Application #6 - Basic Network Sniffer

This application is a basic network sniffer, which captures some predefined protocols and saves info about each network packet in an external file.

As with the other applications in this course, the full code is available for download.

Based on what you have learned so far in the course, it's your job now to study, understand and test the code against a network device, as you've seen me doing with the previous applications.

Feel free to alter the code in any way you want, add new protocols to be captured, more data to be exported in the external file and so on. New functionality of any kind is welcome. Just make sure to adapt your code to the contents of the packet in Scapy.

Also, please read the first 33 lines in the code carefully, as they are a good introduction to the code that follows.

As you've probably guessed, I used Scapy to build this sniffer, because this tool allows packet handling, decoding and analysis in a very intuitive way.

Also, pay special attention to the recommendations and settings that I made before starting to build the user menu and so on. I am referring to these lines and the ones above them:

net_iface = raw_input("* Enter the interface on which to run the sniffer (like 'eth1'): ")

subprocess.call(["ifconfig", net_iface, "promisc"], stdout=None, stderr=None, shell=False)

Further more, **please read the comments before every code block**, as they are good guidelines to what functionality is covered by that piece of code.

As you can see at line 72, the program asks the user what network interface is the capture process going to be executed on. A good example is entering "eth1".

net_iface = raw_input("* Enter the interface on which to run the sniffer (like 'eth1'): ")

Then, at line 80, the user is asked to enter the number of packets he wishes to be captured by the sniffer:

pkt_to_sniff = raw_input("Enter the number of packets to capture (0 is infinity): ")

At line 92, the program requires the number of seconds to run the capture:

time_to_sniff = raw_input("* Enter the number of seconds to run the capture: ")

At line 103, the program asks the user for the protocol to filter the packets by:

proto_sniff = raw_input("* Enter the protocol to filter by (arp|bootp|icmp|0 is all): ")

Lines 115 and 116 are dedicated to choosing the file name and creating the file, by opening it for writing ("w"):

file_name = raw_input("* Please give a name to the log file: ")

sniffer_log = open(file_name, "w")

At line 124, you can find the function that takes care of the parameter extraction from each packet and logging the packet info to the file: ***def packet_log(pkt)***

The program implements a counter for each packet, then records the source MAC address and destination MAC address to the file, on a single row.

Finally, the sniffing process is initialized by the *sniff()* function in Scapy, at line 138, passing the values collected from the user as arguments to this function.

pkt = sniff(iface=net_iface, count=int(pkt_to_sniff), timeout=int(time_to_sniff), prn=packet_log)

Now, to test the program, first you should have direct connectivity from the Debian VM to the router in GNS3 (R1 - 192.168.2.101 was my test device):

root@debian:/home/debian/workingdir# ping 192.168.2.101

PING 192.168.2.101 (192.168.2.101) 56(84) bytes of data.

64 bytes from 192.168.2.101: icmp_req=1 ttl=255 time=429 ms

Let's choose ICMP packets for capturing purposes and after the capture is started, I am going to ping the VM (192.168.2.100) from R1.

Please see the following way to use the program menu as an example:

root@debian:/home/debian/workingdir# python Sniffer.py

! Make sure to run this program as ROOT !

*** Enter the interface on which to run the sniffer (like 'eth1'): eth1**

Interface eth1 was set to PROMISC mode.

Enter the number of packets to capture (0 is infinity): 0

The program will capture packets until the timeout expires.

*** Enter the number of seconds to run the capture: 10**

The program will capture packets for 10 seconds.

*** Enter the protocol to filter by (arp|bootp|icmp|0 is all): icmp**

The program will capture only ICMP packets.

*** Please give a name to the log file: udemy.txt**

*** Starting the capture... Waiting for 10 seconds...**

At this point, the program listens for all the ICMP packets it receives in the next 10 seconds on **eth1** (ping from R1 now!). The results will be exported to the **udemy.txt** file.

And these are the results in this case:

root@debian:/home/debian/workingdir# cat udemy.txt

Packet 1: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 2: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00

Packet 3: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 4: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00

Packet 5: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 6: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00

Packet 7: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 8: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00

Packet 9: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 10: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00

Packet 11: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 12: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00

Packet 13: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 14: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00

Packet 15: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 16: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00

Packet 17: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 18: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00

Packet 19: SMAC: c0:01:24:c0:00:00 DMAC: 08:00:27:f2:9b:7c

Packet 20: SMAC: 08:00:27:f2:9b:7c DMAC: c0:01:24:c0:00:00