Please read this first!

For Lab 5, it is the mixture of two SEED labs (each with reduced scope). There are three required Tasks in total: in the Packet Sniffing and Spoofing Lab, tasks 1.1, 1.2, and in the ARP Cache Poisoning Attack Lab, task 2.1. Task 1.4 in the first lab is extra credit (20% bonus at most).

See the submission requirements in the end (section 4). Remember to include as much details and explanations as possible, and answer all questions in the lab.

Due date: 04/18 at midnight (Monday). Submit the electronic version in D2L (prefer PDF files).

Packet Sniffing and Spoofing Lab

1 Overview

Packet sniffing and spoofing are two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security measures in networking. There are many packet sniffing and spoofing tools, such as `Wireshark`, `Tcpdump`, `Netwox`, `Scapy`, etc. Some of these tools are widely used by security experts, as well as by attackers. Being able to use these tools is important for students, but what is more important for students in a network security course is to understand how these tools work, i.e., how packet sniffing and spoofing are implemented in software.

The objective of this lab is two-fold: learning to use the tools and understanding the technologies underlying these tools. For the second object, students will write simple sniffer and spoofing programs, and gain an in-depth understanding of the technical aspects of these programs. This lab covers the following topics:

- Scapy
- Sniffing using the `pcap` library
- Raw socket

Readings and related topics. Detailed coverage of TCP attacks can be found in Chapter 12 of the SEED book, *Computer Security: A Hands-on Approach*, by Wenliang Du.

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website.

Note for Instructors. There are two sets of tasks in this lab. The first set focuses on using tools to conduct packet sniffing and spoofing. It only requires a little bit of Python programming (usually a few lines of code); students do not need to have a prior Python programming background. The set of tasks can be used by students with a much broader background.

~~The second set of tasks is designed primarily for Computer Science/Engineering students. Students need to write their own C programs from the scratch to do sniffing and spoofing. This way, they can gain a deeper understanding on how sniffing and spoofing tools actually work. Students need to have a solid programming background for these tasks. The two sets of tasks are independent; instructors can choose to assign one set or both sets to their students, depending on their students' programming background~~.

2 Lab Task Set 1: Using Tools to Sniff and Spoof Packets

Many tools can be used to do sniffing and spoofing, but most of them only provide fixed functionalities. Scapy is different: it can be used not only as a tool, but also as a building block to construct other sniffing and spoofing tools, i.e., we can integrate the Scapy functionalities into our own program. In this set of tasks, we will use Scapy for each task.

To use Scapy, we can write a Python program, and then execute this program using Python. See the following example. We should run Python using the root privilege because the privilege is required for spoofing packets. At the beginning of the program (Line ①), we should import all Scapy's modules.

```
$ view mycode.py
#!/bin/bin/python

from scapy.all import *     ①

a = IP()
a.show()

$ sudo python mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  ...
```

We can also get into the interactive mode of Python and then run our program one line at a time at the Python prompt. This is more convenient if we need to change our code frequently in an experiment.

```
$ sudo python
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ IP ]###
  version   = 4
  ihl       = None
  ...
```

2.1 Task 1.1: Sniffing Packets

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)
```

Task 1.1A. The above program sniffs packets. For each captured packet, the callback function `print_pkt()` will be invoked; this function will print out some of the information about the packet. Run the program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations.

```
// Run the program with the root privilege
$ sudo python sniffer.py

// Run the program without the root privilege
$ python sniffer.py
```

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):

- Capture only the ICMP packet
- Capture any TCP packet that comes from a particular IP and with a destination port number 23.
- 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.

2.2 Task 1.2: Spoofing ICMP Packets

As a packet spoofing tool, Scapy allows us to set the fields of IP packets to arbitrary values. The objective of this task is to spoof IP packets with an arbitrary source IP address. We will spoof ICMP echo request packets, and send them to another VM on the same network. We will use Wireshark to observe whether our request will be accepted by the receiver. If it is accepted, an echo reply packet will be sent to the spoofed IP address. The following code shows an example of how to spoof an ICMP packets.

```
>>> from scapy.all import *
>>> a = IP()              ①
>>> a.dst = '10.0.2.3'    ②
>>> b = ICMP()            ③
>>> p = a/b               ④
>>> send(p)               ⑤
.
Sent 1 packets.
```

In the code above, Line ① creates an IP object from the IP class; a class attribute is defined for each IP header field. We can use `ls(a)` or `ls(IP)` to see all the attribute names/values. We can also use a.show() and IP.show() to do the same. Line ② shows how to set the destination IP address field. If a field is not set, a default value will be used.

```
>>> ls(a)
version    : BitField (4 bits)        = 4                 (4)
ihl        : BitField (4 bits)        = None              (None)
tos        : XByteField               = 0                 (0)
len        : ShortField               = None              (None)
id         : ShortField               = 1                 (1)
flags      : FlagsField (3 bits)      = <Flag 0 ()>       (<Flag 0 ()>)
frag       : BitField (13 bits)       = 0                 (0)
ttl        : ByteField                = 64                (64)
proto      : ByteEnumField            = 0                 (0)
chksum     : XShortField              = None              (None)
```

```
src        : SourceIPField            = '127.0.0.1'     (None)
dst        : DestIPField              = '127.0.0.1'     (None)
options    : PacketListField          = []              ([])
```

Line ③ creates an ICMP object. The default type is echo request. In Line ④, we stack `a` and `b` together to form a new object. The `/` operator is overloaded by the IP class, so it no longer represents division; instead, it means adding `b` as the payload field of `a` and modifying the fields of `a` accordingly. As a result, we get a new object that represent an ICMP packet. We can now send out this packet using `send()` in Line ⑤. Please make any necessary change to the sample code, and then demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address.

~~2.3 Task 1.3: Traceroute~~

~~The objective of this task is to use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination. This is basically what is implemented by the `traceroute` tool. In this task, we will write our own tool. The idea is quite straightforward: just send an packet (any type) to the destination, with its Time-To-Live (TTL) field set to 1 first. This packet will be dropped by the first router, which will send us an ICMP error message, telling us that the time-to-live has exceeded. That is how we get the IP address of the first router. We then increase our TTL field to 2, send out another packet, and get the IP address of the second router. We will repeat this procedure until our packet finally reach the destination. It should be noted that this experiment only gets an estimated result, because in theory, not all these packets take the same route (but in practice, they may within a short period of time). The code in the following shows one round in the procedure.~~

```
~~a = IP()~~
~~a.dst = '1.2.3.4'~~
~~a.ttl = 3~~
~~b = ICMP()~~
~~send(a/b)~~
```

~~If you are an experienced Python programmer, you can write your tool to perform the entire procedure automatically. If you are new to Python programming, you can do it by manually changing the TTL field in each round, and record the IP address based on your observation from Wireshark. Either way is acceptable, as long as you get the result.~~

2.4 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.

ARP Cache Poisoning Attack Lab

1 Overview

The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as a MAC address, given an IP address. The ARP protocol is a very simple protocol, and it does not implement any security measure. The ARP cache poisoning attack is a common attack against the ARP protocol. Under such an attack, attackers can fool the victim into accepting forged IP-to-MAC mappings. This can cause the victim's packets to be redirected to the computer with the forged MAC address.

The objective of this lab is for students to gain the first-hand experience on the ARP cache poisoning attack, and learn what damages can be caused by such an attack. In particular, students will use the ARP attack to launch a man-in-the-middle attack, where the attacker can intercept and modify the packets between the two victims A and B.

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website.

2 Task 2.1: ARP Cache Poisoning

The objective of this task is to use packet spoofing to launch an ARP cache poisoning attack on a target, such that when two victim machines A and B try to communicate with each other, their packets will be intercepted by the attacker, who can make changes to the packets, and can thus become the man in the middle between A and B. This is called Man-In-The-Middle (MITM) attack. In this lab, we use ARP cahce poisoning to conduct an MITM attack.

The following code skeleton shows how to construct an ARP packet using Scapy.

```
#!/usr/bin/python3
from scapy.all import *

E = Ether()
A = ARP()

pkt = E/A
sendp(pkt)
```

The above program constructs and sends an ARP packet. Please set necessary attribute names/values to define your own ARP packet. We can use `ls(ARP)` to see the attribute names of the ARP class. If a field is not set, a default value will be used (see the third column of the output):

```
$ python3
```

```
>>> from scapy.all import *
>>> ls(ARP)
hwtype     : XShortField                          = (1)
ptype      : XShortEnumField                      = (2048)
hwlen      : ByteField                            = (6)
plen       : ByteField                            = (4)
op         : ShortEnumField                       = (1)
hwsrc      : ARPSourceMACField                    = (None)
psrc       : SourceIPField                        = (None)
hwdst      : MACField                             = ('00:00:00:00:00:00')
pdst       : IPField                              = ('0.0.0.0')
```

In this task, we have three VMs, A, B, and M. We would like to attack A's ARP cache, such that the following results is achieved in A's ARP cache.

```
B's IP address --> M's MAC address
```

There are many ways to conduct ARP cache poisoning attack. Students need to try the following three methods, and report whether each method works or not.

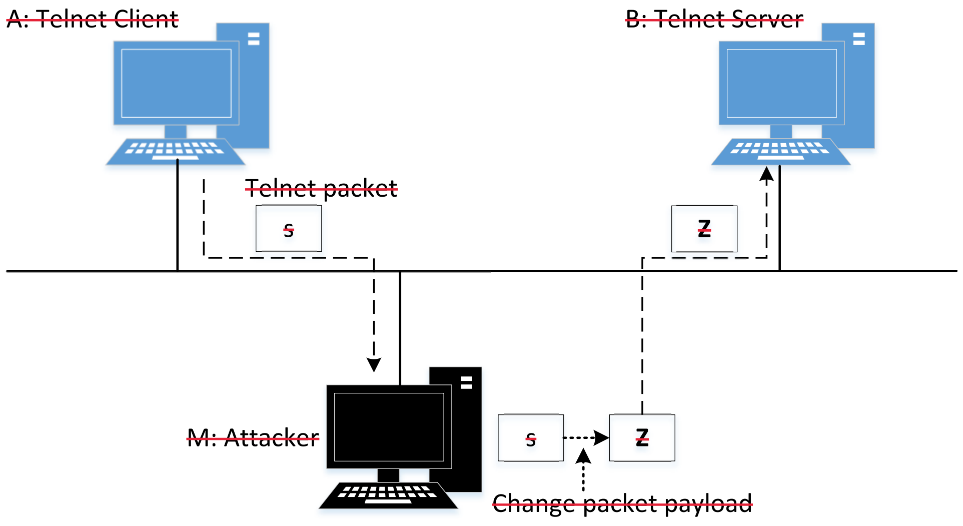
- **Task 1A (using ARP request).** On host M, construct an ARP request packet and send to host A. Check whether M's MAC address is mapped to B's IP address in A's ARP cache.
- **Task 1B (using ARP reply).** On host M, construct an ARP reply packet and send to host A. Check whether M's MAC address is mapped to B's IP address in A's ARP cache.
- **Task 1C (using ARP gratuitous message).** On host M, construct an ARP gratuitous packets. ARP gratuitous packet is a special ARP request packet. It is used when a host machine needs to update outdated information on all the other machine's ARP cache. The gratuitous ARP packet has the following characteristics:
  - The source and destination IP addresses are the same, and they are the IP address of the host issuing the gratuitous ARP.
  - The destination MAC addresses in both ARP header and Ethernet header are the broadcast MAC address (`ff:ff:ff:ff:ff:ff`).
  - No reply is expected.

~~3 Task 2: MITM Attack on Telnet using ARP Cache Poisoning~~

~~Hosts A and B are communicating using Telnet, and Host M wants to intercept their communication, so it can make changes to the data sent between A and B. The setup is depicted in Figure 1.~~

~~**Step 1 (Launch the ARP cache poisoning attack).** First, Host M conducts an ARP cache poisoning attack on both A and B, such that in A's ARP cache, B's IP address maps to M's MAC address, and in B's ARP cache, A's IP address also maps to M's MAC address. After this step, packets sent between A and B will all be sent to M. We will use the ARP cache poisoning attack from Task 1 to achieve this goal.~~

~~**Step 2 (Testing).** After the attack is successful, please try to ping each other between Hosts A and B, and report your observation. Please show Wireshark results in your report.~~



~~Figure 1: Man-In-The-Middle Attack against telnet~~

~~**Step 3 (Turn on IP forwarding).** Now we turn on the IP forwarding on Host M, so it will forward the packets between A and B. Please run the following command and repeat Step 2. Please describe your observation.~~

```
~~$ sudo sysctl net.ipv4.ip_forward=1~~
```

~~**Step 4 (Launch the MITM attack).** We are ready to make changes to the Telnet data between A and B. Assume that A is the Telnet client and B is the Telnet server. After A has connected to the Telnet server on B, for every key stroke typed in A's Telnet window, a TCP packet is generated and sent to B. We would like to intercept the TCP packet, and replace each typed character with a fixed character (say Z). This way, it does not matter what the user types on A, Telnet will always display Z.~~

~~From the previous steps, we are able to redirect the TCP packets to Host M, but instead of forwarding them, we would like to replace them with a spoofed packet. We will write a sniff-and-spoof program to accomplish this goal. In particular, we would like to do the following:~~

- ~~We first keep the IP forwarding on, so we can successfully create a Telnet connection between A to B. Once the connection is established, we turn off the IP forwarding using the following command. Please type something on A's Telnet window, and report your observation:~~

  ```
  ~~$ sudo sysctl net.ipv4.ip_forward=0~~
  ```

- ~~We run our sniff-and-spoof program on Host M, such that for the captured packets sent from A to B, we spoof a packet but with TCP different data. For packets from B to A (Telnet response), we do not make any change, so the spoofed packet is exactly the same as the original one.~~

~~A skeleton sniff-and-spoof program is shown below:~~

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

```
~~def spoof_pkt(pkt):~~
   ~~print("Original Packet.........")~~
   ~~print("Source IP : ", pkt[IP].src)~~
   ~~print("Destination IP :", pkt[IP].dst)~~

   ~~a = IP()~~
   ~~b = TCP()~~
   ~~data = pkt[TCP].payload~~
   ~~newpkt = a/b/data~~

   ~~print("Spoofed Packet.........")~~
   ~~print("Source IP : ", newpkt[IP].src)~~
   ~~print("Destination IP :", newpkt[IP].dst)~~
   ~~send(newpkt)~~

~~pkt = sniff(filter='tcp',prn=spoof_pkt)~~
```

~~The above program sniffs all the TCP packets and then spoof a new TCP packet based on the captured packets. Please make necessary changes to distinguish whether a packet is sent from A or B. If it is sent from A, set all the attribute names/values of the new packet to be the same as those of the original packet, and replace each alphanumeric characters in the payload (usually just one character in each packet) with character Z. If the captured packet is sent from B, no change will be made.~~

~~In Telnet, every character we type in the Telnet window will trigger a TCP packet. Therefore, in a typical Telnet packet from client to server, the payload only contains one character. The character will then be echoed back by the server, and the client will then display the character in its window. Therefore, what we see in the client window is not the direct result of the typing; whatever we type in the client window takes a round trip before it is displayed. If the network is disconnected, whatever we typed on the client window will not displayed, until the network is recovered. Similarly, if attackers change the character to Z during the round trip, Z will be displayed at the Telnet client window.~~

~~Here is a summary what we need to do in order to launch the MITM attack.~~

- ~~Conduct ARP cache poisoning attacks against Hosts A and B.~~
- ~~Turn on IP forwarding on Host M.~~
- ~~Telnet from host A to Host B.~~
- ~~After the Telnet connection has been established, turn off IP forwarding.~~
- ~~Conduct the sniff and spoof attack on Host M.~~

4 Submission

Students need to submit a detailed lab report to describe what they have done, what they have observed, and how they interpret the results. Reports should include evidences to support the observations. Evidences include packet traces, screenshots, etc. Reports should also list the important code snippets with explanations. Simply attaching code without any explanation will not receive credits.