

ECE 471 Lab 5

Packet Sniffing and Spoofing Lab / ARP Cache Poisoning Attack Lab

Mitchell Dzurick

4/15/2020

Github with all documentation - <https://www.github.com/mitchdz/ECE471>

Contents

1	Packet Sniffing and Spoofing Lab	2
1.1	Task 1.1: Sniffing Packets	2
1.1.1	Task 1.1A	2
1.1.2	Task 1.1b	3
1.2	Task 1.2: Spoofing ICMP Packets	3
1.3	Task 1.4: Sniffing and-then Spoofing (Extra Credit)	3
2	ARPCache Poisoning Attack Lab	3
2.1	Task 2.1: ARP Cache Poisoning	3

1 Packet Sniffing and Spoofing Lab

1.1 Task 1.1: Sniffing Packets

1.1.1 Task 1.1A

Wireshark is the most popular sniffing tool, and it is easy to use. We will use it throughout the entire lab. However, it is difficult to use Wireshark as a building block to construct other tools. We will use Scapy for that purpose. The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs. A sample code is provided in the following:

```
#!/usr/bin/python
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='icmp',prn=print_pkt)
```

This code is placed into a file named sniffing.py and made executable with the following command

```
$ sudo chown +x sniffing.py
```

The code can then be ran with the following command

```
$ ./sniffing.py
```

The output after running as root is as follows:

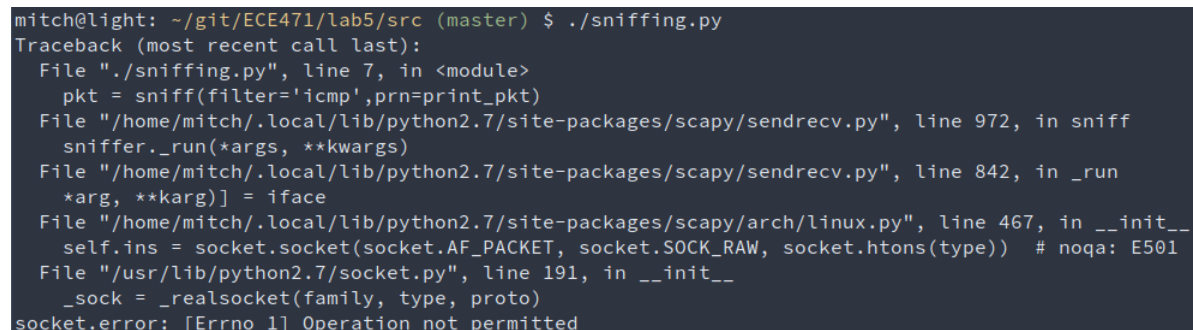
A screenshot of a terminal window with a dark background. The prompt is 'mitch@light: ~/git/ECE471/lab5/src (master)'. The user has run './sniffing.py'. The output is a 'Traceback (most recent call last):' error. The error starts with 'File "/.sniffing.py", line 7, in <module>' and 'pkt = sniff(filter='icmp',prn=print_pkt)'. It then shows the path to 'scapy/sendrecv.py' at line 972, then line 842, then 'arch/linux.py' at line 467, and finally 'socket.py' at line 191. The final error is 'socket.error: [Errno 1] Operation not permitted'.

Figure 1: Running the sniffing program without root

Looks like we need to utilize root to be able to run this script!

here's the output

```
<Sniffed: UDP:0 TCP:0 ICMP:2 Other:0>
```

1.1.2 Task 1.1b

Task 1.1B. Usually, when we sniff packets, we are only interested certain types of packets. We can do that by setting filters in sniffing. Scapy's filter use the BPF (Berkeley Packet Filter) syntax; you can find the BPF manual from the Internet. Please set the following filters and demonstrate your sniffer program again (each filter should be set separately):

1. Capture only the ICMP packet
2. Capture any TCP packet that comes from a particular IP and with a destination port number 23.
3. Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

1.2 Task 1.2: Spoofing ICMP Packets

1.3 Task 1.4: Sniffing and-then Spoofing (Extra Credit)

In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and- then-spoof program. You need two VMs on the same LAN. From VM A, you ping an IP X. This will generate an ICMP echo request packet. If X is alive, the ping program will receive an echo reply, and print out the response. Your sniff-and-then-spoof program runs on VM B, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique. Therefore, regardless of whether machine X is alive or not, the ping program will always receive a reply, indicating that X is alive. You need to use Scapy to do this task. In your report, you need to provide evidence to demonstrate that your technique works.

2 ARPCache Poisoning Attack Lab

2.1 Task 2.1: ARP Cache Poisoning