AARHUS UNIVERSITET
INGENIØRHØJSKOLEN

Network Security E20, Sniff-and-spoof

Peter Marcus Hoveling, Student No.:201508876

December 21, 2020

Github: https://github.com/201508876PMH/Sniff-and-spoof

Practical assignment 3 part I, Sniff-and-spoof
Date for handin: December 19 2020
Lecturers: Diego F. Aranha

1 Introduction

This assignment has three parts: (I) the first deals with sniffing and spoofing network packets; (II) the second deals with implementing a TCP throttling tool; and (III) the third with implementing a small VPN tunneling program [1]. **From these parts, part one has been chosen, Sniff-and-spoof**.

The objective of this task is to spoof IP packets with an arbitrary source IP address. We will specifically spoof ICMP echo reply packets for all hosts in the same local network, as a way to confuse a system administrator attempting to diagnose the network conditions. For this we will write a sniffer program that listens for ICMP echo request packets and spoof a response back to the originator host. Furthermore we will collect evidence of the malicious behavior through Wireshark and screenshots of replies received by the `ping` program.

2 ICMP packets

Before doing anything, we first need to understand what an ICMP packet is. The ICMP packet comes in two formats IPv4 and IPv6. The IPv4 packet is the one we will be focusing on, and has the following datagram (see figure 1).

IPv4 Datagram

	Bits 0–7	Bits 8–15	Bits 16–23	Bits 24–31
Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		*flags* and *offset*	
	Time To Live (TTL)	Protocol	Header Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Header Data			
ICMP Payload (*optional*)	Payload Data			

Figure 1: Picture of ICMP packet, from Wikipedia[5]

Looking at the IPv4 datagram we first see the IPv4 header, where information about where the packet is to be sent, who sent it, the calculated checksum and more. Of these different header fields, we will be interested in the Source IP address and Destination address. Right below we have the ICMP header, it is here we have the possibility to change what type of message we wish to send. The 'Type of message' (identified by a "Type" field) has many possible messages. Alot of these ICMP types are obsolete and are no longer seen in the internet[2], however some are widely used and include Echo Reply (0), Echo Request (8), Redirect (5), Destination Unreachable (3) and Time Exceeded (11). Looking at figure 2 a list from some of the different types of ICMP types can be seen.

```
Type    Name                                     Reference
----    -------------------------                ---------
  0     Echo Reply                                [RFC792]
  1     Unassigned                                   [JBP]
  2     Unassigned                                   [JBP]
  3     Destination Unreachable                   [RFC792]
  4     Source Quench                             [RFC792]
  5     Redirect                                  [RFC792]
  6     Alternate Host Address                       [JBP]
  7     Unassigned                                   [JBP]
  8     Echo                                      [RFC792]
  9     Router Advertisement                     [RFC1256]
 10     Router Selection                         [RFC1256]
 11     Time Exceeded                             [RFC792]
 12     Parameter Problem                         [RFC792]
 13     Timestamp                                 [RFC792]
 14     Timestamp Reply                           [RFC792]
 15     Information Request                       [RFC792]
```

Figure 2: Picture of different types, from abdn.ac.uk[2]

3 Implementation

Having a better understanding of what an ICMP packet is and what to manipulate, we begin to look at an implementation.
For writing low-level networking code, we have the options of the C programming library libnet/libcap or the equivalent Scapy package in python. Both of these libraries enable us to send, sniff, dissect and forge network packets. Allowing us the capabilities to construct a tool that can probe, scan or attack networks.

Python and the library Scapy[4] has been chosen, as a matter of preference, and if we look at figure 3 we can see the main function `spoof_reply(pkt)`.

spoof_reply.py

```python
def spoof_reply(pkt):
    if(pkt[2].type == 8):
        print("Creating spoof packet...")

        dst = pkt[1].dst
        src = pkt[1].src
        ttl = pkt[1].ttl
        id_IP = pkt[1].id

        seq = pkt[2].seq
        id_ICMP = pkt[2].id

        reply = Ether(src=pkt[0].dst, dst=pkt[0].src, type=pkt[0].type)
                /IP(id=id_IP, ttl=ttl,src=dst, dst=src)
                /ICMP(type=0, code=0, id=id_ICMP, seq=seq)

        # contruct the packet with a new checksum for the IP header
        del reply[IP].chksum

        # contruct the packet with a new checksum for the ICMP packet
        del reply[ICMP].chksum

        raw_bytes = reply.build()
        reply[IP].chksum = Ether(raw_bytes)[IP].chksum
        reply[ICMP].chksum = Ether(raw_bytes)[ICMP].chksum

        reply.show2()
        sendp(reply, iface="ens18")
```

Figure 3: Picture of spoof function, from Github [3]

In this function we first check to see if the packet, which is being sniffed, has its `type` set to 8. If this is the case, we know we have sniffed an ICMP Echo packet. The reason we look at index 2 of the packet is because of the following packet structure.

Figure 4: A sniffed ICMP echo packet with shown structure

```
###[ Ethernet ]###
  dst       = ca:2e:39:a0:62:de
  src       = 88:e9:fe:56:f4:b0
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 4631
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x3828
     src       = 192.168.87.113
     dst       = 192.168.87.168
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x87af
        id        = 0xbe15
        seq       = 0x0
###[ Raw ]###
           load      = '_\xd3?%\x00\x05(:\x08\t\n\x0b\x0c\r\x0e\x0f
                          \x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a
                          \x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

Looking at figure 4 we see a sniffed ICMP echo packet. We know this is an echo packet, since the ICMP header `type`-field is listed as an echo-request. Looking back at our code, it is here clear to see why we check at index 2. Index 0 is the Ethernet header, index 1 is the IP header and index 2 is the ICMP header.

When we are assured that the sniffed packet is an ICMP echo packet, we begin to contruct our reply. At line 13 of our spoof function, we first create our Ethernet header with our source address being the sniffed packets destination and the new destination being the sniffed packets source.

We then create the next header, the IP. For this we model our new packet to mimic our sniffed packet, only changing the src and dst fields.
We then look at our ICMP header. We change the type from being a echo request, to being an echo reply. This is done by changing the type to being a value 0 and code 0. By changing this and then lastly recalibrating our checksum, we send back the packet as being an echo-reply over the internet interface `ens18`.

We now just need to call our function and await our victim, see figure 5.

sniff-and-spoof-main.py

```python
if __name__=="__main__":

    # define the network interface
    iface = "ens18"

    # filter for only ICMP trafic
    filter = "icmp"

    # start sniffing
    sniff(iface=iface, prn=spoof_reply, filter=filter)
```

Figure 5: main function for calling the `spoof_reply()` function

4 Spoofing

Testing our program we try to disable the possibilities for pinging the target machine. The target machine is in this case a vm with headless Ubuntu running, where we use the following command to disable pings: `echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_all`. Trying to send a ping to machine without our program running, we get an echo-reply stating a 100% packet loss, as seen in figure 6.

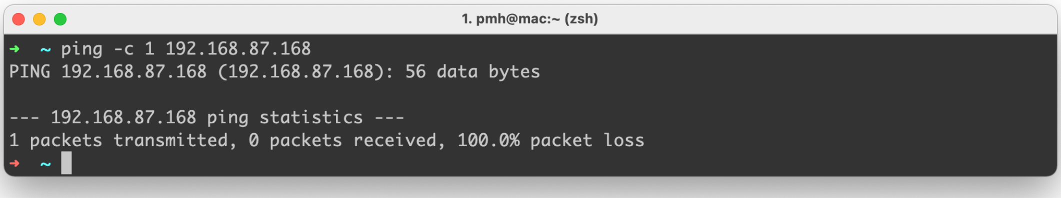

Figure 6: Sending a ping to machine, where pings are disabled

When we then start our program, as seen in figure 7, we see the output of the packet we send back. The previous destination is now our source and the previous source is now our destination. Furthermore our type is changed to an echo-reply, code changed to 0 and the checksums have been recalculated.
In this case, we choose to send back a echo-reply. However we could also spoof the system administrator by sending back a type 3 "Destination Unreachable" or a type 11 "Time exceeded".

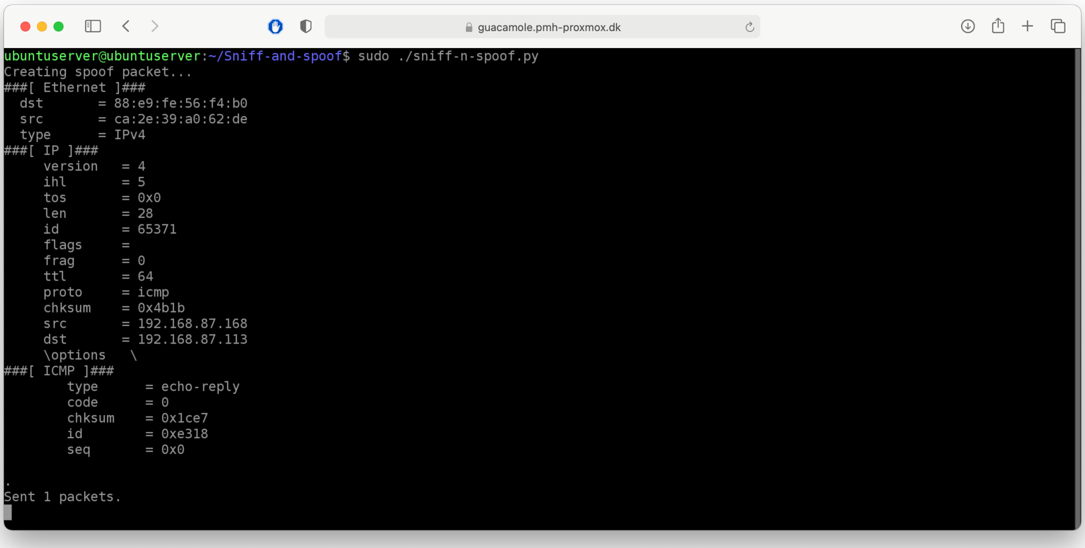

Figure 7: The output from our Sniff and spoof program

Trying to send the same ping request, but with our program running, we see in figure 8, that we successfully get back the 1 packet we sent.

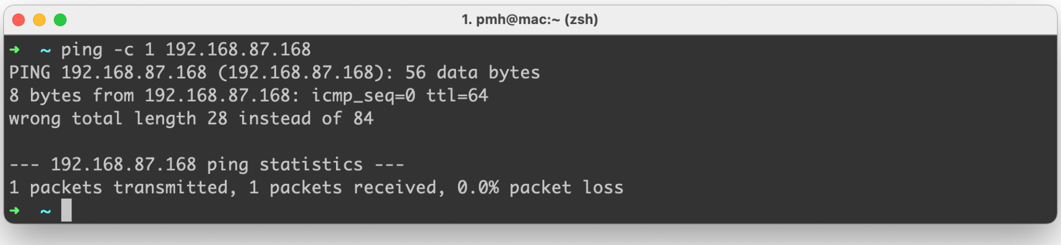

Figure 8: Sending a ping to machine, where our program is running

Having a look at Wireshark, as seen in figure 9, we've set a filter of being `icmp.type==8 or icmp.type==0 or`

icmp.type==3 and ip.addr==192.168.87.168. The filter simply filters anything that isnt of an ICMP packet type 8, 0 or 3. Furthermore it must be sent to or from the target machine, with IP 192.168.87.168. We can see that Wireshark first pics up the ping request from the administrators machine IP 192.168.87.113 and then a echo-reply packet from the target machine.
Looking at the packet details, we can see the type being IPv4, the source and destination addressed being correct, the ICMP header having the correct type, code and matching checksum as what our program output displayed.

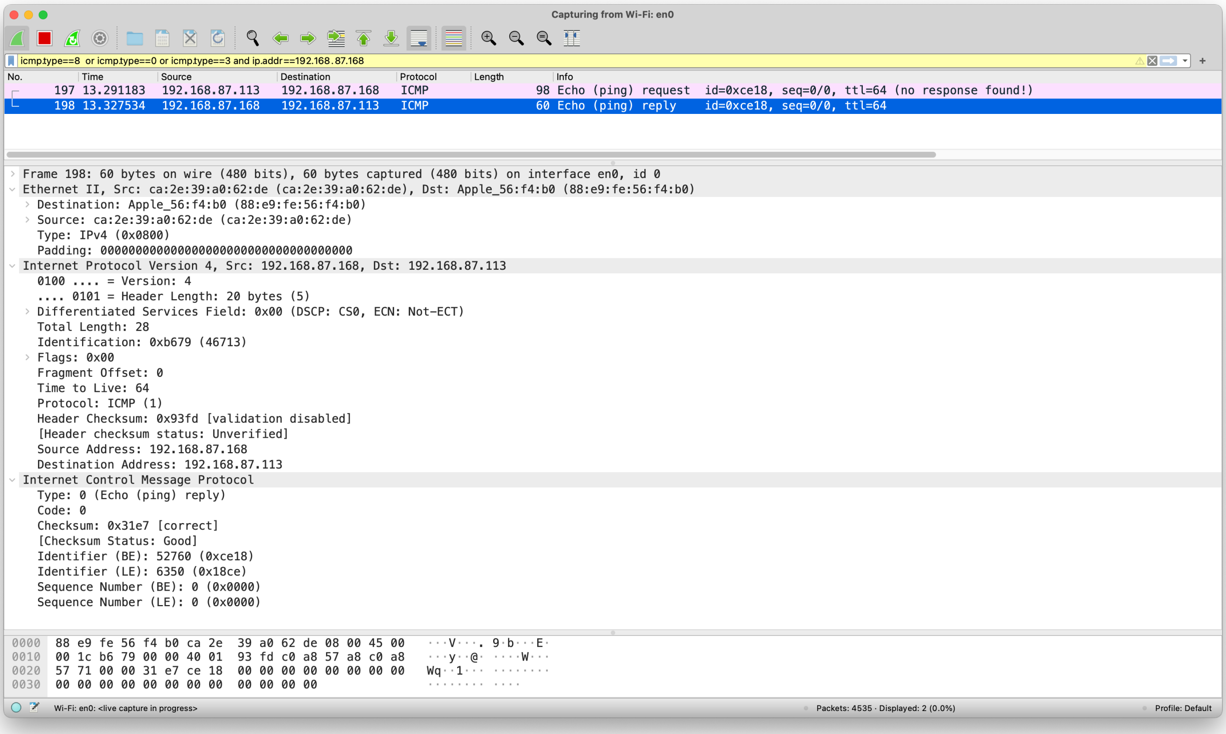

Figure 9: The captured reply from WireShark

Looking at the bigger picture, in the context of network security, a spoofing attack is in general a situation in which a person or program can successfully falsify data by identifying as another. In this assignment, we managed to spoof a ping, by sending back data which in theory wasn't from the attacked machine. In theory, we could have our program running and upon ping request, could send back a multitude of replies to confuse the system administrator and maybe even cause system maintenance.

5 Bibliography

Websites

[1] Diego F. Aranha. *Practical assignment 3*. URL: https://blackboard.au.dk/webapps/blackboard/content/listContent.jsp?course_id=_136793_1&content_id=_2769375_1&mode=reset.

[2] Gorry Fairhurst. *ICMP Type Numbers*. URL: https://erg.abdn.ac.uk/users/gorry/course/inet-pages/icmp-code.html.

[3] Peter Marcus Hoveling. *Github repo*. URL: https://github.com/201508876PMH/Sniff-and-spoof.

[4] Scapy. *Scapy*. URL: https://scapy.net.

[5] Wikipedia. *ping (networking utility)*. URL: https://en.wikipedia.org/wiki/Ping_(networking_utility).