COMP-8677 Final Exam

Time: 8:30am-11:30am, Dec 18 (no late submission allowed)

IMPORTANT: 1. this paper is for ID pattern xxxxxxxx0; 2. Write name and student ID on your solution file**;** 3. **No** cheat is allowed; if caught, your final grade for this course will be **0**.

P1. **(20 points)** Choose the most appropriate answer for each question below.

1. Which of the following about linux password security is incorrect?
   A. the user password is stored in the password file at /etc/passwd
   B. password is hashed and stored in the shadow file at /etc/shadow
   C. salt can be used to prevent a rainbow attack
   D. remote login using telnet might suffer from a sniffing attack that leaks the password
2. Which layer does TLS provide security directly to?
   A. Application Layer
   B. Transport Layer
   C. Network Layer
   D. Physical Layer
3. Which of the following statements about packet sniffing is correct?
   A. A sniffer at U of Windsor can sniff the packet traffic to a computer in New York.
   B. To sniff packets not destined to the sniffer, we must turn on the promiscuous mode.
   C. Sniffer can be easily detected because it is an active attack
   D. The packet provided to a sniffer generally does not contain the link layer header.
4. Which of the following is not co-authored by Rivest?
   A. RSA
   B. MD5 hash function
   C. MD6 hash function
   D. SHA1 hash function
5. which of the following security services is not provided by TLS?
   A. confidentiality
   B. data integrity
   C. identity authentication
   D. identity anonymity

P2. (25 points) This problem is concerned with syn flooding attack. We will use **victim VM** (10.0.2.4), **attacker VM** (10.0.2.6) and **user VM** (10.0.2.5). They belong to the same LAN 10.0.2.0/24. In this experiment, we target on the telnet server on victim VM. The steps are listed as follows.

Step 1 (on victim VM) $ sudo sysctl -w net.ipv4.tcp_syncookies=0

Step 2 (on attacker VM) $ sudo netwox 76 -i 10.0.2.4 -p 23

Step 3 (on victim VM) $ netstat -tna

Step 4 (on user VM) $ telnet -l seed 10.0.2.4; then $ ssh -l seed 10.0.2.4

Answer the following questions.

(1) **[4 points]** What is Step 1 doing? Describe the idea of syn flooding attack.
(2) **[4 points]** Please interpret the command in step 2.
(3) **[4 points]** What would you expect to see in Step 3? Explain the phenomenon.
(4) **[3 points]** In step 4, do you expect both telnet and ssh connections to be successful? Why?
(5) **[10 points]** Please describe the idea of the counter measure taught in class (especially, how the attack can be avoided).

P3. **(21 points)** In this problem, you will practice the RSA encryption and signature. Let p=11, q=3.

1. **[8 points]** Generate modulus **n** using **p** and **q**. Compute the private key **d** with public key **e**=3.
2. **[5 points]** Compute the ciphertext **c** for message **m=2**.
3. **[8 points]** Generate the RSA signature (without using hash function) for m=2 and verify that your signature is valid.

P4. **(10 points)** Diffie-Hellman Key Exchange) Let g=2, p=23. Alice and Bob execute Diffie-Hellman key exchange protocol using g and p as the public parameters. Assume Alice sends A=9 to Bob while Bob sends B=18 to Alice. Suppose you are an attacker who sniffs A and B. Please determine the shared key between Alice and Bob (you do not have to simplify your calculation result).

P5. (24 points) Read the following sniffing_spoofing program **test_spoof.py**.

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

def spoof_pkt(pkt):
  if ICMP in pkt and pkt[ICMP].type == 8:
     print("Original Packet.........")
     print("Source IP : ", pkt[IP].src)
     print("Destination IP :", pkt[IP].dst)

     ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
     icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
     data = pkt[Raw].load
     newpkt = ip/icmp/data

     print("Forged Response ...........")
     print("Source IP : ", newpkt[IP].src)
     print("Destination IP :", newpkt[IP].dst)
     print("IP header protocol:",newpkt[IP].proto)

     send(newpkt,verbose=0)

pkts = sniff(filter='dst host 10.0.2.6',count=10, prn=spoof_pkt)

for i in range(0, 10):
    print("This is {}th packet with src IP adress {}".format(i, pkts[i][IP].src))
```

Please answer the following questions with **justifications** (**note**: the condition in one question does not impact another question).

(1) **[4 points]** When we run the above program, it gives an error. Explain the reason.

```
[12/11/20]seed@VM:~/.../Sniff_Spoof$ python test_spoof.py
Traceback (most recent call last):
  File "test_spoof.py", line 22, in <module>
    pkt = sniff(filter='dst host 10.0.2.6',prn=spoof_pkt)
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/sendrecv.py", line 7
31, in sniff
    *arg, **karg)] = iface
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/arch/linux.py", line
 567, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
e))
  File "/usr/lib/python2.7/socket.py", line 191, in __init__
    _sock = _realsocket(family, type, proto)
socket.error: [Errno 1] Operation not permitted
```

(2) **[4 points]** We run **test_spoof.py** on 10.0.2.4. Then, on 10.0.2.6, run **$ ping 8.8.8.8**. Consequently, the program generates the following output. It obviously does not get into the "**if block**" of **spoof_pkt**. Can you explain why?

```
[12/11/20]seed@VM:~/.../Sniff_Spoof$ sudo python test_spoof.py
This is 0th packet with src IP adress 8.8.8.8
This is 1th packet with src IP adress 8.8.8.8
This is 2th packet with src IP adress 8.8.8.8
This is 3th packet with src IP adress 8.8.8.8
This is 4th packet with src IP adress 8.8.8.8
This is 5th packet with src IP adress 8.8.8.8
This is 6th packet with src IP adress 8.8.8.8
This is 7th packet with src IP adress 8.8.8.8
This is 8th packet with src IP adress 8.8.8.8
This is 9th packet with src IP adress 8.8.8.8
```

(3) **[4 points]** We change the filter in sniff() function to **filter="src host 10.0.2.6"**. Then, run **test_spoof.py** on 10.0.2.4. Next, on 10.0.2.6, run **$ ping 8.8.8.8** which receives the following outputs.

```
[12/11/20]seed@VM:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=32.2 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=51.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=25.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=36.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=18.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=29.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=17.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=118 time=32.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=18.8 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=118 time=28.7 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, +5 duplicates, 0% packet loss, time 4046ms
rtt min/avg/max/mdev = 17.258/29.060/51.206/9.641 ms
[12/11/20]seed@VM:~$ ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=24.9 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=20.9 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=22.7 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=18.7 ms
```

It seems every ping packet receives two replies. However, on 10.0.2.6, if we run **$ ping 1.2.3.4**, every ping packet only receives one reply. Can you explain this?

(4) **[4 points]** As in (3), we use **filter="src host 10.0.2.6"** for sniff() function. Then, run **test_spoof.py** on 10.0.2.4. Next, on 10.0.2.6, run **$ ping 8.8.8.8**. In this case, the callback function will be executed for our ping request packet. Please determine **newpkt[IP].src**, **newpkt[IP].dst**, **newpkt[IP].proto**.

(5) **[4 points]** We want to change the filter to sniff the packet from source IP 10.0.2.5 to destination IP either 8.8.8.8 or 1.2.3.4. Please specify the new filter.

(6) **[4 points]** The following is the printout of **pkts[5].show2()** in the setting of question 4. Please determine the size of **pkts[5][ICMP]** (based on IP header).

```
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:53:e3:b2
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 48088
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x6ec5
     src       = 10.0.2.6
     dst       = 1.2.3.4
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xaa06
        id        = 0x1a50
        seq       = 0x1a7
###[ Raw ]###
           load      = '\xcbB\xd4_\xa4\\\x03\x00\x08\t\n\x0b\x0c\r
```