Google IT Support Certificate: Course 5: IT Security - Defense Against the Digital Dark Arts

Week 4: Securing Your Networks

Quiz: Introduction to TCP Dump

Introduction

In this lab, you'll be introduced to tcpdump and some of its features. Tcpdump is the premier network analysis tool for information security and networking professionals. As an IT Support Specialist, having a solid grasp of this application is essential if you want to understand TCP/IP. Tcpdump will help you display network traffic in a way that's easier to analyze and troubleshoot.

What you'll do

- **Command basics** - You'll learn how to use tcpdump and what some of its flags do, as well as interpret the output.
- **Packet captures** - You'll practice saving packet captures to files, and reading them back.

Using tcpdump

Now, you'll perform some tasks using tcpdump, starting with basic usage and working up to slightly more advanced topics.

Basic Usage

We'll kick things off by introducing tcpdump and running it without any options. Head's up that `tcpdump` does require root or administrator privileges in order to capture traffic, so every command must begin with `sudo`. At a minimum, you must specify an interface to listen on with the `-i` flag. You may want to check what the primary network interface name is using `ip link`. In this case, we'll be using the interface `eth0` for all the examples; this is not necessarily the interface you'd use on your own machine, though.

To use `tcpdump` to start listening for any packets on the interface, enter the command below.

Head's up: This command will fill your terminal with a constant stream of text as new packets are read. It won't stop until you press **Ctrl+C**.

```
sudo tcpdump -i eth0
```

Copied!

content_copy

This will output some basic information about packets it sees directly to standard out. It'll continue to do this until we tell it to stop. Press **Ctrl+C** to stop the stream at any time.

You can see that once `tcpdump` exits, it prints a summary of the capture performed, showing the number of packets captured, filtered, or dropped:

```
10:26:27.868546 IP nginx-us-west1-b.c.qwiklabs-terminal-vms-prod-
00.internal.46564 > 987cae65f07e.5000: Flags [P.], seq 1:15, ack 14375,
win 506, options [nop,nop,TS val 1540850423 ecr 3482513360], length 14
^C
527 packets captured
527 packets received by filter
0 packets dropped by kernel
```

By default, tcpdump will perform some basic protocol analysis. To enable more detailed analysis, use the `-v` flag to enable more verbose output. By default, tcpdump will also attempt to perform reverse DNS lookups to resolve IP addresses to hostnames, as well as replace port numbers with commonly associated service names. You can disable this behavior using the `-n` flag. It's recommended that you use this flag to avoid generating additional traffic from the DNS lookups, and to speed up the analysis. To try this out, enter this command:

Head's up: This command will fill your terminal with a constant stream of text as new packets are read. It won't stop until you press **Ctrl+C**.

```
sudo tcpdump -i eth0 -vn
```

Copied!

content_copy

You can see how the output now provides more details for each packet:

```
172.19.0.2.46564 > 172.17.0.2.5000: Flags [.], cksum 0xbb04 (correct), ack 11863, win 504, options [nop,nop,TS val 1540898645 ecr 3482561595], length 0
10:27:16.103384 IP (tos 0x0, ttl 63, id 28613, offset 0, flags [DF], proto TCP (6), length 63)
172.19.0.2.46564 > 172.17.0.2.5000: Flags [P.], cksum 0xacf9 (correct), seq 1:12, ack 11863, win 504, options [nop,nop,TS val 1540898650 ecr 3482561595], length 11
10:27:16.103391 IP (tos 0x0, ttl 64, id 11488, offset 0, flags [DF], proto TCP (6), length 52)
172.17.0.2.5000 > 172.19.0.2.46564: Flags [.], cksum 0x584f (incorrect -> 0xbaf1), ack 12, win 501, options [nop,nop,TS val 3482561601 ecr 1540898650], length 0
^C
306 packets captured
306 packets received by filter
0 packets dropped by kernel
```

Without the verbose flag, tcpdump only gives us:

- the layer 3 protocol, source, and destination addresses and ports
- TCP details, like flags, sequence and ack numbers, window size, and options

With the verbose flag, you also get all the details of the IP header, like time-to-live, IP ID number, IP options, and IP flags.

Filtering

Let's explore `tcpdump`'s filter language a bit next, along with the protocol analysis. `Tcpdump` supports a powerful language for filtering packets, so you can capture only traffic that you care about or want to analyze. The filter rules go at the very end of the command, after all other flags have been specified. We'll use filtering to only capture DNS traffic to a specific DNS server. Then, we'll generate some DNS traffic, so we can demonstrate tcpdump's ability to interpret DNS queries and responses.

Go ahead and enter the following command now.

```
sudo tcpdump -i eth0 -vn host 8.8.8.8 and port 53 &
```

Copied!

content_copy

Let's analyze how this filter is constructed, and what exactly it's doing. `Host 8.8.8.8` specifies that we only want packets where the source or destination IP address matches what we specify (in this case 8.8.8.8). If we only want traffic in one direction, we could also add a direction qualifier, like `dst` or `src` (for the destination and source IP addresses, respectively). However, leaving out the direction qualifier will match traffic in either direction.

Next, the `port 53` portion means we only want to see packets where the source or destination port matches what we specify (in this case, DNS). These two filter statements are joined together with the logical operator "`and`". This means that both halves of the filter statement must be true for a packet to be captured by our filter.

To move ahead, hit **Enter**.

To list all running jobs, use the following command:

```
jobs -l
```

Copied!

content_copy

```
[1]+   618 Running        sudo tcpdump -i eth0 -vn host 8.8.8.8 and port
53 &
```

Now, note down the **job ID** of the above process, in your local text editor.

Next, execute the following command.

```
dig @8.8.8.8 A example.com
```

Copied!

content_copy

You should see this output to the screen.

```
10:30:18.461037 IP (tos 0x0, ttl 64, id 11001, offset 0, flags [none],
proto UDP (17), length 80)
    172.17.0.2.35281 > 8.8.8.8.53: 5649+ [1au] A? example.com. (52)
10:30:18.462607 IP (tos 0x80, ttl 114, id 43094, offset 0, flags [none],
proto UDP (17), length 84)
    8.8.8.8.53 > 172.17.0.2.35281: 5649$ 1/0/1 example.com. A
93.184.216.34 (56)
; <<>> DiG 9.11.5-P4-5.1+deb10u5-Debian <<>> @8.8.8.8 A example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5649
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;example.com.                   IN      A
;; ANSWER SECTION:
example.com.            1868    IN      A       93.184.216.34
;; Query time: 2 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Aug 11 10:30:18 UTC 2021
;; MSG SIZE  rcvd: 56
```

This uses the `dig` utility to query a specific DNS server (in this case 8.8.8.8), asking it for the `A record` for the specified domain (in this case "example.com").

Once that's done, use the **job ID** (that you've noted down earlier) to bring the process to foreground with the following command:

```
fg % [job-id]
```

Copied!

content_copy

To stop this process, hit **Ctrl+C**.

```
sudo tcpdump -i eth0 -vn host 8.8.8.8 and port 53
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
```

Now, you will see two captured packets, as our filter rules should filter out any other traffic.

The first one is the DNS query, which is our question (from the process running on the terminal) going to the server. Note that, in this case, the traffic is UDP. `Tcpdump`'s analysis of the DNS query begins right after the UDP `checksum` field. It starts with the DNS ID number, followed by some UDP options, then the query type (in this case `A?` which means we're asking for an `A record)`. Next is the domain name we're interested in (`example.com`).

The second packet is the response from the server, which includes the same DNS ID from the original query, followed by the original query. After this is the answer to the query, which contains the IP address associated with the domain name.

Up next, we'll explore `tcpdump`'s ability to write packet captures to a file, then read them back from a file.

Saving Captured Packets

In the terminal, run this command:

```
sudo tcpdump -i eth0 port 80 -w http.pcap &
```

Copied!

content_copy

This starts a capture on our eth0 interface that filters for only HTTP traffic by specifying port 80. The `-w` flag indicates that we want to write the captured packets to a file named `http.pcap`.

Once that's running in the background, now generate some http traffic that'll be captured in the terminal. Don't stop the capture you started with the previous command just yet. (If you have, you can restart it now.)

To move ahead, hit **Enter**.

To list all running jobs, use the following command:

```
jobs -l
```

Copied!

content_copy

```
[1]+   648 Running        sudo tcpdump -i eth0 port 80 -w http.pcap &
```

Now, note down the **job ID** of the above process, in your local text editor.

Now, execute the following command to generate some traffic:

```
curl example.com
```

Copied!

content_copy

This command fetches the html from `example.com` and prints it to your screen. It should look like the below. (Head's up that only the first part of the output is shown here.)

```
<!doctype html>
<html>
<head>
    <title>Example Domain</title>
    <meta charset="utf-8"/>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe
UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
```

```
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>
<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You
may use this
    domain in literature without prior coordination or asking for
permission.</p>
    <p><a href="https://www.iana.org/domains/example">More
information...</a></p>
</div>
</body>
</html>
```

Once that's done, use the **job ID** (that you've noted down earlier) to bring the process to foreground with the following command:

```
fg % [job-id]
```

Copied!

content_copy

To stop this process, hit **Ctrl+C**.

It should return a summary of the number of packets captured.

```
sudo tcpdump -i eth0 port 80 -w http.pcap
^C10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

A binary file containing the packets we just captured, called `http.pcap`, will also have been created. Don't try to print the contents of this file to the screen; since it's a binary file, it'll display as a bunch of garbled text that you won't be able to read.

```
http.pcap
```

Somewhere in that file, there's information about the packets created when you pulled down the html from `example.com`. We can read from this file using tcpdump now, using this command:

```
tcpdump -r http.pcap -nv
```

Copied!

content_copy

Output:

```
reading from file http.pcap, link-type EN10MB (Ethernet)
10:33:00.317909 IP (tos 0x0, ttl 64, id 31614, offset 0, flags [DF], proto
TCP (6), length 60)
    172.17.0.2.43280 > 93.184.216.34.80: Flags [S], cksum 0xe21c
(incorrect -> 0xb084), seq 989487956, win 65320, options [mss
1420,sackOK,TS val 1202857771 ecr 0,nop,wscale 7], length 0
```

```
10:33:00.325430 IP (tos 0x60, ttl 53, id 24192, offset 0, flags [none],
proto TCP (6), length 60)
    93.184.216.34.80 > 172.17.0.2.43280: Flags [S.], cksum 0xc4f9
(correct), seq 2553104025, ack 989487957, win 65535, options [mss
1460,sackOK,TS val 315226344 ecr 1202857771,nop,wscale 9], length 0
10:33:00.325444 IP (tos 0x0, ttl 64, id 31615, offset 0, flags [DF], proto
TCP (6), length 52)
    172.17.0.2.43280 > 93.184.216.34.80: Flags [.], cksum 0xe214
(incorrect -> 0xf1c0), ack 1, win 511, options [nop,nop,TS val 1202857779
ecr 315226344], length 0
10:33:00.325500 IP (tos 0x0, ttl 64, id 31616, offset 0, flags [DF], proto
TCP (6), length 127)
    172.17.0.2.43280 > 93.184.216.34.80: Flags [P.], cksum 0xe25f
(incorrect -> 0x4f95), seq 1:76, ack 1, win 511, options [nop,nop,TS val
1202857779 ecr 315226344], length 75: HTTP, length: 75
        GET / HTTP/1.1
        Host: example.com
        User-Agent: curl/7.64.0
        Accept: */*
10:33:00.332224 IP (tos 0x60, ttl 53, id 24193, offset 0, flags [none],
proto TCP (6), length 52)
    93.184.216.34.80 > 172.17.0.2.43280: Flags [.], cksum 0xf2ed
(correct), ack 76, win 128, options [nop,nop,TS val 315226351 ecr
1202857779], length 0
10:33:00.332588 IP (tos 0x60, ttl 53, id 24194, offset 0, flags [none],
proto TCP (6), length 1659)
    93.184.216.34.80 > 172.17.0.2.43280: Flags [P.], cksum 0xe85b
(incorrect -> 0xa3b5), seq 1:1608, ack 76, win 128, options [nop,nop,TS
val 315226351 ecr 1202857779], length 1607: HTTP, length: 1607
        HTTP/1.1 200 OK
        Accept-Ranges: bytes
        Age: 501852
        Cache-Control: max-age=604800
        Content-Type: text/html; charset=UTF-8
```

Note that we don't need to use `sudo` to read packets from a file. Also note that tcpdump writes full packets to the file, not just the text-based analysis that it prints to the screen when it's operating normally.

For example, somewhere in the output you should see the html that was returned as the body of the original query in the terminal.

```
<!doctype html>
<html>
<head>
    <title>Example Domain</title>
    <meta charset="utf-8"/>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe
UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>
```

```
<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

Click Check my progress to verify the objective.

Writing packets to a file

Check my progress

Conclusion

Congrats! You've successfully used tcpdump for basic network monitoring, including filtering for specific traffic. You've also learned how to interpret the information that tcpdump outputs about a packet, along with how to save and load summaries of the packets captured during a session