Packet Sniffing and Spoofing Lab [1]

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`, 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 computer 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 for students to master and use the technologies underlying most of the sniffing and spoofing tools.

Additional reading and related topics: A detailed coverage of the packet sniffing and spoofing attack can by found in Chapter 15 of *Computer and Internet Security: A Hands-on Approach*, 2nd edition (2019) by Wenliang Du.

Lab envirnoment: This lab has been tested a pre-built Ubuntu 16.04 VM installed on the lab machine, which can also be downloaded from the SEED website.

2 Lab Tasks: Using Scapy 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 textttScapy 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
```

[1]This lab is adapted from SEEDS labs available at `https://seedsecuritylabs.org`. Please see the end of this document for the related copyright statement.

```
  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 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 2.1.a: 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 2.1.b: Usually, when we sniff packets, we are only interested certain types of packets. We can do that by setting filters in sniffing. `Scapys` 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 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 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 texttt 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 ④ . 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.

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 a 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.

4 Sniff-and-Spoof

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, regard- less 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.

5 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. You also will be asked to demo your work, and answer any questions the TA may have about it. It is **important** to note that without doing a demo, you will receive a grade of zero for the lab. All members of your group should attend, and be able to answer TA's question, that is you should make sure that every member of your group knows and can answer these questions. If the answer to questions are incomplete by any members of the group, your lab group mark will reflect that. If any member of your group is not present during the demo session, his/her mark will be zero, even if his/her name is included in the report. You are encouraged to work closely with all group members on all parts of the lab, and report any potential issues to the TAs immediately.

Have Fun! ☺