Tcpdump Packet Filtering

Tcpdump provides a robust and efficient way to parse the data included in our captures via packet filters. This section will examine those filters and get a glimpse at how it modifies the output from our capture.

Filtering and Advanced Syntax Options

Utilizing more advanced filtering options like those listed below will enable us to trim down what traffic is printed to output or sent to file. By reducing the amount of info we capture and write to disk, we can help reduce the space needed to write the file and help the buffer process data quicker. Filters can be handy when paired with standard tcpdump syntax options. We can capture as widely as we wish, or be super specific only to capture packets from a particular host, or even with a particular bit in the TCP header set to on. It is highly recommended to explore the more advanced filters and find different combinations.

These filters and advanced operators are by no means an exhaustive list. They were chosen because they are the most frequently used and will get us up and running quickly. When implemented, these filters will inspect any packets captured and look for the given values in the protocol header to match.

Helpful TCPDump Filters

Filter	Result
host	`host` will filter visible traffic to show anything involving the designated host. Bi-directional
src / dest	`src` and `dest` are modifiers. We can use them to designate a source or destination host or port.
net	`net` will show us any traffic sourcing from or destined to the network designated. It uses / notation.
proto	will filter for a specific protocol type. (ether, TCP, UDP, and ICMP as examples)
port	`port` is bi-directional. It will show any traffic with the specified port as the source or destination.
portrange	`portrange` allows us to specify a range of ports. (0-1024)
less / greater "< >"	`less` and `greater` can be used to look for a packet or protocol option of a specific size.
and / &&	`and` `&&` can be used to concatenate two different filters together. for example, src host AND port.
or	`or` allows for a match on either of two conditions. It does not have to meet both. It can be tricky.
not	`not` is a modifier saying anything but x. For example, not UDP.

With these filters, we can filter the network traffic on most properties to facilitate the analysis. Let us look at some examples of these filters and how they look when we use them. When using the `host` filter, whatever IP we input will be checked for in the source or destination IP field. This can be seen in the output below.

Host Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: host [IP]
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 host 172.16.146.2

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
14:50:53.072536 IP 172.16.146.2.48738 > ec2-52-31-199-148.eu-west-1.compute.amazonaws.com.https: Fl
14:50:53.108740 IP 172.16.146.2.55606 > 172.67.1.1.https: Flags [P.], seq 4227143181:4227143273, ac
14:50:53.173084 IP 172.67.1.1.https > 172.16.146.2.55606: Flags [.], ack 92, win 69, length 0
```

Cheat Sheet
Resources
Go to Questions

Table of Contents

Introduction
- Network Traffic Analysis
- Networking Primer - Layers 1-4
- Networking Primer - Layers 5-7

Analysis
- The Analysis Process
- Analysis in Practice

Tcpdump
- Tcpdump Fundamentals
- Capturing With Tcpdump (Fundamentals Labs)
- Tcpdump Packet Filtering
- Interrogating Network Traffic With Capture and Display Filters

Wireshark
- Analysis with Wireshark
- Familiarity With Wireshark
- Wireshark Advanced Usage
- Packet Inception, Dissecting Network Traffic With Wireshark
- Guided Lab: Traffic Analysis Workflow
- Decrypting RDP connections

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

```
14:50:53.175017 IP 172.16.146.2.35744 > 172.16.146.1.domain: 55991+ PTR? 148.199.31.52.in-addr.arpa
14:50:53.175714 IP 172.16.146.1.domain > 172.16.146.2.35744: 55991 1/0/0 PTR ec2-52-31-199-148.eu-w
```

This filter is often used when we want to examine only a specific host or server. With this, we can identify with whom this host or server communicates and in which way. Based on our network configurations, we will understand if this connection is legitimate. If the communication seems strange, we can use other filters and options to view the content in more detail. Besides the individual hosts, we can also define the source host as well as the target host. We can also define entire networks and their ports.

Source/Destination Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: src/dst [host|net|port] [IP|Network Range|Port]
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 src host 172.16.146.2

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
14:53:36.199628 IP 172.16.146.2.48766 > ec2-52-31-199-148.eu-west-1.compute.amazonaws.com.https: Fl
14:53:36.203166 IP 172.16.146.2.55606 > 172.67.1.1.https: Flags [P.], seq 4227144035:4227144103, ac
14:53:36.267059 IP 172.16.146.2.36424 > 172.16.146.1.domain: 40873+ PTR? 148.199.31.52.in-addr.arpa
14:53:36.267880 IP 172.16.146.2.51151 > 172.16.146.1.domain: 10032+ PTR? 2.146.16.172.in-addr.arpa.
14:53:36.276425 IP 172.16.146.2.46588 > 172.16.146.1.domain: 28357+ PTR? 1.1.67.172.in-addr.arpa. (
14:53:36.337722 IP 172.16.146.2.48766 > ec2-52-31-199-148.eu-west-1.compute.amazonaws.com.https: Fl
14:53:36.338841 IP 172.16.146.2.48766 > ec2-52-31-199-148.eu-west-1.compute.amazonaws.com.https: Fl
14:53:36.339273 IP 172.16.146.2.48766 > ec2-52-31-199-148.eu-west-1.compute.amazonaws.com.https: Fl
14:53:36.339334 IP 172.16.146.2.48766 > ec2-52-31-199-148.eu-west-1.compute.amazonaws.com.https: Fl
14:53:36.370791 IP 172.16.146.2.32972 > 172.16.146.1.domain: 3856+ PTR? 1.146.16.172.in-addr.arpa.
```

Source and destination allow us to work with the directions of communication. For example, in the last output, we have specified that our `source` host is `172.16.146.2`, and only packets sent from this host will be intercepted. This can be done for ports, and network ranges as well. An example of this utilizing `src port #` would look something like this:

Utilizing Source With Port as a Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 tcp src port 80

06:17:08.222534 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [S.]
06:17:08.783340 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:08.993643 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:09.123830 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:09.754737 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:09.864896 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [P.]
06:17:09.945011 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
```

Notice now that we only see one side of the conversation? This is because we are filtering on the source port of 80 (HTTP). Used in this manner, `net` will grab anything matching the `/` notation for a network. In the example, we are looking for anything destined for the `172.16.146.0/24` network.

Using Destination in Combination with the Net Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 dest net 172.16.146.0/24

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
16:33:14.376003 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [.], ack 1486880537, win 316, opt
16:33:14.442123 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [P.], seq 0:385, ack 1, win 316,
16:33:14.442188 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [P.], seq 385:1803, ack 1, win 31
16:33:14.442223 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [.], seq 1803:4639, ack 1, win 31
16:33:14.443161 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [P.], seq 4639:5817, ack 1, win 3
16:33:14.443199 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [.], seq 5817:8653, ack 1, win 31
16:33:14.444407 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [.], seq 8653:10071, ack 1, win 3
16:33:14.445479 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [.], seq 10071:11489, ack 1, win
```

```
16:33:14.445531 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [.], seq 11489:12907, ack 1, win
16:33:14.446955 IP 64.233.177.103.443 > 172.16.146.2.36050: Flags [.], seq 12907:14325, ack 1, win
```

This filter can utilize the common protocol name or protocol number for any IP, IPv6, or Ethernet protocol. Common examples would be `tcp[6], udp[17], or icmp[1]`. In the outputs below, we will utilize both the common name (top) and the protocol number (bottom). We can see it produced the same output. For the most part, these are interchangeable, but utilizing `proto` will become more useful when you are starting to dissect a specific part of the IP or other protocol headers. It will be more apparent later in this section when we talk about looking for TCP flags. We can take a look at this resource for a helpful list covering protocol numbers.

Protocol Filter - Common Name

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: [tcp/udp/icmp]
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 udp

06:17:09.864896 IP dialin-145-254-160-237.pools.arcor-ip.net.3009 > 145.253.2.203.domain: 35+ A? pa
06:17:10.225414 IP 145.253.2.203.domain > dialin-145-254-160-237.pools.arcor-ip.net.3009: 35 4/0/0
```

Protocol Filter - Number

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: proto [protocol number]
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 proto 17

06:17:09.864896 IP dialin-145-254-160-237.pools.arcor-ip.net.3009 > 145.253.2.203.domain: 35+ A? pa
06:17:10.225414 IP 145.253.2.203.domain > dialin-145-254-160-237.pools.arcor-ip.net.3009: 35 4/0/0
```

Using the `port` filter, we should keep in mind what we are looking for and how that protocol functions. Some standard protocols like HTTP or HTTPS only use ports 80 and 443 with the transport protocol of TCP. With that in mind, picture ports as a simple way to establish connections and protocols like TCP and UDP to determine if they use an established method. Ports by themselves can be used for anything, so filtering on port 80 will show all traffic over that port number. However, if we are looking to capture all HTTP traffic, utilizing `tcp port 80` will ensure we only see HTTP traffic.

With protocols that use both TCP and UDP for different functions, such as DNS, we can filter looking at one or the other `TCP/UDP port 53` or filter for `port 53`. By doing this, we will see any traffic-utilizing that port, regardless of the transport protocol.

Port Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: port [port number]
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 tcp port 443

06:17:07.311224 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [S],
06:17:08.222534 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [S.]
06:17:08.222534 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:08.222534 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [P.]
06:17:08.783340 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:08.993643 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:09.123830 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:09.123830 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:09.324118 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:09.754737 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:09.864896 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [P.]
06:17:09.864896 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:09.945011 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:10.125270 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:10.205385 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:10.295515 IP dialin-145-254-160-237.pools.arcor-ip.net.3371 > 216.239.59.99.http: Flags [P.],
```

Apart from the individual ports, we can also define specific ranges of these ports, which are then listened to by

Apart from the individual ports, we can also define specific ranges of these ports, which are then listened to by TCPdump. Listening on a range of ports can be especially useful when we see network traffic from ports that do not match the services running on our servers. For example, if we have a web server with TCP ports 80 and 443 running in a particular segment of our network and suddenly have outgoing network traffic from TCP port 10000 or others, it is very suspicious.

The `portrange` filter, as seen below, allows us to see everything from within the port range. In the example, we see some DNS traffic along with some HTTP web requests.

Port Range Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: portrange [portrange 0-65535]
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 portrange 0-1024

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
13:10:35.092477 IP 172.16.146.1.domain > 172.16.146.2.32824: 47775 1/0/0 CNAME autopush.prod.mozaws
13:10:35.093217 IP 172.16.146.2.48078 > 172.16.146.1.domain: 30234+ A? ocsp.pki.goog. (31)
13:10:35.093334 IP 172.16.146.2.48078 > 172.16.146.1.domain: 32024+ AAAA? ocsp.pki.goog. (31)
13:10:35.136255 IP 172.16.146.1.domain > 172.16.146.2.48078: 32024 2/0/0 CNAME pki-goog.l.google.co
13:10:35.137348 IP 172.16.146.1.domain > 172.16.146.2.48078: 30234 2/0/0 CNAME pki-goog.l.google.co
13:10:35.137989 IP 172.16.146.2.55074 > atl26s18-in-f3.1e100.net.http: Flags [S], seq 1146136517, w
13:10:35.174443 IP atl26s18-in-f3.1e100.net.http > 172.16.146.2.55074: Flags [S.], seq 345110814, a
13:10:35.174481 IP 172.16.146.2.55074 > atl26s18-in-f3.1e100.net.http: Flags [.], ack 1, win 502, o
13:10:35.174716 IP 172.16.146.2.55074 > atl26s18-in-f3.1e100.net.http: Flags [P.], seq 1:379, ack 1
13:10:35.208007 IP atl26s18-in-f3.1e100.net.http > 172.16.146.2.55074: Flags [.], ack 379, win 261,
```

Next, we are looking for any packet less than 64 bytes. From the following output, we can see that for this capture, those packets mainly consisted of `SYN`, `FIN`, or `KeepAlive` packets. Less than and greater than can be a helpful modifier set. For example, let us say we are looking to capture traffic that includes a file transfer or set of files. We know these files will be larger than regular traffic. To demonstrate, we can utilize `greater 500` (alternatively `'>500'`), which will only show us packets with a size larger than 500 bytes. This will strip out all the extra packets from the view we know we are not concerned with already.

Less/Greater Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: less/greater [size in bytes]
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 less 64

06:17:07.311224 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [S],
06:17:08.222534 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [S.]
06:17:08.222534 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:08.783340 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
06:17:09.123830 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:09.324118 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:09.864896 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:10.125270 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:10.325558 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:10.806249 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:10.956465 IP 216.239.59.99.http > dialin-145-254-160-237.pools.arcor-ip.net.3371: Flags [.],
06:17:11.126710 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:11.266912 IP dialin-145-254-160-237.pools.arcor-ip.net.3371 > 216.239.59.99.http: Flags [.],
06:17:11.527286 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:11.667488 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:11.807689 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:12.088092 IP dialin-145-254-160-237.pools.arcor-ip.net.3371 > 216.239.59.99.http: Flags [.],
06:17:12.328438 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:25.216971 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [F.]
06:17:25.216971 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [.],
06:17:37.374452 IP dialin-145-254-160-237.pools.arcor-ip.net.3372 > 65.208.228.223.http: Flags [F.]
06:17:37.704928 IP 65.208.228.223.http > dialin-145-254-160-237.pools.arcor-ip.net.3372: Flags [.],
```

Above was an excellent example of using `less`. We can utilize the modifier `greater 500` to only show me packets with 500 or more bytes. It came back with a unique response in the ASCII. Can we tell what happened here?

Utilizing Greater

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 greater 500

21:12:43.548353 IP 192.168.0.1.telnet > 192.168.0.2.1550: Flags [P.], seq 401695766:401696254, ack
E...;...@.................d.......C........
.%.6..)(Warning: no Kerberos tickets issued.
OpenBSD 2.6-beta (OOF) #4: Tue Oct 12 20:42:32 CDT 1999

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code.  With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.
```

`AND` as a modifier will show us anything that meets both requirements set. For example, `host 10.12.1.122 and tcp port 80` will look for anything from the source host and contain port 80 TCP or UDP traffic. Both criteria have to be met for the filter to capture the packet. We can see this in action below. Here we utilize `host 192.168.0.1 and port 23` as a filter. So we will see only traffic that is from this particular host that is only port 23 traffic.

AND Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: and [requirement]
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 host 192.168.0.1 and port 23

21:12:38.387203 IP 192.168.0.2.1550 > 192.168.0.1.telnet: Flags [S], seq 2579865836, win 32120, opt
21:12:38.389728 IP 192.168.0.1.telnet > 192.168.0.2.1550: Flags [S.], seq 401695549, ack 2579865837
21:12:38.389775 IP 192.168.0.2.1550 > 192.168.0.1.telnet: Flags [.], ack 1, win 32120, options [nop
21:12:38.391363 IP 192.168.0.2.1550 > 192.168.0.1.telnet: Flags [P.], seq 1:28, ack 1, win 32120, o
21:12:38.537538 IP 192.168.0.1.telnet > 192.168.0.2.1550: Flags [P.], seq 1:4, ack 28, win 17349, o
```

The other modifiers, `OR` and `NOT` provide us with a way to specify multiple conditions or negate something. Let us play with that a bit now. What do we notice about this output?

Basic Capture With No Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
14:39:51.224071 IP 172.16.146.2 > dns.google: ICMP echo request, id 19623, seq 72, length 64
14:39:51.251635 IP dns.google > 172.16.146.2: ICMP echo reply, id 19623, seq 72, length 64
14:39:51.329340 IP 172.16.146.2.39003 > 172.16.146.1.domain: 8231+ PTR? 8.8.8.8.in-addr.arpa. (38)
14:39:51.330334 IP 172.16.146.1.domain > 172.16.146.2.39003: 8231 1/0/0 PTR dns.google. (62)
14:39:51.330613 IP 172.16.146.2.50633 > 172.16.146.1.domain: 65266+ PTR? 2.146.16.172.in-addr.arpa.
14:39:51.331461 IP 172.16.146.1.domain > 172.16.146.2.50633: 65266 NXDomain* 0/0/0 (43)
14:39:51.399970 IP 172.16.146.2.54742 > 72.21.91.29.http: Flags [.], ack 358742210, win 501, option
14:39:51.420559 IP 72.21.91.29.http > 172.16.146.2.54742: Flags [.], ack 1, win 131, options [nop,n
14:39:51.432107 IP 172.16.146.2.41928 > 172.16.146.1.domain: 7232+ PTR? 1.146.16.172.in-addr.arpa.
14:39:51.432795 IP 172.16.146.1.domain > 172.16.146.2.41928: 7232 NXDomain* 0/0/0 (43)
14:39:51.433048 IP 172.16.146.2.47075 > 172.16.146.1.domain: 18932+ PTR? 29.91.21.72.in-addr.arpa.
14:39:51.434374 IP 172.16.146.1.domain > 172.16.146.2.47075: 18932 NXDomain 0/1/0 (113)
14:39:52.225941 IP 172.16.146.2 > dns.google: ICMP echo request, id 19623, seq 73, length 64
14:39:52.252523 IP dns.google > 172.16.146.2: ICMP echo reply, id 19623, seq 73, length 64
14:39:52.683881 IP 172.16.146.2.47004 > 151.139.128.14.http: Flags [.], ack 2006616877, win 501, op
14:39:52.712283 IP 151.139.128.14.http > 172.16.146.2.47004: Flags [.], ack 1, win 507, options [no
```

We have a mix of different sources and destinations along with multiple protocol types. If we were to use the `OR` (alternatively `||`) modifier, we could ask for traffic from a specific host or just ICMP traffic as an example. Let us rerun it and add in an `OR`.

OR Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: or/|| [requirement]
MisaelMacias@htb[/htb]$ sudo tcpdump -r sus.pcap icmp or host 172.16.146.1

reading from file sus.pcap, link-type EN10MB (Ethernet), snapshot length 262144
14:54:03.659163 IP 172.16.146.2 > dns.google: ICMP echo request, id 51661, seq 21, length 64
14:54:03.691278 IP dns.google > 172.16.146.2: ICMP echo reply, id 51661, seq 21, length 64
14:54:03.879882 ARP, Request who-has 172.16.146.1 tell 172.16.146.2, length 28
14:54:03.880266 ARP, Reply 172.16.146.1 is-at 8a:66:5a:11:8d:64 (oui Unknown), length 46
14:54:04.661179 IP 172.16.146.2 > dns.google: ICMP echo request, id 51661, seq 22, length 64
14:54:04.687120 IP dns.google > 172.16.146.2: ICMP echo reply, id 51661, seq 22, length 64
14:54:05.663097 IP 172.16.146.2 > dns.google: ICMP echo request, id 51661, seq 23, length 64
14:54:05.686092 IP dns.google > 172.16.146.2: ICMP echo reply, id 51661, seq 23, length 64
14:54:06.664174 IP 172.16.146.2 > dns.google: ICMP echo request, id 51661, seq 24, length 64
14:54:06.697469 IP dns.google > 172.16.146.2: ICMP echo reply, id 51661, seq 24, length 64
14:54:07.666273 IP 172.16.146.2 > dns.google: ICMP echo request, id 51661, seq 25, length 64
14:54:07.701475 IP dns.google > 172.16.146.2: ICMP echo reply, id 51661, seq 25, length 64
14:54:08.668364 IP 172.16.146.2 > dns.google: ICMP echo request, id 51661, seq 26, length 64
14:54:08.694948 IP dns.google > 172.16.146.2: ICMP echo reply, id 51661, seq 26, length 64
14:54:09.670523 IP 172.16.146.2 > dns.google: ICMP echo request, id 51661, seq 27, length 64
14:54:09.694974 IP dns.google > 172.16.146.2: ICMP echo reply, id 51661, seq 27, length 64
14:54:10.672858 IP 172.16.146.2 > dns.google: ICMP echo request, id 51661, seq 28, length 64
14:54:10.697834 IP dns.google > 172.16.146.2: ICMP echo reply, id 51661, seq 28, length 64
```

Our traffic looks a bit different now. That is because a lot of the packets matched the ICMP variable while some matched the host variable. So in this output, we can see some ARP traffic and ICMP traffic. The filter worked since 172.16.146.2 matched the other variable and appeared as a host in either the source or destination field. Now, what happens if we utilize the `NOT` (alternatively `!`) modifier.

NOT Filter

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ ### Syntax: not/! [requirement]
MisaelMacias@htb[/htb]$ sudo tcpdump -r sus.pcap not icmp

14:54:03.879882 ARP, Request who-has 172.16.146.1 tell 172.16.146.2, length 28
14:54:03.880266 ARP, Reply 172.16.146.1 is-at 8a:66:5a:11:8d:64 (oui Unknown), length 46
14:54:16.541657 IP 172.16.146.2.55592 > ec2-52-211-164-46.eu-west-1.compute.amazonaws.com.https: Fl
14:54:16.568659 IP 172.16.146.2.53329 > 172.16.146.1.domain: 24866+ A? app.hackthebox.eu. (35)
14:54:16.616032 IP 172.16.146.1.domain > 172.16.146.2.53329: 24866 3/0/0 A 172.67.1.1, A 104.20.66.
14:54:16.616396 IP 172.16.146.2.56204 > 172.67.1.1.https: Flags [S], seq 2697802378, win 64240, opt
14:54:16.637895 IP 172.67.1.1.https > 172.16.146.2.56204: Flags [S.], seq 752000032, ack 2697802379
14:54:16.637937 IP 172.16.146.2.56204 > 172.67.1.1.https: Flags [.], ack 1, win 502, length 0
14:54:16.644551 IP 172.16.146.2.56204 > 172.67.1.1.https: Flags [P.], seq 1:514, ack 1, win 502, le
14:54:16.667236 IP 172.67.1.1.https > 172.16.146.2.56204: Flags [.], ack 514, win 66, length 0
14:54:16.668307 IP 172.67.1.1.https > 172.16.146.2.56204: Flags [P.], seq 1:2766, ack 514, win 66,
14:54:16.668319 IP 172.16.146.2.56204 > 172.67.1.1.https: Flags [.], ack 2766, win 496, length 0
14:54:16.670536 IP ec2-52-211-164-46.eu-west-1.compute.amazonaws.com.https > 172.16.146.2.55592: Fl
14:54:16.670559 IP 172.16.146.2.55592 > ec2-52-211-164-46.eu-west-1.compute.amazonaws.com.https: Fl
```

It looks much different now. We only see some ARP traffic, and then we see some HTTPS traffic we did not get to before. This is because we negated any ICMP traffic from being displayed using `not icmp`.

Pre-Capture Filters VS. Post-Capture Processing

When utilizing filters, we can apply them directly to the capture or apply them when reading a capture file. By applying them to the capture, it will drop any traffic not matching the filter. This will reduce the amount of data in the captures and potentially clear out traffic we may need later, so use them only when looking for something specific, such as troubleshooting a network connectivity issue. When applying the filter to capture, we have read from a file, and the filter will parse the file and remove anything from our terminal output not matching the specified filter. Using a filter in this way can help us investigate while saving potential valuable data in the captures. It will not permanently change the capture file, and to change or clear the filter from our output will require we rerunning our command with a change in the syntax.

Interpreting Tips and Tricks

Using the `-S` switch will display absolute sequence numbers, which can be extremely long. Typically, tcpdump displays relative sequence numbers, which are easier to track and read. However, if we look for these values in another tool or log, we will only find the packet based on absolute sequence numbers. For example, 13245768092588 to 100.

The `-v`, `-X`, and `-e` switches can help you increase the amount of data captured, while the `-c`, `-n`, `-s`, `-S`, and `-q` switches can help reduce and modify the amount of data written and seen.

Many handy options that can be used but are not always directly valuable for everyone are the `-A` and `-l` switches. A will show only the ASCII text after the packet line, instead of both ASCII and Hex. `L` will tell tcpdump to output packets in a different mode. `L` will line buffer instead of pooling and pushing in chunks. It allows us to send the output directly to another tool such as `grep` using a pipe `|`.

Tips and Tricks

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$sudo tcpdump -Ar telnet.pcap

21:12:43.528695 IP 192.168.0.1.telnet > 192.168.0.2.1550: Flags [P.], seq 157:217, ack 216, win 173
E..p;...@..p.............c.......C........
.%.6..(.Last login: Sat Nov 27 20:11:43 on ttyp2 from bam.zing.org

21:12:43.546441 IP 192.168.0.2.1550 > 192.168.0.1.telnet: Flags [.], ack 217, win 32120, options [n
E..4FP@.@.s...................d...}x.......
..)(.%.6
21:12:43.548353 IP 192.168.0.1.telnet > 192.168.0.2.1550: Flags [P.], seq 217:705, ack 216, win 173
E...;...@.................d.......C........
.%.6..)(Warning: no Kerberos tickets issued.
OpenBSD 2.6-beta (OOF) #4: Tue Oct 12 20:42:32 CDT 1999

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code.  With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.


21:12:43.566442 IP 192.168.0.2.1550 > 192.168.0.1.telnet: Flags [.], ack 705, win 32120, options [n
E..4FQ@.@.s...................e...}x.0.....
..)*.%.6
```

Notice how it has the ASCII values shown below each output line because of our use of `-A`. This can be helpful when quickly looking for something human-readable in the output.

Piping a Capture to Grep

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ sudo tcpdump -Ar http.cap -l | grep 'mailto:*'

reading from file http.cap, link-type EN10MB (Ethernet), snapshot length 65535
  <a href="mailto:ethereal-web[AT]ethereal.com">ethereal-web[AT]ethereal.com</a>
  <a href="mailto:free-support[AT]thewrittenword.com">free-support[AT]thewrittenword.com</a>
  <a href="mailto:ethereal-users[AT]ethereal.com">ethereal-users[AT]ethereal.com</a>
  <a href="mailto:ethereal-web[AT]ethereal.com">ethereal-web[AT]ethereal.com</a>
```

Using `-l` in this way allowed us to examine the capture quickly and grep for keywords or formatting we suspected could be there. In this case, we used the `-l` to pass the output to `grep` and looking for any instance of the phrase `mailto:*`. This shows us every line with our search in it, and we can see the results above. Using modifiers and redirecting output can be a quick way to scrape websites for email addresses, naming standards, and much more.

We can dig as deep as we wish into the packets we captured. It requires a bit of knowledge of how the protocols are structured, however. For example, if we wanted to see only packets with the TCP SYN flag set, we could use the following command:

Looking for TCP Protocol Flags

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ tcpdump -i eth0 'tcp[13] &2 != 0'
```

This is counting to the 13th byte in the structure and looking at the 2nd bit. If it is set to 1 or ON, the SYN flag is set.

Hunting For a SYN Flag

Tcpdump Packet Filtering

```
MisaelMacias@htb[/htb]$ sudo tcpdump -i eth0 'tcp[13] &2 != 0'

tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
15:18:14.630993 IP 172.16.146.2.56244 > 172.67.1.1.https: Flags [S], seq 122498858, win 64240, opti
15:18:14.654698 IP 172.67.1.1.https > 172.16.146.2.56244: Flags [S.], seq 3728841459, ack 122498859
15:18:15.017464 IP 172.16.146.2.60202 > a23-54-168-81.deploy.static.akamaitechnologies.com.https: F
15:18:15.021329 IP 172.16.146.2.49652 > 104.16.88.20.https: Flags [S], seq 1954080833, win 64240, o
15:18:15.022640 IP 172.16.146.2.45214 > 104.18.22.52.https: Flags [S], seq 1072203471, win 64240, o
15:18:15.042399 IP 104.18.22.52.https > 172.16.146.2.45214: Flags [S.], seq 215464563, ack 10722034
15:18:15.043646 IP a23-54-168-81.deploy.static.akamaitechnologies.com.https > 172.16.146.2.60202: F
15:18:15.044764 IP 104.16.88.20.https > 172.16.146.2.49652: Flags [S.], seq 2086758283, ack 1954080
15:18:16.131983 IP 172.16.146.2.45684 > ec2-34-255-145-175.eu-west-1.compute.amazonaws.com.https: F
15:18:16.261855 IP ec2-34-255-145-175.eu-west-1.compute.amazonaws.com.https > 172.16.146.2.45684: F
```

Our results include only packets with the TCP `SYN` flag set from what we see above.

TCPDump can be a powerful tool if we understand our networking and how hosts interact with one another. Take the time to understand typical protocol header structures to spot the anomaly when the time comes. Here are a few links to further our studies on standard Protocols and their structures. Except for the Wikipedia link, each link should take us directly to the RFC that sets the standard in place for each.

Protocol RFC Links

Link	Description
IP Protocol	`RFC 791` describes IP and its functionality.
ICMP Protocol	`RFC 792` describes ICMP and its functionality.
TCP Protocol	`RFC 793` describes the TCP protocol and how it functions.
UDP Protocol	`RFC 768` describes UDP and how it operates.
RFC Quick Links	This Wikipedia article contains a large list of protocols tied to the RFC that explains their implementation.

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Cheat Sheet

Answer the question(s) below to complete this Section and earn cubes!

+ 0 What filter will allow me to see traffic coming from or destined to the host with an ip of 10.10.20.1?

host 10.10.20.1

Submit Hint

+ 0 What filter will allow me to capture based on either of two options?

OR

Submit Hint

+ 0 True or False: TCPDump will resolve IPs to hostnames by default.

True

Submit Hint

Previous Next

Mark Complete & Next