Main menu

Articles Resources Downloads About

Open Organization

Use Wireshark at the Linux command line with TShark

Learning to analyze network packets is a powerful skill.

20 Jan 2020 | Gaurav Kamathe (Red Hat, Correspondent) (/users/gkamathe) | 72 | 1 comment

Most of the time when we connect to the internet, we don't think about the network protocols at work underneath that make it all possible. Right now, while you are reading this article, numerous packets are being exchanged by your computer and traveling across the internet.

To understand these protocols, you need a tool that can capture and help you analyze these packets. Wireshark (https://www.wireshark.org/) is a popular open source graphical user interface (GUI) tool for analyzing packets. However, it also provides a powerful command-line utility called TShark (https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html) for people who prefer to work on the Linux command line.

To try the examples in this article, you need to be connected to the internet. For any changes to TShark's command-line options or flags, please refer to the appropriate man pages and online documentation (https://www.wireshark.org/docs/man-pages/tshark.html). Also, I am using Fedora for these examples.

```
[gaurav@testbox ~]$ cat /etc/fedora-release
Fedora release 30 (Thirty)
[gaurav@testbox ~]$
```

Check your installation

First, ensure the required packages are installed:

```
[gaurav@testbox ~]$ rpm -qa | grep -i wireshark
wireshark-cli-3.0.1-1.fc30.x86_64
```

installed and, if so, which version:

```
[gaurav@testbox ~]$ tshark -v
TShark (Wireshark) 3.0.1 (23f278e2)

Built using gcc 9.0.1 20190312 (Red Hat 9.0.1-0.10).
[gaurav@testbox ~]$
```

If you are logged in as a regular, non-root user, you need sudo rights to use the TShark utility. Root users can skip sudo and directly run the **tshark** command.

Find network devices available to TShark

Before TShark can analyze packets, it needs to capture those packets. Network packets are processed via a network interface card (NIC) on servers, workstations, or desktops or a WiFi card on laptops. Begin by identifying the NIC or WiFi card used to connect to the internet.

To identify what network devices are available to TShark, run the following command. My laptop (which I am using for these examples) shows:

```
[gaurav@testbox ~]$ sudo tshark -D
Running as user "root" and group "root". This could be dangerous.
1. wlp61s0
2. lo (Loopback)
3. any
4. virbr0
5. enp0s31f6
6. bluetooth-monitor
7. nflog
```

I am using my WiFi card to connect to my home router for accessing the internet. You can use the **ifconfig -a** command to view all network interfaces on a system. If the **ifconfig** command is not installed, you can use the newer **ip addr show** command instead. One of the interfaces should have an IP address assigned to it. For a specific interface, you can use **ifconfig <interface-name>**, for example:

```
ifconfig wlp61s0
```

Capture some packets

Now that you know which interface is being used to connect to the internet, you can start capturing some packets using it. The **-i** option can be used to capture packets on this specific interface. You'll see a bunch of output that shows the network packets being transmitted via the interface, but you can stop it with the **Ctrl+C** command:

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
    1 0.000000000  192.168.1.9 → 192.168.1.1  DNS 77 Standard query 0
    2 0.000128115  192.168.1.9 → 192.168.1.1  DNS 77 Standard query 0
    3 0.000316195  192.168.1.9 → 192.168.1.1  DNS 77 Standard query 0
    4 0.000616019  192.168.1.9 → 192.168.1.1  DNS 77 Standard query 0
    5 0.007963200  192.168.1.1 → 192.168.1.9  DNS 93 Standard query r
    6 0.009171815  192.168.1.1 → 192.168.1.9  DNS 93 Standard query r
    7 0.011075350  192.168.1.1 → 192.168.1.9  DNS 322 Standard query
    8 0.012458151  192.168.1.1 → 192.168.1.9  DNS 322 Standard query
^C8 packets captured
[gaurav@testbox ~]$
```

```
1 0.000000000  192.168.1.9 → 192.168.1.1  DNS 77 Standard query 0xa02
2 0.000128115  192.168.1.9 → 192.168.1.1  DNS 77 Standard query 0xcc4
```

These lines include two IP addresses on either side of an arrow—these are the hosts that are exchanging the packet. The arrow's direction indicates which direction the packet is going. Therefore, **192.168.1.9 → 192.168.1.1** means the packet originated at host **192.168.1.9**, which is my laptop, and is headed for destination **192.168.1.1**, which is my home router. After the destination IP address, you see **DNS**, which is just the Domain Name System protocol, followed by a DNS query. More about that later.

You can limit the number of packets captured and displayed on the screen using the **-c** (count) option. The following example shows 10 packets being captured. Notice the protocols—you saw DNS above, and here there are other protocols like NTP and TCP:

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0 -c 10
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
    1 0.000000000  192.168.1.9 → 10.5.26.10   NTP 90 NTP Version 4, c
    2 0.803303963  192.168.1.9 → 10.5.27.10   NTP 90 NTP Version 4, c
    3 3.524867645  192.168.1.9 → 192.168.1.1  DNS 69 Standard query 0
    4 6.227373094  192.168.1.9 → 192.168.1.1  DNS 89 Standard query 0
    5 6.227395145  192.168.1.9 → 192.168.1.1  DNS 89 Standard query 0
    6 6.234878912  192.168.1.1 → 192.168.1.9  DNS 105 Standard query
    7 6.238110416  192.168.1.1 → 192.168.1.9  DNS 223 Standard query
    8 6.238446999  192.168.1.9 → 34.253.23.107 TCP 74 35326 → 443 [SY
    9 6.438833991 34.253.23.107 → 192.168.1.9  TCP 74 443 → 35326 [SY
   10 6.438947001  192.168.1.9 → 34.253.23.107 TCP 66 35326 → 443 [AC
10 packets captured
[gaurav@testbox ~]$
```

uses the **nslookup** command to query the name servers to resolve a hostname to an IP address. Before you proceed, ensure the **bind-utils** package is installed:

```
[gaurav@testbox ~]$ rpm -qa | grep -i bind-utils
bind-utils-9.11.5-13.P4.fc30.x86_64
[gaurav@testbox ~]$
```

In order to query your name server, you need to find out which one your machine is talking to. You can find that information in the **/etc/resolv.conf** file. In my case, the name server is pointed to **1.1.1.1**, which is a public DNS service provided by Cloudflare:

```
[gaurav@testbox ~]$ cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 1.1.1.1
[gaurav@testbox ~]$
```

Hostnames like Opensource.com are easy for humans to understand, but machines use IP addresses to connect to other machines over a network or the internet. For your computer to connect to opensource.com, it needs to find the site's IP address; you can find it with the command:

```
nslookup opensource.com
```

If **nslookup** is not available on your machine, you can use the **dig** command instead:

```
dig opensource.com
```

(e.g., 1.1.1.1):

```
sudo tshark -i wlp61s0 host 1.1.1.1
```

Keep that terminal running and return to the other one, then run **nslookup** (or **dig**). When the command completes, it gives Opensource.com's IP address, which is 54.204.39.132. Here is **nslookup**'s output:

```
[gaurav@testbox ~]$ nslookup opensource.com
Server:         1.1.1.1
Address:        1.1.1.1#53

Non-authoritative answer:
Name:   opensource.com
Address: 54.204.39.132

[gaurav@testbox ~]$
```

And **dig**'s output:

```
[gaurav@testbox ~]$ dig opensource.com

; <<>> DiG 9.11.5-P4-RedHat-9.11.5-13.P4.fc30 <<>> opensource.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33030
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1452
;; QUESTION SECTION:
;opensource.com.                        IN      A
```

```
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Sat Nov 02 21:05:54 IST 2019
;; MSG SIZE  rcvd: 59

[gaurav@testbox ~]$
```

So far, so good, but what is happening at the packet level? Move to the terminal where you entered the **tshark** command. It captured a few packets:

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0 host 1.1.1.1
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
    2 1.798275687  192.168.1.9 → 1.1.1.1      DNS 74 Standard query 0:
    3 1.827143443      1.1.1.1 → 192.168.1.9  DNS 90 Standard query r
    ^C packets captured
[gaurav@testbox ~]$
```

The packet below originated from my laptop **192.168.1.9** and is headed for destination **1.1.1.1**. The packet is for the DNS protocol, and it's querying (Standard query) the name server for Opensource.com:

```
2 1.798275687 192.168.1.9 → 1.1.1.1 DNS 74 Standard query 0xcda0 A op
```

The packet below is a reply coming from my name server **1.1.1.1** to my machine **192.168.1.9**. Again, it's DNS, but now it's a response for the query (Standard query response) for Opensource.com's IP address:

```
3 1.827143443      1.1.1.1 → 192.168.1.9  DNS 90 Standard query respo
```

If you know beforehand what protocol you are looking for, you can add it to

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0 udp
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
    1 0.000000000  192.168.1.9 → 1.1.1.1      DNS 89 Standard query 0:
    2 0.000068640  192.168.1.9 → 1.1.1.1      DNS 89 Standard query 0:
    3 0.032616053      1.1.1.1 → 192.168.1.9  DNS 189 Standard query
    4 0.108203529      1.1.1.1 → 192.168.1.9  DNS 241 Standard query
    5 1.268489014  192.168.1.9 → 1.1.1.1      DNS 69 Standard query 0:
    6 1.302652455      1.1.1.1 → 192.168.1.9  DNS 144 Standard query
    7 6.268558254  192.168.1.9 → 1.1.1.1      DNS 79 Standard query 0:
    8 6.268618039  192.168.1.9 → 1.1.1.1      DNS 79 Standard query 0:
    9 6.664992312      1.1.1.1 → 192.168.1.9  DNS 143 Standard query
   10 6.665088305      1.1.1.1 → 192.168.1.9  DNS 143 Standard query
^C10 packets captured
[gaurav@testbox ~]$
```

The **ping** command is often used to check if a machine is up or accessible over a network. You can run the **ping** command against Opensource.com's IP address to see if the server is up and running.

Before you do that, start a packet capture so you can analyze the packet later. Open a terminal and run the following command, which will keep running and looking for packets that are originating in or destined for IP address 54.204.39.132:

```
sudo tshark -i wlp61s0 host 54.204.39.132
```

In another terminal, run the following **ping** command. The **-c** is for count, so **-c 2** means it should send only two packets to the given host:

```
ping -c 2 54.204.39.132
```

packet loss, which suggests that the destination 54.204.39.132 responded to the **ping** requests:

```
[gaurav@testbox ~]$ ping -c 2 54.204.39.132
PING 54.204.39.132 (54.204.39.132) 56(84) bytes of data.
64 bytes from 54.204.39.132: icmp_seq=1 ttl=43 time=357 ms
64 bytes from 54.204.39.132: icmp_seq=2 ttl=43 time=278 ms

--- 54.204.39.132 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1ms
rtt min/avg/max/mdev = 278.045/317.410/356.776/39.369 ms
[gaurav@testbox ~]$
```

Move back to the terminal where TShark is running. It shows four packets: the requests in the **ping** command (**-c 2**) and two replies, hence a total of four packets:

```
Packet 1 - request (1st request)
Packet 2 - reply (to Packet 1)
Packet 3 - request (2nd request)
Packet 4 - reply (to Packet 3)
```

The output shows that it is using the [ICMP (https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol)](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol) protocol. **Ping** works over ICMP to complete its tasks:

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0 host 54.204.39.132
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
    1 0.000000000  192.168.1.9 → 54.204.39.132 ICMP 98 Echo (ping) re
    2 0.356750411 54.204.39.132 → 192.168.1.9  ICMP 98 Echo (ping) re
    3 1.000295229  192.168.1.9 → 54.204.39.132 ICMP 98 Echo (ping) re
```

Network packets are sent in binary format, so if you want to see how they look on the network, you can dump the packet's hexadecimal format by simply adding an **-x** to the **tshark** command, and you will see the hexadecimal output. The following output shows a **ping** request sent by running the command **ping -c 1 54.204.39.132**:

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0 -x -c 2 host 54.204.39.132
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
0000  28 c6 8e 3e 39 3a 48 89 e7 a0 33 db 08 00 45 00   (..>9:H...3..
0010  00 54 e6 29 40 00 40 01 34 7e c0 a8 01 09 36 cc   .T.)@.@.4~...
0020  27 84 08 00 25 5f 27 d1 00 01 7e aa bd 5d 00 00   '...%_'...~..
0030  00 00 a2 f3 0d 00 00 00 00 00 10 11 12 13 14 15   .............
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012
0060  36 37                                              67

0000  48 89 e7 a0 33 db 28 c6 8e 3e 39 3a 08 00 45 00   H...3.(..>9:.
0010  00 54 31 06 00 00 2b 01 3e a2 36 cc 27 84 c0 a8   .T1...+.>.6.'
0020  01 09 00 00 2d 5f 27 d1 00 01 7e aa bd 5d 00 00   ....-_'...~..
0030  00 00 a2 f3 0d 00 00 00 00 00 10 11 12 13 14 15   .............
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012
0060  36 37                                              67

2 packets captured
[gaurav@testbox ~]$
```

Save your output

Seeing output on the screen is OK, but often you need to save data to a file to use it later. Use the **ping** command but add **-w** to tell TShark to dump the output to a file. For example, the following saves the output to file named

Now run the **ping** command again from another terminal, but this time with a count of five packets:

```
ping -c 5 54.204.39.132
```

The TShark terminal shows that 10 packets were captured. Why 10? Because you asked **ping** to send five requests, and you got five replies, hence 10 packets. Use **Ctrl+C** to stop the packet capture:

```
[gaurav@testbox ~]$ sudo tshark -w /tmp/nlog.pcap -i wlp61s0 host 54.
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
10 ^C
[gaurav@testbox ~]$
```

TShark saved the output to the file /**tmp**/**nlog.pcap**:

```
[gaurav@testbox ~]$ ls -l /tmp/nlog.pcap
-rw-------. 1 root root 1692 Nov  2 21:10 /tmp/nlog.pcap
[gaurav@testbox ~]$
```

The **file** command shows the file type is a **pcapng** capture file, so you can't just open the file using an editor like Vim and start reading; all you'll see is a bunch of garbage characters:

```
[gaurav@testbox ~]$ sudo file /tmp/nlog.pcap
/tmp/nlog.pcap: pcapng capture file - version 1.0
[gaurav@testbox ~]$
```

packets (five requests and five replies):

```
[gaurav@testbox ~]$ sudo tshark -r /tmp/nlog.pcap
Running as user "root" and group "root". This could be dangerous.
    1 0.000000000  192.168.1.9 → 54.204.39.132 ICMP 98 Echo (ping) re
    2 0.270098703 54.204.39.132 → 192.168.1.9  ICMP 98 Echo (ping) re
    3 1.000485186  192.168.1.9 → 54.204.39.132 ICMP 98 Echo (ping) re
    4 1.323571769 54.204.39.132 → 192.168.1.9  ICMP 98 Echo (ping) re
    5 2.000955585  192.168.1.9 → 54.204.39.132 ICMP 98 Echo (ping) re
    6 2.347737132 54.204.39.132 → 192.168.1.9  ICMP 98 Echo (ping) re
    7 3.000912998  192.168.1.9 → 54.204.39.132 ICMP 98 Echo (ping) re
    8 3.269412434 54.204.39.132 → 192.168.1.9  ICMP 98 Echo (ping) re
    9 4.001573635  192.168.1.9 → 54.204.39.132 ICMP 98 Echo (ping) re
   10 4.293431592 54.204.39.132 → 192.168.1.9  ICMP 98 Echo (ping) re
[gaurav@testbox ~]$

#TCP handshake
```

A TCP handshake (https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Connection_establishment) is done before establishing a connection over a network. The examples above just queried a name server or tried to determine whether a machine is reachable via a **ping** command, neither of which requires establishing a connection with the host. Try to fetch www.opensource.com (http://www.opensource.com) via the **wget** command.

Before you run **wget**, run the following command in another terminal to capture packets. I have deliberately kept the count to three since the handshake involves initial packets:

```
sudo tshark -i wlp61s0 -c 3 host 54.204.39.132
```

```
--2019-11-02 21:13:54--  https://www.opensource.com/
Resolving www.opensource.com (www.opensource.com)... 54.204.39.132
Connecting to www.opensource.com (www.opensource.com)|54.204.39.132|:
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://opensource.com/ [following]
--2019-11-02 21:13:56--  http://opensource.com/
Resolving opensource.com (opensource.com)... 54.204.39.132
Connecting to opensource.com (opensource.com)|54.204.39.132|:80... co
HTTP request sent, awaiting response... 302 Found
Location: https://opensource.com/ [following]
--2019-11-02 21:13:57--  https://opensource.com/
Connecting to opensource.com (opensource.com)|54.204.39.132|:443... c
HTTP request sent, awaiting response... 200 OK
Length: 71561 (70K) [text/html]
Saving to: ‘index.html’

index.html                        100%[===============================

2019-11-02 21:13:59 (105 KB/s) - ‘index.html’ saved [71561/71561]

[gaurav@testbox ~]$ ^C
```

You can view the three packets below. The first packet sends a **SYN** request from my laptop to the Opensource.com server. The second packet is the Opensource.com server replying with a **SYN, ACK** flag set. Finally, the third packet is my laptop sending an **ACK** request to acknowledge receiving the second packet. This is called a TCP handshake. After this handshake, both nodes (i.e., my laptop and the Opensource.com server) can exchange data.

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0 -c 3 host 54.204.39.132
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
    1 0.000000000  192.168.1.9 → 54.204.39.132 TCP 74 58784 → 443 [SY
```

If you exclude **-c 3**, it will capture all packets, and you will see a similar ritual to close a connection. Only this time, my laptop sent a **FIN, ACK** packet to Opensource.com (in packet 1 below), followed by a **FIN, ACK** from Opensource.com to my laptop (in packet 2 below), and finally an **ACK** packet sent by my laptop to the Opensource.com server. This concludes the network connection that was established earlier, and any future connections will have to set up a TCP handshake again.

```
73 4.505715716  192.168.1.9 → 54.204.39.132 TCP 66 59574 → 443 [FI
74 4.737227282 54.204.39.132 → 192.168.1.9  TCP 66 443 → 59574 [FI
75 4.737389399  192.168.1.9 → 54.204.39.132 TCP 66 59574 → 443 [AC
```

Encrypt handshake data

These days, most websites are accessed over HTTPS instead of HTTP. This ensures the data passed between the two nodes is encrypted on the wire as it passes through the internet. To ensure data is encrypted, a TLS handshake (https://en.wikipedia.org/wiki/Transport_Layer_Security#TLS_handshake) method, which is similar to the TCP handshake, happens.

Fire another **wget** command, but this time it captures 11 packets from the beginning:

```
[gaurav@testbox ~]$ wget https://www.opensource.com
--2019-11-02 21:15:21--  https://www.opensource.com/
Resolving www.opensource.com (www.opensource.com)... 54.204.39.132
Connecting to www.opensource.com (www.opensource.com)|54.204.39.132|:
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://opensource.com/ [following]
```

```
Location: https://opensource.com/ [following]
--2019-11-02 21:15:28--  https://opensource.com/
Connecting to opensource.com (opensource.com)|54.204.39.132|:443... c
HTTP request sent, awaiting response... 200 OK
Length: 71561 (70K) [text/html]
Saving to: ‘index.html’

index.html                       100%[================================

2019-11-02 21:15:31 (114 KB/s) - ‘index.html’ saved [71561/71561]

[gaurav@testbox ~]$
```

The TCP handshake concludes in the first three packets, and the fourth to the ninth have various packets that have TLS strings, which follow a similar handshake ritual to set up a secure, encrypted connection between the two hosts:

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0 -c 11 host 54.204.39.132
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
    1 0.000000000  192.168.1.9 → 54.204.39.132 TCP 74 58800 → 443 [SY
    2 0.305006506 54.204.39.132 → 192.168.1.9  TCP 74 443 → 58800 [SY
    3 0.305135180  192.168.1.9 → 54.204.39.132 TCP 66 58800 → 443 [AC
    4 0.308282152  192.168.1.9 → 54.204.39.132 TLSv1 583 Client Hello
    5 0.613210220 54.204.39.132 → 192.168.1.9  TCP 66 443 → 58800 [AC
    6 0.613298883 54.204.39.132 → 192.168.1.9  TLSv1.2 3139 Server He
    7 0.613356054  192.168.1.9 → 54.204.39.132 TCP 66 58800 → 443 [AC
    8 0.617318607  192.168.1.9 → 54.204.39.132 TLSv1.2 192 Client Key
    9 0.919718195 54.204.39.132 → 192.168.1.9  TLSv1.2 324 New Sessio
   10 0.940858609  192.168.1.9 → 54.204.39.132 TLSv1.2 240 Applicatio
   11 1.228530079 54.204.39.132 → 192.168.1.9  TLSv1.2 754 Applicatio
11 packets captured
[gaurav@testbox ~]$
```

```
sudo tshark -i wlp61s0 host 54.204.39.132 and port 443
```

Timestamps are essential when you need to analyze packets offline to reconstruct events from the past, e.g., for debugging. Adding a **-t ad** flag to TShark will add timestamps to the beginning of each packet capture:

```
[gaurav@testbox ~]$ sudo tshark -n -i wlp61s0 -t ad
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
    1 2019-11-02 21:43:58.344158174 25:c9:8e:3f:38:3a → 48:89:e7:a0:3
    2 2019-11-02 21:43:58.344194844 48:89:e7:a0:33:db → 25:c9:8e:3f:3
    3 2019-11-02 21:44:00.223393961  192.168.1.9 → 1.1.1.1      DNS 7
    4 2019-11-02 21:44:00.223460961  192.168.1.9 → 1.1.1.1      DNS 7
    5 2019-11-02 21:44:00.266325914      1.1.1.1 → 192.168.1.9  DNS 1
    6 2019-11-02 21:44:00.269102767      1.1.1.1 → 192.168.1.9  DNS 1
^C6 packets captured
[gaurav@testbox ~]$
```

View an entire packet

So far, you have seen several examples of packets and ways to interpret them but not an entire packet. Here's how to use **ping** and the **nslookup** utility to dump an entire packet:

```
[gaurav@testbox ~]$ ping -c 1 54.204.39.132
PING 54.204.39.132 (54.204.39.132) 56(84) bytes of data.
64 bytes from 54.204.39.132: icmp_seq=1 ttl=43 time=357 ms

--- 54.204.39.132 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 356.961/356.961/356.961/0.000 ms
[gaurav@testbox ~]$
```

information on the screen. The output is divided into various sections, starting with Frames, then moving to Ethernet, then to Internet Protocol, and so on.

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0 -c 1 -V host 54.204.39.132
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on
    Interface id: 0 (wlp61s0)
        Interface name: wlp61s0
    Encapsulation type: Ethernet (1)
    Arrival Time: Nov  2, 2019 21:17:55.556150846 IST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1572709675.556150846 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp:data]
Ethernet II, Src: IntelCor_a0:33:db (48:89:e7:a0:33:db), Dst: Netgear_
    Destination: Netgear_3f:38:3a (25:c9:8e:3f:38:3a)
        Address: Netgear_3f:38:3a (25:c9:8e:3f:38:3a)
        .... ..0. .... .... .... .... = LG bit: Globally unique addre
        .... ...0 .... .... .... .... = IG bit: Individual address (u
    Source: IntelCor_a0:33:db (48:89:e7:a0:33:db)
        Address: IntelCor_a0:33:db (48:89:e7:a0:33:db)
        .... ..0. .... .... .... .... = LG bit: Globally unique addre
        .... ...0 .... .... .... .... = IG bit: Individual address (u
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.1.9, Dst: 54.204.39.132
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
```

```
        Identification: 0x8f68 (36712)
        Flags: 0x4000, Don't fragment
            0... .... .... .... = Reserved bit: Not set
            .1.. .... .... .... = Don't fragment: Set
            ..0. .... .... .... = More fragments: Not set
            ...0 0000 0000 0000 = Fragment offset: 0
        Time to live: 64
        Protocol: ICMP (1)
        Header checksum: 0x8b3f [validation disabled]
        [Header checksum status: Unverified]
        Source: 192.168.1.9
        Destination: 54.204.39.132
    Internet Control Message Protocol
        Type: 8 (Echo (ping) request)
        Code: 0
        Checksum: 0xcfc5 [correct]
        [Checksum Status: Good]
        Identifier (BE): 7399 (0x1ce7)
        Identifier (LE): 59164 (0xe71c)
        Sequence number (BE): 1 (0x0001)
        Sequence number (LE): 256 (0x0100)
        Timestamp from icmp data: Nov  2, 2019 21:17:55.000000000 IST
        [Timestamp from icmp data (relative): 0.556150846 seconds]
        Data (48 bytes)

    0000  5b 7c 08 00 00 00 00 00 10 11 12 13 14 15 16 17   [|..........
    0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$
    0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234
            Data: 5b7c080000000000101112131415161718191a1b1c1d1e1f…
            [Length: 48]

    1 packet captured
    [gaurav@testbox ~]
```

Similarly, run the following **nslookup** command and, on the side, dump the entire packet via TShark:

```
Non-authoritative answer:
Name:   opensource.com
Address: 54.204.39.132

[gaurav@testbox ~]$
```

Here is how the packet looks when you do a DNS lookup—notice the UDP protocol is being used:

```
[gaurav@testbox ~]$ sudo tshark -i wlp61s0 -c 1 -V host 1.1.1.1
Running as user "root" and group "root". This could be dangerous.
Capturing on 'wlp61s0'
Frame 1: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on
    Interface id: 0 (wlp61s0)
        Interface name: wlp61s0
    Encapsulation type: Ethernet (1)
    Arrival Time: Nov  2, 2019 21:19:32.161216715 IST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1572709772.161216715 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 88 bytes (704 bits)
    Capture Length: 88 bytes (704 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:dns]
Ethernet II, Src: IntelCor_a0:33:db (48:89:e7:a0:33:db), Dst: Netgear_
    Destination: Netgear_3f:38:3a (25:c9:8e:3f:38:3a)
        Address: Netgear_3f:38:3a (25:c9:8e:3f:38:3a)
        .... ..0. .... .... .... .... = LG bit: Globally unique addre
        .... ...0 .... .... .... .... = IG bit: Individual address (u
    Source: IntelCor_a0:33:db (48:89:e7:a0:33:db)
        Address: IntelCor_a0:33:db (48:89:e7:a0:33:db)
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable
    Total Length: 74
    Identification: 0x907d (36989)
    Flags: 0x4000, Don't fragment
        0... .... .... .... = Reserved bit: Not set
        .1.. .... .... .... = Don't fragment: Set
        ..0. .... .... .... = More fragments: Not set
        ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0xe672 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.1.9
    Destination: 1.1.1.1
User Datagram Protocol, Src Port: 60656, Dst Port: 53
    Source Port: 60656
    Destination Port: 53
    Length: 54
    Checksum: 0x2fd2 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
    [Timestamps]
        [Time since first frame: 0.000000000 seconds]
        [Time since previous frame: 0.000000000 seconds]
Domain Name System (query)
    Transaction ID: 0x303c
    Flags: 0x0100 Standard query
        0... .... .... .... = Response: Message is a query
        .000 0... .... .... = Opcode: Standard query (0)
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...1 .... .... = Recursion desired: Do query recursively
        .... .... .0.. .... = Z: reserved (0)
        .... .... ...0 .... = Non-authenticated data: Unacceptable
        clock01.util.phx2.redhat.com: type A, class IN
            Name: clock01.util.phx2.redhat.com
            [Name Length: 28]
            [Label Count: 5]
            Type: A (Host Address) (1)
            Class: IN (0x0001)

1 packet captured
[gaurav@testbox ~]$
```

Next steps

Once you are comfortable with these basics of packet capturing and analysis, you can utilize TShark's various capture and display filters when working on more advanced use cases. Refer to the online documentation for more information on these filters.

(/article/19/11/internet-security-tls-ssl-certificate-authority)

How internet security works: TLS, SSL, and CA (/article/19/11/internet-security-tls-ssl-certificate-authority)

What's behind that lock icon in your web browser?

Bryant Son (Red Hat, Correspondent) (/users/brson)

(/article/19/6/cryptography-basics-openssl-part-1)

Getting started with OpenSSL: Cryptography basics (/article/19/6/cryptography-basics-openssl-part-1)

Need a primer on cryptography basics, especially regarding OpenSSL? Read on.

Marty Kalin (/users/mkalindepauledu)

Topics :

Networking (/tags/networking) **Command line** (/tags/command-line)

(/users/gkamathe)

About the author

Gaurav Kamathe - Gaurav is a seasoned Quality Engineering professional, having worked on various enterprise-level Storage and Security products. His primary interests are Linux, Security and Malware. He loves working on the command-line and is mostly interested in low-level software and understanding how things work.

• More about me (/users/gkamathe)

Recommended reading

1 Comments

heap (/users/heap) on 08 Mar 2020 0

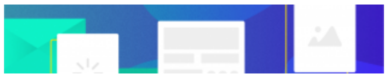
```
curl -4 https://gmail.com (https://gmail.com)

tshark -nnSX port 443
```

Program hardware from the Linux command line (/article/20/9/hardware-command-line?utm_campaign=intrel)

Open ports and route traffic through your firewall (/article/20/9/firewall?utm_campaign=intrel)

An introduction to using tcpdump at the Linux command line (/article/18/10/introduction-tcpdump?utm_campaign=intrel)



Subscribe to our weekly newsletter

Enter your email address...

Select your country or region

Subscribe

Privacy Statement

Get the highlights in your inbox every week.

Find us: