.911808 IP 192.168.122.98.39366 > 54.204.39.132.80: Flags [P.], seq 1:113, ack 1, win 229, options [nop,nop,TS val 133625261 ecr 525532247], length 112: HTTP: GET / HTTP/1.1

E.....@.@..1..zb6.'....P...Ao..'...........

.....R.WGET / HTTP/1.1

User-Agent: Wget/1.14 (linux-gnu)

Accept: \*/\*

Host: opensource.com

Connection: Keep-Alive

................

13:02:14.951199 IP 54.204.39.132.80 > 192.168.122.98.39366: Flags [.], ack 113, win 57, options [nop,nop,TS val 525532257 ecr 133625261], length 0

E..4.F@./.."6.'...zb.P..o..'.......9.2.....

.R.a....................

13:02:14.955030 IP 54.204.39.132.80 > 192.168.122.98.39366: Flags [P.], seq 1:643, ack 113, win 57, options [nop,nop,TS val 525532258 ecr 133625261], length 642: HTTP: HTTP/1.1 302 Found

E....G@./...6.'...zb.P..o..'.......9.......

.R.b....HTTP/1.1 302 Found

Server: nginx

Date: Sun, 23 Sep 2018 17:02:14 GMT

Content-Type: text/html; charset=iso-8859-1

Content-Length: 207

X-Content-Type-Options: nosniff

Location: https://opensource.com/

Cache-Control: max-age=1209600

Expires: Sun, 07 Oct 2018 17:02:14 GMT

X-Request-ID: v-6baa3acc-bf52-11e8-9195-22000ab8cf2d

X-Varnish: 632951979

Age: 0

Via: 1.1 varnish (Varnish/5.2)

X-Cache: MISS

Connection: keep-alive

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html><head>

<title>302 Found</title>

</head><body>

<h1>Found</h1>

<p>The document has moved <a href="https://opensource.com/">here</a>.</p>

</body></html>

................

5 packets captured

5 packets received by filter

0 packets dropped by kernel

For example, capture packages from any interface and display in **ASCII** format:

**$ sudo tcpdump -i any -c3 -nn -X**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on any, link-type LINUX\_SLL (Linux cooked), capture size 262144 bytes

13:14:48.023471 IP 192.168.159.128.22 > 172.17.38.25.62552: Flags [P.], seq 3852176263:3852176327, ack 1729306815, win 63292, length 64

        0x0000:  4510 0068 f737 4000 4006 10f5 c0a8 9f80  E..h.7@.@.......

        0x0010:  ac11 2619 0016 f458 e59b 8b87 6713 20bf  ..&....X....g...

        0x0020:  5018 f73c 32ae 0000 4a9a b7fc 9785 4d80  P..<2...J.....M.

        0x0030:  bcc6 5660 8967 e310 5fc4 771f 5019 fab0  ..V`.g..\_.w.P...

        0x0040:  70e1 ca33 325d bf23 a386 3681 6949 5244  p..32].#..6.iIRD

        0x0050:  dd19 8001 32be c7ce dd8d d4e2 5303 acdd  ....2.......S...

        0x0060:  95b3 407d d5a3 4348                      ..@}..CH

13:14:48.023576 IP 192.168.159.128.22 > 172.17.38.25.62552: Flags [P.], seq 64:112, ack 1, win 63292, length 48

        0x0000:  4510 0058 f738 4000 4006 1104 c0a8 9f80  E..X.8@.@.......

        0x0010:  ac11 2619 0016 f458 e59b 8bc7 6713 20bf  ..&....X....g...

        0x0020:  5018 f73c 329e 0000 6b0b 5350 bcc8 3a57  P..<2...k.SP..:W

        0x0030:  5292 2b7b c6a1 8e8a 58a4 a6e7 66de cf86  R.+{....X...f...

        0x0040:  acff dab8 bfcc 40d5 6477 2695 3e39 0736  ......@.dw&.>9.6

        0x0050:  0395 c4fb d2db 5c4d                      ......\M

13:14:48.023616 IP 192.168.159.128.22 > 172.17.38.25.62552: Flags [P.], seq 112:176, ack 1, win 63292, length 64

        0x0000:  4510 0068 f739 4000 4006 10f3 c0a8 9f80  E..h.9@.@.......

        0x0010:  ac11 2619 0016 f458 e59b 8bf7 6713 20bf  ..&....X....g...

        0x0020:  5018 f73c 32ae 0000 1dcf 724c f36a 3d2f  P..<2.....rL.j=/

        0x0030:  be97 a768 1c43 2938 6c00 29e4 8e81 812d  ...h.C)8l.)....-

        0x0040:  4f6d 6966 8440 84cd 0bd0 c4b3 e8ba fec6  Omif.@..........

        0x0050:  04e6 4cf7 1e8c 45d4 3255 2529 9ef0 6adc  ..L...E.2U%)..j.

        0x0060:  5dc9 32e5 4d07 5b2a                      ].2.M.[\*

3 packets captured

3 packets received by filter

0 packets dropped by kernel

### Saving captures to a file

Saving captured packages to file is helpful when you want to capture packets and analyze them later. It also helps when there are too many packets to analyze since real-time capture can occur too fast.

To save packets to a file instead of displaying them on screen, use the option -w (for *write*).

For example, saves the output in a file named **webserver.pcap**:

**$ sudo tcpdump -i any -c10 -nn -w webserver.pcap port 80**

Now, read the contents of the file, execute tcpdump with the -r (for *read*) option:

**$ tcpdump -nn -r webserver.pcap**

reading from file webserver.pcap, link-type LINUX\_SLL (Linux cooked)

13:36:57.679494 IP 192.168.122.98.39378 > 54.204.39.132.80: Flags [S], seq 3709732619, win 29200, options [mss 1460,sackOK,TS val 135708029 ecr 0,nop,wscale 7], length 0

13:36:57.718932 IP 54.204.39.132.80 > 192.168.122.98.39378: Flags [S.], seq 1999298316, ack 3709732620, win 28960, options [mss 1460,sackOK,TS val 526052949 ecr 135708029,nop,wscale 9], length 0

13:36:57.719005 IP 192.168.122.98.39378 > 54.204.39.132.80: Flags [.], ack 1, win 229, options [nop,nop,TS val 135708068 ecr 526052949], length 0

13:36:57.719186 IP 192.168.122.98.39378 > 54.204.39.132.80: Flags [P.], seq 1:113, ack 1, win 229, options [nop,nop,TS val 135708068 ecr 526052949], length 112: HTTP: GET / HTTP/1.1

13:36:57.756979 IP 54.204.39.132.80 > 192.168.122.98.39378: Flags [.], ack 113, win 57, options [nop,nop,TS val 526052959 ecr 135708068], length 0

**Note**: Tcpdump creates a file in binary format, so you **cannot open it with a text editor**.

# Wireshark

# TShark

# Programming in C/C++

## 'libpcap' Library

Libpcap is a packet capture library for **Linux**.

### Code Flow

The general layout of a pcap sniffer is as follows:

1. **Determine which network interface** (or device) we want to sniff on. In Linux, this may be something like eth0, in BSD it may be xl1, etc. We can choose to sniff on multiple devices if we want.

Mainly-used APIs: pcap\_lookupdev(), pcap\_findalldevs()

1. **Initialize pcap**. Each seccion is differentiated by a unique handle so we can tell it apart from other sessions.

Mainly-used APIs: pcap\_open\_live() or pcap\_open\_offline()

1. If we only want to sniff specific traffic (e.g., only TCP/IP packets, only packets going to port 23, etc.) we must **create a rule set for package filtering**, "compile" it, and apply it. This is a three-phase process, all of which is closely related. The rule set is kept in a string and converted into a format that pcap can read. The compilation is actually done by calling a function within our program; it does not involve the use of an external application. Then we tell pcap to apply it to whichever session we wish for it to filter.

Mainly-used APIs: pcap\_compile() and pcap\_setfilter()

1. We tell pcap to enter its primary **execution loop**. In this state, pcap waits until it has received however many packets we want it to. Every time it gets a new packet in, it calls a callback function we have already defined. This function can do anything we want (dissect packets, print packets to the user, save packets to a file, etc.).

Mainly-used APIs: pcap\_loop() or pcap\_dispatch()

1. After our sniffing needs are satisfied, we **close our session**.

Mainly-used APIs: pcap\_close()

For sample implementation of above code flow, check this guide: <https://www.tcpdump.org/pcap.html>

### APIs

**Pcap APIs**:

<https://www.tcpdump.org/pcap3_man.html>

**Other APIs**:

|  |  |
| --- | --- |
| **API** | **Description** |
| ntohs() | Convert the type from *network* byte order to *host* byte order. |
| htons() | Convert the type from *host* byte order to *network* byte order. |
| inet\_ntop() | Convert the *IP addresses* into a human readable format (i.e., xxx.xxx.xxx.xxx) |
|  |  |

**Supporting tools**:

* Convert hex to IP: <https://www.browserling.com/tools/hex-to-ip>

### Examples

Personal\Tutorials\Network\src\PackageCapturingWithPCAP\src

Live capturing multiple protocols: <https://www.binarytides.com/packet-sniffer-code-c-libpcap-linux-sockets/>

Offline capturing multiple protocols: <https://elf11.github.io/2017/01/22/libpcap-in-C.html>

Various examples on pcap\_loop: <https://cpp.hotexamples.com/examples/-/-/pcap_loop/cpp-pcap_loop-function-examples.html>

Printing whole data of packages in ASCII format: <https://stackoverflow.com/a/63251840/14835442>

### Common Issues

* Warning: Unable to send packet: Error with PF\_PACKET send () [11]: Message too long (errno = 90): [here](https://titanwolf.org/Network/Articles/Article?AID=75e67e45-ff4a-4b53-a1f8-0fcb7ae2b2c9#gsc.tab=0) (*Reason: data packet length has exceeded MTU 1500 (MTU default is 1500)*)

## 'npcap' and 'winpcap' Library

Npcap and WinPcap are the **Window** versions of libpcap. While Npcap is still actively maintained and supported, WinPcap was shut downed and its latest version was in 2013.

Note: The Wireshark includes Npcap as one of its core.

