Challenge 1

Tim Naschke

Notes on Time

In the lecture there was the question how long it took us to complete this challenge. Therefore I wanted to include an estimate here. Including setting up the environment, this challenge has taken me approximately 6 hours. But I have not measured the time, so this is only a rough estimate.

Task 1: Using Scapy to Sniff and Spoof Packets

Task 1.1: Sniffing Packets

Task 1.1A

For this Task, mycode.py looks like

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-8e21ad3a9d02", filter="icmp", prn=print_pkt)
```

Running mycode.py with root privileges and then sending a ping request to the router yields the output

```
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 45055
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x768d
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xb3
        id        = 0x1f
        seq       = 0x1
        unused    = ’’
###[ Raw ]###
           load      = ’\\xe3\\x97$b\x00\x00\x00\x00)‘\x07\x00
                        \x00\x00\x00\x00\x10\x11\x12\x13\x14
                        \x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d
                        \x1e\x1f !"#$%&\’()*+,-./01234567’

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 56659
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x8939
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x8b3
        id        = 0x1f
        seq       = 0x1
        unused    = ’’
###[ Raw ]###
           load      = ’\\xe3\\x97$b\x00\x00\x00\x00)‘\x07\x00
                        \x00\x00\x00\x00\x10\x11\x12\x13\x14
                        \x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d
                        \x1e\x1f !"#$%&\'()*+,-./01234567'
```

Then running mycode.py without root privileges results in scapy throwing a PermissionError. Which makes sense, because a program without root privileges should not be able to record all network traffic for security.

Task 1.1B

Capturing all IMCP Packets can be seen above. For capturing any TCP packet that comes from a particular IP and with a destination port number 23 I used the following code:

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-8e21ad3a9d02", filter="src host 10.9.0.5
            and tcp dst port 23", prn=print_pkt)
```

When I then send a try to log in over telnet from 10.9.0.5 to 10.9.0.6 this yields:

```
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 57780
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x44db
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ TCP ]###
```

```
        sport     = 55980
        dport     = telnet
        seq       = 3496711137
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x144b
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''),
                    ('Timestamp', (759961828, 0)),
                    ('NOP', None), ('WScale', 7)]
```

This is only the first package send, otherwise this would be far to long.

Task 1.2: Spoofing ICMP Packets

The code used for this task is

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

def print_pkt(pkt):
 pkt.show()

ip = IP()
ip.dst = '10.9.0.5'
ip.src = '10.9.0.6'
icmp = ip/ICMP()

send(icmp)
```

In wireshark we can observe the response

```
# Packet 1 from /tmp/wireshark_br-8e21ad3a9d0253KYI1.pcapng
- 34
- 4.212904158
- 10.9.0.5
- 10.9.0.6
- ICMP
- 42
- Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 31)
```

Interestingly wireshark notes that the icmp checksum of the response is incorrect. It should be 0xffff, but is 0x0000. I don't know why that is (maybe there is some overflow happening here, since for 4byte-integers ffff + 1 = 0000, right?), but the checksum of the icmp request is not marked as false.

Task 1.3: Traceroute

For this task I used an automated approach:

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

ttl = 1
while True:
    pkg = IP(dst='216.58.209.36', ttl=ttl)/ICMP()
    answer = sr1(pkg)

    if answer.type == 0:
        print("----------------------------------------")
        print(f"Successfull Answer using ttl {ttl}")
        exit(0)

    if answer.type == 11 and answer.code == 0:
        ttl += 1
    else:
        answer.show()
        print("Something else went wrong!")
        exit(1)
```

Which yields the following output:

```
Begin emission:
Finished sending 1 packets.

Received 2 packets, got 1 answers, remaining 0 packets
Begin emission:
Finished sending 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets

# and so on...

Begin emission:
```

```
Finished sending 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
----------------------------------------
Successfull Answer using ttl 12
```

Task 1.4: Sniffing and-then Spoofing

The code used in this:

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

def spoof_echo_reply(pkt):
    if pkt[ICMP].type != 8:
        return

    ip = IP()
    ip.ihl = pkt[IP].ihl
    ip.src = pkt[IP].dst
    ip.dst = pkt[IP].src

    icmp = ICMP()
    icmp.type = 0
    icmp.id = pkt[ICMP].id
    icmp.seq = pkt[ICMP].seq

    data = pkt[Raw].load

    response = ip/icmp/data
    send(response)


sniff(iface="br-8e21ad3a9d02",
      filter="icmp and src host 10.9.0.5", prn=spoof_echo_reply)
```

For pinging a non-existent host on the internet (1.2.3.4) the script works and we get a working ping:

```
root@0490ff3f5ae5:/# 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=80.1 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=26.2 ms
```

```
# and so on...
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=29.8 ms
^C
--- 1.2.3.4 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7009ms
rtt min/avg/max/mdev = 23.206/34.029/80.102/17.577 ms
```

This result is what we wanted and as expected.

For pinging a existent host on the internet (8.8.8.8) the script yields the following:

```
root@0490ff3f5ae5:/# 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=111 time=40.5 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=71.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=27.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=111 time=45.1 ms (DUP!)
# and so on...
64 bytes from 8.8.8.8: icmp_seq=10 ttl=64 time=22.2 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=111 time=36.5 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
10 packets transmitted, 10 received, +10 duplicates, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 19.208/34.854/71.270/12.658 ms
```

So here we get duplicate ping responses, which makes sense since the script is sending an answer regardless of whether the actual pinged host also sends an answer.

Now for the last case, sending a ping to a non-existing host on the LAN, we get:

```
root@0490ff3f5ae5:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4103ms
```

Here, because the destination is on the same network, the pinging host tries to find out the MAC address of the destination to send the packet. However it gets no answer to its request, since the host doesn't actually exist and we are not spoofing these types of messages. Therefore the source doesn't get a MAC address and doesn't even send the imcp packages, since it needs a physical address to send them to.