Packet sniffing using Scapy

Difficulty Level : Basic ● Last Updated : 05 Jul, 2021

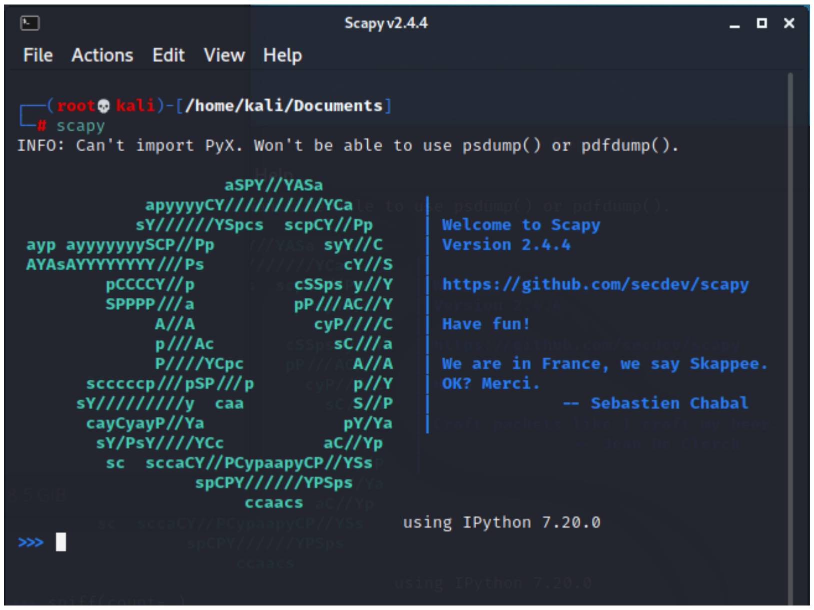
Scapy is a powerful and versatile packet manipulation tool written in python. Using scapy, a user will be able to send, sniff, dissect and forge network packets. Scapy also has the capability to store the sniffed packets in a pcap file. Using scapy, we will be able to handle tasks like trace routing, probing, scanning, unit tests, and network discovery with ease. All of these properties make scapy useful for network-based attacks.

As mentioned before scapy performs a wide range of networking tasks and one such task is packet sniffing. **Packet sniffing** is the process of capturing all the packets flowing across a computer network. The sniffed packets give away a lot of information like what website does a user visit, what contents does the user see, what does the user download and almost everything. The captured packets are usually stored for future analysis.

In this article, we will learn how to sniff packets using scapy and store the sniffed packets in a pcap file.

To work on scapy, we need to have scapy installed on our computer.

your terminal.



```
┌──(root💀kali)-[/home/kali/Documents]
└─# scapy
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.4.4
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | We are in France, we say Skappee.
       scccccp///pSP///p          p//Y   | OK? Merci.
      sY/////////y  caa           S//P   |             -- Sebastien Chabal
       cayCyayP//Ya              pY/Ya   |
        sY/PsY////YCc          aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs
                                       using IPython 7.20.0
>>> █
```

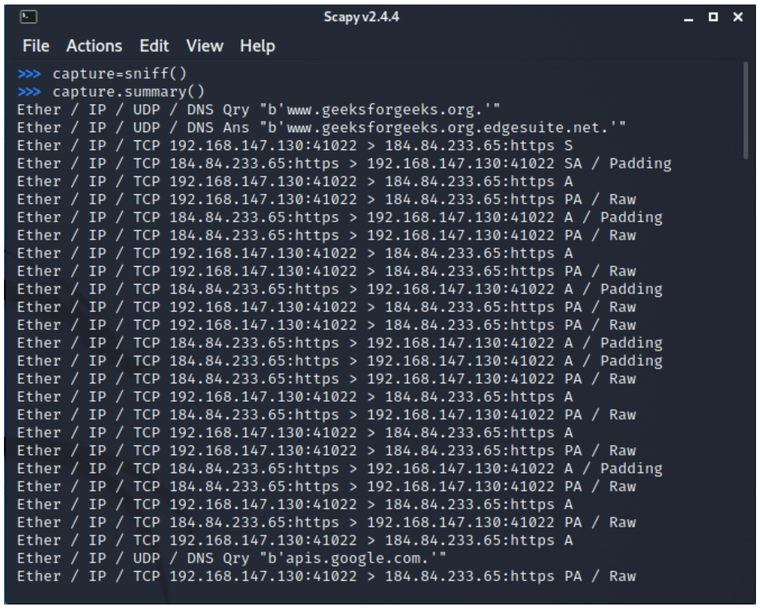
Sniffing packets using scapy:

To sniff the packets use the **sniff()** function. The sniff() function returns information about all the packets that has been sniffed.

```
capture = sniff()
```

To see the summary of packet responses, use **summary().**

```
capture.summary()
```

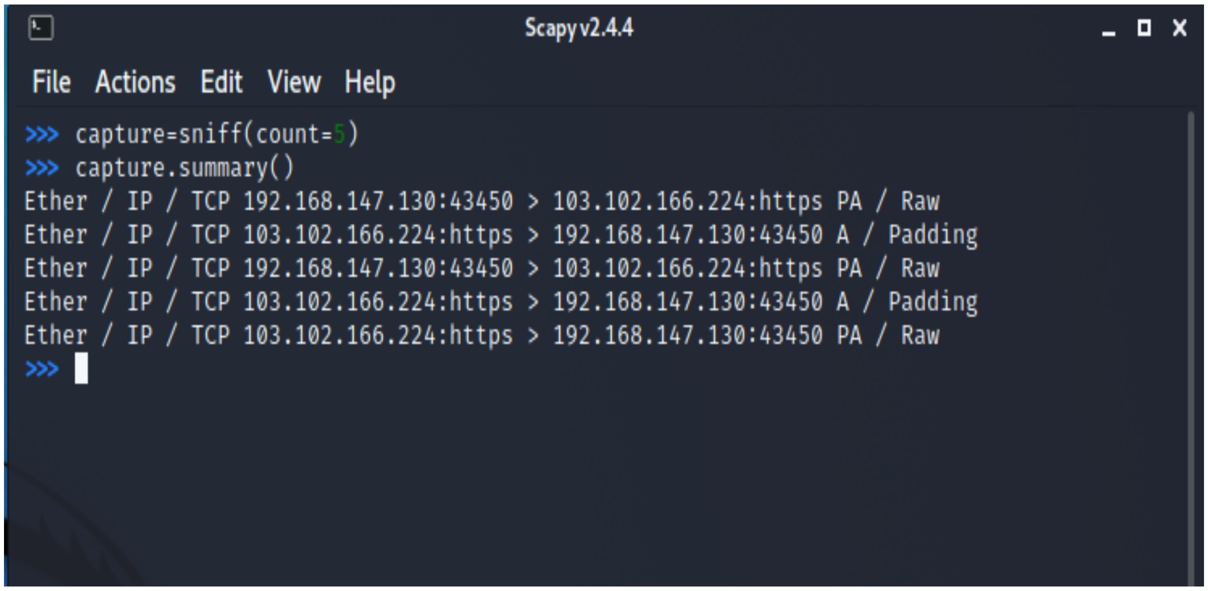


```
>>> capture=sniff()
>>> capture.summary()
Ether / IP / UDP / DNS Qry "b'www.geeksforgeeks.org.'"
Ether / IP / UDP / DNS Ans "b'www.geeksforgeeks.org.edgesuite.net.'"
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https S
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 SA / Padding
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https A
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 A / Padding
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 PA / Raw
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https A
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 A / Padding
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 A / Padding
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 A / Padding
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 PA / Raw
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https A
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 PA / Raw
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https A
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 A / Padding
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 PA / Raw
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https A
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:41022 PA / Raw
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https A
Ether / IP / UDP / DNS Qry "b'apis.google.com.'"
Ether / IP / TCP 192.168.147.130:41022 > 184.84.233.65:https PA / Raw
```

The sniff() function listens for an infinite period of time until the user interrupts.

To restrict the number of packets to be captured sniff() allows a **count** parameter. By specifying a value for the count, the packet capturing will be restricted to the specified number.

```
capture = sniff(count=5)
```



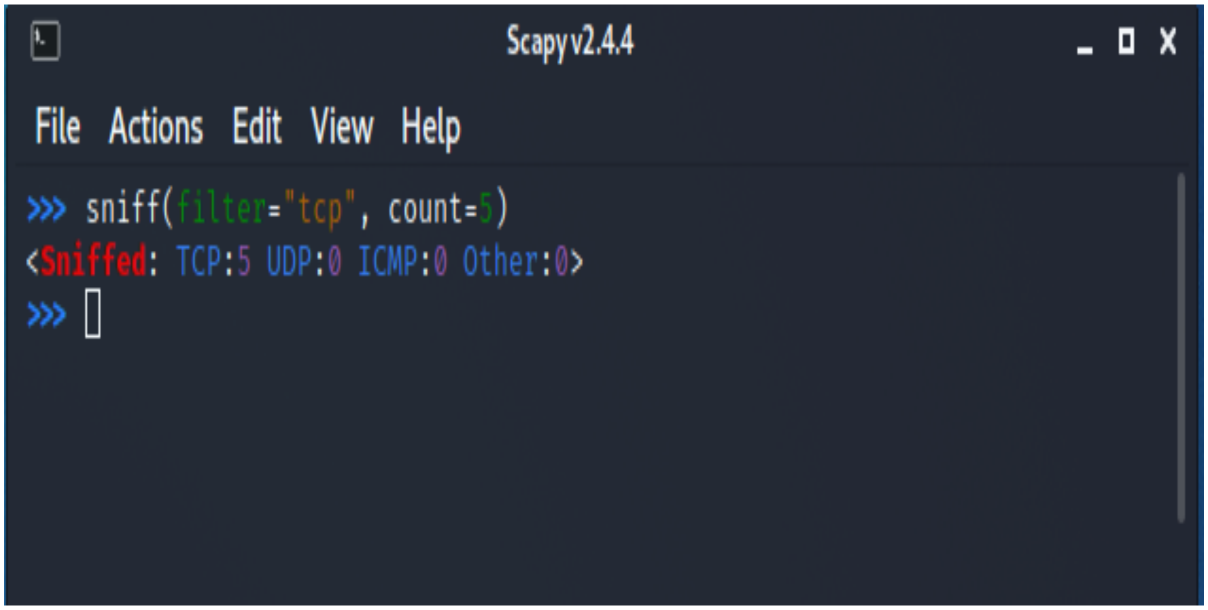
```
>>> capture=sniff(count=5)
>>> capture.summary()
Ether / IP / TCP 192.168.147.130:43450 > 103.102.166.224:https PA / Raw
Ether / IP / TCP 103.102.166.224:https > 192.168.147.130:43450 A / Padding
Ether / IP / TCP 192.168.147.130:43450 > 103.102.166.224:https PA / Raw
Ether / IP / TCP 103.102.166.224:https > 192.168.147.130:43450 A / Padding
Ether / IP / TCP 103.102.166.224:https > 192.168.147.130:43450 PA / Raw
>>>
```

uses a Berkeley Packet Filter (BPF) syntax.

The following command will capture only TCP packets:

```
sniff(filter="tcp", count=5)
```

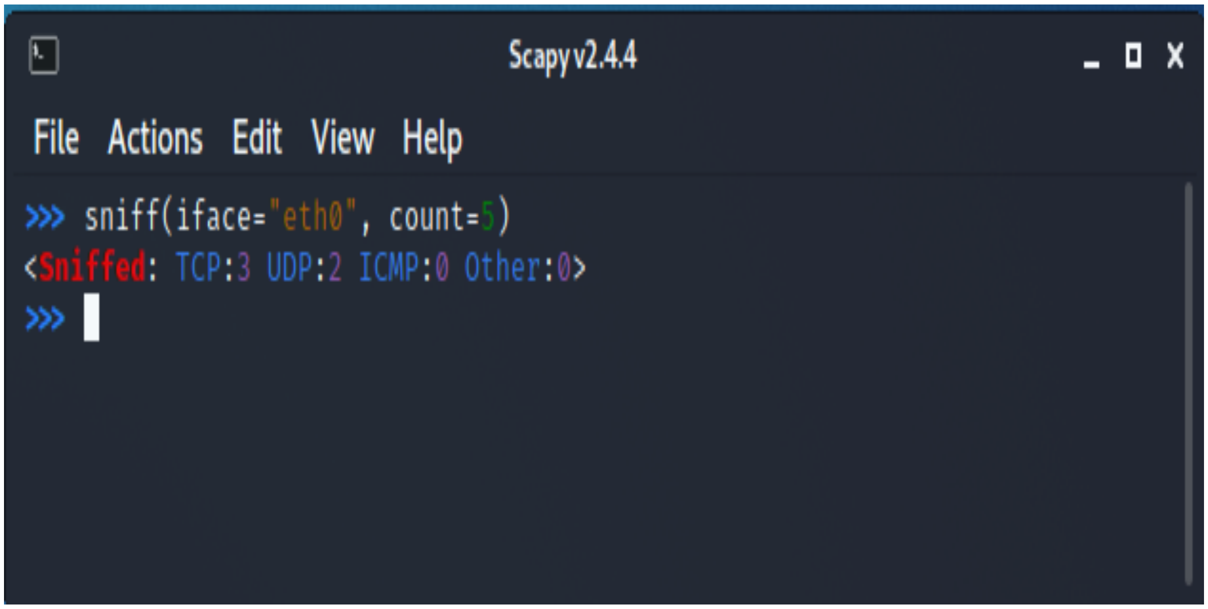
Similarly, you can filter any packet on the basis of source/destination IP address, port number, protocol and lot more by using the **BPF** syntax.



```
>>> sniff(filter="tcp", count=5)
<Sniffed: TCP:5 UDP:0 ICMP:0 Other:0>
>>> 
```

When scapy sniffs packets, it generally sniffs from all of your network interfaces. However, we can explicitly mention the interfaces that we would like to sniff on using the **iface** parameter. The iface can either be an element or a list of elements.

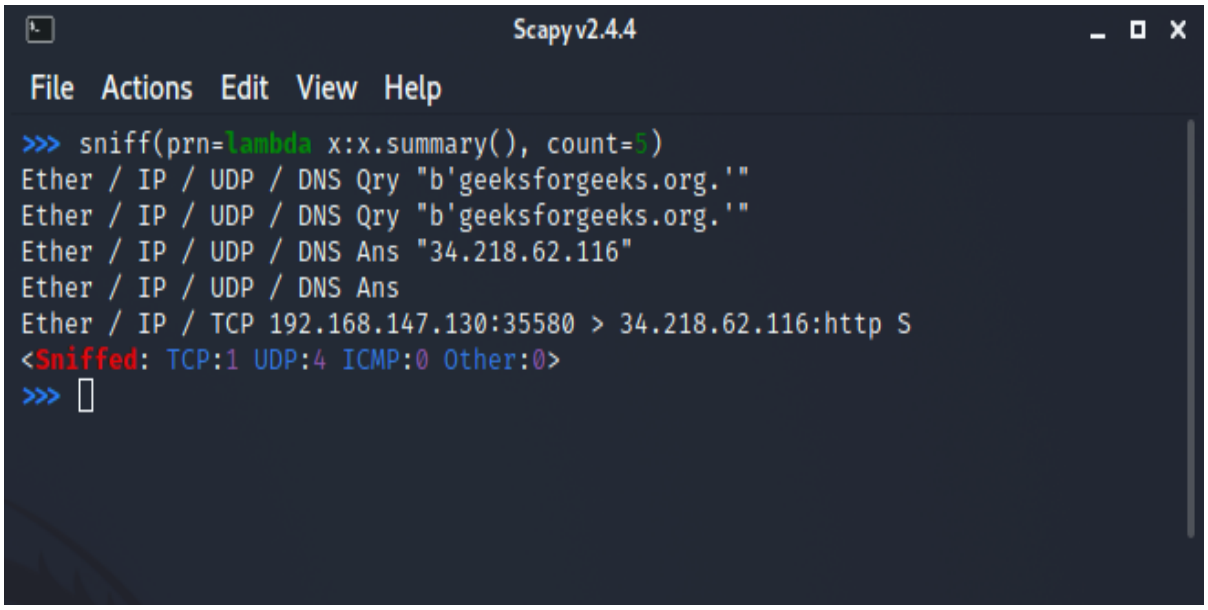
```
sniff(iface="eth0", count=5)
```



```
>>> sniff(iface="eth0", count=5)
<Sniffed: TCP:3 UDP:2 ICMP:0 Other:0>
>>>
```

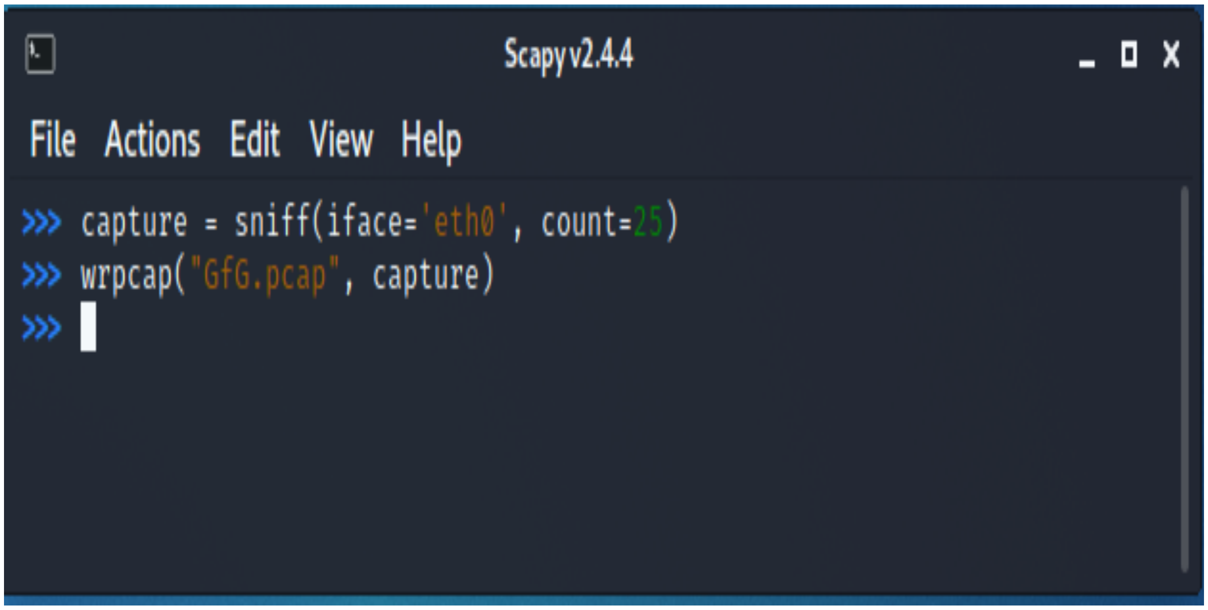
sniff() function has another interesting parameter called **prn** that allows you to pass a function that executes with each packet sniffed. This allows us to do some custom actions with each packet sniffed.

```
sniff(prn=lambda x:x.summary(), count=5)
```



```
>>> sniff(prn=lambda x:x.summary(), count=5)
Ether / IP / UDP / DNS Qry "b'geeksforgeeks.org.'"
Ether / IP / UDP / DNS Qry "b'geeksforgeeks.org.'"
Ether / IP / UDP / DNS Ans "34.218.62.116"
Ether / IP / UDP / DNS Ans
Ether / IP / TCP 192.168.147.130:35580 > 34.218.62.116:http S
<Sniffed: TCP:1 UDP:4 ICMP:0 Other:0>
>>>
```

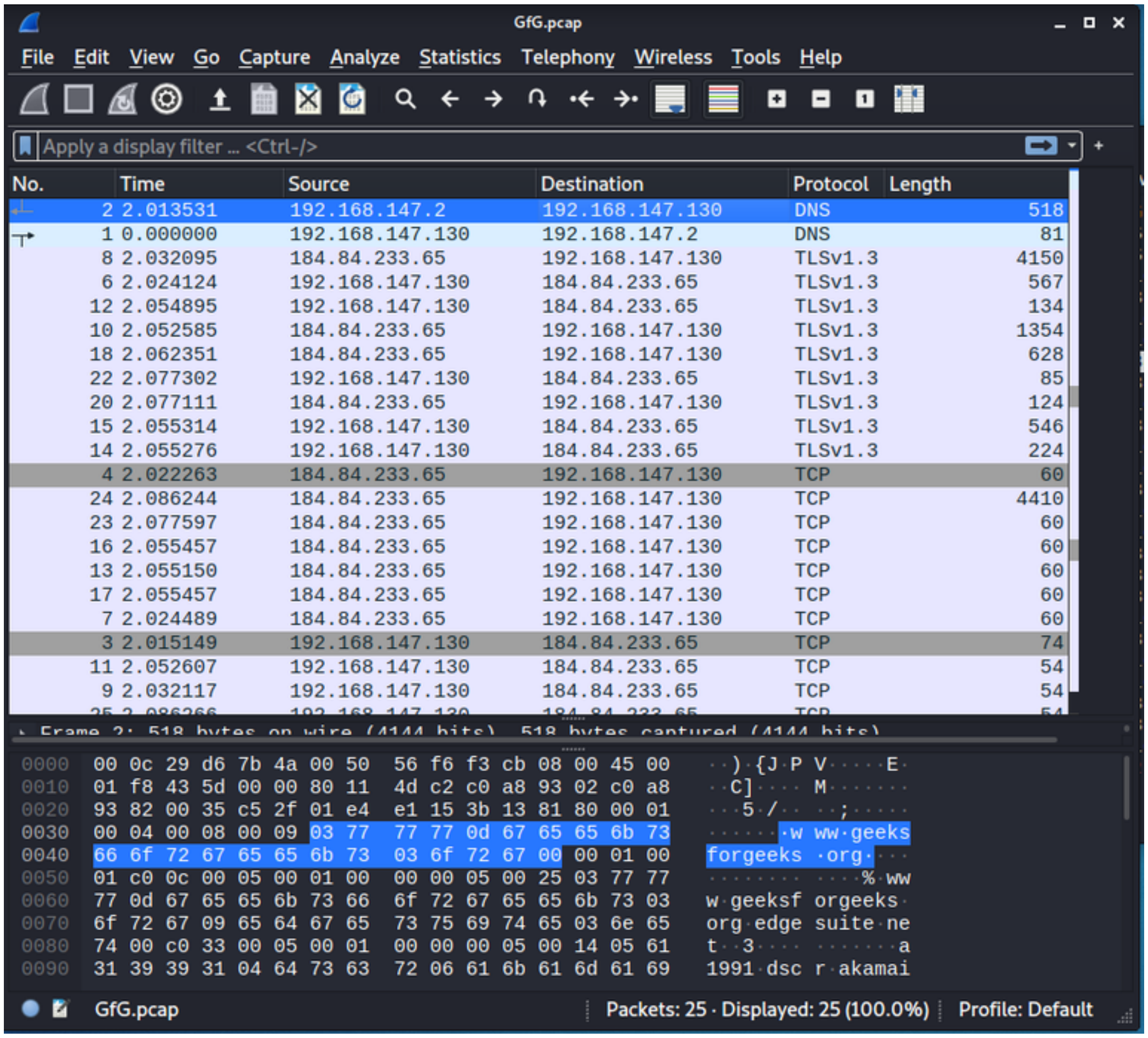
Scapy also allows us to store the sniffed packets in a **pcap** file. Running where capture is the list of sniffed packets.



```
>>> capture = sniff(iface='eth0', count=25)
>>> wrpcap("GfG.pcap", capture)
>>> 
```

The stored pcap files can be analyzed using Wireshark, tcpdump, WinDump, Packet Square, etc.

Opening GfG.pcap using Wireshark:

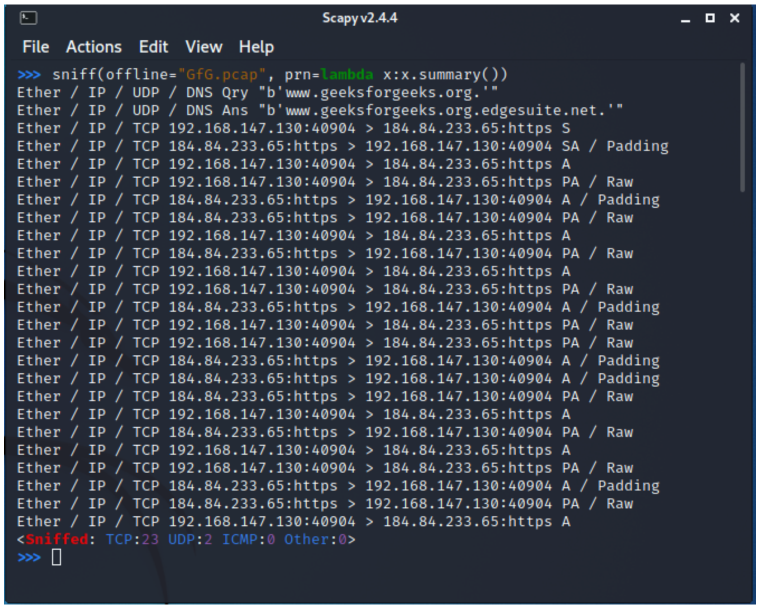

GfG.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time       Source            Destination       Protocol Length
    2 2.013531   192.168.147.2     192.168.147.130   DNS         518
    1 0.000000   192.168.147.130   192.168.147.2     DNS          81
    8 2.032095   184.84.233.65     192.168.147.130   TLSv1.3    4150
    6 2.024124   192.168.147.130   184.84.233.65     TLSv1.3     567
   12 2.054895   192.168.147.130   184.84.233.65     TLSv1.3     134
   10 2.052585   184.84.233.65     192.168.147.130   TLSv1.3    1354
   18 2.062351   184.84.233.65     192.168.147.130   TLSv1.3     628
   22 2.077302   192.168.147.130   184.84.233.65     TLSv1.3      85
   20 2.077111   184.84.233.65     192.168.147.130   TLSv1.3     124
   15 2.055314   192.168.147.130   184.84.233.65     TLSv1.3     546
   14 2.055276   192.168.147.130   184.84.233.65     TLSv1.3     224
    4 2.022263   184.84.233.65     192.168.147.130   TCP          60
   24 2.086244   184.84.233.65     192.168.147.130   TCP        4410
   23 2.077597   184.84.233.65     192.168.147.130   TCP          60
   16 2.055457   184.84.233.65     192.168.147.130   TCP          60
   13 2.055150   184.84.233.65     192.168.147.130   TCP          60
   17 2.055457   184.84.233.65     192.168.147.130   TCP          60
    7 2.024489   184.84.233.65     192.168.147.130   TCP          60
    3 2.015149   192.168.147.130   184.84.233.65     TCP          74
   11 2.052607   192.168.147.130   184.84.233.65     TCP          54
    9 2.032117   192.168.147.130   184.84.233.65     TCP          54
   25 2.086266   192.168.147.130   184.84.233.65     TCP          54
Frame 2: 518 bytes on wire (4144 bits), 518 bytes captured (4144 bits)

0000  00 0c 29 d6 7b 4a 00 50  56 f6 f3 cb 08 00 45 00   ··)·{J·P V·····E·
0010  01 f8 43 5d 00 00 80 11  4d c2 c0 a8 93 02 c0 a8   ··C]···· M·······
0020  93 82 00 35 c5 2f 01 e4  e1 15 3b 13 81 80 00 01   ···5·/·· ··;·····
0030  00 04 00 08 00 09 03 77  77 77 0d 67 65 65 6b 73   ·······w ww·geeks
0040  66 6f 72 67 65 65 6b 73  03 6f 72 67 00 00 01 00   forgeeks ·org····
0050  01 c0 0c 00 05 00 01 00  00 00 05 00 25 03 77 77   ········ ····%·ww
0060  77 0d 67 65 65 6b 73 66  6f 72 67 65 65 6b 73 03   w·geeksf orgeeks·
0070  6f 72 67 09 65 64 67 65  73 75 69 74 65 03 6e 65   org·edge suite·ne
0080  74 00 c0 33 00 05 00 01  00 00 00 05 00 14 05 61   t··3···· ·······a
0090  31 39 39 31 04 64 73 63  72 06 61 6b 61 6d 61 69   1991·dsc r·akamai

GfG.pcap     Packets: 25 · Displayed: 25 (100.0%)     Profile: Default
```

Analyzing scapy sniffed packets in Wireshark

We can also sniff packets **offline** from pcap files by running the following command:

```
sniff(offline="<file name>")
```



```
>>> sniff(offline="GfG.pcap", prn=lambda x:x.summary())
Ether / IP / UDP / DNS Qry "b'www.geeksforgeeks.org.'"
Ether / IP / UDP / DNS Ans "b'www.geeksforgeeks.org.edgesuite.net.'"
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https S
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 SA / Padding
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https A
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 A / Padding
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 PA / Raw
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https A
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 PA / Raw
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https A
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 A / Padding
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 A / Padding
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 A / Padding
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 PA / Raw
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https A
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 PA / Raw
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https A
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https PA / Raw
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 A / Padding
Ether / IP / TCP 184.84.233.65:https > 192.168.147.130:40904 PA / Raw
Ether / IP / TCP 192.168.147.130:40904 > 184.84.233.65:https A
<Sniffed: TCP:23 UDP:2 ICMP:0 Other:0>
>>> 
```

Like 3

Previous

How to plot Timeseries based charts using Pandas?

Next

Project Idea - Object Detection and Tracking